Global System Image Architecture for Cluster Computing*

Hai Jin, Li Guo, and Zongfen Han

Internet and Cluster Computing Center
Huazhong University of Science and Technology,
Wuhan, 430074, China
hjin@hust.edu.cn

Abstract. This paper presents a novel single system image architecture for cluster system, called Glosim. It is implemented in the kernel layer of operating system, and modifies system invokes relative to IPC objects and process signals. This system provides global system image support. It not only support global IPC objects including message queue, semaphore and shared memory, but also a new concept of global working process and it SSI support smoothly and transparently. Combined with Linux Virtual Server, single IO space, it completely constructs a high performance cluster server with SSI.

1 Introduction

Single system image (SSI) [1][2][3][4] is the property of a system that hides the heterogeneous and distributed nature of the available resources and presents them to users and applications as a single unified computing resource. SSI can be enabled in numerous ways, ranging from those provided by extended hardware to various software mechanisms. SSI means that users have a global view of the resources available to them irrespective of the node to which they are physically associated.

The design goals of SSI for cluster systems are mainly focused on complete transparency of resource management, scalable performance, and system availability in supporting user applications. In this paper, we present a novel single system image architecture for cluster system, called *Global System Image* (Glosim). Our purpose is to provide system level solution needed for a single system image on high performance cluster with high scalability, high efficiency and high usability. Glosim aims to solve these issues by providing a mechanism to support global *Inter-Process Communication* (IPC) objects including message queue, semaphore and shared memory and by making all the working process visible in the global process table on each node of a cluster. It completely meets the need of people of visiting a cluster as a single server, smoothly migrating Unix programs from traditional single machine environment to cluster environment almost without modification.

The paper is organized as follows: Section 2 provides the background and related works on single system image in cluster. Section 3 describes a generic and scalable global system image project overview. Section 4 describes Glosim software

* This paper is supported by National High-Tech 863 Project under grant No. 2002AA1Z2102.

architecture and introduces the global IPC and global working process. In Section 5, we discuss the implementation of Glosim including software infrastructure and error detection. Section 6 presents analysis of performance effects on the developed prototype of the Glosim. Section 7 describes the Glosim enabled applications. Finally, a conclusion is made in Section 8 with more future work.

2 Related Works

Research of OS kernel-supporting SSI in cluster system has been pursued in a number of other projects, including SCO UnixWare [5], Mosix [6][7], Beowulf Project [8][9], Sun Solaris-MC [10][11] and GLUnix [12].

UnixWare NonStop cluster is a high availability software. It is an extension to the UnixWare operating system in which all applications run better and more reliably inside a SSI environment. The UnixWare kernel has been modified via a series of modular extensions and hooks to provide single cluster-wide file system view, transparent cluster-wide device access, transparent swap-space sharing, transparent cluster-wide IPC, high performance internode communications, transparent cluster-wide process migration, node down cleanup and resource fail-over, transparent cluster-wide parallel TCP/IP networking, application availability, cluster-wide membership and cluster time sync, cluster system administration, and load leveling.

Mosix provides the transparent migration of processes between nodes in the cluster to achieve a balanced load across the cluster. The system is implemented as a set of adaptive resources sharing algorithms, which can be loaded into the Linux kernel using kernel modules. The algorithms attempt to improve the overall performance of the cluster by dynamically distributing and redistributing the workload and resources among the nodes of a cluster of any size.

Beowulf cluster refers to a general class of clusters built for speed, not reliability, built from commodity off the shelf hardware. Each node runs a free operating system like Linux or Free-BSD. The cluster may run a modified kernel allowing channel bonding, global PID space or DIPC [13][14][15][16]. A global PID space lets users see all the processes running on the cluster with *ps* command. DIPC make it possible to use shared memory, semaphores and wait-queues across the cluster.

Solaris MC is a distributed operating system prototype, extending the Solaris UNIX operating system using object-oriented techniques. To achieve these design goals each kernel subsystem was modified to be aware of the same subsystems on other nodes and to provide its services in a locally transparent manner. For example, a global file system called the proxy file system, or PXFS, a distributed pseudo */proc* file-system and distributed process management were all implemented.

GLUnix is an OS layer designed to provide support for transparent remote execution, interactive parallel and sequential jobs, load balancing, and backward compatibility for existing application binaries. GLUnix is a multi-user system implementation built as a protected user-level library using the native system services as a building block. GLUnix aims to provide cluster-wide name space and uses network PIDs (NPIDs) and virtual node numbers (VNNs). NPIDs are globally unique process identifiers for both sequential and parallel programs throughout the system. VNNs are used to facilitate communications among processes of a parallel program. A suite of user tools for interacting and manipulating NPIDs and VNNs is supported.

3 System Overview

Glosim is a single system image implementation prototype based on Linux cluster. It is mainly used to construct high performance cluster server, such as distributed web server, distributed BBS server, distributed MUD server. The design goal of Glosim is to conveniently migrated UNIX programs from former single machine environment to distributed cluster environment without too much modification. For example, former FireBird BBS can be easily migrated to Glosim distributed cluster environment just by putting global variables into shared memory structure.

Figure 1 is the complete infrastructure of Glosim. Combined with Linux Virtual Server [17], SIOS [18][19], it completely constructs a high performance cluster network server with SSI. Cluster network server based on Glosim consists of three major parts with three logical layers to provide high load network services.

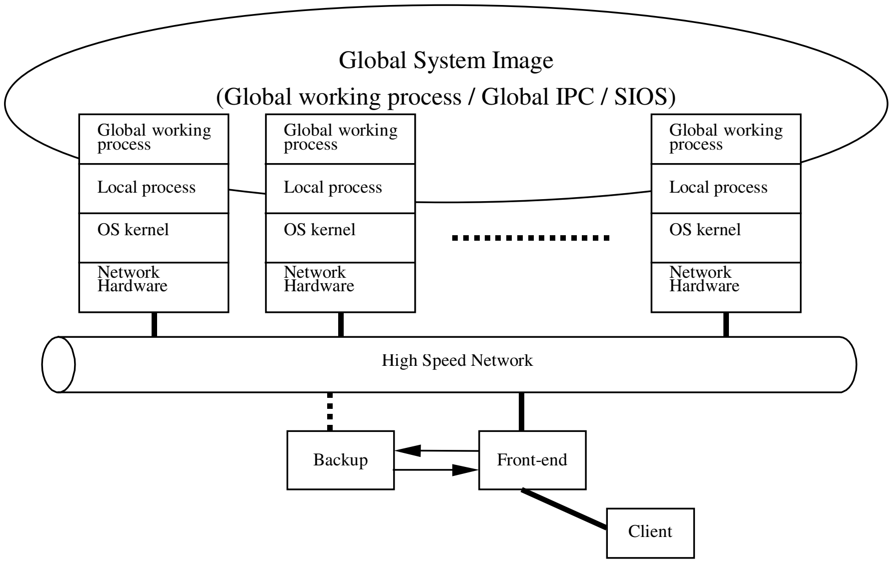

Fig. 1. Glosim Software Architecture

The first layer includes front-end and backup running Linux Virtual Server to schedule and backup each other to improve system reliability. They are the message entrance to the whole server in charge of receiving client requests and distributing these requests to the processing nodes of the cluster according to certain strategy (such as WRR [17] algorithm). Because Front-end and Backup are only in charge of redirections requested by clients and do not involves data processing, they are quite slightly loaded. Besides, in front-end and backup, hot standby daemon is running for fault tolerance. In case of failure in front-end, backup can take charge of its functions by robbing IP and recover Front-end kernel data to achieve high reliability of the whole system network connection.

The second layer are local OS kernel and local processes of the processing nodes inside cluster, which are in charge of system booting, providing basic system call and application software support. Logically, they are outside the boundary of SSI.

The third layer is Glosim layer, as middleware of cluster system, which includes global IPC, global working process, SIOS. Logically, this layer is inside the boundary of SSI. Glosim is implemented in OS kernel and transparently provides SSI service to user applications by modifying system invokes relative to IPC objects and process signals and by mutual cooperation and scheduling between nodes inside the cluster.

4 Glosim Software Architecture

4.1 Global IPC

Linux supports three types of inter-process communication mechanism, which are respectively message queue, semaphore and shared memory. These three communication mechanisms first appear in Unix System V using the same authorization method. Processes can only access these IPC objects by transferring a unique reference designator to the kernels through system invokes. This unique reference designator is the key point to implement the single system image of IPC objects within the cluster.

Global inter-process communication IPC mechanism is the globalization of IPC objects, mainly based on DIPC. Through DIPC, it provides the whole cluster with global IPC, including consistency of shared memory, message queue and semaphore.

What should be specially pointed out here is that distributed shared memory [1][2] in global IPC uses a multiple-read/single-write protocol. Global IPC replicates the contents of the shared memory in each node with reader processes, so they can work in parallel, but there can be only one node with processes that write to a shared memory. The strict consistency model is used here, meaning that a read will return the most recently written value. It also means that there is no need for the programmer to do any special synchronization activity when accessing a distributed shared memory segment. The most important disadvantage with this scheme is possible loss of performance in comparison to other DSM consistency models.

Global IPC can be configured to provide a segment-based or a page-based DSM. In the first case, DIPC transfers the whole contents of the shared memory from node to node, with no regard to whether all that data are to be used or not. This could reduce the data transfer administration time. In the page-based mode, 4KB pages are transferred as needed. This makes multiple parallel writes to different pages possible. In global IPC, each node is allowed to access the shared memory for at least a configurable time quantum. This lessens the chances of the shared memory being transferred frequently over the network, which could result in very bad performance.

The establishment of global inter-process communication inside cluster system provides the basis for data transfer between processes in each node of the cluster. The programmer can use standard inter-process communication functions and interface of System V to conveniently write application programs suitable to cluster system. What’s more, former programs can be migrated to Glosim without too much payout. Actually, Glosim enabled applications only require the programmer to do some special transactions to global variables. For example, global variables can be stored as structures in shared memory, and accessed normally.

4.2 Global Working Process

Global Working Process includes Global Working Process Space, Global Signal Mechanism and Global Proc File System.

Considering a practically running cluster system, it is necessarily a server providing single or multiple services and we just call these serving processes working process. What we should do is to include working processes to the boundary of SSI and provide them SSI service from system level. In this way, working processes in different nodes can communicate transparently and become global working processes.

However, not all processes in each node are necessarily to be actualized with SSI in the whole cluster system. That is to say, only some of the processes reside within the boundary of SSI. By introducing the concept of working process, we may solve the inconsistency of OS system processes between each node. OS system process of each node is relatively independent, outside SSI boundary while global working process is provided by the system with single system image service, inside SSI boundary. The independence of working process is convenient for the cluster system to extend to OS heterogeneous environment.

As for practical programming implementation, it is quite easy to express working process. For example, we can tag working process with a special uid for the convenience of kernels identification and independent transaction. This enactment is closely related to the practical system, such as httpd process in Apache is often running as an apache user and bbsd process in BBS is executed as a bbs user.

(a) Global Working Process Space

Single process space has the following basic features:

- Each process holds one unique pid inside the cluster.
- A process in any node can communicate with other processes in any remote node by signals or IPC objects.
- The cluster supports global process management and allows the management of all working processes just like in local host.

In order to uniquely tag the working process in the whole cluster, we must introduce the concept of global working process space. There are mainly two solutions as follows:

- Method 1: modulus the process number to tag the node number in which the process runs. For example, in a cluster system with 16 nodes, the way of calculating the node number of process 10001 is 10001%16=1. So process 10001 is running in node 1.
- Method 2: distribute different working process pid space to each node. For instance, the working process space of node 1 is between 10000 and 20000, the working process space of node 2 is between 20000 and 30000, and so on.

No matter which algorithm we adopt, the purpose is to make working process pid unique within the whole cluster. As for each node, local process pid is decided by last_pid of OS kernel itself. Each node is independent on each other but every local pid is smaller than the lower limit of global process space fixed in advance. In short, local process pid is independent on each other while global working process pid is unique in the whole cluster.

(b) Global Signal Mechanism

Signal is the oldest inter-process communication mechanism in Unix system, used to transfer asynchronous signals. In global signal mechanism, all working processes can send each other signals using the functions like signal (), kill () and so on just like in the same computer. That is to say, processes can transparently conduct mutual communication within the cluster by using functions like signal (), kill () and so on.

The transaction flow is as follows:

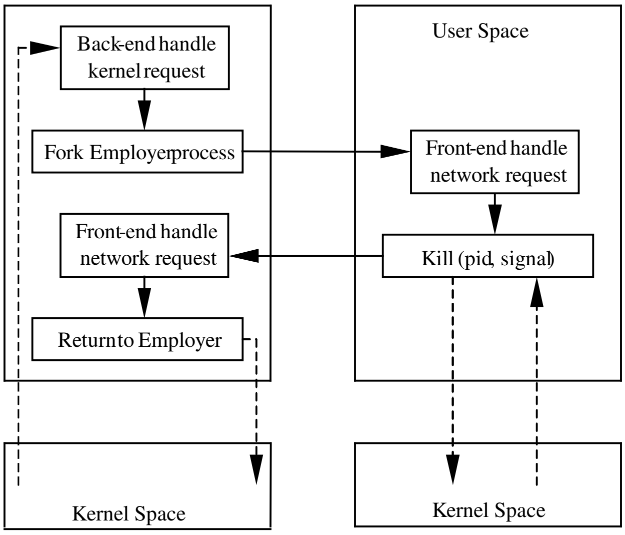
Back-end handle kernel request
Fork Employer process
Front-end handle network request
Return to Employer
Kernel Space
User Space
Front-end handle network request
Kill (pid, signal)
Kernel Space

Fig. 2. Processing Flow of Glosim

(c) Global Proc File System

Global proc file system is to map status messages of all processes in every node to the proc file system of each node within the cluster for the convenience of united control and management. Global proc file system includes local processes with the process number smaller than max_local_pid and global processes with the process number bigger than max_local_pid. This is because Glosim system aims to actualize the single system image of working processes and one basic characteristic of working processes is the process number bigger than max_local_pid. So we can only consider working processes.

With the actualizing of global proc file system, we can use *ps -ax* command to examine local processes running in current node and all global working processes in the whole Glosim system. We can employ *kill* command to send signals to any process and even use *kill -9* command to end any process we want with no regard to which node it actually runs. In this way, complete working process single image is implemented in cluster system.

5 Implementation of Glosim

Glosim has two components: The major one is a program called glosimd, which runs in the user space with root privileges. The other component is hidden inside the linux kernel, which allows glosimd to access and manipulate kernel data. This design is the result of the desire to keep the needed changes in the operating system kernel to a minimum.

Glosimd creates some processes to manage all the necessary decision makings and network operations. They use predetermined data packets to communicate with glosimd processes on other nodes. All the necessary information for a requested action are included in these packets, including the system call's parameters obtained from inside the kernel. The whole system infrastructure is shown in Figure 3.

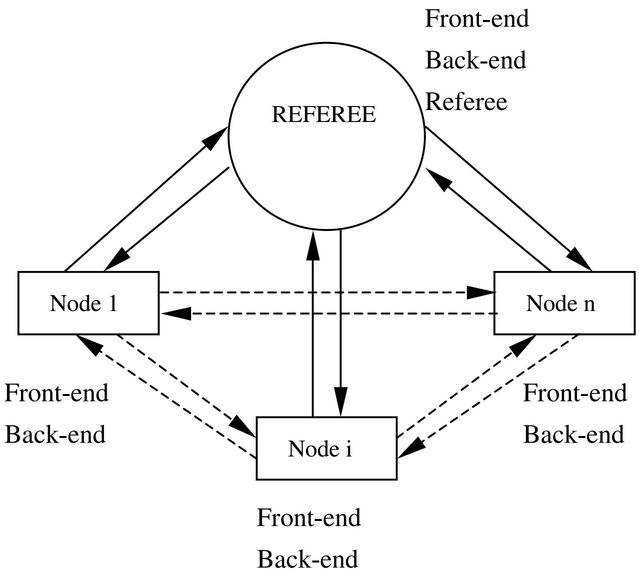

Fig. 3. System Implementation Infrastructure

Glosim includes three function modules: Front-end, Back-end and Referee. Among those, Front-end runs in every node, in charge of transacting data requests from network and forking worker process. Back-end also runs in every node, in charge of transacting data request from kernel and forking employer process. Referee, as the central arbitrating node, only runs in the central node, maintaining a chain table of all IPC structures and pid. Employer, forked by Back-end, executes remote system invokes. Worker, forked by Front-end, transacts remote requests from other nodes.

Glosim adopts a fail-stop distributed environment. So it uses time-outs to find out any problem. Here the at-most-once semantics is used, meaning that Glosim tries everything only once. In case of an error, it just informs it of the relevant processes; either by a system call return value, or, for shared memory read/writes, via a signal. Glosim itself does not do anything to overcome the problem. It is the user processes that should decide how to handle it.

6 Performance Analysis

This section presents some basic analytical results on the performance of the Glosim Cluster. The benchmark program sets up a shared memory, a message queue, and a semaphore set. The tested system calls are: semctl(), msgctl(), shmctl() with the IPC_SET command, semop() and kill() operations. Besides, msgsnd() and msgrcv() with 128, 256, 512, 1024, 2048 and 4096 bytes messages are also tested.

Figure 4 is an experiment conclusion to measure the speed of executing some of the Glosim system calls in a system. It shows the executing times of semctl(), msgctl(), shmctl()with the IPC_SET command, semop(), and kill() in practical system benchmark with the unit ms.

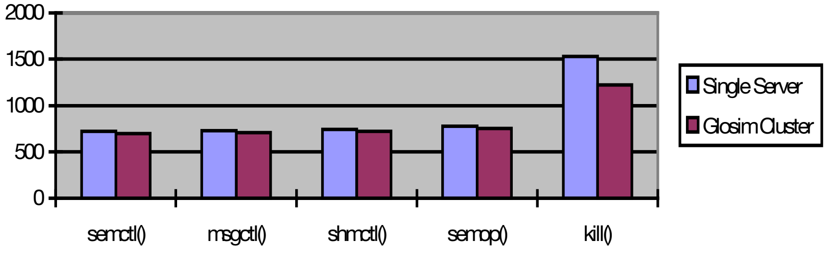

Fig. 4. Speed of executing some Glosim system calls in a system

Figure 5 shows some experiment conclusions to measure the speed of send and receive message in the system. Figure 5(a) shows the system message transfer bandwidth with the message size of 128, 256 up to 4096 bytes in single server. Figure 5(b) shows the system message transfer bandwidth with the message size of 128, 256 up to 4096 bytes in Glosim server with 8 nodes.

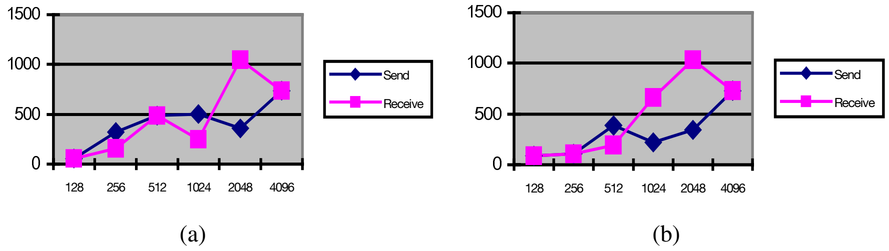

Fig. 5. Speed of send and receive messages for (a) single server (b) Glosim server with 8 nodes

As the data shown in figure 4, Glosim has a normal and acceptable response speed to atom operation of IPC objects and global signal processing mechanism. According to the data shown in the figure 5, in case of low data load (4KB below), msg send/receive bandwidth in Glosim server is a little smaller than that in single server. The results appear to be quite normal and acceptable because Glosim server involves network transfer and the overload of data copy of kernel and user space. Thus, Glosim also has excellent performance support for practical system.

7 Glosim Enabled Applications

In fact, programs based on Glosim can exchange IPC objects just by numerical keys and they can tag each global working process using global process pid. This indicates that there is no need for them to make sure where the corresponding IPC structure lies physically and in which node the practical global working process runs. Glosim system ensures that working processes can find out the needed resources and other working processes only by fixed key values and global pid. These resources and working processes lie in different nodes when running. However, through the logical addressing and global pid addressing of resources, programs can be independent on any physical network characteristics and thus transparently actualize single system image for upper application programs.

Glosim just aims to smoothly migrate Unix programs from traditional single machine environment to cluster environment almost without modification. So, as a traditional Unix program, BBS is quite a typical example which involves different aspects of Unix including network socket, IPC object, signal operation, file I/O operation and so on. What's more, it is very popular and much requires serving efficiency after login. If supported by Glosim, BBS can be smoothly migrated to cluster environment with considerable performance improvement compared with that in single environment, it fully proves that Glosim is of high usability, performance and stability.

Scalable BBS cluster server is a practical application based on Glosim. In order to efficiently extend the traditional BBS with single server to the BBS with cluster server and provide services of high performance, file consistency, balanced load and transparency, we make the design requirement as follows:

(1) BBS original program based on IPC mechanism in former single environment must be extended to the distributed one.

(2) Materials in shared memory and file system of each node in cluster must keep consistent.

(3) Servers can be increased and deleted dynamically or statically.

(4) Numbers of the registration and the login must increase in proportion to the numbers of servers.

(5) Transparently support upper applications from system layer almost without modifying user program codes.

Figure 6 shows the max login numbers of BBS in Glosim cluster with different memory settings and different nodes. X-axis indicates the number of nodes, while Y-axis indicates the numbers of people.

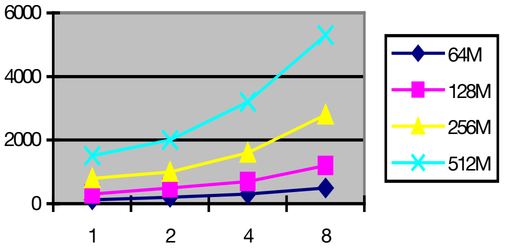

Fig. 6. Max login numbers of BBS in Glosim cluster with different node numbers

8 Conclusions and Future Works

In this paper, we present a novel architecture of a single system image built on cluster system, named Glosim. This system not only supports global IPC objects including message queue, semaphore and shared memory, but also presents a new concept of global working process and provides it SSI support smoothly and transparently. Combined with Linux Virtual Server and SIOS, it completely constructs a high performance SSI cluster network server.

In this case, a great number of applications in single Linux environment such as BBS, Apache and MUD server can be migrated into cluster environment almost unmodified. In this way, single system image service is provided to upper application programs transparently.

Based on Glosim, processes can exchange IPC data through numerical key value. Global process pid is used to identify each global working process, which means it is unnecessary to know where the responding IPC structure physically exists and in which node a global working process physically runs. Logical addressing of resources and global pid addressing make processes independent of physical network and transparently provide single system image to upper application programs.

With all its advantages mentioned, however, Glosim is utilized currently with the disadvantages of relatively high system overhead, fault tolerance to be improved and only indirect support to application program global variables. These are the issues we need to solve in our future plan.

References

[1] K. Hwang and Z. Xu, *Scalable Parallel Computing: Technology, Architecture, and Programming*, McGraw-Hill, New York, 1998.

[2] R. Buyya (ed), *High Performance Cluster Computing: Architectures and Systems*, Vol. 1., Prentice Hall, 1999.

[3] R. Buyya, "Single System Image: Need, Approaches, and Supporting HPC Systems", *Proceedings of Fourth International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'97)*, 1997.

[4] R. Buyya, T. Cortes, and H. Jin, "Single System Image", *The International Journal of High Performance Computing Applications*, Vol.15, No.2, Summer 2001, pp.124–135.

[5] B. Walker and D. Steel, "Implementing a full single system image UnixWare cluster: Middleware vs. underware", *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, (PDPTA'99).*

[6] A. Barak and O. La'adan, "The MOSIX multicomputer operating system for high performance cluster computing", *Future Generation Computer Systems*, 1998.

[7] MOSIX – Scalable Cluster Computing for Linux, http://www.cs.huji.ac.il/mosix/

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

[9] BPROC: Beowulf Distributed Process Space, http://www.beowulf.org/software/bproc.html

[10] Y. A Khalidi, J. M Bernabeu, V. Matena, K. Shirriff, and M. Thadani. "Solaris MC: A Multi-Computer OS", *Proceedings of 1996 USENIX Conference*, January 1996.

[11] Sun Solaris-MC, http://www.cs.umd.edu/~keleher/dsm.html

[12] D. Ghormley, D. Petrou, S. Rodrigues, A. Vahdat, and T. Anderson, "GLUnix: A global layer Unix for a network of workstations", *Software Practice and Experience*, 1998.

[13] K. Karimi and M. Sharifi, “DIPC: A System Software Solution for Distributed Programming”, *Proceedings of Parallel and Distributed Processing Techniques and Applications Conference (PDPTA’97)*, Las Vegas, USA, 1997.
[14] K. Karimi and M. Sharifi, “DIPC: The Linux Way of Distributed Programming”, *Proceedings of the 4th International Linux Conference*, Germany, 1997.
[15] K. Karimi, M. Schmitz, and M. Sharifi, “DIPC: A Heterogeneous Distributed Programming System”, *Proceedings of the Third International Annual Computer Society of Iran Computer Conference (CSICC’97)*, Tehran, Iran, 1997.
[16] DIPC, http://www.gpg.com/DIPC/
[17] W. Zhang, “Linux virtual servers for scalable network services”, *Proceedings of Ottawa Linux Symposium 2000*, Canada.
[18] K. Hwang, H. Jin, E. Chow, C. L. Wang, and Z. Xu, “Designing SSI clusters with hierarchical checkpointing and single I/O space”, *IEEE Concurrency*, 7 (1): 60–69, 1999.
[19] K. Hwang and H. Jin, “Single I/O space for scalable cluster computing”, *Proceedings of 1st International Workshop on Cluster Computing*, 1999, pp.158–166.