KSIX Parallel Environment for Beowulf Cluster

Thara Angskun, Choopan Rattanapoca, Putchong Uthayopas
Parallel Research Group, Computer And Network System Research Laboratory
Department of Computer Engineering, Faculty of Engineering
Kasetsart University, Bangkok, Thailand 10900.
Phone: (662)942-8555 Ext. 1416 Email : {g4265087,b39cor,pu}@ku.ac.th

Abstract *Applications and tools development for Beowulf cluster is still a challenging task due to an ineffective cluster environment. Additional services such as global process control, naming services, and event service can help simplify the programming task substantially. In this paper, we present our work on extending OS services by implementing a new parallel environment for Beowulf cluster called KSIX. KSIX provides a cluster virtual machine with a cluster APIs for many services such as global process control, naming services, ensemble management, distributed event services. The APIs that we proposed allow programmers to easily building applications that capitalize the power of Beowulf cluster for their work.*

Keywords: Parallel environment, Beowulf Cluster

1 Introduction

Beowulf cluster [1] is a class of clusters that emphasis on the use of commodity technologies to provide a high performance computer system at low cost. However, the development of applications and tools under Beowulf cluster environment can be a challenging task because this platform provides only a traditional single processor unix programming environment. Although powerful and flexible, this approach is inadequate in many situation. For example, there is no API to start a group of multiple processes and maintain some control over it. The proper extension of OS services and APIs can help speeding up the development dramatically. To alleviate this problem, a new parallel environment for Beowulf cluster called KSIX has been developed. KSIX provides a cluster virtual machine with a cluster APIs for global process control, naming services, ensemble management, distributed event services. These APIs allow programmers to effectively employ a Beowulf cluster for their work. Currently KSIX is used to support the development of next generation of our SCMS [2] cluster management tool.

The organization of this paper is as follows. In section 2, we describe related work followed by KSIX Environment in section 3. Section 4 presents the detail of KSIX Implementation. Finally we present the conclusion and future work in section 7.

2 Related Work

Although there are many literatures concerning the extension of OS services in cluster environment, there are mainly two approaches to extend the programming environment. The first approach is to extend the operating system functionality at kernel level. One example of such work is Mosix [3], which provides kernel-level, adaptive resource sharing algorithms that are geared for low overhead-free scalability process migration. Mosix provides some APIs for cluster developer to control Mosix operation and query system status. Nevertheless, Mosix does not intend to offer any direct program development support. Nomad [4] is an effort similar to Mosix to build a highly scalable cluster OS that provides a

single-system image, resources management, and distributed file system. Nomad is capable of automatics load balancing and process migration when the Nomad additional kernel extension are installed. Nomad supports coscheduling which is not currently included by KSIX but there are no extensive API support as KSIX does. Both MOSIX and Nomad intend to provides automatics handling of resources while KSIX purpose is to provides rich set of APIs that allow user to easily manipulate various things in cluster environment.

Bproc [5] attempts to provides a transparent global process control through the use of proxy process on central node. Due to the close relationship to uniprocessor unix model, Bproc does not accommodate many necessary features such as the support for parallel tasks, naming, events. Although highly efficient, kernel extension is not likely to be portable, hard to develop and hard to keep pace with rapid kernel changes.

The second approach to extend OS services can be done using a virtual machine layer in user space with APIs to access that its services. Although, the performance and transparency to user is less than kernel approach, the advantage of this approach is the high portability. This is even more important when dealing with rapid changing operating system system such as Linux since it reduce the needs to frequently release new patches. Many parallel programming system such as PVM [6] and MPI [7] provide parallel virtual machine at user level. But the functionalities are very limit since these system focus more on portable parallel programming model based on message passing paradigm. Most of operating system style services are left unimplemented. Glunix [8] is user-level distributed operating system for workstation cluster from project NOW (Network of Workstations). Glunix focus mostly on process control and I/O control. So the functionality is rather limited. Moreover, there are no version for Linux environment available at all. Therefore, Glunix is hardly used in Beowulf Cluster system.

SCore-D [9] is a parallel environment for Beowulf Cluster developed by Real-World Computing Partnership. SCORE-D provides only a limited APIs for programming support. But focus more on the support for efficient gang-scheduling for fine grain parallelism which is a different goal from KSIX.

3 Overview of KSIX

KSIX is a user level software, no kernel modification are required to run KSIX. This feature allows an easy installation and highly portable. KSIX is started by a bootstrapping utility called *kxboot* and stop with a utility *kxhalt*. After KSIX has been loaded, application can use KSIX services by enrolling into KSIX environment. This is done by calling a function *cpi_init()*. KSIX functions are in the form of *cpi_class_name* (CPI comes from Cluster Programming Interface). The term CPI has been used to separate the ideas of API and implementation in the same fashion as MPI. We view KSIX as one possible implementation of these API and services. KSIX Services and APIs can be classified into classes as follows.

3.1 Global Process Space

As the application calls KSIX to spawn a new task, this task will be distributed to a node in the cluster. This node has been selected using KSIX scheduling policy which will be open to modification in the future. KSIX allocates a global process ID and process group to this task. This id is used for task identification. There are 3 modes of task supported.

Normal Mode task acts the same as a normal task.

Restart Mode KSIX will automatically restart this task on the same node when task terminated.

Migration Mode KSIX start the task on different node when task termination is detected.

Table 1: KSIX Process Management APIs.

API	Description
int cpi_spawn(char *task, int flag, char *where, int ntask, int *tid, int *gid, int pclass)	Spawning tasks
int cpi_spawnIO(char *task, int flag, char *where, int ntask, int *tid, int *gid, char *output, char *error, int pclass)	Spawning tasks with specific location of output
int cpi_waitpid(int pid, int *status, int timeout)	Wait for process termination
int cpi_setpmode(int pid, int mode)	Change class of process
int cpi_setgmode(int gid, int mode)	Change class group of process
int cpi_pkill(int pid, int signal)	Send signal to process
int cpi_gkill(int gid, int signal)	Send signal to process
int cpi_allps(KxProcStat *result)	Report process status of all process
int cpi_userps(KxProcStat *result)	Report process status of user process

The reasons that we separate task into classes is to support a future implementation of process migration and high-throughput computing. Scheduler system can employ KSIX to take care of these tasks which help simplify the implementation of batch scheduling system dramatically. KSIX also support the creation of tasks groups, this will simplify the implementation of parallel programming system such as MPI2 since KSIX can help maintain group context which map directly to communicator concept in MPI. Finally, KSIX process control APIs also support the sending of UNIX signal, getting process information etc. The APIs are summarized in table 1.

3.2 Naming Services

In KSIX, processes can locate each other through naming service. The naming service

Table 2: Naming Services APIs

API	Description
int cpi_ds_reg(int, char *, struct servinfo, int *)	Register server with naming service
int cpi_ds_unreg(int, char *, int, int)	Unregister server with naming service
int cpi_ds_getinfo(int, char * ,struct servinfo, struct returninfo **)	Query information of server
int cpi_ds_free(struct returninfo *)	Free dynamic memory

APIs are as shown in table 2. Naming support allow service client and server to be connected logically and KSIX will help maintain low-level association information. This will simplify many programming tasks when using the cluster as a large server. With this service,a server process can register to a *logical service name*. Then, client process can bind with server using this *logical services name*. This allows the service server to be restarted or migrated to other node without any disruption of the service. Using this feature, we has built a feature called *Fault-Tolerance RPC* as shown in table 3. This feature can be used to link between stateless server and client and provides a basic level of high-availability. To test the concept, we has developed a simple textual database. First we start database server using cpi_spawn with migration mode. Client connected to server using this fault tolerant RPC. When the server has been kill, it has been automatic started on other node. This action is totally transparent to user client and the connection of frpc is virtually undisturbed.

3.3 Event Services

Distributed event notification and delivery is crucial part for the implementation of many high level services including High Availability services. KSIX also support event delivering between processes. Process can bind itself to named event. As the event is invoked or trigged by any process on any node. KSIX will reliably deliver the notification to the registered event

Table 3: Fault Tolerant RPC APIs

API	Description
KxFD *cpi_FRPC_cinit(char *service_name)	Initialize cpient
KxFD *cpi_FRPC_sinit(char *service_name)	Initialize server
KxFD *cpi_FRPC_accept(KxFD *cpifd)	Accept a connection on a socket
void cpi_FRPC_close(KxFD *cpifd)	Close a socket descriptor
int cpi_FRPC_send(KxFD *cpifd, void *buf, int size)	Send a message
int cpi_FRPC_recv(KxFD *cpifd, void *buf, int size)	Receive a message

owner. The APIs are as shown in table 4

3.4 Ensemble Management

For large cluster, system software, tools and application must be acknowledge about the change in cluster topology. KSIX subsystem that responsible for this task are called *ensemble management*. KSIX delete mal-function node from the ensemble automatically system it has been detected. KSIX also automatic add a new node to ensemble after the boot up process. The APIs for this class of service are illustrated in table 5.

4 KSIX Implementation

KSIX architecture consists of 2 layers of software: KSIX Kernel Layer and KSIX Services Layer(See Figure 4). KSIX Kernel layer provides basic services such as process control and ensemble management. This layer consists of KSIX daemon named KXD running on each node. Also, there is a system master con-

Table 4: Event Services APIs.

API	Description
int cpi_em_reg (int, void *, struct servinfo, int *)	Register event handler
int cpi_em_unreg (int, void *, int, int)	Unregister event handler
int cpi_em_raise (int, char *, struct servinfo, char *, int, struct answer **, int)	Raise event
int cpi_em_read (int *, char *, int, int *, struct timeval)	Event handler read message from event manager or raw TCP/IP
int cpi_em_write (int, char *, int)	Event handler write message to event manager
int cpi_em_free (struct answer *)	Free dynamic memory

troller called *KSIX partition manager(KXPM)* KXPM keeps track of global process ID and status of every processes in KSIX. Currently, KXPM and KXD organized into a form of tree as also shown in Figure 4. This allows us an easy support for global heartbeat and collective communication mechanism. KSIX Service layer is on top of KSIX kernel and provides high level KSIX services to KSIX based application. Each service will consist of service server, services and API library that link client of service to the service server itself. Current available services are Event service and Directory Service (DS). Since KSIX operate in pure message passing manner, KSIX services can be used by application on any node in the cluster.

5 Experimental Evaluation

One effective use of KSIX application is the use of fast process creation to support scalable unix commands. Currently, our SCMS cluster management tool provides a set of shell based parallel unix command with the package. KSIX can be used in stead of rsh to improve the performance especially for a short command. We have conducted the experiments as follows.

Table 5: Ensemble Management APIs.

API	Description
int cpi_addhost(char *hostname)	Add host to KSIX system
int cpi_delhost(char *hostname)	Delete host from KSIX system
int cpi_gethostbyrank(int rank, char *result)	Convert rank to hostname
int *cpi_getrankbyhost(char *hostname)	Convert hostname to rank
int cpi_getallhost(KxHostInfo *hostinfo)	Get array of hostname sort by rank

The experimental system is our Beowulf Cluster System called PIRUN. The system configuration is as listed below:

- 72 PentiumIII 500MHz with a memory of 128 Mbytes/nodes.
- 3 File server nodes. Pentium Xeon 500 Mhz and 1Gbytes memory.
- 100 Mbps Fast Ethernet Switches are used as the interconnection network among nodes.

The experiments has been conducted using a program that start a set of remote tasks on multiple hosts. This program will be referred to as master task. Master task starts a set of slave tasks on multiple hosts and wait for the reply messages from all of the slave tasks. Once started, each slave task sends a UDP message back to the master task. After received all UDP messages from every task, master tasks terminate. The time from the start of all the slave tasks to when the master task received all the reply has been measured. We compared two versions of the experiment. One using "rsh" mechanism to start the remote tasks and one using KSIX process service to start the remote tasks. The experiment has been repeated for five times on 4,8,16,32, and 64 processes on 4,8,16, and 32 nodes. The results shown is the average value obtained from each

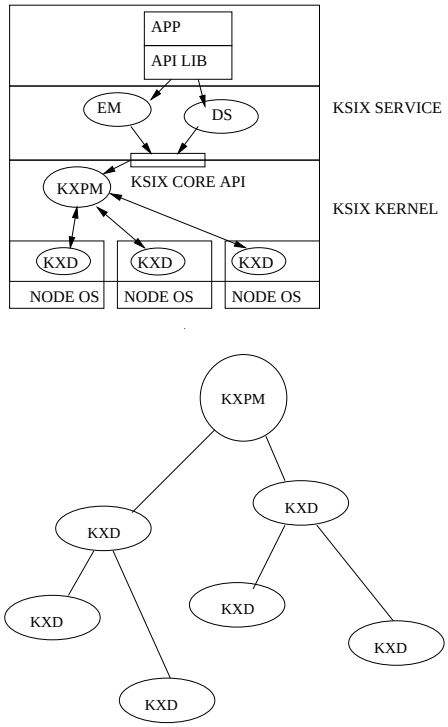

Figure 1: KSIX Architecture

data point. The purpose of this test is to simulate the working of scalable unix command and measure how fast the scalable unix command can work using KSIX support. The results of the test are as shown in Figure **??**. The results can be summarized as follows.

- As illustrated in Figure 2, KSIX can start 64 processes on 64 nodes using a little over 100 millisecond while rsh based implementation spent much longer time. Also, the increase in process startup time is much less in KSIX than in rsh based implementation. This means that KSIX scale much better for large cluster.
- Figure 3 shows speed up of KSIX implementation of process startup over rsh based implementation. As number of machines increases , KSIX speed up also increases. At 64 nodes, the speedup increases to almost 6 times. Again, this clearly show that using KSIX to implement scalable unix command will be much more efficient in a large cluster.

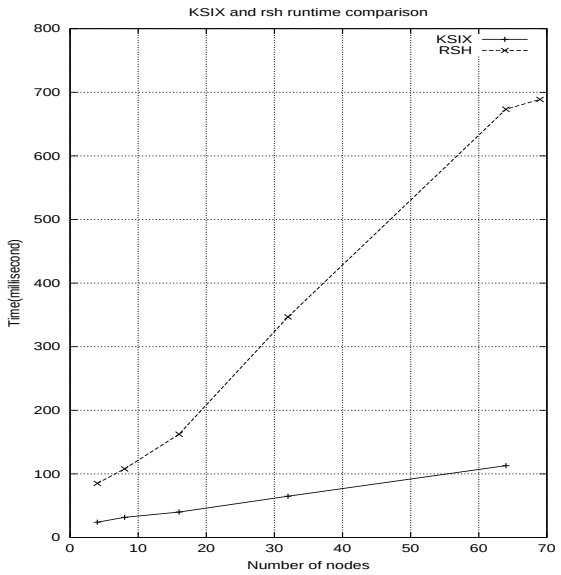

Figure 2: Comparison of KSIX and rsh based remote process creation time

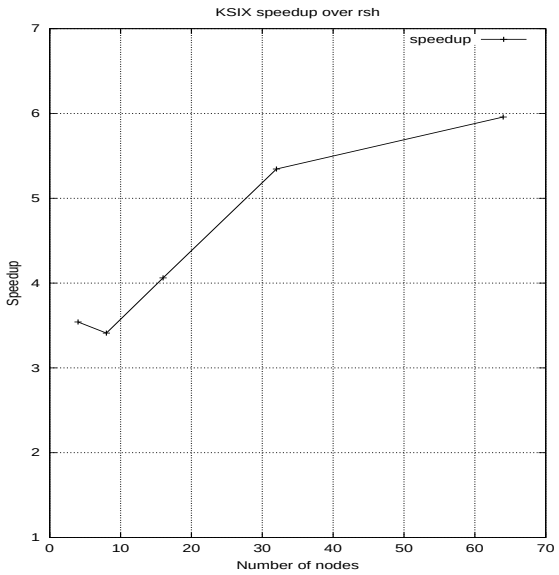

Figure 3: Speedup of KSIX over rsh based remote process creation time

The efficiency provided by KSIX is very important for scalable unix command. The reason is that it allows users to use these commands frequently without degrading the system performance.

6 Implementation Status

Currently all of the functions and APIs mentioned in this paper has already been implemented. KSIX version 1.1 is now available for download at http://smile.cpe.ku.ac.th/software. KSIX has been tested on 76 nodes Pentium III Beowulf Cluster that we currently have.

7 Conclusion

KSIX is a very improtant part of our full cluster environment called SCE (SMILE Cluster Environment) which consists of SCMS Cluster Management System, KCAP Web based management and Visualization system, and SQMS Queuing System. KSIX is a rapidly evolving project. Many improvements are currently working on such as:

1. Extending the APIs to meet more needs in cluster development. We are currently designing a simple file services. We do not intended to implement full support for parallel file system since there are many works such as PVFS that can be used. KSIX file service is intend for user to move and manipulate single file object easily and transparently.

2. Enhancing the scalability. Current implementation still have many centralize part since we focus on exploring API and services and deliver working system to researchers first. The plan is to make KSIX to scale to 10^4 nodes in the future.

3. Adding the high-availability support. We are extending KSIX so it can detect many performance problems at an early stage. When a malfunction has been detected, KSIX will help notify a correct subsystem that handle the recovery using KSIX event mechanism.

4. Implementing an extensible service architecture that allows developer to easily plug in a new service to KSIX.

With all these features, we hope to build a very effective open source environment for Beowulf cluster system that allows user to fully exploit the power of Beowulf cluster for their works.

References

[1] T. Sterling, D. J. Becker, D. Savarese, J. E. Dorband, U. A. Ranawake, and C. E. Packer, "Beowulf: A Parallel Workstation for Scientific Computation ," in *Proceedings of the International Conference on Parallel Processing 95*, 1995.

[2] P. Uthayopas, S. Paisitbechapol, T. Angskun, and J. Maneesilp, "System Management Framework and Tools for Beowulf Cluster," in *Proceedings of HPCAsia 2000*, (Beijing, China), May 2000.

[3] A. Barak, O. La'adan, and A. Shiloh, "Scalable Cluster Computing with MOSIX for LINUX," in *Linux Expo 99*, 1999.

[4] E. Pinheiro and R. Bianchini, "Nomad : A scalable operating system for clusters of uni and multiprocessors," in *Proceedings of IWCC99*, (Melbourne, Australia), December 1999.

[5] E. Hendriks, "BPROC : A distributed PID space for Beowulf clusters," in *Proceedings of Linux Expo 99*, (Raleigh Convention Center, Raleigh, N.C.), May 18-22 1999.

[6] V. Sunderam and J. Dongarra, "PVM: A Framework for Parallel Distributed Computing," *Concurrency: Practice and Experience*, pp. 315–339, 1990.

[7] W. Gropp, E. Lusk, and A. Skjellum, *USING MPI: Portable Parallel Programming with the Message-Passing Interface*. MIT Press, 1994.

[8] D. P. Ghormley, D. Petrou, S. H. Rodrigues, A. M. Vahdat, and T. E. Anderson, "GLUnix: A Global Layer Unix for a Network of Workstations," *Software-Practice and Experience*, vol. 28, pp. 929–961, 1998.

[9] A. Hori, H. Tezuka, and Y. Ishikawa, "An Implementation of Parallel Operating System for Clustered Commodity Computers," in *Proceedings of USENIX 99*, 1999.