

SCE: A Fully Integrated Software Tool for Beowulf Cluster System

Putchong Uthayopas, Thara Angskun, Somsak Sriprayoonskul, and Sugree Phatanapherom
*Parallel Research Group, CONSYL
Department of Computer Engineering, Faculty of Engineering
Kasetsart University, Bangkok, Thailand*

Abstract

One of the problems with the wide adoption of clusters for mainstream high performance computing is the difficulty in building and managing the system. There are many efforts in solving this problem by building fully automated, integrated software distribution from several open source software. However, these sets of software come from many sources and never been designed to work together as a truly integrated system. So, some problem is still remaining unsolved.

With the experiences and tools developed to build many clusters on our site, we decided to build an integrate software tool that is easy to use for cluster user community. This software tool, called SCE (Scalable Computing Environment), consists of a cluster builder tool, complex system management tool (SCMS), scalable real-time monitoring, web base monitoring software (KCAP), parallel Unix command, and batch scheduler (SQMS). This software run on top of our cluster middleware that provides cluster wide process control and many services. MPICH are also included. SCE are truly integrated since our group builds all tool but MPICH. SCE also provides more than 30 APIs to access system resources information, control remote process execution, ensemble management and more. These APIs and the interaction among software components allow user to extends and enhance SCE in many ways. To make things easy, the installation and configuration in SCE are fully automated completely by GUI. This paper will discuss the current SCE design, implementation, and experiences. SCE is expected to be available as a developer version in June.

1 Introduction

Beowulf Cluster [1] has been accepted as a platform that creates a great deal of impact in a HPC area. The reason is that Beowulf cluster bring extremely high computing power to scientists and engineers at a very low cost. As a result, the use of this platform rapidly increases during the last few years. Nevertheless, building and operating a Beowulf cluster requires many expertises. The problem is more severe as the system size grows. To address this problem, many research groups [2][3] have built many tools that help reduce the complexity of building and maintaining Beowulf clusters.

Recently, there are many packages such as SCYLD [4] and Oscar[5] that attempt to provide an integrated cluster distribution that includes all necessary tools in one package. Nevertheless, there are still some problems left unsolved. First, the configuration of each tool is still separated. Second, these tools usually start many components with duplicated functions. This is inefficient and wastes a lot of system resources. By building a fully integrated cluster environment, all these problems can be eliminated totally.

In this paper, SCE a fully integrated software tool for Beowulf cluster is introduced. SCE consist of a set of tools includes cluster builder tools, system management tool, real-time monitoring system, cluster middleware, batch scheduler, parallel Unix command and much more. These tools are building by the same research group and hence, the integration of these tools are considered since the beginning. Therefore, the installation and operation of SCE are very easy for users and system administrators.

The remainder of this paper is organized as follows. Section 2 explains some of the important term used. Section 3 gives an overview of SCE, follows by the description of SCE installation process in Section 4. Section 5 briefly describes the functionality of cluster builder tool called Beowulf Builder. Section 6 discusses about SCMS cluster management tool follows by the discussion of SQMS, batch-scheduling system in SCE. Section 8 presents the current status of SCE. Finally, Section 9 presents the conclusion and future work.

2 Terminology

In this section, the terms used through out this paper are defined. First, this paper assumes that standard cluster system consists of one or more *master node* and many of the *slave nodes*. This master node responsible for the management of the system and also act as a centralize file server for small cluster. Slave node is a complete computer that usually used to perform the computation. Slave node can be either diskless node or disk full node. For diskless node, the operating system and root file system is totally stored on master node. Although diskless node can have a local disk, this local disk is used for data storage and swap space only. In contrast, diskfull nodes use local storage as a boot device that store complete

operating system image. All operating system start up can and will be done locally. Master and slave nodes are connected through one or more *interconnection network*. This network is primarily used for message passing in parallel programming.

3 Overview of SCE

SCE consists of a set of feature rich software environment that allows users to easily build and maintain cluster configuration, monitoring various performance parameters, schedule sequential and parallel job. As shown in Figure 1, SCE consists of 4 main components. First, Beowulf builder is a software tools that create cluster and maintain cluster configuration. User use Beowulf Builder to automatically create all necessary configurations that allows a set of diskless nodes to remotely boot from master node. Once user finishes the installation, a middleware layer called KSIX [6] controls normal operations of a cluster. KSIX always run in background and provides many services to upper layer software tools. There are 2 main software systems running on top of KSIX, that is SCMS [7] cluster management system and SQMS batch scheduling system. SCE also include MPICH [8][9], one of the most widely use MPI implementation so user can start programming in parallel under SCE immediately after the installation finish. In the following section, each part of the system will be explained in more detail.

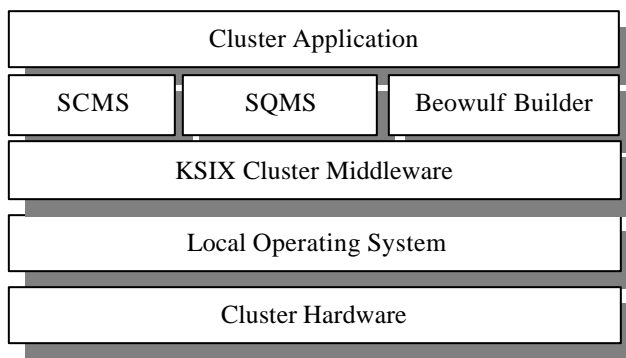


Figure 1: SCE Architecture

The clear advantage of SCE approach in integrating all cluster software tools together are as follows.

- Reducing the needs to do a complicated setup of multiple tools and keep the global configuration consistent.
- Sharing of software components make the system smaller, consume less system resources, and works faster.

- Software component interact better since they are design from the beginning to work together. For instance, batch scheduler can use middle service for process control and use performance-monitoring services to do a better resources management.

4 SCE Installation

A typical cluster configuration that SCE expect to see is as shown in Figure 2. SCE assume that a Beowulf cluster system consists of one or more master node and many slave node that connected through IP network. For users, SCE comes in 2 formats: A downloadable tar/gzip format or CD ready ISO file format. Once unpacked, a directory will be created. This directory will be referred in a subsequent explanation and SCE home directory.

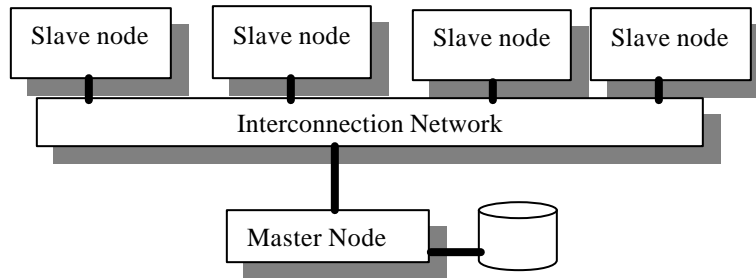


Figure 2: Typical SCE based Beowulf cluster system

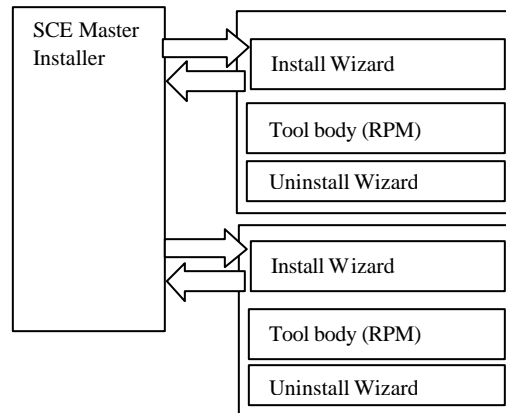


Figure 3: SCE Installation process

SCE installation begins by running file name *setup* found in SCE home directory. This operation start a component of SCE called *SCE Master Installer*. Main function of SCE master installer is to install the rest of SCE packages and additional library using rpm tool. The installation process is shown in Figure 3.

Basically, each SCE tool can be divided into 3 parts: Install Wizard that helps configure the tool initially, Tools Body which is the real working part of the tools, and Uninstall Wizard that do the cleanup. Therefore, in SCE, one can easily added more tools with only a minor modification in SCE master Installer. In addition, users will have an option of de-installation of a particular tool of required. During the installation, SCE master will run in the background and automate the invocation of each tool's wizard. The goal of SCE installation is to allow users to finish the installation process up to the point that MPICH can run without typing anything. This is possible by defining a good defaults and using a standard cluster platform.

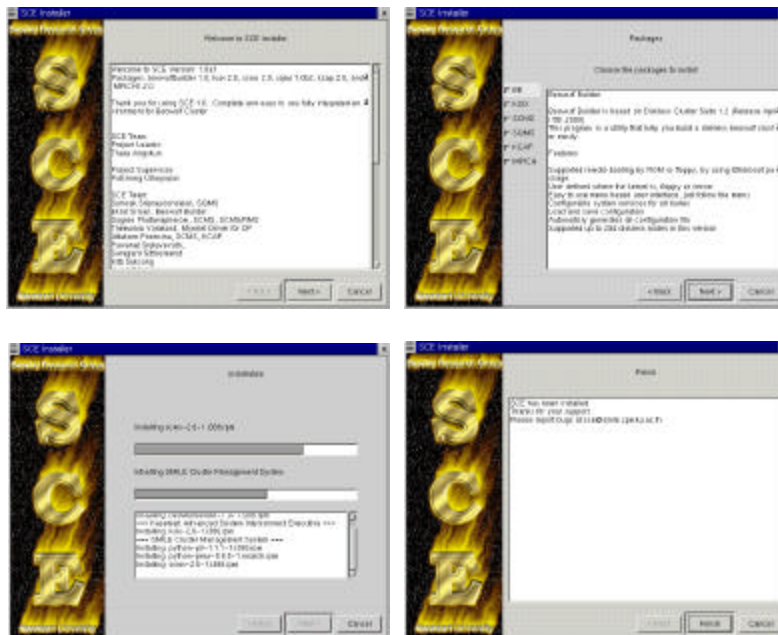


Figure 4: Screen shots of SCE 1.0 (Alpha version) Installation

5 Building Cluster with Beowulf Builder

Two major approaches are used to remotely installed slave nodes. LUI [10] software from IBM enables users to remotely boot slave node first, then use rpm command to remotely install RPM based packages are later. VA System Imager

[11] uses a different approach by having user install a complete slave node called golden slave first. Then, there are utilities that help user grab the configuration to centralize server and push it to multiple slave node.

In SCE, a tool called Beowulf Builder is built for the same purpose. The installation process of Beowulf builder use a little of previously described approaches. Basically, SCE assume that user must install the master node first, then run Beowulf builder later. Once running, Beowulf builder will have a wizard that get basic configuration information from users about slave nodes . The screen shot of Beowulf Builder Wizard is as illustrated in Figure 5.

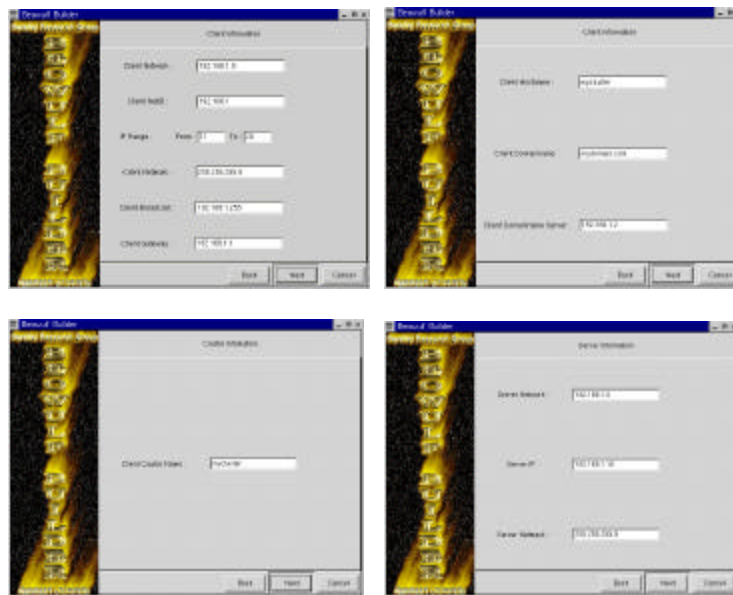


Figure 5: Screen Short of Beowulf Builder Wizard

After the configuration, Beowulf builder automatically extracts required system files and library to build root file system and /usr file system for each diskless slave node in /tftpboot. User have can use Beowulf builder to generate a boot floppy or binary image for NIC boot prompt. The easiest way to boot slave node is to insert this so-called “magic boot floppy” into floppy disk drive of slave node and power on that node. The boot process start automatically using combination of DHCP, TFTP based remote boot protocol. After slave node has been started, it will use NFS root file system and /usr file system on server. This approach makes the installation very automatic and work well for small cluster to about 100 nodes clusters

```

//
// SCE Cluster Building Process
//
main()
{
Install Linux RedHat Package;
SCEInstaller();
}
void SCEInstaller()
{
InstallKSIXrpm();   InstallSCMSrpm();
InstallKCAPrpm();  InstallSQMSrpm();
InstallExternalPackage();
// now we have packages in place on master node
ConfigandBuildCluster();
KSIXWizard();  SCMSWizard();
SQMSWizard();
// OK We are ready
RebootMaster();
    forallslave {
        BootSlavenode();
    }
}
void InstallExternalPackage()
{
    InstallPIL();      // Python Image Library
    InstallLibprg();   // our own component
}
void ConfigandBuildCluster()
{
    GetUserDefineNodeSetParameter();
    Loopcreatconfigure();
    bootmedia=GetBootMediaFromuser();
    Switch(bootmedia) {
        case Bootrom : Createbootromimage();
            break;
        case Floppy  : CreateBootFloppy();
            break;
        case CDROM   : CreateBootCDROM(); // not support yet
            break;
        case FLASH   : CreateBootFlashImage(); // not support yet
            break;
    }
}
void BootSlaveNode()
{
    SlavesendDHCP();      GetDHCPreply();
    LoadKernelbyTFTP();  BootKernel();
    MountNFSroot();      NormalBoot();
}

```

Figure 6: Beowulf Builder Boot Process

Process of building a cluster are as shown by pseudocode in Figure 6. Besides using Beowulf builder to install cluster, user can later use this software to customize cluster configuration as well. Web interface has been partly supported but not complete yet.

5.1 KSIX Cluster Middleware

KSIX is user level software, no kernel modification are required to run KSIX. This feature allows for easy installation and high portability. KSIX is start by a bootstrapping utility called *kxboot* and stop with a utility called *kxhalt*. After KSIX are loaded, applications enroll into KSIX environment by calling a function *api_init()* (CPI comes from Cluster Programming Interface). In the following subsection, services offered by KSIX are explained.

5.2 Global Process Space

KSIX application can use KSIX to spawn a new task, which is distributed among nodes in the cluster. KSIX uses an automatic scheduling policy to select the target nodes. The policy module is open to the modification in the future. KSIX also allocates a set of global process ID and process group for these tasks. The id is used for task identification in the subsequent call. There are 3 modes of task supported.

- Normal Mode: Task acts the same as a normal Unix 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.

KSIX process control APIs support the sending of UNIX signal, getting process information and more. These APIs are summarized in Table 1.

5.3 Naming Services

In KSIX, processes can locate each other through naming service. The naming service APIs are as shown in table 2. 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.

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

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

Table 3: Fault Tolerant RPC APIs.

API	Description
KxFD *cpi_FRPC_cinit(char *service_name)	Initialize client
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_rcv(KxFD *cpifd, void *buf, int size)	Receive a message size

5.4 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 triggered by any process on any node. KSIX will reliably deliver the notification to the registered event owner. The APIs are as shown in table 4

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

5.5 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 bootup process. The APIs for this class of service are illustrated in table 5.

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 following subsection gives some ideas about the application of KSIX in improving cluster environment. This support will be added in SCE in the near feature.

5.6 KSIX Support for Scalable Unix Tools

In cluster environment, the capability to issues a command to execute on every node and collect the results back is very important. User usually rely on rsh and ssh mechanism for remote command execution. But these command lack the

feature of collective operations. Hence, the execution is slow and not very scalable for large system. There is an effort to define a parallel extension to Unix command by parallel tool consortium. This SUT (Scalable Unix Tools)[12] effort is well explained in the literature. Currently, our tool, SCMS support an implementation of SUT in a form of shell script that rely on rsh for remote execution. Using KSIX fast and collective process management, a powerful SUT implementation can be done by replacing rsh with KSIX based remote execution command. Remote process can be started simultaneously on the remote machines to execute local Unix command. Then KXIO can be used to relay back the result efficiently.

5.7 KSIX Support for MPI2 Implementation

KSIX dynamic process management are designed such that process creation, process termination, process group, and signal delivery can be extended to support dynamic process management of MPI-2 standard with ease. MPI_COMM_SPAWN can be mapped to KSIX spawn. Process in KSIX always form into a group or context, this can easily be mapped to context-based concept of communicator in MPI. Parent and child group can be create by first create KSIX group, then using KX_Spawn to create child group. Group id and intercommunicator can be build and keep track later. Finally, efficient dynamic process creation and control provided by KSIX can be mapped directly to MPI2 approach and help ease the development effort greatly.

6 SCMS Cluster Management System

SCMS is a main tool that will be seen in SCE. SCMS is divided into 2 layers as illustrated in Figure 7.

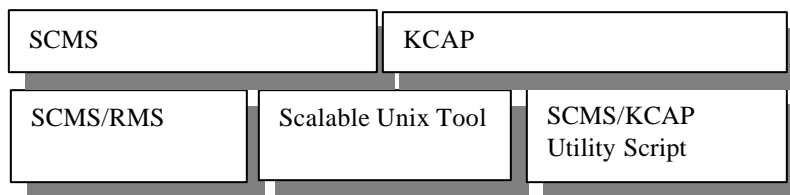


Figure 7 SCMS Architecture

SCMS lower layer is a set of daemon subsystems and utilities written in C, C++, and Python script language. This layer consists of:

- Scalable Unix tool: A parallel implementation of frequently used Unix commands that follows the guideline given by Parallel Tool Consortium.

- SCMS/KCAP Script which is a set of script written mostly in Python and Shell script. These scripts help perform many administrative task in the system.
- SCMS/RMS, a fast, scalable real-time monitoring and a set of powerful API in C, C++, Java, and Python that user can use to develop monitoring application. These API are as shown in **Table 6** and **Table 2**.

Table 6: RMI API for C language

RMI API	Description
int rmi_init(char *addr,int port);	Establish the connection
int rmi_finalize(int sd);	Close connection
int rmi_get_node(int sd,rmi_node_struct *nodes, int max);	Retrieve node name and node id of all alive nodes
int rmi_get_nodeid(int sd,rmi_int_struct *hid);	Retrieve node id of all alive nodes
int rmi_get_async(int sd,rmi_int_struct *hid, rmi_int_struct *pid,char *opt, char *buf,int max);	Retrieve objects in particular nodes to the buffer using asynchronous mode
int rmi_get_sync(int sd,rmi_int_struct *hid, rmi_int_struct *pid,char *opt, char *buf,int max);	Retrieve objects in particular nodes to the buffer using synchronous mode
int rmi_set(int sd,rmi_int_struct *hid, char *key,char *value);	Set internal variable "key" to "value"
int rmi_load_plugin(int sd,rmi_int_struct *hid, char *plugin);	Load plugin on specified node
int rmi_unload_plugin(int sd,rmi_int_struct *hid, char *plugin);	Unload plugin on specified node
int rmi_int_init(rmi_int_struct *lst,int max);	Allocate resource vector
int rmi_int_finalize(rmi_int_struct *lst);	Free resource vector
int rmi_int_add(rmi_int_struct *lst,int i);	Add to resource vector
int rmi_int_pack(rmi_int_struct *lst,char *buf, int max);	Pack list of integer to a string

The upper layer of SCMS consists of 2 main tools. SCMS and KCAP. SCMS is a GUI application based on Python and Tkinter. SCMS enable user to navigate, manage, monitor, and control the operation of Beowulf cluster from a single point. Some unique features of SCMS are:

- Interface to SCMS/RMS real-time monitoring
- Cluster Configuration collector and browser
- Control command that allows system administrator to shutdown, reboot any node or set of nodes.

- Innovative user interface in 3D grid format that allows user to manipulate thousands node cluster.
- Some of the screen shot from SCMS are illustrated in Figure 7.

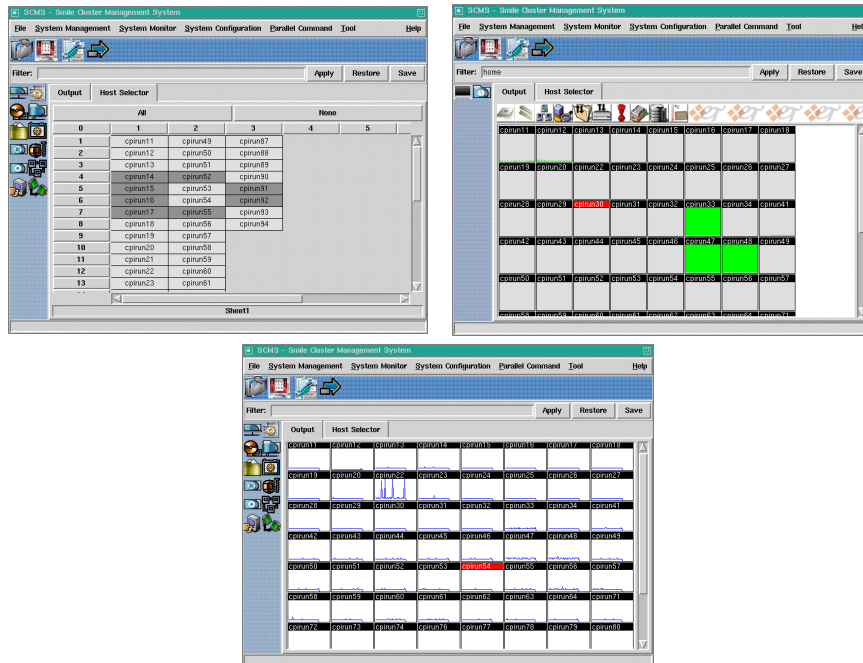


Figure 8: SCMS Screenshot showing (a) Host Selector (b) Real-time Monitoring (c) Heart Beat Checking (d) Configuration Browser

SCE also offers a web base monitoring package called KCAP that allows system administrator to check monitor system remotely,. Many most of the monitoring function appears in SCMS also available in KCAP as well. Also,

KCAP can help keep log of system performance, and perform cluster walk-in visualization using VRML and java based technology. Examples of KCAP user interface and cluster visualization are as shown in Figure 9.

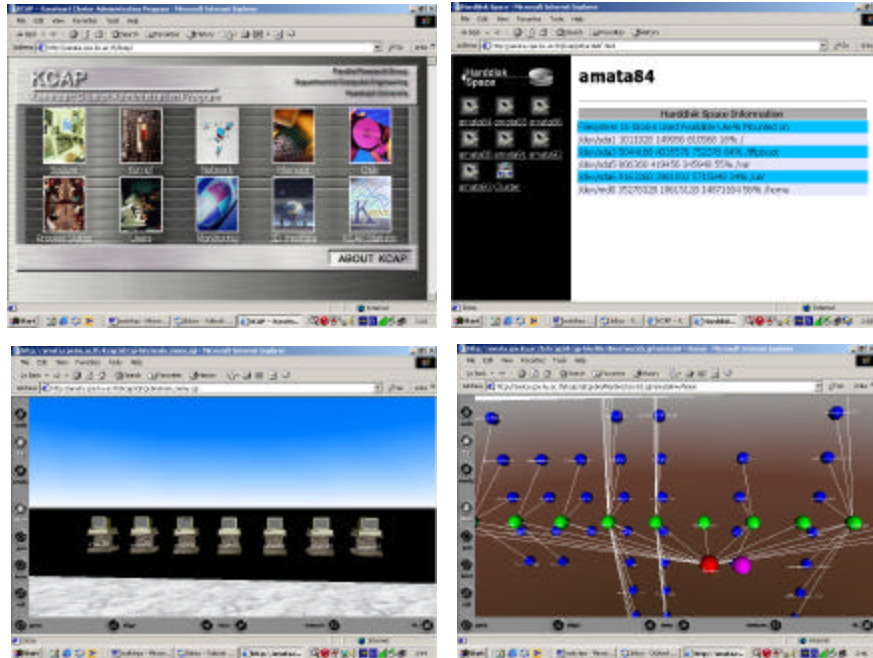


Figure 9: KCAP Screen shot showing the main menu, file system , 3D visualization of cluster nodes and file system in one node

7 SQMS Batch scheduling

One of the most important components in cluster software tool is a batch scheduler. Batch scheduler received user request for program execution, select optimum set of nodes, sending a job to run, and finally, collect the result back. There are many well-known batch schedulers such as OpenPBS [13], DQS [14]. Although very powerful, these schedulers are usually very complex to install, use, and maintain. SCE try to solve this problem by giving the scientists a simple but workable scheduler called SQMS. For more intricate requirement, user can also install more complicated scheduler such as OpenPBS for their use.

High-light of some features offered by SQMS are:

- Support both sequential and MPI based parallel program

- Provides command line to submit job, query queue status, and delete jobs.
- Move result into user home file system
- Support multiple form of reporting result such as email and ICQ users.
- Round robin, node
- Allows users to add new task scheduler
- Provides C/C++ API for user to develop complex load balancing policy if required.

```

[ubuntu@localhost ~]$ qstat
JobID  Name      Command      JobType  Status  Reason
13     h295sg/cpi1  local      wait    local   wait
14     h295sg/cpi1  mpi        wait    local   wait
15     h295sg/cpi1  local      wait    local   wait
16     h295sg/cpi1  mpi        wait    local   wait
17     h295sg/cpi1  local      wait    local   wait
18     h295sg/cpi1  mpi        wait    local   wait
19     h295sg/cpi1  local      wait    local   wait
20     h295sg/cpi1  mpi        wait    local   wait
21     h295sg/cpi1  local      wait    local   wait
22     h295sg/cpi1  mpi        wait    local   wait
3      h295sg/cpi1  local      run     cpirun63
4      h295sg/cpi1  mpi        run     cpirun64, cpirun65, cpirun66
5      h295sg/cpi1  local      run     cpirun67
6      h295sg/cpi1  mpi        run     cpirun68, cpirun71, cpirun72
7      h295sg/cpi1  local      run     cpirun73
8      h295sg/cpi1  mpi        run     cpirun74, cpirun75, cpirun76
9      h295sg/cpi1  local      run     cpirun77
10     h295sg/cpi1  mpi        run     cpirun78, cpirun81, cpirun82
11     h295sg/cpi1  local      run     cpirun83
12     h295sg/cpi1  mpi        run     cpirun84, cpirun85, cpirun86
[ubuntu@localhost ~]$
[ubuntu@localhost ~]$

```

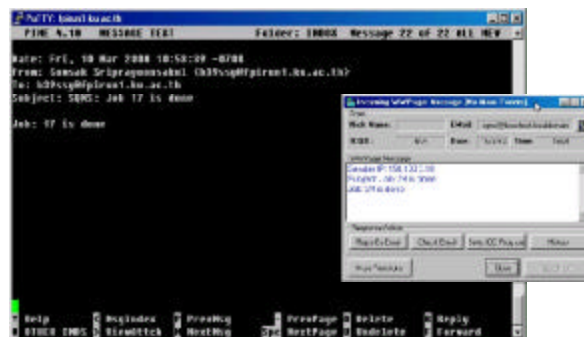


Figure 10: SQMS Screen Shot (a)listing the queue (b) result reported through mail or ICQ

These functions are enough to allow multiple users to submit jobs to the system. For more detail features comparison, please refer to Table 3. From Table 3, it seems that SQMS have much less feature than OpenPBS. This is due to the focus of the development that aims more toward the integration of software in this version. Moreover, the emphasis is to develop a simple and easy to use batch scheduler. So, many of OpenPBS features are ignore since it increase a learning time of the user and, in many cases, some of these features are hardly used.

8 Current Status of SCE

Current version of SCE, SCE alpha 1.0, is now available for early download at <http://smile.cpe.ku.ac.th/sce> . This version of SCE is intended to be a test version that developers can gain and early experience and feedback to SCE development team. Beowulf Builder is still not fully function since there is an ongoing work on a new version of builder tool that is much more powerful. According to the internal schedule, SCE 1.0 Beta1, which is more stable, will be released in June.

9 Conclusion and Future Direction

In this paper, SCE, a fully integrated Software Tool for Beowulf cluster has been partly described. SCE is a rapidly evolving and long-term project. The goal is to deliver a simple but high quality cluster environment for engineers and scientists who use Beowulf cluster to do their work. Many software in SCE is the results of more than 5 years of our software tool research effort.

SCE is a very active and long-term research program. There are many works that is being done now to improve SCE. First, SQMS team is now working quickly on improving SQMS in many ways. The focus will be on better support of parallel task, better scheduler, and more supporting tool that enhance system usability. Moreover, There is a new project called SCENIC (SCE on Network of Interconnected Clusters) that is investigating the addition of grid like capability so that all SCE based cluster to exchange computational task seamlessly.

For KSIX, there are many related projects to enhance its capability. For example, AMATA project that is now exploring the High availability support in middleware layer, SCK project currently produce kernel level checkpointing so KSIX2 which is due next year will start to partially have checkpointing support and process migration. Better integration with MPICH will be added into SCE. There is now a work on using KSIX, KXIO, and SCMS/RMS to build a debugger and runtime visualization software for MPICH. Finally, more services and tools will be added in the next releases to enhance the usability and power of SCE.

10 Acknowledgement

SCE project is sponsored by Kasetsart University Research and Development SRU Grant, Faculty of Engineering, Kasetsart University Grant. Many types of equipment and Athlon based cluster system used are sponsored by AMD Far East Inc.

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 International Conference on Parallel Processing 95, 1995
- [2] SMILE Project, Parallel Research Group, <http://smile.cpe.ku.ac.th>
- [3] R. Flanery, A. Geist, B. Luethke, and S. Scott, "Cluster Command & Control (C3) Tools Suite", <http://www.epm.ornl.gov/~sscott/>
- [4] SCYLD Beowulf, SCYLD Computing Corporation, <http://www.scyld.com/page/products/beowulf/>
- [5] OSCAR Linux distribution, Open Cluster Group, <http://www.epm.ornl.gov/oscar/>
- [6] Thara Angskun, Putchong Uthayopas, Choopan Ratanpocha, "KSIX parallel programming environment for Beowulf Cluster", Technical Session Cluster Computing Technologies, Environments and Applications (CC-TEA), International Conference on Parallel and Distributed Proceeding Techniques and Applications 2000 (PDPTA'2000), Las Vegas, Nevada, USA, June 2000
- [7] Putchong Uthayopas, Jullawadee Maneesilp, Paricha Ingongnam, "SCMS: An Integrated Cluster Management Tool for Beowulf Cluster System", Proceedings of the International Conference on Parallel and Distributed Proceeding Techniques and Applications 2000 (PDPTA'2000), Las Vegas, Nevada, USA, 26-28 June 2000
- [8] MPICH Portable MPI implementation, MCS, Argonne National Laboratory, <http://www-unix.mcs.anl.gov/mpi/mpich/>
- [9] W. Gropp, E. Lusk and A. Skjellum, "Using MPI: Portable Parallel Programming with the Message-Passing Interface", MIT Press, 1994
- [10] IBM LUI project, IBM Corp, <http://oss.software.ibm.com/developerworks/projects/lui>
- [11] VA System Imager, VA Linux, <http://systemimager.sourceforge.net/>
- [12] W. Gropp and E. Lusk, "Scalable Unix Tools on Parallel Processors", Proceedings of the Scalable High-Performance Computing Conference, May 23-25, 1994, Knoxville, Tennessee, 56-62, 1994.
- [13] OpenPBS web site, "www.openpbs.org"
- [14] DQS project web page, "<http://www.scri.fsu.edu/~pasko/dqs.html>"