SCE: Scalable Cluster Environment 1.0
User Manual Release 1.0

Parallel Research Group
Computer and Network Systems Research Laboratory
Department of Computer Engineering
Faculty of Engineering, Kasetsart University
Bangkok, Thailand

June 21, 2001

Scalable Cluster Environment Team.

Project Leader:
Thara Angskun

Project Supervisor:
Putchong Uthayopas

Developer:
Jullawadee Maneesilp
Somsak Sriprayoonsakul
Prakai Nadee
Uchot Jitpaisarnsook
Sugree Phatanapherom
Theewara Vorakosit
Ekrat Srisan
Attakorn Poonsina
Puwanat Sretavevuth
Suragard Sittisereerut
Krakit Dittatatree
Suchada Buaderm
Kitti Suksong
Pruettichai Somrongtong

Parallel Research Group
Computer and Network System Research Laboratory,
Department of Computer Engineering,
Faculty of Engineering, Kasetsart University.
Bangkok, Thailand.

For more information :
http://prg.cpe.ku.ac.th/research/sce
sce@prg.cpe.ku.ac.th

This work is supported in part by Faculty of Engineering Research Grant and Kasetsart University Research and Development Institute under SRU (Special Research Unit) Grant.

Update : | Version 1.0 | June | 2001 |

Contents

Chapter 1

Introduction

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

As shown in Figure 1.1, 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 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 run-

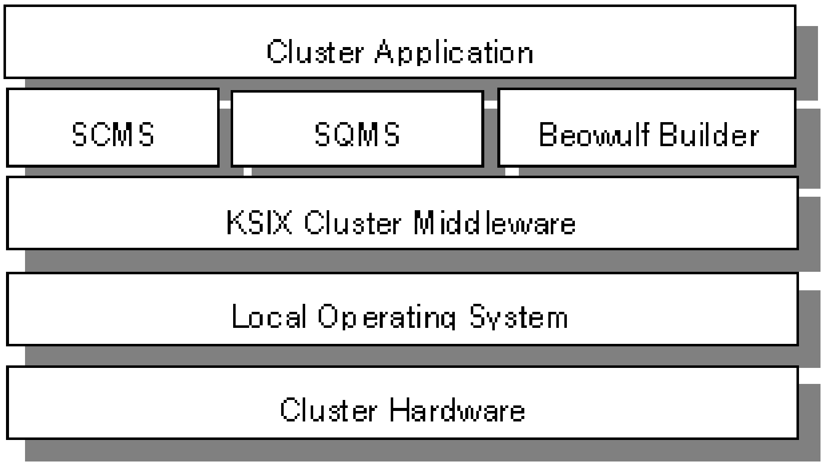

Figure 1.1: SCE Architecture

ning on top of KSIX, that is SCMS cluster management system and SQMS batch scheduling system. SCE also include MPICH, 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 chapter, each part of the system will be explained in more detail.

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.

Chapter 2

Installation Guide

SCE 1.0 is bundled with almost automatic installer wizard to help users on installation process. Users just answer some simple question and click on **Next** button to continue the process or click on **Back** button in case of you want to change the answer.

2.1 Requirements

- dhcp
- nfs-utils
- tk
- python
- ncurses
- perl
- chkconfig

2.2 Distribution

SCE 1.0 distribution has 2 formats:

1. sce-1.0.iso
2. sce-1.0.tar.gz

2.3 Installation

1. Download SCE 1.0 from http://prg.cpe.ku.ac.th/
2. Extract the archive

   ```
   $ su -
   # tar xzvf sce-1.0.tar.gz
   ```

3. Run **./setup**

   ```
   # cd sce
   # ./setup
   ```

   The splash screen as Figure 2.1 will appear for a few seconds.

4. The installer displays welcome message and software overview as Figure 2.2.

Figure 2.1: SCE Splash Screen

Figure 2.2: Welcome

5. Read license agreement and choose **accept agreement** to continue as Figure 2.3.

Figure 2.3: License Agreement

6. Choose the packages that you want to install it on the system as Figure 2.4

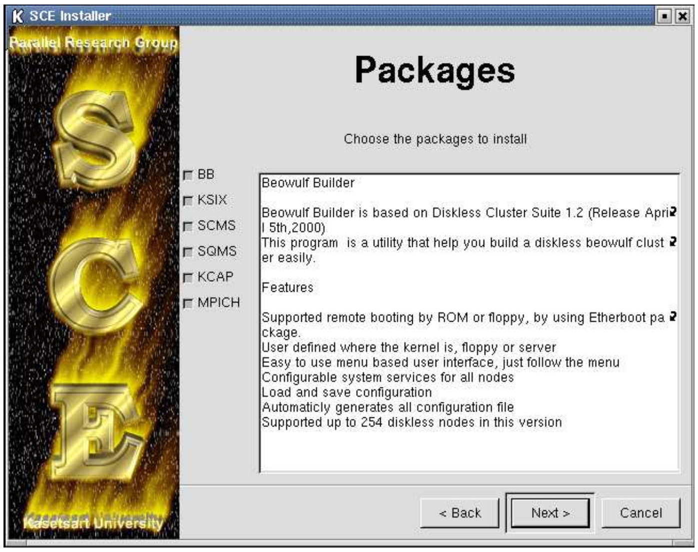

Figure 2.4: Packages

7. After confirm the packages as Figure 2.5, they will be installed one by one as Figure 2.6 and Figure 2.7.

Figure 2.5: Confirmation

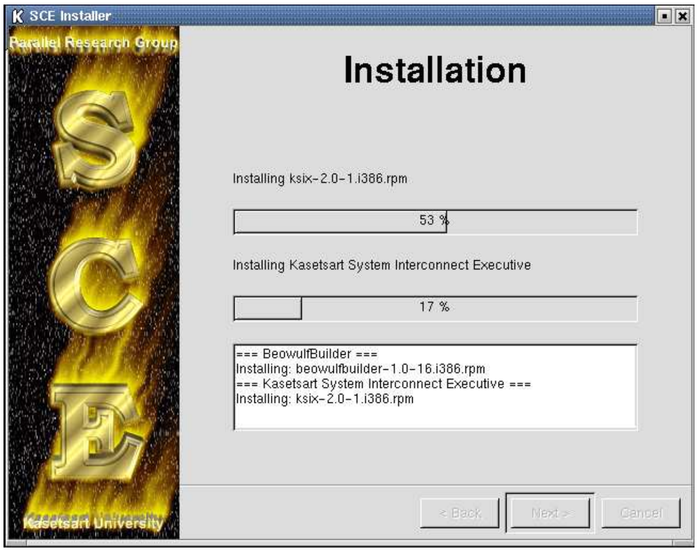

Figure 2.6: Installation

Figure 2.7: Installation Finish

8. If the installation have no any errors, the dialog as Figure 2.8 will be shown.

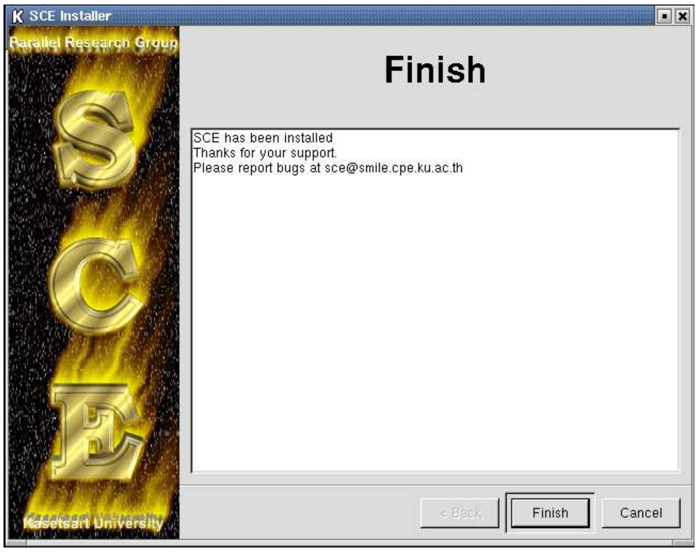

Figure 2.8: Finish

9. Next step is to build the cluster, we have 2 choices:
   (a) **Let Beowulf Builder build my cluster** Bring Beowulf Builder to build the cluster now (Figure 2.9).
   (b) **Use current configuration** We assume that the cluster is already installed properly, so the installer will skip Beowulf Builder Wizard (Figure 2.10).

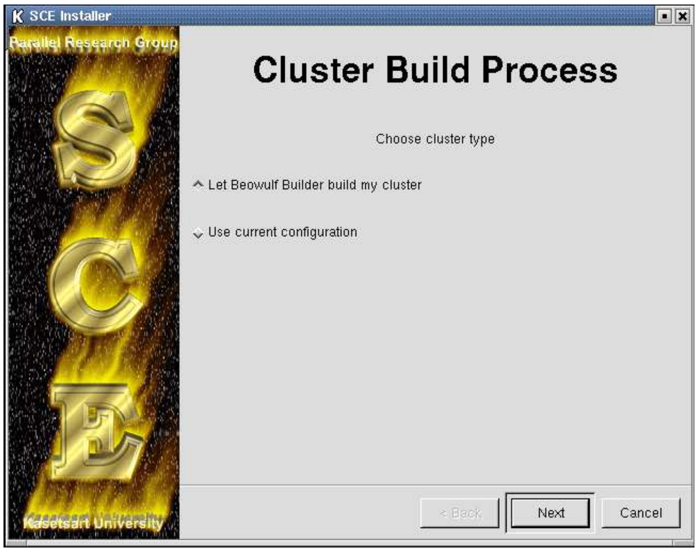

Figure 2.9: Cluster Build Process

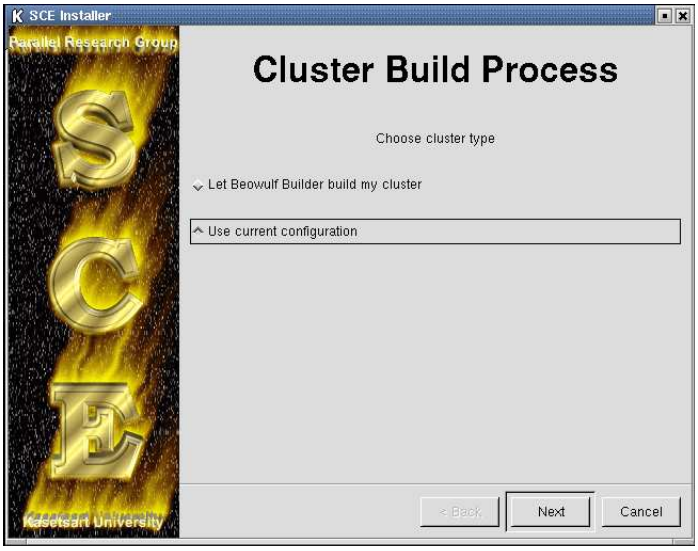

Figure 2.10: Use current configuration

After installation, the installer will bring up each Wizard of selected packages.

2.4 Beowulf Builder Wizard

1. **Warning!!** Beowulf Builder supposes to overwrite current configuration, please click on **Cancel** button to ignore the building.

2. Fill Network, IP, and Netmask of server in Figure 2.12.

Figure 2.11: Beowulf Builder Wizard Warning

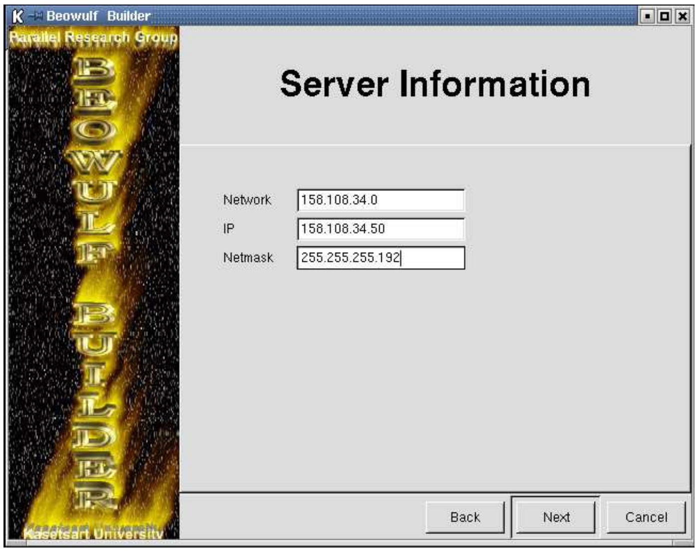

Figure 2.12: Server Information

3. Fill Network, Network ID, IP Range, Netmask, Broadcast, and Gateway of client in Figure 2.13.

4. Fill Based-host name, Domain Name, and Domain Name Server of the cluster in Figure 2.14.

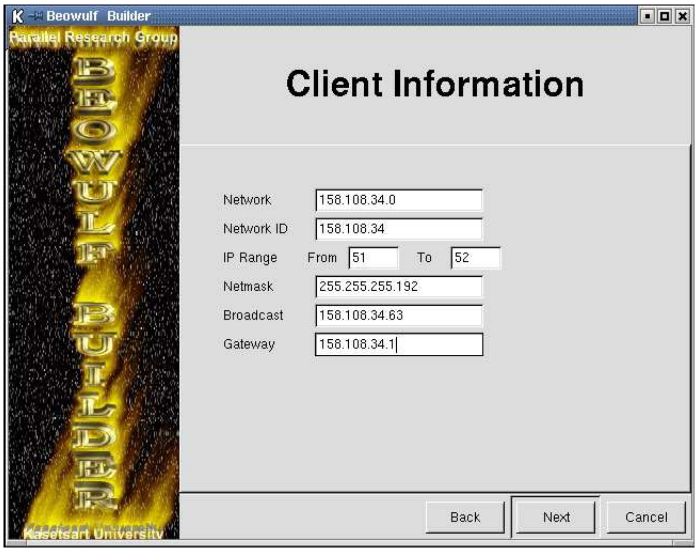

Figure 2.13: Client Information

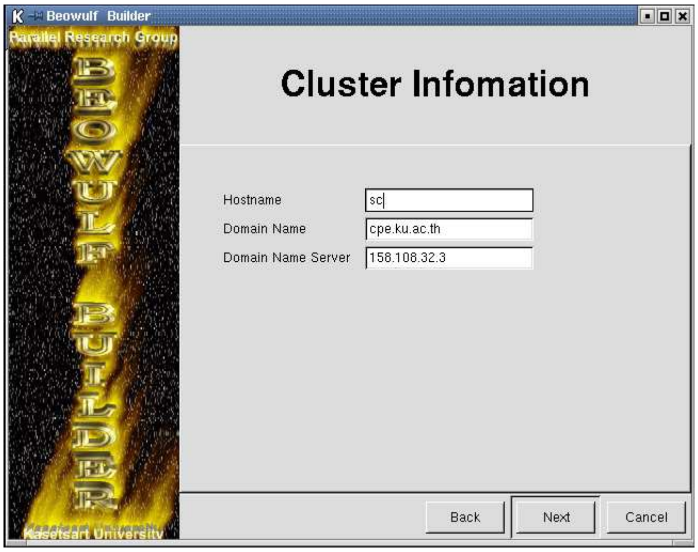

Figure 2.14: Cluster Information

5. Fill global cluster name in Figure 2.15.

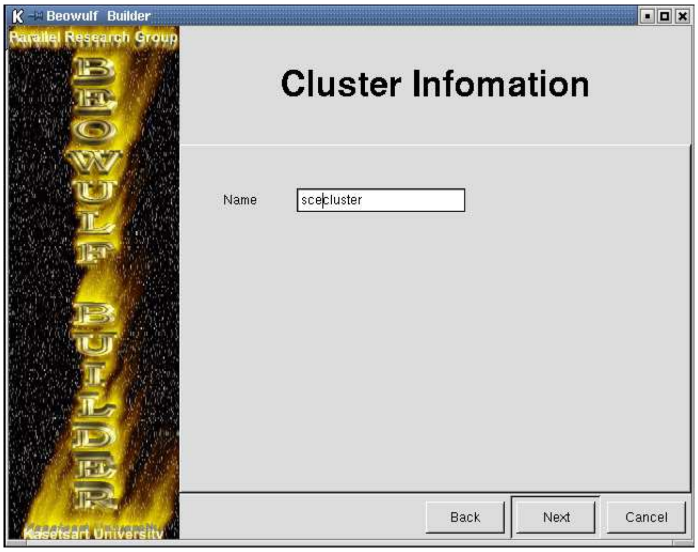

Figure 2.15: More Cluster Information

6. Next step, Beowulf Builder installs /usr, and root of each client as Figure 2.16 and Figure 2.17.

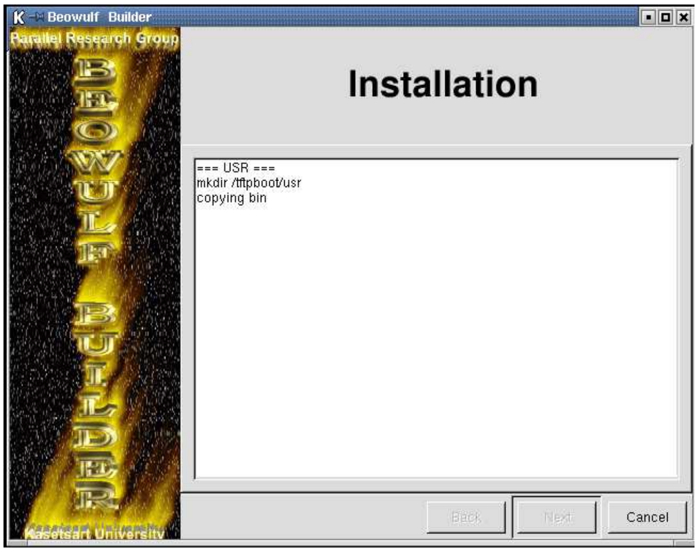

Figure 2.16: Installation

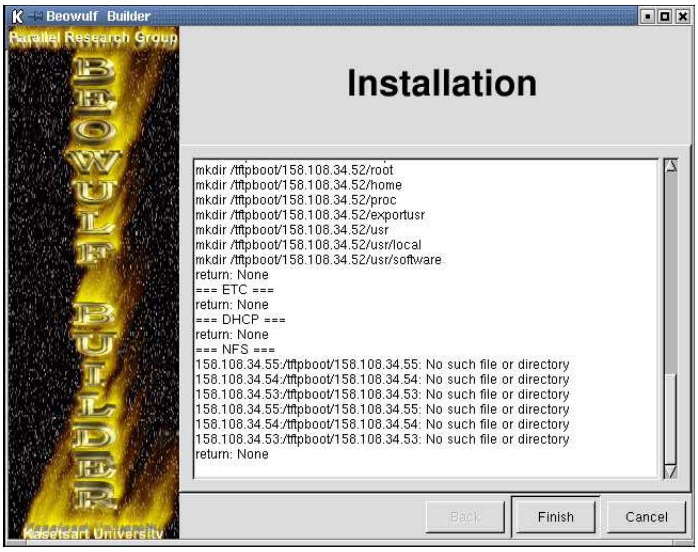

Figure 2.17: Installation Finish

2.5 SCMS Wizard

1. SCMS Wizard shows license agreement as Figure 2.18.

Figure 2.18: SCMS License Agreement

2. Choose SMA port, an unused port on all node as Figure 2.19.

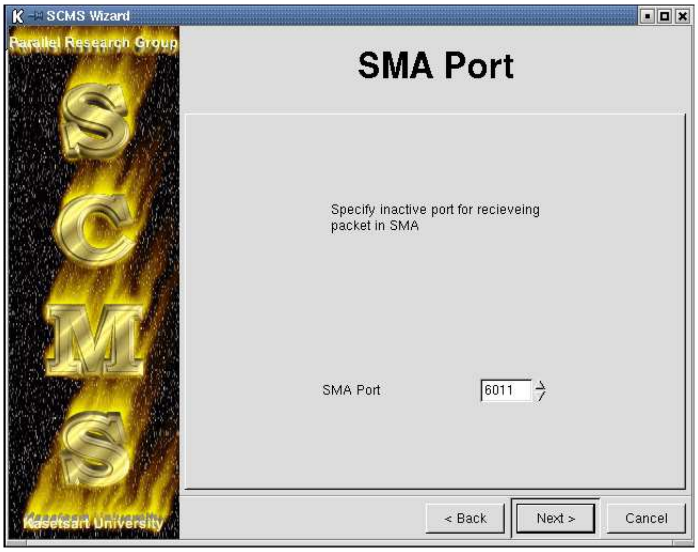

Figure 2.19: SMA Port

3. Choose CMA port, an unused port on all nodes as Figure 2.20.

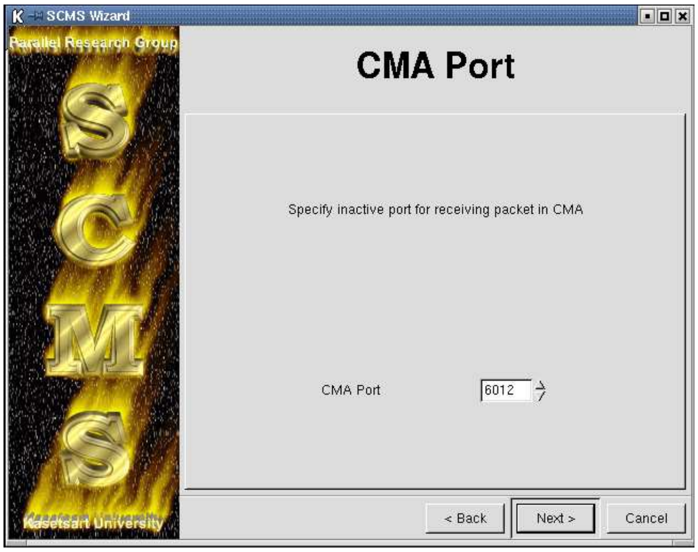

Figure 2.20: CMA Port

4. Choose RMI port, an unused port on all nodes as Figure 2.21.

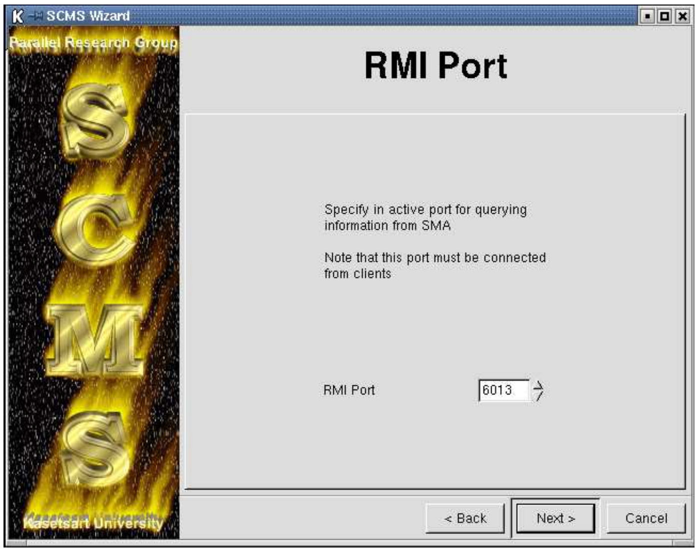

Figure 2.21: RMI Port

5. Choose number of partition, it depends on number of nodes as Figure 2.22.

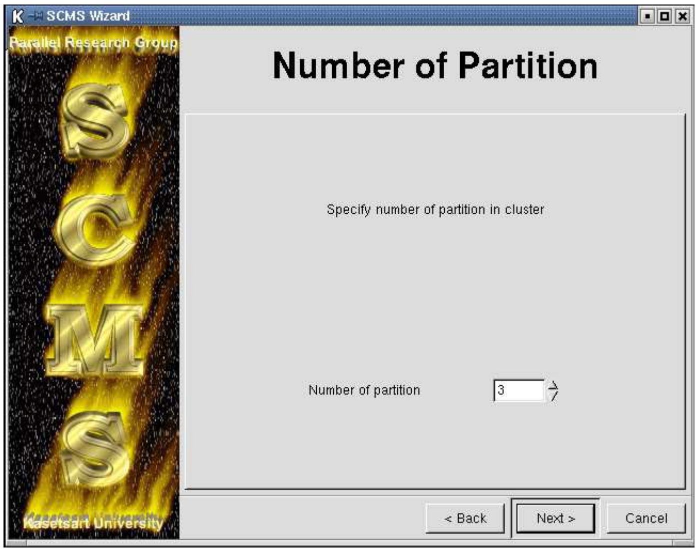

Figure 2.22: Number of Partition

6. Verify/Add/Remove/Change hostlist as Figure 2.23.

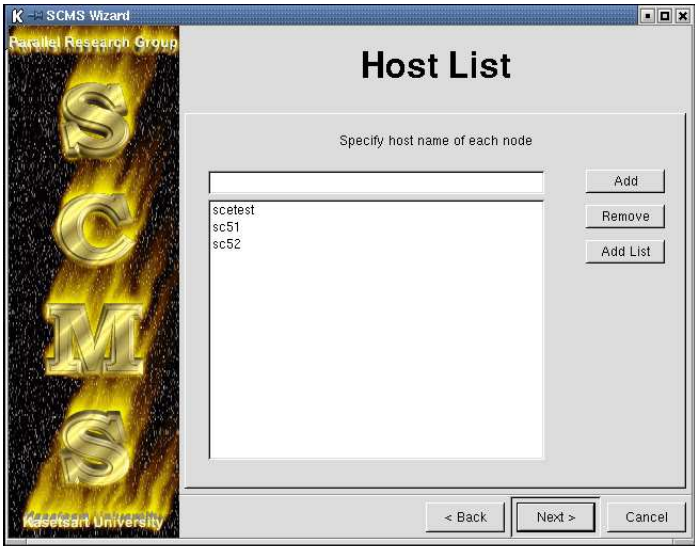

Figure 2.23: Hostlist

7. Fill hostname or IP of node which you plan to run **configd**.

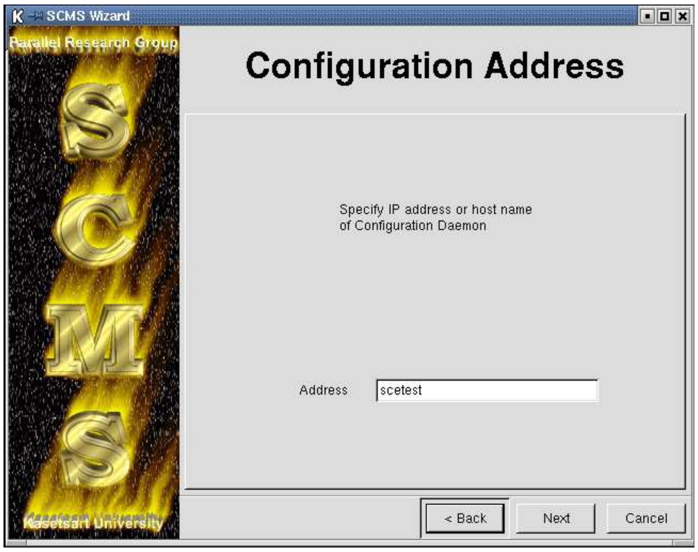

Figure 2.24: Configuration Address

8. Verify generated configuration as Figure 2.25. Be careful, if any errors occur, please back to previous steps.

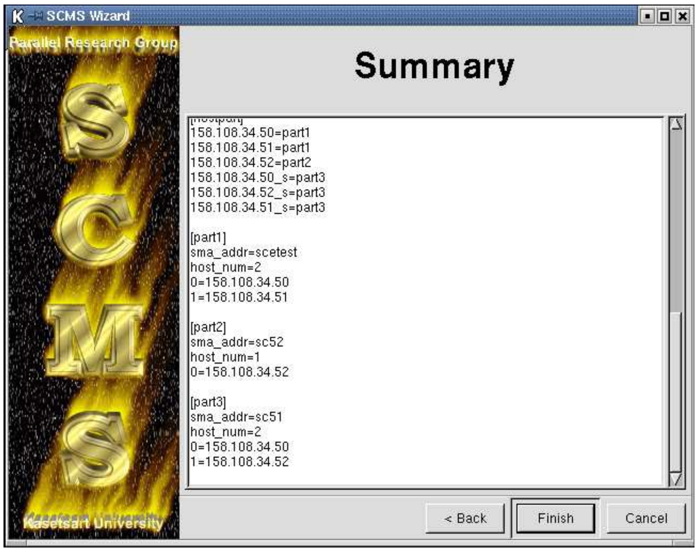

Figure 2.25: sce.conf

Chapter 3

Building Cluster with Beowulf Builder

3.1 Overview

BB – Beowulf Builder is a utility that help you build a diskless beowulf cluster easily.

3.2 Features

- Support remote booting using Etherboot package.
- Definable kernel localtion
- Easy to use menu-based user interface
- Configurable system services for all nodes
- Load and save configuration
- Automatically generate all configuration file
- Support up to 254 diskless nodes

3.3 Requirements

1. DHCP
2. NFS
3. TFTP
4. Newt library

3.4 Installation

1. Download the package from http://prg.cpe.ku.ac.th/
2. Extract the archive.

   ```
   # gzip -cd beowulfbuilder-x.x.tar.gz | tar -xvf -
   ```

3. Compile

   ```
   # make
   ```

4. Install

```
# make PREFIX=/usr install
```

3.5 Manual

3.5.1 Running

To run Beowulf Builder, simply run **beowulfbuilder**

```
# beowulfbuilder
```

3.5.2 Configuration

Beowulf Builder configuration consists of 3 parts:

1. **Cluster**
   - **Name** The name of the whole cluster.
2. **Server**
   - **IP** IP of the server.
   - **Netmask**
   - **Network**
3. **Client**
   - **Network ID** The first 3-byte of client IP; e.g. `192.168.1`
   - **Start IP** A number in range 0-255.
   - **Stop IP** A number in range Start IP-255.
   - **Gateway**
   - **Netmask**
   - **Network**
   - **Broadcast**
   - **Basename** Prefix name for each client to produce hostname in form `base\#\#`, while ## is client IP.
   - **Domain**
   - **DNS**

Beowulf Builder allows user to load/save configuration through GUI.

3.5.3 Generating Server

Since there are many step to build the cluster, Beowulf Builder separate them as the following:

- Generate DHCPd configuration
- Make a copy of /usr
- Generate client's root directory
- Generate NFS configuration
- Generate client configuration

To generate server, click on **server** button on the left panel. Administrator may choose to do only some steps, in case of he/she have done it before.

3.5.4 Editing client system services

Beowulf Builder allows administrator to edit system services of some/all client nodes easily by clicking on **client** button.

3.5.5 Creating bootdisk

To boot each node in the system, you need at least one bootdisk. Beowulf Builder let administrator to create bootdisk easily using GUI tool by clicking on **boot** button.

You have to choose where kernel is:

- floppy
- server (need Boot ROM)

And network interface driver also, since it depends on network interface card.

3.5.6 Upgrade bootimage

1. Compile customize kernel as you need

```
# cd /usr/src/linux
# make menuconfig
# make dep;make clean;make bzImage
```

2. Mount $prefix/share/beowulfbuilder-1.0/image/bootImage via loop device

```
# cd /usr/share/beowulfbuilder/image
# mkdir tmp
# mount -o loop bootImage tmp
```

3. Copy your kernel image to overwrite **vmlinuz**

```
# cp /usr/src/linux/arch/i386/boot/bzImage bootImage
```

4. Unmount

```
# cd ../..
# umount tmp
```

Chapter 4

SCMS Cluster Management System

4.1 Overview

SCMS – Scalable Cluster Management System – is a very interactive and extensible cluster management tool. The goal of SCMS is to allow users to perform the administrative task easily. SCMS provides powerful graphics user interface, huge amount of command tools, real-time monitoring subsystem, web-based interface and more. With SCMS, the system administration task of large cluster will become much simpler.

4.2 Features

- System Management Tools
- Real-time Performance Monitoring
- Parallel Unix Command
- Configuration Browser

4.3 Requirements

- PIL – Python Imaging Library
- Pmw – Python Megawidgets
- libprg – Parallel Research Group Library

4.4 Installation

1. Extract the archive.

   ```
   # tar xzvf scms-2.0.tar.gz
   ```

2. Run **./configure** to configure the system

   ```
   # cd scms-2.0
   # ./configure
   ```

   SCMS fully supports GNU-style autoconf which provides additional arguments listed below:

- –prefix
- –with-etcdir
- –with-piddir
- –with-libprg

3. Run **make** to compile.

```
# make
```

4. Run **make install** to install.

```
# make install
```

5. Run **make rpm** to build RPM.

```
# make rpm
```

4.5 Manual

4.5.1 Running

To start scms, simply type **scms**

```
$ scms
```

4.5.2 Configuration

SCMS configuration files are located in `ETCDIR/sce`, there are 2 necessary files:

1. scms
2. sce.conf

ETCDIR/sce/scms

`scms` uses shell-style configuration file: Sharp sign (`#`) at the first character indicates commentation; An assignment forms by key and value seperated by equal sign (`=`) without among additional spaces. **This file must be duplicated to every nodes in the cluster.**

- **CONFIGD_ADDR** – Hostname or IP that you suppose to run. `configd`
- **CONFIG_FILE** – The absolute path to configuration file. Default is `/etc/sce/sce.conf`

```
#
# scms startup configuration
#

# hostname or ip which you supposed to run configuration server
CONFIGD_ADDR=amata1

# absolute path to configuration file
CONFIG_FILE=/etc/sce/sce.conf
```

ETCDIR/sce/sce.conf

`sce.conf` uses section/option-style based on `ConfigParser` in python. There are few necessary sections:

1. **main** – For future using.
2. **scms** – Keep SCMS configuration, specially network ports.
3. **plugin** – Specify default plug-in.
4. **hostlist** – List all host's name.
5. **partlist** – List all partition's name.
6. **hostpart** – Associate host to partition.
7. **part*** – Subsection of each partition represents host in partition and SMA address for them.

```
[main]

[scms]
sma_port=6011
cma_port=6012
rmi_port=6013
sma_rlog_port=9900
cma_rlog_port=9910

[plugin]
#plugin_path=<plugin_path>
plugin_num=2
0=libhwplugin.so
1=libpsplugin.so

[hostlist]
host_num=12
0=amata1
1=amata2
2=amata3
3=amata4
4=amata5
5=amata6
6=amata7
7=amata8
8=amata9
9=amata10
10=amata11
11=amata12

[partlist]
part_num=1
0=part1

[hostpart]
158.108.35.35=part1
158.108.35.36=part1
158.108.35.37=part1
158.108.35.38=part1
```

```
158.108.35.39=part1
158.108.35.40=part1
158.108.35.41=part1
158.108.35.42=part1
158.108.35.43=part1
158.108.35.44=part1
158.108.35.45=part1
158.108.35.46=part1
158.108.35.35_s=part1

[part1]
sma_addr=amata1
host_num=12
0=158.108.35.35
1=158.108.35.36
2=158.108.35.37
3=158.108.35.38
4=158.108.35.39
5=158.108.35.40
6=158.108.35.41
7=158.108.35.42
8=158.108.35.43
9=158.108.35.44
10=158.108.35.45
11=158.108.35.46
```

4.5.3 Real-time Monitoring Subsystem

SCMS/RMS consists of 3 necessary daemons:

1. **System Management Agent** (SMA)
2. **Control and Monitoring Agent** (CMA)
3. **Resource Management Agent** (RMI)

Normally, CMA should always run on all nodes and periodically forwords data to its SMA. Using above method, SCMS allows administrator to configure flexible connection in hierarchical structure by setting SMA to have its SMA.

We define **Partition** as a group of CMAs or SMAs and a SMA. That means each CMA or SMA in the same partition have the same SMA.

After installation, all three daemons are started as system services at start up (RedHat). The user may start it manually using init.d script.

```
# /etc/rc.d/init.d/configd start
# /etc/rc.d/init.d/sma start
# /etc/rc.d/init.d/cma start
```

Note that `configd` **must** be started before sma and cma. However, in case of the user want to stop/start/restart SCMS from scratch, SCMS provides the special scripts:

```
# cms_start
# cms_stop
```

Chapter 5

KCAP Web-based Cluster Management System

5.1 Overview

KCAP is a web-based software that help monitor and manages a Beowulf Cluster. This software can monitor and display system resources interactively. Moreover, KCAP provides many features that helps both system administrators and users use the system effectively. KCAP is additional part to SCMS Cluster management system.

SCMS Must be installed properly before KCAP can be installed and used. This software is part of SMILE Project, Parallel Research group, Computer and Network System Research Laboratory, Department of Computer Engineering, Kasetsart University

5.2 Features

- Cluster configuration can be browse using a standard web browser.
- Display various useful information from any node or all nodes in the cluster. This information includes:
  - **Kernel Message** display kernel message.
  - **Kernel Module** display list of active kernel module.
  - **Network Config** display network configuration and routing information.
  - **Harddisk Space** display information about disk space.
  - **Memory Space** display about memory left in each node.
  - **User Status** display about user in each node.
  - **Process Status** display about process status of the cluster.
- Use Java Applet to display real-time information such as:
  - **CPU** display about user cpu, system cpu, nice cpu, and idle cpu.
  - **Context Switch** display about number of context switch.
  - **Board Temperature** display about cpu temperature and fan speed.
  - **Load Average** display about computer load in every 1,5,15 minutes, running process and number of process
  - **Memory** display about used memory, idle memory, shared memory and buffer.
  - **Page** display about number of page in and page out.
  - **Swap** display about swap page.

5.3 Requirements

1. SCMS latest version
2. Java Run-time Environment

5.4 Installation

1. Download the package from http://prg.cpe.ku.ac.th
2. Extract the archive

   ```
   # tar xzvf kcap-x.x.tar.gz
   ```

3. Generate configuration

   ```
   # make config
   ```

4. Install KCAP

   ```
   # make install
   ```

5. Install KSTAT

   ```
   # make kstat
   ```

Chapter 6

SQMS Batch Scheduling

6.1 Overview

SQMS is a simple queueing management system developed to simplify the task management in beowulf cluster. It was a part of SCE (Scalable Cluster Environment, http://smile.cpe.ku.ac.th/research/sce). SQMS provides a set of command to support developer in beowulf cluster environment to submit, query status, deleting his/her program to the computing node(s) in beowulf cluster. SQMS is released under Opensource license (BSD based).

6.2 Features

1. Support any type of cluster matches above requirements
2. Support both sequential task and parallel task, currently supports only MPI program
3. Developer can easily creates a new Load Balancing Policy using our API

6.3 Requirements

1. GNU/Linux RedHat 6.0 or later with 2.2.x kernel
2. rsh and rlogin
3. pthread
4. A shared directory which is shared on all nodes in cluster system. SQMS needs to be installed on this type of directory for now.

6.4 Installation

1. Download the package from http://prg.cpe.ku.ac.th/
2. Unpack SQMS Distribution

   ```
   # gzip -dc sqms-1.x.x.tar.gz | tar -xvf -
   ```

   or

   ```
   # tar xzvf sqms-1.x.x.tar.gz
   ```

   After unpack, the directory structure of scms will be as listed below.

   - **README** Release Note, read this first

- **src** directory containing source program
- **doc** directory containing document
- **bin** directory containing the executable file.
- **lib** directory containing the SQMS libraries
- **api** directory containing SQMS API.
- **include** directory containing all header files.
- **man** directory containing manual page.
- **plugpol** directory containing plug-in policy example.

3. Compile and install SQMS. Currently, SQMS support only RedHat-6.x or RedHat 7.0. First, change directory to scms directory after unpack and then:

   ```
   # configure
   ```

   or

   ```
   # configure --prefix=<install directory>
   # make
   # make install
   ```

   where ¡install directory¿ is where you want SQMS to install. Default is /usr/software/sqms-¡version¿

4. Add ¡install directory¿/sqms-1.x.x-x/bin to your default path.

   ```
   export PATH=$PATH:<install directory>/sqms-1.x.x-x/bin
   ```

   Example:

   ```
   export PATH=$PATH:/usr/software/sqms-1.0b-2/bin
   ```

6.5 Manual

6.5.1 Start SQMS Services

After config SQMS. You can start it by:

```
# $etcdir/rc.d/init.d/sqms start
*****************
Start each daemon
*****************

Starting SQMS scheduler                                      [  OK  ]
Starting SQMS loadbalancer                                   [  OK  ]
Starting SQMS dispatcher                                     [  OK  ]
```

Or you may start it manually by:

```
# $sbindir/scheduler
# $sbindir/loadbalancer
# $sbindir/dispatcher
```

6.5.2 Stop SQMS Services

If you want to shutdown SQMS. You can do it by:

```
# $etcdir/rc.d/init.d/sqms stop
```

or

```
# killall scheduler loadbalancer dispatcher
```

6.5.3 Submit job

sqsub – submit a job to SQMS. A job is any executable program. For example

```
$ sqsub myprog
1
```

Output and error will be redirected to myprog.1out, myprog.1err. 1 is Job ID. If you want to apply arguments to your program. You should create a shell script contains your program and arguments. For example, you might create a file called 'myprog.sh' which contains

```
#!/bin/sh
myprog arg1 arg2 arg3 arg4 arg5
```

and submit this program to SQMS by

```
$ sqsub myprog.sh
```

For more information, try `man sqsub`

6.5.4 Delete job

sqdel – When you want to delete a job from SQMS. You can do this by:

```
$ sqdel 1
```

where 1 is job id. For more information, please try `man sqdel`

6.5.5 View job status

sqstat – When you want to see your job status. Just type

```
$ sqstat
JOBID   OWNER   COMMAND         JOBTYPE STATUS  RUNHOSTS
28      ssy     test            local   wait
29      ssy     test            local   wait
30      ssy     test            local   wait
3       ssy     test            local   run     amata11
5       ssy     test            local   run     amata1
6       ssy     test            local   run     amata2
7       ssy     test            local   run     amata3
8       ssy     test            local   run     amata4
9       ssy     test            local   run     amata5
10      ssy     test            local   run     amata6
11      ssy     test            local   run     amata7
15      ssy     test            local   run     amata11
17      ssy     test            local   run     amata1
18      ssy     test            local   run     amata2
19      ssy     test            local   run     amata3
20      ssy     test            local   run     amata4
```

```
21      ssy     test            local   run     amata12
22      ssy     test            local   run     amata1
23      ssy     test            local   run     amata2
24      ssy     test            local   run     amata3
25      ssy     test            local   run     amata4
26      ssy     test            local   run     amata5
27      ssy     test            local   run     amata6
```

For more information, please try `man sqstat`

6.5.6 Configuration

Normally, SQMS reads configuration from 2 files:

1. $etcdir/sce/sqms.conf
2. $etcdir/sce/sqms_hostlist

Since $etcdir is etc directory that was set on compile time using option –with-etcdir. After fresh SCE installation, SQMS RPM will determine the installed hostname automatically. SCE wizard will update sqms_hostlist to match the cluster configuration. If you wish, you can edit sqms_hostlist by hand.

$etcdir/sce/sqms.conf

This file should exists on each nodes in the system. If not, user will not be able to submit the job from the host other than installed host.

- working_directory The path to directory which SQMS will uses as working directory. Usually $prefix/var/sqms
- scheduler_host The name of the host running SQMS scheduler.
- scheduler_port The port number which scheduler bind.
- dispatcher_host The name of the host running SQMS dispatcher.
- dispatcher_port The port number which dispatcher bind.
- dispatcher_port The port number which dispatcher bind.
- loadbal_host The name of the host running SQMS load balancer.
- loadbal_port The port number which load balancer bind.
- heartbeat_time Time interval between each hostlist update.
- max_exec_job The length of run queue.

$etcdir/sce/sqms_hostlist This file contains hostlist of the cluster. One node per line. You can weight the host by adding duplicate entry in the list.

Example

/etc/sce/sqms.conf

```
################################################################################
#Configuration file for SQMS
#$Id: sqms.tex,v 1.5 2001/06/21 08:55:22 b40sup Exp $
################################################################################

#Below is the current working directory of SQMS. If not set. SQMS 'll
```

```
#determine the working directory from the program's name
#working_directory   /usr/local/share/sqms

#Default scheduler, dispatcher, load balancer address and port
#These daemon might be in different host. But the hosts should shared the
#working directory of SQMS

#Scheduler address
scheduler_host    amata1.cpe.ku.ac.th
scheduler_port    3456

#Dispatcher address
dispatcher_host   amata1.cpe.ku.ac.th
dispatcher_port   3457

#Load balancer address
loadbal_host      amata1.cpe.ku.ac.th
loadbal_port      3458

#Define Maximum execution job
max_exec_job       20

sqms_hostlist amata1
amata2
amata2
amata2
amata3
amata4
amata5
amata6
amata7
amata8
amata9
amata10
amata11
amata12
```

In this example. amata2 will have more job than other host

Chapter 7

KSIX Cluster Middleware

7.1 Overview

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.

7.1.1 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.

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 7.1.

7.1.2 Naming Services

In KSIX, processes can locate each other through naming service. The naming service APIs are as shown in table 7.2. 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

Table 7.1: 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 7.2: 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 7.3: 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

Table 7.4: 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

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 7.3. 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.

7.1.3 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 owner. The APIs are as shown in table 7.4

Table 7.5: 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

7.1.4 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 7.5.

7.2 KSIX Implementation

KSIX architecture consists of 2 layers of software: KSIX Kernel Layer and KSIX Services Layer(See Figure 7.2). 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 controller 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 7.2. 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.

7.3 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.

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.

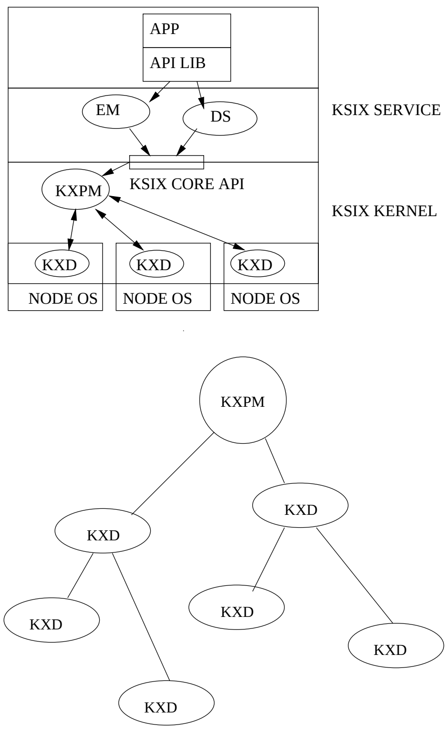

Figure 7.1: KSIX Architecture

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 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 7.3.

The results can be summarized as follows.

- As illustrated in Figure 7.2, 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 7.3 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.

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.

7.4 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.

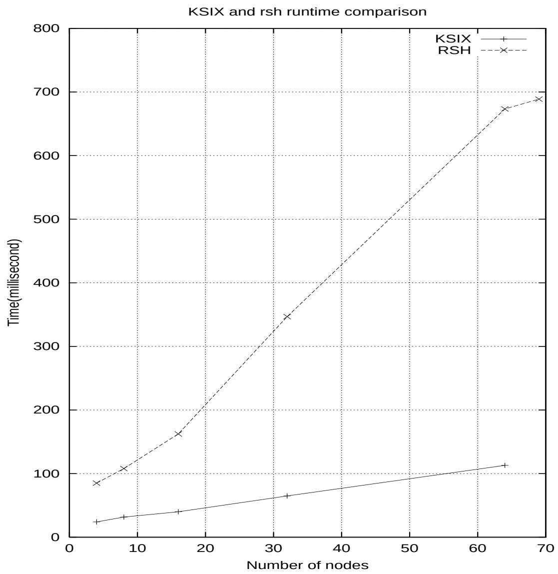

Figure 7.2: Comparison of KSIX and rsh based remote process creation time

7.5 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.

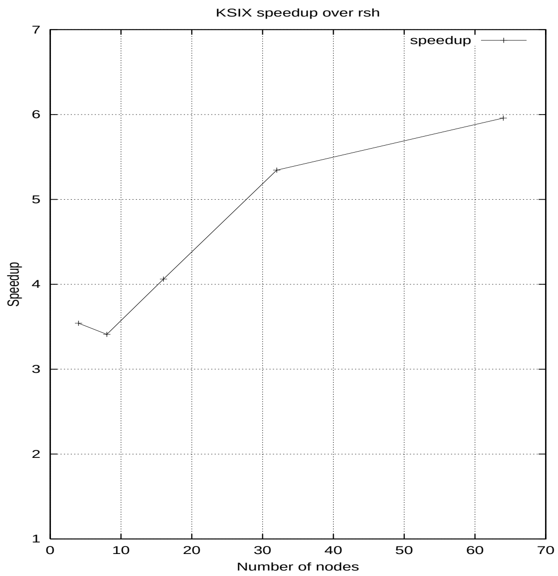

Figure 7.3: Speedup of KSIX over rsh based remote process creation time

Index