

SpringOS Tutorial for Scenario Execution

Toshiyuki Miyachi

–version 0.2–

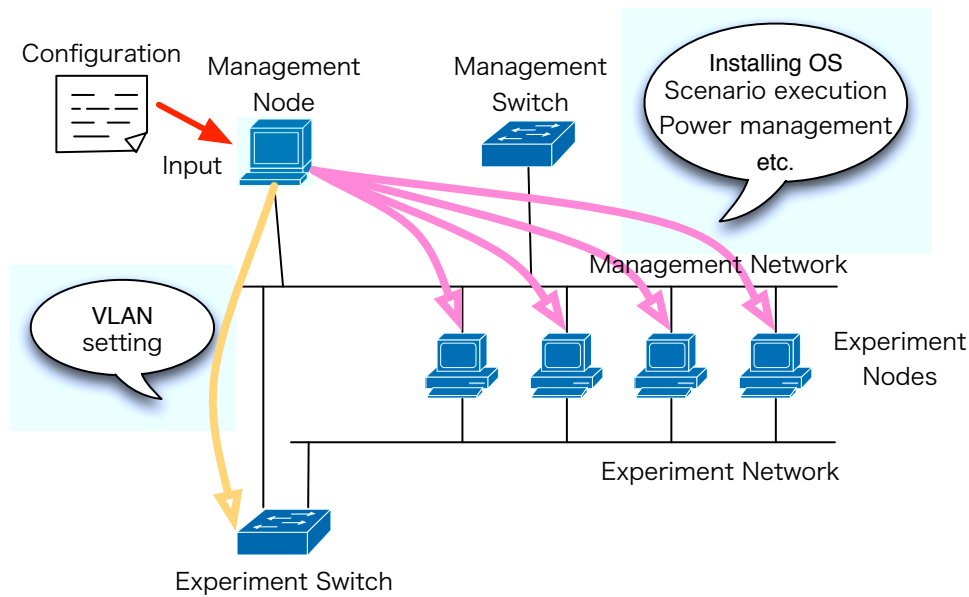


Figure 1: SpringOS behavior concept

1 Introduction

StarBED is a facility that uses a large number of PCs to verify the implementation for real environment of a large-scale environment. From the resources available at StarBED, the experiment executor will be assigned resources that have the required performance and features. Then, the executor will build his own experiment environment for the intended experiment.

In StarBED, it is not unusual to build an experiment environment that utilizes more than a dozen actual nodes. However, the larger the scale of the environment, the higher the building cost.

For the purposes of reducing experiment executor costs, SpringOS is provided as a set of software to assist in automatic experiment execution.

The flow of an experiment using StarBED and SpringOS is as follows:

1. Reserve StarBED nodes
2. Prepare management node
3. Set up experiment nodes
4. Prepare experiment scenario file
5. Execute experiment

As shown in Figure 1, SpringOS configures various settings and controls the experiments via the management network of StarBED.

This manual explains the user's task in each step mentioned above.

However, the details of the StarBED facility and how to reserve nodes are outside the scope of this manual.

2 Setting up the Management Node

SpringOS is not the name of a single program, but a collective name for the modules that set up the various types of nodes, and the modules that manage experiments by issuing requests to various modules.

The module which acts as the management actor for the experiment is implemented as kuroyuri master (hereinafter called “master”).

master runs in the management node of the experiment. In the current situation, an experiment executor has to prepare this node for every experiment.

A set of management modules that change the configuration of the devices in the StarBED facility (for example, switch nodes) is provided as part of the StarBED facility.

In addition, scenario execution on experiment nodes and control of the power supply using SNMPd require kuroyuri slave (hereinafter called “slave”) and snmpmine, which provides respective features. They must be installed into the experiment nodes.

First, this section explains how to set up management nodes.

The above-mentioned master is necessary for experiment management in SpringOS. In addition, you can use sbpsh and bswc.pl as modules that assist your experiment.

The master basically executes the setting up of the experiment environment, including automatic assignment of nodes, to the execution of a scenario, collectively in a group.

Meanwhile, sbpsh is a module that accomplishes changes in power supply methods and booting methods, and bswc.pl is a module that reflects the settings described as static data to the switch.

These modules are provided on the website of StarBED. To use the modules, you have to download and compile them. They run on various OSs, including Linux and FreeBSD.

In this document, we will use SpringOS install directory /usr/local/springos and save an executable file in /usr/local/springos/bin, and place source codes in /usr/local/springos/src.

Download spring-os-1.4.1.tgz from the download page on the website <http://www.starbed.org/> and unpack under the /usr/local/springos/src directory.

```
# tar xzf spring-os-1.4.1 -C /usr/local/springos/src
# cd /usr/local/springos/src/spring-os-1.4.1
# ls
ChangeLog      dman           ifsetup       sheepdog      wolagent
README         erm           kuroyuri      swmg
cfunc          ev            mine          tv
coil           fncp         ni            walk
```

Among these unpacked files, compile the files in the kuroyuri directory, which includes master and sbpsh.

```
# cd kuroyuri
# ./configure
# make
```

From the `/usr/local/springos/bin` directory, create symbolic links to each of the binary files of the master, `sbpsh`, `pickup`, and `wipeout`. `pickup` and `wipeout` are the binary files that will be used to install the OS. This will be explained in a later section.

```
# cd /usr/local/springos/bin
# ln -s ../src/swmg/src/bswc.pl
```

In addition, from the `/usr/local/springos/bin` directory create a symbolic link to `bswc.pl` located under `swmg/src`.

Create a configuration file of `sbpsh` in `/usr/local/springos/etc` directory. Figure 2 shows an example of this configuration file.

```
set rm 10.211.55.5 1234
set tftpdman 10.211.55.5 1236
set wolagent 10.211.55.5 5959 10.211.55.0/24

set user starbed
set project starproj
```

Figure 2: Sample of `sbpsh.rc` file

The main entries in this file are the IP address and port numbers on which the various modules are running. Basically, you can obtain these settings by asking the StarBED staff for them.

However, you have to describe the assigned user name and project name in the last two lines of the file. Save this file in the `/usr/local/springos/etc/` directory.

All of the above are in relation to configuring the management node. In the following section, we will configure an experiment node.

3 Setting up Experiment Nodes

To accomplish OS switching and the addition of applications on the experiment nodes, as mentioned above, we set up one machine as a template, pick up the contents of the disk as an image, and then distribute them to the other nodes.

You can create the template by installing the OS into one of the experiment nodes, or by modifying Fedora, FreeBSD, or Windows, which are already installed as default.

To reinstall the OS, you have to install bootloader in the partition where the OS has been installed, without changing the configuration of the partition table which is set up as the standard.

However, slave is not supported in Windows.

3.1 Installing SpringOS

We will use one node in the StarBED as a template node. Install the SpringOS after you have re-installed the OS and have installed any additional software.

The source code of Spring OS can be downloaded using the same procedure as used in the management node.

The modules that will be required for the experiment nodes are: `slave`, which is included in the `kuroyuri` directory; `snmpmine`, which is included in `mine` directory; and `ifscan`, which is included in the `ifsetup` directory. `ifscan` is a supporting program used to recognize the network interfaces for which IP addresses need to be set up. The network interfaces of the nodes in StarBED have been connected to the switch in advance, and the experiment network will be built by setting up a VLAN in the switch.

In a UNIX-like OS, you need to specify a device name for setting up configurations including the IP address. However, this device name varies depending on the OS; moreover, it may change every time you reboot the OS.

In SpringOS, these changes are handled by mapping the MAC address of each interface to the MAC address that has been registered in the database, and then configuring the interface based on this mapping result. The module which does this mapping is `ifscan`.

The compilation of these modules is executed by using `configure` and `make`, as you did with the management node. Create the symbolic links from the `/usr/local/springos/bin/` directory to `slave`, `snmpmine`, and `ifscan` that have been compiled by executing `make`.

`slave` and `snmpmine` are required to a scenario execution and the power management, and when you use these functions `slave` and `snmpmine` must be running on each experiment node. Therefore, set up the nodes so that these modules will be started when the experiment nodes boot up. The procedure to start up these processes during the node boot-up sequence varies depending on the OS. It needs to be set up using the applicable method for each OS. The following is an example of the `rc.local` file which is used in FreeBSD or other OSs:

```
#!/bin/sh

/usr/local/springos/bin/slave -t -d > /var/log/springos/slave.log 2>&1 &
/usr/local/springos/bin/snmpmine &
```

3.2 Configuring the network

StarBED makes the initial configuration of the experiment nodes possible by providing the management network. SpringOS has been designed under the assumption that this function exists. The set up of the address on the management network side will be configured using DHCP. For FreeBSD, write the contents of `/etc/rc.conf` as follows: `wm0` shown here is a device name.

```
ifconfig_wm0="DHCP"
```

In addition, install the applications that you need for your experiment.

3.3 Creating a Disk Image

The set up of the nodes, other than the template node, is accomplished by copying the disk image according to the above instruction. Use the `pickup` module to save, as a template, the contents of the hard disk you installed a moment ago. The `pickup` module saves, into the file server and as a binary image, the contents of the specified node and partition number. Since you need to specify many options when using the `pickup` module, write the static data you want to specify in a separate file. Name the file `pickup.opt`. The following is an example of the contents of the file. Like the other configuration files, save this file in `/usr/local/springos/etc/`.

```

-u starbed
-p starpass
-j starproj
-r localhost:1234
-k 10.211.55.5:1236
-f 10.211.55.5:1238
-s 10.211.55.5
-K FreeBSD/ni05.fs:recover_system/kernel-groupf
-P recover_system/pxeboot-nohang
-w 10.211.55.5:5959:10.211.55.0/24

```

Before starting an experiment in StarBED, it is desirable to obtain files with the appropriate contents from the StarBED staff. However, you have to write your own entries for options -u, -j, -p, and -s. Using -u, -j and -p, you will specify the username, password, and project name respectively, which were assigned when you reserved the test site. For -s, specify the IP address of the network interface connected to the management network of the management node on which you execute pickup. In this instruction, let us save the pickup.opt file in the /usr/local/springos/etc/ directory.

You can execute pickup as follows:

```

$ pickup -F /usr/local/springos/etc/pickup.opt \
-X ftp://install:install@10.211.55.5 sintcla209:1

```

Specify the location of pickup.opt you just created using -F. -X is used to specify the destination directory of the disk image. In this example, the FTP server is specified with a login name and password. sintcla209:1 represents the node name and partition number of SpringOS, separated by a “.” character. As a node name, specify the name that is managed by SpringOS in its own the database. By issuing the above command, the disk image you specified using the -X option will be saved in the FTP server. A file will be created with the following name:

```
Node name-device name-date+time.gz
```

In the above example, a disk image with the following file name will be saved with the node name sintcla209, device name rad0s1, and execution date and time 15:50, May 22, 2008 (200805221550).

```
sintcla209-rad0s1-200805221550.gz
```

Now, the image you need is ready. If you need to use multiple images, create multiple disk images following the above procedure. In this case, it is preferable to change the name of the file stored in the FTP server to make it easy to understand.

The wipeout module which has been installed together with pickup is convenient when you want to just install the disk image without executing the whole experiment using the master module. In other words, by using wipeout, you can verify that the experiment node will boot up properly using the disk image that you just saved. In this case, automatic assignment of the nodes or any network-related configuration will not be carried out. wipeout can be executed by using the same pickup.opt as is:

```

$ /usr/local/springos/bin/wipeout -F /usr/local/springos/etc/pickup.opt\
-X ftp://starbed:starpass@10.211.55.5/sintcla209-rad0s1-200805221550.gz sintcla210:1

```

When executing wipeout, use -X to specify the disk image location where you have saved the image when you used pickup, and the target node and partition number.

```
rmanager ipaddr "10.211.55.5" port "1234"
wolagent ipaddr "10.211.55.5" port "5959" ipaddrrange "10.211.55.0/24"
fncp ipaddr "10.211.55.5"
tftpdman ipaddr "10.211.55.5"
nidiskimage "FreeBSD/ni04b.fs"
nikernel "recover_system/kernel_recover"
pxeloder "recover_system/pxeboot"
setuptimeout total 86400 warm 5
```

Figure 3: Example of pre.sc

4 Experiment Scenario

The experiment scenario file includes the following information:

- Disk image used for each node
- Description about nodes, such as partition ID.
- Topology settings accomplished by VLAN configuration of the switch node to which network interface is connected
- Scenario that will be executed on each node

This file will be input into master, and then the experiment will be driven by the master issuing requests to each module in the facility according to the contents of the file.

In addition, you have to specify the IP addresses and port numbers of the nodes where modules provided by the facility will be running. Since most of them are specified as fixed values, we can create a separate file as you do with other configuration files. Of course, there is no problem even if you wrote them as a part of the scenario file.

In this section, we named the file pre.sc. Please see the example shown in Figure 3. For accuracy and convenience, as in the case of the configuration files, it is recommended that you request this file from the staff of StarBED.

4.1 Dealing with Variables

You do not need to define variables beforehand. You can define them by specifying character strings (except reserved names) that you want to use as the variable name on the left side, and then specifying the number or expression to substitute on the right side. It is not necessary to specify the variable type upon definition; however, when assigning a numeric value saved in a numeric value variable to a character string variable, you must convert it to a value type such as toString (<variable name>).

Variables are basically used as local variables; however, you can define global variables by using export. For example, you can express the directory where the SpringOS-related binary files reside, by writing the codes written in Figure 4 before the node definition block.

Use '+' for string concatenation. If you write the scenario as follows, the "hoge" file under the bindir directory that you have specified will be executed.

```
1: bindir="/usr/local/springos/bin/"
2: pathifscan=bindir+"ifscan"
3: export bindir
4: export pathifscan
```

Figure 4: Dealing with Variables

```
callw bindir + "hoge"
```

In addition, you can specify the variables that you want to change for every experiment from arguments in the master module. For a global variable specified with arguments, define it using assure as follows: In this example, 1 is specified as a default value for use whenever the value is not specified.

```
assure <variable name>=1
export <variable name>
```

You can change this value by giving an argument such as '`<variable name>=2`' in the master's argument.

4.2 Global settings

For the global settings, write the lines shown below on the top of configuration file. However, the leading number and ':' are added just for explanation. They should not be typed in the actual file.

```
1: user "starbed" "no-one@starbed.org"
2: project "starproj"
3: encd ipaddr "10.211.55.10"
4: sparenodemain 0
5: sparenoderatio 100
```

The first line is the setting of the user name which was assigned at the time of facility reservation and the setting of e-mail addresses that are used as contacts. Currently, e-mail addresses are not used when executing the experiment. The second line specifies the project name that was assigned during facility reservation; write in the same way you did in the first line. The third line specifies the IP address of the node which executes the master.

The fourth and fifth lines specify the number of nodes that are used as spare nodes. In SpringOS, you can define spare nodes that will be used in case of node failure. In the fourth line, specify the minimum number of nodes numerically, and in the fifth line, specify it by percentage (%). 100 represents 100%, which means there are no spare nodes. 110 represents 110%, which means that it prepares as many as 10% spare nodes.

The actual number of spare nodes will be determined as the larger of these two. A spare node will be provided for each node set, which will be explained in a later section.

4.3 Settings for Experiment Nodes

The following will be configured for the experiment nodes.

4.3.1 method

This specifies the booting method.

HDD Install the new OS into the hard disk and boot the node using the OS.

thru Use the OS (which the slave is running on) which has been installed in the hard disk.

4.3.2 partition

Specify the partition number when HDD or thru is specified in method.

4.3.3 ostype

Describe the OS name, such as FreeBSD and Linux. Originally, used for resolving the differences in the device names which are recognized in each OS. However, currently they are not used.

4.3.4 diskimage

Location of the disk image to use.

4.3.5 netif

Specifies network interface value as follows:

```
netif media <media type>
```

You can use FastEthernet and GigabitEthernet as media types. If you need multiple network interfaces, write as many lines as you need. In this case, write each of the defined network interfaces using the format `netif[<number>]`, according to the order of writing. The first interface would be represented as `netif[0]`.

4.3.6 scenario

Written as a block enclosed in braces '{' and '}'. The contents enclosed in the brackets will be executed in each node. The scenario defined here will be called node scenario.

4.3.7 nodeclass definition

In SpringOS, a node is defined using classes, as in object oriented programming. It is used by defining the instance of the class as a node set. Use `nodeclass` for defining classes and specify them as follows:

```
nodeclass <class name> {  
    < Node settings>  
}
```

4.4 Description of Node Scenarios

In this section, we explain how to describe node scenarios. Basically, the node scenario is executed in the order that it is described.

4.4.1 Executing application

To execute various applications that have been installed on the OS, you can use `callw` or `call`.

callw

`callw` "`<Path to execution file>`" "`<Option>`" ...

This command executes the specified application. Scenario execution will be stopped until execution of this application has stopped (foreground execution).

call

`call` "`<Path to execution file>`" "`<Option>`" ...

This command executes the specified application. Execute the next scenarios without waiting for the completion of the application (background execution).

`callw` and `call` also support re-direct. You can specify re-direct to the file on each node by using "`>`" and "`>>`" as you do in shell.

4.4.2 Repeat control

For repeating, you can use `loop`, `for`, and `while`.

loop

Repeats the contents enclosed between '`{`' and '`}`'.

for

```
for (Contents executed at start; continuation condition; contents repeated on every repetition) {  
    <command list>  
}
```

Repetition control using the same format as C languages.

while

```
while (Repetition condition) {  
    <command list>  
}
```

Repetition control using the same format as C languages.

4.4.3 Message Exchange

SpringOS accomplishes node synchronization and exchange of required information by using message exchange.

send

send "<Character string>"

Sends to master the character string which is specified. In SpringOS, basically all messages are exchanged via master.

recv

recv <variable>

Saves received message contents into variables.

msgswitch

```
msgswitch <variable> {  
  "message 1" {  
    <command list>  
  }  
  "message 2" {  
    <command list>  
  }  
  ...  
}
```

Depending on the message content that is saved in the variable, the control after that will be changed.

4.4.4 Setting up Network Interfaces

When setting up a network interface, first, you will map the MAC address with the interface information that is saved in the database as mentioned above. After you have finished this mapping using netifit, call ifconfig as a part of the scenario and set up the IP address.

In the following example, in `self.netif[0]`, the device name of the zeroth network interface will be assigned. In `self.netif[0].ipaddr`, the IP address specified for the zeroth network interface address will be assigned.

```
netifit <path of ifscan>  
callw "/sbin/ifconfig" self.netif[0].rname self.netif[0].ipaddr
```

4.4.5 Others

Here are the additional commands that can be used in node scenarios.

sleep

sleep <seconds>

Stops the scenario execution for the specified number of seconds.

chdir

```
chdir "<directory>"
```

Moves to the specified directory.

netput/netget

```
netput "<targetfile>" "<ftpserver>"
```

Uploads targetfile to the specified server.

```
netget "ftp://<ftpserver>/<filename>"
```

Downloads the specified file.

Also, this supports four arithmetic operations.

4.5 Creating a Node Set

Next, we will define node sets. Node set will create the nodes defined in the above class definition.

```
nodeset <node set name> class <class name> num <number of nodes>
```

Specify any name you want in the <node set name>. In <class name>, specify the class name you already specified, and in the <number of nodes>, specify the number of nodes that you need. Use the node set name [number] to access the node that you created. In this case, the first node number would be 0.

4.6 Defining Networks

Next, we will define networks. Like nodes, a network is also defined using classes. It is used by defining the instance of the class as a network set.

The network class is defined using the following format:

```
netclass <class name> {  
...  
}
```

The items which you can specify in netclass are media and ipaddrange. In media, you can specify FastEthernet and GigabitEthernet, and in ipaddrange you can specify an address range to be used.

Network set will be created using the following format:

```
netset <network set name> class <class name> num <number of networks>
```

To connect the interface of each node to the network, use attach as follows:

```
attach <node set name>[number].netif[number] network set name[number]
```

By connecting the network interface of each node to the network set created by the above command, you can connect to the L2 network.

4.7 Description of Global Scenarios

A global scenario is described mainly for the purpose of accomplishing the coordination between nodes. Its main role is to send messages to nodes.

send

send <node name> "<message>"

In the node scenario, you specified only character string texts in the message, since messages are basically transmitted to the master. On the other hand, in a global scenario, you specify where you send messages.

sync

Stops the scenario execution until all statements in the block enclosed in braces become true.

multimsgmatch

multimsgmatch <node set name> "<message>"

Stops the scenario execution until receipt of the specified message from all nodes included in the specified node set.

multisend

multisend <node set name> "<message>"

Sends the specified message to all nodes included in the specified node set.

Write the global scenario according to the contents of the messages that are sent and received in the node scenario.

5 Example of an Experiment Scenario

In this section, we will explain the configuration example that is written using the grammar that we explained in the above section.

5.1 Executing command

The scenario shown in Figure 5 is a simple scenario that executes only the touch command.

The first to fifth lines define the global settings. These lines specify the user and project assigned at the time of reservation, and then, using `encl`, they specify the IP address of the management node where the master is running on. (However, if you are always running the master on the same node, you can script it in the `pre.sc`.) Also, `sparenodemim` is 0 and `sparenoderatio` is 100. This means that no spare nodes are prepared.

The sixth to the fourteenth lines define the node class. This class is defined so that you can access it using the name `class`. In this range, it is defined to install a FreeBSD image into partition 1 of the hard disk.

The node scenario from the 11th to the 13th lines creates the file by executing `touch`.

The 15th line prepares one node set belonging to the node class `class` that you defined in the above line.

```
1: user "starbed" "no-one@starbed.org"
2: project "starproj"

3: encl ipaddr "10.211.55.5"

4: sparnodemim 0
5: sparnoderatio 100

6: nodeclass clclass {
7:     method "HDD"
8:     partition 1
9:     ostype "FreeBSD"
10:    diskimage "ftp://starbed:starpas@10.211.55.5:sintcla209-rad0s1-200805221550.gz"
11:    scenario {
12:        callw "/usr/bin/touch" "/tmp/huga"
13:    }
14: }

15: nodeset client class clclass num 1

16: scenario {
17:     sleep 10
18: }
```

Figure 5: touch.sc

The 16th to 18th lines are global scenarios. The end of the global scenario means the end of the experiment itself. The global scenario must be effective until the node scenario is completed. This code section specifies to simply execute a 10-second sleep until the node scenario is completed.

5.2 Message Exchange

```
1: nodeclass clclass {
2:     <Omitted>
3:     scenario {
4:         send "setupdone"
5:         recv val
6:         callw "/bin/echo" val > "/tmp/huga"
7:         send "done"
8:     }
9: }
```

Figure 6: Message Exchange Node Scenario

```
1: scenario {
2:     sync {
3:         msgmatch client[0] "setupdone"
4:     }
5:     send client[0] "Hello"
6:     sync {
7:         msgmatch client[0] "done"
8:     }
9: }
```

Figure 7: Message Exchange Global Scenario

You can control the execution of each scenario by using message exchange. To do that, change the scenario in which the above-mentioned touch is executed so that it includes the message exchange portion. The master is notified upon completion of touch at the node, and then the global scenario will exit.

Replace the node scenario section in Figure 5 with that of Figure 6, and the global scenario section in Figure 5 with that of Figure 7.

The <Omitted> part in Figure 5 is the same as the scenario which executed touch in the above example.

The timing at which the node scenario starts is (almost) at the same moment as when the node configuration is completed. At the time of the scenario execution, a “setupdone” message is sent to the master, and then execution waits until a message is received from the master. When the message is received, it will be saved in the variable “val.” When it receives any message, it executes an echo and

then outputs the contents of the message into/tmp/huga. Message contents are not distinguished in this action. After the processing it sends the message “done” to the master.

In the global scenario, execution stands by until a message “setupdone” is received from client[0]. The name client is assigned by the experiment executor when the node set is defined. The first node will be specified as [0]. Execution waits for the message “setupdone” from client[0] by using sync and msgmatch.

5.3 Configuration related to networks

Set up the IP address of each network as follows: ethnet is used as a network set.

```
ethnet[0].ipaddrange = "192.168.0.0/24"
```

When you define multiple networks, it would be convenient if you set them up using a for statement, as follows:

```
for(i=0;i<pair;i++){
    ethnet[i].ipaddrange = "192.168."+toString(i)+".0/24"
}
```

These will be written outside of the blocks of the node definition, network definition, and global scenario definition.

By the same token, connect the interface of each node to the network.

```
attach client[0].netif[0] ethnet[0]
```

You can set this up by using the for statement as well.

```
for(i=0;i<pair;i++) {
    attach client[i].netif[0] ethnet[i]
}
```

In the node scenario, set up the IP address by using a statement like the following:

```
netiffit pathifscan
callw "/sbin/ifconfig" self.netif[0].rname self.netif[0].ipaddr
```

After you have matched the device name and MAC address by using netiffit, you can gain access to each device with self.netif[<variable>].rname, and to the IP with self.netif[<variable>].ipaddr. In the above statement, an IP address of the zeroth network interface is set up.

6 Executing the Experiment

You can execute an experiment by entering the setting definition as an input into the master.

```
$ master -p starpass -s <Password of the switch> pre.sc <Scenario file>
```

-p is the password that is assigned to each user. -s is the password for the switch. For the scenario files, enter the pre.sc and the scenario file together on the same command line. If you have already entered the contents of pre.sc in the scenario file, specify only the scenario file. (You can specify three or more scenario files. If you specified more than one scenario, they are processed as a single file combined in the order they are specified.)

Appendix 1

This chapter shows examples of all setting definitions which have been used here.

`/home/starbed/conf/sc/pre.sc`

```
rmanager ipaddr "127.0.0.1" port "1234"
wolagent ipaddr "172.16.1.101" port "5959" ipaddrrange "172.16.0.0/24"
wolagent ipaddr "172.16.1.101" port "5959" ipaddrrange "172.16.1.0/24"
wolagent ipaddr "172.16.4.253" port "5904" ipaddrrange "172.16.4.0/24"
fncp ipaddr "172.16.3.101"
tftpdman ipaddr "172.16.3.101"

nidiskimage "FreeBSD/ni04b.fs"
nikernel "recover_system/kernel_recover"
pxeloder "recover_system/pxeboot-nohang"
setuptimeout total 86400 warm 5
```

```
/home/starbed/conf/sc/touch.sc
```

```
user "starbed" "no-one@starbed.org"  
project "starproj"
```

```
encd ipaddr "10.211.55.10"
```

```
sparenodemain 0  
sparenoderatio 100
```

```
nodeclass clclass {  
    method "HDD"  
    partition 1  
    ostype "FreeBSD"  
    diskimage "ftp://install:install@10.211.55.100:/diskimages/hoge.gz"  
    scenario {  
        callw "/usr/bin/touch" "/tmp/huga"  
    }  
}
```

```
nodeset client class clclass num 1
```

```
scenario {  
    sleep 10  
}
```

```
/home/starbed/conf/sc/net.sc
```

```
user "starbed" "no-one@starbed.org"  
project "starproj"
```

```
encl ipaddr "10.211.55.10"  
ipaddrange "192.168.100.0/24"
```

```
sparenodemain 0  
sparenoderatio 100
```

```
bindir="/home/starbed/bin/"  
pathifscan=bindir+"ifscan"  
export bindir  
export pathifscan
```

```
nodeclass clclass {  
    method "HDD"  
    partition 1  
    ostype "FreeBSD"  
    diskimage "ftp://install:install@10.211.55.100:/diskimages/hoge.gz"  
    netif media fastethernet  
    scenario {  
        netifit pathifscan  
        callw "/sbin/ifconfig" self.netif[0].rname self.netif[0].ipaddr  
        send "cdone"  
    }  
}
```

```
nodeclass svclass {  
    method "HDD"  
    partition 1  
    ostype "FreeBSD"  
    diskimage "ftp://install:install@10.211.55.100:/diskimages/hoge.gz"  
    netif media fastethernet  
    scenario {  
        netifit pathifscan  
        callw "/sbin/ifconfig" self.netif[0].rname self.netif[0].ipaddr  
        send "sdone"  
    }  
}
```

```
netclass ethclass {  
    media fastethernet  
}
```

```
nodeset client class clclass num 1
nodeset server class svclass num 1
netset ethnet class ethclass num 1

attach client[0].netif[0] ethnet[0]
attach server[0].netif[0] ethnet[0]

scenario {
    sync {
        msgmatch client[0] "cdone"
        msgmatch server[0] "sdone"
    }
}
```

/home/starbed/conf/sc/netperf.sc

```
user "starbed" "no-one@starbed.org"
project "starproj"

encl ipaddr "10.211.55.10"
#ipaddrange "192.168.100.0/24"

sparenodemini 0
sparenoderatio 100

bindir="/home/starbed/bin/"
workdir="/home/starbed/work/"
pathifscan=bindir+"ifscan"
pathnetperf="/usr/local/bin/netperf"
pathnetserv="/usr/local/bin/netserver"
epoch=time()
export bindir
export workdir
export pathifscan
export pathnetperf
export pathnetserv
export epoch

assure pair=1
export pair

nodeclass clclass {
    method "HDD"
    partition 1
    ostype "FreeBSD"
    diskimage "ftp://starbed:starbed@10.211.55.100:/diskimages/hoge.gz"
    netif media fastethernet
    scenario {
        logfile=self.rname+"-"+tostring(epoch)+".log"
        netifit pathifscan
        callw "/sbin/ifconfig" self.netif[0].rname self.netif[0].ipaddr
        send "setupdone"
        chdir workdir

        recv dst
        callw pathnetperf "-H" dst > logfile
        sleep 180

        netput logfile "ftp://starbed:starbed@10.211.55.10/log/clients/"
    }
}
```

```

        send "perfdone"
    }
}

nodeclass svclass {
    method "HDD"
    partition 1
    ostype "FreeBSD"
    diskimage "ftp://starbed:starbed@10.211.55.100:/diskimages/hoge.gz"
    netif media fastethernet
    scenario {
        netifit pathifscan
        callw "/sbin/ifconfig" self.netif[0].rname self.netif[0].ipaddr
        send "setupdone"
        loop {
            recv val
            msgswitch val {
                "start" {
                    call pathnetserv
                    send "started"
                }
                "stop" {
                    callw "/usr/bin/pkill" "-9" "netserver"
                    send "stopped"
                }
            }
        }
    }
}

netclass ethclass {
    media fastethernet
}

nodeset client class clclass num pair
nodeset server class svclass num pair
netset ethnet class ethclass num pair

for(i=0;i<pair;i++){
    ethnet[i].ipaddr = "192.168."+toString(i)+".0/24"
}

for(i=0;i<pair;i++) {
    attach client[i].netif[0] ethnet[i]
    attach server[i].netif[0] ethnet[i]
}

```

```
scenario {
    sync {
        multimgmatch server "setupdone"
        multimgmatch client "setupdone"
    }

    multiset server "start"

    sync {
        multimgmatch server "started"
    }

    for(i=0;i<pair;i++) {
        send client[i] haddr(server[i].netif[0].ipaddr)
    }

    sync {
        multimgmatch client "perfdone"
    }

    multiset server "stop"

    sync {
        multimgmatch server "stopped"
    }
}
```

Appendix 2

This chapter provides a glossary of the terms that we refer to in this manual and any terms related to SpringOS.

Terms related to SpringOS	
bswc.pl	Script to configure static settings of the switch
ifscan	Module used for mapping between MAC address and device names
kuroyuri master (master)	Main module to control execution of the experiment
kuroyuri slave (slave)	Module that runs on the experiment node and actually executes the scenario
pickup	Module used to extract disk images
sbpsh	Shell interface used for power management and changing the booting method of nodes
snmpmine	The module that runs on in experiment nodes and carries out reboot / power interruption based on request
swmg	Module that runs on the facility side and configures the switch
wipeout	Module that delivers to other experiment nodes the disk image which is created by pickup

Terms that are referred to in this manual	
IPMI	Node management standard advocated by Intel
PXE	Standard used to boot nodes via network
SNMP	Protocol used to obtain node information and manage nodes
VLAN	Standard used to separate a switch virtually, and to create L2 topology
WoL	Standard which is advocated by AMD used to power on nodes

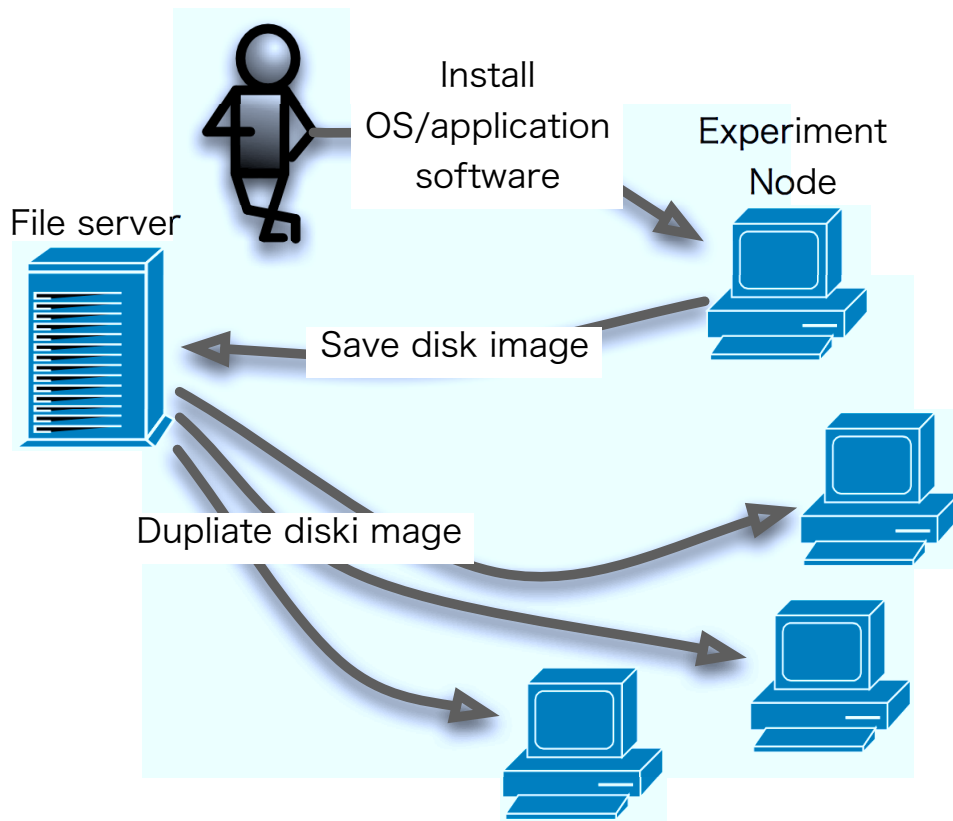


Figure 8: installation concept using SpringOS

Appendix 3

This section explains the implementation part of the main functionalities of SpringOS shown in the following list:

- Power management of nodes
- Change boot method of nodes
- Install OS to nodes
- Configure switches (VLAN)
- Scenario execution by processing commands at nodes
- Managing experiment resources

Nodes of StarBED are connected to the management network and are set up with static settings so that it can always be controlled even when the experiment is proceeding. Basically, SpringOS executes all kinds of operations through this management network.

Power management of nodes In the experiments node, a wide range of software troubles are expected. To respond to this, it is desirable that we can use a mechanism that supports power interruption and rebooting available at the hardware level. Among experiment nodes provided in StarBED, group F nodes support IPMI, which allows node control at the hardware level. Control is accomplished by power-on, shut down, and rebooting.

On the other hand, because the timing of the introduction date is old, the group A-E does not support IPMI. Power-on on these nodes is accomplished by using WoL which has been implemented as one capability of the Magick packet technology by AMD.

However, although WoL supports power on, it does not support shutdown and rebooting.

Therefore, an exclusive SNMP agent is provided in SpringOS. Power management is accomplished by running this SNMP agent on the nodes and using SNMP. However, you may be unable to use SNMP when a trouble occurs on the node, because the SNMP is implemented at the software level.

Of course, you can use SNMP in the group F.

Change boot method of nodes Basically, four partitions are provided in StarBED nodes, and Windows/FreeBSD/Linux is installed in each partition in a normal condition. You can use the one remaining partition as a spare area which is formatted in FAT. Experiment executors can just use these OSs as is, or overwrite them with another OS. Generally, in a multi-boot environment, you would specify the partition which should be used when booting the node. However, such switching methods are not efficient in an environment that has a lot of nodes, such as StarBED.

StarBED nodes always use network boot by using PXE and obtain a bootloader. So, you can switch the partition to use by changing the bootloader that will be transmitted to each node. Nodes can be booted as a diskless system.

Install OS to nodes The cost for newly installing an OS into a large number of nodes is not small. In SpringOS, focusing on the fact that one OS setup can be shared with multiple nodes in most experiment environments, a mechanism to reproduce nodes that have the same settings as the original is employed. This is accomplished by copying the entirety or a part of the hard disk contents installed in one node to the hard disk of the other nodes. In this case, IP addresses and other applicable settings are configured by using a separate method. In addition, you can use it as a diskless system without using hard disks which are connected to each node. Figure 8 shows the concept of OS installation.

Configure switches (VLAN) Modification of the L2 topology is necessary to build the experiment environment; however, modification by manual operation is not realistic in a large-scale environment. The automatic change of experiment topologies could be accomplished by changing the configuration in either of two methods: One uses a crossbar telephone switchboard which has a function to physically change the wiring, and the other uses switches that have VLAN capability. In StarBED, this function is accomplished using the latter method: Changing VLAN settings and setup of a virtual network. SpringOS follows the settings definition, configures the required switch, and builds the intended topology.

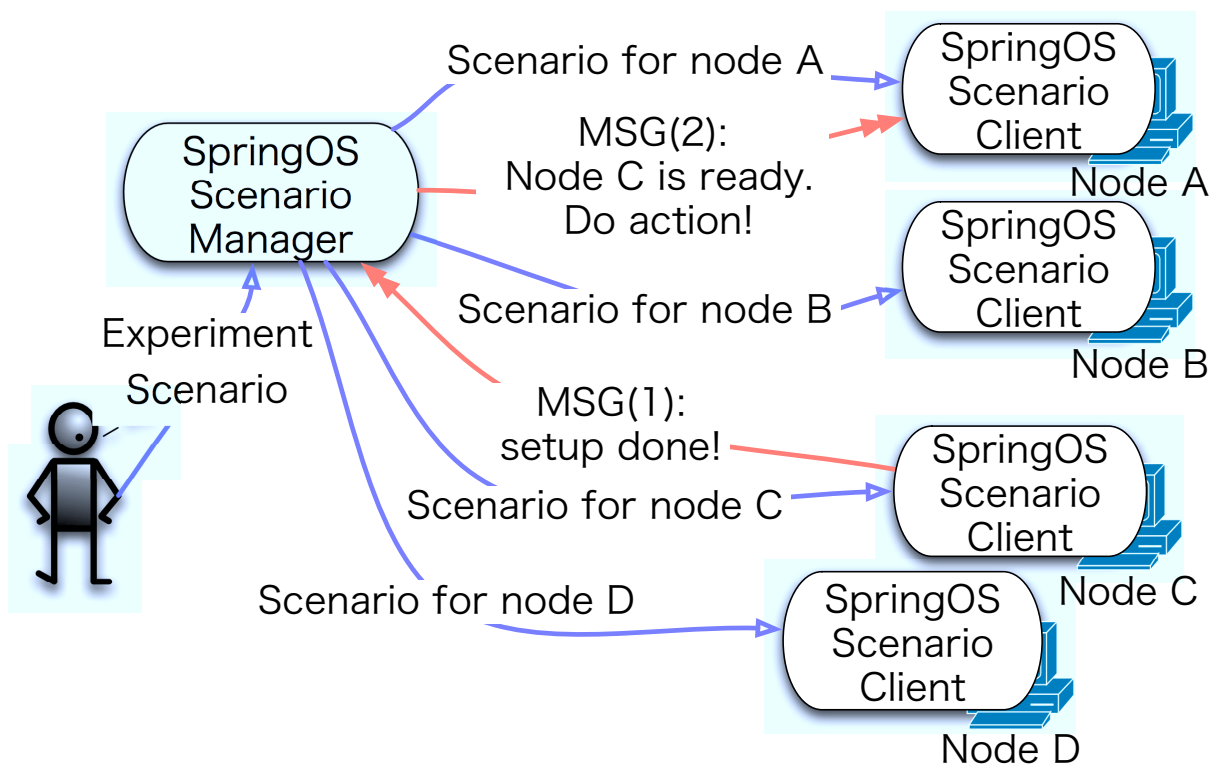


Figure 9: Scenario execution in SpringOS

Scenario execution In SpringOS, the script that contains the list of commands to be executed in each node and describes the message exchange to control the execute timing of these commands is called a “scenario.” Based on the description input by the experiment executor, the management node will transmit the scenario description that is required for each experiment node and let it process it.

The management node mass-transmits the command list which should be executed after completion of the environment building. In each experiment node, the client program that executes the commands is running. As the program receives the command list, it executes the commands sequentially.

When synchronization with other nodes is necessary, it is established by using message exchange through the management program which is running in the management node, and then exchanging information, stopping and resuming scenarios. Figure 9 shows the concept of scenario execution.

Managing experiment resources There are six kinds of nodes in StarBED as of April, 2008, and they vary in their specifications, such as the number or types of network interfaces and CPU. Before experiment environment building, you had to decide which node you will use on which part in the topology, but the cost involved with this task would not be small. SpringOS manages specification information of the individual node, and assigns the physical node required based on the experiment scenario to the logical node in the experiment scenario. It also provides mediation processing so that the node you assigned in this case will not be assigned to another experiment executor.

In the current SpringOS version, each node, VLAN ID, and each port of the switches are managed as a resource.