Automating Initial Setup and Management of Sun Fire™ V20z and V40z Servers

Jacques Bessoudo, Network Systems Group

Sun BluePrints™ OnLine
June 2005

Part No. 819-2798-10
Revision 3.2,
Edition: June 2005

Please
Recycle

Automating Initial Setup and Management of Sun Fire™ V20z and V40z Servers

Many compute- and network-centric applications can benefit from pools or grids of smaller, horizontally-scaled servers due to thier lower initial cost, flexibility, scalability, and performance for certain tasks. However, installing and managing tens or hundreds of servers in a consistent manner can be time consuming and prone to errors that further increase the time required to manage large pools of servers. Fortunately, many administrative tasks can be easily automated using the integrated service processor in the Sun Fire V20z and V40z servers..

There are three areas of setup and management of the Sun Fire V20z and V40z server's service processor that are usually performed manually — where automated scripts can save time and can eliminate errors:

- Service processor setup
- Service processor management
- Server (BIOS) and service processor firmware updates

This article describes a method for helping system administrators save time by automating these processes and running them on multiple systems simultaneously. It details the steps for creating scripts to automate these tasks and run them in parallel and includes examples of several of the more common tasks. The article contains the following sections:

- "Updating Servers and Service Processor Firmware" on page 16 provides a full script for automating these tasks.
- "Appendix A: Connecting a Laptop to the Service Processor" on page 26 provides detailed steps on connecting a laptop to the serial port via the Solaris™ Operating System, Linux, and Windows.
- "Appendix B: Connecting a Terminal Concentrator to the Serial Port" on page 29.

Typographic Conventions

Table 1-1 defines the typographic conventions used in this article.

TABLE 1 Typographic Conventions

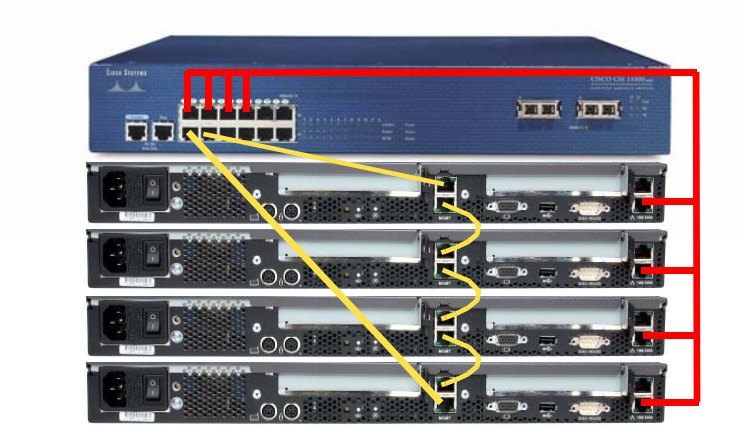
Typeface	Meaning	Examples
`AaBbCc123`	The names of commands, files, and directories — on-screen computer output	Edit your `.login` file. Use `ls -a` to list all files. `% You have mail.`
`AaBbCc123`	What is typed, when contrasted with on-screen computer output	% **su** Password:
AaBbCc123	Book titles, new words or terms, words to be emphasized	Read Chapter 6 in the *User's Guide*. There are called *class* options. You *must* be superuser to do this.
AaBbCc123	Command-line placeholder text — replace with a real name or value	To delete a file, type rm *filename*.

Communicating with Sun's Servers Based on AMD Opteron Processors Using the Service Processor

Once all of the servers are installed in the rack or racks, the next step is to install cables to each of the nodes. The minimum cables required are power and network cables, but some deployments might also require serial connections to the console ports. In most cases, network cables are needed for the 2-gigabit Ethernet platform interfaces and for the remote management interface.

In Sun Fire V20z and V40z servers, the service processors (SPs) can be daisy chained. By using cross-over cables from one service processor to the next, all of the SPs can be managed from one console. Daisy chaining also requires fewer switch ports to connect to all of the SPs. This is illustrated in Figure 1.

FIGURE 1 Daisy Chaining Service Processors



Caution – Performance degradation can occur if the cross-over cables are less than 1 meter. Redundant connections to the management network are strongly recommended since removal of power to a unit breaks the chain.

RJ-45 cross-over cables must be used to interconnect the servers. Cables can be connected to either the top or bottom SP port. To configure servers in a daisy chain, connect the first and last server in the chain to different switches.

Managed spanning-tree capable switches are required to redundantly connect both the top and bottom of the chain. If the switch is not capable of spanning-tree discovery, then only connect either to the top or the bottom of the chain, but not both.

The Service Processor

The Sun Fire V20z and V40z servers include a dedicated service processor (SP) for operating system independence and system management availability. The SP enables remote control of server operations such as boot, shutdown, and reboot of the server's operating system, halting the server's boot process in BIOS, and upgrading the BIOS. The SP is powered via auxiliary power so that it is available even when the platform is powered down.

The Sun Fire V20z and V40z servers support both *in-band* and *out-of-band* management. In-band refers to traffic that follows the same path as the normal data, using the gigabit Ethernet ports and the running operating system. Out-of-band refers to the traffic that travels on a

separate media than the data, using the dedicated 10/100 MB management ports and the SP. Access to the SP is available via its private LAN connection (default) or the serial port connector on the back of the system.

Many environments require that the SP is available on the serial port at all times. For example, a server farm may require that the operating system (OS) be re-initiated every 24 hours. If the SP is available via the serial port, it is convenient to reboot the system and re-load the OS. By default, the SP is available on the serial port. If the behavior is modified after the server is unpacked, then the serial port can present the platform console directly (BIOS upon boot, for example).

An option to force the SP to be available on the serial port is accomplished by installing a jumper on the motherboard's jumper labeled J19, which is located next to the PCI slot on the back of the server (note that for the Sun Fire V40z server the power supplies must be removed to access jumper J19).

After the jumper is installed, a null modem serial cable can be connected to the DB-9 port on the back of the system, enabling the administrator to manage the SP from a laptop or from a workstation directly connected to the system. For more information on how to connect a workstation or laptop to the serial port of the Sun Fire V20z and V40z servers, please refer to Appendix A.

If the SP is not available on the serial port, it might be because the default settings have been changed. If this is the case, it must be configured using the LAN-based interface.
To configure the SP over the LAN port, the interface must be connected to a DHCP enabled network, where the SP can acquire an IP address. Once the IP address is configured it is displayed on the front panel LCD. Another procedure to set up the IP address is to enter the IP address using the LCD panel. These procedures are detailed in the *Sun Fire V20z and V40z Servers Installation Guide.*

As an alternative, if a terminal concentrator (otherwise known as Network Access Server) is used, the administrator can access a number of SPs as soon as power is applied to the servers.

The settings for the terminal concentrator or terminal software are:

Baud rate: **115200**

Data bits: **8**

Parity: **None**

Stop bits: **1**

Flow control: **Off**

More information on connecting a terminal concentrator to the serial port is contained in Appendix B.

Once the settings are correct on either the software or the terminal concentrator, a connection to the SP can be established. Below is an example of a terminal connection to a Sun Fire V40z server from a Solaris server connected to the SP. The output below the `tip` command is the boot sequence that is generated when power is applied to the server.

```
bash-2.05# tip -115200 /dev/term/a
PPCBOOT Starting

CPU:   XPC860xxZPnnD4 at 64 MHz: 4 kB I-Cache 4 kB D-Cache FEC
present
Board: Service Processor Stage 1 - Rev. 0
DRAM:  64 MB
FLASH: 16 MB
In:    serial
Out:   serial
Err:   serial
Product_ID = 0xef (239)
PPCBOOT revision = V2.1.0.16
Board_Revision = 0x04 (4)
PRS revision = 0x0e (14)
Hit any key to stop autoboot:  0
## Booting image at 40080000 ...
   Image Name:   2.4 kernel for sp
   Image Type:   PowerPC Linux Kernel Image (gzip compressed)
   Data Size:    668721 Bytes = 653 kB = 0 MB
   Load Address: 00000000
   Entry Point:  00000000
   Verifying Checksum ... OK
   Uncompressing Kernel Image ... OK
## Loading RAMDisk Image at 40180000 ...
    .
    .

Freeing unused kernel memory: 60k init
INIT: version 2.78 booting
Mounting local filesystems...
Booted from ramdisk..
Service Processor GPIO driver, ver. V2.1.0.16
Service Processor I2C driver, ver. V2.1.0.16
Service Processor HAD/PRS driver, ver. V2.1.0.16
Configuring iptables
Hostname: localhost.localdomain
Setting up IP spoofing protection: rp_filter.
Disable TCP/IP Explicit Congestion Notification: done.
Configuring network interfaces: done.
Starting portmap daemon: portmap.
```

```
INIT: Entering runlevel: 2
Starting metalog for logging services
Starting internet superserver: inetd.
Starting system management applications...
Starting OpenBSD Secure Shell server: sshd.

localhost login:
```

After the SP is loaded, the administrator can proceed to login as the `setup` user in order to add the top level administrator to the SP as shown below. A setup account is included with each server. This setup account does not have a password. When logging into the SP for the first time using the setup account, the user is prompted to define the initial manager account with a password and an optional public key.

This is an important step, as failure to secure the SP with a user name and password when the server is first deployed can expose the server to security and availability risks through the SP network interface.

```
localhost login: setup
Enter the name of the first manager level user: admin
Password:
Confirm password:
Enter the user's public key or press ENTER for no key:

Successfully added the manager level user.

localhost login:
```

Note – An operating system can be installed on the server without configuring the SP or the Network Share Volume (NSV). However, if the SP and NSV setup is not performed, it is not possible to use the remote management capabilities or the diagnostics of the system.

Creating Expect Scripts

The setup task, as well as others, can be captured with little effort by using a tool called autoexpect, enabling setup to be automated for many servers.

Brief Introduction to Autoexpect

Autoexpect is a tool written by Don Libes from the National Institute of Standards and Technology and is available to users from http://expect.nist.gov/. Autoexpect is included with recent distributions of expect.

The instructions to use autoexpect are simple. The user starts autoexpect and runs a command or series of commands. Autoexpect generates a script called `script.exp` in the directory specified (current directory is default). The automatically generated expect script includes all of the commands issued to accomplish a task and keeps track of all of the responses from the server.

Once `script.exp` is generated, the user can edit it to modify the details that might be different when accessing multiple hosts. Key things to look for are unique text strings, such as dates and hostnames.

Creating an Automated Script

In order to create the script using autoexpect, the user must interact with the SP in a natural way, issuing the commands necessary to carry out a specific task. In the following example, a simple interaction with the SP over the LAN configures the initial user *admin*:

```
[admin@localhost]$ autoexpect
autoexpect started, file is script.exp
[admin@localhost]$ ssh -l setup 10.6.164.16

Sun Microsystems
IPMI v1.5 Service Processor

Version: V2.1.0.16
Enter the name of the first manager level user: admin
Password:
Confirm password:
Enter the user's public key or press ENTER for no key:

Successfully added the manager level user.
Connection to 10.6.164.16 closed by remote host.
Connection to 10.6.164.16 closed.
[admin@localhost]$ exit
autoexpect done, file is script.exp
[admin@localhost]$
```

The output of this exercise includes all of the comments and parameters that autoexpect generates for the benefit of the user. The resulting `script.exp` looks like this:

```
#!/usr/bin/expect -f
#
# This Expect script was generated by autoexpect on Thu Feb 3 15:19:40 2005
# Expect and autoexpect were both written by Don Libes, NIST.
#
# Note that autoexpect does not guarantee a working script. It
# necessarily has to guess about certain things. Two reasons a script
# might fail are:
#
# 1) timing - A surprising number of programs (rn, ksh, zsh, telnet,
# etc.) and devices discard or ignore keystrokes that arrive "too
# quickly" after prompts. If the new script is hanging up at
# one spot, try adding a short sleep just before the previous send.
# Setting "force_conservative" to 1 (see below) makes Expect do this
# automatically - pausing briefly before sending each character. This
# pacifies every program I know of. The -c flag makes the script do
# this in the first place. The -C flag allows the user to define a
# character to toggle this mode off and on.
set force_conservative 0;# set to 1 to force conservative mode even if
                        ;# script wasn't run conservatively originally
if {$force_conservative} {
        set send_slow {1 .1}
        proc send {ignore arg} {
                sleep .1
                exp_send -s -- $arg
        }
}
#
# 2) differing output - Some programs produce different output each time
# they run. The "date" command is an obvious example. Another is
# ftp, if it produces throughput statistics at the end of a file
# transfer. If this causes a problem, delete these patterns or replace
# them with wildcards. An alternative is to use the -p flag (for
# "prompt") which makes Expect only look for the last line of output
# (i.e., the prompt). The -P flag allows the users to define a character to
# toggle this mode off and on.
#
# Read the man page for more info.
#
# -Don
spawn $env(SHELL)
match_max 100000
expect -exact "]0;admin@localhost\[admin@localhost\]\$"
send -- "ssh -l setup 10.6.164.16\r"
```

```
expect -exact "ssh -l setup 10.6.164.16\r
\r
Sun Microsystems\r
IPMI v1.5 Service Processor\r
\r
Version: V2.1.0.16\r
Enter the name of the first manager level user: "
send -- "admin\r"
expect -exact "admin\r
Password: "
send -- "demopasswd\r"
expect -exact "\r
Confirm password: "
send -- "demopasswd\r"
expect -exact "\r
Enter the user's public key or press ENTER for no key: "
send -- "\r"
expect -exact "\r
\r
Successfully added the manager level user.\r
Connection to 10.6.164.16 closed by remote host.\r\r
Connection to 10.6.164.16 closed.\r\r
]0;admin@localhost\[admin@localhost\]\$"
set timeout -1
```

After a draft script is generated, it can be modified to include details that are required and cleaned up to avoid details that are not needed. In the following examples, the scripts are generated using the `-p` flag of autoexpect for clarity. The `-p` flag tells autoexpect to consider only the prompt before interaction, instead of all of the lines that come up after the command is issued.

Often, when a command is executed, different output can be expected from the terminal. For example, if the `ssh` command is run and the target host has never been accessed from the current administration console before, the following prompt appears:

```
The authenticity of host '<ip address (<hostname>)' can't be
established.
RSA key fingerprint is
a9:9f:c2:21:97:e9:e2:62:d0:5f:94:64:8c:08:2c:1c.
Are you sure you want to continue connecting (yes/no)?
```

In these cases, expect provides a way to branch into options of expected output as shown below:

```
expect {
          "^*(yes/no)?"     {send -- "yes\r"; exp_continue}
          "^*level user:" {send -- "setup\r"}
         }
```

This way, if the target host has never been used from the administration host, expect replies that the target host should be added to the list of `known_hosts`. If the target host has already been used, then the appropriate response is sent to the target host.

The final results, with the changes discussed above, are shown in the following scrips. The first example uses the IP address as a target host for LAN-based connectivity to the SP.

```
#!/usr/bin/expect -f

set force_conservative 0  ;# set to 1 to force conservative mode even if
                          ;# script wasn't run conservatively originally
if {$force_conservative} {
        set send_slow {1 .1}
        proc send {ignore arg} {
                sleep .1
                exp_send -s -- $arg
        }
}

set timeout -1
spawn $env(SHELL)
match_max 100000
expect -exact "\$
send -- "ssh -l setup <IP address of the SP>\r"
expect {
          "^*(yes/no)?"     {send -- "yes\r"; exp_continue}
          "^*level user:" {send -- "admin\r"}
         }
expect -exact "Password: "
send -- "demopass\r"
expect -exact "Confirm password: "
send -- "demopass\r"
expect -exact "Enter the user's public key or press ENTER for no key: "
send -- "\r"
expect -exact "\$"
expect eof
```

If the server's serial port is attached to a terminal concentrator, it requires a different script that includes the IP and port requirements of the terminal concentrator. Note in this case that the script does not need to branch between two different options before connecting to the host because it employs plain `telnet` to the terminal concentrator.

```
#!/usr/bin/expect -f

set force_conservative 0  ;# set to 1 to force conservative mode even if
                          ;# script wasn't run conservatively originally
if {$force_conservative} {
        set send_slow {1 .1}
        proc send {ignore arg} {
                sleep .1
                exp_send -s -- $arg
        }
}

set timeout -1
spawn $env(SHELL)
match_max 100000
expect -exact "\$"
send -- "telnet <TC's IP address> <port number>\r"
expect -exact ""
send -- "\r"
send -- "\r"
expect -exact "login: "
send -- "\r"
expect -exact "login: "
send -- "setup\r"
expect -exact "Enter the name of the first manager level user: "
send -- "admin\r"
expect -exact "Password: "
send -- "demopasswd\r"
expect -exact "Confirm password: "
send -- "demopasswd\r"
expect -exact "Enter the user's public key or press ENTER for no key: "
send -- "\r"
expect -exact "login: "
send -- "^]"
expect -exact "telnet> "
send -- "quit\r"
expect -exact "\$"
expect eof
```

By setting up the initial user, the administrator can now login to setup all of the preferences in the SP. For example, typical deployments that use the SP's serial port for initial configuration normally setup the IP address of the SP's network interface with a variation of the following command:

```
localhost $ sp set ip static -i 192.168.0.1 -n 255.255.255.0 -g
192.168.0.251
```

Data centers that have DHCP enabled in the management network use the command below to instruct the system to find a DHCP server for an address:

```
localhost $ sp set ip dhcp
```

This is a simple procedure, but when it must to be implemented across many servers it can become a time-consuming exercise. There are several methods to automate this process, which are described below.

If the serial interface is available, the easiest way to configure the server with an IP address is to write an expect script that performs the initial setup of the SP. With this script, all the of IP addresses can be assigned according to the physical location of the node in the datacenter or rack.

```
#!/usr/bin/expect -f

set force_conservative 1  ;# set to 1 to force conservative mode even if
                          ;# script wasn't run conservatively originally
if {$force_conservative} {
        set send_slow {1 .1}
        proc send {ignore arg} {
                sleep .1
                exp_send -s -- $arg
        }
}

set timeout -1
spawn $env(SHELL)
match_max 100000
expect -exact "\$"
send -- "telnet <TC's IP address> <port number>\r"
expect -exact ""
send -- "\r"
send -- "\r"
expect -exact "login: "
send -- "\r"
```

```
expect -exact "login: "
send -- "admin\r"
expect -exact "Password: "
send -- "demopasswd\r"
expect -exact "\$"
send -- "sp set ip static -i 192.168.0.1 -n 255.255.255.0 -g 192.168.0.251
-W\r"
expect -exact "\$"
send -- "exit\r"
expect -exact "login: "
send -- "#"
expect -exact "telnet> "
send -- "quit\r"
expect -exact "\$"
send -- "#"
expect eof
```

Managing Multiple Servers Simultaneously

The scripts detailed above, while saving time and errors by automating a task, are still performed on one SP at a time. It is possible to further automate the process by spawning multiple scripts in parallel, enabling multiple systems to be setup simultaneously. The following *spawner* script launches a number of processes at the same time.

```
#!/usr/bin/perl

for ($i=1;$i<=128;$i++) {
        print "192.168.0.$i \n";
        system ("<expect program name> 192.168.0.$i &");
}
```

This script is written in perl, but if perl is not available, a spawner script can also be written in any shell since the script does not require any perl-specific functionality. It is possible to improve these scripts by using the expect libraries in perl, but for introductory purposes it is much easier to maintain the expect scripts running independently from the perl code.

With a spawner script and some well-written expect scripts that execute different actions on the server, an administrator can construct a complete toolkit for managing a cluster of servers.

For example, one function that can be easily automated is powering servers on and off. In the following example, the spawner script calls the *poweron* script with a parameter that is the IP addresses of the SPs the users needs to control. Note that one of the parameters to launch the expect script is '*on*'. By running the same script with the '*off*' parameter, all of the servers in the IP range are turned off. Writing the expect scripts to accept parameters makes them more versatile and can result in fewer scripts.

```
#!/usr/bin/perl

for ($i=1;$i<=128;$i++) {
        print "192.168.0.$i \n";
        system ("./powercontrol.exp on 192.168.0.$i admin
demopasswd &");
}
```

In order to avoid confusion among different hosts, the `-exact` tags added after each expect command are removed, providing more flexibility in the hostnames the SPs or management consoles might have. The following expect script is called `powercontrol.exp` and should reside in the same directory as the spawner script above:

```
#!/usr/bin/expect --

set timeout -1

set action [lindex $argv 0]
set spip [lindex $argv 1]
set userid [lindex $argv 2]
set password [lindex $argv 3]

spawn $env(SHELL)
match_max 100000
expect "^*\$"
send -- "ssh -l $userid $spip\r"
expect {
         "^*(yes/no)?"     {send -- "yes\r"; exp_continue}
         "^*password: "   {send -- "$password\r"}
        }
expect "^* \$"
send -- "platform set power state $action\r"
expect "^* \$"
send -- "exit\r"
expect "^*\$"
send -- ""
expect eof
```

Updating Servers and Service Processor Firmware

It is occasionally necessary to update the BIOS of the system and/or the SP firmware. This is a manual process that must be performed carefully in order to maintain the integrity of the system. A full firmware update of the BIOS and the SP firmware can be accomplished safely, quickly, automatically, and simultaneously on multiple systems by creating small expect scripts that perform certain actions to configure or update pieces of the BIOS or SP.

The following example is a full script that updates the firmware (SP, BIOS, PPC, etc.) on a number of Sun Fire V20z servers. It is based on the principles outlined in previous sections of this article. In order for the script to work properly, a specific file system structure is required on the launch server to accommodate the paths, as shown in Table 2. The script can be modified to match the paths in a particular environment.

TABLE 2 Update Script Path Structure

Path	Description
`/usr/bin`	Location of the Java™ executable (configurable in spawner script)
`/export/scripts`	Spawner script goes here
`/export/scripts/expect`	Expect scripts go here
`/export/scripts/output`	Where the update status is logged
`/export/nsv2.2`	Where the firmware resides (configurable in spawner script)
`/export/nsv2.2/v20z`	Firmware for the Sun Fire V20z server
`/export/nsv2.2/v40z`	Firmware for the Sun Fire V40z server

This script requires that the directory where the firmware resides (called `nsvpath` in the script) must be exported using NFS with at least read permissions. In addition, the IP addresses of the target SPs must previously assigned in order for the script to be run on the range of addresses to update.

For version 2.2.0.6 of the firmware, first download and uncompress the firmware, that is in the file called nsv_V2_2_0_6.zip to the update system and unzip the file under a new folder (in this example the path is /export/nsv2.2/). Within that folder make two additional folders, one called “v20z” and one called “v40z”. In those folders download the nsv-v20z-bios-fw_V2_2_0_6h.zip and nsv-v40z-bios-fw_V2_2_0_6h.zip respectively, and uncompress them. This sets up the environment to allow for Sun Fire V20z and V40z platform update. These file below are the main files that should be in the directories::

```
Files included in this paper:
/export/scripts/update
/export/scripts/expect/update2.sp
/export/scripts/expect/update3.sp
/export/scripts/expect/setup.sp

Files downloaded from http://www.sun.com/
/export/nsv2.2/v20z/sw_images/platform/firmware/bios/V1.32.7.2/swinventory.xml
/export/nsv2.2/v20z/sw_images/platform/firmware/bios/V1.32.7.2/bios.sp
/export/nsv2.2/v20z/sw_images/sp/spbase/V2.2.0.18/swinventory.xml
/export/nsv2.2/v20z/sw_images/sp/spbase/V2.2.0.18/install.image

/export/nsv2.2/v40z/sw_images/platform/firmware/bios/V2.32.8.2/swinventory.xml
/export/nsv2.2/v40z/sw_images/platform/firmware/bios/V2.32.8.2/bios.sp
/export/nsv2.2/v40z/sw_images/sp/spbase/V2.2.0.18/swinventory.xml
/export/nsv2.2/v40z/sw_images/sp/spbase/V2.2.0.18/install.image

/export/nsv2.2/update_server/V2.2.0.6/swinventory.xml
/export/nsv2.2/update_server/V2.2.0.6/updateServer.jar
/export/nsv2.2/update_server/V2.2.0.6/updateServer.config
```

As root, share the `/export` directory using NFS. To do this, edit the `/etc/dfs/dfstab` file and add the following entry:

```
share -F nfs /export
```

Restart the NFS server using the following command as root:

```
# /etc/init.d/nfs.server restart
```

The next step is to edit the spawner script to reflect the correct network settings and desired preferences. These changes include a number of variables that may be left as default, but some should change as they are different in every case. Here are a few suggestions of settings that may need to be changed:

- The path to the `expect` executable needs to be correct in all of the `expect` scripts in order to run.
- Once the update system is setup, it is possible to reset all of the server SPs to factory defaults in order to have a consistent way of logging in to the SPs and running the update commands. If there is already a standard login and password for the SPs, they need to be configured in the spawner script.
- If only one SP is to be updated, simply assign the same value to both the `$first` and `$last` variables in the spawner script. This forces the script to only run once.

The full spawner script is below. As stated in the script, it should be launched on the update Network Share Volume (NSV) server. The NSV files must be installed on the NSV server and the NSV server must have perl, Java Runtime Environment (JRE) 1.4.2 or later, and expect installed. For more information on updating the BIOS, refer to the SunSolve™ article *"How to Update Sun Fire V20/40z SP and Platform Firmware/BIOS", Document ID: 79931*.

```
#!/usr/bin/perl
# Service Processor Update utility
#
# Purpose: The purpose of this script is to give an example of how to write
# a script that updates the firmware (SP, BIOS, PPC, ...) on a number
# of Sun Fire V20z/V40z servers.
#
# This utility is provided as-is and is not supported in anyway.
#
# Usage: Modify the variables below to match the environment
# and launch the script on the update (NSV) server.
#
# Requirements: The NSV files must be downloaded and extracted on the NSV
# server.
# The latest NSV files can be downloaded from these URLs:
# http://www.sun.com/servers/entry/v20z/downloads.html
# http://www.sun.com/servers/entry/v40z/downloads.html
## The NSV server must have perl, JRE 1.4.2 or later and expect installed.
#
# This script was written for the Sun Fire V20z and will not fully update
# the V40z (The PRS is not updated).
#
# -----------------------------------------------------------------------
#          Modify the variables below before launching the script
#
# NOTE: Make sure to leave a colon ';' at the end of each variable assignment
#
# Network information
#
# Network that the Service Processors are connected to
#

$network="10.6.164.0";

# IP range
#
# IP range assigned to the Service Processors, provide first and last IP
#
$first=15;
$last=15;
```

```
# SP Authentication info
#
# Authentication to be used to log on the Service Processors
# $firstsetup = "yes" if the SPs need to be initialized or "no" if the first
# setup has already been done. If the SP have already been initialized
# use $manager and $password to provide the login and password that the SPs
# have been initialized with. If the SP have never been setup or if they have
# been reset to default settings, use $manager and $password to provide
# the new login and password to be created on the SPs.
#

$firstsetup="yes";
$manager="sun";
$password="sun";

# Update server information
# $serverip is the IP address of the update (NSV) server that the SP will
# connect to. This server must be running JRE 1.4.2 or later and this script.
# $javabinpath is the path to the java executable. If configured, make sure
# sure there is a trailing "/". Leave blank if already in path.
# $nsvpath is the path where the NSV software distribution has been unzipped
#
$serverip="129.146.75.253";
$javabinpath="/usr/bin/";
$nsvpath="/export/nsv2.2/";
# File names of the version images to be installed
# The filenames and paths below are documented in the update procedure
# available for download with the NSV files.
# $ppcupdate is the PPCboot Update file name including the PATH relative
# to the NSV folder ($nsvpath declared above). Must start with a '/'
# $picupdate is the PIC revision. Same format as the $ppcupdate.
# $spversion is the SP revision. It is used in the path, the filename
# is always the same.
# $portnumber is the port number to be used for the spupdate connection and
# the range for this parameter is 49152 - 65535.
# $bios is the BIOS revision. It is used in the path, the filename is
# always the same.
#
$platform="v20z";
$spversion="V2.2.0.18";
$portnumber="50000";
$bios="V1.32.7.2";
#
```

```
#
# DO NOT MODIFY ANYTHING BELOW THIS LINE UNLESS YOU KNOW WHAT YOU ARE DOING
# -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
$network =~ s/\.0//;

$test=`ps -fea|grep java|grep -i update |grep -v grep |wc -l`;
chomp $test;
$test=~s/ //g;
if ($test == 1) {
        print "Killing previously running java server to avoid conflicts\n\n";
        system "kill -9 `ps -fea|grep java|grep -i update|grep -v kill |sed -e
's/ [ ]*/ /g'|cut -d' ' -f2`";
}

#
# Remove the fingerprint for the RSA key from the known_hosts file
#
for ($i=$first;$i<=$last;$i++) {
    `grep -v $network.$i $ENV{HOME}/.ssh/known_hosts > $ENV{HOME}/.ssh/
known_hosts.new`;
    `mv $ENV{HOME}/.ssh/known_hosts.new $ENV{HOME}/.ssh/known_hosts`;
}

#
# Do we need to do initial setup on the SP's?
#

if ($firstsetup eq "yes") {
        for ($i=$first;$i<=$last;$i++) {
                print "Setting up SP $network.$i \n";
                system ("./expect/setup.sp $network.$i $manager $password >
output/update/setup.$network.$i.out &");
        }
print "Waiting for initial setup of the SPs to complete\n";
while ($test != 2) {
        print ".";
        $test=(`ps -fea|grep setup|wc -l`);
        chomp $test;
        $test=~s/ //g;
        sleep 2;
}
print "\n";
sleep 15;
}
#
# Run SP fw update on the SP
#
```

```
#
# Start the server
#
print "Starting the java-based update server locally... \n\n";
system (`echo "$javabinpath/java -jar $nsvpath/update_server/V2.2.0.6/
updateServer.jar -f $nsvpath/$platform/sw_images/sp/spbase/$spversion/
install.image -p $portnumber" > output/update/jar.out`);

system (`$javabinpath/java -jar $nsvpath/update_server/V2.2.0.6/
updateServer.jar -f $nsvpath/$platform/sw_images/sp/spbase/$spversion/
install.image -p $portnumber >> output/update/jar.out &`);

sleep 5;
#
# Launch the clients
#

print "Now running the service processor firmware update on the SPs\n\n";
for ($i=$first;$i<=$last;$i++) {
        print "$network.$i \n";
        system (`echo "*** Running SP fw update now ***" >> output/update/
$network.$i.out`);
        system ("./expect/update2.sp $network.$i $serverip $portnumber
$manager $password >> output/update/$network.$i.out &");
}
print "Waiting for SP firmware update to complete\n";
$test=0;
$nomachines=$last-$first+1;
while ($test != $nomachines) {
        print ".";
        $test=(`grep 'Update complete' output/update/jar.out|wc -l`);
        chomp $test;
        $test=~s/ //g;
        sleep 2;
}
print "\n";
print "Waiting for SPs to reboot\n";
sleep 30;
while ($test ne "alive") {
        print ".";
        $test=(`ping $network.$last | cut -d' ' -f3`);
        chop $test;
        sleep 5;
}
print "\n\n";
```

```
#
# HERE WE HAVE TO KILL THE java SERVER!
#

system "kill -9 `ps -fea|grep java|grep -i update|grep -v grep|sed -e 's/ [
]*/ /g'|cut -d' ' -f2`";

#
# Update the BIOS on the servers
#
print "Now updating the pic and the BIOS on the servers\n\n";
for ($i=$first;$i<=$last;$i++) {
        print "$network.$i \n";
        system (`echo "*** Running BIOS update now ***" >> output/update/
$network.$i.out`);
        system ("./expect/update3.sp $network.$i $bios $manager $password
$platform $nsvpath $serverip >> output/update/$network.$i.out &");
}
print "Waiting for BIOS update to complete\n";
while ($test != 2) {
        print ".";
        $test=(`ps -fea|grep -i update3|wc -l`);
        chomp $test;
        $test=~s/ //g;
        sleep 2;
}
print "\n";
print "Waiting for SP to reboot\n";
sleep 3;
while ($test ne "alive") {
        print ".";
        $test=(`ping $network.$last | cut -d' ' -f3`);
        chop $test;
        sleep 5;
}
print "\n\n";
```

setup.sp

The `setup.sp` script is used if an initial setup user is required for the SP..

```
#!/opt/csw/bin/expect --

set timeout -1

set spip [lindex $argv 0]
set userid [lindex $argv 1]
set password [lindex $argv 2]

spawn ssh -l setup $spip
expect {
        (yes/no) {send "yes\r"
                        exp_continue}
        user:   {send "$userid\r"
                        }
    assword: {exit}
        }
expect assword:
sleep 1
send "$password\r"
expect assword:
sleep 1
send "$password\r"
expect key:
sleep 2
send "\r"
```

update2.sp

The `update2.sp` script runs the `sp update flash all` subcommand. which updates the entire SP flash image (kernel, base file system, and value add) as part of a major SP software update. This command requires the Java Update Server and verifies its availability before beginning the update process. Once verified, the SP is rebooted and the update process is initiated. When completed, the SP is automatically rebooted using the updated image.

```
#!/opt/csw/bin/expect --

set timeout -1

set spip [lindex $argv 0]
set serverip [lindex $argv 1]
set portnumber [lindex $argv 2]
set userid [lindex $argv 3]
set password [lindex $argv 4]

spawn ssh -l $userid $spip
expect {
    (yes/no) {send "yes\r"
         exp_continue}
    password:   {send "$password\r"
         }
    }
expect " \$ "
send "sp update flash all -i $serverip -p $portnumber\r"
expect {
    " \$ "     {send "exit\r"}
    "Error."   {sleep 2
        send "sp update flash all -i $serverip -p $portnumber\r"
        exp_continue}
    }
```

update3.sp

The `update3.sp` script runs the command `platform set os state update-bios` command to update the BIOS of the system. This command can take several minutes to run. When the BIOS update completes, the server automatically shuts down.

```
#!/opt/csw/bin/expect --

set timeout -1

set spip [lindex $argv 0]
set bios [lindex $argv 1]
set userid [lindex $argv 2]
set password [lindex $argv 3]
set platform [lindex $argv 4]
set nsvpath [lindex $argv 5]
set serverip [lindex $argv 6]

spawn ssh -l $userid $spip
expect {
    (yes/no) {send "yes\r"
         exp_continue}
    password:   {send "$password\r"
    }
}
expect " \$ "
send "sp add mount -r $serverip:$nsvpath -l /mnt\r"
expect " \$ "
send "platform set power state off -f\r"
expect " \$ "
send "platform set os state update-bios /mnt/$platform/
sw_images/platform/firmware/bios/$bios/bios.sp\r"
expect " \$ "
send "exit\r"
```

Summary

Although smaller, horizontally scalable servers such as the Sun Fire V20z and V40z servers are more cost effective initially, it can be costly and time consuming to administer large numbers of them manually. Any steps toward automating tasks that are usually performed manually can help decrease the total cost of ownership of large pools of small servers, as well as decrease downtime due to human error.

The expect and spawner scripts provided in this article are easy to create or modify and can help administrators save time and effort by automating low-level repetitive tasks.

Appendix A: Connecting a Laptop to the Service Processor

This appendix outlines the procedures for connecting a laptop to the service process via a Solaris platform, a Linux platform, or a Windows platform. Laptop connectivity offers a convenient way to learn how serial connections work, making it easier to connect a terminal concentrator.

The following components are required to connect a laptop to the SP on a Sun Fire V20z or V40z server:

- A cross-over serial cable (null modem)
- A laptop or workstation with a serial port or a laptop or workstation with a USB port and a USB-to-serial adapter
- A communications software package such as:
  - Minicom for Linux
  - tip for Solaris OS
  - Hyperterm for Windows

There are many combinations of adapters and cables that can be used to connect a laptop or workstation to the serial port. The following combination is known to work:

- Two DB25-(male) to-DB9 (female) cables
- One DB25-(female) to-DB25 (female) null modem cable
- One USB-to-DB9 (male) adapter

Connect a DB25-to-DB9 connector to the null modem cable on both ends, and connect the DB9 end of the USB-to-DB9 cable to one of the ends of the resulting cable.

Connecting via a Solaris Platform

The following steps detail how to connect to the serial port via a Solaris platform. Keep in mind that not all USB-to-serial adapters work, so make sure the adapter is recognized by the Solaris OS before attempting this connection. If the adapter is not recognized, it may be necessary to use a workstation or laptop with a serial port.

1. Plug in the cable (not a USB-to-adapter) to the laptop on one end, and to the serial port of the server on the other end. Open up a terminal window on the Solaris laptop and enter:

```
# tip -115200 /dev/term/a
```

2. Press the '**Enter'** key and the SP should return a prompt. The administration user can now be configured.

Connecting via a Linux Platform

The following steps detail how to connect to the serial port via a Linux platform.

1. Determine how the kernel recognizes the USB-to-serial adapter. To do this, monitor the `/var/log/messages` log with the following command as root:

```
# tail -f /var/log/messages
```

2. Plug in the device and watch the output on the log file and look for a line that reads something like:

```
Jan  4 17:28:01 localhost kernel: usb 3-2: PL-2303 converter now
attached to ttyUSB0
```

   This indicates that the serial device to use is `ttyUSB0`.

3. Configure the minicom communication software to use this serial device. As root, start minicom in setup mode:

```
# minicom -s
```

4. Go to '*Serial port setup*' and type '**A'** to correctly configure the serial device identified above. For example: `/dev/ttyUSB0`.

5. Change the '*E*' setting to reflect the following settings:
   - Speed: **115200**
   - Parity: **None**
   - Data: **8**
   - Stopbits: **1**
6. Press '**Enter**' to exit.
7. Hardware and software flow control should be turned '*off*'. Go back to the '*Configuration*' menu by pressing '**Enter**' at the '*Serial port setup*' menu.
8. Select the '*Modem and dialing*' menu to clear the '*init*' strings for modem communications. Clear the '*A*' and 'B' options from the menu and leave them blank. Press '**Enter**' at the '*Modem and dialing parameter setup*' menu to return to the '*Configuration*' menu.
9. Select '*Save setup as dfl*' to save these settings.
10. Run minicom with the new settings.

```
# minicom
```

Connecting via a Windows Platform

The following steps detail how to connect to the serial port via a Windows platform.

1. Connect the USB-to serial-adapter to the laptop or workstation and the serial port.
2. Windows requires a driver disk for this device. Insert the provided CD in the drive and inform the Wizard the driver for the new hardware is to be provided. Windows then searches for the driver in the CD and identifies the new hardware as a USB-to serial-device assigning it a COM port. For example, it could be COM5.
3. To determine which COM port is assigned to the USB-to-serial adapter, open the Windows Device Manager (*My Computer > properties > Hardware > device manager*) and look under '*Ports (COM & LPT)*' and identify the USB-to serial-adapter or 'bridge' and which COM port is assigned to it.
4. Next, open Hyperterm and type a name for this connection. When the initial setup screen opens up, go to the bottom combination box and change the setting to whichever COM port was discovered in Step 3 and press OK.

5. Change the terminal settings to be:
   - Speed: **115200**
   - Parity: **None**
   - Data: **8**
   - Stopbits: **1**
   - Flow control: **None**

Communicating to the Service Processor

Once the laptop or workstation is configured correctly and the communications program is running, make sure the server is turned off (the OS is not running) and the laptop or workstation is connected to the serial port.

If the server is new and has never been configured, press '**Enter**' to trigger the login prompt for the SP. If the SP's serial port behavior has been modified, it is necessary to access the SP via the SSH/LAN interface to configure the SP to access it from the serial port. This is accomplished by issuing the following command to the SP:

```
localhost$ platform set console -s sp -e -S <baud rate speed>
```

Now press '**Enter**' at the platform console to trigger a login prompt from the SP. If this is the first time the SP is accessed, use the `setup` user to proceed with the initial setup of the SP. Otherwise, simply login to the SP.

Appendix B: Connecting a Terminal Concentrator to the Serial Port

A terminal concentrator can provide several benefits when managing multiple servers. For example, a terminal concentrator can access the SP of a server without first assigning an IP address to the server. A terminal concentrator can provide an *always on* out-of-band connection to a number of servers.

The following components are required to connect a terminal concentrator to the serial port:

- DB-9 (female) to RJ-45 connector
- Rollover cable (flat, green cable that comes with CISCO terminal concentrator)

1. Plug the DB-9 connector into the server's serial port and attach the rollover cable from the connector to the terminal concentrator.
2. Access the terminal concentrator and make sure the settings for the port are:
   - Speed: **115200**
   - Parity: **None**
   - Data: **8**
   - Stopbits: **1**
   - Flow control: **None**

On CISCO terminal concentrators, the settings for the port should look like this:

```
smpk16-nts1#show line 97
   Tty Typ     Tx/Rx    A Modem  Roty AccO AccI   Uses   Noise  Overruns   Int
    97 TTY 115200/115200-    -      -    -    -     13      22     1/2       -

Line 97, Location: "", Type: ""
Length: 24 lines, Width: 80 columns
Baud rate (TX/RX) is 115200/115200, no parity, 1 stopbits, 8 databits
Status: Ready, Modem Speed Locked
Capabilities: EXEC Suppressed
Modem state: Ready
Modem hardware state: noCTS* noDSR*  DTR RTS
Special Chars: Escape  Hold  Stop  Start  Disconnect  Activation
                ^^x    none   -     -       none
Timeouts:      Idle EXEC    Idle Session   Modem Answer  Session   Dispatch
               00:10:00        never                      none     not set
                            Idle Session Disconnect Warning
                              never
                            Login-sequence User Response
                             00:00:30
                            Autoselect Initial Wait
                              not set
Modem type is unknown.
Session limit is not set.
Time since activation: 00:00:59
Editing is enabled.
History is enabled, history size is 10.
DNS resolution in show commands is enabled
Full user help is disabled
Allowed input transports are telnet.
Allowed output transports are pad v120 lapb-ta telnet rlogin.
Preferred transport is telnet.
No output characters are padded
No special data dispatching characters
```

Once these settings are configured, the SP should be accessible via the terminal concentrator. If there is no output from the console, make sure all the settings are correct. If there is still no output, reboot the SP by either unplugging the power supplies and then plugging them back in, or by accessing the SP via the LAN port and rebooting it using the command or by press-and-holding the service processor's *reset* (||) button on the rear of the server:

```
localhost$ sp reboot
```

About the Author

Jacques Bessoudo is the Technical Marketing Engineer Specialist on Field Engagement and the blade platform. With Sun for seven years, Jacques started as a telco-oriented Systems Engineer in the Mexico City sales office. He joined Technical Marketing with the Netra™ server group to support the field and sales development activities. He later joined the Competitive Intelligence group providing technical insight on competitive platforms and tracking/analyzing benchmark results, before moving to his current role.

References

*Sun Fire V20z and V40z Server Architectures, A Technical White Paper, July 2004 (*on `www.sun.com/servers/entry/v40z/arch-wp.pdf`*)*

Sun Fire V20z and Sun Fire V40z Servers — Server Management Guide, Part No. 817-5249-14 (on `www.sun.com/products-n-solutions/hardware/docs/Servers/Workgroup_Servers/Sun_Fire_V20z/index.html`)

Sun Fire V20z and Sun Fire V40z Servers Installation Guide, Part No. 817-5246-15 (on `www.sun.com/products-n-solutions/hardware/docs/Servers/Workgroup_Servers/Sun_Fire_V20z/index.html`)

How to Update Sun Fire V20/40z SP and Platform Firmware/BIOS, SunSolve ID: 79931 (on `sunsolve.sun.com`, insert ID into “Search Knowledgebase” search box)

Ordering Sun Documents

The SunDocs℠ program provides more than 250 manuals from Sun Microsystems, Inc. If you live in the United States, Canada, Europe, or Japan, you can purchase documentation sets or individual manuals through this program.

Accessing Sun Documentation Online

The `docs.sun.com` Web site enables access Sun technical documentation online, where it is possible to browse the `docs.sun.com` archive or search for a specific book title or subject. The URL is `http://docs.sun.com/`

To reference Sun BluePrints™ OnLine articles, visit the Sun BluePrints OnLine Web site at: `http://www.sun.com/blueprints/online.html/`

Accessing SunSolve Online

The `sunsolve.sun.com` Web site is a portal for support resources, features and articles, product support sites, Sun alerts, and diagnostic tools. Sun support customers can access even more information with their Sun support user name and password.