Documentation: Zones

The following are a few examples for the configuration of Solaris 10 Zones. Currently with a focus on Resource Controls.
Note: This document is not written by Sun.
Brendan Gregg, 11-Sep-2006, version 2.32.

What is a Zone?

- Virtual instance of Solaris
- Software partition for the OS

A large SunFire server with hardware domains allows many isolated systems to be created. Zones achieve this in software and is far more flexible - it is easy to move individual CPUs between zones as needed, or to configure a more sophisticated way to share CPUs and memory.

Contents

Operating Systems

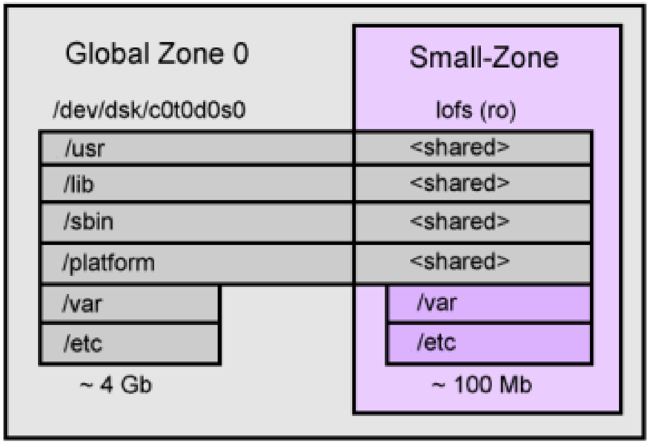
This document is based on the first customer release of Solaris 10. In the later revisions, many of the commands and syntax may be improved. Check the "what's new" sections on docs.sun.com to keep up to date with changes.

Examples

The following are examples that demonstrate a particular function of Zones.

Small-Zone

This demonstrates creating a simple zone that uses the default settings which share most of the operating system with the global zone. The final layout will be like the following,



To create such a zone involves letting the system pick default settings, which includes the loopback filesystem (lofs) read only mounts that share most of the OS. The following commands were used,

```
# zonecfg -z small-zone
small-zone: No such zone configured
Use 'create' to begin configuring a new zone.
zonecfg:small-zone> create
zonecfg:small-zone> set autoboot=true
zonecfg:small-zone> set zonepath=/export/small-zone
zonecfg:small-zone> add net
zonecfg:small-zone:net> set address=192.168.2.101
zonecfg:small-zone:net> set physical=hme0
zonecfg:small-zone:net> end
zonecfg:small-zone> info
zonepath: /export/small-zone
autoboot: true
pool:
inherit-pkg-dir:
        dir: /lib
inherit-pkg-dir:
        dir: /platform
inherit-pkg-dir:
        dir: /sbin
inherit-pkg-dir:
        dir: /usr
net:
        address: 192.168.2.101
        physical: hme0
zonecfg:small-zone> verify
zonecfg:small-zone> commit
zonecfg:small-zone> exit
#
# zoneadm list -cv
  ID NAME             STATUS         PATH
   0 global           running        /
   - small-zone       configured     /export/small-zone
```

The new zone is in a configured state. Those inherited-pkg-dir's are filesystems that will be shared lofs (loopback filesystem) readonly from the global; this saves copying the entire operating system over during install, but can make adding packages to the small-zone difficult as /usr is readonly. (See the big-zone example that uses a different approach).

We can see the zonecfg command has saved the info to an XML file in /etc/zones,

```
# cat /etc/zones/small-zone.xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE zone PUBLIC "-//Sun Microsystems Inc//DTD Zones//EN" "file:///usr/share/lib/xml/dtd/zone
<!--
    DO NOT EDIT THIS FILE.  Use zonecfg(1M) instead.
-->
<zone name="small-zone" zonepath="/export/small-zone" autoboot="true">
  <inherited-pkg-dir directory="/lib"/>
  <inherited-pkg-dir directory="/platform"/>
  <inherited-pkg-dir directory="/sbin"/>
  <inherited-pkg-dir directory="/usr"/>
  <network address="192.168.2.101" physical="hme0"/>
</zone>
```

Next we begin the zone install, it takes around 10 minutes to initialise the packages it needs for the new zone. A verify is run first to check our zone config is ok, then we run the install, then boot the zone,

```
# mkdir /export/small-zone
# chmod 700 /export/small-zone
#
# zoneadm -z small-zone verify
#
# zoneadm -z small-zone install
Preparing to install zone <small-zone>.
Creating list of files to copy from the global zone.
Copying <2574> files to the zone.
Initializing zone product registry.
Determining zone package initialization order.
Preparing to initialize <987> packages on the zone.
Initialized <987> packages on zone.
Zone <small-zone> is initialized.
Installation of these packages generated warnings: <SUNWcsr SUNWdtdte>
The file </export/small-zone/root/var/sadm/system/logs/install_log> contains a log of the zone ins
#
# zoneadm list -cv
  ID NAME             STATUS         PATH
   0 global           running        /
   - small-zone       installed      /export/small-zone
#
# zoneadm -z small-zone boot
#
# zoneadm list -cv
  ID NAME             STATUS         PATH
   0 global           running        /
   1 small-zone       running        /export/small-zone
```

We can see small-zone is up and running. Now we login for the first time to the console, so we can answer system identification questions such as timezone,

```
# zlogin -C small-zone
[Connected to zone 'small-zone' console]
100/100
What type of terminal are you using?
 1) ANSI Standard CRT
 2) DEC VT52
 3) DEC VT100
 4) Heathkit 19
 5) Lear Siegler ADM31
 6) PC Console
 7) Sun Command Tool
 8) Sun Workstation
 9) Televideo 910
 10) Televideo 925
 11) Wyse Model 50
 12) X Terminal Emulator (xterms)
 13) CDE Terminal Emulator (dtterm)
 14) Other
Type the number of your choice and press Return: 13
...standard questions...
```

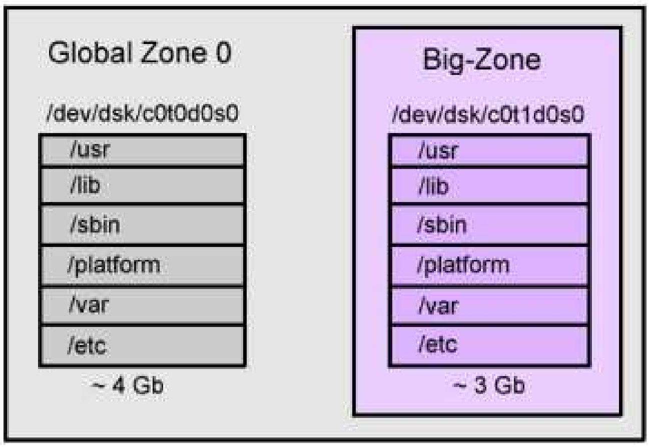
The system then reboots. To get an idea of what this zone actually is, lets poke around it's zonepath from the global zone,

```
/> cd /export/small-zone
/export/small-zone> ls
dev    root
/export/small-zone> cd root
/export/small-zone/root> ls
bin       etc       home      mnt       opt       proc      system    usr
dev       export    lib       net       platform  sbin      tmp       var
/export/small-zone/root> grep lofs /etc/mnttab
/export/small-zone/dev  /export/small-zone/root/dev     lofs    zonedevfs,dev=4e40002   111044677(
/lib    /export/small-zone/root/lib     lofs    ro,nodevices,nosub,dev=2200008 1110446770
/platform       /export/small-zone/root/platform        lofs    ro,nodevices,nosub,dev=2200008  11
/sbin   /export/small-zone/root/sbin    lofs    ro,nodevices,nosub,dev=2200008 1110446770
/usr    /export/small-zone/root/usr     lofs    ro,nodevices,nosub,dev=2200008 1110446770
/export/small-zone/root> du -hs etc var
  38M   etc
  30M   var
/export/small-zone/root>
```

From the directories that are not lofs shared from the global zone, the main ones are /etc and /var. They add up to around 70Mb, which is roughly how much extra disk space was required to create this small-zone.

Big-Zone

This demonstrates creating a zone that resides on it's own slice, which has it's own copy of the operating system. The final layout will be like the following,



First we create the slice,

```
# newfs /dev/dsk/c0t1d0s0
newfs: construct a new file system /dev/rdsk/c0t1d0s0: (y/n)? y
/dev/rdsk/c0t1d0s0:     16567488 sectors in 16436 cylinders of 16 tracks, 63 sectors
        8089.6MB in 187 cyl groups (88 c/g, 43.31MB/g, 5504 i/g)
super-block backups (for fsck -F ufs -o b=#) at:
 32, 88800, 177568, 266336, 355104, 443872, 532640, 621408, 710176, 798944,
 15700704, 15789472, 15878240, 15967008, 16055776, 16144544, 16233312,
 16322080, 16410848, 16499616,
#
# ed /etc/vfstab
362
$a
/dev/dsk/c0t1d0s0  /dev/rdsk/c0t1d0s0  /export/big-zone   ufs   1   yes   -
.
w
455
q
#
# mkdir /export/big-zone
# mountall
checking ufs filesystems
/dev/rdsk/c0t1d0s0: is clean.
mount: /tmp is already mounted or swap is busy
#
```

Now we configure the zone to not use any inherit-pkg-dir's by using several "remove inherit-pkg-dir" commands.

```
# zonecfg -z big-zone
big-zone: No such zone configured
Use 'create' to begin configuring a new zone.
zonecfg:big-zone> create
zonecfg:big-zone> remove inherit-pkg-dir dir=/sbin
zonecfg:big-zone> remove inherit-pkg-dir dir=/usr
zonecfg:big-zone> remove inherit-pkg-dir dir=/platform
zonecfg:big-zone> remove inherit-pkg-dir dir=/lib
zonecfg:big-zone> set autoboot=true
zonecfg:big-zone> set zonepath=/export/big-zone
zonecfg:big-zone> add net
zonecfg:big-zone:net> set address=192.168.2.201
zonecfg:big-zone:net> set physical=hme0
zonecfg:big-zone:net> end
zonecfg:big-zone> info
zonepath: /export/big-zone
autoboot: true
pool:
net:
        address: 192.168.2.201
```

```
        physical: hme0
zonecfg:big-zone> verify
zonecfg:big-zone> commit
zonecfg:big-zone> exit
#
# cat /etc/zones/big-zone.xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE zone PUBLIC "-//Sun Microsystems Inc//DTD Zones//EN" "file:///usr/share/lib/xml/dtd/zone
<!--
    DO NOT EDIT THIS FILE.  Use zonecfg(1M) instead.
-->
<zone name="big-zone" zonepath="/export/big-zone" autoboot="true">
  <network address="192.168.2.201" physical="hme0"/>
</zone>
#
# chmod 700 /export/big-zone
#
# df -h /export/big-zone
Filesystem             size   used  avail capacity  Mounted on
/dev/dsk/c0t1d0s0      7.8G   7.9M   7.7G     1%    /export/big-zone
#
# zoneadm -z big-zone verify
# zoneadm -z big-zone install
Preparing to install zone <big-zone>.
Creating list of files to copy from the global zone.
Copying <118457> files to the zone.
...
```

After the zone has been installed and booted, we now check the size of the dedicated zone slice,

```
# df -h /export/big-zone
Filesystem             size   used  avail capacity  Mounted on
/dev/dsk/c0t1d0s0      7.8G   2.9G   4.8G    39%    /export/big-zone
```

Wow! 2.9Gb, pretty much most of Solaris 10. This zone resides on it's own slice, and can add many packages as though it was a separate system. Using inherit-pkg-dir as happened with small-zone can be great, but it's good to know we can do this as well.

Scripts 1

The following are some scripts to make life with zones easier.

zonerun can be used to run a command in all non global zones in one blow. For example,

```
# zonerun uname -a
SunOS workzone1 5.10 Generic sun4u sparc SUNW,Ultra-5_10
SunOS workzone2 5.10 Generic sun4u sparc SUNW,Ultra-5_10
SunOS workzone3 5.10 Generic sun4u sparc SUNW,Ultra-5_10
SunOS workzone4 5.10 Generic sun4u sparc SUNW,Ultra-5_10
#
# zonerun -v uptime
workzone1,
    11:18pm  up  2:40,  0 users,  load average: 0.24, 0.22, 0.40
workzone2,
    11:18pm  up  2:38,  0 users,  load average: 0.24, 0.22, 0.40
workzone3,
    11:18pm  up  2:39,  1 user,  load average: 0.24, 0.22, 0.40
workzone4,
    11:18pm  up  2:39,  0 users,  load average: 0.24, 0.22, 0.40
```

so above we can see the boot times for all the zones by running uptime in each of them. The load averages are the same as they are for the overall system, not per zone (Kstat currently has system wide avenrun's only).

zonefss can help manage the Fair Share Schedular (FSS) setting when using FSS resource control for CPU usage. For example,

```
# zonefss -l
  ID NAME                SHARES
   0 global              1
   0 workzone1           5
   0 workzone2           10
```

The above lists the number of FSS shares for each zone. Next we change the number of shares for the global zone; this change is temporary and will revert to the default at next reboot,

```
# zonefss -z global 20
#
# zonefss -l
  ID NAME            SHARES
   0 global          20
   0 workzone1       5
   0 workzone2       10
```

Other zone shares can be changed live in the same way.

zoneflogcpu is designed to generate CPU load on all non-global zones, for the purposes of testing CPU resource control. In the following demonstration the command is run with "start" to create CPU load, which is then observed using uptime and prstat; then the command is run with "stop" and the CPU load can be seen to dissapear.

```
global# zoneflogcpu -v start
workzone1, started.
workzone2, started.
workzone3, started.
workzone4, started.
global#
global# uptime
 12:41am  up 1 day(s), 22:57,  2 users,  load average: 34.33, 21.64, 15.82
global#
global# prstat -n8 -Z
   PID USERNAME  SIZE   RSS STATE  PRI NICE      TIME  CPU PROCESS/NLWP
 20804 root     3960K 2136K run     20    0   0:00:51 3.5% perl/1
 20834 root     3960K 2136K run     31    0   0:00:55 3.3% perl/1
 20756 root     3960K 2136K run     22    0   0:00:51 3.1% perl/1
 20744 root     3960K 2136K run     32    0   0:00:53 3.0% perl/1
 20765 root     3960K 2136K run     22    0   0:00:52 3.0% perl/1
 20840 root     3960K 2136K run     11    0   0:00:46 3.0% perl/1
 20786 root     3960K 2136K run     22    0   0:00:48 3.0% perl/1
 20774 root     3960K 2136K run     32    0   0:00:50 3.0% perl/1
ZONEID    NPROC  SIZE   RSS MEMORY      TIME  CPU ZONE
    11       43  159M   97M   0.6%   0:07:29  25% workzone4
     9       43  159M   97M   0.5%   0:07:33  25% workzone2
     8       42  155M   94M   0.5%   0:07:53  25% workzone1
    10       43  159M   97M   0.5%   0:07:29  25% workzone3
     0       48  183M   80M   0.4%   0:00:50 0.1% global
Total: 219 processes, 595 lwps, load averages: 34.27, 21.54, 15.78
global# zoneflogcpu -v stop
stopped on all zones.
global#
global# uptime
 12:46am  up 1 day(s), 23:02,  2 users,  load average: 0.44, 9.65, 12.34
global#
```

eat memory The following perl command can be used to eat around 20 Megabytes of memory, and to stay that way for an hour; we run a "ps" command to check that it worked,

```
# perl -e '$a = "A" x 10_000_000; sleep 3600' &
[2] 2808
#
# ps -o pid,vsz,rss,args -p 2808
  PID   VSZ   RSS COMMAND
 2808 22440 21104 perl -e $a = "A" x 10_000_000; sleep 3600
#
```

To consume a different number of megabytes, change the "10_000_000" value to half your desired bytes (eg, 50_000_000 will eat around 100 Mb).

zcp can be used to copy a file from the global zone to a non global zone. This pipes the file through zlogin; there are far better ways to move large files or directories, such as copying from the global zone to the zonepath, or by using shared mounts. This can just be thought of as a handy last resort,

```
# zcp /etc/syslog.conf workzone1:/tmp
zcp from /etc/syslog.conf, to zone workzone1, to file /tmp/syslog.conf.
Copy successful (/tmp/syslog.conf, 1001 bytes).
#
# ls -l /etc/syslog.conf
-rw-r--r--   1 root     sys         1001 Mar  4 11:15 /etc/syslog.conf
#
# zlogin -S workzone1 ls -l /tmp/syslog.conf
-rw-r--r--   1 root     root        1001 Mar 10 19:26 /tmp/syslog.conf
```

```
#
```

Resource Control Summary

The following is a summary of resource control techniques. See the sections below for further details on each,

Resource	Fine Control	Coarse Control
CPU	shares with FSS	Processor Sets
Memory	projects and rcapd	*Memory Sets*
Disk Size	ZFS, SVM soft partitions	Separate Disks or Slices
Disk Throughput		Separate Disks or Controllers
Network	IPQoS	Separate NICs
Swap		*Swap Sets*

Features in italic don't exist yet, but these or similar such features may appear in future updates to Solaris 10.

This is subject to change as developments occur.

Commands 1

The following is a cheatsheet of zone resource control commands.

```
### CPU ###

pooladm -e      # enable pools

pooladm         # print active config from memory, check pools are enabled
pooladm -x      # flush active config in memory
pooladm -s      # save active config from memory to /etc/pooladm.conf
pooladm -c      # activate config from /etc/pooladm.conf to memory

psrset          # print psets and their current CPUs
poolbind        # immediately bind zones to pools

poolcfg -dc info        # print active config from memory
poolcfg -df scriptfile  # read scriptfile config to memory

### Memory ###

rcapadm -E      # enable the rcap daemon
rcapstat        # check rcapd is running, print stats

projadd -K 'rcap.max-rss=bytes' newproject      # add a project
newtask -p newproject command                   # run command in project
```

Commands 2

The following demonstrates some handy tricks for managing resource control configs.

The poolcfg command allows us to configure pools and processor sets. Some example syntax,

```
poolcfg -dc info
poolcfg -dc 'create pset work1-pset'
poolcfg -dc 'create pool work1-pool'
poolcfg -dc 'modify pset work1-pset ( uint pset.min = 2; uint pset.max = 3 )'
poolcfg -dc 'modify pool work1-pool ( string pool.scheduler = "FSS" )'
poolcfg -dc 'modify pool work1-pool ( int pool.importance = 10 )'
```

```
poolcfg -dc 'destroy pool work1-pool'
```

The "-d" option lets us change the memory configuration directly. Using the poolcfg command as above can become confusing, especially if a large configuration requires twenty or so poolcfgs to be typed; you can lose track of where you are and what you've done.

The following is a much easier receipe. We edit a script file called mypools in vi which has a single CPU pool called work1-pool. Then we reload and activate it with the final four commands. The mypools file contains comments including the required commands as a reminder. Remember to have enabled pools first using "pooladm -e",

```
vi mypools

 # mypools - my pool configuration file.
 #
 # load using: pooladm -x; pooladm -s; poolcfg -f mypool; pooladm -c
 #
 create pset work1-pset ( uint pset.min = 1; uint pset.max = 1 )
 create pool work1-pool
 associate pool work1-pool ( pset work1-pset )


pooladm -x                      # flush current in-memory config
pooladm -s                      # blank /etc/pooladm.conf config
poolcfg -f mypool               # save "mypool" config as /etc/pooladm.conf
pooladm -c                      # activate new /etc/pooladm.conf config
```

Great! However there is a catch - the above technique assumes you will then boot, reboot or use poolbind on the zones so that they attach to the pools. If you run this on a server with running zones that are already using pools, then the end result is that all zones will be running in the default_pool. oops! Reboot or use poolbind to reattach the pools (poolbind is demonstrated at the end of Resource CPU 1).

CPU Resource Control Intro

Resource control for CPUs can be achieved in three ways,

- Fixed CPUs per Zone.
- Varying CPUs per Zone.
- CPU Shares per Zone.

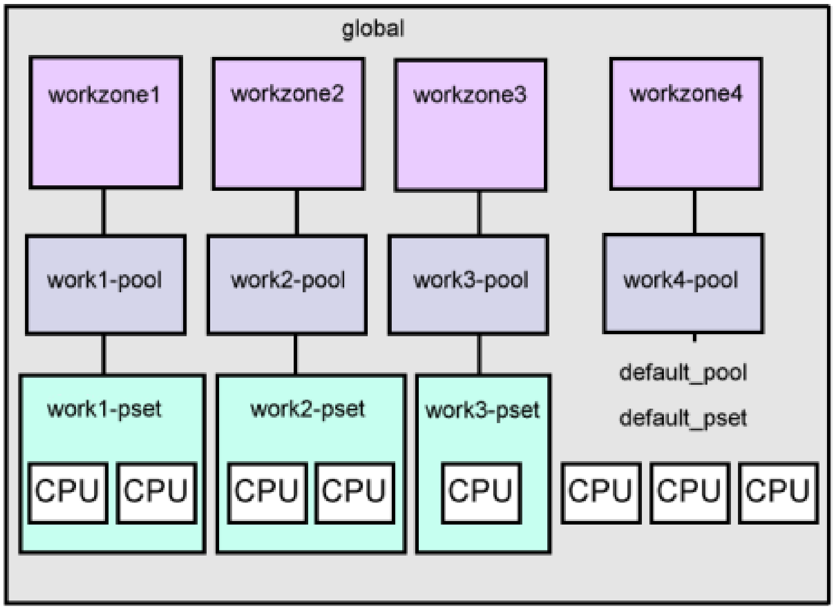
Fixed CPUs
This involves pools and processor sets to set a fixed number of CPUs per zone. One zone may have 2 CPUs, one 4 CPUs, etc. We demo this below. This may be desirable for political reasons - a way departments could share the cost of a new server, or licensing reasons - if a software vendor would only charge for the CPU number in the zone. Another advantage may be better level 1 and level 2 cache hit rates for each zone. A disadvantage would be that zones that were idle would not share CPU resources with zones that had high demand.

Varying CPUs
This involves creating pools that have a min and max number of CPUs, like in the example. The daemon poold can shuffle CPUs from one zone to another as demand varies. This may be used for many of the same reasons as fixed CPUs, on a server with extra CPUs that can move about with the load.

CPU Shares
This makes use of the Fair Share Schedular (FSS), which allows us to allocate a share value to each zone. One zone may have 10 shares, another 20, etc. For the zones that are busy, CPU allocation is based on the ratio of the zone's shares over the total shares of the busy zones. It's advantage is the best division of CPU resources amongst the zones, where shares are the quantum not CPUs, and as such may be the only way to go if you have more zones than CPUs. It's great, and makes a lot of sense.

CPU Resource Control 1

The following is an example of zone resource controls where the zones are configured to use a fixed number of CPUs. The final layout will be like the following,



The steps to make the above happen are,

1. Enable pools
2. Configure pools
3. Load pools
4. Configure zones
5. Reboot zones

1. Enable pools

In the following checks if pools have been enabled (they haven't), then enables pools (pooladm -e), then checks again to show both a new poold is running and that pooladm is now happy,

```
# pgrep -l poold
# pooladm
pooladm: couldn't open pools state file: Facility is not active
#
# pooladm -e
#
# pgrep -l poold
 1429 poold
# pooladm

system titan
        string  system.comment
        int     system.version 1
[...]
```

2. Configure pools

We have a total of 8 CPUs, and our goal for this example is,

- workzone1 - 2 CPUs (work1-pool, work1-pset)
- workzone2 - 2 CPUs (work2-pool, work2-pset)
- workzone3 - 1 CPUs (work3-pool, work3-pset)
- workzone4 - default resources (work4-pool)

workzone4 will not have a processor set and CPUs assigned so that the default action can be observed.

The zones have the names "workzone1"..., and the pools will be "work1-pool"... We'll use a poolcfg script (as explained in commands2), where we create a reusable script to contain all of our pool settings,

```
# myscript - poolcfg script for fixed CPUs.
#
# Load using:
#       pooladm -x; pooladm -s; poolcfg -f myscript; pooladm -c
#
# 14-Jan-2005   Brendan Gregg   Created this.

create pset work1-pset ( uint pset.min = 2; uint pset.max = 2 )
create pset work2-pset ( uint pset.min = 2; uint pset.max = 2 )
create pset work3-pset ( uint pset.min = 1; uint pset.max = 1 )
create pool work1-pool
create pool work2-pool
create pool work3-pool
create pool work4-pool
associate pool work1-pool ( pset work1-pset )
associate pool work2-pool ( pset work2-pset )
associate pool work3-pool ( pset work3-pset )
```

3. Load pools

Now we load the script created above. We'll check something happened by running "psrset" before and after to check if any processor sets have been automatically created,

```
# psrset
#
# pooladm -x; pooladm -s; poolcfg -f myscript; pooladm -c
#
# psrset
user processor set 1: processors 0 1
user processor set 2: processors 2 3
user processor set 3: processor 512
```

So far, so good. Three processor sets have been created which contain 2, 2 and 1 CPUs respectively. This matches the desired allocation of CPUs. (As explained earlier), be careful when rerunning those four commands on a system that already has zones and pools configured, you may need to run through step 5 (below) to reconnect zones and pools).

Now we can run "pooladm" and check the current state in more detail. To help make sense of the following verbose output, I have highlighted settings related to workzone1 in bold (there is both a pool section, and a processor set section),

```
# pooladm

system titan
        string  system.comment
        int     system.version 1
        boolean system.bind-default true
        int     system.poold.pid 247

        pool pool_default
                int     pool.sys_id 0
                boolean pool.active true
                boolean pool.default true
                int     pool.importance 1
                string  pool.comment
                pset    pset_default

        pool work2-pool
                int     pool.sys_id 13
                boolean pool.active true
                boolean pool.default false
                int     pool.importance 1
                string  pool.comment
                pset    work2-pset

        pool work1-pool
                int     pool.sys_id 12
                boolean pool.active true
                boolean pool.default false
                int     pool.importance 1
                string  pool.comment
                pset    work1-pset

        pool work4-pool
                int     pool.sys_id 15
                boolean pool.active true
                boolean pool.default false
                int     pool.importance 1
                string  pool.comment
                pset    pset_default

        pool work3-pool
                int     pool.sys_id 14
```

```
                boolean pool.active true
                boolean pool.default false
                int     pool.importance 1
                string  pool.comment
                pset    work3-pset

        pset work1-pset
                int     pset.sys_id 1
                boolean pset.default false
                uint    pset.min 2
                uint    pset.max 2
                string  pset.units population
                uint    pset.load 0
                uint    pset.size 2
                string  pset.comment

                cpu
                        int     cpu.sys_id 1
                        string  cpu.comment
                        string  cpu.status on-line

                cpu
                        int     cpu.sys_id 0
                        string  cpu.comment
                        string  cpu.status on-line

        pset work3-pset
                int     pset.sys_id 3
                boolean pset.default false
                uint    pset.min 1
                uint    pset.max 1
                string  pset.units population
                uint    pset.load 0
                uint    pset.size 1
                string  pset.comment

                cpu
                        int     cpu.sys_id 512
                        string  cpu.comment
                        string  cpu.status on-line

        pset work2-pset
                int     pset.sys_id 2
                boolean pset.default false
                uint    pset.min 2
                uint    pset.max 2
                string  pset.units population
                uint    pset.load 0
                uint    pset.size 2
                string  pset.comment

                cpu
                        int     cpu.sys_id 3
                        string  cpu.comment
                        string  cpu.status on-line

                cpu
                        int     cpu.sys_id 2
                        string  cpu.comment
                        string  cpu.status on-line

        pset pset_default
                int     pset.sys_id -1
                boolean pset.default true
                uint    pset.min 1
                uint    pset.max 65536
                string  pset.units population
                uint    pset.load 5
                uint    pset.size 3
                string  pset.comment

                cpu
                        int     cpu.sys_id 515
                        string  cpu.comment
                        string  cpu.status on-line

                cpu
                        int     cpu.sys_id 514
                        string  cpu.comment
                        string  cpu.status on-line

                cpu
                        int     cpu.sys_id 513
                        string  cpu.comment
                        string  cpu.status on-line
```

We can see that the work4-pool has been associated with the default processor set.

4. Configure zones

Here are some zones I prepared earlier (see small zone or big zone for setup examples),

```
# zoneadm list -cv
  ID NAME             STATUS         PATH
   0 global           running        /
   1 workzone4        running        /export/workzone4
   2 workzone3        running        /export/workzone3
   3 workzone2        running        /export/workzone2
   4 workzone1        running        /export/workzone1
#
```

We need to connect our zones to the pools that we have just created. This is configured using the zonecfg command.

```
# zonecfg -z workzone1
zonecfg:workzone1> set pool=work1-pool
zonecfg:workzone1> verify
zonecfg:workzone1> exit
#
# zonecfg -z workzone2
zonecfg:workzone2> set pool=work2-pool
zonecfg:workzone2> verify
zonecfg:workzone2> exit
#
# zonecfg -z workzone3
zonecfg:workzone3> set pool=work3-pool
zonecfg:workzone3> verify
zonecfg:workzone3> exit
#
# zonecfg -z workzone4
zonecfg:workzone4> set pool=work4-pool
zonecfg:workzone4> verify
zonecfg:workzone4> exit
#
```

5. Reboot zones

Finally the zones are rebooted so that they bind to the new pools.

```
# zlogin workzone1 init 6
# zlogin workzone2 init 6
# zlogin workzone3 init 6
# zlogin workzone4 init 6
#
```

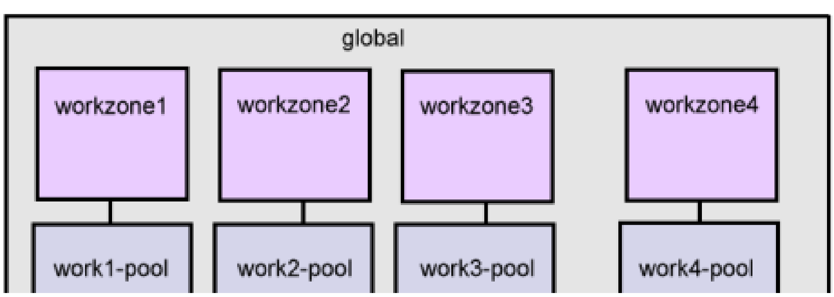
... Instead of rebooting, the poolbind command could be used to bind all process in each of the zones to the resources associated with each pool. Here I demonstrate workzone1,

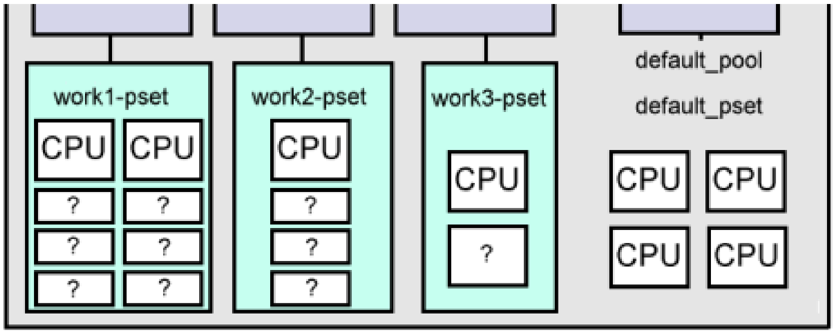
```
# poolbind -p work1-pool -i zoneid workzone1
```

and so on for the other non-global zones...

CPU Resource Control 2

The following is an example of zone resource controls where the zones are configured to use a varying number of CPUs controlled by dynamic reconfiguration. The final layout will be like the following,





The steps to make the above happen are,

1. Enable pools
2. Configure pools
3. Load pools
4. Configure zones
5. Reboot zones

1. Enable pools

In the following checks if pools have been enabled. In this case they haven’t, so we enable pools (pooladm -e), then check again to show both a new poold is running and that pooladm is now happy,

```
# pgrep -l poold
# pooladm
pooladm: couldn’t open pools state file: Facility is not active
#
# pooladm -e
#
# pgrep -l poold
 1429 poold
# pooladm

system titan
        string  system.comment
        int     system.version 1
[...]
```

2. Configure pools

The goal for this 8 CPU server is,

- workzone1 - 2 to 8 CPUs (work1-pool, work1-pset)
- workzone2 - 1 to 4 CPUs (work2-pool, work2-pset)
- workzone3 - 1 to 2 CPUs (work3-pool, work3-pset)
- workzone4 - default resources (work4-pool)

So that the number of CPUs used by zones 1 to 3 will vary from a total of 4 to 14. See the poold man page to documentation on when poold will reshuffle CPU assignments - which includes CPU DR. The CPU numbers can be rearranged manually, using poolcfg transfer.
The poolcfg script is,

```
# myscript - poolcfg script for varying CPUs.
#
# Load using:
#       pooladm -x; pooladm -s; poolcfg -f myscript; pooladm -c
#
# 14-Jan-2005   Brendan Gregg   Created this.

create pset work1-pset ( uint pset.min = 2; uint pset.max = 8 )
create pset work2-pset ( uint pset.min = 1; uint pset.max = 4 )
create pset work3-pset ( uint pset.min = 1; uint pset.max = 2 )
create pool work1-pool
create pool work2-pool
create pool work3-pool
create pool work4-pool
```

```
associate pool work1-pool ( pset work1-pset )
associate pool work2-pool ( pset work2-pset )
associate pool work3-pool ( pset work3-pset )
```

3. Load pools

Now we load it, then check if something happened,

```
# pooladm -x; pooladm -s; poolcfg -f myscript; pooladm -c
#
# psrset
user processor set 1: processors 0 1 2
user processor set 2: processors 513 514
user processor set 3: processors 3 512
```

(See previous comments about running those four commands).

And to check it properly (workzone1 settings have been highlighted to serv as an example),

```
# pooladm

system titan
        string  system.comment
        int     system.version 1
        boolean system.bind-default true
        int     system.poold.pid 247

        pool work2-pool
                int     pool.sys_id 61
                boolean pool.active true
                boolean pool.default false
                int     pool.importance 1
                string  pool.comment
                pset    work2-pset

        pool work1-pool
                int     pool.sys_id 60
                boolean pool.active true
                boolean pool.default false
                int     pool.importance 1
                string  pool.comment
                pset    work1-pset

        pool work4-pool
                int     pool.sys_id 63
                boolean pool.active true
                boolean pool.default false
                int     pool.importance 1
                string  pool.comment
                pset    pset_default

        pool work3-pool
                int     pool.sys_id 62
                boolean pool.active true
                boolean pool.default false
                int     pool.importance 1
                string  pool.comment
                pset    work3-pset

        pool pool_default
                int     pool.sys_id 0
                boolean pool.active true
                boolean pool.default true
                int     pool.importance 1
                string  pool.comment
                pset    pset_default

        pset work1-pset
                int     pset.sys_id 1
                boolean pset.default false
                uint    pset.min 2
                uint    pset.max 8
                string  pset.units population
                uint    pset.load 29
                uint    pset.size 3
                string  pset.comment

                cpu
                        int     cpu.sys_id 1
                        string  cpu.comment
                        string  cpu.status on-line

                cpu
                        int     cpu.sys_id 0
                        string  cpu.comment
                        string  cpu.status on-line

                cpu
```

```
                        int     cpu.sys_id 2
                        string  cpu.comment
                        string  cpu.status on-line

        pset work3-pset
                int     pset.sys_id 3
                boolean pset.default false
                uint    pset.min 1
                uint    pset.max 2
                string  pset.units population
                uint    pset.load 22
                uint    pset.size 2
                string  pset.comment

                cpu
                        int     cpu.sys_id 3
                        string  cpu.comment
                        string  cpu.status on-line

                cpu
                        int     cpu.sys_id 512
                        string  cpu.comment
                        string  cpu.status on-line

        pset work2-pset
                int     pset.sys_id 2
                boolean pset.default false
                uint    pset.min 1
                uint    pset.max 4
                string  pset.units population
                uint    pset.load 35
                uint    pset.size 2
                string  pset.comment

                cpu
                        int     cpu.sys_id 514
                        string  cpu.comment
                        string  cpu.status on-line

                cpu
                        int     cpu.sys_id 513
                        string  cpu.comment
                        string  cpu.status on-line

        pset pset_default
                int     pset.sys_id -1
                boolean pset.default true
                uint    pset.min 1
                uint    pset.max 65536
                string  pset.units population
                uint    pset.load 14040
                uint    pset.size 1
                string  pset.comment

                cpu
                        int     cpu.sys_id 515
                        string  cpu.comment
                        string  cpu.status on-line
```

poold has choosen how to divide the CPU resources when the "pooladm -c" command was executed, and assigned CPUs to processor sets guided by the min and max values. The poolcfg transfer command can be used to manually move CPU assignments around, limited by the min and max values.

4. Configure zones

We need to connect our zones to the pools that we have just created. This is configured using the zonecfg command, in exactly the same was as in the first example.

```
# zonecfg -z workzone1
zonecfg:workzone1> set pool=work1-pool
zonecfg:workzone1> verify
zonecfg:workzone1> exit
#
[...]
```

5. Configure zones

Just like in the first example, our final step is to reboot the non-global zones, or to use poolbind.

CPU reallocation

The above has set max and min values for the number of CPUs in a zone. When do they get shuffled around?

My current understanding is this (subject to updates): CPUs will be reallocated by poold in response to the following events

- a DR (Dynamic Reconfiguration) event. eg, a system board goes offline.
- when pooladm -c (from step 3 above) is run.
- when a "poolcfg -dc transfer" command is used (see examples in man poolcfg).
- when poold attempts to meet "Objectives" (below).

check this website for updates.

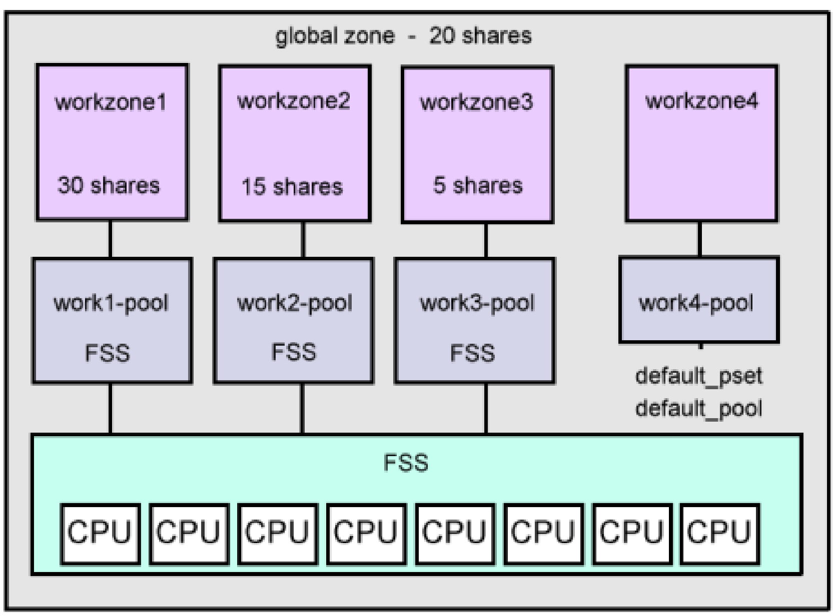
Objectives

The above demonstration is a fairly basic configuration. There is quite a lot more we can do with varying CPUs per zone, based on rules or "objectives" that are configured per pool. These objectives may be to ensure zones have a certain level of utilisation or load. "man libpool" may be a starting point to learn about objectives (and check this website for updates).

CPU Resource Control 3

The following is an example of zone resource controls where the zones are configured to use the Fair Share Scheduler. This is capable of sharing the CPUs much more efficiently than whole CPU assignments, and uses a zone.cpu-shares to determine CPU allocation.

The final layout will be like the following,



The steps to make the above happen are,

1. Enable pools
2. Configure pools
3. Load pools
4. Change system schedular to FSS
5. Configure zones
6. Reboot zones
7. Test FSS

1. Enable pools

In the following checks if pools have been enabled. In this case they haven't, so we enable pools (pooladm -e), then check again to show both a new poold is running and that pooladm is now happy,

```
# pgrep -l poold
# pooladm
pooladm: couldn't open pools state file: Facility is not active
#
# pooladm -e
#
# pgrep -l poold
 1429 poold
# pooladm

system titan
        string  system.comment
        int     system.version 1
[...]
```

2. Configure pools

This time we dive straight into the pool creation, and will configure share numbers later. We use the following poolcfg script,

```
# myscript - poolcfg script for varying CPUs.
#
# Load using:
#       pooladm -x; pooladm -s; poolcfg -f myscript; pooladm -c
#
# 14-Jan-2005   Brendan Gregg   Created this.

create pool work1-pool ( string pool.scheduler = "FSS" )
create pool work2-pool ( string pool.scheduler = "FSS" )
create pool work3-pool ( string pool.scheduler = "FSS" )
create pool work4-pool
```

3. Load pools

We load the above script, then check if something happened,

```
# pooladm -x; pooladm -s; poolcfg -f myscript; pooladm -c
#
# psrset
#
```

(See previous comments about running those four commands).

No processor sets are created as we are using FSS now.

The following is the full listing, with workzone1 settings highlighted as an example,

```
# pooladm

system titan
        string  system.comment
        int     system.version 1
        boolean system.bind-default true
        int     system.poold.pid 220

        pool work4-pool
                int     pool.sys_id 4
                boolean pool.active true
                boolean pool.default false
                int     pool.importance 1
                string  pool.comment
                pset    pset_default

        pool work1-pool
                int     pool.sys_id 1
                boolean pool.active true
                boolean pool.default false
                string  pool.scheduler FSS
                int     pool.importance 1
                string  pool.comment
                pset    pset_default

        pool pool_default
                int     pool.sys_id 0
```

```
                boolean pool.active true
                boolean pool.default true
                int     pool.importance 1
                string  pool.comment
                pset    pset_default

        pool work3-pool
                int     pool.sys_id 3
                boolean pool.active true
                boolean pool.default false
                string  pool.scheduler FSS
                int     pool.importance 1
                string  pool.comment
                pset    pset_default

        pool work2-pool
                int     pool.sys_id 2
                boolean pool.active true
                boolean pool.default false
                string  pool.scheduler FSS
                int     pool.importance 1
                string  pool.comment
                pset    pset_default

        pset pset_default
                int     pset.sys_id -1
                boolean pset.default true
                uint    pset.min 1
                uint    pset.max 65536
                string  pset.units population
                uint    pset.load 48155
                uint    pset.size 8
                string  pset.comment

                cpu
                        int     cpu.sys_id 1
                        string  cpu.comment
                        string  cpu.status on-line

                cpu
                        int     cpu.sys_id 0
                        string  cpu.comment
                        string  cpu.status on-line

                cpu
                        int     cpu.sys_id 3
                        string  cpu.comment
                        string  cpu.status on-line

                cpu
                        int     cpu.sys_id 2
                        string  cpu.comment
                        string  cpu.status on-line

                cpu
                        int     cpu.sys_id 515
                        string  cpu.comment
                        string  cpu.status on-line

                cpu
                        int     cpu.sys_id 514
                        string  cpu.comment
                        string  cpu.status on-line

                cpu
                        int     cpu.sys_id 513
                        string  cpu.comment
                        string  cpu.status on-line

                cpu
                        int     cpu.sys_id 512
                        string  cpu.comment
                        string  cpu.status on-line
```

That's all - only a pool. The details (number of shares) are configured in step 5.

4. Change system schedular to FSS

For FSS to work well, the entire system should be placed under FSS control,

```
# dispadmin -d FSS
# init 6

(system reboots)

# ps -efc
     UID   PID  PPID  CLS PRI    STIME TTY         TIME CMD
    root     0     0  SYS  96 05:25:38 ?           0:14 sched
    root     1     0  FSS  29 05:25:38 ?           0:00 /sbin/init
    root     2     0  SYS  98 05:25:38 ?           0:00 pageout
```

```
    root     3     0  SYS  60 05:25:38 ?           0:00 fsflush
    root   334     1  FSS  59 05:25:53 ?           0:00 /usr/sadm/lib/smc/bin/smcboot
    root     7     1  FSS  29 05:25:40 ?           0:02 /lib/svc/bin/svc.startd
    root     9     1  FSS  29 05:25:40 ?           0:08 svc.configd
    root   162     1  FSS  29 05:25:46 ?           0:00 /usr/lib/ssh/sshd
    root    92     1  FSS  29 05:25:44 ?           0:00 /usr/sbin/nscd
  daemon   142     1  FSS  29 05:25:46 ?           0:00 /usr/sbin/rpcbind
    root    94     1  FSS  59 05:25:44 ?           0:01 /usr/lib/picl/picld
  daemon    93     1  FSS  29 05:25:44 ?           0:00 /usr/lib/crypto/kcfd
  daemon   148     1  FSS  29 05:25:46 ?           0:00 /usr/lib/nfs/nfs4cbd
  daemon   150     1  FSS  29 05:25:46 ?           0:00 /usr/lib/nfs/statd
    root   189   184  FSS  29 05:25:47 ?           0:00 /usr/lib/saf/ttymon
  daemon   149     1  FSS  29 05:25:46 ?           0:00 /usr/lib/nfs/nfsmapid
    root   172     1  FSS  29 05:25:46 ?           0:00 /usr/lib/utmpd
[...]
```

Note the system deamons are now running in FSS.

A "/etc/init.d/sysetup" can be used instead of the reboot, as it changes the process scheduling classes live (and is what runs at boot anyway).

5. Configure zones

The following goal for this 8 CPU servers is,

- workzone1 - 30 shares = 60% (work1-pool)
- workzone2 - 15 shares = 30% (work2-pool)
- workzone3 - 5 shares = 10% (work3-pool)
- workzone4 - default resources (work4-pool)

So when zones 1, 2 and 3 are all wanting CPU: workzone1 uses 60% of the CPU capacity, workzone2 uses 30%, and workzone3 can use 10% of the CPU capacity. (this assumes the default pool is not busy - which includes workzone4 and global)

As in the first example, the zones are associated with the pool names. However this time a resource control is added that contains the value of the CPU shares,

```
# zonecfg -z workzone1
zonecfg:workzone1> set pool=work1-pool
zonecfg:workzone1> add rctl
zonecfg:workzone1:rctl> set name=zone.cpu-shares
zonecfg:workzone1:rctl> add value (priv=privileged,limit=30,action=none)
zonecfg:workzone1:rctl> end
zonecfg:workzone1> verify
zonecfg:workzone1> exit
#
# zonecfg -z workzone2
zonecfg:workzone2> set pool=work2-pool
zonecfg:workzone2> add rctl
zonecfg:workzone2:rctl> set name=zone.cpu-shares
zonecfg:workzone2:rctl> add value (priv=privileged,limit=15,action=none)
zonecfg:workzone2:rctl> end
zonecfg:workzone2> verify
zonecfg:workzone2> exit
#
# zonecfg -z workzone3
zonecfg:workzone3> set pool=work3-pool
zonecfg:workzone3> add rctl
zonecfg:workzone3:rctl> set name=zone.cpu-shares
zonecfg:workzone3:rctl> add value (priv=privileged,limit=5,action=none)
zonecfg:workzone3:rctl> end
zonecfg:workzone3> verify
zonecfg:workzone3> exit
#
```

6. Reboot zones

The non-global zones can now be rebooted to use the new pools (this time it’s a little more complex to achieve without rebooting the non-global zones - you would need to use both poolbind and prctl).

```
# zlogin workzone1 init 6
# zlogin workzone2 init 6
# zlogin workzone3 init 6
```

```
# zlogin workzone4 init 6
```

We can also check our FSS shares for each zone using the zonefss script,

```
# zonefss -l
  ID NAME             SHARES
   0 global           1
   0 workzone1        30
   0 workzone2        15
   0 workzone3        5
   0 workzone4        1
```

Looks good.

7. Test FSS

Now the real fun begins; we load zones 1, 2 and 3 by running the following CPU hungry program in each,

```
# cat -n cpuhog.pl
     1  #!/usr/bin/perl
     2
     3  print "eating the CPUs\n";
     4
     5  foreach $i (1..16) {
     6          $pid = fork();
     7          last if $pid == 0;
     8          print "created PID $pid\n";
     9  }
    10
    11  while (1) {
    12          $x++;
    13  }
    14
```

And now we can see how well sharing is performing by running "prstat -Z",

```
# prstat -Z
   PID USERNAME  SIZE   RSS STATE  PRI NICE      TIME  CPU PROCESS/NLWP
  2008 root     4000K 1168K cpu513  28    0   0:02:11 3.7% cpuhog.pl/1
  2018 root     4000K 1168K cpu1    32    0   0:02:11 3.7% cpuhog.pl/1
  2015 root     4000K 1168K cpu515  30    0   0:02:13 3.6% cpuhog.pl/1
  2020 root     4000K 1168K cpu3    29    0   0:02:13 3.6% cpuhog.pl/1
  2010 root     4000K 1168K run     17    0   0:02:11 3.5% cpuhog.pl/1
  2013 root     4000K 1168K run     28    0   0:02:11 3.5% cpuhog.pl/1
  2005 root     4008K 2320K run      8    0   0:02:11 3.5% cpuhog.pl/1
  2014 root     4000K 1168K cpu0    30    0   0:02:11 3.5% cpuhog.pl/1
  2007 root     4000K 1168K run     20    0   0:02:11 3.5% cpuhog.pl/1
  2016 root     4000K 1168K cpu512  28    0   0:02:12 3.5% cpuhog.pl/1
  2021 root     4000K 1168K run     17    0   0:02:11 3.4% cpuhog.pl/1
  2009 root     4000K 1168K run     14    0   0:02:14 3.3% cpuhog.pl/1
  2012 root     4000K 1168K run     16    0   0:02:08 3.3% cpuhog.pl/1
  2006 root     4000K 1304K run     18    0   0:02:13 3.3% cpuhog.pl/1
  2017 root     4000K 1168K run     25    0   0:02:10 3.3% cpuhog.pl/1
ZONEID    NPROC  SIZE   RSS MEMORY      TIME  CPU ZONE
     2       51  182M   93M   0.5%   0:37:27  59% workzone1
     4       51  182M   92M   0.5%   0:16:25  30% workzone2
     3       51  183M   93M   0.5%   0:16:30  10% workzone3
     0       61  359M  194M   1.1%   0:00:11 0.1% global
     1       34  116M   72M   0.4%   0:00:12 0.0% workzone4
Total: 248 processes, 659 lwps, load averages: 51.19, 40.28, 20.52
```

Brilliant! The division between zones 1, 2 and 3 is 59%, 30% and 10% - matching perfectly our desired 60%, 30% and 10% requested by selecting the zone.cpu-shares values.

The following demo is also worth noting, here I kill the cpuhog.pl program in zones 1 and 2, and leave it running in 3,

```
# prstat -Z
   PID USERNAME  SIZE   RSS STATE  PRI NICE      TIME  CPU PROCESS/NLWP
  1980 root     4000K 1168K cpu513  49    0   0:02:36 7.1% cpuhog.pl/1
  1972 root     4000K 1168K cpu3    49    0   0:02:26 7.1% cpuhog.pl/1
  1968 root     4000K 1168K cpu1    46    0   0:02:37 7.1% cpuhog.pl/1
  1977 root     4000K 1168K run     13    0   0:02:35 7.1% cpuhog.pl/1
  1979 root     4000K 1168K cpu2    49    0   0:02:34 6.9% cpuhog.pl/1
  1967 root     4000K 1168K run      7    0   0:02:30 6.5% cpuhog.pl/1
```

```
  1976 root      4000K 1168K cpu0    46    0   0:02:32 6.3% cpuhog.pl/1
  1969 root      4000K 1168K run     46    0   0:02:18 6.0% cpuhog.pl/1
  1970 root      4000K 1168K run      1    0   0:02:29 5.4% cpuhog.pl/1
  1974 root      4000K 1168K run     13    0   0:02:16 5.3% cpuhog.pl/1
  1975 root      4000K 1168K cpu512  52    0   0:02:21 5.3% cpuhog.pl/1
  1971 root      4000K 1168K run     45    0   0:02:21 5.1% cpuhog.pl/1
  1981 root      4000K 1168K run     13    0   0:02:29 5.0% cpuhog.pl/1
  1966 root      4000K 1304K run      7    0   0:02:18 4.9% cpuhog.pl/1
  1973 root      4000K 1168K cpu514  49    0   0:02:27 4.9% cpuhog.pl/1
ZONEID    NPROC  SIZE   RSS MEMORY      TIME  CPU ZONE
     3       51  183M   93M   0.5%   0:41:33  99% workzone3
     0       61  359M  194M   1.1%   0:00:12 0.1% global
     1       34  116M   72M   0.4%   0:00:12 0.0% workzone4
     2       34  116M   72M   0.4%   0:00:11 0.0% workzone1
     4       34  116M   71M   0.4%   0:00:10 0.0% workzone2
Total: 214 processes, 624 lwps, load averages: 20.29, 35.61, 27.58
```

With no other FSS zones competing for the CPUs, the fair share scheduler allows workzone3 to use 100% (near enough!) of the CPU resources. This is exactly the behaviour that is desirable.

Last but not least, we need to address the number of shares allocated to the global zone. By default the global zone only gets one share, we probably want it to have plenty of shares for urgent administration - and to assume it will not normally use them. Here we check the current value is 1, then set it to be 20,

```
# prctl -n zone.cpu-shares -i zone global
zone: 0: global
NAME    PRIVILEGE       VALUE    FLAG   ACTION                       RECIPIENT
zone.cpu-shares
        privileged          1       -   none                                 -
        system          65.5K     max   none                                 -
#
# prctl -n zone.cpu-shares -v 20 -r -i zone global
#
# prctl -n zone.cpu-shares -i zone global
zone: 0: global
NAME    PRIVILEGE       VALUE    FLAG   ACTION                       RECIPIENT
zone.cpu-shares
        privileged         20       -   none                                 -
        system          65.5K     max   none                                 -
```

Ok, so that's changed the value on a live system. The Zone guide on docs.sun.com, Chapter 26, recommends that this must be done after every reboot. I guess that means we add a script to SMF.

Memory Resource Control Intro

Resource control for Memory can be achieved in at least these ways,

- Projects for applications.
- Modifying the system project.

All of these techniques use the memory resource cap daemon "rcapd", which sends pages of a process that exceeds it's RSS limit to the physical swap device. On servers where you use rcapd memory resource control, you'll want plenty of swap available. Also, I should note that this technique isn't tied to zonecfg, it's using standard Solaris commands.

While the following techniques do work, they are not based on any best practices document from Sun - they are methods that I have tried on Solaris 10 FCS that work. It is possible that in the future Sun will release a document that details other ways to do zone memory caps (and quite possibly more elegant ways than these techniques), so keep an eye out.

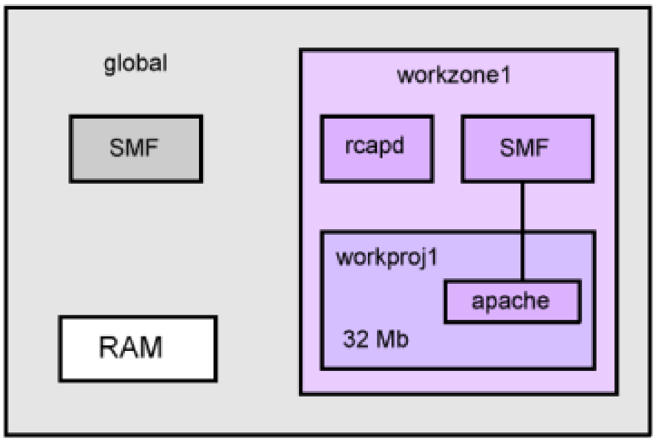
Projects for applications
Here we create new projects to run our applications in. We may have several projects within a zone, or just one project for zone applications, eg "workproj1". Applications run under the project by changing how they are invoked using svccfg and prefixing their start method with a newtask. If the application runs under a unique username, it may be possible to implement a similar memory control by creating a user project. See example.

Modifying the system project
Applications and daemons started from svc.startd, which is likely to be everything, will begin in the "system"

project. Here we modify the system project so that it has memory limits. See [example](#).

Memory Resource Control 1

This demonstrates memory resource control for applications running in a non-global zone. The non-global zone for this demonstration is "workzone1". A project will be created to cap memory, "workproj1", and the Solaris Management Framework (SMF) will be configured to run apache within this project. The final layout will be like the following,



We begin by ensuring the nsswitch.conf file is of the correct style, then we enable the memory resource cap daemon "rcapd" by using "rcapadm -E".

```
workzone1# cp /etc/nsswitch.files /etc/nsswitch.conf
workzone1#
workzone1# ps -ef | grep rcapd
    root 21781 21738   0 21:43:35 pts/3         0:00 grep rcapd
workzone1#
workzone1# rcapadm -E
workzone1#
workzone1# ps -ef | grep rcapd
    root 21786 21738   0 21:45:15 pts/3         0:00 grep rcapd
  daemon 21784 20584   0 21:45:13 ?             0:00 /usr/lib/rcap/rcapd
workzone1#
```

If "rcapadm -E" fails to start rcapd, you may need to use the Solaris 10 services administration command to get things started - "svcadm disable rcap" then "rcapadm -E".

Now we add a project for our applications to run in. The new project is called "workproj1", and has a memory cap of 32Mb RSS - set via the "rcap.max-rss" property. We check if workproj1 has been picked up by rcapd by using the rcapstat command, and bounce rcap so that it is,

```
workzone1# projadd -K 'rcap.max-rss=32Mb' workproj1
workzone1# cat /etc/project
system:0::::
user.root:1::::
noproject:2::::
default:3::::
group.staff:10::::
workproj1:100::::rcap.max-rss=33554432
workzone1#
workzone1# rcapstat
    id project         nproc     vm    rss    cap    at  avgat    pg  avgpg
    id project         nproc     vm    rss    cap    at  avgat    pg  avgpg
^C
workzone1# svcadm restart rcap
workzone1# rcapstat
    id project         nproc     vm    rss    cap    at  avgat    pg  avgpg
   100 workproj1           0     0K     0K    32M    0K     0K    0K     0K
   100 workproj1           0     0K     0K    32M    0K     0K    0K     0K
^C
```

Now we configure our application to run under the project workproj1. The application we will demonstrate will be apache2. We begin by enabling apache, and checking that it is running under the usual project (system). Then it is configured to run under workproj1 by using the Solaris 10 service configuration command "svccfg",

```
workzone1# cd /etc/apache2
workzone1# cp httpd.conf-example httpd.conf
workzone1# svcadm enable http
workzone1# ps -eo user,project,args | grep httpd
webservd   system /usr/apache2/bin/httpd -k start
webservd   system /usr/apache2/bin/httpd -k start
webservd   system /usr/apache2/bin/httpd -k start
    root   system /usr/apache2/bin/httpd -k start
webservd   system /usr/apache2/bin/httpd -k start
workzone1#
workzone1# svccfg
svc:> select network/http:apache2
svc:/network/http:apache2> listprop start/exec
start/exec  astring  "/lib/svc/method/http-apache2 start"
svc:/network/http:apache2>
svc:/network/http:apache2> setprop start/exec = astring: continued,
 "/usr/bin/newtask -p workproj1 /lib/svc/method/http-apache2 start"
svc:/network/http:apache2> listprop start/exec
start/exec  astring  "/usr/bin/newtask -p workproj1 /lib/svc/method/http-apache2 start"
svc:/network/http:apache2>
svc:/network/http:apache2> end
workzone1#
workzone1# svcadm refresh http
workzone1# svcadm restart http
workzone1# ps -eo user,project,args | grep httpd
webservd workproj1 /usr/apache2/bin/httpd -k start
webservd workproj1 /usr/apache2/bin/httpd -k start
    root user.root grep httpd
webservd workproj1 /usr/apache2/bin/httpd -k start
webservd workproj1 /usr/apache2/bin/httpd -k start
    root workproj1 /usr/apache2/bin/httpd -k start
webservd workproj1 /usr/apache2/bin/httpd -k start
```

We test this by creating http traffic, and checking if the size of apache is limited to 32Mb of RSS. Traffic is created by running many recursize wgets in another terminal,

```
[term1]
workzone1# cd /tmp
workzone1# /usr/sfw/bin/wget -q -r http://localhost &
workzone1# /usr/sfw/bin/wget -q -r http://localhost &
workzone1# /usr/sfw/bin/wget -q -r http://localhost &
[term2]
workzone1# rcapstat
    id project         nproc    vm   rss   cap    at  avgat    pg  avgpg
   100 workproj1           6   42M   11M   32M    0K     0K    0K     0K
   100 workproj1           6   42M   12M   32M    0K     0K    0K     0K
   100 workproj1           7   49M   15M   32M    0K     0K    0K     0K
   100 workproj1           7   49M   16M   32M    0K     0K    0K     0K
   100 workproj1           7   49M   17M   32M    0K     0K    0K     0K
   100 workproj1           7   49M   21M   32M    0K     0K    0K     0K
   100 workproj1           7   50M   22M   32M    0K     0K    0K     0K
   100 workproj1          17  120M   56M   32M  296K     0K  280K     0K
   100 workproj1          17  120M   29M   32M   74M    74M   12M    12M
   100 workproj1          16  113M   29M   32M    0K     0K    0K     0K
   100 workproj1          16  113M   29M   32M    0K     0K    0K     0K
workzone1#
workzone1# prstat -n8 -J
   PID USERNAME  SIZE   RSS STATE  PRI NICE      TIME  CPU PROCESS/NLWP
  7318 root     4888K 3680K cpu513  20    0   0:00:03 1.9% wget/1
  7319 root     4888K 3680K cpu1    20    0   0:00:03 1.8% wget/1
  7320 root     4888K 3680K sleep   20    0   0:00:03 1.8% wget/1
  7322 root     4752K 3864K sleep   10    0   0:00:02 1.5% wget/1
  7323 root     4784K 4376K cpu514  30    0   0:00:02 1.5% wget/1
  7288 webservd 7256K 2944K sleep   33    0   0:00:04 0.5% httpd/1
  7283 webservd 7256K 1904K sleep   33    0   0:00:06 0.5% httpd/1
  7278 webservd 7256K 1864K sleep   33    0   0:00:06 0.5% httpd/1
PROJID    NPROC  SIZE   RSS MEMORY      TIME  CPU PROJECT
   100       11  107M   29M   0.1%   0:00:43 8.6% workproj1
     1       10   15M 9928K   0.1%   0:00:13 4.5% user.root
     0       36  131M   56M   0.3%   0:00:10 0.0% system
Total: 52 processes, 98 lwps, load averages: 0.11, 0.23, 0.20
```

Looks good - the project workproj1 has been kept at 29Mb RSS, while the virtual memory size has grown. The excess memory has been paged out to the swap device.

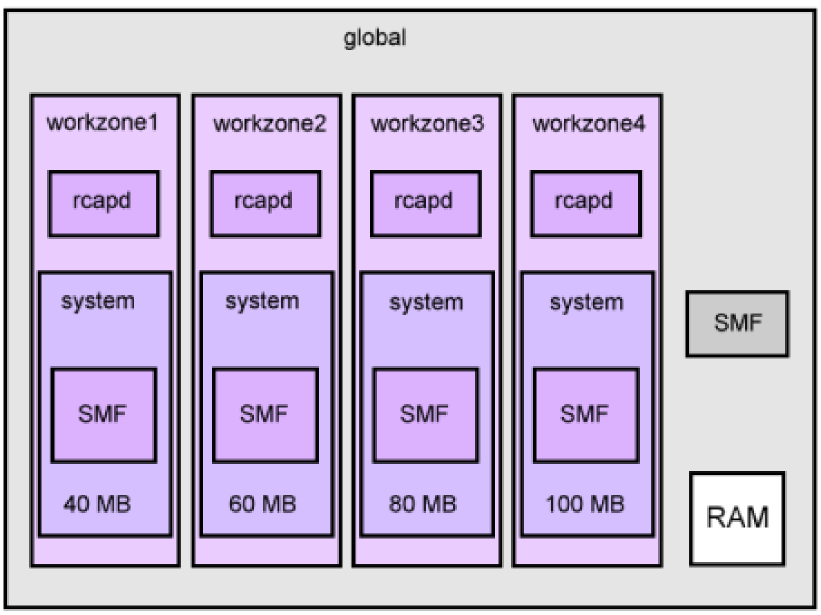
There is an easier way to do this, that sometimes works (depending on the application). For applications that run under a unique username, a memory cap can be applied directly to that user. For example, the following could be attempted for Apache,

```
# rcapadm -E
#
# projadd -K 'rcap.max-rss=32Mb' user.webservd
#
# cat /etc/project
system:0::::
user.root:1::::
noproject:2::::
default:3::::
group.staff:10::::
user.webservd:100::::rcap.max-rss=33554432
#
```

Now processes owned by webservd will be capped at 32Mb. Unfortunately, this doesn't work for Apache. But, since it's easier than modifying the SMF start methods it's worth a try.

Memory Resource Control 2

This demonstrates memory resource control Zones by changing their "system" project. The system start services managed by the Solaris Management Framework (SMF) usually begin under the system project, so applications managed by SMF can be managed by capping the system project. The final layout will be like the following,



We start the resource cap daemon in four non-global zones: workzone1, workzone2, workzone3 and workzone4,

```
workzone1# cp /etc/nsswitch.files /etc/nsswitch.conf
workzone1#
workzone1# ps -ef | grep rcapd
    root 21781 21738   0 21:43:35 pts/3         0:00 grep rcapd
workzone1#
workzone1# rcapadm -E
workzone1#
workzone1# ps -ef | grep rcapd
    root 21786 21738   0 21:45:15 pts/3         0:00 grep rcapd
  daemon 21784 20584   0 21:45:13 ?             0:00 /usr/lib/rcap/rcapd
workzone1#
[repeated in workzone2]
[repeated in workzone3]
[repeated in workzone4]
```

In each zone, we modify the memory cap of the "system" project. The goal for this example is,

- workzone1 - 40Mb RSS
- workzone2 - 60Mb RSS
- workzone3 - 80Mb RSS
- workzone4 - 100Mb RSS

Keep an eye on the prompt as I change between zones,

```
workzone1# projmod -K 'rcap.max-rss=40000000' system
workzone1# cat /etc/project
system:0::::rcap.max-rss=40000000
user.root:1::::
noproject:2::::
default:3::::
group.staff:10::::
workproj1:100::::
workzone1#
workzone2# projmod -K 'rcap.max-rss=60000000' system
workzone3# projmod -K 'rcap.max-rss=80000000' system
workzone4# projmod -K 'rcap.max-rss=100000000' system
```

It's important to now check that these changes have been noticed in each zone, as rcapd only rereads /etc/project every 60 seconds (tunable, see rcapadm). rcapstat can be used to check that the changes have worked. It may require bouncing rcapd as described in the first example.

Now I create memory load on all the non-global zones by running a perl program in each that eats memory. We monitor the memory caps by running prstat in the global zone,

```
global# prstat -n16 -Z
   PID USERNAME  SIZE   RSS STATE  PRI NICE      TIME  CPU PROCESS/NLWP
 14206 daemon   4128K 2344K sleep   59    0   0:00:00 0.1% rcapd/1
 13331 daemon   3936K 2160K sleep   49    0   0:00:00 0.1% rcapd/1
 13502 daemon   4704K 2936K sleep   59    0   0:00:01 0.0% rcapd/1
 15030 root       13M   11M sleep   59    0   0:00:00 0.0% mem10.pl/1
 15027 root       13M   11M sleep   59    0   0:00:00 0.0% mem10.pl/1
 14444 daemon   4896K 3024K sleep   59    0   0:00:01 0.0% rcapd/1
 15031 root     4808K 4448K cpu514  49    0   0:00:00 0.0% prstat/1
   169 daemon   2608K  544K sleep   59    0   0:00:00 0.0% rpcbind/1
   123 root     2208K  648K sleep   59    0   0:00:04 0.0% in.routed/1
   253 root     4984K  880K sleep   59    0   0:00:00 0.0% automountd/3
    98 root     4592K 2616K sleep   59    0   0:00:05 0.0% nscd/26
   330 root     1992K  432K sleep   59    0   0:00:00 0.0% smcboot/1
 14962 root       13M   24K sleep   59    0   0:00:00 0.0% mem10.pl/1
   332 root     1984K  296K sleep   59    0   0:00:00 0.0% smcboot/1
   101 root     4664K 3296K sleep   59    0   0:00:07 0.0% picld/6
   182 root     5496K 2408K sleep   59    0   0:00:02 0.0% inetd/4
ZONEID    NPROC  SIZE   RSS MEMORY      TIME  CPU ZONE
    31       42  221M   98M   0.5%   0:00:10 0.1% workzone4
    34       42  221M   79M   0.4%   0:00:10 0.1% workzone3
    32       41  208M   60M   0.3%   0:00:11 0.0% workzone2
    33       40  206M   41M   0.2%   0:00:11 0.0% workzone1
     0       50  187M   65M   0.3%   0:06:17 0.0% global
Total: 215 processes, 597 lwps, load averages: 0.04, 0.18, 0.19
```

Superb! Our zones have been capped at 41Mb, 60Mb, 79Mb and 98Mb - which corresponds to our requested values of 40Mb, 60Mb, 80Mb and 100Mb.

Disk Size Resource Control Intro

The following are a few techniques to control the disk space consumed by a zone. Here we want to cap the size per zone,

- Separate Disks or Slices.
- Separate Volumes.
- ZFS file system quotas.

Separate Disks or Slices
If a zone was placed on a separate disk or slice, then it's file system would obviously be limited by the size of the disk or slice. This form of isolating a zone may be quite desirable, perhaps even for political reasons. One way to do this is to mount the slice in the global zone and update /etc/vfstab, then set the zone's zonepath to the

mountpoint. This was demonstrated in the big-zone example, however it dosen't necessarily need to be for the entire OS, the shared OS zone type would work on a separate slice as well.

Separate Volumes
A volume manager could be used to create a separate volume for a zone. The Solaris Volume Manager could be used (SVM, was called Solstice Disk Suite, SDS). This would be done for many of the same reasons as using separate disks or slices, with the usual advantages of volumes - mirroring, concatenations, RAID5, RAID10, etc. Depending on the requirements, soft partitions may be the most appropriate.

ZFS file system quotas
The Zetabyte File System for Solaris 10 can place several disks into a shared pool of storage, from which a filesystem can be supplied to each zone. ZFS is really both a volume manager and file system rolled into one, and can do mirrored and striped volumes. Quotas can be set on each zone's file system, to both limit the maximum space and set a reservation for minimum ensured space. NOTE: ZFS has not yet been released, what we know about it so far can be read here.

Disk Throughput Resource Control Intro

Controlling the rate at which zones use the disks may be desired to prevent one zone saturating the disks with activity and interfering with the other zones. So far there are few ways this can be achieved,

- Separate Disks or Controllers.

This category could well have developments in the future. Keep an eye out for announcements from Sun.

Separate Disks or Controllers
If a zone is placed on a separate disk, then disk activity by that zone will be served by it's own dedicated disk and interference with other zones will be minimised. However the zone may drive the disk at such a rate that the bus itself becomes saturated, in which case separate busses or controllers could be considered. This could be taken even further to consider system bus interference, hardware and software cache usage, interrupt interference, etc; however practically it may be best to stop with the disk and controller layout only.

Network Resource Control Intro

Resource control for network throughput may be necessary to prevent one zone's network traffic interfering with others. It can be implemented using the following techniques,

- Separate NICs.
- IPQoS.

Separate NICs
If a system had separate network interface cards or ports, a zone could be assigned to each. This provides a dedicated network interface per zone. Administrating network interface setting for zones is easy, as is reconfiguration if needed. As an example, a quad-fast ethernet card (qfe) could supply four zones. Each interface could be configured with a dummy address in the global zone, and then real addresses in the non-global zone via virtual interfaces.

IPQoS
IP Quality of Service was a new feature for Solaris 9 12/02, it is an implementation of the differentiated services model (RFC2475) - a way we can mark priority on network traffic based on some rules. The source address of each zone can be targeted by IPQoS to apply metering for sustained and burst Mb/s. /etc/inet/ipqosconf.2.sample may be a good place to start reading. This is most effective when other network devices such as routers also pay attention to IPQoS.

(PS. Can you even measure network throughput in Solaris? See nicstat, or in C, and it's example).

Network Resource Control 1

The following is a demonstration of assigning a dedicated network interface to a zone. The zone is workzone1, which has been previously created but lacks a network interface. The interface we will use is "ce1". ce1 needs to be configured in the global zone so that the zone can add a virtual address during boot, however the global zone address is not important,

```
global# echo dummy > /etc/hostname.ce1
global# echo '0.0.0.0   dummy   # zones interface' >> /etc/inet/hosts
global# ifconfig ce1 plumb
global# ifconfig ce1
ce1: flags=1000842<BROADCAST,RUNNING,MULTICAST,IPv4> mtu 1500 index 8
        inet 0.0.0.0 netmask 0
        ether 0:3:ba:63:13:35
```

Ok, now we configure the existing zone workzone1 to use the ce1 interface. If the zone did not currently exist, the zonecfg steps would be added to the usual setup,

```
global# zonecfg -z workzone1
zonecfg:workzone1> add net
zonecfg:workzone1:net> set address=192.168.7.32
zonecfg:workzone1:net> set physical=ce1
zonecfg:workzone1:net> end
zonecfg:workzone1> info
zonepath: /export/workzone1
autoboot: true
pool:
inherit-pkg-dir:
        dir: /lib
inherit-pkg-dir:
        dir: /platform
inherit-pkg-dir:
        dir: /sbin
inherit-pkg-dir:
        dir: /usr
net:
        address: 192.168.7.32
        physical: ce1
zonecfg:workzone1> verify
zonecfg:workzone1> commit
zonecfg:workzone1> exit
global#
```

Done. Now all we need to do is reboot workzone1 so that it picks up it's new interface, and check that it is configured ok,

```
global# zlogin -S workzone1 init 6
global#
global# zoneadm list -cv
  ID NAME             STATUS         PATH
   0 global           running        /
  31 workzone4        running        /export/workzone4
  32 workzone2        running        /export/workzone2
  33 workzone1        running        /export/workzone1
  34 workzone3        running        /export/workzone3
global#
global# ping -s 192.168.7.32
PING 192.168.7.32: 56 data bytes
64 bytes from 192.168.7.32: icmp_seq=0. time=0.264 ms
64 bytes from 192.168.7.32: icmp_seq=1. time=0.182 ms
64 bytes from 192.168.7.32: icmp_seq=2. time=0.109 ms
^C
----192.168.7.32 PING Statistics----
3 packets transmitted, 3 packets received, 0% packet loss
round-trip (ms)  min/avg/max/stddev = 0.109/0.185/0.264/0.078
global#
global# zlogin workzone1
[Connected to zone 'workzone1' pts/4]
Last login: Thu Mar 24 01:22:49 on pts/4
Sun Microsystems Inc.   SunOS 5.10      Generic January 2005
Welcome to Sol10_Generic on sfe2900
workzone1#
workzone1# ifconfig -a
lo0:4: flags=2001000849<UP,LOOPBACK,RUNNING,MULTICAST,IPv4,VIRTUAL> mtu 8232 index 1
        inet 127.0.0.1 netmask ff000000
ce1:1: flags=1000803<UP,BROADCAST,MULTICAST,IPv4> mtu 1500 index 4
```

```
        inet 192.168.7.32 netmask ffffff00 broadcast 192.168.7.255
workzone1#
```

Excellent. We have given workzone1 it's own dedicated network interface.

Zone Recommendations

The following are suggestions for managing zones. They are not official recommendations from Sun.

Best done from the Global Zone

- patching - to keep all zones in sync.
- backups - to skip shared filesystems.
- storage administration - as accessing /dev files are easier from the global zone.
- network administration - such as changing the IP address, which must be done from global.

SunFreeware.com packages

> These like to install themselves under /usr/local, and for many zones /usr/local is a read only shared filesystem. This prevents non-global zone admins from easily adding their own packages from SunFreeware.com.
>
> Now, if you use these packages from the Software Companion CD there isn't a problem, as they install themselves under /opt/sfw - which is always read write!
>
> A suggestion to solve the shared /usr/local issue is this: create a symlink on the global zone from "/usr/local" to "/opt/local", and create an /opt/local directory in every non global zone. Now any package adds in the non global zone will add to their /opt/local, not attempt to write to the read only /usr/local.

Zone Backups

The following is a summary of methods that can be used to backup zones. Many zones share files with the global zone, through use of loopback filesystem readonly mounts (usually /usr, /lib, /sbin and /platform); we don't want to backup these lofs directories as they may be already backed up via the usual global zone backup strategy, which makes zone backups not exactly straightforward.

- Shutting down the zone
- UFS snapshots
- find and cpio
- NetBackup
- Legato

These methods backup the zone files. Remember to also backup the config file from /etc/zones - for example, /etc/zones/smallzone.xml.

Shutting down the zone
So that we don't descend down lofs shares and backup duplicates of the global zone's files, here we shutdown the zone first before doing a backup. Any backup tool could be used, ufsdump is demonstrated here,

```
# zlogin -S smallzone init 5
# zoneadm list -cv
  ID NAME             STATUS         PATH
   0 global           running        /
```

```
   - smallzone        installed      /export/smallzone
# ufsdump 0f /backup/smallzone.ufsdump /export/smallzone
  DUMP: Date of this level 0 dump: Thu Apr 14 08:55:56 2005
  DUMP: Date of last level 0 dump: the epoch
  DUMP: Dumping /dev/rdsk/c0t0d0s0 (lambda:/) to /backup/smallzone.ufsdump.
  DUMP: Mapping (Pass I) [regular files]
  DUMP: Mapping (Pass II) [directories]
  DUMP: Writing 32 Kilobyte records
  DUMP: Estimated 235508 blocks (114.99MB).
  DUMP: Dumping (Pass III) [directories]
  DUMP: Dumping (Pass IV) [regular files]
  DUMP: 234494 blocks (114.50MB) on 1 volume at 2394 KB/sec
  DUMP: DUMP IS DONE
# zoneadm -z smallzone boot
```

UFS snapshots
This approach uses the "fssnap" command, which first appeared in Solaris 8 2/02. This can give us a clean, consistant backup of the zone files only, and can be run on a live zone. In the following example we have a zone under /export/smallzone, a separate /var to use as the backing store, and our destination backup is /backup/smallzone.ufsdump,

```
# fssnap -o bs=/var/tmp /
/dev/fssnap/0
# mount -o ro /dev/fssnap/0 /mnt
# ufsdump 0f /backup/smallzone.ufsdump /mnt/export/smallzone
  DUMP: Date of this level 0 dump: Wed Apr 13 09:08:14 2005
  DUMP: Date of last level 0 dump: the epoch
  DUMP: Dumping /dev/rfssnap/0 (lambda:/mnt) to /backup/smallzone.ufsdump.
  DUMP: Mapping (Pass I) [regular files]
  DUMP: Mapping (Pass II) [directories]
  DUMP: Writing 32 Kilobyte records
  DUMP: Estimated 231034 blocks (112.81MB).
  DUMP: Dumping (Pass III) [directories]
  DUMP: Dumping (Pass IV) [regular files]
  DUMP: 230014 blocks (112.31MB) on 1 volume at 2227 KB/sec
  DUMP: DUMP IS DONE
# umount /mnt
# fssnap -d /
Deleted snapshot 0.
```

find and cpio
The following method uses find and cpio on a live zone. We assume the zone is installed under UFS, and use find switches to restrict the search to UFS - and to not descend down lofs mount points,

```
# cd /
# find export/smallzone -fstype lofs -prune -o -fstype ufs | cpio -oc -O /backup/smallzone.cpio
# ls -l backup/smallzone.cpio
-rwxr-xr-x   1 root     root     98566144 Apr 12 12:34 backup/smallzone.cpio
```

NetBackup
It is possible to use NetBackup to backup a live zone, it will need to be configured to skip the lofs shared mounts (/usr, /lib, ...) that were configured for the zone.

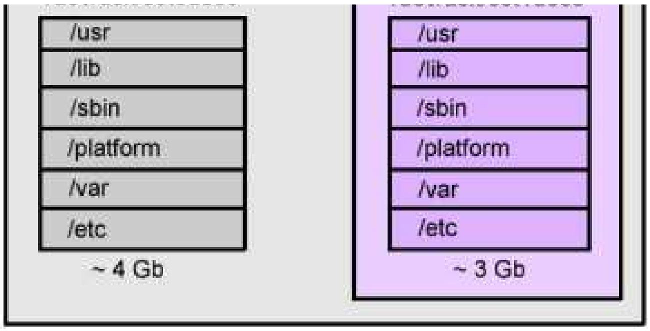
Legato
Legato can be used in the same way as NetBackup, it needs to be configured to skip the lofs mounts.

Zone Patches

To understand how patches works, lets go back and look at the different models for zones.

Big Zone

This is the Big-Zone, or the "Whole Root Model".



From the above we can see that on the global zone all directories are read+write, and an the big zone all directories are also read+write. The following may occur,

1. After logging into the big zone, a patch may be added using "patchadd" (but not a kernel patch).
2. After logging into the global zone, a patch may be added to the global zone only by using "patchadd -G".
3. After logging into the global zone, a patch may added to all the zones using "patchadd".

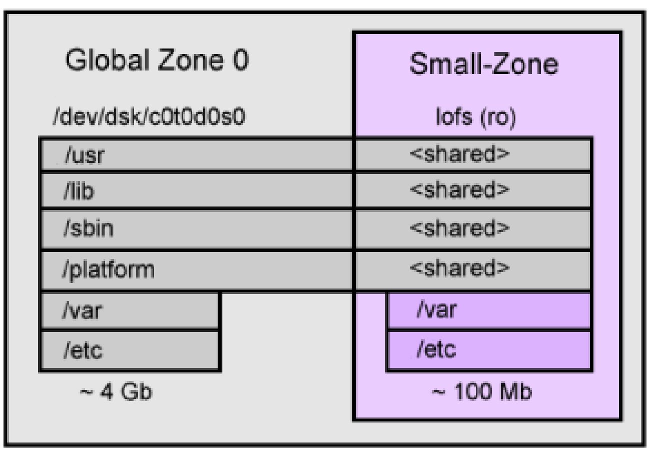
For scenario 3, read the output of patchadd carefully - you will see the command iterate over all the non-global zones. If a non-global zone is not running when patchadd is run, patchadd will boot the non-global zone into single user mode so that the patchadd will work, and then return the zone to it’s original state. The end result is that all zones are patched.

Do you want every zone to be running at a different patch level? Probably not. Depending on your requirements, it may well be best to manage patching from the global zone only.

If you run a patch cluster from the global zone and you have zones that aren’t booted, the current behaviour is to boot and shutdown each zone for each patch. This adds considerable time to the patch cluster - it helps to boot all zones before beginning so that patchadd doesn’t need to toggle their state. (this behaviour may be improved, check how your current version of Solaris 10 behaves).

Small Zone

This is the Small-Zone, also called the "Sparse Root Model",



From the above we can see that the global zone has all directories read+write, and the small zone has most of the operating system directories read only. The following may occur,

1. After logging into the small zone, regular patches that modify /usr can not be added.
2. After logging into the global zone, using "patchadd -G" will also update files shared by the small zone.
3. After logging into the global zone, a patch may be added to all the zones properly using "patchadd".

Scenario 2 leaves the small zone in an odd state (some shared files are patched, but /var not updated).

Scenario 3 becomes more interesting. When the patch is initially added to the global zone, since the non-global zones share the operating system directories, they will immediately be accessing patched files. The patchadd command will still iterate over all the non-global zones - it needs to, to update other patch related files in /var; during which it may complain that many files are read only. The end result is that all zones are patched.

Recommendations

Although this depends on your environment and requirements, it would seem best to manage patching from the global zone only, and let the global zone iterate over all non-global zones to patch them also. It helps if all zones are booted before patching.

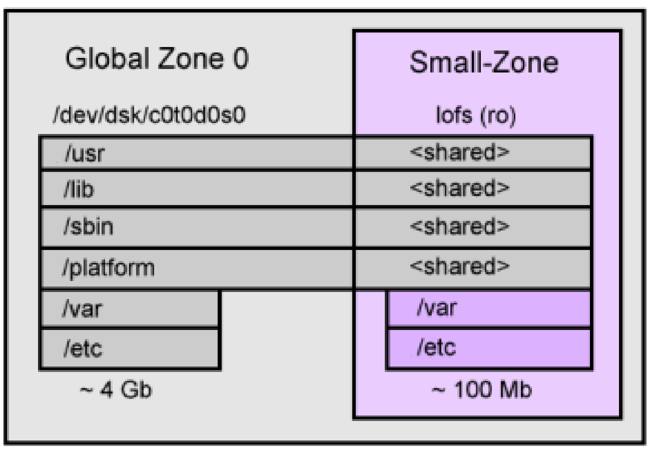
Sun is also promoting the use of Patch Manager to patch servers these days. See "man smpatch", especially the examples.

Zone Packages

To understand how packages work, lets look at the different models for zones.

Small Zone

This is the Small-Zone, also called the "Sparse Root Model",



From the above we can see that the global zone has all directories read+write, and the small zone has most of the operating system directories read only.

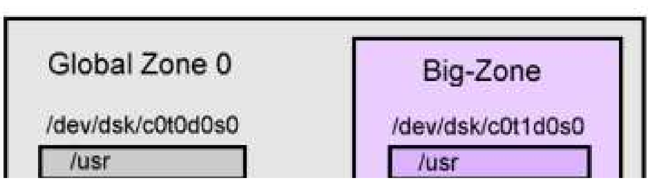
The following may occur,

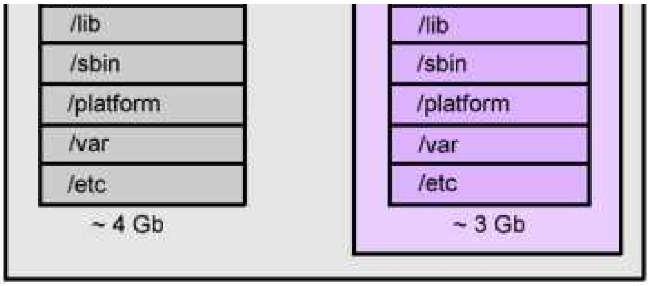
1. After logging into the small zone, packages may be added so long as they don’t modify /usr, /lib, /platform and /sbin. Eg, to install them under /opt. "pkgadd" is used.
2. After logging into the global zone, using "pkgadd -G" will add the package to the global zone, and may share some files to the small zone.
3. After logging into the global zone, using "pkgadd" will install everything in all zones.

Scenario 2 leaves the small zone in an odd state - it may contain much of the package, without being (pkginfo) aware.

Big Zone

This is the Big-Zone, or the "Whole Root Model".





From the above we can see that on the global zone all directories are read+write, and an the big zone all directories are also read+write. The following may occur,

1. After logging into the big zone, a package may be added using "pkgadd".
2. After logging into the global zone, a package may be added to the global zone only by using "pkgadd -G".
3. After logging into the global zone, a package may added to all the zones using "pkgadd".

The big zone is more flexible than the small zone - here we can add package independently and usually without problems.

Recommendations

Although this depends on your environment and requirements, it would seem that if the non-global zone had the need to install and maintain their own packages, they are better off as a big zone. If not, the small zone model would be fine.

Screenshots 1 - Ultra 5

The following is a screenshot of an Ultra 5 running 11 zones (one global plus ten non-global). All zones are configured to be in the Fair Share Schedular (FSS) for CPU sharing. This demonstration was created by Joe McKinnon (Brisbane, Australia),

```
# uname -a
SunOS fir 5.10 Generic sun4u sparc SUNW,Ultra-5_10
#
# psrinfo -v
Status of virtual processor 0 as of: 03/10/2005 09:23:02
  on-line since 03/08/2005 16:29:29.
  The sparcv9 processor operates at 400 MHz,
        and has a sparcv9 floating point processor.
#
# zoneadm list -cv
  ID NAME             STATUS         PATH
   0 global           running        /
   1 test5            running        /export/test5
   2 test1            running        /export/test1
   3 test             running        /export/test
   4 test6            running        /export/test6
   6 test7            running        /export/test7
   5 test2            running        /export/test2
   7 test3            running        /export/test3
   8 proteus          running        /export/pro
   9 test4            running        /export/test4
  12 silly            running        /export/silly
```

Now multiple find commands are run in each zone to create load,

```
# prstat -Z -n 9,11 -R
   PID USERNAME  SIZE   RSS STATE  PRI NICE      TIME  CPU PROCESS/NLWP
 23663 root     5744K 5416K cpu0   100    -   0:00:00 1.5% prstat/1
 22241 root     5680K 3424K sleep   40    0   0:00:07 1.4% prstat/1
   661 root       46M   15M sleep   13    0   0:15:40 1.2% Xsun/1
  5793 root       72M 7704K sleep    3    0   0:06:48 0.9% gnome-terminal/2
 20024 root      109M   25M sleep   60    0   0:01:01 0.2% mozilla-bin/3
  5769 root       67M 3328K sleep   55    0   0:01:33 0.2% gnome-netstatus/1
   304 root       93M 9728K run     59    0   0:03:10 0.2% poold/8
```

```
 23787 root     1256K 1016K sleep   60    0   0:00:00 0.1% find/1
 23799 root     1256K 1016K sleep   60    0   0:00:00 0.1% find/1
ZONEID    NPROC  SIZE   RSS MEMORY      TIME  CPU ZONE
     0       94 1504M  137M    19%   0:38:38 6.1% global
     9       76  161M   57M   8.0%   0:00:55 3.6% test4
     1       77  167M   59M   8.2%   0:00:54 3.6% test5
     3       76  160M   57M   8.0%   0:00:53 3.5% test
     8       77  168M   59M   8.3%   0:01:05 3.5% proteus
     2       76  161M   57M   8.0%   0:00:52 3.5% test1
     4       76  160M   57M   8.0%   0:00:53 3.5% test6
     5       76  161M   57M   8.0%   0:00:54 3.5% test2
    12       76  163M   58M   8.2%   0:00:36 3.4% silly
     6       76  160M   57M   8.0%   0:00:55 3.4% test7
     7       76  160M   57M   8.0%   0:00:54 3.3% test3
Total: 856 processes, 1782 lwps, load averages: 20.95, 14.38, 6.85
```

Find is causing a lot of disk activity and interrupts are making the system unresponsive, including a monitoring shell in the Real Time (RT) class.

Next, multiple CPU intensive commands such as sar are executed,

```
# prstat -Z -n 9,11
   PID USERNAME  SIZE   RSS STATE  PRI NICE      TIME  CPU PROCESS/NLWP
 25754 root     5904K 5600K cpu0   120    -   0:00:00 2.4% prstat/1
 25060 root     1536K 1104K run     58    0   0:00:01 0.5% sadc/1
 24950 root     1528K 1096K run     52    0   0:00:01 0.5% sadc/1
 25116 root     1528K 1096K run     51    0   0:00:01 0.5% sadc/1
 25359 root     1528K 1096K run     55    0   0:00:01 0.5% sadc/1
 24485 root     1536K 1080K run     52    0   0:00:02 0.5% sadc/1
 25172 root     1528K 1096K run     55    0   0:00:01 0.5% sadc/1
 25299 root     1528K 1096K run     55    0   0:00:01 0.5% sadc/1
 24821 root     1536K 1080K run     55    0   0:00:02 0.5% sadc/1
ZONEID    NPROC  SIZE   RSS MEMORY      TIME  CPU ZONE
     1       94  191M   73M   8.8%   0:01:42  10% test5
     3       94  190M   73M   8.8%   0:01:44 9.6% test
     2       94  190M   73M   8.8%   0:01:44 9.5% test1
     6       94  190M   73M   8.8%   0:01:42 9.3% test7
     4       94  190M   73M   8.8%   0:01:41 9.1% test6
    12       94  192M   74M   8.9%   0:01:22 9.0% silly
     8       94  192M   74M   8.9%   0:01:50 8.7% proteus
     5       94  190M   73M   8.8%   0:01:41 8.6% test2
     9       94  191M   73M   8.7%   0:01:25 6.4% test4
     7       94  190M   72M   8.7%   0:01:21 6.1% test3
     0       93 1498M   99M    12%   0:39:23 3.6% global
Total: 1033 processes, 1963 lwps, load averages: 380.93, 290.58, 151.64
```

This time the FSS copes really well dividing CPU between the zones, an RT monitoring shell runs without delay.

Further Info

For more Solaris 10 Zones see,

- Zones the OpenSolaris zones website has the best collection of links.
- Zones FAQ many useful answers to common questions.
- Zones Sun's website that links to the documentation and forum.
- Zone Answerbook the Zone Answerbook from docs.sun.com.
- SRMBlog contains various hints, tips and tricks for Solaris Resource Pools.
- Get in the Zone is Dennis Clark's walkthrough on Zones, and the first document many of us ever saw on Zones.
- Oracle in Zones Best Practices for running Oracle in Solaris Zones

Back to Brendan Gregg's Homepage

Created: 12-Jan-2005
Last updated: 24-May-2006