

Wednesday, June 16, 2010

## Less known Solaris Features: CPU resource Pools

After writing about the processor sets and their role in computing the number of garbage collection threads recently, there were some questions about configuring processor sets. In this article i want to explain the usage of resource pools in Solaris in regard of CPU resources. You don't configure the processor sets directly. Instead of this the pool facility does this for you.

I want to explain how to configure, how to use and how to monitor them. However i can just scratch the surface in such an article.

Prerequisites When you want to demonstrate a feature of the OS that influences the way processes are dispatched on the CPUs it's useful to have something that produces a lot of load. A normal household cpuhog will be sufficient, so i use the one from the resource manager tutorial.

```
jmoekamp@hivemind:~$ cat cpuhog.pl
```

```
#!/usr/bin/perl
```

```
while (1) { my $res = ( 3.3333 / 3.14 ) }
```

Configuring the CPU resource pool There are two important commands to control the behavior of the pool facility. pooladm and poolcfg.

Normally the pool facility isn't activated, so you can't work with it.

```
jmoekamp@hivemind:~# pooladm
```

```
pooladm: couldn't open pools state file: Facility is not active
```

```
Additionally it's not possible to configure
```

```
it.
```

```
jmoekamp@hivemind:~# poolcfg -c "discover"
```

```
poolcfg: cannot create the configuration, hivemind: Facility is not active
```

```
So we have to activate the pool facility
```

```
first.
```

```
jmoekamp@hivemind:~# pooladm -e
```

```
Now we are able to start the configuration.
```

```
jmoekamp@hivemind:~# poolcfg -c
```

```
"discover"
```

```
The "discover" subcommand scans the system and creates a matching configuration and writes them into the
```

```
default location
```

```
cpu
  int   cpu.sys_id 3
  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
```

With the poolcfg -c "info" you can view the currently persistent configuration in regard of the processor distribution. This configuration is contained in the file /etc/pooladm.cfg. In its native appearance it's a XML file, but poolcfg -c "info" makes something a little bit (but not much ) readable format out of it. When Solaris detects this file at startup it will automatically start the pool facility.  
jmoekamp@hivemind:~# poolcfg -c "info"

```
system default
  string system.comment
  int   system.version 1
  boolean system.bind-default true
  string system.poold.objectives wt-load
```

```
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 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 218
  uint  pset.size 4
  string pset.comment
```

```
cpu
  int   cpu.sys_id 1
  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 0
  string cpu.comment
  string cpu.status on-line
```

```
cpu
  int   cpu.sys_id 2
```

```
string cpu.comment
string cpu.status on-line
```

At start both outputs are identical, but the difference will be much clearer soon.

Let's configure the pool facility. At first we create a new processor set.jmoekamp@hivemind:~# poolcfg -c 'create pset hog\_set (uint pset.min=1 ; uint pset.max=1)'

This command creates a processor set that has at least one CPU and at most one cpu. Or to say it easier: It has exactly one CPU all the time. In the next step we create a pool. You can think of the pool as the entity that poold use as a administrative unit. jmoekamp@hivemind:~# poolcfg -c 'create pool hog\_pool' For my example I've created a pool called hog\_pool. Okay ... now we have to configure the connection between the pool and the processor set.jmoekamp@hivemind:~# poolcfg -c 'associate pool hog\_pool (pset hog\_set)'

With the associate subcommand you configure this connection.

Okay, let's have short look into the state of of pool facility.jmoekamp@hivemind:~# poolstat

```
      pset
id pool      size used load
 0 pool_default 4 0,00 0,18
```

We see just one pool at the moment. We have to activate the currently configured configuration first. We have to do this by using the the pooladm command.jmoekamp@hivemind:~# pooladm -c

When we use the poolstat command again, you will see two pools and the distribution matches the configured state. One CPU for hog\_pool and the rest (3 in my case) for the default pool.jmoekamp@hivemind:~# poolstat

```
      pset
id pool      size used load
 3 hog_pool   1 0,00 0,00
 0 pool_default 3 0,00 0,17
```

When you look at the currently running configuration, you will see that one processor has moved to the hog\_set processor set.jmoekamp@hivemind:~# pooladm

system default

```
string system.comment
int system.version 1
boolean system.bind-default true
string system.poold.objectives wt-load
```

```
pool hog_pool
  int pool.sys_id 3
  boolean pool.active true
  boolean pool.default false
  int pool.importance 1
  string pool.comment
  pset hog_set
```

```
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 hog_set
  int pset.sys_id 1
  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 0
```

```
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 177  
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 3  
string cpu.comment  
string cpu.status on-line
```

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

But there is a little gotcha in the configuration. The currently persistent configuration looks a little bit differently. Let's look up that configuration. `jmoekamp@hivemind:~# poolcfg -c "info"`

```
system default  
string system.comment  
int system.version 1  
boolean system.bind-default true  
string system.poold.objectives wt-load
```

```
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 hog_pool  
boolean pool.active true  
boolean pool.default false  
int pool.importance 1  
string pool.comment  
pset hog_set
```

```
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 218  
uint pset.size 4  
string pset.comment
```

```
cpu
  int   cpu.sys_id 1
  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 0
  string cpu.comment
  string cpu.status on-line
```

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

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

The hog\_set and hog\_pool exist, but hog\_set has no processor and pset\_default has all the procs ...

To make the current distribution boot persistent you have to write the currently active configuration in the persisting configuration. `jmoeekamp@hivemind:~# pooladm -s` When you now recheck the persisting configuration, you will see that it's equal to the configuration you get with the `pooladm` command. However you should do this step only when you are sure that the configuration matches your expectations. Let's recheck the persistent configuration. `jmoeekamp@hivemind:~# poolcfg -c "info"`

```
system default
  string system.comment
  int   system.version 1
  boolean system.bind-default true
  string system.poold.objectives wt-load
```

```
pool hog_pool
  int   pool.sys_id 3
  boolean pool.active true
  boolean pool.default false
  int   pool.importance 1
  string pool.comment
  pset  hog_set
```

```
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 hog_set
  int   pset.sys_id 1
```

```
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 0
  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 166
  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 3
  string cpu.comment
  string cpu.status on-line
```

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

jmoekamp@hivemind:~#

Using the the resource pools Okay, now we have this nice processor set, but how do we use it. There are several possibilities, but i my example i will explain the usage of projects to use the resource pools.

At first i create a project. jmoekamp@hivemind:~# projadd -U jmoekamp -K project.pool=hog\_pool hog\_project The commands create a project with the name hog\_project, the user jmoekamp is member of this project. The most interesting part is the -K project.pool=hog\_pool part. This tells Solaris to use the pool hog\_pool for everything executed within this project.

When you think about the Resource Manager tutorial you are already know how to start command under a dedicated project (and then you will know, you are already running your processes within a project called user.jmoekamp, too). You can do this by using the newtask command. I will start some the CPU hogs to demonstrate the impact of the CPU pool. jmoekamp@hivemind:~\$ newtask -p hog\_project ./cpuhog.pl &

[1] 24633

jmoekamp@hivemind:~\$ newtask -p hog\_project ./cpuhog.pl &

[2] 24636

jmoekamp@hivemind:~\$ newtask -p hog\_project ./cpuhog.pl &

[3] 24639

jmoekamp@hivemind:~\$ newtask -p hog\_project ./cpuhog.pl &

[4] 24642

jmoekamp@hivemind:~\$ newtask -p hog\_project ./cpuhog.pl &

[5] 24645 When you look at output of poolstat a little bit later, you will see an unloaded default set as well as an really hard working hog\_pool. jmoekamp@hivemind:~# poolstat

```

pset
id pool      size used load
 3 hog_pool  1 0,00 4,84
 0 pool_default 3 0,00 0,07
prstat -J -C 1
  PID USERNAME SIZE  RSS STATE PRI NICE   TIME CPU PROCESS/NLWP
24645 jmoekamp 9068K 2200K cpu0  41  0  0:01:50 5,5% cpuhog.pl/1
24642 jmoekamp 9068K 2200K run   32  0  0:01:44 5,1% cpuhog.pl/1
24636 jmoekamp 9068K 2200K run   32  0  0:01:44 5,0% cpuhog.pl/1
24639 jmoekamp 9068K 2200K run   32  0  0:01:37 4,7% cpuhog.pl/1
24633 jmoekamp 9068K 2200K run   32  0  0:01:35 4,5% cpuhog.pl/1
[...]
PROJID  NPROC SWAP  RSS MEMORY   TIME CPU PROJECT
  115    5 1540K 8988K  0,2%  0:08:30 25% hog_project
[...]
Total: 5 processes, 5 lwps, load averages: 4,99, 4,05, 2,12

```

Albeit there are at least two CPUs, there is just one cpuhog on a CPU and all other are runnable, but wait until they are dispatched on the cpu0.

Okay, we don't need those hogs any longer ...

```

jmoekamp@hivemind:~# pkill "cpuhog.pl"
[1] Terminated      newtask -p hog_project ./cpuhog.pl
[2] Terminated      newtask -p hog_project ./cpuhog.pl
[3] Terminated      newtask -p hog_project ./cpuhog.pl
[4] Terminated      newtask -p hog_project ./cpuhog.pl
[5] Terminated      newtask -p hog_project ./cpuhog.pl

```

Posted by Joerg Moellenkamp in English, Solaris, Sun/Oracle at 09:16

Great article!

This can be very useful when I want to limit resources stolen by antivirus/antispam daemons in the mail queue. A new question about cpu pools comes to my mind now: what about using it in conjunction with latest VirtualBox? I mean, if I configure a guest with 2 processors, using all the virt flags, where it is actually running? Is it still moving from core to core? Maybe combining the multiprocessor virt guest and cpu pools would let my guest run on 2 specific cores all the time, and get some benefit?

Anonymous on Jun 16 2010, 10:04

It may be worth mentioning that if a zone is configured with a number of "dedicated-cpu"s, a resource pool will be created on start up, which the zone is then running in.

Anonymous on Jun 16 2010, 11:48

Since resource pools allocate certain CPU cores, by ID, only those certain cores will be usable by VirtualBox if started inside a resource pool.

Anonymous on Jun 16 2010, 11:50

Yes, but considering that some of the VBox daemons run shared and in the kernel, would it be sufficient to run the VBoxHeadless inside a configured project?

And, would I gain any better performance from it?

Anonymous on Jun 16 2010, 13:01

Thanks, Jörg.

I guess I was one of the guys asking for this kind of configuration...

Anonymous on Jun 16 2010, 13:42

Hi,  
nice article!

However, one question. When processes run binded to a pool, I do see that it consumes only 1 CPU, but load averages goes to 4, even 5.

Why is it so?

Thanks,  
Nik

Anonymous on Jun 16 2010, 21:42

## Blog Export: c0t0d0s0.org, <http://www.c0t0d0s0.org/>

That's due to the way load avg is computed: It's usr time + systime + thread wait. When a CPU is totally consumed by usr and sys by a hog, all other processes have to wait, so they accumulate thread wait time. So it's normal that 5 processes would yield a load avg of 5 when scheduled to a single cpu.

Anonymous on Jun 16 2010, 22:31