

Thursday, January 7, 2010

The Deliverer's knowledge: Sun Cluster Step by Step - How to add an additional node to an existing single-node cluster?

(Foreword from Joerg: The next installment of "The Deliverer's Knowledge" is a pretty advanced topic: Sun Cluster. This article it's a cooperation, Heiko wrote the technical side of tutorial, i expanded his notes with a few comments. So typos are my, not his fault)

Installing a single-node cluster is drop-dead simple, when you just follow the User Manual. But now you've played a while with it. It got a little bit boring. You want to see a resource switching from one node to another. So how do you get a cluster node into an existing single node cluster. So how do you expand the cluster to a two-node cluster?

Environment This example makes some assumption about the environment:

```
OSSolaris 10
SC3.2
Clusternamet-clust
existing nodenode1
Metasetsc-set
Resourcegroupsc-rg
Resourcesstor-rslh-rs
new Nodenode2
NICs for interconnectce0,ce1
IPMPsnode1-group@1,node2-group@2
```

Requirements: Furthermore there are some tasks, you have to do before you start the configuration. It's important, that you wire the interconnect before you start the configuration.

Afterwards you should check, if the operating system of both nodes is patched to the same level. It's a good practice to have similar systems at the start of your system.

A new member for the cluster Okay, at first look up the the version and the patch level on both nodes. It should be equal. You can look it up on both nodes after installing the cluster packages.

```
# scinstall -pv
Solaris Cluster 3.2u2 for Solaris 10 sparc
SUNWscu: 3.2.0,REV=2006.12.05.22.58, 125511-02 126106-27 140208-03 126106-35
SUNWsccomu: 3.2.0,REV=2006.12.05.22.58, 125511-02 126106-27 140208-03 126106-35
SUNWsczr: 3.2.0,REV=2006.12.05.22.58, 125511-02 126106-27 140208-03 126106-35 140017-02
SUNWsccomzu: 3.2.0,REV=2006.12.05.22.58, 125511-02 126106-27 140208-03 126106-35
SUNWsczu: 3.2.0,REV=2006.12.05.22.58, 125511-02 126106-27 140208-03 126106-35
SUNWscsckr: 3.2.0,REV=2006.12.05.22.58, 125992-03 140208-03 125992-04
SUNWscscku: 3.2.0,REV=2006.12.05.22.58, 125992-03 140208-03 125992-04
SUNWscr: 3.2.0,REV=2006.12.05.22.58, 125511-02 126106-27 140208-03 126106-35 140017-02
SUNWscrtlh: 3.2.0,REV=2006.12.05.22.58, 126106-27 140208-03 126106-35
SUNWscnmr: 3.2.0,REV=2008.11.20.15.31
SUNWscnmu: 3.2.0,REV=2008.11.20.15.31
SUNWscdev: 3.2.0,REV=2006.12.05.22.58, 125511-02 126106-27 140208-03 126106-35
SUNWscgds: 3.2.0,REV=2006.12.05.22.58, 126106-27 140208-03 126106-35
SUNWscsmf: 3.2.0,REV=2006.12.05.22.58, 126106-27 140208-03 126106-35
SUNWscman: 3.2.0,REV=2006.09.26.13.45, 128556-03 140210-02
SUNWscsal: 3.2.0,REV=2006.12.05.22.58, 126106-27 140208-03 126106-35
SUNWscsam: 3.2.0,REV=2006.12.05.22.58, 126106-27 140208-03 126106-35
SUNWscvm: 3.2.0,REV=2006.12.05.22.58, 140208-03
SUNWmdmr: 3.2.0,REV=2006.12.05.22.58, 140208-03
SUNWmdmu: 3.2.0,REV=2006.12.05.22.58, 125514-05 140208-03
```

SUNWscmasa: 3.2.0,REV=2006.12.05.22.58, 126106-27 140208-03 126106-35
SUNWscmasar: 3.2.0,REV=2006.12.05.22.58, 126106-27 140208-03 126106-35
SUNWscmasasen: 3.2.0,REV=2006.12.05.22.58, 126106-27 140208-03 126106-35
SUNWscmasau: 3.2.0,REV=2006.12.05.22.58, 125511-02 126106-27 140208-03 126106-35
SUNWscmautil: 3.2.0,REV=2006.12.05.22.58, 125508-08 126106-27 140208-03 126106-35
SUNWscmautilr: 3.2.0,REV=2006.12.05.22.58, 140208-03
SUNWjfreechart: 3.2.0,REV=2006.12.05.22.58, 139921-02 140208-03
SUNWscspmr: 3.2.0,REV=2006.12.05.22.58, 140208-03
SUNWscspmu: 3.2.0,REV=2006.12.05.22.58, 125508-08 126106-27 140208-03 126106-35
SUNWscderby: 3.2.0,REV=2006.12.05.22.58, 126106-27 140208-03 126106-35
SUNWsc telemetry: 3.2.0,REV=2006.12.05.22.58, 125511-02 126106-27 140208-03 126106-35
SUNWcsc: 3.2.0,REV=2006.10.26.11.18, 126095-05 140210-02
SUNWcscspmu: 3.2.0,REV=2006.10.26.11.18, 126095-05 140210-02
SUNWdsc: 3.2.0,REV=2006.10.26.11.14, 126095-05 140210-02
SUNWdscspmu: 3.2.0,REV=2006.10.26.11.14, 126095-05 140210-02
SUNWesc: 3.2.0,REV=2006.10.26.11.16, 126095-05 140210-02
SUNWescspmu: 3.2.0,REV=2006.10.26.11.16, 126095-05 140210-02
SUNWfsc: 3.2.0,REV=2006.10.26.11.13, 126095-05 140210-02
SUNWfscspmu: 3.2.0,REV=2006.10.26.13.24, 126095-05 140210-02
SUNWhsc: 3.2.0,REV=2006.10.26.11.20, 126095-05 140210-02
SUNWhscspmu: 3.2.0,REV=2006.10.26.11.20, 126095-05 140210-02
SUNWjsc: 3.2.0,REV=2006.10.26.11.21, 126095-05 140210-02
SUNWjscman: 3.2.0,REV=2006.10.26.11.21, 126095-05 140210-02
SUNWjscspmu: 3.2.0,REV=2006.10.26.11.21, 126095-05 140210-02
SUNWksc: 3.2.0,REV=2006.10.26.11.17, 126095-05 140210-02
SUNWkscspmu: 3.2.0,REV=2006.10.26.11.17, 126095-05 140210-02
SUNWscmys: 3.2.0,REV=2006.12.06.18.32, 126032-05 140213-02 126032-08
SUNWscsmb: 3.2.0,REV=2006.12.06.18.32, 139908-01 140213-02

Configuring the cluster interconnect There are some components a single-node cluster doesn't need. You don't need a cluster interconnect when you have just one node. Obvious, isn't it? So let's start by configuring the interconnect on node1:

```
#clinterconnect add node1:ce0  
#clinterconnect add node1:ce1
```

In the case you have a switched interconnect, you have to configure the so-called junctions.

```
#clinterconnect add switch0  
#clinterconnect add node1:ce0,switch0  
#clinterconnect add switch1  
#clinterconnect add node1:ce1,switch1
```

Now you've configured the interconnect, but you have to enable it before you can use it:

```
#clinterconnect enable node1:ce0,switch0@1  
#clinterconnect enable node1:ce1,switch1@1
```

Okay, let's check it:
clinterconnect show
Transport Cables ===

```
Transport Cable:                node1:ce0,switch0@1  
Endpoint1:                      node1:ce0  
Endpoint2:                      switch0@1  
State:                          Enabled
```

```
Transport Cable:                node1:ce1,switch1@1  
Endpoint1:                      node1:ce1  
Endpoint2:                      switch1@1  
State:                          Enabled
```

Transport Switches ===

```
Transport Switch:               switch0  
State:                          Enabled
```

```
Type:                switch
Port Names:         1
Port State(1):      Enabled
```

```
Transport Switch:   switch1
State:              Enabled
Type:               switch
Port Names:         1
Port State(1):      Enabled
```

--- Transport Adapters for node1 ---

```
Transport Adapter:  ce0
State:              Enabled
Transport Type:     dlpi
device_name:        ce
device_instance:    0
lazy_free:          1
dlpi_heartbeat_timeout: 10000
dlpi_heartbeat_quantum: 1000
nw_bandwidth:       80
bandwidth:          70
ip_address:         172.18.0.129
netmask:            255.255.255.128
Port Names:         0
Port State(0):      Enabled
```

```
Transport Adapter:  ce1
State:              Enabled
Transport Type:     dlpi
device_name:        ce
device_instance:    5
lazy_free:          1
dlpi_heartbeat_timeout: 10000
dlpi_heartbeat_quantum: 1000
nw_bandwidth:       80
bandwidth:          70
ip_address:         172.18.1.1
netmask:            255.255.255.128
Port Names:         0
Port State(0):      Enabled
```

The configuration is ready, but we have to tell your cluster, that node2 is allowed to join the cluster. That's easy:
`#claccess allow -h node2`

To check the successful completion you can just look into the file `/etc/cluster/ccr/infrastructure`
`# more infrastructure`

```
...
cluster.properties.auth_joinlist_hostslist  node2
```

...Now you have to login to your new cluster node node2. You start the configuration will all the necessary data, so the installation won't ask you interactively. Afterwards restart the cluster node.

```
#cluster show | grep Name
```

```
Cluster Name:          t-clust
```

```
# scinstall -i -C t-clust -N node1 -A trtype=dlpi,name=ce0 -A trtype=dlpi,
name=ce1 -m endpoint=:ce0,endpoint=switch0 -m endpoint=:ce1,endpoint=switch1
```

`# init 6`After the boot, you will see several new messages while you start the system. After the boot, log in as root and check the cluster configuration.

```
#clnode list
```

```
node1
```

```
node2
```

#clinterconnect show
Transport Cables ===

Transport Cable: node1:ce0,switch0@1
Endpoint1: node1:ce0
Endpoint2: switch0@1
State: Enabled

Transport Cable: node1:ce1,switch1@1
Endpoint1: node1:ce1
Endpoint2: switch1@1
State: Enabled

Transport Cable: node2:ce0,switch0@2
Endpoint1: node2:ce0
Endpoint2: switch0@2
State: Enabled

Transport Cable: node2:ce1,switch1@2
Endpoint1: node2:ce1
Endpoint2: switch1@2
State: Enabled

Transport Switches ===

Transport Switch: switch0
State: Enabled
Type: switch
Port Names: 1 2
Port State(1): Enabled
Port State(2): Enabled

Transport Switch: switch1
State: Enabled
Type: switch
Port Names: 1 2
Port State(1): Enabled
Port State(2): Enabled

--- Transport Adapters for node1 ---

Transport Adapter: ce0
State: Enabled
Transport Type: dlpi
device_name: ce
device_instance: 0
lazy_free: 1
dlpi_heartbeat_timeout: 10000
dlpi_heartbeat_quantum: 1000
nw_bandwidth: 80
bandwidth: 70
ip_address: 172.18.0.129
netmask: 255.255.255.128
Port Names: 0
Port State(0): Enabled

Transport Adapter: ce1
State: Enabled
Transport Type: dlpi

```
device_name:          ce
device_instance:      5
lazy_free:            1
dlpi_heartbeat_timeout: 10000
dlpi_heartbeat_quantum: 1000
nw_bandwidth:         80
bandwidth:            70
ip_address:           172.18.1.1
netmask:              255.255.255.128
Port Names:          0
Port State(0):        Enabled
```

--- Transport Adapters for node2 ---

```
Transport Adapter:    ce0
State:                Enabled
Transport Type:       dlpi
device_name:          ce
device_instance:      0
lazy_free:            1
dlpi_heartbeat_timeout: 10000
dlpi_heartbeat_quantum: 1000
nw_bandwidth:         80
bandwidth:            70
ip_address:           172.18.0.130
netmask:              255.255.255.128
Port Names:          0
Port State(0):        Enabled
```

```
Transport Adapter:    ce1
State:                Enabled
Transport Type:       dlpi
device_name:          ce
device_instance:      5
lazy_free:            1
dlpi_heartbeat_timeout: 10000
dlpi_heartbeat_quantum: 1000
nw_bandwidth:         80
bandwidth:            70
ip_address:           172.18.1.2
netmask:              255.255.255.128
Port Names:          0
Port State(0):        Enabled
```

#clquorum status

Quorum devicesNo we have to create a quorum device. For many people the quorum devices are a little bit difficult concept. The quorum device has an important role in Sun Cluster. It prevents two partitioning effects ("cluster amnesia" and "split brain"), that could lead to data corruption due to applications accessing the same data at the same time. You can find a description of these effects in the Sun Cluster Concepts Guide.

You have to know when the cluster is partitioned there is a vote about "Who is the operational cluster?". The rule is simple: Every member of the cluster gets one vote. The cluster partition, that has more votes is the operational cluster. That's easy for a 3-Node cluster. Whenever a partitioning occurs, you can be sure that there is one partition with 2 nodes and one partition with one node. Thus the vote goes 2:1. You have a winner. But what is the situation with a splitted two node cluster. Both partitions have a single node, Both have one vote - 1:1 . You need a tie breaker and the quorum device has exactly this role. In essence the quorum device is something like a flag. Whoever gets it first, has won and is the operational half of a cluster.

The practical side is a little bit more complex, as the vote count is configurable. You need to configure it in some

situation to ensure that a cluster can start, even when just node comes up. Remember, that a cluster just can get up, when it's operational and an operational cluster needs the majority of votes.

Let's assume you have a three node cluster. So the vote count is three. Let's assume that two of your nodes are failed. Thus the surviving node can't get the majority of votes as he has just one vote. So you can configure a quorum device that has two votes. Your cluster has a total vote count of five. A single member has still a vote count of 1. When this member gets the quorum, this cluster part has a vote count of 3, thus it would have the majority.

Okay, let's check the current quorum configuration:

```
# clquorum status
```

```
Cluster Quorum ===
```

```
--- Quorum Votes Summary ---
```

Needed	Present	Possible
2	2	2

```
--- Quorum Votes by Node ---
```

Node Name	Present	Possible	Status
node1	1	1	Online
node2	1	1	Online

As you see ... an even number of vote counts. This cluster would just start only if all nodes available. At first we have to choose a device for it.

```
#cldev list -v
```

DID Device	Full Device Path
d1	node1:/dev/rdisk/c0t0d0
d2	node1:/dev/rdisk/c1t3d0
d3	node1:/dev/rdisk/c1t0d0
d4	node1:/dev/rdisk/c1t1d0
d5	node1:/dev/rdisk/c1t4d0
d6	node1:/dev/rdisk/c4t500AAB800066B202000002AD4AEEFBD2d0
d6	node2:/dev/rdisk/c4t500AAB800066B202000002AD4AEEFBD2d0
d10	node2:/dev/rdisk/c0t0d0
d11	node2:/dev/rdisk/c1t4d0
d12	node2:/dev/rdisk/c1t1d0
d13	node2:/dev/rdisk/c1t0d0
d14	node2:/dev/rdisk/c1t3d0

In our case we will use the device d6

```
#clquorum add d6
```

So let's check our quorum configuration again.

```
#clquorum status
```

```
Cluster Quorum ===
```

```
--- Quorum Votes Summary ---
```

Needed	Present	Possible
-----	-----	-----

2 3 3

--- Quorum Votes by Node ---

Node Name	Present	Possible	Status
node1	1	1	Online
node2	1	1	Online

--- Quorum Votes by Device ---

Device Name	Present	Possible	Status
d6	1	1	Online

Nice, we have a odd number of total votes, the node that gets the quorum disk is the functional one.

Final stepsBut we are not at the end of the configuration. At first we have to tell the Solaris Volume manager, that node2 is allowed to use the metaset sc-set, thus accessing the data on the metaset.

```
# metaset -s sc-set -a -h node2
# metaset -s sc-set -a -m node1
# metaset -s sc-set -a -m node2
```

Now you have to tell the cluster that it has new resources. At first you tell the resource lh-res that it has an additional interface on a second node. At second, you add the new node to the resource-group sc-rg

```
#clrs set -p Netlflist=node1-group@1,node2-group@2 lh-res
#clrg add-node -n node2 sc-rg
Now you are done
```

Posted by Heiko Stein in English, the deliverer's knowledge at 10:59

Wednesday, November 18, 2009

The deliverer's knowledge: Some comments on performance data for capped CPU's with Solaris 10 zones

(An introduction by Jörg: This is the second blog entry written by Heiko. It's an interesting article about a weirdness you can stumble upon while working with Solaris 10.)

Solaris 10 Update 5 aka Solaris 10 5/08 introduced a feature to limit the absolute CPU usage of zones or projects defined by a percentage. The relevant resource control is zone.cpu-cap. While this is nothing really new, you should be aware about the interactions (and possible misunderstandings arising out of them) of this capping feature and values showing the utilization of a system like LoadAvg

A simple example Here is a simple example. As a test system I used a virtual machine with 4 CPU's and 2 running zones.

```
#>zoneadm list -icv
ID NAME      STATUS  PATH                BRAND IP
0 global    running /                native shared
9 zone1     running /zones/zone1  native shared
10 zone2    running /zones/zone2  native shared
- zone3     installed /zones/zone3  native shared
- zone4     installed /zones/zone4  native shared
```

The availability of 4 CPU can be shown by the output of psrinfo

```
# >psrinfo
0 on-line since 11/03/2009 08:19:02
1 on-line since 11/03/2009 08:19:08
2 on-line since 11/03/2009 08:19:08
3 on-line since 11/03/2009 08:19:10
```

We take a short look at the CPU utilization. In essence, the system just runs itself at the moment. A very relaxing situation.

```
# >vmstat 1
kthr  memory      page      disk      faults  cpu
r b w  swap free re mf pi po fr de sr cd cd f0 s0 in sy cs us sy id
0 0 0 978940 596200 116 439 95 0 0 0 79 10 2 -0 -0 701 3321 1456 4 14 82
0 0 0 812240 438140 8 30 0 0 0 0 0 0 0 0 0 430 173 232 0 1 98
0 0 0 812240 438140 0 6 0 0 0 0 0 16 0 0 0 715 142 350 0 2 98
0 0 0 812240 438140 0 6 0 0 0 0 0 0 0 0 0 442 157 229 0 1 99
0 0 0 812240 438140 0 6 0 0 0 0 0 0 0 0 0 433 200 217 0 1 99
0 0 0 812240 438140 0 6 0 0 0 0 0 0 0 0 0 437 214 233 0 1 99
0 0 0 812240 438140 0 6 0 0 0 0 0 0 0 0 0 438 162 243 0 1 99
^C
```

```
# >prstat -mLZ 1
PID USERNAME USR SYS TRP TFL DFL LCK SLP LAT VCX ICX SCL SIG PROCESS/LWPID
2203 root 0.2 3.0 0.0 0.0 0.0 0.0 97 0.0 37 0 525 0 prstat/1
122 root 0.1 0.1 0.0 0.0 0.0 0.0 100 0.0 36 0 218 0 nscd/3
2137 noaccess 0.1 0.0 0.0 0.0 0.0 0.0 100 0.0 1 0 1 0 java/15
...
9 root 0.0 0.0 0.0 0.0 0.0 0.0 100 0.0 0 0 0 0 svc.configd/1
7 root 0.0 0.0 0.0 0.0 0.0 0.0 100 0.0 0 0 0 0 svc.startd/323
7 root 0.0 0.0 0.0 0.0 0.0 0.0 100 0.0 0 0 0 0 svc.startd/316
7 root 0.0 0.0 0.0 0.0 0.0 0.0 100 0.0 0 0 0 0 svc.startd/66
7 root 0.0 0.0 0.0 0.0 0.0 0.0 100 0.0 0 0 0 0 svc.startd/10
ZONEID NLWP SWAP RSS MEMORY TIME CPU ZONE
0 183 140M 212M 21% 0:01:16 0.3% global
2 123 135M 204M 20% 0:00:52 0.0% zone2
```

```
1 126 138M 208M 20% 0:00:50 0.0% zone1
Total: 102 processes, 432 lwps, load averages: 0.04, 0.39, 0.49
```

Configuration of the CPU cappingNext, the capping for a local Zone called zone1 is activated and set to 10%. Keep in mind: The activation via prctl is not boot persistent. The calculation for the ratio of capping value can be done like this: 4CPU = 400 → 400 = 100% → 40 = 10 %

```
# >prctl -t privileged -n zone.cpu-cap -s -v 40 -i zone zone1
```

Let's check the configuration:

```
# >prctl -P -i zone zone1
```

```
zone: 1: zone1
```

```
zone.max-swap system 18446744073709551615 max deny -
```

```
zone.max-locked-memory system 18446744073709551615 max deny -
```

```
zone.max-shm-memory system 18446744073709551615 max deny -
```

```
zone.max-shm-ids system 16777216 max deny -
```

```
zone.max-sem-ids system 16777216 max deny -
```

```
zone.max-msg-ids system 16777216 max deny -
```

```
zone.max-lwps system 2147483647 max deny -
```

```
zone.cpu-cap privileged 40 - deny -
```

```
zone.cpu-cap system 4294967295 inf deny -
```

```
zone.cpu-shares privileged 1 - none -
```

zone.cpu-shares system 65535 max none -BTW, there is also a corresponding kstat module for the CPU Capping. The name of the relevant field is cpucaps_zone_ .

```
#kstat -m caps -n cpucaps_zone_ zoneadm list -icv | grep zone1 | awk '{print $1}'
```

```
module: caps
```

```
instance: 1
```

```
name: cpucaps_zone_1 class: zone_caps
```

```
above_sec 0
```

```
below_sec 1335
```

```
crtime 4801.079480706
```

```
maxusage 12
```

```
nwait 0
```

```
snaptime 6135.748493858
```

```
usage 1
```

```
value 40
```

```
zonename zone1
```

Pedal to the metal - or testing the capTo test the capping, just use your favorite CPU hog. I wrote a small - albeit not very elegant C program - to fulfill this purpose.

```
# >vi load.c
```

```
#include
```

```
#include
```

```
#include
```

```
#include
```

```
int C,I;
```

```
void loop1()
```

```
{
```

```
double x=0, m;
```

```
int i=0;
```

```
while(i == 0)
```

```
{
```

```
m = acos(x);
```

```
x++;
```

```
}
```

```
}
```

```
main()
```

```
{
```

```
pthread_attr_t attr;
```

```
pthread_attr_init(&attr);
```

for (C=0; C man ps

```
...
S (l)      The state of the process:
...
W          Waiting: process is waiting
           for CPU usage to drop to
           the CPU-caps enforced lim-
           its.
```

...
Let's use this knowledge on our system:

```
#> ps -o s=state -o comm=command -aelfL | grep load | more
```

```
W ./load.bin
W ./load.bin
W ./load.bin
```

```
...
W ./load.bin
W ./load.bin
W ./load.bin
W ./load.bin
```

```
# > ps -o s=state -o comm=command -aelfL | grep load | grep W | wc -l
2135
```

The number of wait kthreads can also be controlled via kstat. The value of interest for this situation is "nwait".

```
# >kstat -m caps -n cpucaps_zone_`zoneadm list -icv | grep zone1 | awk '{print $1}'`
```

```
module: caps          instance: 1
name: cpucaps_zone_1  class: zone_caps
  above_sec           2141
  below_sec           3753
  crtime              4801.079480706
  maxusage            135
  nwait               2156
  snaptime            10694.627372473
  usage               40
  value               40
  zonename            zone1
```

What's the reason for this difference between LoadAvg and the data displayed by other tools? Commands such as prstat / uptime / w etc. use the syscall getloadavg(), which apparently evaluates the number of entries in the runqueues, but isn't aware of the wait-flag. We can check this by a short dtrace one-liner:

```
# >dtrace -n 'syscall:::/probefunc=="getloadavg"/ {trace(execname)}'
```

```
dtrace: description 'syscall:::' matched 466 probes
```

```
CPU  ID      FUNCTION:NAME
  2  2626      getloadavg:entry  uptime
  2  2627      getloadavg:return  uptime
  2  2626      getloadavg:entry  prstat
  2  2627      getloadavg:return  prstat
  2  2626      getloadavg:entry  prstat
  2  2627      getloadavg:return  prstat
  2  2626      getloadavg:entry  prstat
  2  2627      getloadavg:return  prstat
  2  2626      getloadavg:entry  prstat
  2  2627      getloadavg:return  prstat
  2  2626      getloadavg:entry  w
  2  2627      getloadavg:return  w
```

```
^C
```

Conclusion There is an interesting interaction between getloadavg() and zone.cpu-cap that leads to misleading, but perfectly correct numbers. You should keep this in mind, when you try to make sense out of a system with a extremely high load average that's still responsive.

Do you want to learn more?

Misc

opensolaris.org: The implementation of CPU caps

Posted by Heiko Stein in English, the deliverer's knowledge at 13:56

Wednesday, October 28. 2009

The Deliverer's Knowledge: What's the max. number of IP instances per NIC?

A foreword by Jörg: This is the first blog entry written by Heiko. "the deliverer's knowledge" is just a working title and it's referring to the fact, that Heiko delivers vast knowledge on a daily basis at various customer sites. I just add the c0t0d0s0.org uncoporate non-design. So this is Heiko direct and unfiltered. Please welcome him. PS: Another substantially longer article will be online soon.

Take a look at:
ndd -get /dev/ip ip_addrs_per_if
256

That's the default. You can set the count up to 8191. (Btw: That's already the max. number of local zones per global zone)

ndd -set /dev/ip ip_addrs_per_if 8191
But keep in mind, the newly set value isn't boot persistent, so you should set it during the system startup. For example by creating a transient SMF service or an init.d-script

Posted by Heiko Stein in English, Solaris, Sun/Oracle, the deliverer's knowledge at 21:42