



# Solaris 10 New Features

**Stefan Parvu**

**Client Solutions**

**[stefan.parvu@sun.com](mailto:stefan.parvu@sun.com)**



# Agenda

- **What's New ?**
- N1 Grid Containers - Zones
- Dynamic Tracing - DTrace
- Predictive Self Healing - SMF
- Java Desktop System - JDS

# What's New ?

- N1 Grid Containers - Zones
- Dynamic Tracing - DTrace
- Predictive Self Healing - SMF
- Security - privileges(5)
- A new file system: ZFS
- Desktop: JDS based on Gnome 2.6
- Solaris on Sparc, x86 and AMD Opteron
- Java 5
- A new logo !

# What's New ?

## Janus Linux Compatibility

A LSB compliant environment, under Solaris x86, to execute unmodified Linux binaries

You can develop Linux applications on Solaris taking advantage of its performance, scalability and security, even your production is RHAS !

Next-generation *lrun* replacement

*lrun* = user-space emulation layer

Janus = direct support of Linux kernel interfaces

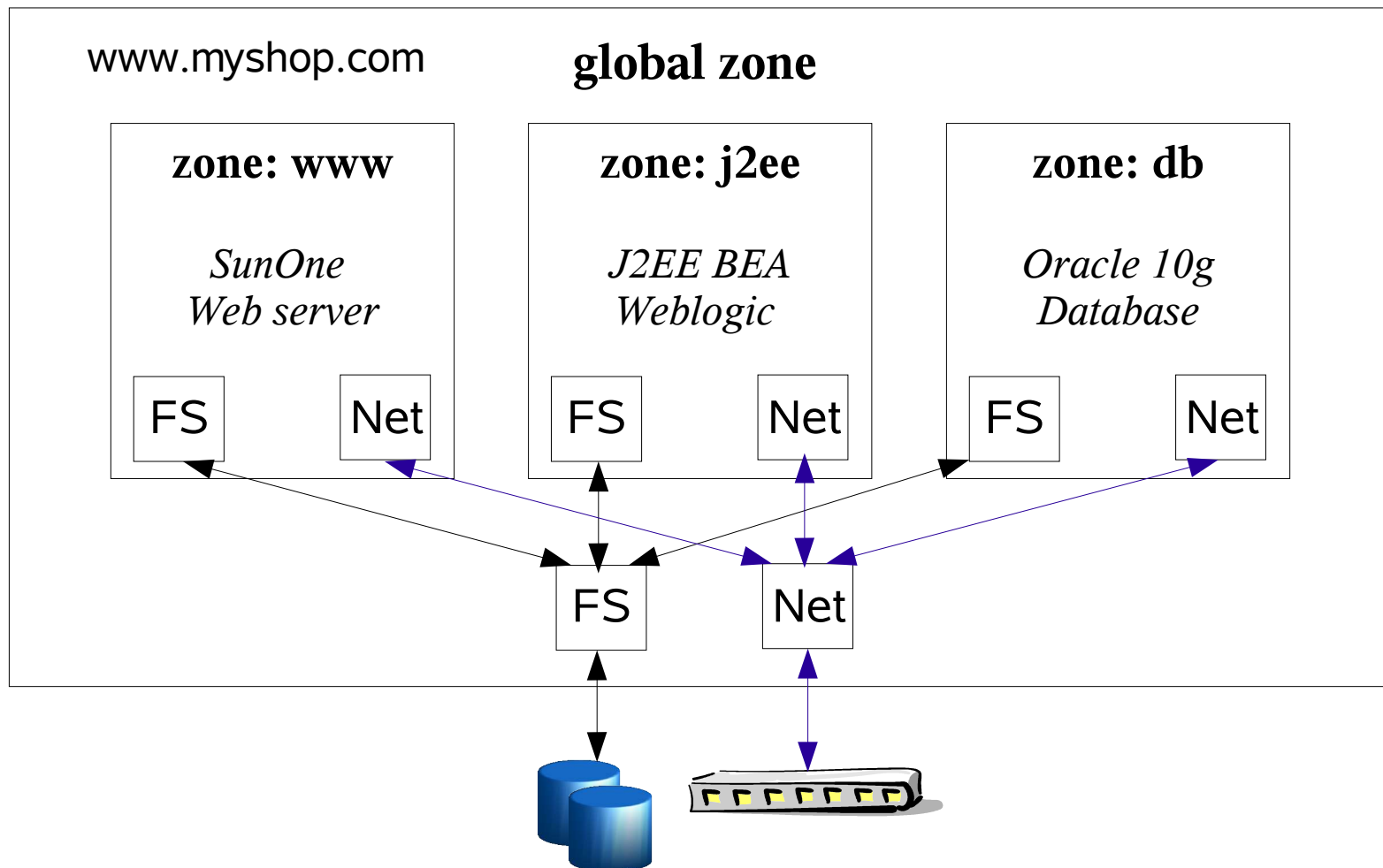
# Agenda

- What's New ?
- **N1 Grid Containers - Zones**
- Dynamic Tracing - DTrace
- Predictive Self Healing - SMF
- Java Desktop System - JDS

# N1 Grid Containers - Zones

- A new isolation primitive for Solaris: secure, flexible, scalable and lightweight
- Isolations: Partitioning/Domains, VMs, software partitions (Zones, Jails, Linux Vservers)
- Reduce costs by running multiple workloads on same system and isolates applications from each other
- Resource management framework + Zones = N1 Grid Containers

# Zones: Global and Non-Global



# Primary Zone States

- **Configured:** The configuration is done and committed to stable storage
- **Installed:** Packages have been installed under the zone's root file system
- **Ready:** Virtual platform it is up: logical network interfaces are plumbed, file systems are mounted, devices are created /dev
- **Running:** User processes are executing in the zone

# Configuring a Zone

Use `zonectfg(1M)` to start configuring a zone

- `zonpath`: path in the global zone root directory under which zone will be installed
- `autoboot`: to boot or not to boot when the global zone boots
- `pool`: resource pool to which zone should be bound
- `fs`: file system
- `net`: network device
- `device`: device
- `inherit-pkg-dir`: directory inherited from the global zone

# Configuring a Zone

Start configuring a zone under `/zone/1`

```
# zoneadm list -vc
```

ID	NAME	STATUS	PATH
0	global	running	/

```
# chmod -R 700 zone
```

```
# ls -lap /zone
```

```
total 6
```

drwx-----	3	root	root	512	Nov 20 22:53	./
drwxr-xr-x	40	root	root	1024	Nov 20 22:53	../
drwx-----	2	root	root	512	Nov 20 22:53	1/

```
# zonecfg -z zone1
```

```
zone1: No such zone configured
```

```
Use 'create' to begin configuring a new zone.
```

```
zonecfg:zone1> create
```

```
zonecfg:zone1> set zonepath=/zone/1
```

```
zonecfg:zone1> set autoboot=true
```

# Configuring a Zone

```
zonecfg:zone1> add net
zonecfg:zone1:net> set address=192.168.1.20
zonecfg:zone1:net> set physical=iprb0
zonecfg:zone1:net> end
zonecfg:zone1> info
zonepath: /zone/1
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: 192.168.1.20
    physical: iprb0
```

```
zonecfg:zone1>
zonecfg:zone1> verify
zonecfg:zone1> commit
zonecfg:zone1>
```

# Configuring a Zone

Lets check our zone, zone1

```
# zonecfg -z zone1 info
```

```
zonepath: /zone/1
```

```
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: 192.168.1.20
```

```
    physical: iprb0
```

And now lets check all zones

```
# zoneadm list -vc
```

ID	NAME	STATUS	PATH
0	global	running	/
-	zone1	<b>configured</b>	/zone/1

# Installing a Zone

Next, install the zone using `zoneadm(1M)`

```
# zoneadm -z zone1 install
```

```
Preparing to install zone <zone1>.
```

```
Creating list of files to copy from the global zone.
```

```
Copying <97980> files to the zone.
```

```
Initializing zone product registry.
```

```
Determining zone package initialization order.
```

```
Preparing to initialize <1157> packages on the zone.
```

```
Initialized <1157> packages on zone.
```

```
Zone <zone1> is initialized.
```

```
...
```

The file `</zone/1/root/var/sadm/system/logs/install_log>` contains a log of the zone installation.

```
# zoneadm list -vc
```

ID	NAME	STATUS	PATH
0	global	running	/
-	zone1	<b>installed</b>	/zone/1

# Booting a Zone

And then boot and start using the zone:

```
# zoneadm -z zone1 boot
```

```
# zoneadm list -vc
```

ID	NAME	STATUS	PATH
0	global	running	/
1	zone1	<b>running</b>	/zone/1

```
# ping 192.168.1.20
```

```
192.168.1.20 is alive
```

```
# zlogin -C zone1
```

```
# uname -a
```

```
SunOS zone1 5.10 s10_72 i86pc i386 i86pc
```

# Zone Security

- A zone has a security boundary around it
- Cannot compromise another zone or entire system
- Processes running in non-global zones are not able to affect activity in other zones
- Global zone root user can do and see everything
- Restrictions: loading/unloading kernel modules, accessing kernel memory, details of physical resource are hidden

# Process Model in a Zone

- Processes in the same zone interact as usual
- Processes may not see or interact with processes in other zone
- Information via `proc(4)` for processes from that zone only
- Each active zone contains a process `zsched`
- The process tree in a zone is rooted at the `zsched`
- Global zone is able to see processes in all zones

# File Systems in a Zone

- Virtualized view of the file system namespace
- The root of the zone is *\$zonepath/root* and it is part of the configuration
- Processes cannot escape out of a zone
- Private: */* , */etc* and */var*
- Shared: */usr* , */lib* , */platform* and */sbin* read-only loopback mounts, *lofs(7FS)* from global zone and */proc* */etc/mnttab* are virtualized for each non-global zone

# Networking in a Zone

- Single TCP/IP stack for the entire system
- Zones are assigned one or more IPv4/IPv6 addresses
- Zones cannot view another zone's network traffic but inter-zone traffic is permitted
- When a zone is booted a logical interface is plumbed for each of its addresses
- Except for ICMP, raw IP socket access is not allowed within zone

# More information ?

Sun BigAdmin site

<http://www.sun.com/bigadmin/content/zones/>

Man pages:

getzoneid(3C), getzoneidbyname(3C)

getzonenamebyid(3C)

zcons(7D), zlogin(1)

zoneadm(1M), zonecfg(1M)

zonename(1), zones(5)

# Agenda

- What's New ?
- N1 Grid Containers - Zones
- **Dynamic Tracing - DTrace**
- Predictive Self Healing - SMF
- Java Desktop System - JDS

# Dynamic Tracing - DTrace

- A new power tool for real-time analysis and debug. System and process centric
- Hard to debug transient problems with: `truss(1)`, `pstack(1)`, `prstat(1)`
- Only `mdb(1)` designed for systemic problems but only for postmortem analysis
- Designed for live production systems: a totally safe way to inspect live data on production systems

# Dynamic Tracing - DTrace

- Safe and comprehensive: over 30.000 data monitoring points, inspect kernel and user space level
- Reduced costs: solutions usually found in minutes or hours not days or months.
- Flexibility: DTrace lets you create your own custom programs to dynamically instrument the system
- No need to instrument your applications, no need to stop or restart them

# Concepts

- Probe: it is a point of instrumentation, made available by a provider, which has a name
  - defined as (provider: module: function: name)
  - E.g.: syscall::read:entry
- Provider: a methodology for instrumenting the system. Makes available all known probes e.g. syscall, lockstat, fbt

# Concepts

- Consumer: a process which interacts with DTrace, `dtrace(1)` for instance
- D Language: a simple dynamically interpreted language what `dtrace` uses. It is like a C language with some constructs similar with `awk(1)`
  - Supports ANSI C operators, support for strings
  - D expressions based on built-in variables: `pid`, `execname`, `timestamp`, `curthread`

# Concepts

- D Language:

- Actions: are taken when a probe fires. You can use `trace()` which records the result of trace into a specific buffer. Actions are indicated by following a probe specification with “{ action }”

E.g:

```
dtrace -n BEGIN'{trace("Im here"); exit(0)}'
```

- Predicates: allow actions to only be taken when certain conditions are met. A predicate has this form : “/predicate/”

E.g:

```
dtrace -n syscall:::entry'/execname=="mozilla-bin"/{...}'
```

# Hello World

```
BEGIN
{
trace("hello, world");
exit(0);
}
```

## Call it from a script:

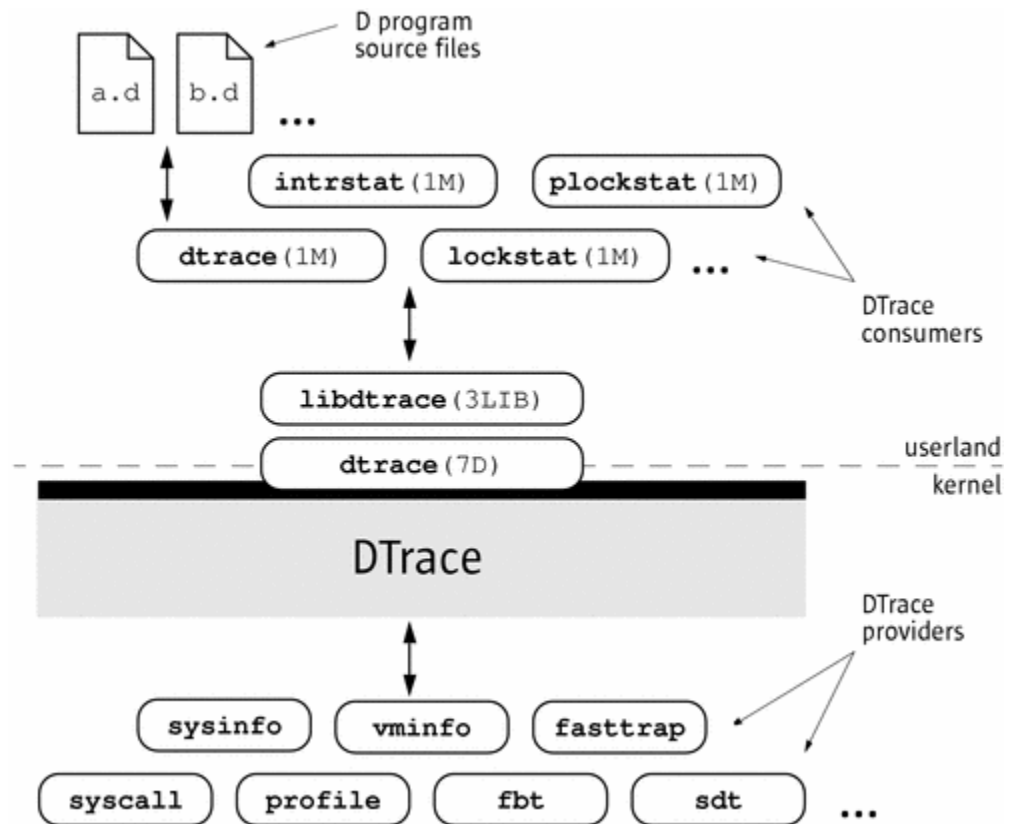
```
# dtrace -s hello.d
dtrace: script 'hello.d' matched 1 probe
CPU      ID          FUNCTION:NAME
  0       1              :BEGIN    hello, world
```

## or from command line:

```
# dtrace -q -n BEGIN'{ trace("Hello world"); exit(0) }'
Hello world
#
```

# Behind DTrace

- Compilation: D programs are compiled into a safe intermediate form that is used for execution when your probes fire which is validated by DTrace.



# Behind DTrace

- Programming mistake: DTrace will report your error to you, disable your instrumentation
- Execution environment: DTrace also handles any run-time errors: dividing by zero, dereferencing invalid memory, and so on, and reports them to you
- Safe: you can never construct a bad script that would cause DTrace to damage the Solaris kernel or one of the processes running on your system

# A simple case

Listing all probes offered by different providers:

```
# dtrace -l | more
```

ID	PROVIDER	MODULE	FUNCTION	NAME
1	dtrace		BEGIN	
2	dtrace		END	
3	dtrace		ERROR	
4	syscall		nosys	entry
5	syscall		nosys	return
6	syscall		rexit	entry
7	syscall		rexit	return
8	syscall		forkall	entry
9	syscall		forkall	return
10	syscall		read	entry
11	syscall		read	return
12	syscall		write	entry
13	syscall		write	return
14	syscall		open	entry
15	syscall		open	return
...				

# A simple case

Count all probes offered by all providers:

```
# dtrace -l | wc -l  
39270
```

Count all probes offered by the syscall provider:

```
# dtrace -l -P syscall | wc -l  
451
```

Count all probes offered by the ufs module:

```
# dtrace -l -m ufs | wc -l  
900
```

How to enable a probe: run dtrace without -l

```
# dtrace -q -n BEGIN'{trace("Hello World");}'  
Hello World
```

# A simple case

What's happening on my system:

```
# dtrace -n syscall:::entry
```

Which applications are using these system calls

```
# dtrace -n syscall:::entry '{trace(execname)}'
```

We want to aggregate on each application name:

```
# dtrace -n syscall:::entry '{@[execname] = count()}'
```

```
^C
```

ssh	3243
metacity	3270
wnck-applet	4040
search	9072
mv	9316
awk	15630
Xorg	16239
sort	26578
sh	41929
tar	151067

# A simple case

What “tar” is doing on my system ?

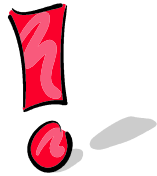
```
# dtrace -n syscall:::entry/execname=="tar"/{@[probefunc] = count  
( )}'  
dtrace: description 'syscall:::entry' matched 228 probes  
^C
```

waitsys	13
exece	13
vfork	13
sigpending	13
lstat64	14
...	
brk	196
write	201
mmap	210
lseek	420
llseek	6862
<b>read</b>	<b>7967</b>

# A simple case

We will try to find out what process is and get its pid

```
# dtrace -n syscall:::entry'/execname=="tar"/{@[pid] = count()}'  
dtrace: description 'syscall:::entry' matched 228 probes  
^C
```



<b>3898</b>	5
<b>3873</b>	5
<b>3893</b>	5
<b>3878</b>	5
<b>3857</b>	145
<b>3917</b>	206
<b>3862</b>	558
<b>3882</b>	559
<b>3877</b>	565
<b>3902</b>	566
<b>3912</b>	567
<b>3907</b>	568

# A simple case

We know now: that there are many “tar” processes running, but what file(s) and who is the user ?

```
# dtrace -s /usr/demo/dtrace/iosnoop.d
```

DEVICE	FILE	RW
sd1	/export/home/stefan/archivelibs.tar	W
sd1	<none>	R
sd1	<none>	W
sd1	/var/tmp/stmAAzJa43n.00000001	W
sd1	/export/home/stefan/archivelibs.tar	W
sd1	/var/tmp/stmAAyNaG4n.00000001	W
sd1	/export/home/stefan/archivelibs.tar	W

```
# dtrace -s /usr/demo/dtrace/whoio.d
```

```
^C
```

DEVICE	APP	PID	BYTES
sd1	tar	12393	12288
sd1	tar	12413	12288
sd1	tar	12358	16384
sd1	tar	12388	16384

```
...
```

# A simple case

We found the filename, the process name and we know that there are many tar processes running from time to time. But why !?

```
# ptree `pgrep tar`
7      /lib/svc/bin/svc.startd
  233  /usr/lib/saf/sac -t 300
    249  /usr/lib/saf/ttymon
      330  /usr/lib/saf/ttymon -g -d /dev/console -l console -T sun-
color -m ldterm,ttcomp
511    /lib/svc/bin/svc.startd
  609  /usr/lib/saf/sac -t 300
    612  /usr/lib/saf/ttymon
      617  /usr/lib/saf/ttymon -g -d /dev/console -l console -T sun-
color -m ldterm,ttcomp
2170  /usr/bin/gnome-terminal
    2191  ksh
      6221  /bin/ksh -p ./search
        16107 tar uvf /export/home/stefan/archivelibs.tar ./
iconv/amd64/UTF-32LE%UTF-8.so
```



# Solution: A simple case

One of developers was working to one script, called search:

```
...  
while true  
do  
  ...  
  for file in $(find /usr/lib/iconv -type f -name \*.so)  
  do  
    tar cvf ${HOME}/archivelibs.tar $file 2>&1 > /dev/null  
  done  
  ...  
done
```

So, that's why dtrace was reporting us lots of tar processes running. The developer fixed the code as:

# Solution: A simple case

```
...  
while true  
do  
  ...  
  tar cvf ${HOME}/archivelibs.tar $( find /usr/lib/iconv -type f -name  
  \*.so ) 2>&1 > /dev/null  
  ...  
done
```

At the end: using DTrace we were able to detect a slow and bad written application, to see what this was executing and to report and fix the problem !

# More information ?

Sun BigAdmin site

<http://www.sun.com/bigadmin/content/dtrace/>

Man pages:

dtrace(1M), dtrace(7D)

libdtrace(3LIB)

# Agenda

- What's New ?
- N1 Grid Containers - Zones
- Dynamic Tracing - DTrace
- **Predictive Self Healing - SMF**
- Java Desktop System - JDS

# Predictive Self Healing - SMF

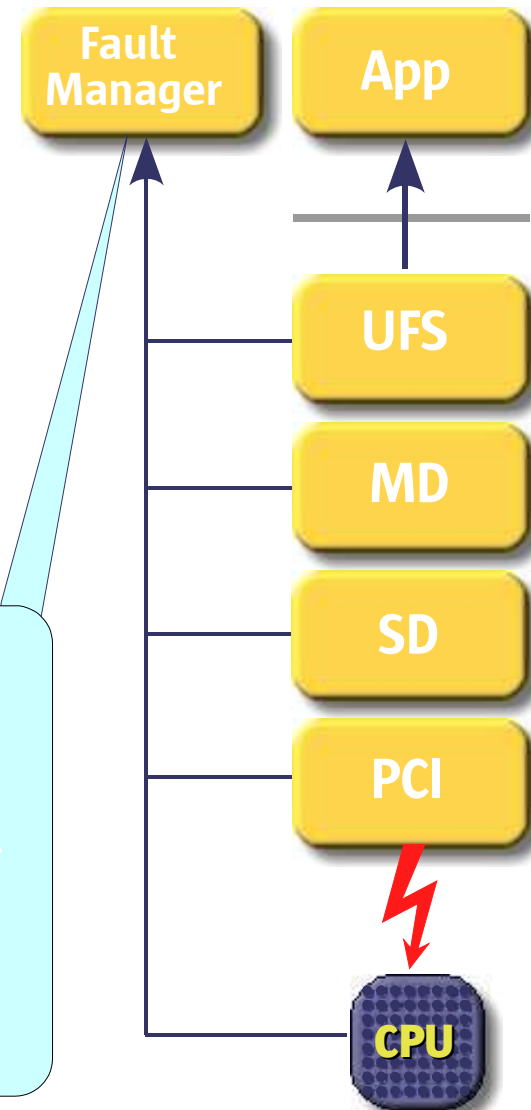
- New approach to service availability
  - Error detection & aggregation, auto recovery
- Reduced downtime
  - Components proactively offlined before failure
  - Automatic service restart
  - Diagnosis & mitigation in milliseconds, not hours
- Reduced complexity
  - Simplified error reporting
  - All system & service interdependencies recorded and correlated
- Reduced costs
  - Reduced system downtime, increased utilization



# Predictive Self Healing - SMF

- User-friendly error messages with impact and action statements
- All events are managed and coordinated through a single fault manager service

**SUNW-MSG-ID:** SFV440-8000-A6, **TYPE:** Fault, **VER:** 1, **SEVERITY:** Major  
**EVENT-TIME:** Thu Feb 26 18:08:26 PST 2004  
**PLATFORM:** SUNW,Sun-Fire-V440, **CSN:** -, **HOSTNAME:** mix  
**SOURCE:** cpumem-diagnosis, **REV:** 0.1  
**EVENT-ID:** 322fe6d5-fe14-6a73-b802-cc6c30b2afcd  
**DESC:** The number of errors associated with this CPU has exceeded acceptable levels.  
Refer to <http://sun.com/msg/SFv440-8000-A6> for more information.  
**AUTO-RESPONSE:** An attempt will be made to remove the affected CPU from service.  
**IMPACT:** Performance of this system may be affected.  
**REC-ACTION:** Schedule a repair procedure to replace affected CPU.



# Predictive Self Healing - SMF

The Service Management Facility smf(5) provides a new infrastructure enlarging the traditional UNIX start-up scripts, init run levels and configuration files:

- An infrastructure to start and restart services automatically
- A mechanism to formalize relationships between services
- A repository for storage of service startup behavior and configuration information

# Predictive Self Healing - SMF

## Components:

- svcs(1) service status listings, diagnosis
- svcadm(1M) administrative actions
- svccfg(1M) general property manipulation
- svcprop(1) property reporting

## inetd(1M)specific support:

- inetadm(1M) administrative property mods
- inetconv(1M) conversion of legacy inetd.conf

## private commands:

- lsvcrun(1) run legacy rc\*.d scripts
- mfstscan(1) detect updated manifests

# Predictive Self Healing - SMF

## svcs(1) in action

list active instances, sorted by state and time

show dependencies (-d)

show member processes (-p) and additional info (-v)

```
$ svcs
```

STATE	STIME	FMRI
legacy_run	Nov_25	lrc:/etc/rc3_d/S84appserv
legacy_run	Nov_25	lrc:/etc/rc3_d/S84patchserver
legacy_run	Nov_25	lrc:/etc/rc3_d/S90samba
online	Nov_25	svc:/system/fmd:default
online	Nov_25	svc:/platform/i86pc/kdmconfig:default
online	Nov_25	svc:/system/console-login:default
online	Nov_25	svc:/milestone/multi-user:default
online	Nov_25	svc:/milestone/multi-user-server:default
online	Nov_25	svc:/system/zones:default
offline	Nov_25	svc:/application/print/ipp-listener:default
offline	Nov_25	svc:/application/print/rfc1179:default

# Predictive Self Healing - SMF

```
$ svcs -p /network/ssh:default
```

```
STATE          STIME          FMRI
online         Nov_25        svc:/network/ssh:default
               Nov_25          124 sshd
```

```
$ svcs -d /network/ssh:default
```

```
STATE          STIME          FMRI
online         Nov_25        svc:/network/loopback:default
online         Nov_25        svc:/system/filesystem/usr:default
online         Nov_25        svc:/system/cryptosvc:default
```

```
$ svcs -v /network/ssh:default
```

```
STATE          NSTATE          STIME          CTID          FMRI
online         -               Nov_25          27           svc:/network/ssh:default
```

```
$ svcs -x
```

```
svc:/application/print/server:default (LP Print Service)
```

```
State: disabled since Thu Nov 25 19:36:15 2004
```

```
Reason: Disabled by an administrator.
```

```
See: http://sun.com/msg/SMF-8000-05
```

```
See: lpsched(1M)
```

```
Impact: 2 dependent services are not running. (Use -v for list.)
```

# More information ?

Sun BigAdmin site

<http://www.sun.com/bigadmin/content/selfheal/>

Man pages:

smf(5)

svcs(1),svcadm(1M)

svccfg(1M), svcprop(1)

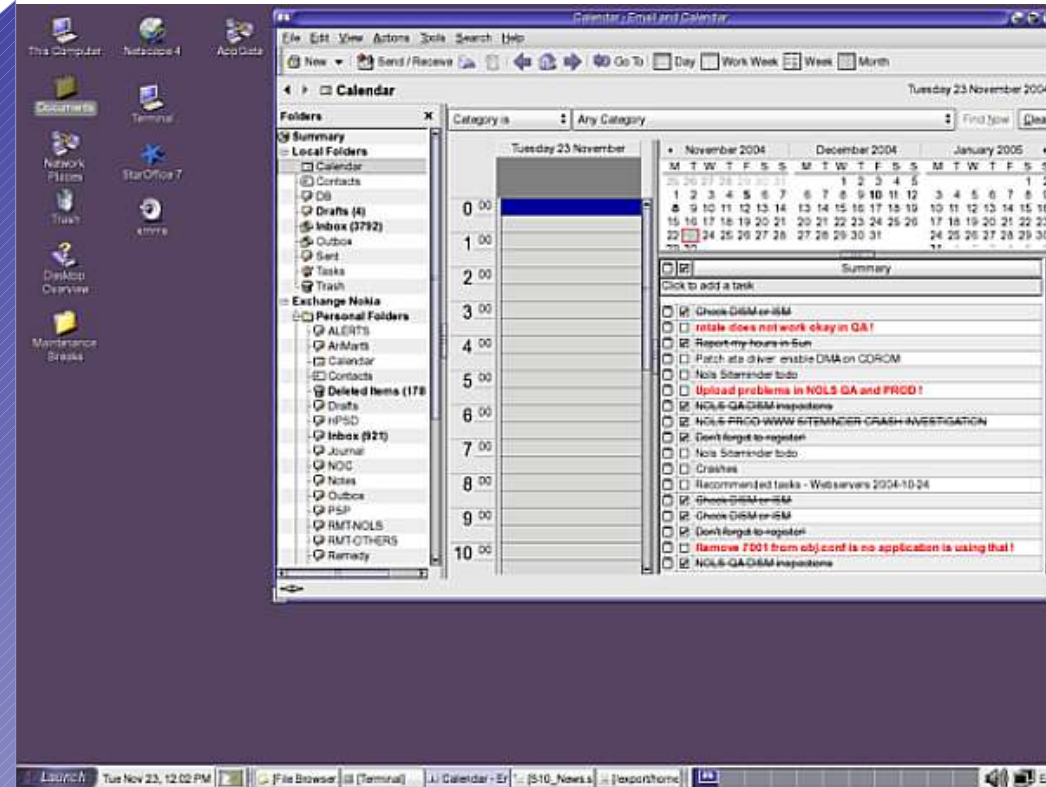
inetadm(1M), inetconv(1M)

# Agenda

- What's New ?
- N1 Grid Containers - Zones
- Dynamic Tracing - DTrace
- Predictive Self Healing - SMF
- **Java Desktop System - JDS**

# Java Desktop System - JDS

Open



StarOffice

Evolution

Mozilla

Java

GNOME Look and Feel

Solaris

Linux

Security

# More information ?

Sun BigAdmin site

<http://www.sun.com/software/javadesktopsystem/>

Man pages:

evolution(1)

jds-help(1), gnome-\*(1)



**Thank You !**

**Stefan Parvu**

**Client Solutions**

**[stefan.parvu@sun.com](mailto:stefan.parvu@sun.com)**

