ulm university universität ulm

Analyse und Monitoring mit DTrace

Thomas Nau, kiz (Thomas.Nau@uni-ulm.de)

©2015 Thomas Nau

Timeline

- Wer bin ich und was tu ich eigentlich
- Tool Historie
- DTrace (very) quick tour
- DTrace für Entwickler
- Use the Force, DTrace Nerd-Stuff
- wrap-up
- Q & A

©2015 Thomas Nau

Über mich

- eigentlich Physiker
- stlv. Direktor des kiz und Leiter der Abteilung „Infrastruktur“
  - Balanceakt zwischen Technik und Management
- erste IT Berührung mit einer PDP-11 vor langer Zeit
- Schwerpunkte UNIX und Storage Lösungen

©2015 Thomas Nau

Das tolle Team der Abteilung Infrastruktur

- One team to serve them all:
  erbringt Dienstleistungen für ~14.000 Personen
  - Netzwerke (LAN, MAN und WLAN)
    - Anbindung an BelWue mit 10Gbit demnächst 100Gbit
    - Telefonie mit 14.000 Anschlüssen
  - Rechnerbetrieb
    - gesamte zentrale Server-IT inklusive der Universitätsverwaltung
    - ca. 600 Desktop PCs und Notebooks mit Windows und Linux
  - Dienstleistungen im Rahmen von Landeskooperationen
    - Backup Service für mehrere Universitäten in Baden-Württemberg
    - HPC Landes-Cluster mit Schwerpunkt „Theoretische Chemie“
  - all die neuen Hypes (IaaS, Cloud, Fog, Rain, ...)

©2015 Thomas Nau

Die Vorteile

- ein Team mit einem Ziel unter einer Verantwortung
  - Team umfasst neben Rechnerbetrieb auch IT- Security und Netzwerk; großes Plus durch „reality check“
  - macht es einfacher Entscheidungen zu treffen und gemeinsam zu tragen
  - zeitkritische Projekte sind auf Grund der geringeren „inneren Reibung“ einfacher umzusetzen
- alle werden dazu ermutigt eigene Ideen zu entwickeln und sie im Gesamtkontext zu verfolgen

©2015 Thomas Nau

Safe Harbor Statement

Keines

©2015 Thomas Nau

Statistische Tools

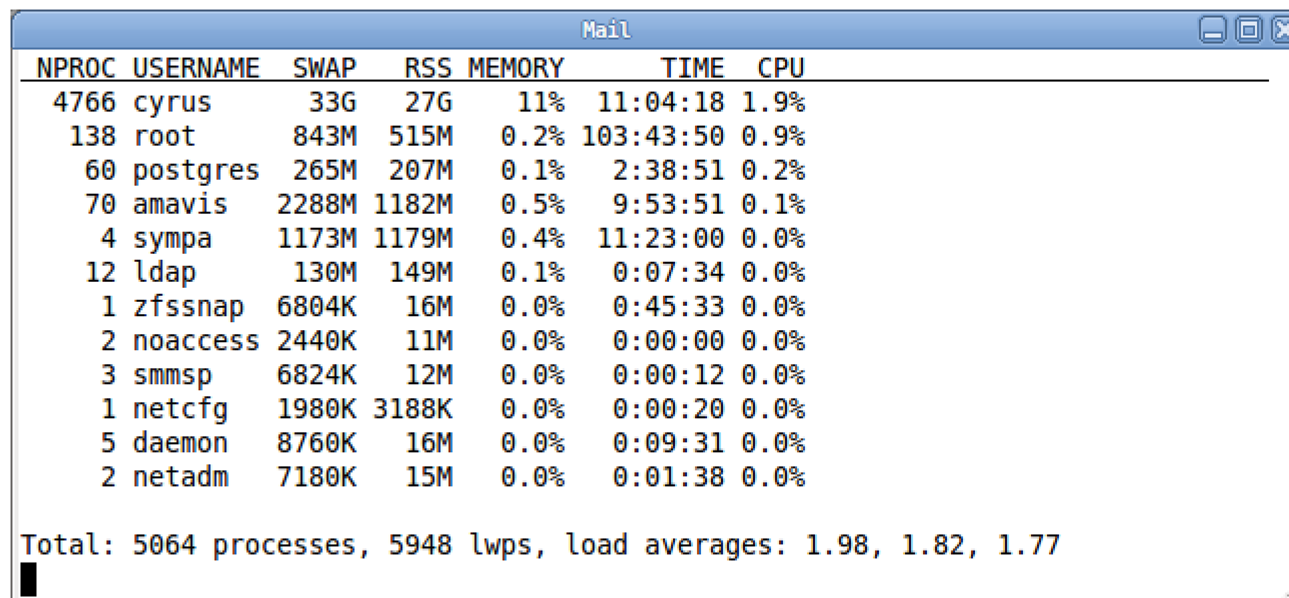
- liefern statistische Daten über Kernel Threads, Platten IO, CPU Last, virtuellen Speicher, …
  - *vmstat(1m), netstat(1m), kstat(1m), ...*
- liefern grobe Anhaltspunkte wo Engpässe zu finden sind
  - hoher Wert von „Parameter #1“

```
jedi# vmstat 5
 kthr      memory            page            disk          faults      cpu
 r b w   swap  free  re  mf pi po fr de sr m0 m1 m3 m1   in   sy   cs us sy id
 0 0 0 8620144 4617688 67 69 1  1  1  0  0  0  1  0  0 6147 1254 5529  0  1 99
 0 0 0 8702320 4730536 6  5  0  0  0  0  0  0  0  0  0 8010  532 7370  0  2 98
 0 0 0 8788856 4816776 6  1  0  0  0  0  0  0  0  0  0 7990  534 7428  0  2 98
^C
```


©2015 Thomas Nau

Sampling Tools

- z.B. liefert *prstat(1m)* viele Informationen über laufende Prozesse und deren Threads
  - guter Ausgangspunkt falls Anwendungsprobleme vermutet werden
  - Einschränkungen aus Basis des Nyquist-Shannon Abtasttheorems
    - kurze und Spitzen-Ereignisse wie sind oft nicht erfassbar (load vs. CPU)



NPROC	USERNAME	SWAP	RSS	MEMORY	TIME	CPU
4766	cyrus	33G	27G	11%	11:04:18	1.9%
138	root	843M	515M	0.2%	103:43:50	0.9%
60	postgres	265M	207M	0.1%	2:38:51	0.2%
70	amavis	2288M	1182M	0.5%	9:53:51	0.1%
4	sympa	1173M	1179M	0.4%	11:23:00	0.0%
12	ldap	130M	149M	0.1%	0:07:34	0.0%
1	zfssnap	6804K	16M	0.0%	0:45:33	0.0%
2	noaccess	2440K	11M	0.0%	0:00:00	0.0%
3	smmsp	6824K	12M	0.0%	0:00:12	0.0%
1	netcfg	1980K	3188K	0.0%	0:00:20	0.0%
5	daemon	8760K	16M	0.0%	0:09:31	0.0%
2	netadm	7180K	15M	0.0%	0:01:38	0.0%

Total: 5064 processes, 5948 lwps, load averages: 1.98, 1.82, 1.77

©2015 Thomas Nau

Event basierte Tools

- Beispiele *truss(1), gdb(1)*
- sind nicht dynamisch
  - Analyse vorübergehender oder kurzlebiger Probleme ist schwierig oder gar unmöglich
  - stoppt die Anwendung um Daten zu sammeln und beeinflusst dadurch besonders das Timing der Anwendung
- Auswertung zusammenhängender Prozesse schwierig oder gar unmöglich (Bsp.: user login)
- die Grenze zum Kernel kann meist nicht überschritten werden
  - Ausnahme sind kernel-debugger wie etwa *mdb(1m)*

©2015 Thomas Nau

„A process cannot be understood by stopping it. Understanding must move with the flow of the process, must join it and flow with it.“

„Erste Regel von Mentat“ aus „Dune – Der Wüstenplanet“

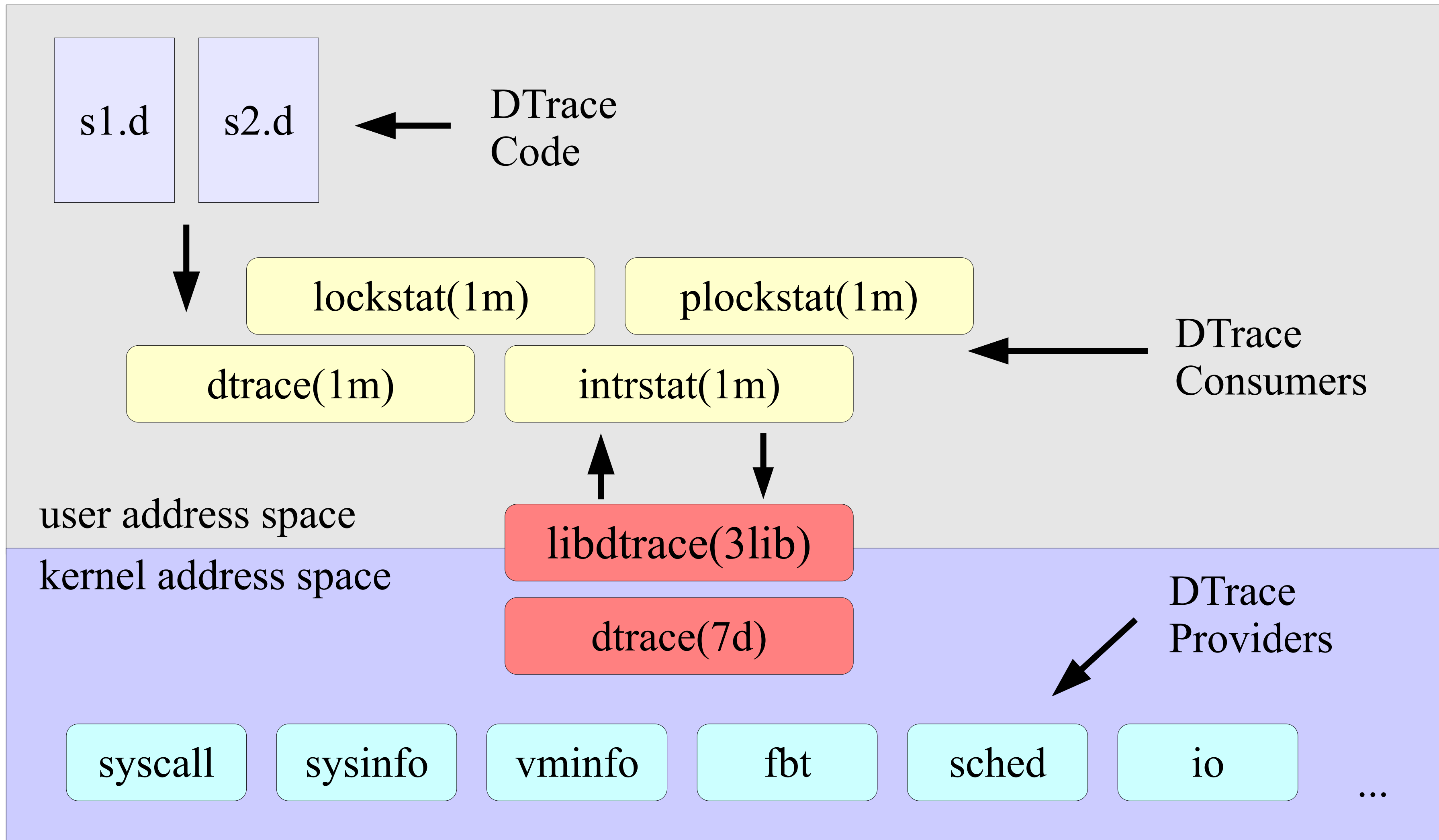
©2015 Thomas Nau

DTrace, die „Dynamic Tracing Facility“

- instrumentiert dynamisch und effizient Kernel- aber auch Bibliotheks- und Anwendungs-Code
- derzeit 100.000+ „Probes“ verfügbar
  - Zahl ist von geladenen Kernel-Modulen abhängig
  - ~90% sind Kernelfunktionen (*function boundary trace provider,* FBT)
  - lassen sich beliebig und effizient ein- bzw. ausschalten
- skriptfähig durch „C“ ähnliche Sprache „D“
- wird im Kernel Kontext ausgeführt
  - keine Schleifen (for, while, …) aus Sicherheitsgründen möglich
- viele Solaris Tools verwenden DTrace unter der Haube

©2015 Thomas Nau

DTrace Block Schaubild



©2015 Thomas Nau

Provider stellen Probes bereit

- Probes „Triggerpunkte“ die viele Aktionen auslösen können
  - Aufzeichnung von Kernel- oder User-Stacks sowie von Datenstrukturen oder Speicherinhalten
  - Manipulation von Daten etwa Rückgabewerten
  - Namens-Syntax für Probes

    provider:module:function:name

- Provider sammeln Daten spezifischer Bereiche, bereiten sie auf und stellen sie asynchron für Weiterverarbeitung bereit
  - proc: Informationen über Prozesse und Threads
  - syscall: Infromationen über Solaris system-calls
  - io: Disk-IO bezogene Probes

©2015 Thomas Nau

Beispiel: Datenaufbereitung

```
typedef struct ipinfo {
        uint8_t ip_ver;           /* IP version (4, 6)    */
        uint16_t ip_plength;      /* payload length       */
        string ip_saddr;          /* source address       */
        string ip_daddr;          /* destination address  */
} ipinfo_t;

typedef struct ipv4info {
    ...
    uint8_t ipv4_protocol;        /* next level protocol  */
    string ipv4_protostr;         /* same as a string     */
    ...
    ipaddr_t ipv4_src;            /* source address       */
    ipaddr_t ipv4_dst;            /* destination address  */
    string ipv4_saddr;            /* src address, string  */
    string ipv4_daddr;            /* dest address, string */
    ...
} ipv4info_t;
```


©2015 Thomas Nau

„D“ Sprachstruktur

- „D“ definiert eine Art **„Unterprogramm-Sammlung“**
- einfache Struktur, „Bedingungen“ sind optional

```
Proben-Namen
/ Bedingung /
{  Aktionen  }
```

- aus Sicherheitsgründen kennt „D“ keine Schleifen, …
- Typen, Datenstrukturen, Operatoren und Funktionen sind stark an „C“ angelehnt
  - viele vordefinierte und initialisierte Variable verfügbar
  - Variable müssen nicht vor Verwendung definiert werden

©2015 Thomas Nau

„D“ Sprachstruktur Beispiel

```
obi-wan# cat ./reads.d
#!/usr/sbin/dtrace -s

#pragma D option quiet

syscall::read:entry
/ execname == "bacula-fd" || execname == "nscd" /
{
    printf("%-16s %10s %s\n",
        execname, probefunc, fds[arg0].fi_pathname
    );
}
```

```
obi-wan# ./reads.d
bacula-fd    read      /backup/mail/imap/l/user/PRIVACY/53065.
bacula-fd    read      /backup/mail/imap/l/user/PRIVACY/53065.
nscd         read      /etc/passwd
```


©2015 Thomas Nau

DTrace's Warp Antrieb: Aggregations

$$f(f(x_0) \cup f(x_1) \cup \ldots \cup f(x_n)) = f(x_0 \cup x_1 \cup \ldots \cup x_n)$$

Aggregations nutzen eine besondere mathematische Eigenschaft bestimmter Funktionen aus

©2015 Thomas Nau

Beispiel: *sum()* Aggregation

$$\Sigma\ 1, 2, 3, \ldots 99, 100 = 5050$$

$\Sigma\ 1, \ldots 10$	$= 55$
$\Sigma\ 11, \ldots 20$	$= 155$
...	
$\Sigma\ 91, \ldots 100$	$= 955$

$$\Big\} \ \Sigma = 5050$$

$$\sum_{n=1}^{100} n = \sum \left(\sum_{n=1}^{10} n, \sum_{n=11}^{20} n, \ldots \sum_{n=90}^{100} n \right)$$

©2015 Thomas Nau

Aggregations

- halten den Speicherbedarf gering
  - keine Notwendigkeit alle Daten zwischen zu speichern
  - Probleme mit der Skalierung werden vermieden
  - nahezu beliebiger Index etwa *ustack()* oder *execname, pid, ...*

Aggregation	Aufgabe
count	zählt die Zahl der Aufrufe
sum	Gesamtsumme der Ausdrücke
min, max	kleinster und größter Wert der Ausdrücke
avg, stddev	arithmetisches Mittel und Standardabweichung
lquantize	lineare Verteilung
quantize	2^n Verteilung
llquantize	log-lineare Verteilung (ab Solaris 11.2)

©2015 Thomas Nau

Hilfreiche Funktionen für Aggregations

- *trunc(@aggr [, n])*
  löscht eine Aggregation oder Teile davon

  n > 0 die obersten n Einträge bleiben erhalten

  n < 0 die untersten n Einträge bleiben erhalten

- *clear(@aggr)*
  setzt die Werte einer Aggregation auf 0
- *normalize(@aggr, val)*
  dividiert alle Werte durch *val*
- *denormalize()*
  macht die Normierung rückgängig
- **Tipp:** im Zusammenspiel mit der *tick* probe lassen sich einfach top-ten artige Ausgaben realisieren

©2015 Thomas Nau

Sortierung von Aggregations

- die Sortierung lässt sich über Optionen steuern

```
setopt("aggsortkey", true);
```

- Möglichkeiten

aggsortkey	Sortierung nach Index, nicht Wert
aggsortkeypos	Nummer des Index nach dem sortiert wird
aggsortrev	Umkehrung der Reihenfolge

©2015 Thomas Nau

Beispiel: physikalischer IO, Bandbreite (1)

```
BEGIN {
    setopt("aggsortkey", "true");   /* sort by index  */
    setopt("aggsortkeypos", "1");   /* sort by device */
    ts = timestamp;
}

io:::start  /* trace physical read-IO */
/ args[0]->b_flags & B_READ /
{
    @r[execname, args[1]->dev_statname] =
        sum(args[0]->b_bcount);
}

io:::start  /* trace physical write-IO */
/ args[0]->b_flags & B_WRITE /
{
    @w[execname, args[1]->dev_statname] =
        sum(args[0]->b_bcount);
}
```


©2015 Thomas Nau

Beispiel: physikalischer IO, Bandbreite (2)

```
/* make use of first commandline argument set rate */
tick-$1 {
        /* normalize to seconds and kilo-bytes
         * remember, dtrace times are nanoseconds */
    factor = 1024 * (timestamp - ts) / 1000000000;
    normalize(@r, factor);
    normalize(@w, factor);

        /* print headera and data then reset counters
         * and store new timestamp *(
    printf("\n%-16s %-10s %16s %16s\n",
        "executable", "device",
        "read[kb/s]","write[kb/s]");
    printa("%-16s %-10s %@16d %@16d\n", @r, @w);
    clear(@r);
    clear(@w);
    ts = timestamp;
}
```


©2015 Thomas Nau

Beispiel: physikalischer IO, Bandbreite (3)

```
obi-wan# ./demo_iostat.d 5s
executable          device                  read[kb/s]       write[kb/s]
zpool-rpool         sd189                            0                47
postgres            sd2                            762                 0
postgres            sd254                         1335                 0
postgres            sd255                         1213                 0
zpool-rpool         sd4                              0                47
```

Sortierung

©2015 Thomas Nau

Beispiel: physikalischer IO, Latenz (1)

```
/* keep start time per device and scheduled
 * starting block (more than one IO might be in flight)
 */
io:::start
{

    start[args[0]->b_edev, args[0]->b_blkno] =
        timestamp;
}

/* tick-provider fires on single CPU to trigger
 * output of data; makes use of first commandline
 * argument to set rate
 */
tick-$1 {
    printa("%@d\n", @q);
    clear(@q);
}
```

 ©2015 Thomas Nau

Beispiel: physikalischer IO, Latenz (2)

```
/* only handle IOs for which we have a start time */
io:::done
/start[args[0]->b_edev, args[0]->b_blkno]/
{
        /* use efficient "this" variables to set
         * read/write flag and convert nano- into
         * microseconds
         */
    this->rwf = args[0]->b_flags & B_WRITE ? "W" :
                (args[0]->b_flags & B_READ  ? "R" : "*");
    this->us  = (timestamp - start[args[0]->b_edev,
                 args[0]->b_blkno]);
    this->us /= 1000;
    @q[this->rwf] = quantize(this->us);

        /* clear start time flag */
    start[args[0]->b_edev, args[0]->b_blkno] = 0;
}
```


©2015 Thomas Nau

Beispiel: physikalischer IO, Latenz (3)

```
jedi# ./demo_iolat.d 60s
...
R
           value  ------------- Distribution ------------- count
              32 |                                         0
              64 |                                         1
             128 |@@@@@@@@@@@@@@@@@                        759
             256 |@@@@@                                    242
             512 |                                         2
            1024 |                                         0
            2048 |                                         2
            4096 |@                                        33
            8192 |@@@@@@@@                                 340
           16384 |@@@@@@                                   272
           32768 |@@                                       73
           65536 |@                                        25
          131072 |                                         8
          262144 |                                         5
          524288 |                                         4
         1048576 |                                         0
```


©2015 Thomas Nau

„Neuartige“ Performance-Probleme

- Ungleichgewicht bei der Lastverteilung von Threads auf CPU cores
- NUMA (Non-Uniform Memory Access)
  - bei heutigen Systemen mit 2 und mehr Sockeln quasi inhärent
  - kann zu nicht deterministischer Performance führen
    - „first touch“
- TLB (translation look-aside buffer)
  - Übersetzungstabelle virtuelle → physikalische Speicherseiten
  - durch Hardware begrenzt, Problem mit viele kleinen „pages“ und random Zugriffen

©2015 Thomas Nau

pid-provider, DTrace für Anwendungsentwickler (1)

```
/* CPU time when calling a routine which matches
 * patterns passed on the command line
 */
pid$target:$1:$2:entry {
    self->ts = vtimestamp;
}

/* do the bookkeeping if we stored data on entry
 */
pid$target:$1:$2:return
/ self->ts /
{
    self->delta = vtimestamp -self->ts;
    @total[probemod, probefunc]        =
        sum(self->delta);
    @thread[tid, probemod, probefunc] =
        sum(self->delta);
    self->ts = 0;
}
```


©2015 Thomas Nau

pid-provider, DTrace für Anwendungsentwickler (1)

```
obi-wan# OMP_NUM_THREADS=2 ./lib_timing.d -c './partest 100
10' libmtsk ''
...
Timing total, top-10 only
 time [us]              module:function
      3837        libmtsk.so.1:thread_cancel_point
      3923        libmtsk.so.1:getfsr
      3959        libmtsk.so.1:push_context
      4431        libmtsk.so.1:barrier_reset_nthreads
      4532        libmtsk.so.1:getpsr
      4693        libmtsk.so.1:package_a_task
      5122        libmtsk.so.1:pop_context
      7433        libmtsk.so.1:_omp_affinity_mode
     35567        libmtsk.so.1:ready_to_work
     60326        libmtsk.so.1:sleep_at_barrier
```

„Verbrauch“ an CPU Zeit
deutet auf spin-wait

©2015 Thomas Nau

pid-provider, DTrace für Anwendungsentwickler (1)

```
obi-wan# OMP_NUM_THREADS=4 ./lib_timing.d -c ./transpose
transpose ''
...
Timing per thread, top-10 only
 time [us]    TID        module:function
        5      1     transpose:_start
      269      1     transpose:init_misaligned_data_trap_handler
   459892      1     transpose:_$d1A17.MAIN_
   462289      4     transpose:_$d1A17.MAIN_
   464534      3     transpose:_$d1A17.MAIN_
   466945      2     transpose:_$d1A17.MAIN_
  1515176      4     transpose:_$d1B27.MAIN_
  4265414      3     transpose:_$d1B27.MAIN_
  6830205      2     transpose:_$d1B27.MAIN_
  9938128      1     transpose:_$d1B27.MAIN_
...
```

„Lastverteilung“ auf die 4 Threads im Verhältnis

44 : 30 : 19 : 7

Dank an Dieter an Mey von der RWTH Aachen für die Bereitstellung der Fortran-90 Codes

©2015 Thomas Nau

„Neuartige“ Performance-Probleme

- Analyse oft hochgradig Platform spezifisch
- DTrace kann auf Hardware Counter der CPUs zugreifen
  - PAPI (Performance Application Programming Interface)
  - auch Platform spezifische Zähler

```
T4# dtrace -n 'cpc:::DTLB_fill_trap-all-10000 \
               { @[execname] = count(); }'
dtrace: description ' cpc:::DTLB_fill_trap-all-10000 ' matched
1 probe
^C
  sshd                                                          2
  ldmd                                                          6
  gzip                                                         27
  zpool-ssd                                                    35
  postgres                                                     36
  sched                                                        70
  tar                                                         106
```


©2015 Thomas Nau

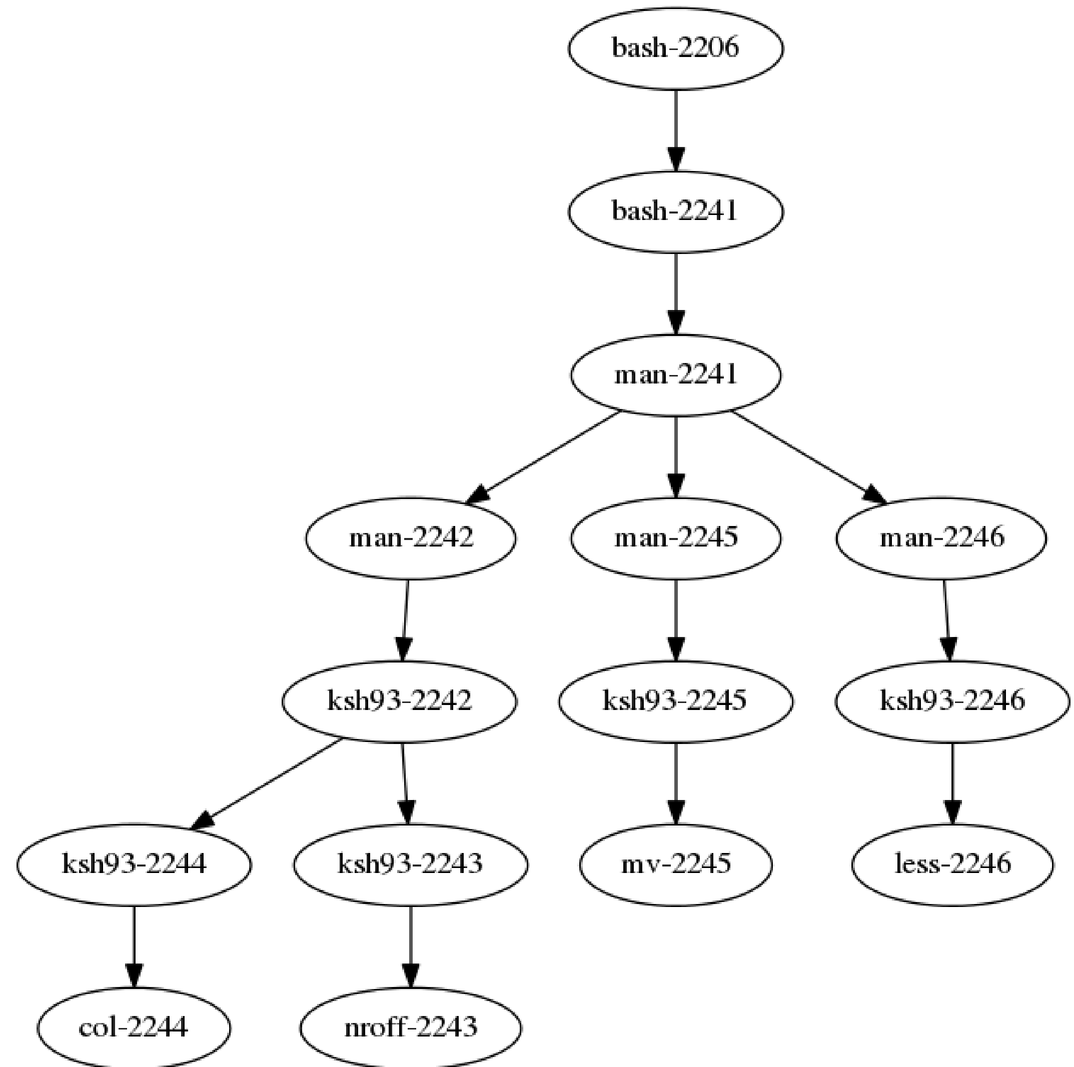
„Neuartige“ Performance-Probleme

- *sched* und *proc* Provider decken u.a. folgende Aspekte ab
  - Prozess- und Thread-Erzeugung
  - CPU scheduling
  - locality groups (LG)
- damit lassen sich Thread Migrationen zwischen cores oder LGs aufzeichnen
  - sprengt den Umfang für heute
- Prozessketten sind einfach verfolgbar
  - visualisiert mit *dot(1)* aus dem *GraphViz* Paket
  - DTrace erzeugt Input

©2015 Thomas Nau

fork() / exec() bei Aufruf von *man ls*

```
digraph ExecPaths {
   "bash-2206"  -> "bash-2241";
   "ksh93-2244" -> "col-2244";
   "ksh93-2243" -> "nroff-2243";
   "man-2242"   -> "ksh93-2242";
   "ksh93-2242" -> "ksh93-2243";
   "ksh93-2242" -> "ksh93-2244";
   "man-2245"   -> "ksh93-2245";
   "ksh93-2245" -> "mv-2245";
   "man-2246"   -> "ksh93-2246";
   "ksh93-2246" -> "less-2246";
   "man-2241"   -> "man-2242";
   "man-2241"   -> "man-2245";
   "man-2241"   -> "man-2246";
   "bash-2241"  -> "man-2241";
}
```



©2015 Thomas Nau

Jedi Power

©2015 Thomas Nau

The tough stuff, Jedi power required

- Fehler tritt nur sporadisch auf (1:10000)
  - Datenflut sofern alle Ereignisse aufgezeichnet werden
- Problem läßt sich erst im nach hinein als solches identifizieren
  - Bsp.: ungewöhnlich lange dauernder IO („Ausreißer“)
  - oft sind zur Analyse notwendige Daten dann nicht mehr verfügbar
- DTrace Speculations helfen
  - sammeln Daten „spekulativ“
  - später wird über Bereitstellung an consumer entschieden

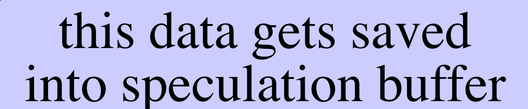
©2015 Thomas Nau

DTrace Speculations (1)

```
#pragma D option nspec=16

BEGIN
{
    us_fac = 1000;
    limit  = $1 *us_fac;     /* limit on command line */
}

/* no tracing of stdin, stdout and stderr */
syscall::write:entry
/ arg0 >= 3 /
{
    self->ts    = timestamp;
    self->spec = speculation();
    speculate(self->spec);
    printf("%-9s %6d bytes FD %2d (%s):   ",
        execname, arg2, arg0, fds[arg0].fi_pathname);
}
```



 ©2015 Thomas Nau

DTrace Speculations (2)

```
/* for successful but slow operations we
 * end up here */
syscall::write:return
/ self->spec && arg0 != 1 &&
  (timestamp -self->ts) >= limit /
{
    speculate(self->spec);
    printf("%dus\n", (timestamp -self->ts) / us_fac);
    commit(self->spec);
    self->spec = 0;
}

/* discard collected data in any other case */
syscall::write:return
/ self->spec /
{
    discard(self->spec);
    self->spec = 0;
}
```


©2015 Thomas Nau

DTrace Speculations (3)

```
jedi# ./demo_spec.d 10000
imapd        96 bytes FD 17 (/mail/.../cyrus.index):        167131us
lmtpd      1512 bytes FD 20 (/mail/.../cyrus.cache):         33896us
postgres   8192 bytes FD 10 (/mail/postgres/.../16418):      12605us
lmtpd      1360 bytes FD 20 (/mail/.../cyrus.cache):         24313us
lmtpd      1360 bytes FD 20 (/mail/.../Mizar/cyrus.cache):   28209us
sendmail   5120 bytes FD 23 (<unknown>):                     33377us
postgres  16384 bytes FD 30 (/mail/postgres/.../pg_xlog/...):78915us
...
```

Die Dateinamen wurden für die Folie gekürzt

©2015 Thomas Nau

DTrace Lightsaber

©2015 Thomas Nau

DTrace "Destructive Actions"

- einige Befehle von DTrace können den Zustand von Prozessen oder des Kernel ändern
  - auf Prozess Ebene
    - *stop(), raise(), copyout(), copyoutstr(), system()*
  - auf Kernel Ebene
    - *breakpoint(), panic(), chill()*
- aus Sicherheitsgründen müssen diese Operationen bei Bedarf explizit freigeschaltet werden
  - "-w" Kommandozeilen Option
  - *option destructive* Pragma

©2015 Thomas Nau

DTrace "Destructive Actions" (1)

- Manipulation von Rückgabewerten des Kernels kann hilfreich sein falls
  - „installer“ unbedingt auf riesigen swap-space beharren weil auch viel Hauptspeicher vorhanden ist
  - „installer“ die kompatible Betriebssystem Version nicht unterstützen wollen
    - das kiz ist Solaris 12 Platinum Site

```
case `uname -r` in
    5.10) /usr/sbin/svccfg -v import $file ;;

    5.11) svcadm restart manifest-import ;;

    *) echo "SMF Manifest not available on OS version `uname -r`"
       exit 1 ;;
esac
```


©2015 Thomas Nau

DTrace "Destructive Actions" (2)

```
struct utsname  uts;

syscall::uname:entry
{
    self->buf = arg0;
}

syscall::uname:return
/ self->buf /
{
    this->p = (struct utsname *) copyin(self->buf,
               sizeof(struct utsname));
    bcopy("neo", this->p->nodename, sizeof(uts.nodename));
    bcopy("5.11", this->p->release, sizeof(uts.release));
    copyout(this->p, self->buf, sizeof(struct utsname));
    self->buf = 0;
}
```


©2015 Thomas Nau

DTrace "Destructive Actions" (3)

```
trinity# uname -a
SunOS trinity 5.12 ****** i86pc i386 i86pc

trinity# ./demo_uname.d
dtrace: script './demo_uname.d' matched 2 probes
dtrace: could not enable tracing: Destructive actions not
allowed


trinity# ./demo_uname.d -w &
[1] 8630
dtrace: script './demo_uname.d' matched 2 probes
dtrace: allowing destructive actions

trinity:.../~# uname -a
SunOS neo 5.11 ****** i86pc i386 i86pc
```


©2015 Thomas Nau

Danke für die Aufmerksamkeit

...

©2015 Thomas Nau