DTrace - Die Antwort auf (fast) alle Fragen

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

Kommunikations- und Informationszentrum (kiz)

- Die Aufgaben der „Abteilung Infrastruktur“ umfassen u.a.
  - Azubi Ausbildung
  - Cluster basierende Universitäts weite Mail-, LDAP-, Portal-, Datenbank- und File-Services, ...
  - Betreuung von ca. 600 Desktop und Laptop Arbeitsplätzen
    - 25% Linux, 75% Windows
  - Backup Service (Bacula Enterprise) für mehrere Universitäten in Baden-Württemberg
  - Landes HPC Cluster mit Schwerpunkt „Theoretische Chemie“
  - 4 lokale Netzwerke plus flächendeckendes Campus WLAN und MAN im Ulmer Stadtbereich
  - Telefonanlage mit ca. 14.000 Anschlüssen unter Einsatz von VoIP und 2-Draht Technik

The early days

Analyse / Debugging

==

Logic Analyzer

+

Schaltpläne

Aus den 90ern

vmstat(1m)

- liefert statistische Daten über Kernel Threads, Platten IO, CPU Last, virtuellen Speicher und traps
- liefert nur wenige Anhaltspunkte wo Engpässe zu finden sind
  - hohe Anzahl von system-calls oder context-switches
  - viele page-in oder page-out Ereignisse

```
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
```

prstat(1m)

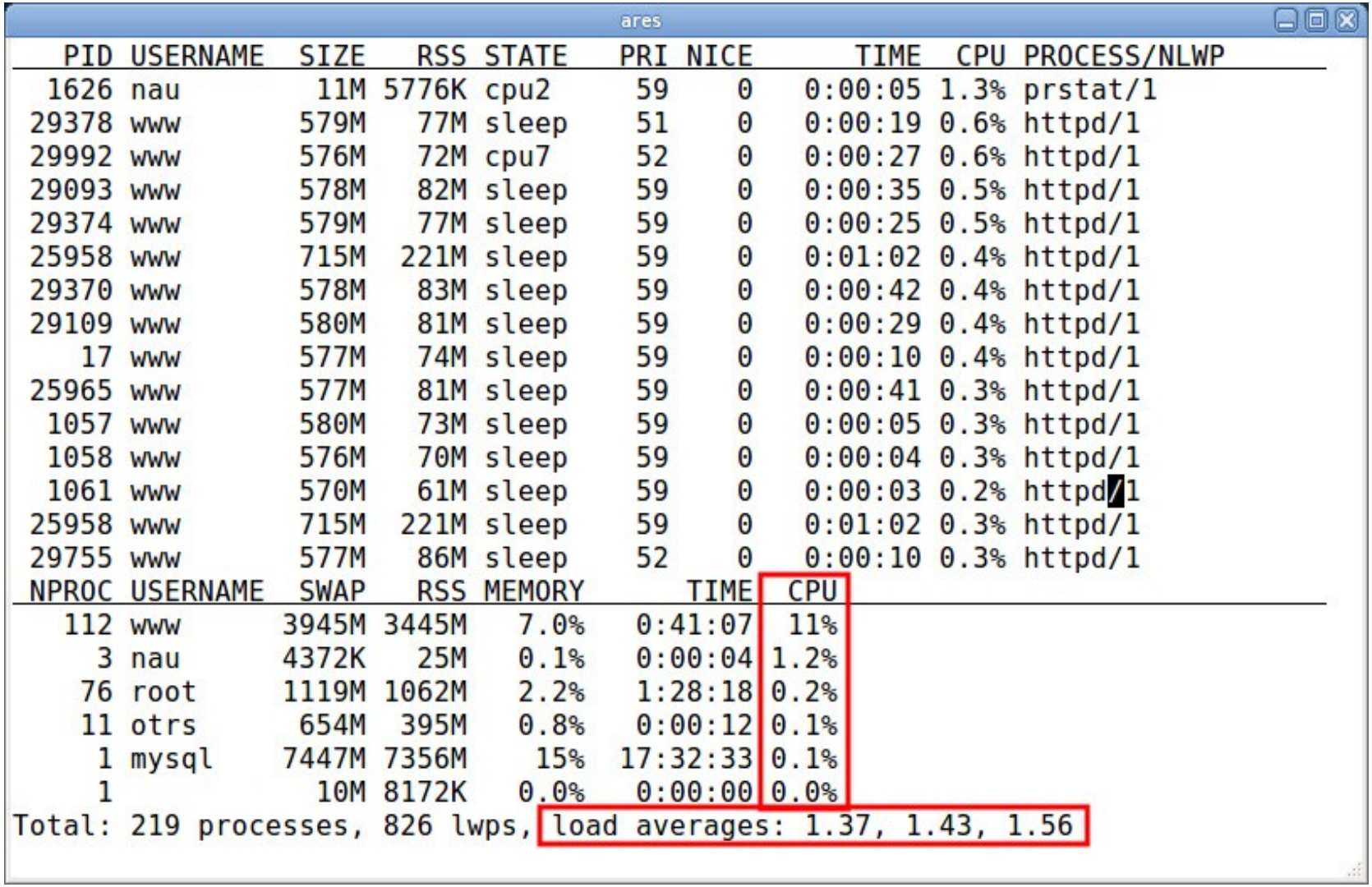
- liefert viele Informationen über laufende Prozesse und deren Threads
  - guter Ausgangspunkt falls Anwendungsprobleme vermutet werden

```
jedi# prstat -Lm
   PID USERNAME USR SYS TRP TFL DFL LCK SLP LAT VCX ICX SCL SIG PROCESS/LWPID
 16480 xvm      6.9 9.8 0.0 0.0 0.0  27  54 1.8 30K 114 .2M   0 qemu-dm/3
   363 xvm      0.1 0.2 0.0 0.0 0.0 0.0 100 0.0   4   1  2K   0 xenstored/1
 16374 root     0.0 0.1 0.0 0.0 0.0 100 0.0 0.0  10   0  1K   0 dtrace/1
  1644 xvm      0.1 0.1 0.0 0.0 0.0  33  66 0.0 569   7 835   0 qemu-dm/3
  2399 root     0.0 0.1 0.0 0.0 0.0 0.0 100 0.0  49   0 388   0 sshd/1
 16376 root     0.0 0.1 0.0 0.0 0.0 0.0 100 0.0  38   0 297   0 prstat/1
 11705 xvm      0.0 0.1 0.0 0.0 0.0  50  50 0.0 576  15 858   0 qemu-dm/4
 16536 root     0.0 0.1 0.0 0.0 0.0 0.0 100 0.0  48   0 286   0 vncviewer/1
```

prstat(1m)

- Anhaltspunkte:
  - USR+SYS — CPU-Zeit — Effizienz?
  - LAT — Latenz — Zu wenig CPUs?
  - DFL — Data Page Fault — Zu wenig Speicher?
  - SLP — Sleep — Auf was wird gewartet?
  - LCK — Locks — Wer sperrt was/warum?

prstat(1m) Erklärungsnot



truss(1)

- *truss(1)* ist 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 Kernel-Grenze kann nicht überschritten werden

kstat(1m)

- *kstat(1m)* ermöglicht Zugriff auf alle statistischen Daten des Solaris Kernels
- C und Perl-API verfügbar

```
obi-wan# ipadm show-addr imaprepl0
ADDROBJ           TYPE     STATE        ADDR
imaprepl0/v4      static   ok           134.60.1.28/24

obi-wan# kstat -p ::imaprepl0:obytes64 ::imaprepl0:rbytes64 1 2
link:0:imaprepl0:obytes64       254797953722
link:0:imaprepl0:rbytes64       391165973353

link:0:imaprepl0:obytes64       254797955526
link:0:imaprepl0:rbytes64       391165984296
```

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

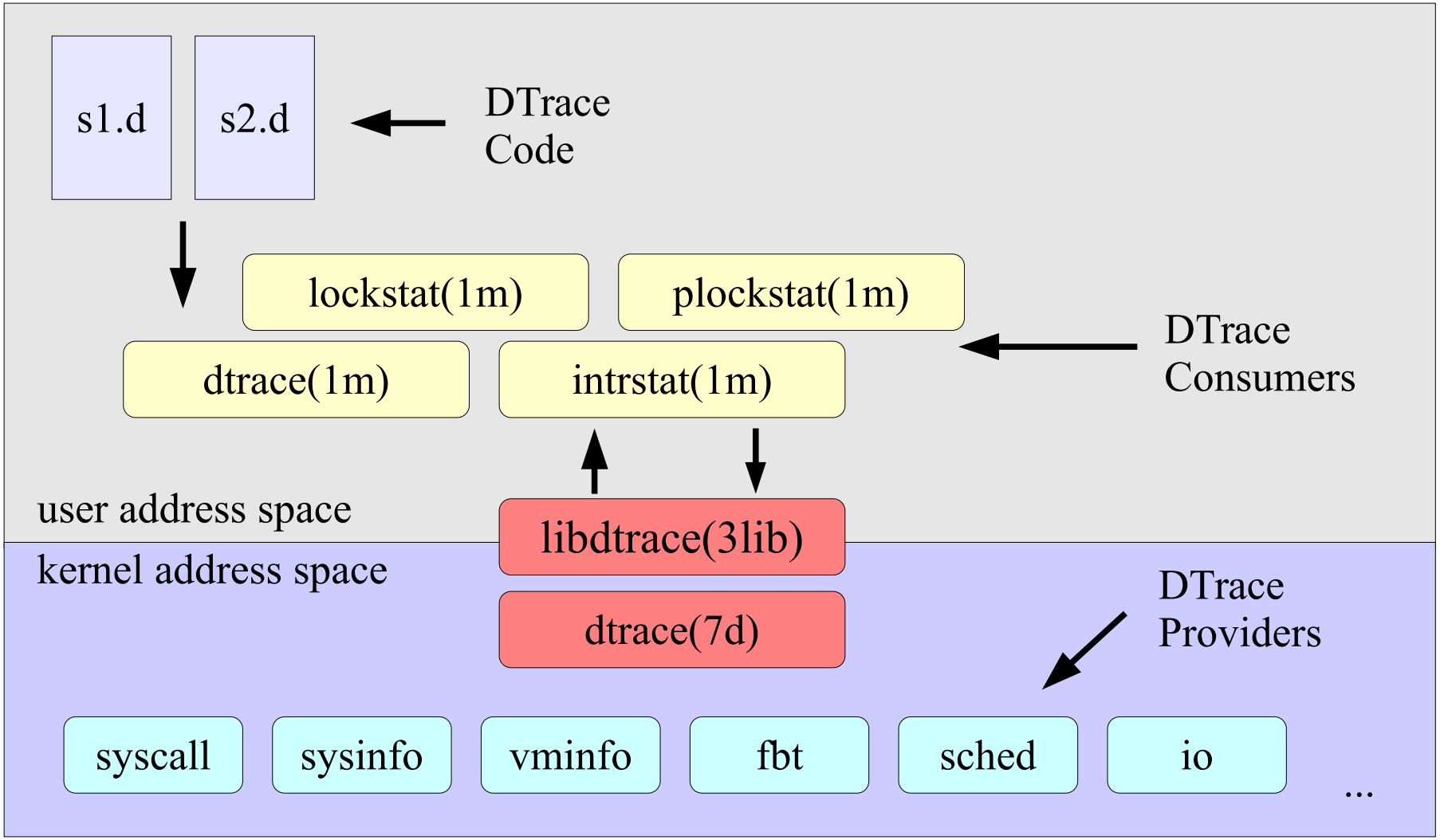
„Erste Regel von Mentat“ aus „Dune – Der Wüstenplanet“

Eine Frage des Tools...

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

DTrace Block Schaubild



Provider

- stellen sogenannte „Probes“ zur Verfügung
- sammeln Daten, bereiten sie sinnvoll auf und stellen sie in Puffern für asynchrone Weiterverarbeitung bereit
- decken spezifische Bereiche ab z.B.
  - proc: Informationen über Prozesse und Threads
  - syscall: Informationen über Solaris system-calls
  - io: Disk-IO bezogene Probes
- Updates bringen neue Provider
  - Solaris 11: ip, tcp, NFSv3, NFSv4, iSCSI, …
  - Solaris 11.2 DTrace Dokumentation: *http://docs.oracle.com/cd/E36784_01/html/E36846/*

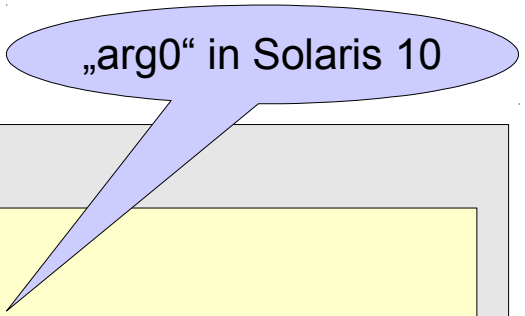
Probes

- „Triggerpunkte“ die vielfältige Aktionen auslösen können
  - Aufzeichnung von Kernel- oder User-Stacks
  - Aufzeichnung von Datenstrukturen oder Speicherinhalten
  - Manipulation von Daten oder Speicherinhalten
  - ...
- Namens-Syntax für Probes

  provider:module:function:name

  sched:unix:resume:off-cpu

Beispiel: wer öffnet welche Datei?




```
obi-wan# dtrace -q -n \
  'syscall::open*:entry {
     printf("%-20s %s\n", execname, copyinstr(arg1));
}'
```

```
httpd                /www/htdocs/guc/bilder/grafik/m_kontakt.gif
httpd                /www/cms/uploads/pics/rw_logo2_03.png
mysqld               /www/mysqldata/cms/fe_users.MYI
mysqld               ./cms/fe_users.MYD
mysqld               /www/mysqldata/cms/fe_groups.MYI
...
```

„D“ Sprachstruktur

- „D“ definiert eine Art „Unterprogramm-Sammlung“
- einfache Struktur die sequentiell von oben nach unten ausgewertet wird; „Bedingungen“ sind optional

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

- aus Sicherheitsgründen kennt „D“ keine Schleifen, …
- einfach auch in eigene Anwendungen integrierbar
  - perl, MySQL, ...

Variable, Typen und Operatoren

- „C look and feel“ mit geringen Unterschieden

  int8_t, uintptr_t, ...

- Skalare, Strings, Zeiger, *structs* und *unions* sind verfügbar
  - provider und consumer laufen in unterschiedlichen Adressräumen
- Variable müssen nicht vor Verwendung definiert werden
  - viele schon vordefiniert: *pid, uid, execname, curcpu, …*
  - lesender Zugriff auf Kernelvariable möglich: *`kmem_flags*
- grundlegende Arithmetik-, Logik- und Vergleichs-Operatoren stehen zur Verfügung
- Vergleichs-Operatoren können auf Strings angewendet werden; liefern 1 im Falle von Gleichheit sonst 0

Ausgabe bzw. Sammeln von Daten

- printf() arbeitet wie in „C“ mit Format-Strings
  - nützliche Erweiterungen verfügbar
    - 'a' Ausgabe eines Zeigers als Kernel Symbol
    - 'p' Hexadezimal Ausgabe eines Zeigers
    - 'S' Ausgabe von Strings wobei nicht druckbare Zeichen mit „\“ dargestellt werden
    - 'Y' formatiert ein Argument „Nanosekunden seit 1.1.1970“ als für Menschen verständlichen String
  - *printa()* arbeitet analog zu *prinf()* mit “aggregations“
- *stack(), ustack()*
- *tracemem()*

Beispiel: alle Lesezugriffe auf Dateien im System

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

#pragma D option quiet

syscall::read:entry {
    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.
hwmgmtd          read       /tmp/hmptemp/ssmlibipmitool_stdout_Bnb3b
sshd             read       /devices/pseudo/clone@0:ptm
nscd             read       /etc/passwd
```

DTrace's Warp Antrieb: Aggregations

f(f(x0) U f(x1) U ... U f(xn)) = f(x0 U x1 U ... U xn)

Aggregations nutzen eine besondere mathematische Eigenschaft bestimmter Funktionen aus

Beispiel: *sum()* Aggregation

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

$$\sum 1, \ldots 10 = 55$$

$$\sum 11, \ldots 20 = 155$$

...

$$\sum 91, \ldots 100 = 955$$

$$\Big\} \sum = 5050$$

DTrace's Warp Antrieb: Aggregations

- halten den Speicherbedarf gering
  - keine Notwendigkeit alle Daten zwischenzuspeichern
  - Probleme mit der Skalierung werden vermieden
- derzeit
  - count(), sum()
  - min(), max(), avg(), stddev()
  - quantize(), lquantize()
  - ab Solaris 11.2: llquantize()
- der Index von Aggregations ist nahezu beliebig, etwa *ustack()* und *stack()* oder *execname, pid, ...*

Beispiel: Solaris 11 IP Provider Probes

- send — Solaris Netzwerk Stack verschickt ein IP Paket
- receive — Solaris Netzwerk Stack empfängt ein IP Paket

- der IP Provider übergibt Daten über 6 Pointer auf:

pktinfo_t *	csinfo_t *	ipinfo_t *
ifinfo_t *	ipv4info_t *	ipv6info_t *

ipinfo_t und *ipv4info_t* Definitionen

```
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;
```

Beispiel: Verteilung der IP Paketgröße

- basiert auf Beispiel von

  https://wikis.oracle.com/display/DTrace/ip+Provider

- gut geeignet für IP basierte File- oder Storage Server

```
#!/usr/sbin/dtrace -s

#pragma D option quiet

ip:::send
{
    @ipstats[args[2]->ip_daddr, execname] =
        quantize(args[2]->ip_plength);
}
```

2^n Verteilung

Beispiel: Verteilung der IP Paketgröße

```
...
192.168.23.23                                                  nfsd

     value  ------------- Distribution ------------- count
        16 |                                         0
        32 |                                         1
        64 |                                         0
       128 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@        244
       256 |@@@@@@@@@@@                              88
       512 |                                         0
      1024 |@@@@                                     32
      2048 |                                         0
...
```

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

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

Beispiel: „Top-10“ Netzverkehr einzelner Clients

```
#!/usr/sbin/dtrace -s

#pragma D option quiet

dtrace:::BEGIN {
    ts = timestamp;
}

ip:::send {
    @ipstats[args[2]->ip_daddr, probename] =
        sum(args[2]->ip_plength);
}

ip:::receive {
    @ipstats[args[2]->ip_saddr, probename] =
        sum(args[2]->ip_plength);
}
```

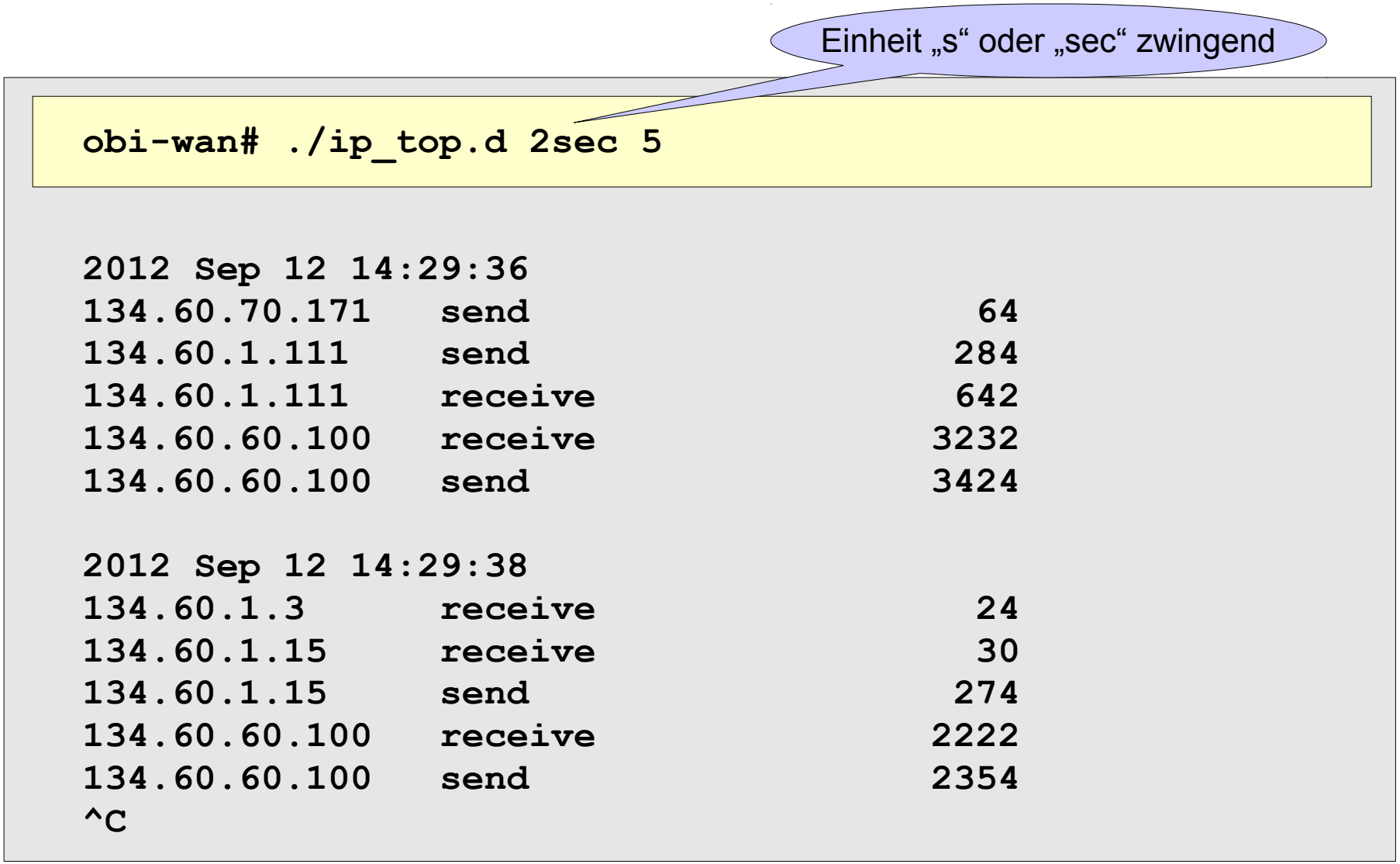
Beispiel: „Top-10“ Netzverkehr einzelner Clients

```
/* output collected data
 */
tick-$1 {
        /* top-n only */
    trunc(@ipstats, $2);

        /* print per second rate */
    normalize(@ipstats,(timestamp-ts) / 1000000000);
    printf("\n%Y\n", walltimestamp);
    printa("%-15s %-10s %@15d\n", @bytes);

        /* start next interval from scratch */
    ts = timestamp;
    trunc(@ipstats);
}
```

Beispiel: „Top-10“ Netzverkehr einzelner Clients

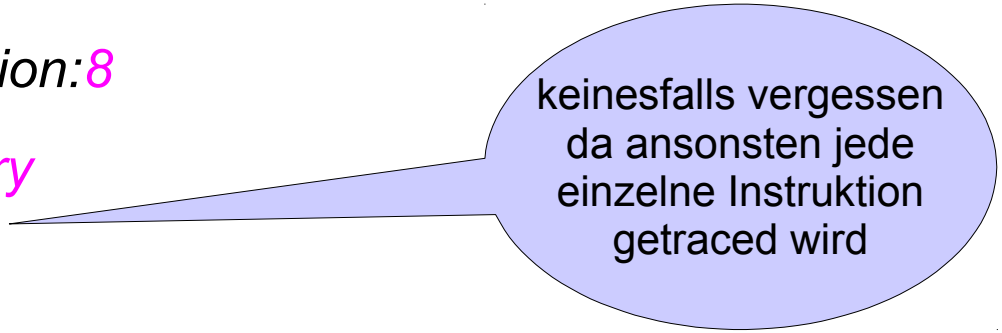


Der *pid* Provider

- *der pid* Provider erlaubt die Verfolgung von Funktionen oder Instruktionen einer Anwendung
  - Einschränkung: „inlining“ durch den Compiler
- Offset Adressierung bezüglich Start der Routine

  pid54321:my-object:my-function:8

  pid54321:libc.so.1:strcpy:entry
  pid54321:libc.so:strcpy:entry
  pid54321:libc:strcpy:entry



- die Probes werden bei Bedarf generiert und sind daher nicht Bestandteil der Ausgabe von „*dtrace -l*“
- Tipp: Kommandozeilen Optionen „-p“ und „-c“

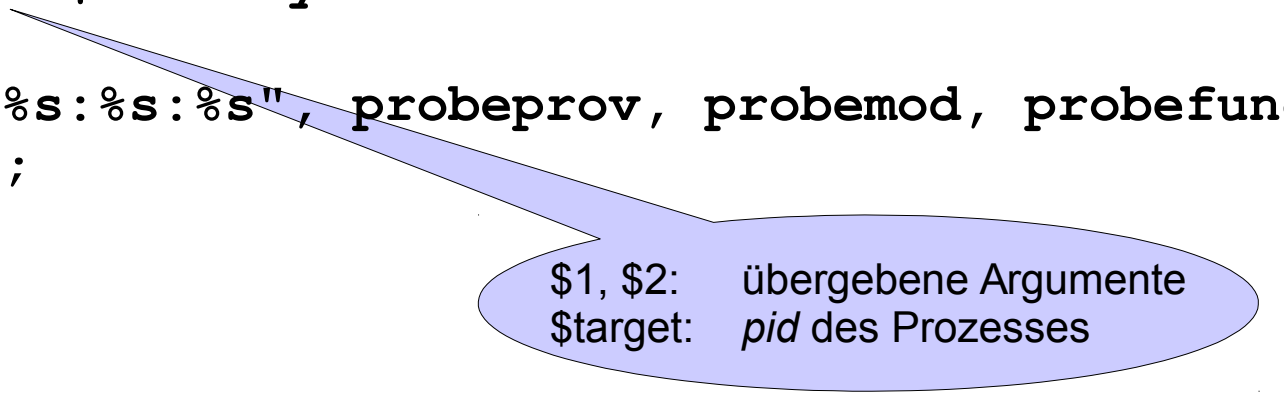
Beispiel: Aufruf-Stacks (lib_call_stack.d)

```
#!/usr/sbin/dtrace -s

/* uses the 'pid' Provider to print call stacks */

#pragma D option quiet

pid$target:$1:$2:entry
{
    printf("%s:%s:%s", probeprov, probemod, probefunc);
    ustack();
}
```



Beispiel: Aufruf-Stacks

```
obi-wan# ./lib_call_stack.d -c ls 'libc' 'str*'
```

```
lib_call_stack.d
```

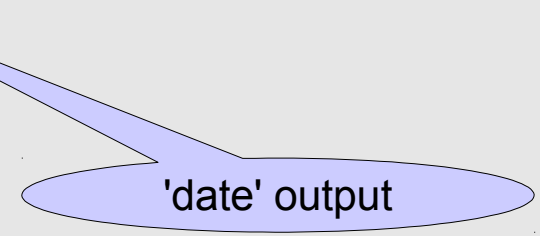
'ls' output

```
pid1002:libc.so.1:strcmp
              libc.so.1`strcmp
              libc.so.1`setlocale+0x1378
              ls`main+0x22
              ls`_start+0x7a
pid1002:libc.so.1:strlen
              libc.so.1`strlen
              ls`xstrdup+0x12
              ls`main+0x486
              ls`_start+0x7a
pid1002:libc.so.1:strlen
              libc.so.1`strlen
              ls`xstrdup+0x12
```

Weiteres Beispiel: libc Trace

```
obi-wan# ./libc_trace.d -F -c date

dtrace: script './libc_trace.d' matched 5789 Probes
Sunday, November 11, 2012 11:50:04 AM CET
dtrace: pid 16648 has exited
CPU FUNCTION
  5  -> __tls_static_mods
  5    -> lmalloc
  5      -> getbucketnum
  5      <- getbucketnum
  5      -> initial_allocation
  5        -> __systemcall
  5        <- __systemcall
  5      <- initial_allocation
  5    <- lmalloc
...
```



Showcase
„Bacula Enterprise Backup“

Showcase „Bacula Enterprise Backup“

- Infrastruktur
  - 2x Oracle SPARC-T4 Server mit 128GB Speicher
    - 3 dual-port SAS-2 HBA (LSI) pro Server
    - 1 dual-port 16Mb Fiber-Channel HBA pro Server
  - 2x Quanta M4600H JBODs mit je 60 HGST 4TB SAS-2 Enterprise Platten
    - 9x 6+2 RAIDZ“ Konfiguration mit 4 hot-spare pro Chassis
    - multi-stream IO-Messungen mit *filebench* liefern je nach Konfiguration 3-5 GB/s,
  - 8x IBM 3592-E07 Fiber-Channel Bandlaufwerke (dual-ported)
    - 250 MB/s IO bei nicht komprimierbaren Daten (gemessen) sonst bis über 500 MB/s
  - PostgreSQL 9.2 auf 6x 800GB SAS-2 Enterprise SSDs

Showcase „Bacula Enterprise Backup“

- Strategie
  - Daten aller Clients werden zuerst auf Disk gesichert
  - Incrementelle-Backups und derzeit auch Differential-Backups verbleiben für ihre Lebensdauer auf den Plattensystemen
  - Full-Backups werden abhängig von Füllgrad des Pools und Alter auf Band migriert
- Pre-Production Tests lieferten für Daten > ~2 TB bei disk-to-tape Migration schlechtere Durchsatzwerte als erwartet
  - Migration von 6TB in 4,5 Millionen Dateien benötigt 30 Stunden; Datenrate nur ca. 61 MB/s
  - enttäuschend im Vergleich zum initialen Backup vom Client (36h)

Showcase „Bacula Enterprise Backup“

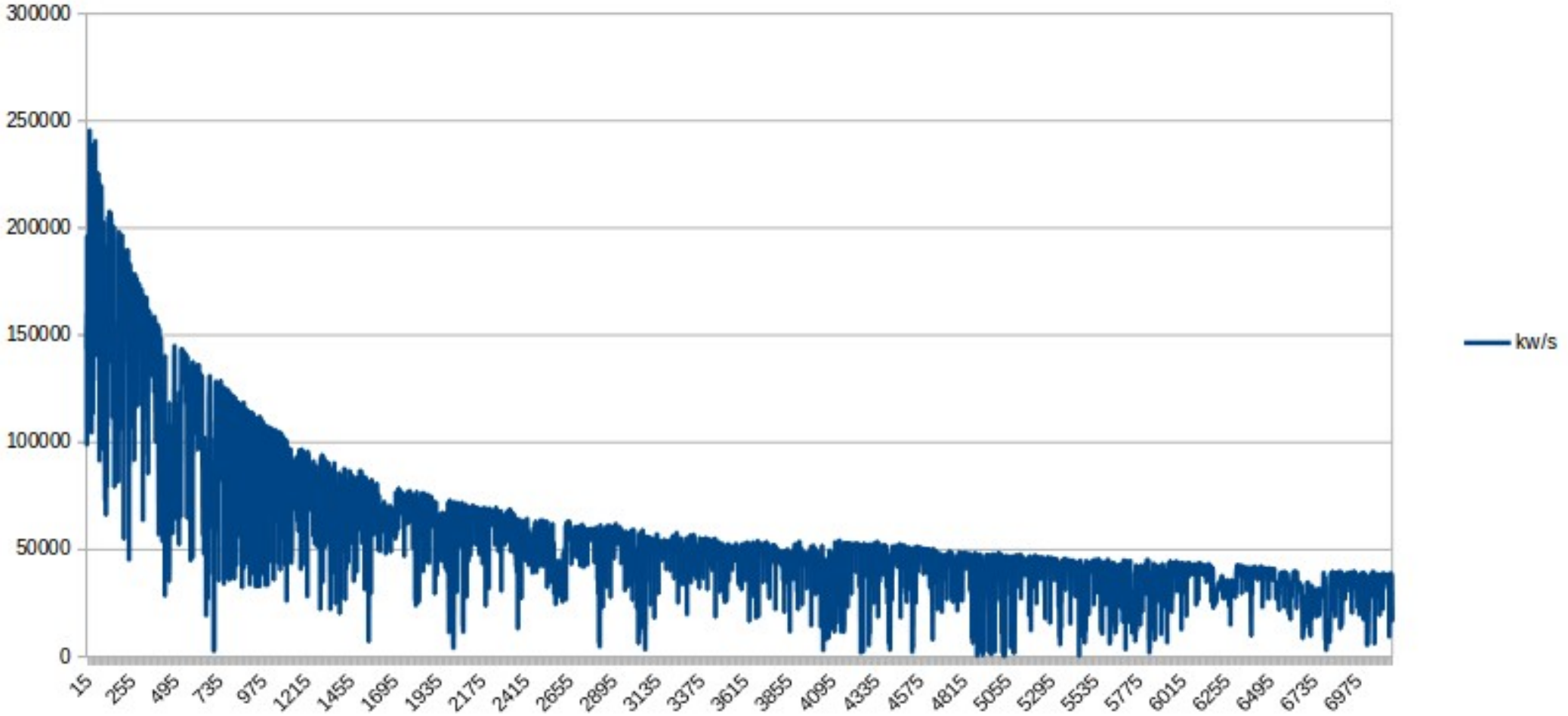
- DTrace sampling des Programmzählers mit einer Frequenz von 1001Hz zeigt keine CPU Engpässe

```
Routine                                  calls   prct.
SPARC-T4`copyin                           8630    0.2%
zfs`fletcher_4_native                    10902    0.3%
zfs`vdev_raidz_generate_parity_pq        19815    0.5%
zfs`lzjb_compress                       234676    5.9%
unix`cpu_halt                          3670636   91.8%
```

- *IO-provide*r bestätigt gleichbleibend große IO chunks (256kB)

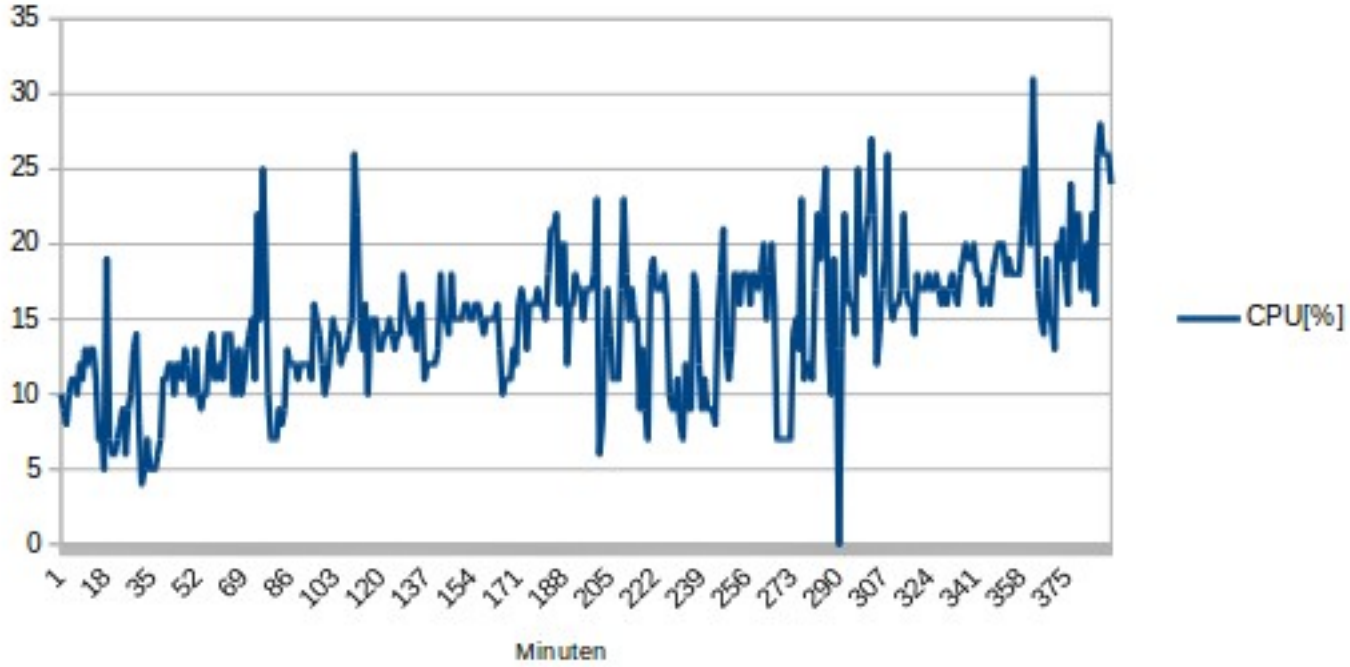
Showcase „Bacula Enterprise Backup“

- *IO-provider* liefert aber auch erste Hinweise auf „schleichendes Problem“



Showcase „Bacula Enterprise Backup“

- weiteres Vorgehen mit DTrace
  - hotspot sampling der Anwendung bzgl. Zahl der Aufrufe, CPU Zeit und zeitlichem Verlauf



  - eine Funktion benötigt nach 6 Stunden Laufzeit bereits ca. die doppelte CPU Zeit wie zu Beginn

Showcase „Bacula Enterprise Backup“

- call-stack Analyse um Code Pfad zu finden und Übergabe an die Entwickler
- Vermutung eines schlecht skalierenden Algorithmus ließ sich mit einfachem Workaround verifizieren
- Patch liefert innerhalb weniger Tage knapp 4x an Verbesserung hinsichtlich der Laufzeit

Literatur

- hervorragende Informationsquelle zumal auch alle neuen und nur in Solaris 11 verfügbaren Provider dokumentiert sind

  http://wikis.oracle.com/display/DTrace

- das DTrace Manual (PDF)

  http://docs.oracle.com/cd/E26502_01/pdf/E28556.pdf

- Beispiele finden sich unter

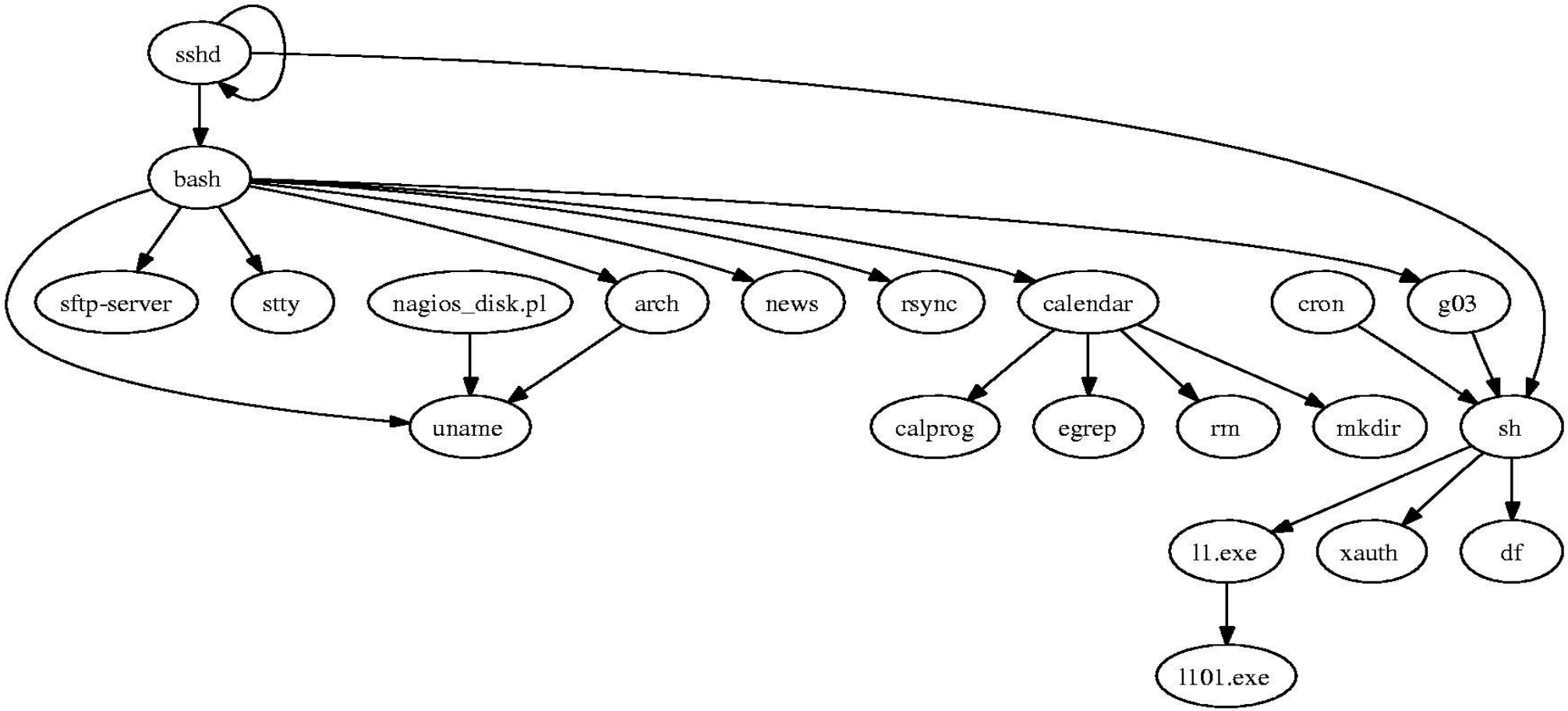
  https://wikis.oracle.com/display/DTrace/DTrace

  http://dtracebook.com

  http://www.brendangregg.com/dtrace.html

  /usr/demo/dtrace

Zu guter letzt: „Pimp my Dtrace“



Danke für's Zuhören!