Teamwork – DTrace und Analytics

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

Übersicht

- mein Hintergrund
- DTrace know-how Auffrischung
- DTrace, was ist neu in Solaris 11.4?
- DTrace Beispiele
- StatsStore und Analytics
- wrap-up
- Q & A

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 (sehr) langer Zeit
- Schwerpunkte UNIX und Storage Lösungen

One team to serve them all

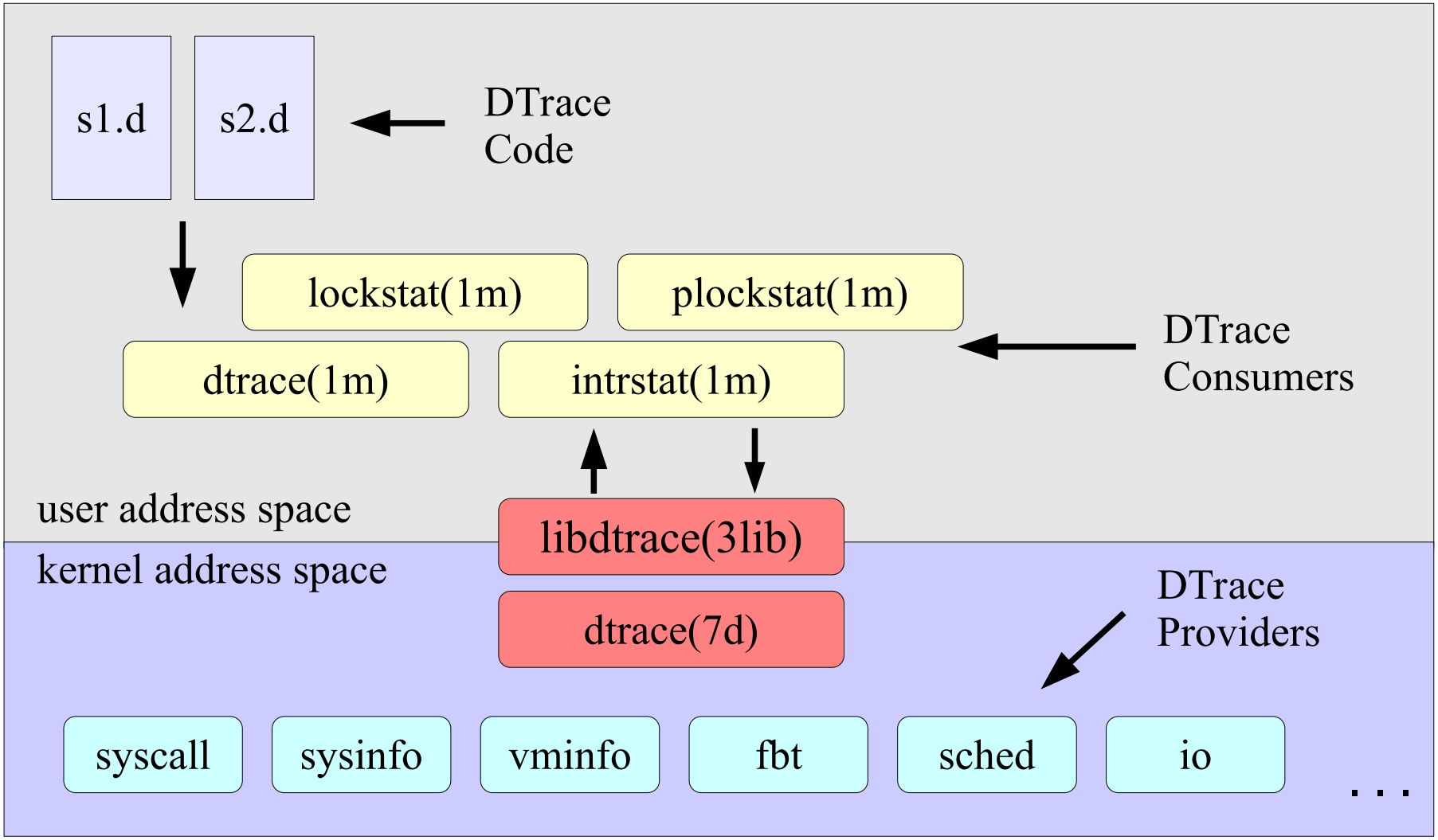
- die „Abteilung Infrastruktur“ des kiz erbringt Dienstleistungen für ~18.000 Personen
  - Netzwerke (LAN, MAN und WLAN) und Telefonie
    - Anbindung an das Landeshochschulnetz BelWue mit 100 Gbit
  - Betrieb der gesamten zentrale Server-IT inklusive ca. 700 Desktop PCs sowie Laptops
  - Dienstleistungen im Rahmen von Landeskooperationen
    - Backup Service für mehrere Universitäten in Baden-Württemberg
    - HPC Landes-Cluster „Computergestützte Chemie und Quantenwissenschaften“
    - bwCloud: „IaaS für Forschung und Lehre in Baden-Württemberg“
  - bedient Cloud-Hype, Next-Hype, Whatever-else Hype

Die Vorteile einer Abteilung

- ein Team, ein Ziel
  - Server- und Desktop-Betrieb, Netz- und TK-Gruppen sind Bestandteil des Teams
    - kein Elfenbeinturm sondern **„reality exposure“** und **„eat your own dog food“** Philosophie
    - Entscheidungen sind einfacher zu treffen und gemeinsam zu tragen
    - hilft realistische und umsetzbare Ziele zu definieren
- spannendes Arbeitsumfeld mit vielen Freiheiten für Leute die bereit sind Verantwortung zu übernehmen
  - „erlaubt ist was funktioniert und wartbar ist“
    bleeding edge ist kein Tabu

DTrace recap

DTrace Block Schaubild



Provider stellen Probes bereit

- Probes „Triggerpunkte“ die viele Aktionen auslösen können
  - Aufzeichnung von Kernel- oder User-Stacks sowie von Datenstrukturen oder Speicherinhalten
- Provider sammeln Daten spezifischer Bereiche, bereiten sie auf und stellen sie asynchron für Weiterverarbeitung bereit
  - Bsp.: *io-provider* für File/Block-IO bezogene Probes
- DTrace / D stellt zusätzliche nützliche Routinen und Variable bereit
  - cleanpath(), inet_ntoa(), progenyof(), …
  - strlen(), strjoin()
  - pid, ppid, execname, fds[], ...

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;
```

„D“ Beispiel-Skript

```
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
```

DTrace's Warp Antrieb: Aggregations

- hilfreiche Funktionen

Funktion	Aufgabe
trunc(@aggr [,n])	löscht eine Aggregation ganz oder teilweise n > 0 : die n obersten Einträge bleiben erhalten n < 0 : die n untersten Einträge bleiben erhalten
clear(@aggr)	setzt die Werte eine Aggregation auf 0
normalize(@aggr, val)	dividiert alle Werte einer Aggregation durch „val“ jedoch ohne die Originaldaten zu verändern
denormalize(@aggr)	macht eine Normalisierung rückgängig

DTrace's Warp Antrieb: 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)

Beispiel: IO Latenz

```
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
```

Solaris 11.4
DTrace update

Solaris 11.4 DTrace update

- Solaris 11.4 führt neue *provider* und *actions* ein

scsi	*probes* für das tracing von SCSI Kommandos
icmp	send/receive probes für das Internet Control Message Protocol (ICMP)
igmp	send/receive probes für das Internet Group Management Protocol (IGMP)
sctp	*probes* für das tracing des Stream Control Transmission Protocols (SCTP)
fileops	stellt *probes* für Fileoperationen (open, close, …) zur Verfügung die unabhängig von Filesystem oder storage devices bedient werden
pcap	mit *trace* vergleichbare *action* die bei der Ausgabe von Netzwerkdaten in eine Datei das libpcap Format verwendet (lesbar mit *wireshark(1)*)

Solaris 11.4 DTrace update: fileops

- schließt die Lücke zwischen dem *syscall* und dem *io-provider*
- dient dem Monitoring von Dateioperationen unabhängig vom Filesystem und ob physikalischer IO stattfindet
  - *probes* umfassen sowohl 32-bit als auch 64-bit Operationen
  - die Daten werden dem *consumer* nur bei erfolgreicher Operation bereitgestellt

Solaris 11.4 DTrace update: fileops

- Daten werden als *fileinfo_t* Struktur übergeben, die Latenz der Operation als *hrtime_t*
  - *probe* spezifisch werden bis zu drei weitere Zeiger übergeben

```
typedef struct fileinfo {
    string fi_name;       /* name(basename of
                             fi_pathname) */
    string fi_dirname;    /* directory (dirname of
                             fi_pathname) */
    string fi_pathname;   /* full pathname */
    offset_t fi_offset;   /* offset within file */
    string fi_fs;         /* filesystem */
    string fi_mount;      /* mount point of file system */
} fileinfo_t;
```

fileops: Leselatenz pro Filesystem in [ns]

```
# dtrace -n 'fileops:::read {
        @[args[0]->fi_mount] = quantize(args[1]);
  }'
dtrace: description 'fileops:::read ' matched 1 probe
^C
…
/zones/cifs/root/smb/data/department2
    value  ------------- Distribution ------------- count
     4096 |                                         0
     8192 |@                                        2
    16384 |                                         1
    32768 |                                         1
    65536 |                                         0
   131072 |@@@                                      11
   262144 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@  120
   524288 |                                         0
…
```

fileops: Schreiblatenz pro Filesystem in [us]

```
# dtrace -n 'fileops:::write /args[0]->fi_fs == "zfs"/ {
        @[args[0]->fi_mount] = quantize(args[1] / 1000);
  }'
dtrace: description 'fileops:::write ' matched 1 probe
^C

/pool1/home/kiz
    value  ------------- Distribution ------------- count
     1024 |                                         0
     2048 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ 12
     4096 |@@@@@@@@@@@@@@@@                         7
     8192 |                                         0

/pool1/home/student1
    value  ------------- Distribution ------------- count
     1024 |                                         0
     2048 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ 34
     4096 |@@@@@@@@                                 8
     8192 |
```

Solaris 11.4: Latenzen messen mit *fsstat(8)*

- Solaris 11.4 bietet noch weitere Möglichkeiten um Latenzen im Filesystem bzw. Disk-IO zu messen

```
# fsstat -l zfs 10

 read read   read write write write rddir rddir rddir
  ops bytes  time   ops bytes  time   ops bytes  time
71.4M 1.50T    8n 24.5M  746G  585n  286M 62.4G 1.54u zfs
   98 8.92M    0n   490 6.18M  423n   784 54.9K 4.32u zfs
1.81K 11.8M    4n 1.99K 18.8M  335n   114 46.8K 40.0n zfs
```

Solaris 11.4: Latenzen messen mit *iostat(8)*

```
# iostat -Lxn c0t5000C5003C8C82BFd0 10
latency           range          count      density distribution
                   <16us              0        0.00%         0.00%
                 16-32us          74388        0.66%         0.66%
                 32-64us         145306        1.29%         1.95%
                64-128us        1833151       16.23%        18.18%
               128-256us        2235358       19.79%        37.97%
               256-512us        2232688       19.77%        57.74%
              512-1024us        2353436       20.84%        78.58%
                   1-2ms         635769        5.63%        84.21%
                   2-4ms         159311        1.41%        85.62%
                   4-8ms         538704        4.77%        90.39%
                  8-16ms         733114        6.49%        96.88%
                 16-32ms         269782        2.39%        99.27%
                 32-64ms          68960        0.61%        99.88%
                64-128ms          12311        0.11%        99.99%
               128-256ms            729        0.01%       100.00%
               256-512ms            106        0.00%       100.00%
              512-1024ms              4        0.00%       100.00%
                >1024ms              0        0.00%       100.00%
```

Solaris 11.4 DTrace update: *pcap action*

- Problem
  - es ist mühsam die etwa mit wireshark(8) gesammelten Pakete einem Prozess zuzuordnen
- *pcap(mblk, protocol)* to the rescue
  - kopiert Daten, vergleichbar zur *trace action*, in Puffer
  - verwendet *libpcap*-Format sofern Ausgabe mittels *freopen()* in Datei umgeletiet wurde
    - ACHTUNG: *freopen()* leitet alle Ausgaben um
  - vordefinierte Typen von Paketen
    - PCAP_ETHER, PCAP_WIFI, PCAP_PPP, PCAP_IP, PCAP_IPNET, PCAP_IPOIB
  - Paketdaten lassen sich mit *wireshark(8)* weiter analysieren

Zuordnungsproblem bei IP Paketen

```
# dtrace -q -n 'ip:::send, ip:::receive {
      @[probename, execname, pid] = count();
    }
    tick-10s {
      printa("%-10s %-12s %5d %@6d\n", @);
      exit(0);
    }'
...
send       nfsd          1130      1
send       rpcbind        970      1
send       smbd         29769      1
send       sshd         26943      1
send       smbd         29861     26
send       smbd          5848     30
send       sshd          7503    162
send       nfsd_kproc    1132   4231
send       sched            0   5995
receive    sched            0  27625
```

empfangene **IP** Pakete können noch keinem Prozess zugeordnet werden, gesendete schon

Zuordnungsproblem bei TCP Paketen

```
# dtrace -q -n '
    tcp:::send, tcp:::receive {
      @[probename, execname, pid] = count();
    }
    tick-10s {
      printa("%-10s %-12s %5d %@6d\n", @);
      exit(0);
    }'
```

```
…
send       smbd          29861     26
receive    sshd           7503     35
send       smbd           5848     37
send       smbd            144    157
send       sshd           7503    171
receive    nfsd_kproc     1132    727
send       nfsd_kproc     1132   1286
```

Process-ID (pid) ist bei TCP Verbindungen verlässlich da sie dem Verbindungskontext args[1]→cs_pid entnommen werden kann

Demo

Beispiel: *pcap* in der Praxis

```
trinity# dtrace -q -w -n '
    BEGIN { freopen("/tmp/nau.pcap"); }

    tcp:::send, tcp:::receive
    / args[1]->cs_pid == $target / {
      pcap(args[0]->pkt_addr, PCAP_IP);
    }

    END { freopen(""); }' -c "nc -k -l 1234"
```

```
neo# echo "What is the Matrix?" | nc trinity 1234
```

```
trinity# tshark -r /tmp/nau.pcap \
            -o data.show_as_text:TRUE -T fields \
            -e frame.number -e ip.src -e ip.dst -e data.text
1       134.60.1.94          134.60.1.113
2       134.60.1.94          134.60.1.113
3       134.60.1.94          134.60.1.113  What is the Matrix?
4       134.60.1.113         134.60.1.94
```

Weitere Verbesserung: *print*

- ergänzt die *actions printf(), printa()* und *trace()*
- ermöglicht die Ausgabe von Variablen in weiten Bereichen ohne das ihr Typ bekannt sein muss
- Beispiel Namensauflösung in *vnode* Struktur mit Hilfe von VOB_LOOKUP

```
extern int fop_lookup(vnode_t *, char *, vnode_t **,
    struct pathname *, int, vnode_t *, cred_t *,
    caller_context_t *, int *, struct pathname *);
```

Weitere Verbesserung: *print*

```
# dtrace -q -n '
    fop_lookup:entry {
      self->vnode = args[0];
    }
    fop_lookup:return
    / self->vnode / {
      print(*self->vnode);
      exit(0);
    }'

vnode_t {
  v_lock = {
    _opaque = [ NULL ]
  }
  v_flag = 0x0
  v_count = 0x2
  v_data = 0xffff8106e5e6c498
…
  v_path = "/nau/Documents/DOAG_2018/DTrace_is_cool.odp"
```

Weitere Verbesserung: *print*

```
# dtrace -q -n '
    fop_lookup:entry {
      self->vnode = args[0];
    }
    fop_lookup:return
    / self->vnode / {
      print(self->vnode->v_path);
      self->vnode = 0;
    }'

char * "/zones/cifs/root"
char * "/zones/cifs/root/smb"
char * "/zones/cifs/root/smb/sw"
char * "/zones/cifs/root/smb/sw/var"
char * "/zones/cifs/root/smb/sw/var/log"
char * "/zones/cifs/root"
^C
```

Solaris 11.4
StatsStore

StatsStore

- vereinheitlicht Sicht auf Performance-Daten und Ereignisse
  - kstat(), audit-event, fault-events (FMA), …
  - Hardware events
  - Informationen über Prozesse, ...
- einheitlicher Namensraum
  - Klassen, Ressourcen, Statistiken und Partitionen
- automatisierte Erfassung und Konsolidierung durch SMF-service

  svc:/system/sstore
- Ausgabe auch im CSV oder JSON Format
  - ssid(7), sstore.csv(5), sstore.json(5)

StatsStore

```
# sstore list //:class.disk//:res.name/sd0//:stat.*
IDENTIFIER
//:class.disk//:res.name/sd0//:stat.read-bytes
//:class.disk//:res.name/sd0//:stat.read-ops
//:class.disk//:res.name/sd0//:stat.write-bytes
//:class.disk//:res.name/sd0//:stat.write-ops


# sstore export -t 2018-09-27T15:07:10 -p 1 \
    //:class.cpu//:res.id/0//:stat.usage//:part.mode

TIME                 VALUE IDENTIFIER
2018-09-27T15:07:10
//:class.cpu//:res.id/0//:stat.usage//:part.mode
                     idle: 1199651699.766952
                     intr: 7320396.622533
                     kernel: 115791538.181657
                     user: 13138812.588451
```

StatsStore capture

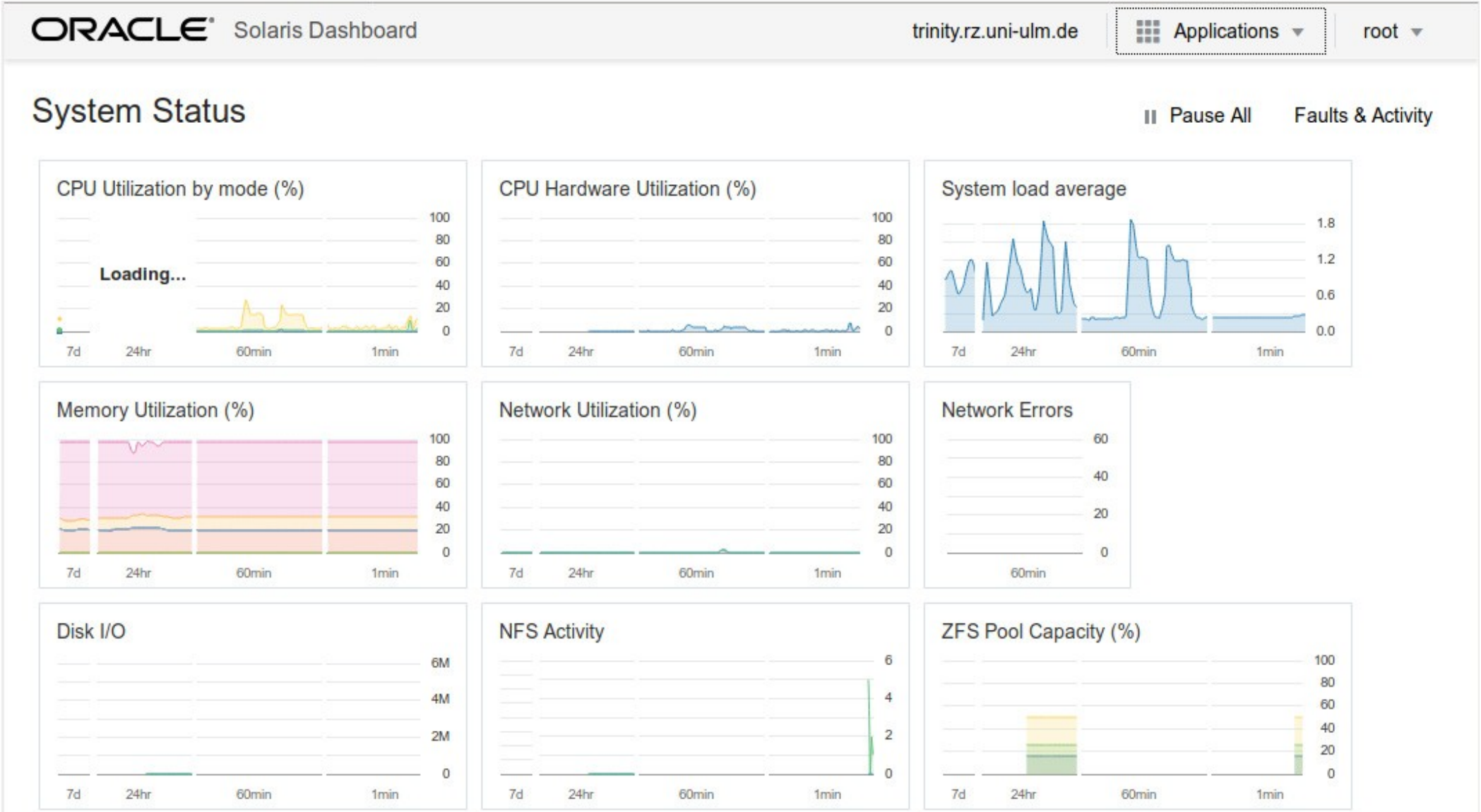
- Statistiken lassen sich individuell oder als vordefinierte collections aktivieren
  - die folgenden Ausgaben wurden angepasst um die Lesbarkeit in dieser Präsentation zu verbessern

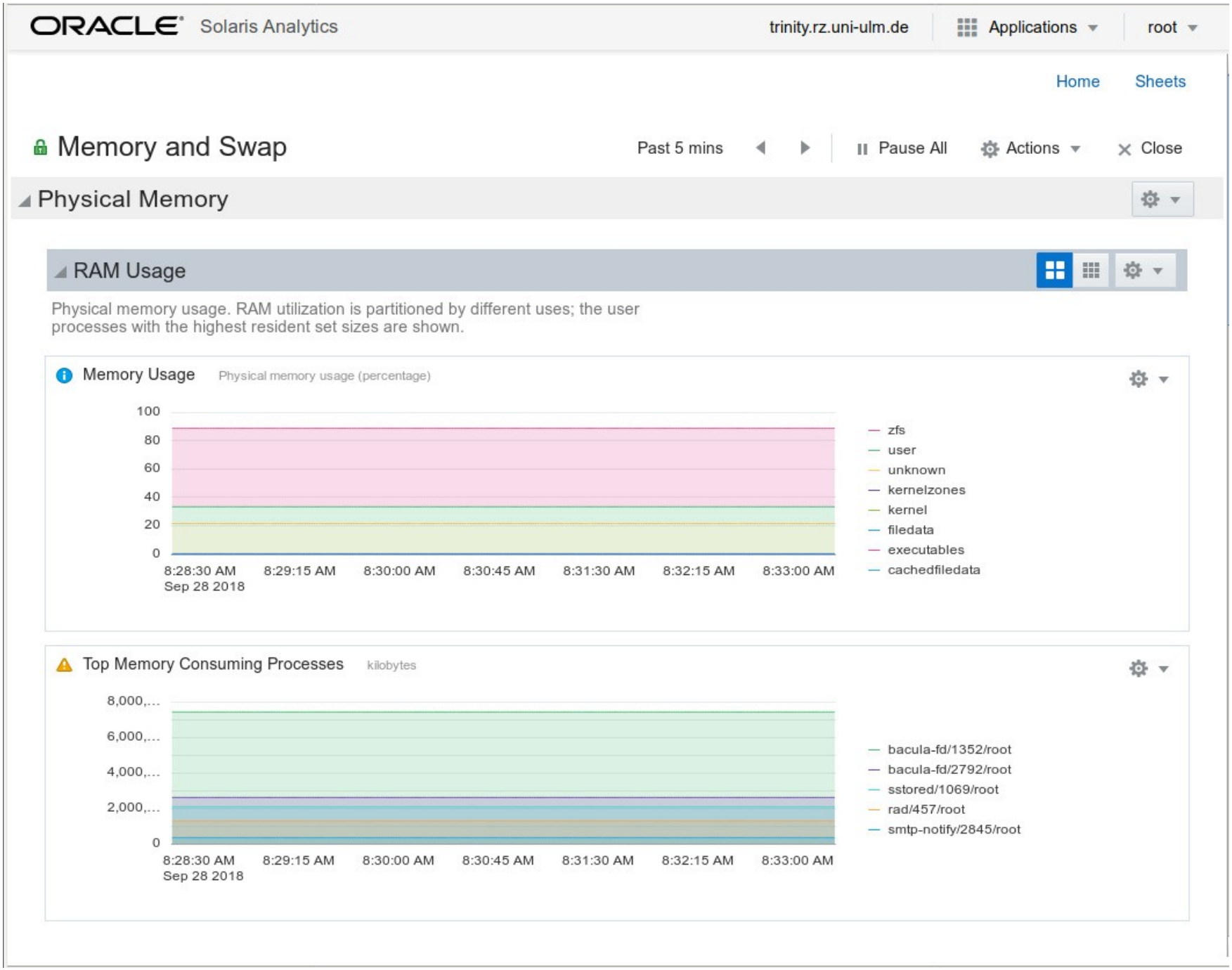
```
# sstore capture \
//:class.disk//:res.name/sd0//:stat.{read-bytes,write-bytes} 5

TIME     VALUE  IDENTIFIER
07:42:48 0      //:class.disk//:res.name/sd0//:stat.write-bytes
07:42:48 131480 //:class.disk//:res.name/sd0//:stat.read-bytes
07:42:53 0      //:class.disk//:res.name/sd0//:stat.write-bytes
07:42:53 131480 //:class.disk//:res.name/sd0//:stat.read-bytes
…
```

StatsStore collections

```
# sstore info //:class.collection//:collection.*
Identifier: //:class.collection//:collection.name/root/cpu-
stats
  ssid: //:class.cpu//:stat.context-switches
  ssid: //:class.cpu//:stat.integer-pipe-capacity
  ssid: //:class.cpu//:stat.integer-pipe-usage
  ssid: //:class.cpu//:stat.interrupt-count
  ssid: //:class.cpu//:stat.interrupt-time
  ssid: //:class.cpu//:stat.involuntary-context-switches
  ssid: //:class.cpu//:stat.system-calls
  ssid: //:class.cpu//:stat.thread-migrations-core
  ssid: //:class.cpu//:stat.thread-migrations-socket
  ssid: //:class.cpu//:stat.traps
  ssid: //:class.cpu//:stat.usage
  ssid: //:class.cpu//:stat.xcalls
 state: disabled
  uuid: c42ae7f3-3721-4d81-be79-ca7f5f9d0b1f
 owner: root
 cname: cpu-stats
crtime: 153672525977637l
```

StatsStore Zugriff via Dashboard





Q & A