dtrace

Frank Kargl | CCC Ulm | 5.5.08

dtrace?

- Was ist dtrace?
- Wie (und wofür) nutze ich dtrace?
  - D scripts
  - Instruments
- Wie funktioniert dtrace?

Wozu dtrace?

- „Mein Programm läuft zu langsam“
- „Mein Kernel verbraucht viel zu viel Speicher“
- „Warum greift meine Java Anwendung ständig auf die Festplatte zu?“
- „Wie lange dauern bestimmte Krypto-Operationen in einer C/Java Krypto Lib?“

Profiling

- Trace:
  Aufzeichnung von Ereignissen während eines Programmablaufs
- Profile:
  Statistische Auswertung eines Programmablaufs
- Aufzeichnungsmethoden
  - Event-based Profilers (.NET, JVM-Tools Interface, Python, Ruby, ...)
  - Statistical Profilers (gprof)
  - Instrumentation

Profiling von (Binär-)Anwendungen

- Erneutes Übersetzen der Anwendung?
- Erzeugt der Profiling Code Overhead?
  - Auch wenn Profiling gar nicht läuft?
- Verändert Profiling das Verhalten der Anwendung?
  - Heisenbugs
- Funktioniert das auch im Kernel? In einer Java VM? In einer Embedded Anwendung?

dtrace in a Nutshell (1)

- dynamic tracing facility
- Januar 2005: Open Solaris / CDDL
- Bryan Cantrill, Mike Shapiro, and Adam Leventhal
- Verfügbar für OpenSolaris, Java, FreeBSD, MacOS, QNX, Oracle, Linux(?) uvm.

dtrace in a Nutshell (2)

- Dynamic Runtime Instrumentation
- System instrumentation (Kernel/userspace)
- Safe, zero probe effect wenn nicht aktiviert
- Unterstützt >> 10.000 Instrumentation Points
- Umfangreiche Skriptsprache für Instrumentierung und Profiling
  - Unterstützt Aggregation und speculative tracing

dtrace in a Nutshell (3)

- Probes
  - Stellen, an denen etwas gemessen werden kann
    - 30.000+ in Open Solaris
- D Skripte
  - Steuern das Profiling
  - Ähnlich awk Skripte
  - Werden kompiliert und im Kernel ausgeführt

Probes

- Format:
  `provider:module:function:name`
- dtrace -l: liste alle Probes im System
  `dtrace -l -n syscall:::`
- dtrace -n: führe Probe aus
  `dtrace -n syscall::seteuid:entry`
  `dtrace -n ´syscall:::entry {trace(10);}´`
  `dtrace -n ´syscall:::entry {trace(pid);}´`

D Skripte

- Variablen: execname, timestamp (ns), walltimestamp (s), pid, uid, gid, probeprov, probemod, probefunc, probename, arg0, arg1, ...
- Predicates:

```
syscall::::entry
/execname==´Keynote´/
{printf(´%s - %x´,arg0, arg1);}
```

Aggregates

- Wie oft wird ein Systemcall aufgerufen?
- Datei nrcalls.d:

```
syscall:::entry
{
  @entries[probefunc] = count();
}
```

- dtrace -s nrcalls.d

quantize()

- Histogram erstellen
- Datei histo.d

```
syscall::write:entry
{
  @hist = quantize(arg2);
  @biggest = max(arg2);
  @average = avg(arg2);
  @smallest = min(arg2);
}
```

- dtrace -s histo.d

Gültigkeit von Var.

- global, thread-lokal (self->), probe-lokal
- Datei trace-ioctl.d

```
#pragma D option flowindent
syscall::ioctl:entry
{ self->follow = 1; }
fbt:::
/self->follow/
{}
syscall::ioctl:return
/self->follow/
{ self->follow = 0; exit(0); }
```

Timing von Funktionen

- Datei method-time.d

```
#!/usr/sbin/dtrace -s
syscall::write:entry
{ self->start[probefunc] = timestamp; }

syscall::write:return
/ self->start[probefunc] /
{
printf("%s - %d\n", probefunc,
     timestamp - self->start[probefunc]);
@hist = quantize(timestamp -
     self->start[probefunc]);
self->start[probefunc] = 0;
}
```

User Level Tracing

- Der pid Provider kann jede Instruktion in jedem Prozess tracen!
- Probeformat:
  `pid<`*`pid`*`>:`*`object`*`:`*`func`*`:`*`name`*
- Beispiel: Dauer von MD5 Aufrufen in openssl

test-md5.c

```
#include <openssl/md5.h>

void gen_rand_string(char* buf) {... }

int main (int argc, char * const argv[]) {
    ...
    MD5_CTX* md5ctx;
    MD5_Init(&md5ctx);
    unsigned char mdbuf[MD5_DIGEST_LENGTH];
    char c_string[buf_len];

    int runs=nr_runs;
    while (runs-- > 0) {
        gen_rand_string(c_string);
        MD5_Update(&md5ctx, c_string, buf_len);
    }
    MD5_Final(mdbuf, &md5ctx);
    return 0;
}
```

method-time-pid.d

```
pid$target::$1:entry
{
  self->start[probefunc] = timestamp;
}

pid$target::$1:return
/ self->start[probefunc] /
{
  printf("%s - %d\n", probefunc,
      timestamp - self->start[probefunc]);
   @hist = quantize(timestamp -
      self->start[probefunc]);
   self->start[probefunc] = 0;
}

$ dtrace -s method-time-pid.d -c "./test-md5 100 1000"
MD5_Update
```

Was noch? (1)

- stack() und ustack() für Stack-Traces
- Zugriff auf Daten im Userspace mit
  - copyin(addr, size)
  - copyinstr(addr)
- Probe-lokale Variablen: this->
- Destructive actions: mit -w aktivieren
  - copyout, copyoutstr
  - z.B. Rückgabewerte von syscalls verändern
  - stop(), raise(signal), system()

Was noch? (2)

- Profile Provider: regelmäßige Probes
- Aggregation
  - clear(): lösche Werte
  - trunc(): lösche Werte und Schlüssel
  - printa(): Formatiere Aggregates
- Speculative Tracing

Trace Files

- Datei trace-files.d:

```
io:::start
{
  @[args[2]->fi_pathname]=count();
}
tick-10s
{
  trunc(@, 10);
  printa(@);
  trunc(@, 0);
}
```

dtrace und MacOS (1)

- Einige Beispiele:
  - syscalls_per_process.d
  - objc_msg.d
  - open_calls.d
- Interessante Skripte (Brendan Gregg):
  - man -k dtrace | grep "\.d"
  - /usr/share/examples/DTTk
  - http://www.brendangregg.com/dtrace.html

dtrace und MacOS (2)

- Apple und DRM:

```
#if defined(__APPLE__)
 /* If the thread on which this
  * probe has fired belongs to a
  * process marked P_LNOATTACH
  * then this enabling is not
  * permitted to observe it.
  * Move along, nothing to see
  * here. */
if (ISSET(current_proc()-
>p_lflag, P_LNOATTACH)) {
continue; }
#endif /* __APPLE__ */
```

dtrace und MacOS (3)

- Verfügbare Provider (Auszug)
  - FinderSpotlight
  - dtrace
  - fbt
  - io
  - lockstat
  - mach_trap
  - objc
  - pid
  - proc
  - profile
  - sdt
  - syscall
  - vminfo

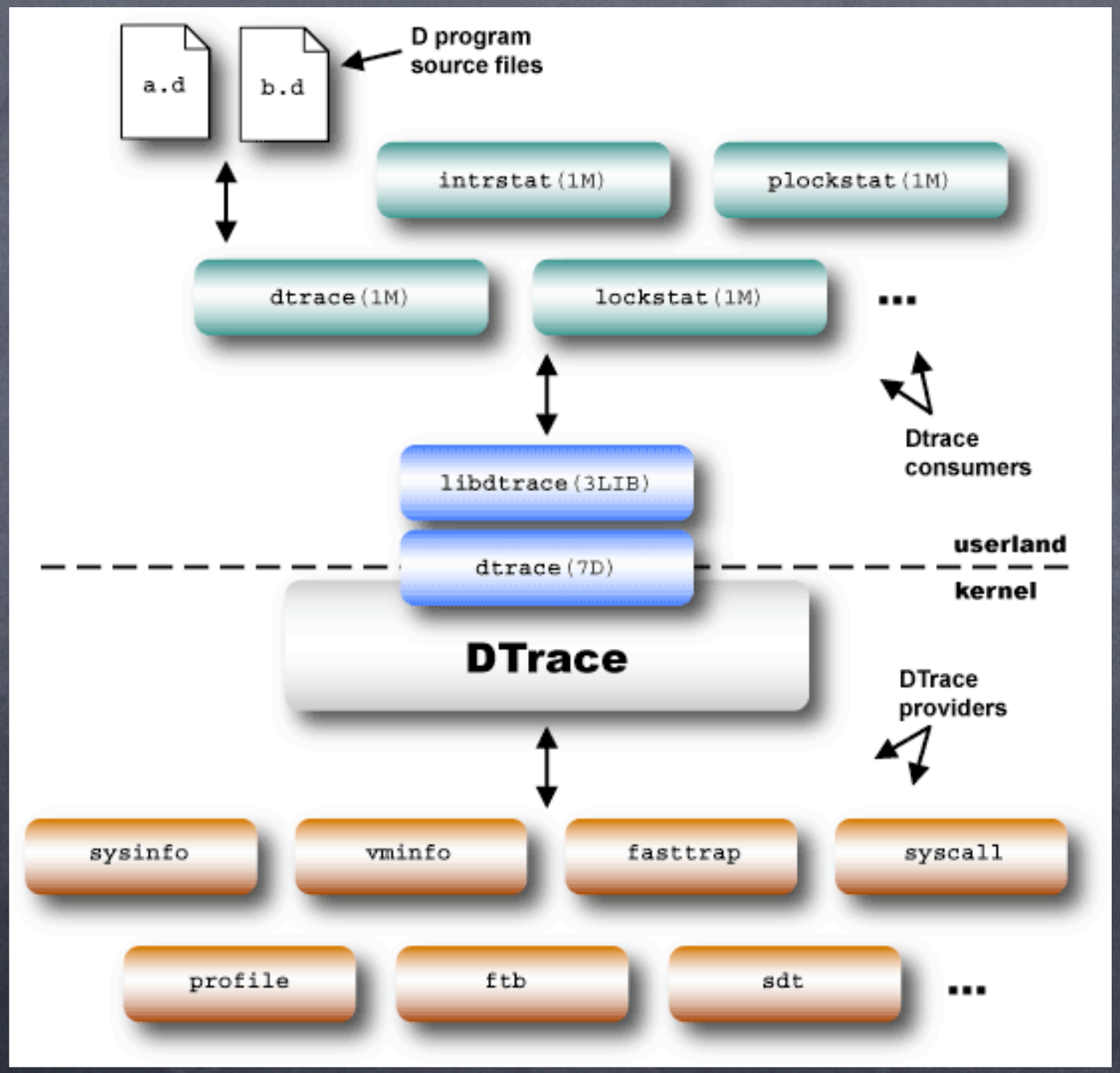
dtrace und Java

- Es ist nach 1:30 und ich hab keine Lust mehr ;-)

- hotspot$VMPID hilft weiter

- http://www.devx.com/Java/Article/33943
- http://java.sun.com/javase/6/docs/technotes/guides/vm/dtrace.html

Wie funktioniert dtrace?



Wie funktioniert dtrace?

- Skripte werden im Userland kompiliert, zum Kernel Modul geschickt und dort interpretiert.
- Dort instrumentieren Provider geeignete Teile des Systems
  - „Umbiegen“ von Syscall Tables, Austausch von Assembler-Anweisungen beim Anlegen von Stackframes, Verwendung von traps
- Snapshots werden in Puffern abgelegt und regelmäßig aus dem Userland geleert
  - Überläufe werden als Fehler gemeldet

Links (1)

- DTrace Tutorial: http://learningsolaris.com/docs/dtrace_course.2005.8.18.pdf
- Solaris Dynamic Tracing Guide: http://docs.sun.com/app/docs/doc/817-6223
- Dtrace by Example: http://developers.sun.com/solaris/articles/dtrace_example.pdf
- Exploring Dtrace with Leopard: http://www.mactech.com/articles/mactech/Vol.23/23.11/ExploringLeopardwithDTrace/index.html

Links (2)

- Java Wiki: http://www.solarisinternals.com/wiki/index.php/DTrace_Topics_Java
- Dtrace Tipps: http://kr.sun.com/developers/solaris/techdocs/dtrace_tips_public.pdf
- Cantrill e.a., „Dynamic Instrumentation of Production Systems“, USENIX 2004