Dynamische Instrumentierung von Produktionssystemen mit DTrace

Dipl.-Chem. Rainer Orth
Technische Fakultät
Universität Bielefeld
ro@TechFak.Uni-Bielefeld.DE

Übersicht

- Problemstellung
- Lösungsanforderungen
- DTrace
- DTrace-Toolkit

Problemstellung: Untersuchung komplexer Software-Systeme

- Performance-Probleme oder Fehler treten zunehmend erst in Produktionssystemen auf
- oft systemische Probleme als Wechselwirkung von Applikationen und Kernel oder sporadische Fehler

Probleme existierender Werkzeuge

- ausgerichtet auf Entwicklungssysteme (z.B. Debugversionen), erfordern Neustart, Codemodifikationen; Performanceeinbußen
- prozeßzentrisch mit enger Anwendbarkeit (`prstat`, `p`-Tools, `truss`, Debugger)
- liefern nur statistische Informationen ohne Bestimmung der Ursache (`iostat`, `vmstat`, ...)
- primär zur statischen Analyse von Crashdumps (`mdb`)

Lösungsanforderungen

- Einschränkungen zum Einsatz auf Produktionssystemen:
  - Instrumentierung darf keine Performance-Einbußen verursachen, wenn nicht aktiviert
  - Lösung muß absolut sicher sein: darf keine Fehler/Ausfälle verursachen
- für eine systemische Sicht:
  - gesamtes System muß instrumentierbar sein: Kernel und Anwendungen
  - Sammlung beliebiger Daten mit Möglichkeiten zur Zusammenfassung zum Erkennen von Trends

DTrace als Lösung

- dynamisches Tracing-Framework
- neu in Solaris 10, Ports auf Mac OS X 10.5 und FreeBSD in Arbeit
- Features:
  - dynamische Instrumentierung: kein Effekt, wenn deaktiviert
  - einheitliche Instrumentierung: sowohl Kernel als auch laufende Anwendungen mit Verfolgung des Kontrollflusses
  - Kernel-Instrumentierung in beliebigem Kontext: selbst Synchronisationsprimitive und Scheduling

Features

- Datenintegrität: kein unbemerkter Datenverlust
- beliebige benutzerdefinierte Aktionen an jedem Instrumentierungspunkt
- Prädikate: Aktionen werden nur unter benutzerdefinierten Bedingungen ausgeführt
- Highlevel-Sprache für Aktionen und Prädikate: `C`-ähnlich, Zugriff auf Kernel-Variablen und -Typen
- benutzerdefinierte Variable: global, thread-lokal, assoziative Arrays
- Datenaggregation auf beliebigen Tupeln
- spekulatives Tracen: Sammlung von Daten, die nur unter bestimmten Bedingungen erhalten bleiben

- Trennung von Instrumentierungsmethoden und Framework zur Datensammlung
- skalierbare Architektur erlaubt tausende Probes und effiziente Angabe von Teilmengen

Probes

- Probes sind Instrumentierungspunkte
- werden von Providern bereitgestellt
- benennen instrumentiertes Modul und Funktion
- jede Probe hat einen Namen
- 4-Tupel aus diesen Attributen identifiziert Probe

Provider

- Provider: Instrumentierungsmethode
- stellen dem DTrace-Framework Probes bereit
- DTrace läßt Provider die Probes aktivieren
- beim Feuern einer aktivierten Probe: Kontrolltransfer an Framework
- zahlreiche Provider, z.B.:
  - *function boundary tracing* (`fbt`): instrumentiert jede Kernel-Funktion
  - `syscall`: jeder System-Call
  - `pid`: jede Instruktion in jedem Prozeß

Aktionen und Prädikate

- Aktionen werden ausgeführt, wenn eine Probe feuert
- sammeln oft Daten
- Prädikate lassen Aktionen nur unter bestimmten Bedingungen ausführen

Die Sprache D

- Aktionen und Prädikate werden in D spezifiziert
- C-ähnliche Sprache:
  - Zugriff auf C-Typen des Kernels
  - Zugriff auf statische und globale Variable
  - ANSI-C-Operatoren
  - Strings als eigener Datentyp
  - thread-lokale Variable
  - assoziative Arrays

D-Programmstruktur

- eine oder mehrere Klauseln
- jede hat folgende Form:

```
probe-descriptions
/predicate/
{
  action-statements
}
```

- Probes werden spezifiert als *provider:module:function:name*
- leere Felder als Wildcards

Beispiel 1

- `dtrace -l`
- `dtrace -n 'syscall:::entry{printf("%s %s\n",execname,probefunc);}'`
- `dtrace -n 'syscall:::entry/execname=="acroread"/{trace(probefunc);}'`
- `dtrace -s ioctl.d -p 'pgrep acroread'`

Aggregations

- bei Performance-Problemen sucht man häufig nach Mustern
- nicht das einzelne Datum ist interessant, sondern man will aggregieren, um Trends zu erkennen
- bisher: Benutzung von `awk` oder `perl` zur Nachbearbeitung
- DTrace unterstützt Aggregation direkt
- dazu: *aggregation function* $f(x)$, x ist eine Menge von Daten:

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

- z.B. `count`, `sum`, `min`, `max`

Aggregations (ii)

- *aggregation*: Ergebnis einer *aggregating function*, indiziert durch beliebiges n-Tupel
- D-Syntax: `@identifier[keys] = aggfunc(args);`
- gültige `aggfunc`: `count`, `sum`, `min`, `max`, `avg`, `quantize`, `lquantize`
- normalerweise werden Aggregations beim Beenden von `dtrace` ausgegeben

Beispiel 2

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

Semantische Instrumentierung

- die verschiedenen Provider ermöglichen nahezu beliebige Instrumentierung ...
- ... aber erfordern Kenntnisse von Implementierungsdetails
- besser/zusätzlich: Instrumentierung mit bestimmter Semantik
- Provider können Interface-Stabilität ihrer Probes definieren
- mit statisch definierten Probes an semantisch sinnvollen Stellen können stabile Provider realsiert werden
- erfordern zusätzlich stabile Datentypen, daher ggfs. Umsetzung erforderlich

Stabile Provider: Kernel

- verschiedene stabile Kernel-Provider verfügbar:
  - `sched`: CPU-Scheduling
  - `proc`: Prozeßmanagement
  - `io`: I/O (Platten und NFS)
  - `sysinfo`: Systemstatistiken
  - `vminfo`: VM-Statistiken
  - weitere in Arbeit, z.B. `nfsv4`, `scsi`

Beispiel 3

- `dtrace -s /usr/demo/dtrace/sig.d`

Stabile Provider: User-Level

- auch für Applikationen wichtig, insbesondere bei interpretierten Sprachen: Instrumentierung mit `pid`-Provider nicht sinnvoll möglich
- implementierbar über *user-level statically defined tracing* (USDT)-Provider
- verfügbar für Java (JDK 1.4, 5, 6)
- Probes für VM, Threads, Classloading, Garbage Collection, Methoden-Übersetzung, -Laden, -Aufruf, Monitore, Objekt-Allokation
- viele andere als Prototypen verfügbar: JavaScript, PHP, Apache, PostgresQL, X-Server, …

DTrace-Toolkit

- Sammlung zahlreicher `D`-Skripte
- nutzbar als fertige Tools als auch als Beispiele
- `http://www.opensolaris.org/os/community/dtrace/dtracetoolkit`
- z.B. `dvmstat`, `errinfo`, `execsnoop`, `iosnoop`, `iotop`, `opensnoop`, `rwsnoop`, `rwtop`, `tcpsnoop`, `tcptop`, ...

Weitere Informationen

- OpenSolaris-DTrace-Community:
  `http://www.opensolaris.org/os/community/dtrace/`
- Solaris Dynamic Tracing Guide:
  `http://docs.sun.com/app/docs/doc/817-6223`
- Cantrill, Shapiro, Leventhal, Dynamic Instrumentation of Production Systems, Proceedings of the 2004 Usenix Annual Technical Conference, p. 15
- Cantrill: Hidden in Plain Sight, ACM Queue 4(1), Feb. 2006, p. 26