Das DTrace-Toolkit als Diagnosewerkzeug

Jan Brosowski
Oracle Deutschland B.V. & Co. KG
Walldorf

Schlüsselworte
Solaris 11, MacOS X, Oracle Linux, FreeBSD, DTrace, DTrace-Toolkit, DTT, Performance-Analyse, System-Analyse,

Einleitung

Seit seiner Veröffentlichung im Jahr 2003 seiner Einführung in Solaris im Jahr 2005 verbreitet sich DTrace in immer mehr unixoiden Betriebssystemen als umfassendes, leistungsstarkes Analysewerkzeug. Mit über 80.000 sogeannten Probes ermöglicht es dem Benutzer, präzise und genau Parameter und Messwerte im Betriebssystem zu erheben praktisch ohne Einfluss auf die Performance und das Systemverhalten. DTrace bietet dazu mit seiner eigenen Sprache d umfangreiche Möglichkeiten der Auswertung.

Diese Stärken von DTrace sind zugleich die größten Hürden für Anfänger in der Materie. Umfangreiche Möglichkeiten gehen einher mit einem langwierigen Lernprozess. So fehlt beispielsweise bei Performance-Probleme in Produktionsumgebungen die Zeit für intensives Lernen, oder die täglichen Pflichten und Aufgaben eines Administrators machen es schwierig, die Zeit für das Lernen aufzuwenden.

Das DTrace-Toolkits bietet nun eine Möglichkeit, für viele wiederkehrende Analyseaufgaben vorgefertigte d-Skripte zu verwenden. Somit kann der Benutzer schnell Analysen durchführen, ohne sich intensiv mit DTrace zu beschäftigen. Zudem sind diese Tools ein guter Ausgangspunkt für das Erlernen von DTrace, da alle Skripte gut kommentiert sind und daher nachvollzogen werden können.

Überblick über DTrace

Die Notwendigkeit zu verstehen, was in einem Software-System wie einem Betriebssystem genau passiert, ist eine kontinuierliche Herausforderung der IT. Daher gibt es schon seit jeher Tools wie:

- **Debugger** für die verschiedenen Programmiersprachen, beispielsweise in Entwicklungsplattformen wie dem Oracle Solaris Studio enthalten
- **Tools auf Betriebssystemebene**, die das Gesamtsystem betrachten, beispielsweise `iostat(1)`, `vmstat(1)` oder `mpstat(1)`
- **Tools auf Prozess- oder Threadebene**, beispielsweise `top(1)`, `ps(1)` oder `prstat(1)`

All diese Werkzeuge zeichnen sich dadurch aus, dass sie mit ihren diversen Messpunkten entweder einen nicht unerheblichenEinfluss auf das Verhalten der Software, und damit auf die Messung selbst haben (insbesondere die Debugger haben einen sehr großen Einfluss). Oder sie stellen nur oberflächliche statistische Daten bereit, die keinen tiefgehenden Einfluss in das Verhalten des Systems gewähren. Dennoch sind gerade die Tools auf Betriebssystem- und Prozessebene gute Einstiegspunkte in die Analyse von Performance-Problemen.

DTrace setzt auf Betriebssystemebene an und stellt Messpunkte bereit, um Programme und deren Ein- und Ausgabewerte zu überwachen, um den Zugriff auf Geräte zu verstehen oder herauszufinden, für welche Aufgaben der Kernel des Betriebssystem wieviel Zeit verwendet. Gerade bei Performance-Analysen sind diese vielfältigen Messpunkte sehr nützlich, da sie unterschiedliche Blickwinkel auf das gleiche Programm ermöglichen. DTrace ergänzt somit die altbekannten Methoden.

Ein deutlicher Unterschied ist die Art und Weise, wie DTrace seine Messpunkte, genannt Probes, nutzt. Diese werden dynamisch aktiviert, also im laufenden Betrieb. Werden sie nicht benötigt, werden die zugehören Code-Pfade auch nicht ausgeführt, sodass die existenz einer Probe keinen Einfluss auf die Performance hat. Daher können sehr viele Probes vorhanden sein, ohne dass darunter die Performance leidet. Nur wenn ein Probe aktiviert wird, erhöht er marginal den Resourcenbedarf.

Ein zweites Merkmal von DTrace ist die weite Verbreitung der Probes im Betriebssystem. Dies ermöglicht zunächst einen einheitlichen Ansatz zur Analyse von Performance-Problemen – es wird immer auf die gleiche Art und Weise vorgegangen. Zusätzlich vereinfacht es aber auch das Korrelieren von Ereignissen, da alle Probes letztlich miteinader für Analysen kombiniert werden können. Heute ist DTrace in FreeBSD, MacOS X, Linux und Solaris zu finden. Auf einem normalen Macbook Pro finden sich in einem aktuellen MacOS X beispielsweise über 100.000 DTrace Probes, verteilt in Kernel, Treibern und betriebssystemnahen Programmen.

Einhergehend mit der großen Anzahl von Messpunkten ergab sich die Notwendigkeit einer geeigneten Methode, die Probes zu nutzen und die Messungen aufzubereiten. Aufgrund der vielfältigen Möglichkeiten, die sich aus der schieren Zahl an Probes und deren Verknüpfungen ergab, wurde neben dem direkten Zugriff auf Probes mit dem Befehl `dtrace(1)` eine eigene Skriptsprache namens „d“ implementiert. Diese Sprache erlaubt es, statt langer, unübersichtlicher Befehlzeilen gut geliedert in Skripten Probes zu adressieren und deren Messwerte aufzubereiten.

Historie des DTrace-Toolkits

Schon bald nach dem Erscheinen von DTrace fingen Benutzer an, d-Skripte untereinander weiterzugeben und in verschiedenen Foren auszustauchen, vorwiegend im Umfeld von Open Solaris. Brendan Gregg, ein Performance-Experte und Autor diverser Publikationen rund um DTrace , sammelte diese Skripte, kommentierte sie und fasste sie in einer zunächst losen, später immer besser dokumentierten Sammlung namens DTrace-Toolkit zusammen. [1]

Die Open-Source-Sammlung DTrace-Toolkit existiert heute noch, doch ist sie heute mit einem Problem konfrontiert: Zwar gibt es eine vielzahl von Messpunkten in allen Betriebssystemen, die DTrace unterstützen, doch ist deren Namensgebung bis heute leider nicht vollständig standardisiert. Folglich müssen, gerade wenn seltener verwendete Probes genutzt werden, die Skripte an die Betriebssysteme angepasst werden.

Das DTrace-Toolkit wurde schon kurz nach seinem Erscheinen als Paket in Open Solaris aufgenommen, und fand so seinen Weg in Solaris 11. Oracle hat das Paket in den normalen Funktionsumfang von Solaris übernommen und supportet das Toolkit vollständig.Es kann daher von jedem Solaris Anwender genutzt werden, auch auf isolierten Systemen, bei denen Pakete nur aus

[1] Brendan Gregg hat auch an der Entwicklung der ZFS Storage Appliances von Sun Microsystems mitgewirkt, dort insbesondere an der Integration von dtrace in die Analytics. Dabei erreichte er auch einen gewissen Bekanntheitsgrad mit dem Video „Shouting in the Datacenter“, welches über 1.000.000 views erzielt hat. https://www.youtube.com/watch?v=tDacjrSCeq4

offiziellen Quellen installiert werden dürfen. Zudem entfällt die Anpassung des Toolkits an Solaris durch den Endanwender, da Oracle diese Arbeit übernimmt.

Erste Schritte mit dem DTrace-Toolkit

Das DTrace-Toolkit gehört ebenfalls wie DTrace zu den Bestandteilen der meisten Solaris-Installationsprofilen. Lediglich in „minimal“, oder darauf aufbauenden eigendefinierten Profilen, ist es nicht enthalten. Es kann aber einfach nachinstalliert werden.

```
root@solaris:/# pkg install dtrace-toolkit
```

Gegebenenfalls werden weitere Abhängigkeiten – unter anderem DTrace selbst – mitinstalliert.

Das Toolkit wird nach `/usr/dtrace/DTT` installiert, und ein Blick auf dieses Verzeichnis zeigt die Einordnung der verschiedenen Skripte nach ihren Aufgaben. Die Benutzung der Skripte kann man grob in zwei Kategorien einteilen:

- Einige Skripte funktionieren, wie man es von Shell-Skripten gewohnt ist. Man ruft den Befehl auf, übergibt einige Parameter.

- Andere Skripte sind klassische, „echte“ d-Skripte, die mit dem `dtrace -s` Befehl aufgerufen werden.

In den verschiedenen Verzeichnissen liegen die unterschiedlichen Skripte, geordnet nach ihren Aufgaben. Hierzu zählen beispielsweise

- Häufig sinnvolle Tools liegen direkt in diesem Verzeichnis, beispielsweise `iotop` (ähnlich wie top, aber auf das IO-System bezogen) `opensnoop` (zeigt an, welcher Prozess gerade welche Datei öffnet), `statsnoop` (zeigt, welcher Prozess welche Statusänderungen im Filesystem verursacht) oder `procsystime` (zeigt, wie lange welche Systemcalls in einem Zeitraum für einen oder mehrere Prozesse gelaufen sind).

- Einige Skripte dienen zur Analyse verschiener Skript- und Programmiersprachen. So finden sich Skripte für Python, Java, Ruby, Javascript oder PHP. Diese setzen voraus, dass in den jeweiligen Umgebungen wie der JAVA VM oder der python interpreter entsprechende DTrace Probes vorhanden sind. Daher funktionieren sie nur mit den entsprechenden Solaris-Paketen, oder es müssen Extensions für die Sprache installiert werden.
  Bei einigen Sprachen muss für die Ausführung teilweise eine Option übergeben weden, damit die DTrace Probes auch scharf geschaltet werden. So müssen beispielsweise Java Applikationen folgendermaßen gestartet werden:
  `java -XX:+ExtendedDTraceProbes classfile`
  Genaue Hinweise, welche Programmiersprache welche Erweiterung benötigt, findet man in den `Readme`-Dateien in den jeweiligen Verzeichnissen.

- Die Verzeichnisse Mem, Net, Cpu, Disk sind mit dem gefüllt, was ihre Namen andeuten: Skripte zur Analyse von Hauptspeicher, Netzwerk, CPU und Disk. Diese Skripte funktionieren ähnlich wie die Tools im Hauptverzeichnis direkt da sie nur auf Probes im Betriebssystem zurückgreifen.

Es existieren man-Pages für alle Skripte des Toolkits, diese liegen im Unterverzeichnis `/usr/dtrace/DTT/Man`. Dieses wird bei der Installation nicht in den manpath eingetragen, was man aber nachholen kann. [2]

Beispiel der Anwendung des DTrace-Toolkits

Das Beispiel aus dem Vortrag – die Analyse eines „wildgewordenen" SAP-Systems – soll hier nur stichwortartig als Hilfestellung zum Nachvollziehen dargestellt werden.

Die Ausgangssituation gestaltete sich so, dass ein Solaris-Server ein Storage Array derart unter Last nahm, dass andere Applikationen unter Performance-Einbußen litten. Eine erste Analyse des Storage-Administrators im Storage-Array zeigte, dass die LUN eines SAP-Systems eine sehr hohe Anzahl von IO-Operationen durchführen musste.

Der Administrator des Solaris-Servers, der das SAP-System beheimatet, sah in seinen ersten Untersuchungen mit `prstat` und `iostat` nur, dass die Datenbank und Applikationsserver des SAP-Systems praktisch keine Systemlast erzeugten, der Server aber in der Tat eine große Anzahl von Request auf die LUN bzw. die darauf befindlichen Filesysteme „feuerte".

Die Aufgabe war also, zunächst herauszufinden, welche Prozess(e) auf dem fraglichen Filesystem die Last erzeugte – und warum diese nicht durch die diversen Caches des ZFS aufgefangen werden konnten. Eine erste Analyse mit `iotop` aus dem DTrace-Toolkit, zeigte dann, dass ein ZFS Filesystem namens „ClientShare" auf der LUN die große Anzahl an IO-Operationen erzeugte. Mit `opensnoop` konnte dann nachvollzogen wurden, dass die Zugriffe per CIFS nicht vom Server selbst, sondern von diversen Clients kommen und vom Server nur weitergegeben wurden.

Es zeigte sich, dass es immer die gleichen Files waren, auf die zugegriffen wurden – es handelt sich um die Files, mit denen das SAP-GUI an alle Windows-Clients ausgeliefert wird. Die Problemursache war also nicht der Server selbst, sondern sie war auf den Clients zu finden. Ein Gespräch mit dem zuständigen Windows-Administator förderte die Erkenntnis zu tage, dass auf allen Windows-Clients ein fehlkonfigurierter Virenscanner den Share scannte, und zwar so gründlich wie eben möglch. Dies erklärte die hohe Anzahl an Operationen, nicht aber das vermutliche Versagen der ZFS Caches.

Auch hier konnten das DTrace-Toolkit abhelfen: Die Caches waren schlicht zu klein dimensioniert, um zusätzlich zu den durch das SAP-System benötigten heißen Datenfiles einige Gigabyte an SAP-GUI-Dateien vorzuhalten. Der Memory des Servers war eher knapp gesized, und sein Hauptspeicher war zu großen Teilen für SGA und die SAP-Applikationsserver reserviert.

Nicht mit dem DTrace-Toolkit erklärt werden konnte allerdings, warum das Storage-Array nicht selbst die Daten in seinen Cache packte.

Weitere Schritte

Für Programmierer bietet es sich an, sich DTrace in die IDE zu integrieren. Bei MacOS X ist dies in der Umgebung xcode bereits im Standard enthalten, hier findet sich eine Umfangreiche Sammlung von DTrace -Skripten mit Visualisierung im Bereich „Instruments".
Ähnliches gilt für die Oracle Solaris Studio-Umgebung, dort ist DTrace in den Collector und Performance Analyzer integriert.

---

[2] `export MANPATH=$MANPATH:/usr/dtrace/DTT/Man` ; in der `Readme` im Verzeichnis finden sich weitere, nicht ganz ernst gemeinte Hinweise.

Nutzer von Netbeans müssen hingegen die DTrace -Unterstützung erst in ihre Umgebung als Plugin integrieren.

Zusammenfassung

Das DTrace-Toolkit unterstützt Administratoren mit einer umfangreichen Sammlung an Tools zur Performance-Messung. Durch die Integration sowohl in das Betriebssystem als auch in verschiedenene Skript- und Programmiersprachen ist der Administrator in der Lage, Performance-Probleme einfach und doch tiefgehend zu analysieren.
Zudem bietet das Toolkit einen guten Einstieg, um eigene d-Skripte zu schreiben, da die vorhandenen Skript gut dokumentiert sind.

Kontaktadresse:

Jan Brosowski
Oracle Deutschland B.V. & Co. KG
Altrottstraße 31
D-69190 Walldorf

Telefon: +49 (0) 6227 -356 201
E-Mail jan.brosowski@oracle.com
Internet: www.oracle.com