Profiler

Voreilige Optimierung ist bekanntlich eine schlimme Falle, in die vor allem Juniorprogrammierer tappen. Doch was tun, falls ein Skript eindeutig zu langsam läuft? Oft lässt sich mit wenig Aufwand viel gewinnen, wenn man den Hebel an der richtigen Stelle ansetzt.

Zunächst einmal gilt es festzustellen, in welchen Programmteilen am meisten Zeit verbraten wird. Hierzu wird das Skript `foo` mit dem Profiler `Devel::DProf` vom CPAN mit `perl -d:DProf foo` aufgerufen. Es läuft so etwas langsamer als normal, während der Profiler Daten über die abgearbeiteten Funktionen sammelt. Ein anschließender Aufruf des Programms `dprofpp` aus der `Devel::DProf`-Distribution liest dann die vom Profiler angelegte Datei `tmon.out` aus und zeigt den Inhalt wie in Abbildung 1 gezeigt an.

Dort ist ersichtlich, dass das Skript 96.6% der Zeit in der Funktion `expensive()` verbringt. Rechts in Abbildung 2 ist der Source Code des Skripts zu sehen. Die Funktion versucht jeweils 1000 Mal vergeblich, einen regulären Ausdruck mit einem String in Einklang zu bringen.

Um herauszufinden, in welchen Zeilen einer Funktion ein Skript CPU-Zeit vergeudet, kommt ein Line-Profiler zum Einsatz. Das Modul `Devel::SmallProf` vom CPAN analysiert beliebige Skripte zur Laufzeit und zeigt hinterher genau an, wie lange der Interpreter in jeder Zeile des Skripts verharrte.
Hierzu wird das Skript mit `perl -d:SmallProf foo` aufgerufen. Zu beachten ist, dass der Line-Profiler `Devel::SmallProf` erheblich langsamer ist, sodass man eventuell nur Teile davon untersuchen kann. Das Ergebnis liegt nach Ablauf des Skripts fertig formatiert in der Datei `smallprof.out` vor.

```
mschilli@mybox:~/DEV/articles/profiler/eg
$ perl -d:DProf foo
$ dprofpp
Total Elapsed Time = 0.107958 Seconds
  User+System Time = 0.117958 Seconds
Exclusive Times
%Time ExclSec CumulS #Calls sec/call Csec/c  Name
 96.6   0.114  0.114   1000   0.0001 0.0001  main::expensive
 0.00       - -0.000      1        -      -  strict::bits
 0.00       - -0.000      1        -      -  strict::import
 0.00       - -0.000      1        -      -  main::BEGIN
$
```

Abb. 1

Abbildung 2 zeigt, dass die Zeile mit dem Regex-Match insgesamt 10.000 Mal durchlaufen wurde. Die Anzahl der *wallclock seconds* entspricht der Anzahl der tatsächlich verstrichenen Sekunden, wenn man alle Aufrufe der Zeile zusammenrechnet. `CPU seconds` hingegen geben die verbrauchte Prozessorzeit an. Bei einem `sleep(1)`-Befehl wäre zum Beispiel der erste Wert ziemlich genau eine Sekunde pro Aufruf, während die verbrauchte CPU-Zeit nahezu Null wäre.

Wer übrigens meint, der regulären Ausdruck `/$m1/` ließe sich durch den Modifizierer `/o` beschleunigen, täuscht sich. `/o` steht für *once* und bewirkt, dass ein regulärer Ausdruck, der eine Variable enthält, nur einmal kompiliert wird. Mit ihm nimmt Perl dann an, dass sich der Wert der Variablen nicht ändert. Allerdings merkt sich das Perl beim zuletzt bearbeiteten regulären Ausdruck schon automatisch. In einer Schleife wie in der Funktion `expensive()` bringt das also nichts.

Mit einem Haudegen-Trick lässt sich das Skript allerdings erheblich beschleunigen: Wenn man mit

```
use Memoize;
memoize('expensive');
```

festlegt, dass die Funktion `expensive` immer die gleichen Werte zurückliefert, wenn man sie mit den gleichen Argumenten aufruft, wird sie nur noch einmal aufgerufen und die restlichen neun Mal 'weiß' `Memoize` bereits das Ergebnis. Und schon läuft das Programm 10x schneller!

Mike Schilli

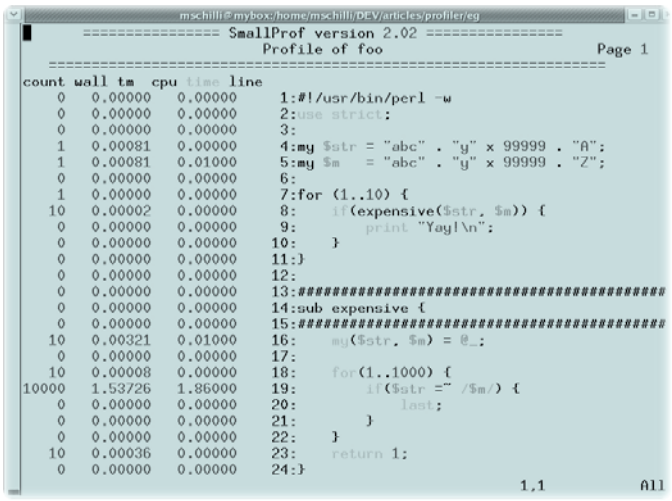

Abb. 2

Schweizer Team ist Sieger in der Kategorie Perl beim "Plat_Forms" Web-Programmier-Wettbewerb

Deutscher Perl-Workshop gratuliert – Besonders der kompakte und leicht erweiterbare Code beeindruckt

Perl erzeugt kompakten und einfach zu erweiternden Code – das bestätigt die Studie "Plat_Forms 2007: The Web Development Platform Comparison" der Freien Universität Berlin.
"Der Entwicklungszyklus ist mit Perl besonders schnell und die Sprache ist sehr flexibel. Für Unternehmen, die auf schnelle Entwicklung und leistungsfähige Software Wert legen, stellt Perl eine echte Alternative dar", so Cedric Bouvier vom schweizer Team Etat de Genève/Optaros, das in der Kategorie Perl den Wettbewerb Plat_Forms 2007 gewonnen hat.
Die anderen Perl-Teams – plusW aus Deutschland und Revolution Systems aus den USA – folgen mit knappem Abstand, die Entscheidung ist der Jury schwer gefallen.
Die teilnehmenden Perl-Teams hoffen, dass die weitreichenden Fähigkeiten der Perl-Plattform zur professionellen Web-Entwicklung nun mehr Verbreitung finden. "Dazu könnten auch Fachhochschulen und Universitäten beitragen, indem sie Perl häufiger als bisher in ihre Lehrpläne aufnehmen", so Dami Laurent, ebenfalls vom Team Etat de Genève/Optaros.

Der Deutsche Perl-Workshop gratuliert den erfolgreichen Teams zu ihrer guten Arbeit.

Perl wird zur Web-Entwicklung beispielsweise bei der Business-Plattform XING (vormals openBC), beim Nachrichten-Portal heise online und der Technik-Community Slashdot eingesetzt.