Herbert Breunung

Dumbbench - eine intelligente Stoppuhr

Ein stets guter Rat: Optimiere nur den Code, der nachgewiesen langsam ist. Für solch einen Nachweis gibt es seit langem das Kernmodul `Benchmark`, welches den Abstand zwischen zwei Zeitpunkten misst. Dem Perl 5-Porter Steffen Müller war es jedoch nicht gut genug und er schuf mit `Dumbbench` eine Alternative, die den Programmieraufwand senken und die Ergebnisse realistischer machen soll - auf Kosten zusätzlicher Rechenzeit. Selbst wenn der Programmierer sich dumm stellt oder die Arbeit scheut, sollten die Ausgaben immer noch nützlich sein. Dies war zumindest der Grundgedanke, welcher zu dem Namen führte.

Warum?

Wie vieles in Perl, so berufen sich auch die Funktionen von Benchmark auf die UNIX-Tradition. Nicht nur die grundlegende Einteilung der Ausgabe in *real*-, *user*- und *sys*-Zeit der UNIX-Kommandos `time` und `times` wurde weitergeführt. Es werden auch hinter dem Vorhang gleichnamige Perlbefehle mit ebenso niedriger Genauigkeit verwendet. Nur mit gesetztem Exporter-Tag `:hireswallclock` gibt es die durch das Modul `Time::HiRes` bereitgestellte, höhere Genauigkeit von mindestens Tausendstel Sekunden. Dumbbench bricht damit. Es stützt sich ausschließlich auf hochauflösende Zeitangaben und zeigt auch nur die *real* vergangene "Wanduhrzeit" an. Das ist jene Zeit, die auch jede Uhr außerhalb des Computers gemessen hätte. Die Überlegung dahinter: Um die wirklich echte Zeit, die ein Stück Code benötigt zu ermitteln, braucht es wesentlich komplexere Überlegungen als nur den Abzug der Zeit, welches das Betriebsystem und andere Prozesse zwischenzeitlich verbraucht haben. *IO*-Geräte wie Festplatten arbeiten bei gleichartigen Zugriffen nicht immer gleich schnell. Aber auch der Prozessverwalter des Betriebsystems (Scheduler) und Prozessoren besitzen ihre Eigendynamik, um Ergebnisse zu verfälschen. Deshalb wählte Müller den Ansatz, den Code genügend oft laufen zu lassen und die Zeiten von Ausreißern, die zu sehr vom Mittelwert (Median, nicht der Durchschnitt) abweichen, auszusortieren. Der Mittelwert der Abweichungen (der übrig gebliebenen Zeiten) vom Mittelwert ist zudem hilfreich um auszurechnen, wie zuverlässig der Endwert in etwa ist. Der Median bezeichnet den Wert an der mittleren Position einer sortierten Folge.

Eine genauere Beschreibung der statistischen Operationen gibt es im zugehörigen Blogbeitrag [1]. Zum Glück ist der Autor ein studierter Physiker und hat Erfahrungen damit, Datenreihen nach Verwertbarem abzugrasen.

Benchmark::Dumb

Praktischerweise fügte er der Distribution zudem das Modul `Benchmark::Dumb` [2] hinzu, dessen Funktionen sehr weit denen von `Benchmark` ähneln. Man erkennt deutlich, dass es als Ersatz gedacht ist. `timeit` dient der stillen Zeitmessung, `timethis` ist die Messung mit Ausgabe. Mehrere Aurufe von Letzterem lassen sich mit `timethese` zusammenfassen. `cmpthese` wird gewählt, wenn es nicht um die Zeiten, sondern um prozentuale Unterschiede geht. Deshalb ist es auch nur konsequent, dass `Dumbbench` angewiesen werden kann, die Anzahl der Wiederholungen selbst zu bestimmen, bis eine gewünschte Messgenauigkeit erreicht wurde. Dazu muss der erste Parameter lediglich Nachkommastellen bekommen. Der ganzzahlige Anteil ist nach wie vor die garantierte Anzahl an Durchläufen, negative Angaben sind nicht gestattet.

```
                          Rate Erestothenes n Erestothenes e
Erestothenes n   14.95+-0.011/s             --           -5.9%
Erestothenes e 15.894+-0.018/s     6.32+-0.14%              --
```

Listing 1

Zum Beispiel wäre 0.005 der Befehl, so viele Wiederholungen einzulegen, bis die Messgenauigkeit bei 0,5 Prozent der Lauflänge des Programmstücks liegt. Nie wieder die Fehlermeldung *warning: too few iterations for a reliable count* und ein Hoch auf die Programmiertugend der intelligenten Faulheit! Durch unsinnige Angaben von zu vielen Wiederholungen oder einer schwer erreichbaren Genauigkeit sind Eigentore aber immer noch nicht ausgeschlossen.

Wie bei `Benchmark` liefert `cmpthese` die Tabelle (siehe Listing 1) der Ergebnisse als Datenstruktur (Array von Arrays) inklusive der geschätzten Genauigkeit.

Die anderen Funktionen liefern ein echtes Objekt. Dessen Methoden geben die Informationen der Messung preis, und können auch die Summe oder Differenz zu einem anderen Messungsobjekt berechnen. Das ist ein weiterer deutlicher Unterschied zu `Benchmark`, das seine Informationen in einem Array lagert, welcher prozedural weiterverarbeitet wird.

Ein dritter eher subtiler Unterschied liegt im Namensbereich, in dem der Beispielcode ausgeführt wird. `Dumbbench` tut es in seinem eigenen. Das hat Auswirkungen, wenn man den Beispielcode nicht als Closure (Block), sondern als später zu evaluierenden String schreiben möchte. Um zu testen wie schnell Perl Zweierpotenzen berechnet, muss man deshalb statt

```
use Benchmark::Dumb qw(:all);

my $i = 3;
timethis(0.02, sub { 2 ** $i++ });
```

folgendes schreiben:

```
our $i = 3;
timethis(0.02, '2 ** $::i++');
```

Dumbbench

Während `Benchmark::Dumb` wie gezeigt Stücke von Perlcode messen kann, ist `Dumbbench` dazu da, ganze Skripte wiederholt laufen zu lassen. Die kleine aber sinnvolle API ist in der Dokumentation gut ausgeschildert und braucht hier nicht wiederholt zu werden. Interesse wecken könnte das mitgelieferte Skript, welches in der Kommandozeile wie `time` eingesetzt wird.

```
dumbbench -p 0.005 -- perl meinskript.pl
```

Durch gut gewählte Defaults (20 Durchläufe, 5% Präzision) reicht meist ein:

```
dumbbench perl report.pl
```

was eine schöne Ausgabe auswirft, die im Unterschied zu `Benchmark::Dumb` nicht nur die Anzahl der Wiederholungen sondern auch die abgelehnten Ausreißer beinhaltet. Die Distribution ist auf jeden Fall einen Probelauf wert. Sie ist mittlerweile zweieinhalb Jahre in Entwicklung, stabil, gut getestet und die Quellen sehen auch solide aus. Die Objekte sind mit `Class::XSAccessor` realisiert, daher sehr schnell und kompakt und die Handvoll eher kleinerer Abhängigkeiten dürfte nicht abschrecken. Im Gegenteil, das ebenfalls von Steffen Müller geschriebene `Statistics::CaseResampling` könnte noch für eigene Datenanalysen nützlich werden.

Links

[1] *http://blogs.perl.org/users/steffen_mueller/2010/09/your-benchmarks-suck.html*
[2] *http://metacpan.org/module/Benchmark::Dumb*