Pontificating on Perl Profiling

Lisa Hagemann
VP Engineering, Dyn Inc.
twitter.com/lhagemann
twitter.com/dyninc
What is Profiling?

- A way to evaluate what is your program doing.
- Commonly evaluates the behavior of the program, measuring frequency and duration of different call paths including the call paths.
- Used to evaluate areas ripe for optimization.
What is Benchmarking?

- Defining a measurement for comparison
- Benchmark time, memory, database calls
- Provides data, not answers
There's More Than One Way To Do It
What's the Best way to do it?

use Benchmark;
use Benchmark;

perldoc Benchmark

Built in module which encapsulates a number of routines to help you figure out how long it takes to execute some code.
use Benchmark;
Built in module encapsulates a number of routines to help you figure out how long it takes to execute some code.

timethese (COUNT, CODEHASHREF,[STYLE])
Time COUNT iterations of CODEHASHREF.

cmpthese ( COUNT, CODEHASHREF, [ STYLE ] )
or
cmpthese
( RESULTSHASHREF, [ STYLE ] )
Uses timethese (or the results of a timethese()) call and outputs in a comparison table.
$ perl simpleloop2.pl
Benchmark: timing 1000000 iterations of first, loop...
  first: 1.59767 wallclock secs ( 1.48 usr +  0.01 sys =  1.49 CPU) @
  671140.94/s (n=1000000)
  loop: 1.08002 wallclock secs ( 0.92 usr +  0.01 sys =  0.93 CPU) @
  1075268.82/s (n=1000000)

         Rate first  loop
     first  671141/s    --  -38%
       loop  1075269/s   60%    --

output from timethese() is the default style ‘auto’
key of coderef followed by the times for 'wallclock' time,
user time, and system time followed by the rate

output from cmpthese() gives us a comparison chart sorted
from slowest to fastest, and shows the percent speed
difference between each pair of tests.
Things to consider

Focus on code that will be executed the most (think loops)

- Are there expensive comparisons/computations that can be cached sensibly?
- Are there chains of comparisons that aren't optimized statistically?

Unnecessary sorting?

Are you reinventing the wheel?
A simple text parsing script

- Uses a package for creating objects
- Simple parsing of a zone file into DNS records: hash ‘em if we know how
- 20K lines to parse
#!/usr/bin/env perl -l
use strict;
use warnings;
use RR;
use Net::DNS::RR;
use Benchmark qw(:hireswallclock);

my $t0 = new Benchmark;
while (my $line = <>) {
    chomp($line);

    # Ignore blank lines
    next unless $line;
    my $obj = RR->new($line);
    my $rr;
    # Generate Net::DNS::RRs
    my $rr;
    if ($obj->as_hash()) {
        $rr = Net::DNS::RR->new($obj->as_hash());
    } else {
        $rr = Net::DNS::RR->new($obj->as_string());
    }
}
my $t1 = new Benchmark;
my $runtime = timestr(timediff($t1, $t0));
print "Zone parse time: $runtime";

use Benchmark;
Built in module encapsulates a number of routines to help you figure out how long it takes to execute some code.

new():
Returns the current time as an object the Benchmark methods use

timediff( T1 , T2 ): A Benchmark object representing the difference between two Benchmark times, suitable for passing to timestr();
timestr( TIMEDIFF, [ STYLE, [ FORMAT ] ] ): returns a string in the requested format suitable for printing. Format defaults to ‘%5.2f’.
4.61 seconds to process the file

Good? Bad? Can it be better?
Profiling Packages

**Devel::DProf**
- Built in, produces an output file, utility to format that
- Watches subroutine calls noting elapsed time
- Totals each run into a total time spent in the subroutine

**Devel::SmallProf**
- Install from CPAN
- Human readable output file, clunky for programs with imported libraries

**Devel::NYTProf**
Devel::NYTProf
http://search.cpan.org/~timb/Devel-NYTProf-4.06/

Devel::NYTProf from CPAN is a powerful, fast, feature-rich Perl source code profiler*

Statement and Subroutine profiling showing Inclusive and Exclusive Time

- Inclusive includes time spent in subroutines called from within another subroutine

- Handy report HTML generator
Run the script with `Devel::NYTProf`

```
$ perl -d:NYTProf zoneparse.pl zone.com.txt
```

- `-d` flag starts the debug mode which is shorthand for `-MDevel::`
- loads the module `Devel::NYTProf` before running the provided script
- produces `nytprof.out` file
- Adds a little overhead
nytprofhtml generates HTML report

Useful flags for keeping multiple runs

- `-f --file`: file name to use; defaults to `./nytprof.out`
- `-o --out`: the output directory to place all the HTML files
Performance Profile Index
For zoneparse.pl

Profile of zoneparse.pl for 13.0s (of 19.3s), executing 8071444 statements and 2414346 subroutine calls in 48 source files and 8 string evals.

Top 15 Subroutines

<table>
<thead>
<tr>
<th>Calls</th>
<th>P</th>
<th>F</th>
<th>Exclusive Time</th>
<th>Inclusive Time</th>
<th>Subroutine</th>
</tr>
</thead>
<tbody>
<tr>
<td>90041</td>
<td>1</td>
<td>1</td>
<td>2.31s</td>
<td>2.69s</td>
<td>Net::DNS::presentation2wire</td>
</tr>
<tr>
<td>60037</td>
<td>1</td>
<td>1</td>
<td>1.11s</td>
<td>1.23s</td>
<td>Net::DNS::wire2presentation</td>
</tr>
<tr>
<td>40019</td>
<td>2</td>
<td>2</td>
<td>592ms</td>
<td>3.28s</td>
<td>Net::DNS::name2labels</td>
</tr>
<tr>
<td>90018</td>
<td>9</td>
<td>1</td>
<td>505ms</td>
<td>615ms</td>
<td>Net::DNS::Header::AUTOLOAD</td>
</tr>
<tr>
<td>10006</td>
<td>1</td>
<td>1</td>
<td>467ms</td>
<td>2.76s</td>
<td>Net::DNS::RR::new_from_string</td>
</tr>
<tr>
<td>20004</td>
<td>2</td>
<td>2</td>
<td>429ms</td>
<td>1.53s</td>
<td>Net::DNS::Packet::dn_comp</td>
</tr>
<tr>
<td>10002</td>
<td>1</td>
<td>1</td>
<td>359ms</td>
<td>3.38s</td>
<td>Net::DNS::Packet::data</td>
</tr>
<tr>
<td>10002</td>
<td>1</td>
<td>1</td>
<td>350ms</td>
<td>1.99s</td>
<td>Net::DNS::RR::data</td>
</tr>
<tr>
<td>624759</td>
<td>2</td>
<td>1</td>
<td>348ms</td>
<td>348ms</td>
<td>Net::DNS::CORE::unpack (opcode)</td>
</tr>
<tr>
<td>10002</td>
<td>1</td>
<td>1</td>
<td>333ms</td>
<td>6.38s</td>
<td>Net::DNS::RR::normalize_rdata</td>
</tr>
<tr>
<td>10002</td>
<td>1</td>
<td>1</td>
<td>313ms</td>
<td>8.66s</td>
<td>Net::DNS::RR::new_from_hash</td>
</tr>
<tr>
<td>20004</td>
<td>2</td>
<td>2</td>
<td>269ms</td>
<td>527ms</td>
<td>Net::DNS::Packet::push</td>
</tr>
<tr>
<td>10002</td>
<td>1</td>
<td>1</td>
<td>279ms</td>
<td>1.04s</td>
<td>Net::DNS::Packet::parse</td>
</tr>
<tr>
<td>20015</td>
<td>6</td>
<td>4</td>
<td>241ms</td>
<td>3.65s</td>
<td>Net::DNS::stripdot</td>
</tr>
<tr>
<td>10002</td>
<td>1</td>
<td>1</td>
<td>212ms</td>
<td>841ms</td>
<td>Net::DNS::RR::parse</td>
</tr>
</tbody>
</table>

See all 585 subroutines

Source Code Files — ordered by exclusive time then name

<table>
<thead>
<tr>
<th>Stmts</th>
<th>Exclusive Time</th>
<th>Reports</th>
<th>Source File</th>
</tr>
</thead>
<tbody>
<tr>
<td>4106502</td>
<td>4.96s</td>
<td>line • block • sub</td>
<td>Net::DNS.pm</td>
</tr>
<tr>
<td>1220447</td>
<td>2.42s</td>
<td>line • block • sub</td>
<td>Net::DNS/RR.pm (including 1 string eval)</td>
</tr>
<tr>
<td>1030219</td>
<td>1.54s</td>
<td>line • block • sub</td>
<td>Net::DNS/Packet.pm</td>
</tr>
<tr>
<td>790182</td>
<td>1.05s</td>
<td>line • block • sub</td>
<td>Net::DNS/Header.pm</td>
</tr>
<tr>
<td>290131</td>
<td>534ms</td>
<td>line • block • sub</td>
<td>/Users/bagomann/Documents/Presentations</td>
</tr>
</tbody>
</table>
#!/usr/bin/perl -l

use strict;
use warnings;

use RR;
use Net::DNS::RR;

use Benchmark qw(:hireswallclock);

my $t0 = new Benchmark;

while (my $line = <>) {
    chomp($line);
    next unless $line;
    my $obj = RR->new($line);
    # Generate Net::DNS::RRs
    my $rr = Net::DNS::RR->new($obj->as_string());
}

my $t1 = new Benchmark;
my $runtime = timestr(timediff($t1,$t0));

print "Zone parse time: $runtime";
4.61 seconds down to 2.03 seconds

> 50% speed up! Any others?

$ perl zoneparse2.pl zone.com.txt
Zone parse time: 2.03943 wallclock secs ( 2.01 usr + 0.01 sys = 2.02 CPU)

$ perl -d:NYTProf zoneparse2.pl zone.com.txt
Zone parse time: 10 wallclock secs ( 9.43 usr + 0.03 sys = 9.46 CPU)

$ nytprofhtml -o ./nytprof_run2 -f ./nytprof_run2.out --open
Reading ./nytprof_run2.out
Processing ./nytprof_run2.out data
Writing sub reports to ./nytprof_run2 directory
  100% ...
Writing block reports to ./nytprof_run2 directory
  100% ...
Writing line reports to ./nytprof_run2 directory
  100% ...
Performance Profile Index
For zoneparse2.pl
Run on Tue Feb 15 10:25:47 2011
Reported on Tue Feb 15 10:27:39 2011

Profile of zoneparse2.pl for 6.22s (of 9.61s), executing 4236315 statements and 1326375 subroutine calls in 48 source files and 8 string evals.

```
/Library/Perl/5.10.0/IO/Net/INET6.pm
```

Top 15 Subroutines

<table>
<thead>
<tr>
<th>Calls</th>
<th>P</th>
<th>F</th>
<th>Ex</th>
<th>Inclusive</th>
<th>Subroutine</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Time</td>
<td>Time</td>
<td></td>
</tr>
<tr>
<td>60037</td>
<td>1</td>
<td>1</td>
<td>1.55s</td>
<td>1.81s</td>
<td>Net::DNS::presentation2wire</td>
</tr>
<tr>
<td>60037</td>
<td>1</td>
<td>1</td>
<td>1.14s</td>
<td>1.26s</td>
<td>Net::DNS::wire2presentation</td>
</tr>
<tr>
<td>20008</td>
<td>1</td>
<td>1</td>
<td>921ms</td>
<td>5.27s</td>
<td>Net::DNS::RR::new_from_string</td>
</tr>
<tr>
<td>20015</td>
<td>1</td>
<td>1</td>
<td>371ms</td>
<td>2.18s</td>
<td>Net::DNS::name2labels</td>
</tr>
<tr>
<td>495853</td>
<td>2</td>
<td>1</td>
<td>273ms</td>
<td>273ms</td>
<td>Net::DNS::CORE::unpack (opcode)</td>
</tr>
<tr>
<td>20015</td>
<td>5</td>
<td>4</td>
<td>240ms</td>
<td>3.67s</td>
<td>Net::DNS::stripdot</td>
</tr>
<tr>
<td>10000</td>
<td>1</td>
<td>1</td>
<td>209ms</td>
<td>240ms</td>
<td>Net::DNS::RR::AAAA::normalize_AAAA</td>
</tr>
<tr>
<td>20008</td>
<td>1</td>
<td>1</td>
<td>192ms</td>
<td>362ms</td>
<td>RR::new</td>
</tr>
<tr>
<td>10000</td>
<td>1</td>
<td>1</td>
<td>111ms</td>
<td>127ms</td>
<td>RR::set</td>
</tr>
<tr>
<td>80032</td>
<td>4</td>
<td>1</td>
<td>107ms</td>
<td>107ms</td>
<td>Net::DNS::RR::CORE::match (opcode)</td>
</tr>
<tr>
<td>20008</td>
<td>1</td>
<td>1</td>
<td>102ms</td>
<td>103ms</td>
<td>Net::DNS::RR::get_subclass</td>
</tr>
<tr>
<td>217909</td>
<td>1</td>
<td>1</td>
<td>92.0ms</td>
<td>92.0ms</td>
<td>Net::DNS::CORE::pack (opcode)</td>
</tr>
<tr>
<td>10000</td>
<td>1</td>
<td>1</td>
<td>81.9ms</td>
<td>322ms</td>
<td>Net::DNS::RR::AAAA::new_from_string</td>
</tr>
<tr>
<td>140056</td>
<td>6</td>
<td>1</td>
<td>73.8ms</td>
<td>73.8ms</td>
<td>Net::DNS::RR::CORE::subst (opcode)</td>
</tr>
<tr>
<td>20008</td>
<td>1</td>
<td>1</td>
<td>72.0ms</td>
<td>5.34s</td>
<td>Net::DNS::RR::new</td>
</tr>
</tbody>
</table>

See all 571 subroutines

Source Code Files — ordered by exclusive time then name

<table>
<thead>
<tr>
<th>Stmts</th>
<th>Exclusive Time</th>
<th>Reports</th>
<th>Source File</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2981884</td>
<td>3.30s</td>
<td>line · block · sub</td>
<td>Net/DNS.pm</td>
</tr>
<tr>
<td>720347</td>
<td>1.18s</td>
<td>line · block · sub</td>
<td>Net/DNS/RR.pm (including 1 string eval)</td>
</tr>
<tr>
<td>250115</td>
<td>413ms</td>
<td>line · block · sub</td>
<td>/Users/Ihagemann/Documents/Presentations/PerlProfiling/RR.pm</td>
</tr>
<tr>
<td>80063</td>
<td>310ms</td>
<td>line · block · sub</td>
<td>/Users/Ihagemann/Documents/Presentations/PerlProfiling/zoneparse2.pl</td>
</tr>
</tbody>
</table>
#!/usr/env/bin perl

use strict;
use warnings;

use RR;
use Net::DNS;
use Net::DNS::RR;
use Benchmark qw(:hireswallclock);

sub stripdot {
    my ($str) = @_; 
    # Replace any period at the end of a label that is not escaped by ' \'
    $str =~ s{(?!\)\.)\s*$}{};
    return $str;
}

# override the Net::DNS stripdot
*Net::DNS::RR::stripdot = \&stripdot;

my $t0 = new Benchmark;

my $t1 = new Benchmark;
my $runtime = timestr(timediff($t1,$t0));
print "Zone parse time: $runtime";

$ perl zoneparse3.pl zone.com.txt
Subroutine Net::DNS::RR::stripdot redefined at zoneparse3.pl line 19.
Zone parse time: 1.10183 wallclock secs ( 1.10 usr + 0.00 sys = 1.10 CPU)

2.03 seconds to 1.10 seconds
Another ~50% speed up!

$ perl -d:NYTProf zoneparse3.pl zone.com.txt
Subroutine Net::DNS::RR::stripdot redefined at zoneparse3.pl line 19.
Zone parse time: 3 wallclock secs ( 3.51 usr + 0.02 sys = 3.53 CPU)

$ nytprofhtml -o ./nytprof_run3 -f ./nytprof_run3.out --open
Reading ./nytprof_run3.out
Processing ./nytprof_run3.out data
Writing sub reports to ./nytprof_run3 directory
 100% ...
Writing block reports to ./nytprof_run3 directory
 100% ...
Writing line reports to ./nytprof_run3 directory
 100% ...

Dyn
Profile of zoneparse3.pl for 2.73s (of 3.70s), executing 1315465 statements and 492803 subroutine calls in 48 source files and 8 string evals.

**Top 15 Subroutines**

<table>
<thead>
<tr>
<th>Calls</th>
<th>P</th>
<th>F</th>
<th>Exclusive Time</th>
<th>Inclusive Time</th>
<th>Subroutine</th>
</tr>
</thead>
<tbody>
<tr>
<td>20008</td>
<td>1</td>
<td>1</td>
<td>925ms</td>
<td>1.78s</td>
<td>Net::DNS::RR::new from string</td>
</tr>
<tr>
<td>10000</td>
<td>1</td>
<td>1</td>
<td>211ms</td>
<td>240ms</td>
<td>Net::DNS::RR::AAAA::normalize AAAA</td>
</tr>
<tr>
<td>20008</td>
<td>1</td>
<td>1</td>
<td>197ms</td>
<td>363ms</td>
<td>RR::new</td>
</tr>
<tr>
<td>20008</td>
<td>1</td>
<td>1</td>
<td>115ms</td>
<td>181ms</td>
<td>main::stripdot</td>
</tr>
<tr>
<td>10002</td>
<td>1</td>
<td>1</td>
<td>109ms</td>
<td>124ms</td>
<td>RR::A::set</td>
</tr>
<tr>
<td>80032</td>
<td>4</td>
<td>1</td>
<td>104ms</td>
<td>104ms</td>
<td>Net::DNS::RR::CORE:match (opcode)</td>
</tr>
<tr>
<td>20008</td>
<td>1</td>
<td>1</td>
<td>103ms</td>
<td>104ms</td>
<td>Net::DNS::RR::get_subclass</td>
</tr>
<tr>
<td>10000</td>
<td>1</td>
<td>1</td>
<td>82.5ms</td>
<td>322ms</td>
<td>Net::DNS::RR::AAAA::new from string</td>
</tr>
<tr>
<td>140056</td>
<td>6</td>
<td>1</td>
<td>74.7ms</td>
<td>74.7ms</td>
<td>Net::DNS::RR::CORE:subst (opcode)</td>
</tr>
<tr>
<td>20008</td>
<td>1</td>
<td>1</td>
<td>71.7ms</td>
<td>1.85s</td>
<td>Net::DNS::RR::new</td>
</tr>
<tr>
<td>20008</td>
<td>1</td>
<td>1</td>
<td>66.0ms</td>
<td>66.0ms</td>
<td>main::CORE:subst (opcode)</td>
</tr>
<tr>
<td>10002</td>
<td>1</td>
<td>1</td>
<td>63.3ms</td>
<td>65.3ms</td>
<td>Net::DNS::RR::AAAA::new from string</td>
</tr>
<tr>
<td>20008</td>
<td>1</td>
<td>1</td>
<td>55.8ms</td>
<td>55.8ms</td>
<td>RR::as_string</td>
</tr>
<tr>
<td>10006</td>
<td>1</td>
<td>1</td>
<td>42.4ms</td>
<td>42.4ms</td>
<td>RR::set</td>
</tr>
<tr>
<td>20013</td>
<td>1</td>
<td>1</td>
<td>32.7ms</td>
<td>32.7ms</td>
<td>main::CORE:readline (opcode)</td>
</tr>
</tbody>
</table>

See all 572 subroutines
77% speed up

Total run time: 4.61 seconds down to 1.10 seconds

Time in external module reduced from nearly 5 secs to 6ms.

This includes the overhead of calling the profiling module.
References

- **CPAN**
  - [http://search.cpan.org/~timb/Devel-NYTPprof-4.06/](http://search.cpan.org/~timb/Devel-NYTPprof-4.06/)
  - [http://search.cpan.org/~salva/Devel-SmallProf-2.02/](http://search.cpan.org/~salva/Devel-SmallProf-2.02/)

- **PerlMonks.org**

- **Perl Best Practices** by Damian Conway

- **Modern Perl** by chromatic