mod_perl 2.0: Profiling & Instrumenting

Philippe M. Chiasson
gozer@ectoplasm.org


I TCP/IP
but mostly IP

What is this about?

Profiling

A profiler is a performance analysis tool that measures the behavior of a program as it runs, particularly the frequency and duration of function calls. The output is a stream of recorded events (a trace) or a statistical summary of the events observed (a profile). Profilers use a wide variety of techniques to collect data, including hardware interrupts, code instrumentation, operating system hooks, and performance counters.

Profiling

- Figuring out what a live program is doing
- Looking in from a distance

Instrumenting

A method of collecting data by inserting code into the executable program to count events, such as the number of times a section of the program executes.

Instrumenting

- Modifying code to better watch it
- Writing code that deals with code

First, assumptions

- You have Apache/mod_perl running
- Your code is fast
- You think something is slow
- You think something could be faster
- You are just wondering

First Approach

- "Well, I know our session managment code is slow"
- "It's all the disk access that slows us down, let's get faster drives"
- "The database must be the bottleneck, let's buy something bigger"

Are you sure?

- It's easy to guess what's the bottleneck in a system
- It's easy to get it wrong
- You'll always get it wrong at least part of the time
- Guesswork doesn't have it's place here

Why should I even care?

- Better performance == Cost savings
- Reducing small bottlenecks can have large effects
- Knowing is important, even if you don't do anything about it

Pareto principle

The 80/20 rule

Our Approach

- Facts, not fiction
- Simple tools
- Measurable quantities
- Reliable & trustworthy
- Extensible

Apache::VMonitor

```
$> cpan Apache::VMonitor

httpd.conf:

PerlModule Apache::VMonitor
ExtendedStatus On
<Location /vmonitor>
  SetHandler perl-script
  PerlHandler Apache::VMonitor
</Location>

$> apachectl graceful
```

Apache::VMonitor

Apache::VMonitor

Refresh rate: [0] [1] [5] [10] [20] [30] [60]

```
9/04/2006 1:51pm  up   7.06d, load average: 3.83 1.37 0.95, 265 processes/threads: 7 running
CPU:   14.0% user, 0.1% nice, 1.8% sys, 64.6% idle
Mem:  1010M av,  997M used, 13.0M free,    0K shared,  152M buff
Swap: 1984M av,  408M used, 1576M free,  248K pagein,  211K pageout
```

	PID	Size	Share	VSize	Rss	M	Elapsed	LastReq	Srvd	Client	Virtual Host
0:	3049	87.5M	9.1M	87.5M	32.4M						
14:	9855	89.3M	4.9M	89.3M	29.9M	_	0.000s	0.769s	1	::1	coupler.activestate.com GET / HTTP/1.0
12:	9853	90.6M	5.3M	90.6M	30.6M	W	3.750s	0.000s	0	192.168.69.209	coupler.activestate.com GET /builds/ HTTP/1.0
15:	9857	90.6M	5.3M	90.6M	30.6M	W	1.420s	0.000s	0	192.168.69.209	coupler.activestate.com GET /builds/ HTTP/1.0
8:	9843	93.1M	5.4M	93.1M	33.2M	W	1.490s	0.000s	2	192.168.69.209	coupler.activestate.com GET /builds/ HTTP/1.0
10:	9847	93.1M	5.4M	93.1M	33.2M	W	1.527s	0.000s	1	192.168.69.209	coupler.activestate.com GET /builds/ HTTP/1.0
9:	9846	93.1M	5.4M	93.1M	33.2M	W	1.450s	0.000s	1	192.168.69.209	coupler.activestate.com GET /builds/ HTTP/1.0
6:	9839	93.1M	5.4M	93.1M	33.2M	_	0.000s	2.262s	2	192.168.69.209	coupler.activestate.com GET /builds/ HTTP/1.0
13:	9854	93.1M	5.4M	93.1M	33.2M	W	1.531s	0.000s	1	192.168.69.209	coupler.activestate.com GET /builds/ HTTP/1.0
5:	9832	93.3M	5.4M	93.3M	33.3M	W	1.461s	0.000s	3	192.168.69.209	coupler.activestate.com GET /builds/ HTTP/1.0
11:	9848	94.4M	5.7M	94.4M	34.0M	W	1.483s	0.000s	1	192.168.69.209	coupler.activestate.com GET /builds/ HTTP/1.0
7:	9842	97.0M	5.8M	97.0M	36.5M	W	0.854s	0.000s	2	192.168.69.209	coupler.activestate.com GET /vmonitor
2:	9829	97.0M	5.8M	97.0M	36.7M	W	1.467s	0.000s	4	192.168.69.209	coupler.activestate.com GET /builds/ HTTP/1.0
4:	9831	97.0M	5.8M	97.0M	36.7M	W	0.074s	0.000s	3	192.168.69.209	coupler.activestate.com GET /vmonitor?pid=0
3:	9830	97.0M	5.9M	97.0M	36.8M	_	0.000s	2.288s	3	192.168.69.209	coupler.activestate.com GET /builds/ HTTP/1.0
1:	9828	97.0M	5.9M	97.0M	36.7M	W	1.374s	0.000s	3	192.168.69.209	coupler.activestate.com GET /builds/ HTTP/1.0

```
Total:     1569013K (1496M) size, 1481949K (1413M) approx real size (-shared)
```

Apache::VMonitor

Apache::VMonitor

Refresh rate: [0] [1] [5] [10] [20] [30] [60]

[Back to multiproc mode]

Extensive Status for PID 9855 (httpd)

```
httpd-specific Info:

  Process type              : Child
  Status                    : Waiting for Connection
  Last request processed in :  3.171s
                            :
                            :        This slot         This child
  Requests Served           :                2                  2
  Bytes Transferred         : (    35987)   35K    (    35987)   35K
                            :
  Client IP or DNS          : 192.168.69.209
  Virtual Host              : coupler.activestate.com
  Request (first 64 chars)  : GET /builds/ HTTP/1.0
                            :
  CPU times (secs)          :    total     utime     stime     cutime     cstime
                            :        0         0         0          0          0


General process info:
  UID                       : apache
  GID                       : apache
  State                     :
  TTY                       : None
  Command line arguments    : /usr/sbin/httpd
```

Apache::VMonitor

Apache::VMonitor

Refresh rate: [0] [1] [5] [10] [20] [30] [60]

[Back to multiproc mode]

Extensive Status for PID 9855 (httpd)

```
Memory Usage (in bytes):

  Size        :   98414592 (93.9M)
  Share       :    5791744 ( 5.5M)
  VSize       :   98414592 (93.9M)
  RSS         :   35553280 (33.9M)


Memory Segments Usage (in bytes):

  Text        :     307200 ( 300K)
  Shlib       :          0 (   0K)
  Data        :   36347904 (34.7M)
  Stack       :          0 (   0K)
```

Apache::VMonitor

- + Nice overview of a running server
- + No cost to running it in production
- + Can be used to get a feel for things
- - Is not a good tool to measure accuratly

Resource Profiling

- Good for measuring global quantities:
  - Memory
  - CPU Usage
  - Swapping
  - IO

Resource Profiling

```
package Example;
sub slurp {
  my ($class, $r) = @_;
  my $f = $r->args('f') || "/tmp/motd.txt";
  open (my $fh, $f);
  my $motd = join '',<$fh>;
  print $motd;
  return OK;
}

<Location /slurp>
  SetHandler modperl
  PerlHandler Example->slurp
</Location>

$> GET http://localhost:8529/slurp
$> GET http://localhost:8529/slurp?f=/etc/motd
```

BSD::Resource

```
use BSD::Resource;
($usertime,         # user time
 $systemtime,       # system time
 $maxrss,           # maximum resident size
 $ixrss,            # integral shared memory
 $idrss,            # current unshared data
 $isrss,            # current unshared stack
 $minflt,           # page reclaims
 $majflt,           # page faults
 $nswap,            # swaps
 $inblock,          # input IO
 $oublock,          # output IO
 $msgsnd,
 $msgrcv,
 $nsignals,         # signals recieved
 $nvcsw,            # volountary context switches
 $nivcsw            # involountary context switches
) = getrusage();
```

Resource Profiling

```
package Example::Resource;
use base qw(Example);
sub slurp {
  my ($class, $r) = @_;
  my $rss_before = getrusage()[2];
  my @o = $class->SUPER::slurp(@_); #Call the original code
  my $rss_after = getrusage()[2];
  my $delta = $rss_after - $rss_before;
  my $req = $r->as_string;
  $r->warn("Request for $req increased memory usage by $delta");
  return @o;
 }
<Location /slurp>
  SetHandler modperl
  PerlHandler Example::Resource->slurp
</Location>

$> GET http://localhost:8529/slurp
$> GET http://localhost:8529/slurp?f=/etc/motd
```

Resource Profiling

```
package Example;
sub slurp {
  my ($class, $r) = @_;
  my $f = $r->args('f') || "/tmp/motd.txt";
  open (my $fh, $f);
  while(<$fh>) {
    print $_;
  }
  return OK;
}
```

```
$> GET http://localhost:8529/slurp
$> GET http://localhost:8529/slurp?f=/etc/motd
```

```
[warn] Request for GET /slurp increased memory usage by 1024
[warn] Request for GET /slurp?f=/large increased memory usage by 102400
```

Timing Profiling

```
package Example;
sub timer_in {
  my ($class, $r) = @_;
  $r->pnotes($class => [gettimeofday]);
  return OK;
}
sub timer_out {
  my ($class, $r) = @_;
  my $e = tv_interval($r->pnotes($class), [gettimeofday]);
  my $req = $r->the_request;
  $r->warn("Request for $req took $e seconds");
  return OK;
}
```

```
<Location /slurp>
  SetHandler modperl
  PerlInitHandler Example->timer_in
  PerlHandler Example->slurp
  PerlCleanupHandler Example->timer_out
</Location>
```

Timing Profiling

```
<Location /slurp>
  SetHandler modperl
  PerlInitHandler Example->timer_in
  PerlHandler Example->slurp
  PerlCleanupHandler Example->timer_out
</Location>
```

```
$> GET http://localhost:8529/slurp
$> GET http://localhost:8529/slurp?f=/etc/motd
```

```
[warn] Request for GET /slurp          took 0.050202 secs
[warn] Request for GET /slurp?f=/large took 3.403233 secs
```

And now...

- + Using inheritance works
- + It doesn't touch the instrumented code
- + It makes it possible for both versions to co-exist
- - It requires per-handler code
- - It requires knowledge of the instrumented code

...for something completely different

- Measure something before
- Measure something after
- compare
- sounds generic...

Apache2::Instrument

- [META: Grab samples from the code]

Apache2::Instrument

- Allows for easy plug n' play instruments
- Can measure very application specific quantities
- Can also measure non mod_perl requests
- Is on CPAN (or will be shortly)

Devel::Profiler

- A pure-Perl code Profiler
- Compatible with Devel::DProf/dprofpp
- Comes with an Devel::Profiler::Apache

Devel::Profiler

```
$> perl -MDevel::Profiler somescript.pl
$> dprofpp

Total Elapsed Time =    0.041 Seconds
  User+System Time =    0.049 Seconds
Exclusive Times
%Time ExclSec CumulS #Calls sec/call Csec/c  Name
 20.4   0.010  0.019      1   0.0100 0.0190  MIME::Parser::Reader::read_line
 20.4   0.010  0.020      6   0.0017 0.0033  MIME::Head::mime_type
 20.4   0.010  0.010     13   0.0008 0.0008  MIME::Field::ParamVal::parse
 20.4   0.010  0.010      3   0.0033 0.0033  MIME::Entity::bodyhandle
 18.3   0.009  0.009     45   0.0002 0.0002  IO::ScalarArray::print
 0.00   0.000  0.000      1   0.0000 0.0000  MIME::Parser::new
 0.00   0.000  0.000      1   0.0000 0.0000  MIME::Parser::init
 0.00   0.000  0.000      4   0.0000 0.0000  MIME::Parser::interface
 0.00   0.000  0.000      2   0.0000 0.0000  MIME::Parser::output_dir
 0.00   0.000  0.000      2   0.0000 0.0000  MIME::Parser::Filer::new
 0.00   0.000  0.000      2   0.0000 0.0000  MIME::Parser::FileInto::init
 0.00   0.000  0.000      2   0.0000 0.0000  MIME::Parser::Filer::cleanup_di
 0.00   0.000  0.000      4   0.0000 0.0000  MIME::Parser::filer
 0.00   0.000  0.000      6   0.0000 0.0000  MIME::Parser::results
 0.00   0.000  0.000      3   0.0000 0.0000  MIME::Parser::Filer::results
```

Devel::Profiler

- + Simple to use
- + Generates tons of useful information
- - mod_perl

Devel::Profiler::Apache

```
startup.pl:

use Devel::Profiler::Apache;

$> cd /var/log/httpd/profiler/
$> ls
20229/
20230/
20233/
$> dprofpp 20229/tmon.out
```

Devel::Profiler::Apache

- + Seamless integration with mod_perl
- + Leaves profiling information behind
- + Great when targeting something specific
- - Adds significant overhead
- - Better run in single-server mode

Closing Thoughts

- Don't trust your instinct, measure it
- When possible, make it part of your tests
- Remember to optimize where it will matter

Thank you!

More info

- *mod_perl User's mailing-list*
  - http://perl.apache.org/maillist/modperl.html
  - <modperl@perl.apache.org>
- *mod_perl Developer's Cookbook*
  - http://www.modperlcookbook.org/
- Practical mod_perl
  - http://www.modperlbook.org/
- mod_perl at the ASF
  - http://perl.apache.org/

Thank You!

Slides and bonus material:

http://gozer.ectoplasm.org/talk/