mod_perl 2.0: Profiling & Instrumenting

Philippe M. Chiasson
gozer@ectoplasm.org
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 watches code
First, assumptions

- You have Apache/mod_perl running
- You are not afraid by Perl
- 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 management 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
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

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,   OK shared,  152M buff
Swap: 1984M av,  408M used, 1576M free,  248K pagein,  211K pageout

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

Total:  1569013K (1496M) size, 1481949K (1413M) approx real size (-shared)
Extensive Status for PID 9855 (httpd)

httpd-specific Info:

- **Process type**: Child
- **Status**: Waiting for Connection
- **Last request processed in**: 3.171s
- **Requests Served**:
  - This slot: 2
  - This child: 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: 0
  - utime: 0
  - stime: 0
  - cutime: 0
  - cstime: 0

General process info:

- **UID**: apache
- **GID**: apache
- **State**: None
- **TTY**: None
- **Command line arguments**: /usr/sbin/httpd
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 things
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
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
 $nvcsaw,          # volountary context switches
 $nivcsaw          # involuntary context switches
 ) = getrusage();
Resource Profiling
(using inheritance)

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->the_request; 
    $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
$> GET http://localhost:8529/slurp?f=/etc/motd
package Example;
sub slurp {
    my ($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 bytes
[warn] Request for GET /slurp?f=/large increased memory usage by 102400 bytes
Resource Profiling
(using inheritance)

- + 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
Timing Profiling
(using wrapping)

- measure before
- interesting stuff happens
- measure after
- compare
$r->(p)notes

• Found in Apache2::RequestUtil
• Persists for the duration of the request
• my $foo = $r->notes('foo')
• $r->notes('foo' => 123);
• $r->pnotes('foo' => $object);
use Apache2::RequestUtil ();

sub handler {  
    my ($class, $r) = @_;  
    $r->notes("UserID" => 123);  
}

sub loghandler {  
    my ($class, $r) = @_;  
    my $uid = $r->notes("UserID");  
}
Timing Profiling
(using wrapping)

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
(using wrapping)

<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=/large

[warn] Request for GET /slurp took 0.050202 secs
[warn] Request for GET /slurp?f=/large took 3.403233 secs
Timing Profiling (using wrapping)

- + Using wrapping works
- + It doesn’t touch the instrumented code
- + It makes it possible for both versions to co-exist
- + It doesn’t require per-handler code
- + It doesn’t require knowledge of the instrumented code
Timing Profiling
(using wrapping)

• + And it can instrument *anything* running inside Apache:
  • PHP
  • Java
  • HTTPD itself
Timing Profiling
(using wrapping)

```xml
<Location /php>
    SetHandler php-script
    PerlInitHandler Example->timer_in
    PerlCleanupHandler Example->timer_out
</Location>
```
Timing Profiling
(using wrapping)

<Location /php>
SetHandler php-script
PerlInitHandler Example->timer_in
PerlCleanupHandler Example->timer_out
</Location>

$> GET http://localhost:8529/php/info.php

[warn] Request for GET /php/info.php took 0.084954 secs
Timing Profiling
(using wrapping)

• + And it’s stackable
<Location /php>
    SetHandler php-script
    PerlInitHandler Example-&gt;timer_in
    PerlCleanupHandler Example-&gt;timer_out
</Location>
Stacked Profiling
(using wrapping)

<Location /php>
  SetHandler php-script
  PerlInitHandler Example-&gt;timer_in Example-&gt;memory_in
  PerlCleanupHandler Example-&gt;timer_out Example-&gt;memory_out
</Location>
Stacked Profiling
(using wrapping)

```xml
<Location /php>
    SetHandler php-script
    PerlInitHandler Example-&gt;timer_in Example-&gt;memory_in
    PerlCleanupHandler Example-&gt;timer_out Example-&gt;memory_out
</Location>

$> GET http://localhost:8529/php/info.php

[warn] Request for GET /php/info.php took 0.084954 secs
[warn] Request for GET /php/info.php memory += 1.6M
```
DBI Profiling
$r->push_handlers()

use Apache2::RequestUtil ();

sub handler {
    my ($class, $r) = @_;
    $r->push_handler('CleanupHandler' => "$\{class\}->report");
}

sub report {
    my ($class, $r) = @_;
package DBIProfiler;

use DBI::Profile;

sub connect {
    my $class = shift;
    unshift @_, $class if ref $class;
    my $drh = shift;

    my @args = map { defined $_ ? $_ : "" } @_;  
    my $h = $drh->connect(@args);
    my $r = Apache2::RequestUtil->request;
    my $notes = $r->pnotes($class) || {};  

    $notes->{profile} ||= DBI::Profile->new();
    $h->{Profile} = $notes->{profile};
    $notes->{h} = $h;
    $r->pnotes($class, $notes);

    return $h;
}

package DBIProfiler;

use DBI::Profile;

sub before {
    my ($class, $r) = @_;  
    my $notes = $r->pnotes($class) || {};  
    $r->push_handler('CleanupHandler' => "${class}->after")  
    $notes->{connect} = $DBI::connect_via;  
    $DBI::connect_via = "${class}::connect";  
    $r->pnotes($class, $notes);  
    return OK;
}

package DBIProfiler;

use DBI::Profile;

sub after {
    my ($class, $r) = @_;  
    my $notes = $r->pnotes;
    $DBI::connect_via = $notes->{connect};
    $notes->{h}{Profile} = undef;
    $class->report;
    return OK;
}

DBI Profiling

package DBIProfiler;

use DBI::Profile;

sub report {
  my ($class, $r) = @_;
  my $notes = $r->pnotes || {};

  # Disable default print STDERR behaviour
  local $DBI::Profile::ON_DESTROY_DUMP = sub { };

  # Grab a pretty output
  my $format = $notes->{profile}->format;

  # Kill the profiling object
  $notes->{profile} = undef;

  warn Dump($format); use YAML;
}

DBI Profiling

<Location /perl>
    PerlInitHandler DBIProfiler
</Location>

>$> GET http://localhost:8529/perl/db.pl

DBI::Profile: 0.578311s 101.51% (8 calls) httpd @ 2006-10-07 16:06:04

'' =>
    0.000062s / 5 = 0.000012s avg (first 0.000010s, min 0.000006s, max
    0.000027s)

'CREATE table foo (id int, val text)' =>
    0.014871s

'DROP table foo' =>
    0.554491s

'INSERT INTO foo (id, val) VALUES (1,'test')' =>
    0.008887s
External Profiling

- Unix systems have profiling tools
- They can inspect a running process
- But they have no real idea what apache is doing
- What if we could launch it on ourselves?
strace profiling

- strace(1) is a Linux-ism
- Solaris: dtrace
- OSX: ktrace
- Others: truss
strace profiling

```bash
$> strace -c tar zxvf file.tar.gz
```

<table>
<thead>
<tr>
<th>% time</th>
<th>seconds</th>
<th>usecs/call</th>
<th>calls</th>
<th>errors</th>
<th>syscall</th>
</tr>
</thead>
<tbody>
<tr>
<td>95.18</td>
<td>0.017156</td>
<td>1</td>
<td>12148</td>
<td></td>
<td>read</td>
</tr>
<tr>
<td>4.44</td>
<td>0.000800</td>
<td>0</td>
<td>4358</td>
<td></td>
<td>write</td>
</tr>
<tr>
<td>0.38</td>
<td>0.000068</td>
<td>68</td>
<td>1</td>
<td></td>
<td>waitpid</td>
</tr>
<tr>
<td>0.00</td>
<td>0.000000</td>
<td>0</td>
<td>28</td>
<td></td>
<td>14 open</td>
</tr>
<tr>
<td>0.00</td>
<td>0.000000</td>
<td>0</td>
<td>18</td>
<td></td>
<td>close</td>
</tr>
<tr>
<td>0.00</td>
<td>0.000000</td>
<td>0</td>
<td>1</td>
<td></td>
<td>execve</td>
</tr>
<tr>
<td>0.00</td>
<td>0.000000</td>
<td>0</td>
<td>2</td>
<td></td>
<td>1 access</td>
</tr>
<tr>
<td>0.00</td>
<td>0.000000</td>
<td>0</td>
<td>1</td>
<td></td>
<td>pipe</td>
</tr>
</tbody>
</table>

```

<table>
<thead>
<tr>
<th>% time</th>
<th>seconds</th>
<th>usecs/call</th>
<th>calls</th>
<th>errors</th>
<th>syscall</th>
</tr>
</thead>
<tbody>
<tr>
<td>100.00</td>
<td>0.018024</td>
<td>16643</td>
<td>23</td>
<td>total</td>
<td></td>
</tr>
</tbody>
</table>
strace profiling

package ExampleStrace;

use constant STRACE => "/usr/bin/strace";

sub before {
    my ($class, $r) = @_; 
    my $notes = $r->pnotes($class);

    my $pid = $$;

    my $cmd = STRACE;
    my @args = ("-c", "-p", $pid);

    my $strace = open(my $out_fh, "$cmd @args 2>&1 | ");

    $notes->{out} = $out_fh;
    $notes->{pid} = $strace;

    return OK;
}
package ExampleStrace;

sub after {
    my ($class, $r) = @_;  
    my $notes = $r->pnotes($class);

    kill INT => $notes->{pid};

    return OK;
}

strace profiling
package ExampleStrace;

sub report {
    my ($class, $r) = @_;  
    my $notes = $r->pnotes($class);

    my $out = $notes->{out};
    my @info;
    my %syscalls;
    while (<$out>) {
        parse($_, %syscalls);
    }

    my @info = map { {$_ => $syscalls{$_}}} sort {
            $syscalls{$b}{time} <=> $syscalls{$a}{time} ||
            $syscalls{$b}{seconds} <=> $syscalls{$a}{seconds} ||
            $syscalls{$b}{calls} <=> $syscalls{$a}{calls} }
            keys %syscalls;

    warn Dump(@info); use YAML;
}

strace profiling

<Location /perl>
  PerlInitHandler ExampleStrace
</Location>

$> GET http://localhost:8529/perl/heavy.pl

[warn] Top system calls from GET /perl/heavy.pl
- ioctl:
  calls: 90
  errors: 90
  seconds: 0.000052
  time: 66.67%
  usecs_call: 1
- fstat64:
  calls: 89
  seconds: 0.000026
  time: 33.33%
  usecs_call: 0
strace profiling

• Could easily be ported to
  • dtrace
  • ktrace
  • truss
  • Linux::ptrace
And now...
...for something completely different

- Measure something before
- Measure something after
- compare before & after
- report
- sounds generic...
Apache2::Instrument

- Allows for easy plug n’ play instruments
- Contains all the example shown so far
- Can measure very application specific quantities
- Can also measure non-mod_perl requests
- Is on CPAN (or will be shortly)
Apache2::Instrument

<Location /perl>
    PerlInitHandler Apache2::Instrument::Time
</Location>

<Directory /var/www/html>
    PerlInitHandler Apache2::Instrument::Strace
</Directory>

<FilesMatch ".php$">
    PerlInitHandler Apache2::Instrument::Memory
</FilesMatch>
Apache2::Instrument

```<Directory /var/www/html/css>
    PerlInitHandler +Apache2::Instrument::Time
</Directory>```
Perl*Handler +

- A nice generic mod_perl feature
- You can use Perl*Handler +something to add handlers to the handler stack
Apache2::Instrument

<Directory /var/www/html>
  PerlInitHandler Apache2::Instrument::Strace
</Directory>

<Directory /var/www/html/css>
  PerlInitHandler +Apache2::Instrument::Time
</Directory>
Apache2::Instrument

- Currently, it’s just:
  - memory
  - time
  - DBI
  - strace
Apache2::Instrument

- But it can grow and get better
- and better
  - more instruments
  - more features
Apache2::Instrument

• `lsf()`
  • list open file descriptors
• interesting to know what is:
  • opened
  • closed
  • left opened
Apache2::Instrument

• What else could be measured/compared?
• patches welcome, new modules encouraged!
Apache2::Instrument

- monitoring perl’s @INC array
- what modules are loaded at runtime?
- what requests alter @INC?
  - You can watch @INC
  - You can push a subref on @INC
Apache2::Instrument

- monitor Perl’s %ENV, and the real underlying environment
- detect what request changed what
- shouldn’t be happening often
Apache2::Instrument

- monitor httpd’s global pools
- detect and measure which ones are leaking
  - apr_pool_num_bytes() only enabled with:
    - #define APR_POOL_DEBUG
Apache2::Instrument

- Dump reports to disk
- Dump reports to some database
Apache2::Instrument

• Improvements to itself:
  • Store reports outside of the error_log
    • flat files
    • database
  • Framework/api for:
    • command-line
      • reporting
      • querying
Profilers

- Devel::DProf
- Devel::Profiler
Single-Process

$> httpd -DNO_DETACH -DONE_PROCESS [other args]
Apache::DProf

PerlModule Apache::DProf

$> apachectl restart
$> cd /var/httpd/logs/prof/12345
$> dprofpp
Total Elapsed Time = 6.238159 Seconds
  User+System Time = 1.188159 Seconds

Exclusive Times

<table>
<thead>
<tr>
<th>%Time</th>
<th>ExclSec</th>
<th>CumulS</th>
<th>#Calls</th>
<th>sec/call</th>
<th>Csec/c</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.8</td>
<td>0.129</td>
<td>0.463</td>
<td>19</td>
<td>0.0068</td>
<td>0.0244</td>
<td>Mail::SpamAssassin::BEGIN</td>
</tr>
<tr>
<td>9.17</td>
<td>0.109</td>
<td>0.207</td>
<td>71</td>
<td>0.0015</td>
<td>0.0029</td>
<td>Mail::SpamAssassin::PerMsgStatus::BEGIN</td>
</tr>
<tr>
<td>6.73</td>
<td>0.080</td>
<td>0.138</td>
<td>4</td>
<td>0.0200</td>
<td>0.0345</td>
<td>Mail::Transport::IMAP4::BEGIN</td>
</tr>
<tr>
<td>6.65</td>
<td>0.079</td>
<td>1.121</td>
<td>24</td>
<td>0.0033</td>
<td>0.0467</td>
<td>main::BEGIN</td>
</tr>
<tr>
<td>3.96</td>
<td>0.047</td>
<td>0.079</td>
<td>3</td>
<td>0.0158</td>
<td>0.0265</td>
<td>Mail::IMAPClient::_read_line</td>
</tr>
<tr>
<td>3.37</td>
<td>0.040</td>
<td>0.049</td>
<td>8</td>
<td>0.0050</td>
<td>0.0062</td>
<td>Mail::SpamAssassin::Conf::BEGIN</td>
</tr>
<tr>
<td>3.37</td>
<td>0.040</td>
<td>0.059</td>
<td>17</td>
<td>0.0023</td>
<td>0.0035</td>
<td>Net::DNS::Resolver::Base::BEGIN</td>
</tr>
<tr>
<td>3.37</td>
<td>0.040</td>
<td>0.186</td>
<td>16</td>
<td>0.0025</td>
<td>0.0117</td>
<td>Mail::Box::IMAP4::BEGIN</td>
</tr>
<tr>
<td>3.28</td>
<td>0.039</td>
<td>0.293</td>
<td>32</td>
<td>0.0012</td>
<td>0.0092</td>
<td>base::import</td>
</tr>
<tr>
<td>2.86</td>
<td>0.034</td>
<td>0.034</td>
<td>1425</td>
<td>0.0000</td>
<td>0.0000</td>
<td>MIME::Type::simplified</td>
</tr>
<tr>
<td>2.52</td>
<td>0.030</td>
<td>0.089</td>
<td>3</td>
<td>0.0100</td>
<td>0.0297</td>
<td>Net::DNS::Resolver::UNIX::BEGIN</td>
</tr>
</tbody>
</table>
Devel::Profiler

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

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

Total Elapsed Time = 0.041 Seconds
User+System Time = 0.049 Seconds

**Exclusive Times**

<table>
<thead>
<tr>
<th>%Time</th>
<th>ExclSec</th>
<th>CumulS</th>
<th>#Calls</th>
<th>sec/call</th>
<th>Csec/c</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>20.4</td>
<td>0.010</td>
<td>0.019</td>
<td>1</td>
<td>0.0100</td>
<td>0.0190</td>
<td>MIME::Parser::Reader::read_lines</td>
</tr>
<tr>
<td>20.4</td>
<td>0.010</td>
<td>0.020</td>
<td>6</td>
<td>0.0017</td>
<td>0.0033</td>
<td>MIME::Head::mime_type</td>
</tr>
<tr>
<td>20.4</td>
<td>0.010</td>
<td>0.010</td>
<td>13</td>
<td>0.0008</td>
<td>0.0008</td>
<td>MIME::Field::ParamVal::parse</td>
</tr>
<tr>
<td>20.4</td>
<td>0.010</td>
<td>0.010</td>
<td>3</td>
<td>0.0033</td>
<td>0.0033</td>
<td>MIME::Entity::bodyhandle</td>
</tr>
<tr>
<td>18.3</td>
<td>0.009</td>
<td>0.009</td>
<td>45</td>
<td>0.0002</td>
<td>0.0002</td>
<td>IO::ScalarArray::print</td>
</tr>
<tr>
<td>0.00</td>
<td>0.000</td>
<td>0.000</td>
<td>1</td>
<td>0.0000</td>
<td>0.0000</td>
<td>MIME::Parser::new</td>
</tr>
<tr>
<td>0.00</td>
<td>0.000</td>
<td>0.000</td>
<td>1</td>
<td>0.0000</td>
<td>0.0000</td>
<td>MIME::Parser::init</td>
</tr>
<tr>
<td>0.00</td>
<td>0.000</td>
<td>0.000</td>
<td>4</td>
<td>0.0000</td>
<td>0.0000</td>
<td>MIME::Parser::interface</td>
</tr>
<tr>
<td>0.00</td>
<td>0.000</td>
<td>0.000</td>
<td>2</td>
<td>0.0000</td>
<td>0.0000</td>
<td>MIME::Parser::output_dir</td>
</tr>
<tr>
<td>0.00</td>
<td>0.000</td>
<td>0.000</td>
<td>2</td>
<td>0.0000</td>
<td>0.0000</td>
<td>MIME::Parser::FileInto::init</td>
</tr>
<tr>
<td>0.00</td>
<td>0.000</td>
<td>0.000</td>
<td>2</td>
<td>0.0000</td>
<td>0.0000</td>
<td>MIME::Parser::Filer::new</td>
</tr>
<tr>
<td>0.00</td>
<td>0.000</td>
<td>0.000</td>
<td>2</td>
<td>0.0000</td>
<td>0.0000</td>
<td>MIME::Parser::FileInto::init</td>
</tr>
<tr>
<td>0.00</td>
<td>0.000</td>
<td>0.000</td>
<td>2</td>
<td>0.0000</td>
<td>0.0000</td>
<td>MIME::Parser::Filer::cleanup_dir</td>
</tr>
<tr>
<td>0.00</td>
<td>0.000</td>
<td>0.000</td>
<td>4</td>
<td>0.0000</td>
<td>0.0000</td>
<td>MIME::Parser::filer</td>
</tr>
<tr>
<td>0.00</td>
<td>0.000</td>
<td>0.000</td>
<td>6</td>
<td>0.0000</td>
<td>0.0000</td>
<td>MIME::Parser::results</td>
</tr>
<tr>
<td>0.00</td>
<td>0.000</td>
<td>0.000</td>
<td>3</td>
<td>0.0000</td>
<td>0.0000</td>
<td>MIME::Parser::Filer::results</td>
</tr>
</tbody>
</table>

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
Re-compiling httpd

- Didn’t he mention somewhere about compiling httpd for performance/instrumenting?
- Don’t do it
  - Unless you have exhausted all other options
  - And I really mean all other options
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*
  - <modperl@perl.apache.org>

• *Apache2::Instrument on CPAN*
  - http://search.cpan.org/dist/Apache2-Instrument

• *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/