

# mod\_perl 2.0: Profiling & Instrumenting

Philippe M. Chiasson  
[gozer@ectoplasm.org](mailto: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 its 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

## 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

	<u>PID</u>	<u>Size</u>	<u>Share</u>	<u>VSize</u>	<u>Rss</u>	<u>M</u>	<u>Elapsed</u>	<u>LastReq</u>	<u>Srvd</u>	<u>Client</u>	<u>Virtual Host</u>
0:	<a href="#">3049</a>	87.5M	9.1M	87.5M	32.4M						
14:	<a href="#">9855</a>	89.3M	4.9M	89.3M	29.9M	_	0.000s	0.769s	1	::1	coupler.activestate.com GET / HTTP/1.0
12:	<a href="#">9853</a>	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:	<a href="#">9857</a>	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:	<a href="#">9843</a>	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:	<a href="#">9847</a>	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:	<a href="#">9846</a>	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:	<a href="#">9839</a>	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:	<a href="#">9854</a>	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:	<a href="#">9832</a>	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:	<a href="#">9848</a>	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:	<a href="#">9842</a>	97.0M	5.8M	97.0M	36.5M	W	0.854s	0.000s	2	192.168.69.209	coupler.activestate.com GET /vmonitor
2:	<a href="#">9829</a>	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:	<a href="#">9831</a>	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:	<a href="#">9830</a>	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:	<a href="#">9828</a>	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 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
```

# 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,        # voluntary context switches
 $nivcsw        # 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
```

# Resource Profiling

## (using inheritance)

```
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);`

# \$r->(p)notes

```
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)

```
<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

# Stacked Profiling

## (using wrapping)

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

# Stacked Profiling

## (using wrapping)

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

# Stacked Profiling

## (using wrapping)

```
<Location /php>  
  SetHandler php-script  
  PerlInitHandler Example->timer_in Example->memory_in  
  PerlCleanupHandler Example->timer_out Example->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) = @_;
}
```

# DBI Profiling

```
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;
}
```

# DBI Profiling

```
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;
}
```



# DBI Profiling

```
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

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

% time	seconds	usecs/call	calls	errors	syscall
95.18	0.017156	1	12148		read
4.44	0.000800	0	4358		write
0.38	0.000068	68	1		waitpid
0.00	0.000000	0	28	14	open
0.00	0.000000	0	18		close
0.00	0.000000	0	1		execve
0.00	0.000000	0	2	1	access
0.00	0.000000	0	1		pipe
[...]					
100.00	0.018024		16643	23	total

# strace profiling

```
package ExampleStrace;

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

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

    my $pid = $$;

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

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

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

    return OK;
}
```

# strace profiling

```
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

- lsof()
  - 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

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

# Devel::Profiler

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

# 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*
  - <http://perl.apache.org/maillist/modperl.html>
  - [<modperl@perl.apache.org>](mailto: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/>