PHP-FPM tuning: Using ‘pm static’ for Max Performance

An unedited version of article was originally published at HaydenJames.io and republished here with the author’s permission.

Let’s take a very quick look at how best to set up PHP-FPM for high throughput, low latency, and a more stable use of CPU and memory. By default, most setups have PHP-FPM’s PM (process manager) string set to dynamic and there’s also the common advice to use `ondemand` if you suffer from available memory issues. However, let’s compare the two management options based on php.net’s documentation and also compare my favorite for high traffic setup — static pm:

pm = dynamic: the number of child processes is set dynamically based on the following directives: `pm.max_children`, `pm.start_servers`, `pm.min_spare_servers`, `pm.max_spare_servers`.

pm = ondemand: the processes spawn on demand when requested, as opposed to dynamic, where `pm.start_servers` are started when the service is started.

pm = static: the number of child processes is fixed by `pm.max_children`.

See the full list of global `php-fpm.conf` directives for further details.

PHP-FPM Process Manager (PM) Similarities to CPUFreq Governor

Now, this may seem a bit off topic, but I hope to tie it back into our PHP-FPM tuning topic. Okay, we’ve all had slow CPU issues at some point, whether it be a laptop, VM or dedicated server. Remember CPU frequency scaling? (CPUFreq governor.) These settings, available on both *nix and Windows, can improve the performance and system responsiveness by

changing the CPU governor setting from *ondemand* to *performance*. This time, let's compare the descriptions and look for similarities:

Governor = ondemand: scales CPU frequency dynamically according to current load. Jumps to the highest frequency and then scales down as the idle time increases.

Governor = conservative: scales the frequency dynamically according to current load. Scales the frequency more gradually than ondemand.

Governor = performance: always run the CPU at the maximum frequency.

See the full list of CPUFreq governor options for further details.

Notice the similarities? I wanted to use this comparison first, with the aim of finding the best way to write an article which recommends using pm static for PHP-FPM as your first choice.

With CPU governor, the performance setting is a pretty safe performance boost because it's almost entirely dependent on your server CPU's limit. The only other factors would be things such as heat, battery life (laptop) and other side effects of clocking your CPU frequency to 100% permanently. Once set to performance, it is indeed the fastest setting for your CPU. For example read about the 'force_turbo' setting on Raspberry Pi, which forces your RPi board to use the performance governor where performance improvement is more noticeable due to the low CPU clock speeds.

Using 'pm static' to Achieve Your Server's Max Performance

The PHP-FPM pm static setting depends heavily on how much free memory your server has. Basically, if you are suffering from low server memory, then pm ondemand or dynamic may be better options. On the other hand, if you have the memory available, you can avoid much of the PHP process manager (PM) overhead by setting pm static to the max

capacity of your server. In other words, when you do the math, `pm.static` should be set to the max amount of PHP-FPM processes that can run without creating memory availability or cache pressure issues. Also, not so high as to overwhelm CPU(s) and have a pile of pending PHP-FPM operations.

```
top - 10:14:15 up 18 days, 14:25,  1 user,  load average: 1.74, 1.05, 0.87
Tasks: 244 total,   2 running, 242 sleeping,   0 stopped,   0 zombie
%Cpu(s): 30.7 us,  3.7 sy,  0.0 ni, 65.4 id,  0.0 wa,  0.0 hi,  0.0 si,  0.1 st
GiB Mem :   31.263 total,    2.582 free,    5.799 used,   22.881 buff/cache
GiB Swap:   15.750 total,   15.740 free,    0.010 used.   24.400 avail Mem

  PID     USER  PR  NI    VIRT    RES    SHR    SWAP S  %CPU  %MEM     TIME+ nTH  COMMAND
26604 [redacted]  20   0 1257.9m 675.5m  36.9m           R  12.3   2.1  28:18.37   1  php-fpm
 9300 [redacted]  20   0 1336.2m 675.0m  34.1m           S  11.3   2.1  38:39.95   1  php-fpm
17019 [redacted]  20   0  748.5m  88.6m  31.7m           S   8.0   0.3  29:42.35   1  php-fpm
 7812 [redacted]  20   0  668.4m  91.2m  38.3m           S   7.6   0.3  32:37.19   1  php-fpm
 9303 [redacted]  20   0  672.6m  95.2m  39.2m           S   6.6   0.3  42:49.42   1  php-fpm
 9288 [redacted]  20   0  672.7m  90.2m  37.3m           S   5.3   0.3  36:56.67   1  php-fpm
 9310 [redacted]  20   0  675.0m  90.9m  35.7m           S   5.0   0.3  34:58.14   1  php-fpm
 9315 [redacted]  20   0  667.0m  91.7m  41.5m           S   3.0   0.3  43:38.56   1  php-fpm
 9268 [redacted]  20   0  746.8m  95.8m  40.8m           S   2.3   0.3  41:09.79   1  php-fpm
 9275 [redacted]  20   0  672.8m  93.1m  39.8m           S   2.3   0.3  33:55.29   1  php-fpm
 9298 [redacted]  20   0  673.0m  88.6m  32.0m           S   2.0   0.3  40:22.87   1  php-fpm
21761 [redacted]  20   0  670.9m  92.1m  36.8m           S   2.0   0.3  37:20.14   1  php-fpm
 6264 [redacted]  20   0  668.6m  89.4m  36.4m           S   1.3   0.3  33:36.36   1  php-fpm
 9285 [redacted]  20   0  668.6m  87.9m  36.4m           S   1.3   0.3  42:20.01   1  php-fpm
 9286 [redacted]  20   0  668.6m  91.6m  38.5m           S   1.3   0.3  37:31.17   1  php-fpm
 9293 [redacted]  20   0  670.7m  86.9m  37.1m           S   1.3   0.3  38:49.71   1  php-fpm
16346 [redacted]  20   0  635.4m  49.0m  30.1m           S   1.3   0.2   6:38.38   1  php-fpm
19345 [redacted]  20   0  668.4m  86.3m  33.5m           S   1.3   0.3  18:08.67   1  php-fpm
 8645 [redacted]  20   0  668.4m  85.0m  32.2m           S   1.0   0.3  19:50.85   1  php-fpm
 9274 [redacted]  20   0  670.9m  93.3m  38.0m           S   1.0   0.3  37:05.21   1  php-fpm
 9278 [redacted]  20   0  746.7m  94.4m  39.4m           S   1.0   0.3  37:35.84   1  php-fpm
 9301 [redacted]  20   0  669.0m  93.3m  41.2m           S   1.0   0.3  34:55.62   1  php-fpm
12344 [redacted]  20   0  670.5m  88.5m  33.4m           S   1.0   0.3  39:25.87   1  php-fpm
14383 [redacted]  20   0  658.8m  77.2m  34.0m           S   1.0   0.2  23:21.34   1  php-fpm
18439 [redacted]  20   0  669.5m  89.7m  35.7m           S   1.0   0.3  21:09.87   1  php-fpm
 9271 [redacted]  20   0  668.7m  91.6m  38.6m           S   0.3   0.3  41:13.72   1  php-fpm
 9280 [redacted]  20   0  668.6m  89.3m  36.3m           S   0.3   0.3  36:20.29   1  php-fpm
 9296 [redacted]  20   0  675.3m  97.5m  42.2m           S   0.3   0.3  36:50.66   1  php-fpm
 9317 [redacted]  20   0  670.6m  85.9m  34.8m           S   0.3   0.3  42:13.75   1  php-fpm
 5481 [redacted]  20   0  666.4m  81.7m  30.8m           S         0.3  23:15.93   1  php-fpm
 5954 [redacted]  20   0  672.5m  88.8m  32.8m           S         0.3  39:42.86   1  php-fpm
 6992 [redacted]  20   0  680.6m  90.5m  37.5m           S         0.3  35:57.92   1  php-fpm
 7214 [redacted]  20   0  664.8m  85.9m  36.7m           S         0.3  33:27.48   1  php-fpm
 9270 [redacted]  20   0  670.6m  89.9m  38.9m           S         0.3  33:32.21   1  php-fpm
 9272 [redacted]  20   0  668.8m  88.8m  35.6m           S         0.3  39:13.35   1  php-fpm
 9283 [redacted]  20   0  679.1m  96.7m  38.2m           S         0.3  43:17.32   1  php-fpm
 9290 [redacted]  20   0  668.0m  88.9m  36.5m           S         0.3  39:39.88   1  php-fpm
 9291 [redacted]  20   0  672.8m  91.9m  36.2m           S         0.3  37:19.58   1  php-fpm
 9299 [redacted]  20   0  743.1m  89.8m  38.5m           S         0.3  40:02.81   1  php-fpm
 9305 [redacted]  20   0  668.6m  83.6m  36.5m           S         0.3  44:29.88   1  php-fpm
 9313 [redacted]  20   0  672.8m  89.6m  36.8m           S         0.3  36:18.63   1  php-fpm
12179 [redacted]  20   0  638.3m  51.9m  29.4m           S         0.2   3:15.23   1  php-fpm
12308 [redacted]  20   0  637.1m  51.2m  29.6m           S         0.2   6:17.30   1  php-fpm
18775 [redacted]  20   0  749.0m  94.3m  36.9m           S         0.3  33:04.19   1  php-fpm
19508 [redacted]  20   0  666.4m  86.5m  35.7m           S         0.3  37:37.54   1  php-fpm
20630 [redacted]  20   0  670.7m  88.6m  34.3m           S         0.3  35:37.50   1  php-fpm
21071 [redacted]  20   0  674.6m  88.2m  33.4m           S         0.3  31:07.71   1  php-fpm
23364 [redacted]  20   0  664.7m  84.8m  35.7m           S         0.3  37:39.98   1  php-fpm
28163 [redacted]  20   0  670.8m  91.0m  36.0m           S         0.3  33:56.49   1  php-fpm
```

In the screenshot above, this server has `pm = static` and `pm.max_children = 100` which uses a max of around 10GB of the 32GB installed. Take note of the self explanatory highlighted columns. During that screenshot there were about 200 'active users' (past 60 seconds) in Google Analytics. At that level, about 70% of PHP-FPM children are still idle. This means PHP-FPM is always set to the max capacity of your server's resources regardless of current traffic. Idle processes stay online, waiting for traffic spikes and responding immediately, rather than having to wait on the pm to spawn children and then kill them off after `x pm.process_idle_timeout` expires. I have `pm.max_requests` set extremely high because this is a production server with no PHP memory leaks. You can use `pm.max_requests = 0` with `static` if you have 110% confidence in your current and future PHP scripts. However, it's recommended to restart scripts over time. Set the number of requests to a high number since the point is to avoid pm overhead. So for example at least `pm.max_requests = 1000` depending on your number of `pm.max_children` and number of requests per second.

The screenshot uses Linux top filtered by 'u' (user) option and the name of the PHP-FPM user. The number of processes displayed are only the 'top' 50 or so (didn't count), but basically top displays the top stats which fit in your terminal window — in this case, sorted by %CPU. To view all 100 PHP-FPM processes you can use something like:

```
top -bn1 | grep php-fpm
```

When to Use pm ondemand and dynamic

Using `pm dynamic` you may have noticed errors similar to:

```
WARNING: [pool xxxx] seems busy (you may need to increase
pm.start_servers, or pm.min/max_spare_servers), spawning 32 children,
there are 4 idle, and 59 total children
```

You may try to increase/adjust settings and still see the same error as someone describes in this Serverfault post. In that case, the `pm.min` was too

low and because web traffic fluctuates greatly with dips and spikes, using `pm dynamic` can be difficult to tune correctly. The common advice is to use pm ondemand. However, that's even worse, because ondemand will shut down idle processes right down to 0 when there's little to no traffic and then you'll end up with just as much overhead issues as traffic fluctuates — unless, of course, you set the idle timeout extremely high … in which case you should just be using `pm.static` + a high `pm.max_requests`.

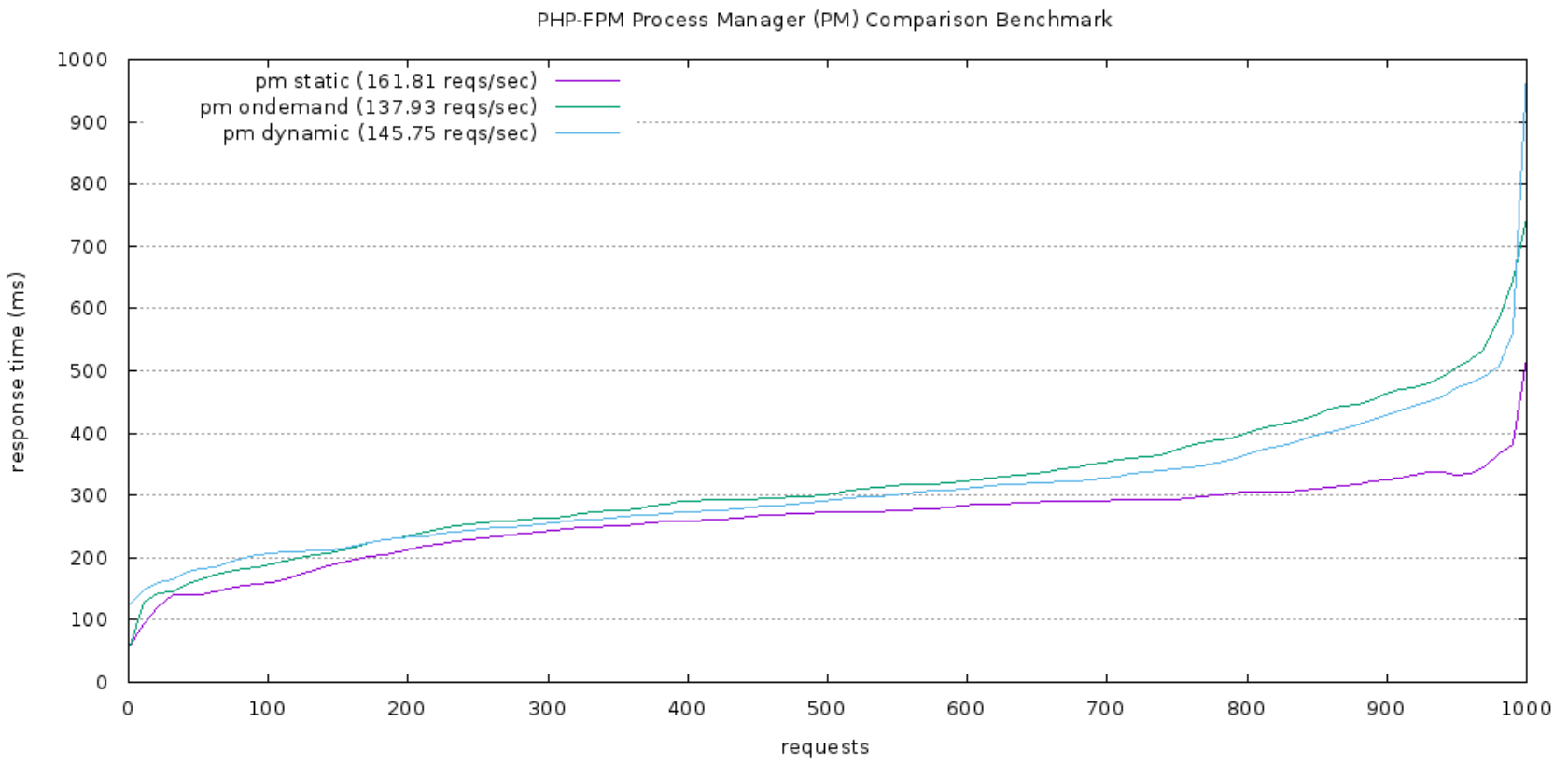
PM dynamic and especially ondemand can save you, however, when you have multiple PHP-FPM pools. For example, hosting multiple cPanel accounts or multiple websites under different pools. I have a server, for example, with 100+ cPanel accounts and about 200+ domains, and it would be impossible for pm.static or even dynamic to perform well. Only ondemand performs well, since more than two thirds of the websites receive little to no traffic. And with ondemand, it means all children will be shut down saving tons of server memory! Thankfully, cPanel devs figured this out and now it defaults to ondemand. Previously with dynamic as default it made PHP-FPM not an option on busy shared servers. Many would use suPHP because of pm dynamic eating up memory even on idle cPanel PHP-FPM pools/accounts. Chances are, if you receive good traffic, you won't be hosted on a server with lots of PHP-FPM pools (shared hosting).

Conclusion

When it comes to PHP-FPM, once you start to serve serious traffic, ondemand and dynamic process managers for PHP-FPM can limit throughput because of the inherent overhead. Know your system and set your PHP-FPM processes to match your server's max capacity. Start with `pm.max_children` set based on max usage of pm dynamic or ondemand and then increase to the point where memory and CPU can process without becoming overwhelmed. You will notice that with pm static, because you keep everything sitting in memory, traffic spikes over time cause less spikes to CPU and your server's load and CPU averages will be smoother. The average size of your PHP-FPM process will vary per web server

requiring manual tuning, thus why the more automated overhead process managers — dynamic and ondemand — are more popular recommendations. Hope this was a useful article.

Update: *Added A/B benchmark comparison graph. Having PHP-FPM processes sit in memory helps performance at the price of increased memory usage to have them sit in wait. Find your setup sweet-spot.*



Meet the author

Hayden James is a Linux Systems Analyst and Internet Entrepreneur from the Caribbean. He relocated to the US ten years ago, where he maintained Linux servers and sold small startups. Today, he supports clients remotely from his island home while also managing a niche web hosting venture. His web blog features Sysadmin and other PHP related articles: haydenjames.io.