PHP-FPM 调优：使用‘pm static’来最大化你的服务器负载能力

让我们来迅速了解一下怎样设置 PHP-FPM，以便达到高吞吐，低延迟以及稳定的使用 CPU 和内存的完美状态。在默认的情况下，大多数设置都将 PHP-FPM PM（进程管理器）设置为 `dynamic` ，或者当你有可用内存的问题时常建议你使用 `ondemand`。接下来，让我们根据 php.net 的官方文档来比较一下这两个管理选项和我最常用的设置 —— `static` 之间的区别：

- **pm = dynamic**：子进程的数量是根据以下指令来动态生成的：`pm.max_children`, `pm.start_servers`, `pm.min_spare_servers`, `pm.max_spare_servers`.
- **pm = ondemand**：在服务启动的时候根据 `pm.start_servers` 指令生成进程，而非动态生成。
- **pm = static**：子进程的数量是由 `pm.max_children` 指令来确定的。

查看完整列表，深入了解 `php-fpm.conf` 的所有指令。

PHP-FPM 进程管理器（PM）和 CPUFreq Governor 的相似之处

现在，我们要说些偏离主题，但我觉得和 PHP-FPM 调优有关的事情。好了，我们都有过在某些时候的 CPU 缓慢问题，无论是笔记本电脑、VM 或者是专用的服务器。还记得 CPU 频率缩放问题吗？（CPUFreq governor）这些设置在类 Unix 系统和 Windows 上是有效的，可以通过修改 CPU governor，将其从 `ondemand` 修改为 `performance` 来提高性能并加快系统的响应。现在，让我们来比较下列 CPUFreq governor 描述和 PHP-FPM PM 有哪些相似之处：

- **Governor = ondemand**：根据当前负荷动态调整 CPU 频率。先将 CPU 频率调整至最大，然后随着空闲时间的增加而缩小频率。
- **Governor = conservative**：根据当前负荷动态调整频率。比设置成 `ondemand` 更加缓慢。

- **Governor = performance**：始终以最大频率运行 CPU。

查看 CPUFreq governor 选项详细列表，获取更多相关信息。

注意到相似之处了吗？这就是我这个比较的首要目的，为了找到一个最好的方式来写这篇文章，推荐你将 PHP-FPM 的 `pm static` 当作你的第一选择。

使用 CPU Governor 的 `performance` 设置是一个非常安全的性能提升方式，因为它能完美的使用你服务器 CPU 的全部性能。唯一需要考虑的因素就是一些诸如散热、电池寿命（笔记本电脑）和一些由 CPU 始终保持 100% 所带来的一些副作用。一旦设置为 `performance`，那么它确实是你 CPU 最快的设置。相关实例请阅读 'force_turbo' 在 Raspberry Pi 上的设置，它教你在 RPi 板上使用 `performance` Governor，由于 CPU 时钟速度较低，性能改善将更加明显。

使用 `pm static` 优化你的服务器性能

PHP-FPM 的 `static` 设置取决于你服务器有多少闲置内存。大多数情况下，如果你服务器的内存不足，那么 PM 设置成 `ondemand` 或 `dynamic` 将是更好的选择。但是，一旦你有可用的闲置内存，那么把 PM 设置成 `static` 的最大值将减少许多 PHP 进程管理器（PM）所带来的开销。换句话说，你应该在没有内存不足和缓存压力的情况下使用 `pm.static` 来设置 PHP-FPM 进程的最大数量。此外，也不能影响到 CUP 的使用和其他待处理的 PHP-FPM 操作。

```
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

在上面的截图中，这台服务器的设置（`pm = static`，`pm.max_children =100`）最多使用了 10GB 的内存。请注意高亮的列。Google 分析图中大概有 200 个活跃用户（60秒内）。在这种用户量下，有 70% 的 PHP-FPM 子进程被闲置。这意味着，无论当前流量如何，PHP-FPM 始终保持着足够多的进程。闲置的进程始终保持在线，就算达到了流量的峰值也能快速响应，而不是等待 PM 生成子进

程，然后在 `x pm.process_idle_timeout` 秒后将此进程结束。我将 `pm.max_requests` 设置的非常高，因为这是一个不可能发生内存泄漏的 PHP 生产服务器。如果你对你的 PHP 脚本有着 110% 的信心，那么你可用选择使用 `pm.max_requests = 0`。但建议适当的重启服务。将请求数量设置的很高，是为了避免过高的 PM 开销。例如，设置 `pm.max_requests = 1000`，但这需要根据 `pm.max_children` 的设置和实际每秒的请求数量来决定。

截图使用 Linux top 通过 'u'（user）选项和 PHP-FPM 用户名进行过滤。并只显示了前 50 个左右（未统计）的进程，但基本上 `top` 命令也只会显示适合你终端窗口大小的内容 —— 在本例中，使用 `%CPU` 排序。要查看全部的 100 条 PHP-FPM 进程的话，你需要使用以下命令：

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

何时使用 ondemand 和 dynamic

使用 `pm dynamic`，您可能会出现类似于下面的错误：

```
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
```

您可能会尝试调整 pm 配置，但仍然会看到同样的错误\
在这种情况下，`pm.min` 太低，并且因为流量和峰值波动很大，使用 `pm dynamic` 可能难以调整

一般的建议是使用 `pm ondemand`。 然而，情况会变的更糟，因为 `ondemand` 会在没有流量时关闭空闲进程，然后最终会产生与流量波动很大一样的开销问题 (除非您设置空闲超时的时间非常非常的长)

但是，当您拥有多个 pm 进程池时，`pm dynamic`， 特别是 `ondemand` 是可以为您节省时间的。例如在共享的 VPS 上，有 100+ 的 cPanel 账号和 200+ 的域名，使用 `pm.static` 或者是 `pm.dynamic` 都是不可能的，即使在没有任何流量的情况下，内存会被瞬间用完，而 `pm.ondemand` 意味着所有

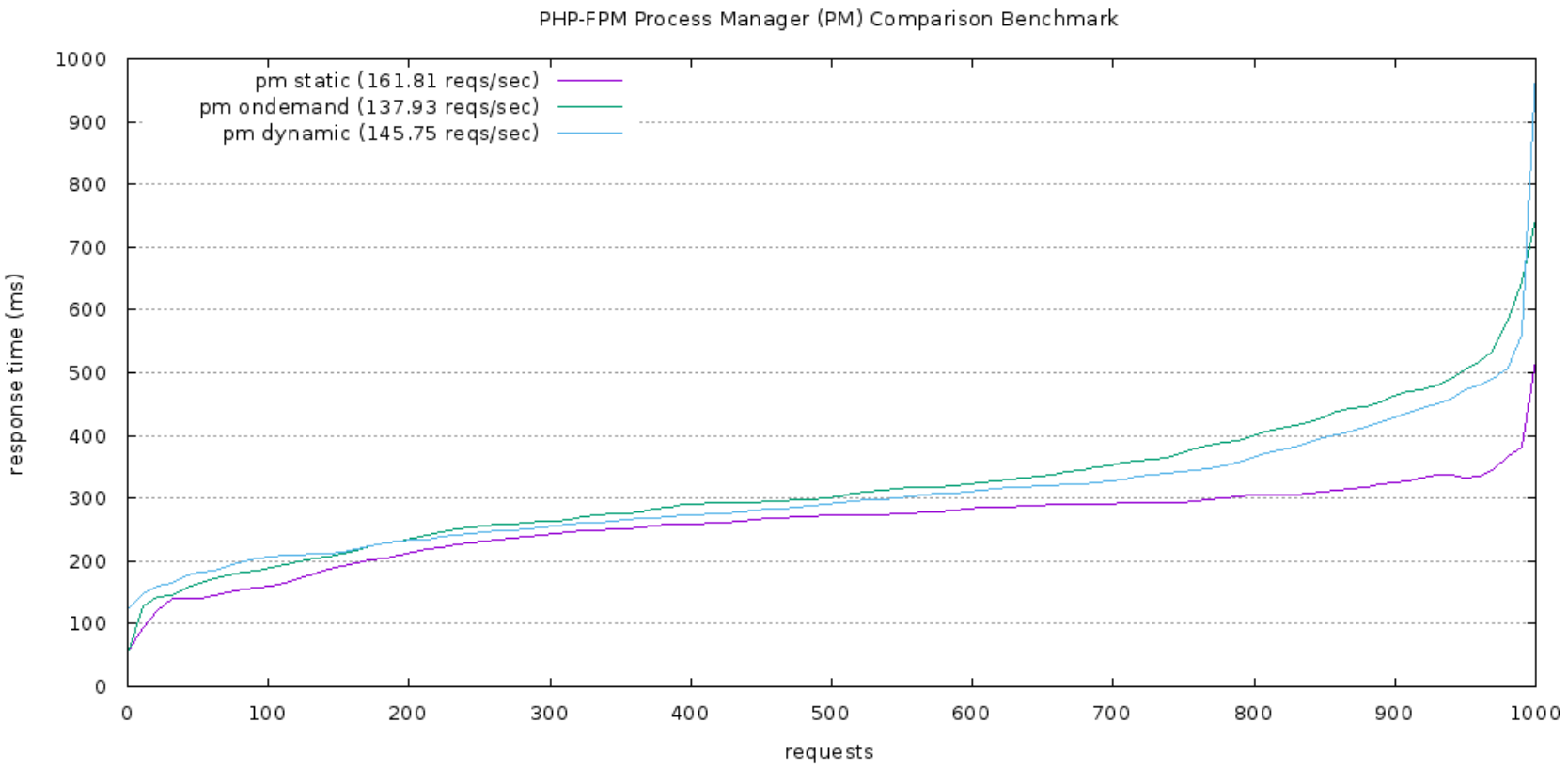
空闲的子进程都会被完全关闭，节省了大量内存。cPanel 的开发者已经意识到了这个问题，现在的 cPanel 默认就是设置为 `pm.ondemand`。

结论

当流量波动比较大的时候，， PHP-FPM 的 `ondemand` 和 `dynamic` 会因为固有开销而限制吞吐量。 您需要了解您的系统并设置 PHP-FPM 进程数，以匹配服务器的最大容量。\
从 `pm.max_children` 开始，根据 `pm dynamic` 或 `ondemand` 的最大使用情况去设置

您会注意到，在 `pm static` 模式下，因为您将所有内容都保存在内存中，所以随着时间的推移，流量峰值会对 CPU 造成比较小的峰值，并且您的服务器负载和 CPU 平均值将变得更加平滑。 每个需要手动调整的 PHP-FPM 进程数的平均大小会有所不同

更新：附上一张 A/B 测试图。



> 转自 PHP / Laravel 开发者社区 https://laravel-china.org/top...