

14 | Linux 性能优化答疑 (二)

2018-12-21 倪朋飞



讲述：冯永吉

时长 13:16 大小 12.16M



你好，我是倪朋飞。

今天是我们第二期答疑，这期答疑的主题是我们多次用到的 perf 工具，内容主要包括前面案例中，perf 使用方法的各种疑问。

perf 在性能分析中非常有效，是我们每个人都需要掌握的核心工具。perf 的使用方法也很丰富，不过不用担心，目前你只要会用 perf record 和 perf report 就够了。而对于 perf 显示的调用栈中的某些内核符号，如果你不理解也没有关系，可以暂时跳过，并不影响我们的分析。

同样的，为了便于你学习理解，它们并不是严格按照文章顺序排列的，如果你需要回顾内容原文，可以扫描每个问题右下方的二维码查看。

问题 1：使用 perf 工具时，看到的是 16 进制地址而不是函数名



北斗狼

写于 2018/11/30

老师，问个问题，我按您的文档，我使用 `perf top -g -p` 进程 ID 怎么无法查看 `php-fpm` 的调用关系，出不来哪个函数较用较高的 CPU？以下是我的显示结果，您帮我看看为什么显示的是 `0x7fd...`

1、开始压力测试

```
ab -c 10 -n 10000 http://bjdsj-test-11-58:10001
```

2、查看调用关系 `perf top -g -p 39826`

```
Samples: 80K of event 'cycles', Event count (approx.): 29700884257
```

```
+ 93.02% 0.69% php-fpm [.] 0x000000000008c296e
```

```
0x7fd84a21c96e
```

```
0x7fd84a21d323
```

```
0x7fd84a185f94
```

```
0x7fd84a0d66fc
```

```
0x7fd84a325642
```

```
0x7fd842a702e1
```

```
0x6cb6258d4c544155
```

```
+ 23.35% 0.38% php-fpm [.] 0x000000000008c4a7c
```

```
+ 5.60% 5.60% libm-2.24.so [.] 0x000000000002127e
```

```
+ 4.22% 0.00% php-fpm [.] 0x0000000000098dd97
```

```
+ 2.54% 2.54% php-fpm [.] 0x0000000000094ede0
```

```
+ 1.96% 1.96% php-fpm [.] 0x00000000000681b9d
```

```
+ 1.58% 1.58% php-fpm [.] 0x00000000000815e70
```

```
+ 1.52% 1.51% php-fpm [.] 0x000000000094eddc
+ 1.48% 1.48% php-fpm [.] 0x000000000008cd729
+ 1.32% 1.32% php-fpm [.] 0x000000000008c4ae5
+ 1.21% 1.21% php-fpm [.] 0x0000000000098dc03
+ 1.04% 1.04% php-fpm [.] 0x00000000000681b99
```

```
Failed to open /opt/bitnami/php/lib/php/extensions/opcache.so, continuing without
symbols
```

引自：Linux性能优化实战


05 | 基础篇：某个应用的CPU使用率居然达到100%，我该怎么办？

识别二维码打开原文
「极客时间」App



这也是留言比较多的一个问题，在 CentOS 系统中，使用 perf 工具看不到函数名，只能看到一些 16 进制格式的函数地址。

其实，只要你观察一下 perf 界面最下面的那一行，就会发现一个警告信息：

 复制代码

```
1 Failed to open /opt/bitnami/php/lib/php/extensions/opcache.so, continuing without symbol
```

这说明，perf 找不到待分析进程依赖的库。当然，实际上这个案例中有很多依赖库都找不到，只不过，perf 工具本身只在最后一行显示警告信息，所以你只能看到这一条警告。


这个问题，其实也是在分析 Docker 容器应用时，我们经常碰到的一个问题，因为容器应用依赖的库都在镜像里面。

针对这种情况，我总结了下面**四个解决方法**。

第一个方法，在容器外面构建相同路径的依赖库。这种方法从原理上可行，但是我并不推荐，一方面是因为找出这些依赖库比较麻烦，更重要的是，构建这些路径，会污染容器主机的环境。

第二个方法，在容器内部运行 perf。不过，这需要容器运行在特权模式下，但实际的应用程序往往只以普通容器的方式运行。所以，容器内部一般没有权限执行 perf 分析。

比方说，如果你在普通容器内部运行 `perf record`，你将会看到下面这个错误提示：


 复制代码

```
1 $ perf_4.9 record -a -g
2 perf_event_open(..., PERF_FLAG_FD_CLOEXEC) failed with unexpected error 1 (Operation not
3 perf_event_open(..., 0) failed unexpectedly with error 1 (Operation not permitted)
```

当然，其实你还可以通过配置 `/proc/sys/kernel/perf_event_paranoid`（比如改成 `-1`），来允许非特权用户执行 perf 事件分析。

不过还是那句话，为了安全起见，这种方法我也不推荐。

第三个方法，指定符号路径为容器文件系统的路径。比如对于第 05 讲的应用，你可以执行下面这个命令：

 复制代码

```
1 $ mkdir /tmp/foo
2 $ PID=$(docker inspect --format {{.State.Pid}} phpfpn)
3 $ bindfs /proc/$PID/root /tmp/foo
4 $ perf report --symfs /tmp/foo
5
6 # 使用完成后不要忘记解除绑定
7 $ umount /tmp/foo/
```

不过这里要注意，`bindfs` 这个工具需要你额外安装。`bindfs` 的基本功能是实现目录绑定（类似于 `mount --bind`），这里需要你安装的是 1.13.10 版本（这也是它的最新发布


版)。

如果你安装的是旧版本，你可以到 [GitHub](#) 上面下载源码，然后编译安装。

第四个方法，在容器外面把分析纪录保存下来，再去容器里查看结果。这样，库和符号的路径也就都对了。


比如，你可以这么做。先运行 `perf record -g -p < pid >`，执行一会儿（比如 15 秒）后，按 `Ctrl+C` 停止。

然后，把生成的 `perf.data` 文件，拷贝到容器里面来分析：

 复制代码

```
1 $ docker cp perf.data phpfpd:/tmp
2 $ docker exec -i -t phpfpd bash
```

接下来，在容器的 `bash` 中继续运行下面的命令，安装 `perf` 并使用 `perf report` 查看报告：

 复制代码

```
1 $ cd /tmp/
2 $ apt-get update && apt-get install -y linux-tools linux-perf procps
3 $ perf_4.9 report
```

不过，这里也有两点需要你注意。


首先是 `perf` 工具的版本问题。在最后一步中，我们运行的工具是容器内部安装的版本 `perf_4.9`，而不是普通的 `perf` 命令。这是因为，`perf` 命令实际上是一个软连接，会跟内核的版本进行匹配，但镜像里安装的 `perf` 版本跟虚拟机的内核版本有可能并不一致。

另外，`php-fpm` 镜像是基于 `Debian` 系统的，所以安装 `perf` 工具的命令，跟 `Ubuntu` 也并不完全一样。比如，`Ubuntu` 上的安装方法是下面这样：

 复制代码

```
1 $ apt-get install -y linux-tools-common linux-tools-generic linux-tools-$(uname -r)
```

而在 php-fpm 容器里，你应该执行下面的命令来安装 perf：

 复制代码

```
1 $ apt-get install -y linux-perf
```

当你按照前面这几种方法操作后，你就可以在容器内部看到 sqrt 的堆栈：

```
Samples: 133K of event 'cpu-clock', Event count (approx.): 33336250000
Children      Self  Command      Shared Object      Symbol
- 93.85%      0.00% php-fpm       libc-2.24.so       [.] __libc_start_main
- __libc_start_main
- 93.72% 0x9cb642
- 93.71% php_execute_script
- 93.70% zend_execute_scripts
- 93.69% zend_execute
- 90.56% execute_ex
- 19.14% 0x98dea3
- 4.57% 0x98dd97
- 4.48% add_function
- 1.04% 0x98dc03
- 0.97% 0x98dbff
- 0.80% 0x98de23
- 0.65% 0x98dbc4
- 0.62% 0x98dbf7
- 0.52% add_function
- 18.59% 0x8c4a7c
- 3.79% sqrt
- 1.14% 0x681b9d
- 1.06% 0x681b99
- 0.84% 0x681b91
- 0.55% 0x681b2c
- 0.55% 0x681b89
- 1.27% 0x8cd729
- 1.21% 0x8cd720
```

事实上，抛开我们的案例来说，即使是在非容器化的应用中，你也可能会碰到这个问题。假如你的应用程序在编译时，使用 strip 删除了 ELF 二进制文件的符号表，那么你同样也只能看到函数的地址。

现在的磁盘空间，其实已经足够大了。保留这些符号，虽然会导致编译后的文件变大，但对整个磁盘空间来说已经不是什么大问题。所以为了调试的方便，建议你还是把它们保留着。

顺便提一下，案例中各种工具的安装方法，可以算是我们专栏学习的基本功，这一点希望你能够熟悉并掌握。还是那句话，不会安装先查文档，还是不行就上网搜索或者在文章里留言提问。

在这里也要表扬一下，很多同学已经把摸索到的方法分享到了留言中。记录并分享，是一个很好的习惯。

问题 2：如何用 perf 工具分析 Java 程序



阿巍-豆夫

写于 2018/11/30

不知道 perl 能不能定位到，java 动态字节码调用的 java 函数，貌似用 jstack 就在动态字节码这里断掉了。

引自：Linux性能优化实战

05 | 基础篇：某个应用的CPU使用率居然达到100%，该怎么办？

识别二维码打开原文
「极客时间」App





阿巍-豆夫

写于 2018/12/05

老师，我这里是个 java 应用，用 top jstack 追踪代码，发现 cpu 标高发生在一个动态字节码生成的类，然后这块是没有源码的，是不知道那一段函数的调用引起的。然后用 perf top -v 可以发现这个应用，kernal _spin_lock 这个长期占用 80% 的 cpu 我现在一点办法都没有，因为我不知道应用中那段程序使用了自旋锁，这个有什么办法追踪整个调用链呢？

引自：Linux性能优化实战

06 | 案例篇：系统的 CPU 使用率很高，但为啥却找不到高 CPU 的应用？

识别二维码打开原文
「极客时间」 App



这两个问题，其实是上一个 perf 问题的延伸。像是 Java 这种通过 JVM 来运行的应用程序，运行堆栈用的都是 JVM 内置的函数和堆栈管理。所以，从系统层面你只能看到 JVM 的函数堆栈，而不能直接得到 Java 应用程序的堆栈。

perf_events 实际上已经支持了 JIT，但还需要一个 /tmp/perf-PID.map 文件，来进行符号翻译。当然，开源项目 [perf-map-agent](#) 可以帮你生成这个符号表。

此外，为了生成全部调用栈，你还需要开启 JDK 的选项 -XX:+PreserveFramePointer。因为这里涉及到大量的 Java 知识，我就不再详细展开了。如果你的应用刚好基于 Java，那么你可以参考 NETFLIX 的技术博客 [Java in Flames](#)（链接为 <https://medium.com/netflix-techblog/java-in-flames-e763b3d32166>），来查看详细的使用步骤。

说到这里，我也想强调一个问题，那就是学习性能优化时，不要一开始就把自己限定在具体的某个编程语言或者性能工具中，纠结于语言或工具的细节出不来。

掌握整体的分析思路，才是我们首先要做的。因为，性能优化的原理和思路，在任何编程语言中都是相通的。

问题 3：为什么 perf 的报告中，很多符号都不显示调用栈

路过

写于 2018/12/07

老师，我使用perf record -g命令收集信息了。收集了很多次，每次都是command列为swapper的左侧有加号，而为app或其他的，都没有加号。但可以在symbol列，看到sys_read,new_sync_read, blkdev_read_iter等信息。我的系统也用Ubuntu 18.04. 不知道为何不能还原您的实现场景。

引自：Linux性能优化实战

08 | 案例篇：系统中出现大量不可中断进程和僵尸进程怎么办？（下）

识别二维码打开原文
「极客时间」App



perf report 是一个可视化展示 perf.data 的工具。在第 08 讲的案例中，我直接给出了最终结果，并没有详细介绍它的参数。估计很多同学的机器在运行时，都碰到了跟路过同学一样的问题，看到的是下面这个界面。

Samples: 38K of event 'cpu-clock', Event count (approx.): 9507250000					
Children	Self	Command	Shared Object	Symbol	
+ 99.29%	0.00%	swapper	[kernel.vmlinux]	[k]	0x000000000002000d5
+ 99.29%	0.00%	swapper	[kernel.vmlinux]	[k]	cpu_startup_entry
+ 99.29%	0.00%	swapper	[kernel.vmlinux]	[k]	do_idle
+ 99.27%	0.00%	swapper	[kernel.vmlinux]	[k]	default_idle_call
+ 99.27%	0.00%	swapper	[kernel.vmlinux]	[k]	arch_cpu_idle
+ 99.27%	0.00%	swapper	[kernel.vmlinux]	[k]	default_idle
+ 99.27%	99.00%	swapper	[kernel.vmlinux]	[k]	native_safe_halt
+ 70.31%	0.00%	swapper	[kernel.vmlinux]	[k]	start_secondary
+ 28.98%	0.00%	swapper	[kernel.vmlinux].init.text	[k]	x86_64_start_kernel
+ 28.98%	0.00%	swapper	[kernel.vmlinux].init.text	[k]	x86_64_start_reservations
+ 28.98%	0.00%	swapper	[kernel.vmlinux].init.text	[k]	start_kernel
+ 28.98%	0.00%	swapper	[kernel.vmlinux]	[k]	rest_init
0.35%	0.00%	app	libc-2.27.so	[.]	__libc_start_main
0.35%	0.00%	app	app	[.]	main
0.34%	0.00%	app	[unknown]	[k]	0x0e86258d4c544155
0.34%	0.00%	app	libc-2.27.so	[.]	read
0.34%	0.00%	app	[kernel.vmlinux]	[k]	entry_SYSCALL_64
0.34%	0.00%	app	[kernel.vmlinux]	[k]	do_syscall_64
0.34%	0.00%	app	[kernel.vmlinux]	[k]	sys_read
0.34%	0.00%	app	[kernel.vmlinux]	[k]	vfs_read
0.34%	0.00%	app	[kernel.vmlinux]	[k]	__vfs_read
0.34%	0.00%	app	[kernel.vmlinux]	[k]	new_sync_read
0.34%	0.00%	app	[kernel.vmlinux]	[k]	blkdev_read_iter

这个界面可以清楚看到，perf report 的输出中，只有 swapper 显示了调用栈，其他所有符号都不能查看堆栈情况，包括我们案例中的 app 应用。

这种情况我们以前也遇到过，当你发现性能工具的输出无法理解时，应该怎么办呢？当然还是查工具的手册。比如，你可以执行 man perf-report 命令，找到 -g 参数的说明：

[复制代码](#)

```

1 -g, --call-graph=<print_type,threshold[,print_limit],order,sort_key[,branch],value>
2     Display call chains using type, min percent threshold, print limit, call orde
3     ordering is not fixed so any parameter can be given in an arbitrary order. On
4     preceded by threshold.
5
6     print_type can be either:
7     - flat: single column, linear exposure of call chains.
8     - graph: use a graph tree, displaying absolute overhead rates. (default)
9     - fractal: like graph, but displays relative rates. Each branch of
10            the tree is considered as a new profiled object.
11     - folded: call chains are displayed in a line, separated by semicolons
12     - none: disable call chain display.
13
14     threshold is a percentage value which specifies a minimum percent to be
15     included in the output call graph. Default is 0.5 (%).
16
17     print_limit is only applied when stdio interface is used. It's to limit
18     number of call graph entries in a single hist entry. Note that it needs
19     to be given after threshold (but not necessarily consecutive).
20     Default is 0 (unlimited).
21

```

```
22         order can be either:
23         - callee: callee based call graph.
24         - caller: inverted caller based call graph.
25         Default is 'caller' when --children is used, otherwise 'callee'.
26
27         sort_key can be:
28         - function: compare on functions (default)
29         - address: compare on individual code addresses
30         - srcline: compare on source filename and line number
31
32         branch can be:
33         - branch: include last branch information in callgraph when available.
34           Usually more convenient to use --branch-history for this.
35
36         value can be:
37         - percent: diplay overhead percent (default)
38         - period: display event period
39         - count: display event count
```


通过这个说明可以看到，`-g` 选项等同于 `--call-graph`，它的参数是后面那些被逗号隔开的选项，意思分别是输出类型、最小阈值、输出限制、排序方法、排序关键词、分支以及值的类型。

我们可以看到，这里默认的参数是 `graph,0.5,caller,function,percent`，具体含义文档中都有详细讲解，这里我就不再重复了。

现在再回过头来看我们的问题，堆栈显示不全，相关的参数当然就是最小阈值 `threshold`。通过手册中对 `threshold` 的说明，我们知道，当一个事件发生比例高于这个阈值时，它的调用栈才会显示出来。

`threshold` 的默认值为 0.5%，也就是说，事件比例超过 0.5% 时，调用栈才能被显示。再观察我们案例应用 `app` 的事件比例，只有 0.34%，低于 0.5%，所以看不到 `app` 的调用栈就很正常了。

这种情况下，你只需要给 `perf report` 设置一个小于 0.34% 的阈值，就可以显示我们想看到的调用图了。比如执行下面的命令：

 复制代码

```
1 $ perf report -g graph,0.3
```

你就可以得到下面这个新的输出界面，展开 app 后，就可以看到它的调用栈了。

Samples: 38K of event 'cpu-clock', Event count (approx.): 9507250000					
Children	Self	Command	Shared Object	Symbol	
+ 99.29%	0.00%	swapper	[kernel.vmlinux]	[k]	0x00000000002000d5
+ 99.29%	0.00%	swapper	[kernel.vmlinux]	[k]	cpu_startup_entry
+ 99.29%	0.00%	swapper	[kernel.vmlinux]	[k]	do_idle
+ 99.27%	0.00%	swapper	[kernel.vmlinux]	[k]	default_idle_call
+ 99.27%	0.00%	swapper	[kernel.vmlinux]	[k]	arch_cpu_idle
+ 99.27%	0.00%	swapper	[kernel.vmlinux]	[k]	default_idle
+ 99.27%	99.00%	swapper	[kernel.vmlinux]	[k]	native_safe_halt
+ 70.31%	0.00%	swapper	[kernel.vmlinux]	[k]	start_secondary
+ 28.98%	0.00%	swapper	[kernel.vmlinux].init.text	[k]	x86_64_start_kernel
+ 28.98%	0.00%	swapper	[kernel.vmlinux].init.text	[k]	x86_64_start_reservations
+ 28.98%	0.00%	swapper	[kernel.vmlinux].init.text	[k]	start_kernel
+ 28.98%	0.00%	swapper	[kernel.vmlinux]	[k]	rest_init
- 0.35%	0.00%	app	libc-2.27.so	[.]	__libc_start_main
-		__libc_start_main			
-		main			
-	0.34%	read			
		entry_SYSCALL_64			
		do_syscall_64			
		sys_read			
-		vfs_read			
-	0.34%	__vfs_read			
		new_sync_read			
		blkdev_read_iter			
		generic_file_read_iter			
		blkdev_direct_IO			
+ 0.35%	0.00%	app	app	[.]	main
+ 0.34%	0.00%	app	[unknown]	[k]	0x0e86258d4c544155
+ 0.34%	0.00%	app	libc-2.27.so	[.]	read
+ 0.34%	0.00%	app	[kernel.vmlinux]	[k]	entry_SYSCALL_64

问题 4：怎么理解 perf report 报告

ninuxer

写于 2018/12/07

打卡day9

在用perf展开的分析详情中，有个children和self都90%多的swapper进程，如果我看到这个，我肯定会先去围绕这个进程展开，而会忽略占用才0.6%得app进程，关于这个情况，是我理解上有什么偏差么？

引自：Linux性能优化实战

08 | 案例篇：系统中出现大量不可中断进程和僵尸进程怎么办？（下）

识别二维码打开原文
「极客时间」App





仲鬼

写于 2018/12/05

老师好，关于 perf 有两个问题：

1. 请问 perf top 加 -g 时，前两列 children 和 self 两个指标各表示什么呢？按 Enter 键展开后的百分比又表示什么？自己研究 man 手册没理解。
2. 另外，我是做分布式存储的，用 perf 做性能分析时，是否需要考虑 perf 本身对于程序性能的影响？

引自：Linux性能优化实战

05 | 基础篇：某个应用的CPU使用率居然达到100%，该怎么办？

识别二维码打开原文
「极客时间」 App



看到这里，我估计你也曾嘀咕过，为啥不一上来就用 perf 工具解决，还要执行那么多其他工具呢？这个问题其实就给出了很好的解释。

在问题 4 的 perf report 界面中，你也一定注意到了，swapper 高达 99% 的比例。直觉来说，我们应该直接观察它才对，为什么没那么做呢？

其实，当你清楚了 swapper 的原理后，就很容易理解我们为什么可以忽略它了。

看到 swapper，你可能首先想到的是 SWAP 分区。实际上，swapper 跟 SWAP 没有任何关系，它只在系统初始化时创建 init 进程，之后，它就成为了一个最低优先级的空闲任务。也就是说，当 CPU 上没有其他任务运行时，就会执行 swapper。所以，你可以称它为“空闲任务”。

回到我们的问题，在 perf report 的界面中，展开它的调用栈，你会看到，swapper 时钟事件都耗费在了 do_idle 上，也就是在执行空闲任务。

```
Samples: 38K of event 'cpu-clock', Event count (approx.): 9507250000
Children      Self  Command      Shared Object      Symbol
- 99.29%      0.00%  swapper      [kernel.vmlinux]   [k] 0x00000000002000d5
- 0x2000d5
- 70.31% start_secondary
- cpu_startup_entry
- 70.31% do_idle
- 70.30% default_idle_call
  arch_cpu_idle
  default_idle
  native_safe_halt
- 28.98% x86_64_start_kernel
  x86_64_start_reservations
  start_kernel
  rest_init
  cpu_startup_entry
- do_idle
- 28.97% default_idle_call
  arch_cpu_idle
  default_idle
  native_safe_halt
+ 99.29%      0.00%  swapper      [kernel.vmlinux]   [k] cpu_startup_entry
+ 99.29%      0.00%  swapper      [kernel.vmlinux]   [k] do_idle
+ 99.27%      0.00%  swapper      [kernel.vmlinux]   [k] default_idle_call
+ 99.27%      0.00%  swapper      [kernel.vmlinux]   [k] arch_cpu_idle
+ 99.27%      0.00%  swapper      [kernel.vmlinux]   [k] default_idle
+ 99.27%      99.00%  swapper      [kernel.vmlinux]   [k] native_safe_halt
+ 70.31%      0.00%  swapper      [kernel.vmlinux]   [k] start_secondary
```

所以，分析案例时，我们直接忽略了前面这个 99% 的符号，转而分析后面只有 0.3% 的 app。其实从这里你也能理解，为什么我们一开始不先用 perf 分析。

因为在多任务系统中，次数多的事件，不一定是性能瓶颈。所以，只观察到一个大数值，并不能说明什么问题。具体有没有瓶颈，还需要你观测多个方面的多个指标，来交叉验证。这也是我在套路篇中不断强调的一点。

另外，关于 Children 和 Self 的含义，手册里其实有详细说明，还很友好地举了一个例子，来说明它们的百分比的计算方法。简单来说，

Self 是最后一列的符号（可以理解为函数）本身所占比例；

Children 是这个符号调用的其他符号（可以理解为子函数，包括直接和间接调用）占用的比例之和。

正如同学留言问到的，很多性能工具确实会对系统性能有一定影响。就拿 perf 来说，它需要在内核中跟踪内核栈的各种事件，那么不可避免就会带来一定的性能损失。这一点，虽然对大部分应用来说，没有太大影响，但对特定的某些应用（比如那些对时钟周期特别敏感的应用），可能就是灾难了。

所以，使用性能工具时，确实应该考虑工具本身对系统性能的影响。而这种情况，就需要你了解这些工具的原理。比如，

perf 这种动态追踪工具，会给系统带来一定的性能损失。

vmstat、pidstat 这些直接读取 proc 文件系统来获取指标的工具，不会带来性能损失。

问题 5：性能优化书籍和参考资料推荐



f

写于 2018/12/03

打卡

老师，是否提供些参考资料，书籍

引自：Linux性能优化实战

03 | 基础篇：经常说的 CPU 上下文切换是什么意思？

(上)

识别二维码打开原文
「极客时间」 App



我很高兴看到留言有这么高的学习热情，其实好多文章后面都有大量留言，希望我能推荐书籍和学习资料。这一点也是我乐意看到的。专栏学习一定不是你性能优化之旅的全部，能够带你入门、帮你解决实际问题、甚至是激发你的学习热情，已经让我非常开心。

在 [如何学习 Linux 性能优化](#) 的文章中，我曾经介绍过 Brendan Gregg，他是当之无愧的性能优化大师，你在各种 Linux 性能优化的文章中，基本都能看到他的那张性能工具图谱。

所以，关于性能优化的书籍，我最喜欢的其实正是他写的那本《Systems Performance: Enterprise and the Cloud》。这本书也出了中文版，名字是《性能之巅：洞悉系统、企

业与云计算》。

从出版时间来看，这本书确实算一本老书了，英文版的是 2013 年出版的。但是经典之所以成为经典，正是因为不会过时。这本书里的性能分析思路以及很多的性能工具，到今天依然适用。

另外，我也推荐你去关注他的个人网站 <http://www.brendangregg.com/>，特别是 [Linux Performance](#) 这个页面，包含了很多 Linux 性能优化的资料，比如：

Linux 性能工具图谱；

性能分析参考资料；

性能优化的演讲视频。

不过，这里很多内容会涉及到大量的内核知识，对初学者来说并不友好。但是，如果你想成为高手，辛苦和坚持都是不可避免的。所以，希望你在查看这些资料时，不要一遇到不懂的就打退堂鼓。任何东西的第一遍学习有不懂的地方很正常，忍住恐惧别放弃，继续往后走，前面很多问题可能会一并解决掉，再看第二遍、第三遍就更轻松了。

还是那句话，抓住主线不动摇，先从最基本的原理开始，掌握性能分析的思路，然后再逐步深入，探究细节，不要试图一口吃成个大胖子。

最后，欢迎继续在留言区写下你的疑问，我会持续不断地解答。我的目的仍然不变，希望可以和你一起，把文章的知识变成你的能力，我们不仅仅在实战中演练，也要在交流中进步。


Linux 性能优化实战

10 分钟帮你找到系统瓶颈

倪朋飞

微软资深工程师
Kubernetes 项目维护者



新版升级：点击「请朋友读」，10位好友免费读，邀请订阅更有**现金**奖励。

© 版权归极客邦科技所有，未经许可不得转载

上一篇 13 | Linux 性能优化答疑（一）

下一篇 15 | 基础篇：Linux内存是怎么工作的？

精选留言 (34)

写留言



我来也

2018-12-21

3

[D14打卡]

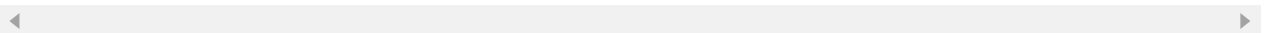
很赞同老师的观点：

"任何东西的第一遍学习有不懂的地方很正常，忍住恐惧别放弃，继续往后走，前面很多问题可能会一并解决掉，再看第二遍、第三遍就更轻松了。"

我好像很早就这样实践了：第一遍不管看不看得懂，先尽量细看。再特意过一段时间回头重...

展开 v

作者回复：英语也是基本要求，多读多看就熟悉了☺



**ninuxer**

2018-12-21

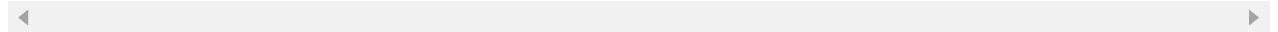


打卡day15

perf report中关于swapper的内容，后面我也去查了，才发现是自己理解有误，感谢老师指出，这里的swapper不是内存概念的swap，而是cpu空闲时执行的一个默认调用要啃啃《性能之巅：洞悉系统、企业与云计算》了，作者博客[http://www.brendangregg.com/...](http://www.brendangregg.com/)

展开 ▾

作者回复: 《深入Linux内核架构》挺不错，不过是大块头，啃下应该要花不少时间

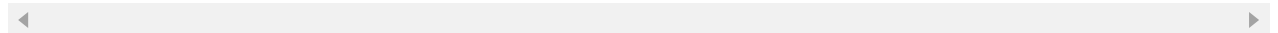
**全大神啊**

2019-02-15



花了两三天时间看完了CPU部分，收获良多，以前不怎么用的命令，看了老师的文章了解知道了工具的作用，以及原理，大爱~接下来内存部分-.-。

作者回复: 🐣

**荒唐_戏_**

2019-01-26



d3

展开 ▾

**刘飞**

2019-01-23



给大家分享下mac 跑docker时如果安装perf工具：

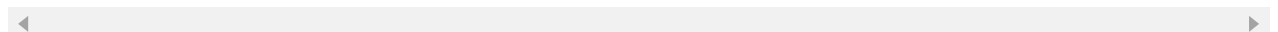
首先启动容器时，要增加启动参数--privileged，否则后续操作提示没权限

安装工具：apt-get install linux-source

可以在/usr/src下找到压缩包，解压，进入tools/perf，然后make&&make install进行编译...

展开 ▾

作者回复: 我还真没用 mac 中的 docker 跑过。谢谢分享





辣椒

2019-01-07

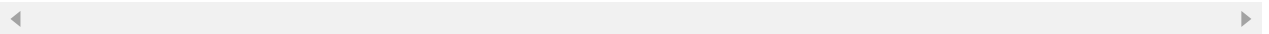


老师，我把perf.data拷贝进容器，然后在容器中按照提示安装了perf, 再执行perf_4.9 report时报以下信息：

```
Kernel address maps (/proc/{kallsyms,modules}) were restricted.  
Check /proc/sys/kernel/kptr_restrict before running 'perf record'  
As no suitable kallsyms nor vmlinux was found, kernel samples ...
```

展开 ▾

作者回复: echo 0 > /proc/sys/kernel/kptr_restrict



training

2019-01-04



在macbook上用docker上运行phpfpm实例，是不是没有办法跑通perf那些测试,学习用。

展开 ▾



training

2019-01-04



在macbook上docker运行phpfpm 那个实例，有什么办法运行perfn

展开 ▾



小美

2019-01-02



我们的内核版本是3.10，安装go，然后再安装pcstat也不太方便。

于是我用perf观察cache，次方案是否有缺陷？

我的命令如下，观察60s

```
perf stat -e cache-misses -p [PID] -a sleep 60
```



花开蝶自来

2019-01-01



d5

展开 ▾



chon

2018-12-31



linux内核相关知识，有啥书推荐没？谢谢

展开 ▾



大飞

2018-12-31



打卡

展开 ▾



如果

2018-12-29



DAY14，打卡

展开 ▾



Only now

2018-12-27



mark

展开 ▾



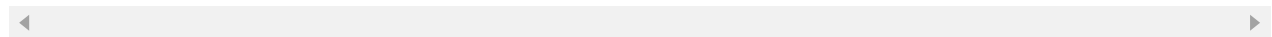
子轩Zixuan

2018-12-25



感谢老师耐心回复，接着线上定位问题想问下，如果要事先做好监控系统，一般需要做到什么程度？目前只有阿里云的监控感觉不够用

作者回复: 对的，至少要从系统和应用两个层面监控。系统层面就是我们专栏里提到的系统资源使用情况；应用层面除了进程的资源监控之外，还有应用对外的接口、与其他服务间调用、数据库、应用内部功能模块等一系列的metrics（可以参考APM）





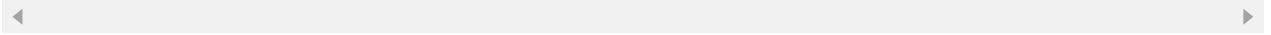
腾达-12-24



docker符号找不到的问题，文章里已经给了几个方法，我想问的是，是不是可以直接把docker的符号下载下来？从宿主直接看函数调用栈？docker不会像java一样，在java代码层面后，从系统层面(c语言层面)很难查问题？

展开 ∨

作者回复: 看应用的类型，有些静态编译的可以下载下来，但也有很多应用依赖大量的动态库，放到容器外面就是这儿提到的方法一了



风飘, 吾独...

2018-12-24



打卡

展开 ∨



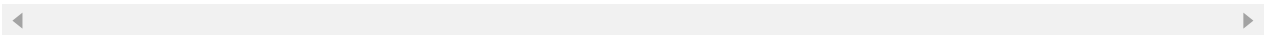
子轩Zixuan

2018-12-23



趁周末跟上了老师的脚步，老师讲解的很好，感谢！另外我注意到老师在专栏里的实例都是现场调试，但是我遇到过实际情况需要尽快恢复服务，先把代码还原重新发布保证服务可用，只留下一台保留问题现场的机器，但是已经不接受请求了，像这种情况除了事先监控以外，还有别的方案能定位问题吗？

作者回复: 是的，线上问题要优先恢复，排查不能花过多时间，所以很多情况下都要靠监控系统了解当时的状况。除此之外，原来服务还在的话，可以试试能否重现问题，比如模拟当时的请求；或者也可以从日志中寻找线索。



许山山

2018-12-23



这两期的答疑帮助很大，谢谢老师了。

展开 ∨



Griffin

2018-12-23



终于赶上了，倪老师能不能讲讲网络问题应该怎么排查呀，最近老是被docker，docker swarm的container寻址困扰。

作者回复: 不要急，我们有网络模块的

