火丁笔记

多研究些问题,少谈些主义。

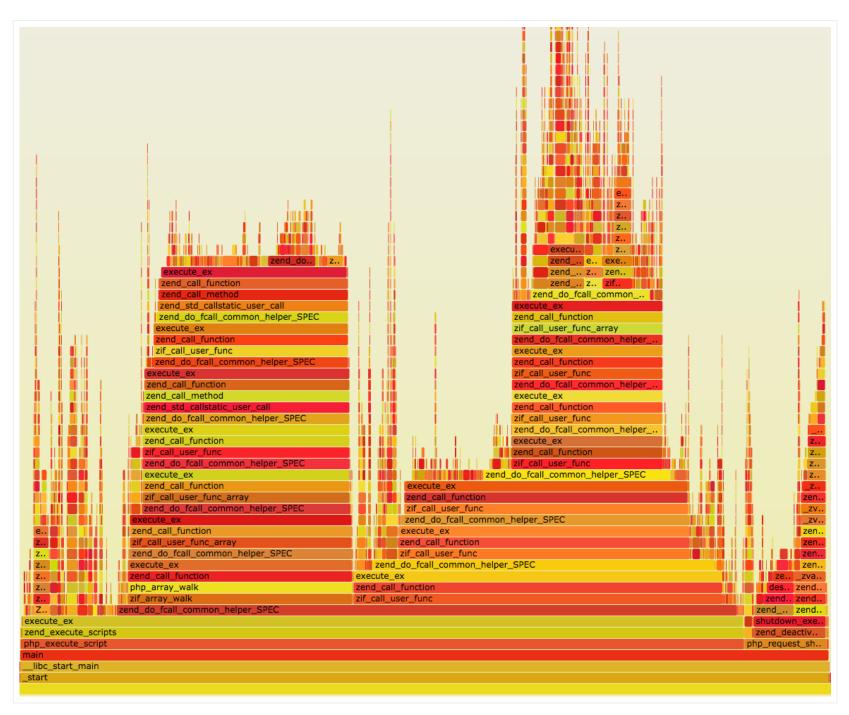
白话火焰图

发表于2016-08-18

很多人感冒发烧的时候,往往会模仿神农氏尝百草的路子:先尝尝抗病毒的药,再试试抗细菌的药,甭管家里有什么药挨个试,什么中药西药, 瞎猫总会碰上死耗子,如此做法自然是不可取的,正确的做法应该是去医院验个血,确诊后再对症下药。

让我们回想一下我们一般是如何调试程序的:通常是在没有数据的情况下依靠主观臆断来瞎蒙,而不是考虑问题到底是什么引起的!毫无疑问,调优程序性能问题的时候,同样需要对症下药。好消息是 Brendan D. Gregg 发明了火焰图,可以一针见血的指出程序的性能瓶颈,坏消息是除了 OpenResty 社区,很少看到还有其他人使用火焰图。

常见的火焰图类型有 On-CPU,Off-CPU,还有 Memory,Hot/Cold,Differential 等等。下面给出某个 PHP 程序的 On-CPU 类型的火焰图例子:



— Flame Graph

关于火焰图详细的介绍可以参考 Blazing Performance with Flame Graphs,简而言之:整个图形看起来就像一团跳动的火焰,这也正是其名字的由来。燃烧在火苗尖部的就是 CPU 正在执行的操作,不过需要说明的是颜色是随机的,本身并没有特殊的含义,纵向表示调用栈的深度,横向表示消耗的时间。因为调用栈在横向会按照字母排序,并且同样的调用栈会做合并,所以一个格子的宽度越大越说明其可能是瓶颈。综上所述,主要就是看那些比较宽大的火苗,特别留意那些类似平顶山的火苗。

要生成火焰图,必须要有一个顺手的 Tracer 工具,如果操作系统是 Linux 的话,那么选择通常是 perf,systemtap 中的一种。其中 perf 相对更常用,多数 Linux 都包含了它,有兴趣的读者稍后可以参考 Linux Profiling at Netflix 中的介绍,尤其是里面关于如何处理 Broken stacks 问题的描述,建议多看几遍,而 systemtap 相对更强大,不过缺点是你需要先学会它本身的编程语言,如果你和我一样觉得麻烦,那么我强烈推荐你使用春哥的 nginx-systemtap-toolkit,乍一看名字你可能会误以为这个工具包是 nginx 专用的,实际上这里面很多工具适用于任何 C/CPP 语言编写的程序:

https://huoding.com/2016/08/18/531

- sample-bt: 用来生成 On-CPU 火焰图的采样数据(DEMO)
- sample-bt-off-cpu: 用来生成 Off-CPU 火焰图的采样数据(DEMO)

那么什么时候使用 On-CPU 火焰图?什么时候使用 Off-CPU 火焰图呢?取决于当前的瓶颈到底是什么,如果是 CPU 则使用 On-CPU 火焰图,如果是 IO 或锁 则使用 Off-CPU 火焰图。如果无法确定,那么可以通过压测工具来确认:通过压测工具看看能否让 CPU 使用率趋于饱和,如果能那么使用 On-CPU 火焰图,如果不管怎么压,CPU 使用率始终上不来,那么多半说明程序被 IO 或锁卡住了,此时适合使用 Off-CPU 火焰图。如果还是确认不了,那么不妨 On-CPU 火焰图和 Off-CPU 火焰图都搞搞,正常情况下它们的差异会比较大,如果两张火焰图长得差不多,那么通常认为 CPU 被其它进程抢占了。

在采样数据的时候,最好通过压测工具对程序持续施压,以便采集到足够的样本。关于压测工具的选择,如果选择 ab 的话,那么务必记得开启-k 选项,以避免耗尽系统的可用端口。此外,我建议尝试使用诸如 wrk 之类更现代的压测工具。

请按照官方说明来安装。需要着重说明的是,当你安装 kernel-devel 和 kernel-debuginfo 的时候,务必保证所安装的版本和当前内核版本一致,以 CentOS 为例:

```
shell> yum install yum-utils
shell> yum install kernel-devel
shell> debuginfo-install kernel
```

当生成的火焰图中有很多十六进制的乱码时,那么意味着对应程序缺失了 debuginfo,可以借助 gdb 来确认这一点,方法如下所示:

```
shell> gdb -p <PID>
```

好消息是如果缺失了某些 debuginfo,那么 gdb 会在结尾提示你用 debuginfo-install 命令来安装,坏消息是如果你直接运行多半没有效果,因为 CentOS 缺省没有激活对应的仓库,所以需要在「/etc/yum.repos.d/CentOS-Debuginfo.repo」中设置 enabled=1。

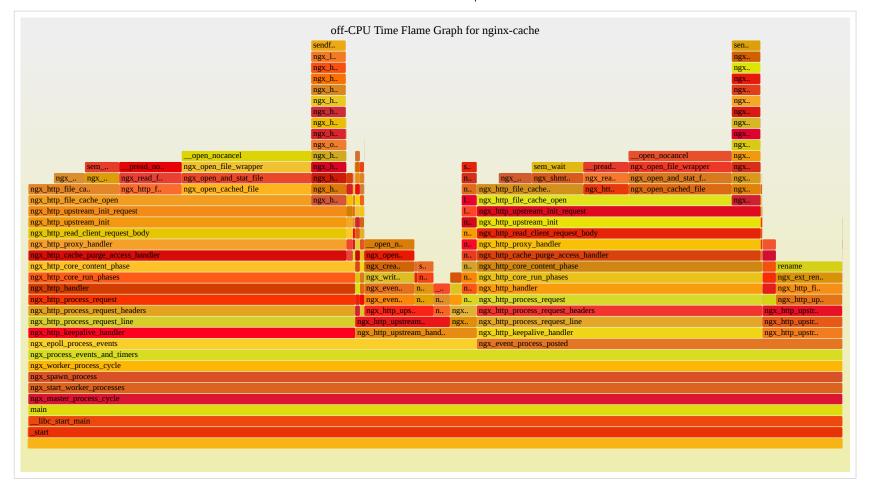
要想熟练的使用火焰图的话,最好的方法就是多看别人的例子。春哥在微博上发过很多例子,这里我列举最具代表性的几个,版权自然归春哥所有:

微博来源:通过 Off-CPU 火焰图可以发现有一个使用互斥锁的 HTTP Cache 检查代码让绝大部分的 Off-CPU 时间都花在了等待进程锁上:

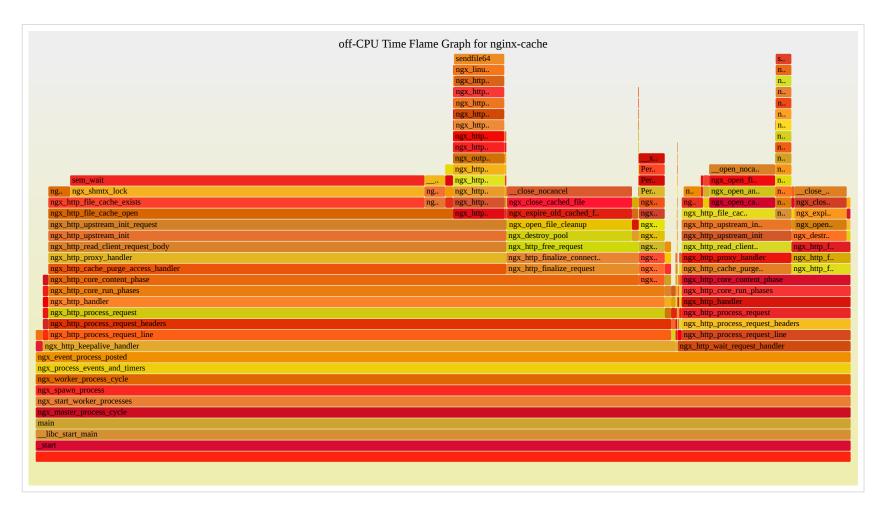


微博来源:在移除了那个引发互斥锁瓶颈的历史代码之后,从图上我们可以清楚地看到 open()系统调用是下一个最明显的性能瓶颈:

18/04/2019 白话火焰图 | 火丁笔记



微博来源:启用 nginx 的 open_file_cache 指令可以对打开的文件句柄进行缓存,从而节约昂贵的 open() 系统调用。但是缓存容量并不是越大越好,比如当达到 20000 个元素的容量时,共享内存的锁就成了瓶颈。



如果没有火焰图,我们可能会在解决一个问题后引入另一个问题。

实际使用火焰图的时候,因为 perf / systemtap 本身对系统性能影响较小,所以我们可以在线上随时采样数据来分析性能,我们甚至可以写一个脚本,自动化定期绘制系统运行状况的火焰图,如此一来,即便发生性能故障时我们没有第一时间在现场,也可以随时根据火焰图历史快照来确诊问题的根源。

18/04/2019 白话火焰图 | 火丁笔记



一 欢迎关注我们的公众号

此条目由老王发表在Technical分类目录,并贴了Linux、Performance标签。将固定链接 [https://huoding.com/2016/08/18/531] 加入收藏夹。

4 THOUGHTS ON "白话火焰图"

刘文学

在2016-09-01 10:39:41说道:

老司机果实厉害

1

在2016-12-08 16:28:18说道:

老司机就是不一样

吴中勤

在2017-03-20 14:25:07说道:

老司机

Pingback引用通告: 使用火焰图做性能分析 | neo时刻准备着