秘籍：分析Linux性能问题！只要一分钟！

秘籍：分析Linux性能问题！只要一分钟！

译者：秦晓辉@快猫星云 2023年2月27日

> 本文翻译自：Linux Performance Analysis in 60,000 Milliseconds，加了一些个人理解，所以只能算意译了 :) 未来如果遇到一些机器性能问题，可以按图索骥，利用本文的命令挨个排查，所以，收藏起来吧，收藏=学会，哈哈。

如果某个Linux有性能问题，你登录之后来排查，第一分钟内你会使用哪些命令做那些方面的排查?

在 Netflix，我们在云端有大量的 EC2 实例，有好几个性能分析工具来监控和排查性能问题。包括全云监控产品 Atlas 以及实例粒度的 Vector。尽管这些工具可以帮我们解决大部分问题，但有的时候，我们还是要登录实例运行一些标准的Linux命令来辅助排查。

第一分钟：总结

在这篇文章中，Netflix 的性能工程团队将会为你展示如何在最初的 60 秒内通过命令行做性能排查，这个过程中，会使用一些标准的 Linux 工具。在 60 秒内，你可以通过运行以下十个命令来大致了解系统资源使用情况和正在运行的进程。寻找错误和饱和度指标，然后是资源利用率相关的指标。饱和度这个词可能不易理解，大家可以Google一下USE方法，所谓了饱和了，是资源的负载超过其处理能力了，可以表现为请求队列的长度或等待时间。

```
uptime
dmesg | tail
vmstat 1
mpstat -P ALL 1
pidstat 1
iostat -xz 1
free -m
sar -n DEV 1
sar -n TCP,ETCP 1
top
```

这就是那10条命令，有些命令需要提前安装sysstat包，请注意。这些命令暴露的指标数据，将会帮助你完成 USE Method（这是定位性能瓶颈的一个方法论） 分析。包括对 CPU、内存、硬盘等各个方面做检查，检查利用率、饱和度、错误相关的指标。另外还要注意应用排除法，当我们排除了某类资源的问题，也就意味着缩小了排查范围，有利于后续排障。

以下部分总结了这些命令，并附有来自生产系统的示例。有关这些工具的更多信息，请参阅它们的使用手册。

1、uptime

```
$ uptime
23:51:26 up 21:31, 1 user, load average: 30.02, 26.43, 19.02
```

这是一种用来快速查看系统平均负载的方法，它表明了系统中有多少要运行的任务（进程）。在 Linux 系统中，这些数字包含了需要在 CPU 中运行的进程以及正在等待 I/O（通常是磁盘 I/O）的进程。它仅仅是对系统负载的一个粗略展示，稍微看下即可。你还需要其他工具来进一步了解具体情况。

这三个数字展示的是一分钟、五分钟和十五分钟内系统的负载总量平均值按照指数比例压缩得到的结果。从中我们可以看到系统的负载是如何随时间变化的。比方你在检查一个问题，然后看到 1 分钟对应的值远小于 15 分钟的值，那么可能说明这个问题已经过去了，你没能及时观察到。

在上面这个例子中，**系统负载在随着时间增加**，因为最近一分钟的负载值超过了 30，而 15 分钟的平均负载则只有 19。这样显著的差距包含了很多含义，比方 CPU 负载。若要进一步确认的话，则要运行 vmstat 或 mpstat 命令，这两个命令请参考后面的章节。

2、dmesg | tail

```
$ dmesg | tail
[1880957.563150] perl invoked oom-killer: gfp_mask=0x280da, order=0, oom_score_adj=0
[...]
[1880957.563400] Out of memory: Kill process 18694 (perl) score 246 or sacrifice child
[1880957.563408] Killed process 18694 (perl) total-vm:1972392kB, anon-rss:1953348kB, fil
e-rss:0kB
[2320864.954447] TCP: Possible SYN flooding on port 7001. Dropping request.  Check SNMP
counters.
```

如果系统中存在系统消息，则这条命令会显示了最近的 10 条。从这些消息中找一下有没有 Errors，这些Error或许可以帮你定位系统性能问题。上面的例子包含了 oom-killer，以及 TCP 丢弃一个请求。

千万不要错过这一步！dmesg 命令永远值得一试。

3、vmstat 1

```
$ vmstat 1
procs ---------memory---------- ---swap-- -----io---- -system-- ------cpu-----
 r  b swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
34  0    0 200889792  73708 591828    0    0     0     5    6   10 96  1  3  0  0
32  0    0 200889920  73708 591860    0    0     0   592 13284 4282 98  1  1  0  0
32  0    0 200890112  73708 591860    0    0     0     0 9501 2154 99  1  0  0  0
32  0    0 200889568  73712 591856    0    0     0    48 11900 2459 99  0  0  0  0
32  0    0 200890208  73712 591860    0    0     0     0 15898 4840 98  1  1  0  0
^C
```

vmstat(8) 是 virtual memory stat 的缩写，是一种常用工具（几十年前首次为 BSD 创建）。它在每一行打印关键服务器统计信息的摘要。

vmstat的运行参数为1，每秒打印一次，统计最近一秒钟的情况，不过请忽略第一行，第一行统计的不是前一秒的情况，是统计的OS启动以来的平均值，对于我们排查问题而言，没啥帮助。

需要重点查看的列

- **r**：正在运行或等待运行的进程总数。相比 uptime 中的平均负载数据，这个 r 的值不包含 I/O，可以更好的体现 CPU 的饱和情况。如果这个值大于 CPU 的核数，就表示过于饱和了。
- **free**：使用 KB 单位统计的空闲内存。如果你发现这里是一长串数字，说明你还有很多内存可用 :) 后面我们会介绍 `free -m` 命令，可以更好的解释空闲内存的情况。
- **si,so**：Swap换入换出的量。如果这些值非0，说明你内存不够用喽。
- **us,sy,id,wa,st**：这些是CPU分解值，是针对所有CPU的一个平均值，并非针对某个核心的。us是user time，sy是system time(kernel)，id是idle，wa是wait I/O，st是stolen time（虚拟机才需要关注st）

把用户态时间和内核态时间相加，可以得知CPU是否繁忙。一个恒定水平的wa值表示I/O方面有性能瓶颈。如果wa很高，idle就会很高，因为CPU一直在等待I/O获取数据，没有办法继续运算。

系统时间system time对于I/O处理是必要的。一个高的系统时间平均值，比如超过20%，可以进一步探查，也许内核处理I/O的效率很低。

上面的例子中，CPU时间主要花在 user 上面，说明是用户态的应用程序在占用 CPU 时间。CPU 的平均利用率也远远超过了 90%。但这不一定是个问题，通过 `r` 列的值来检查饱和程度。

4、mpstat -P ALL 1

```
$ mpstat -P ALL 1
Linux 3.13.0-49-generic (titanclusters-xxxxx)  07/14/2015  _x86_64_ (32 CPU)

07:38:49 PM  CPU   %usr  %nice   %sys %iowait   %irq  %soft  %steal  %guest  %gnice  %id
le
07:38:50 PM  all  98.47   0.00   0.75    0.00   0.00   0.00    0.00    0.00    0.00   0.
78
07:38:50 PM    0  96.04   0.00   2.97    0.00   0.00   0.00    0.00    0.00    0.00   0.
99
07:38:50 PM    1  97.00   0.00   1.00    0.00   0.00   0.00    0.00    0.00    0.00   2.
00
07:38:50 PM    2  98.00   0.00   1.00    0.00   0.00   0.00    0.00    0.00    0.00   1.
00
07:38:50 PM    3  96.97   0.00   0.00    0.00   0.00   0.00    0.00    0.00    0.00   3.
03
[...]
```

这个命令打印每个 CPU 核心的时间花费情况，可以用来检查是否有不均衡的情况。一个高使用率的 CPU，可能是某个单线程应用导致的。

5、pidstat 1

```
$ pidstat 1
Linux 3.13.0-49-generic (titanclusters-xxxxx)  07/14/2015    _x86_64_    (32 CPU)

07:41:02 PM   UID       PID    %usr %system  %guest    %CPU   CPU  Command
07:41:03 PM     0         9    0.00    0.94    0.00    0.94     1  rcuos/0
07:41:03 PM     0      4214    5.66    5.66    0.00   11.32    15  mesos-slave
07:41:03 PM     0      4354    0.94    0.94    0.00    1.89     8  java
07:41:03 PM     0      6521 1596.23    1.89    0.00 1598.11    27  java
07:41:03 PM     0      6564 1571.70    7.55    0.00 1579.25    28  java
07:41:03 PM 60004     60154    0.94    4.72    0.00    5.66     9  pidstat

07:41:03 PM   UID       PID    %usr %system  %guest    %CPU   CPU  Command
07:41:04 PM     0      4214    6.00    2.00    0.00    8.00    15  mesos-slave
```

```
07:41:04 PM     0      6521 1590.00    1.00    0.00 1591.00    27  java
07:41:04 PM     0      6564 1573.00   10.00    0.00 1583.00    28  java
07:41:04 PM   108      6718    1.00    0.00    0.00    1.00     0  snmp-pass
07:41:04 PM 60004     60154    1.00    4.00    0.00    5.00     9  pidstat
^C
```

pidstat 有点像 top 针对特定进程的统计视图，与 top 不同的是，pidstat 会滚动打印，而不是清除屏幕。这对于观察一段时间内的模式很有用，也可以把你看到的东西（复制-粘贴）记录到你的调查记录中。

上面的例子确定了两个负责消耗CPU的java进程。 `%CPU` 一栏是所有CPU的总和； `1591%` 显示这个java进程几乎消耗了16个CPU。

6、iostat -xz 1

```
$ iostat -xz 1
Linux 3.13.0-49-generic (titanclusters-xxxxx)  07/14/2015  _x86_64_ (32 CPU)

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
          73.96    0.00    3.73    0.03    0.06   22.21

Device:   rrqm/s   wrqm/s     r/s     w/s    rkB/s    wkB/s avgrq-sz avgqu-sz   await r_
await w_await  svctm  %util
xvda        0.00     0.23    0.21    0.18     4.52     2.08    34.37     0.00    9.98
13.80    5.42   2.44   0.09
xvdb        0.01     0.00    1.02    8.94   127.97   598.53   145.79     0.00    0.43
1.78    0.28   0.25   0.25
xvdc        0.01     0.00    1.02    8.86   127.79   595.94   146.50     0.00    0.45
1.82    0.30   0.27   0.26
dm-0        0.00     0.00    0.69    2.32    10.47    31.69    28.01     0.01    3.23
0.71    3.98   0.13   0.04
dm-1        0.00     0.00    0.00    0.94     0.01     3.78     8.00     0.33  345.84
0.04  346.81   0.01   0.00
dm-2        0.00     0.00    0.09    0.07     1.35     0.36    22.50     0.00    2.55
0.23    5.62   1.78   0.03
[...]
^C
```

这是了解块状设备（磁盘）的一个很好的工具，包括应用的工作负载和由此产生的性能指标。

- **r/s, w/s, rkB/s, wkB/s**：这些是设备的每秒读次数、写次数、读KB和写KB。使用这些来描述工作负载。一个性能问题可能仅仅是由于过分的负载。

- **await**：I/O的平均时间，以毫秒计。这是应用的I/O请求所耗费的时间，它包括排队的时间和被服务的时间。大于预期的平均时间可能是设备饱和的指标，或设备问题。
- **avgqu-sz**: 发给设备的平均请求数。大于1的值可能是饱和的证据（尽管设备通常可以并行处理请求，特别是在多个后端磁盘前的虚拟设备）。
- **%util**：设备利用率。这实际上是一个繁忙的百分比，显示设备每秒钟进行工作的时间。大于60%的值通常会导致性能不佳（应该在等待中看到），尽管这取决于设备的情况。接近100%的值通常表示饱和。

如果存储设备是一个逻辑磁盘设备，面向许多后端磁盘，那么100%的利用率可能只是意味着一些I/O在100%的时间内被处理，然而，后端磁盘可能远远没有饱和，可能能够处理更多的工作。

请记住，磁盘I/O性能差并不一定是应用程序的问题。许多技术通常用于异步执行I/O，这样应用程序就不会阻塞并直接受到延迟的影响（例如，读取时的超前读取和写入时的缓冲）。

7、free -m

```
$ free -m
             total       used       free     shared    buffers     cached
Mem:        245998      24545     221453         83         59        541
-/+ buffers/cache:      23944     222053
Swap:            0          0          0
```

右侧的两列：

- **buffers**：用于块设备 I/O 的缓冲区缓存
- **cached**：用于文件系统的页面缓存

通常，我们需要检查这些值是否接近 0 了，如果快到 0 了可能会导致更高的硬盘 I/O（使用 iostat 做进一步确认）以及更差的性能。上面例子里的值看起来还不错，每个都还有很多 MB。

`-/+ buffers/cache` 为 used 和 free 内存量提供了更直观的数值。Linux将空闲内存用于缓存，但如果应用程序需要它，可以快速回收。所以在某种程度上，缓存的内存应该包括在空闲内存一栏中，这一行就是这样做的。甚至有一个网站，linuxatemyram，来解释这个让人困惑的情况。

如果在Linux上使用ZFS，就像我们在一些服务中所做的那样，这可能会产生额外的困惑，因为ZFS有它自己的文件系统缓存，并没有被 `free -m` 列正确反映出来。看起来系统的可用内存很低，而实际上这些内存在需要时可以从ZFS缓存中使用。

8、sar -n DEV 1

```
$ sar -n DEV 1
Linux 3.13.0-49-generic (titanclusters-xxxxx)  07/14/2015     _x86_64_    (32 CPU)

12:16:48 AM     IFACE   rxpck/s   txpck/s    rxkB/s    txkB/s   rxcmp/s   txcmp/s  rxmcs
t/s   %ifutil
12:16:49 AM      eth0  18763.00   5032.00  20686.42    478.30      0.00      0.00
0.00      0.00
12:16:49 AM        lo     14.00     14.00      1.36      1.36      0.00      0.00
0.00      0.00
12:16:49 AM   docker0      0.00      0.00      0.00      0.00      0.00      0.00
0.00      0.00

12:16:49 AM     IFACE   rxpck/s   txpck/s    rxkB/s    txkB/s   rxcmp/s   txcmp/s  rxmcs
t/s   %ifutil
12:16:50 AM      eth0  19763.00   5101.00  21999.10    482.56      0.00      0.00
0.00      0.00
12:16:50 AM        lo     20.00     20.00      3.25      3.25      0.00      0.00
0.00      0.00
12:16:50 AM   docker0      0.00      0.00      0.00      0.00      0.00      0.00
0.00      0.00
^C
```

使用这个工具来检查网络接口的吞吐量：rxkB/s和txkB/s，作为衡量工作量的标准，当然，也要检查是否触发了一些限制。在上面的例子中，eth0的接收速度达到了22Mbytes/s，也就是176Mbits/sec（远低于例如1Gbit/sec的限制）。

这个版本还有 `%ifutil` 用于设备利用率（全双工的两个方向的最大值），这也是我们使用Brendan的nicstat工具来测量的东西。和nicstat一样，这很难搞正确，而且在这个例子中似乎不工作(0.00)。

9、sar -n TCP,ETCP 1

```
$ sar -n TCP,ETCP 1
Linux 3.13.0-49-generic (titanclusters-xxxxx)  07/14/2015    _x86_64_    (32 CPU)

12:17:19 AM  active/s passive/s    iseg/s    oseg/s
12:17:20 AM      1.00      0.00  10233.00  18846.00

12:17:19 AM  atmptf/s  estres/s retrans/s isegerr/s   orsts/s
12:17:20 AM      0.00      0.00      0.00      0.00      0.00

12:17:20 AM  active/s passive/s    iseg/s    oseg/s
12:17:21 AM      1.00      0.00   8359.00   6039.00

12:17:20 AM  atmptf/s  estres/s retrans/s isegerr/s   orsts/s
```

```
12:17:21 AM      0.00      0.00      0.00      0.00      0.00
^C
```

这是对一些关键的TCP指标的摘要式视图。这些指标包括：

- **active/s**：每秒本地发起的TCP连接的数量（例如，通过connect()）
- **passive/s**：每秒远程启动的TCP连接的数量（例如，通过accept()）
- **retrans/s**：每秒的TCP重传次数

主动和被动计数通常作为服务器负载的一个粗略衡量标准：新接受的连接数（被动），以及下游连接数（主动）。姑且可以把主动看作是出站，把被动看作是入站，但这并不严格正确（例如，有的时候是本地主机到本地主机）。

重传是网络或服务器问题的标志；可能是一个不可靠的网络（如公共互联网），也可能是由于服务器过载而丢弃数据包。上面的例子显示每秒钟只有一个新的TCP连接。

10、top

```
$ top
top - 00:15:40 up 21:56,  1 user,  load average: 31.09, 29.87, 29.92
Tasks: 871 total,   1 running, 868 sleeping,   0 stopped,   2 zombie
%Cpu(s): 96.8 us,  0.4 sy,  0.0 ni,  2.7 id,  0.1 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem:  25190241+total, 24921688 used, 22698073+free,    60448 buffers
KiB Swap:        0 total,        0 used,        0 free.   554208 cached Mem

   PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 20248 root      20   0  0.227t 0.012t  18748 S  3090  5.2  29812:58 java
  4213 root      20   0 2722544  64640  44232 S  23.5  0.0 233:35.37 mesos-slave
 66128 titancl+  20   0   24344   2332   1172 R   1.0  0.0   0:00.07 top
  5235 root      20   0 38.227g 547004  49996 S   0.7  0.2   2:02.74 java
  4299 root      20   0 20.015g 2.682g  16836 S   0.3  1.1  33:14.42 java
     1 root      20   0   33620   2920   1496 S   0.0  0.0   0:03.82 init
     2 root      20   0       0      0      0 S   0.0  0.0   0:00.02 kthreadd
     3 root      20   0       0      0      0 S   0.0  0.0   0:05.35 ksoftirqd/0
     5 root       0 -20       0      0      0 S   0.0  0.0   0:00.00 kworker/0:0H
     6 root      20   0       0      0      0 S   0.0  0.0   0:06.94 kworker/u256:0
     8 root      20   0       0      0      0 S   0.0  0.0   2:38.05 rcu_sched
```

top 命令包括我们先前检查的许多指标。它适合用来查看相比于之前的命令输出的结果，负载有了哪些变动。

top命令的一个缺点是，它很难看到随时间变化的模式，而像vmstat和pidstat这样提供滚动输出的工具可能更清楚。如果你没有足够快地暂停输出（Ctrl-S暂停，Ctrl-Q继续），屏幕就会清空，那么间歇性问题的证据也会丢失。

后续分析

还有更多的命令和方法，你可以应用这些命令和方法来深入研究。请看Brendan在2015年Velocity上的Linux性能工具教程，它通过40多个命令，涵盖了可观察性、基准测试、调优、静态性能调优、剖析和跟踪。

关于译者

本文译者秦晓辉，Flashcat合伙人，文章内容是Flashcat技术团队共同沉淀的结晶，作者做了编辑整理，我们会持续输出监控、稳定性保障相关的技术文章，文章可转载，转载请注明出处，尊重技术人员的成果。
标签： SRE