


[译]Linux性能分析的前60000毫秒

10 回复 64 查看




(<https://www.shiyanlou.com/user/8490>) 实验楼管理员  (<https://www.shiyanlou.com/vip>)

6小时前

技术分享 (<https://www.shiyanlou.com/questions/?tag=技术分享>)


原文链接: <http://techblog.netflix.com/2015/11/linux-performance-analysis-in-60s.html>

作者是Brendan Gregg，Oracle/Linux系统性能分析方面的大牛。

 分享到微博

全部回答



实验楼管理员 (<https://www.shiyanlou.com/user/8490>)  (<https://www.shiyanlou.com/vip>)

(<https://www.shiyanlou.com/user/8490>)

Linux性能分析的前60000毫秒

为了解决性能问题，你登入了一台Linux服务器，在最开始的一分钟内需要查看什么？

在Netflix我们有一个庞大的EC2 Linux集群，还有非常多的性能分析工具来监控和调查它的性能。其中包括用于云监控的Atlas (<http://techblog.netflix.com/2014/12/introducing-atlas-netflixs-primary.html>)，用于实例按需分析的Vector (<http://techblog.netflix.com/2015/04/introducing-vector-netflixs-on-host.html>)。即使这些工具帮助我们解决了大多数问题，我们有时还是得登入Linux实例，运行一些标准的Linux性能工具来解决问题。

在这篇文章里，Netflix Performance Engineering团队将使用居家常备的Linux标准命令行工具，演示在性能调查最开始的60秒里要干的事，

最开始的60秒.....

运行下面10个命令，你可以在60秒内就对系统资源的使用情况和进程的运行状况有大体上的了解。无非是先查看错误信息和饱和指标，再看下资源的使用量。这里“饱和”的意思是，某项资源供不应求，已经造成了请求队列的堆积，或者延长了等待时间。

```
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
```

有些命令需要你安装 `sysstat` 包。（译注：指 `mpstat`, `pidstat`, `iostat` 和 `sar`，用包管理器直接安装 `sysstat` 即可）这些命令所提供的指标能够帮助你实践USE (<http://www.brendangregg.com/usemethod.html>)方法：这是一种用于定位性能瓶颈的方法论。你可以以此检查所有资源（CPU，内存，硬盘，等等）的使用量，是否饱和，以及是否存在错误。同时请留意上一次检查正常的时刻，这将帮助你减少待分析的对象，并指明调查的方向。（译注：USE方法，就是检查每一项资源的使用量（utilization）、饱和（saturation）、错误（error））

接下来的章节里我们将结合实际例子讲解这些命令。如果你想了解更多的相关信息，请查看它们的man page。

6小时前



1. uptime

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

这个命令显示了要运行的任务（进程）数，通过它能够快速了解系统的平均负载。在Linux上，这些数值既包括正在或准备运行在CPU上的进程，也包括阻塞在uninterruptible I/O（通常是磁盘I/O）上的进程。它展示了资源负载（或需求）的大致情况，不过进一步的解读还有待其它工具的协助。对它的具体数值不用太较真。

最右的三个数值分别是1分钟、5分钟、15分钟系统负载的移动平均值。它们共同展现了负载随时间变动的情况。举个例子，假设你被要求去检查一个出了问题的服务器，而它最近1分钟的负载远远低于15分钟的负载，那么你可能已经扑了个空。

在上面的例子中，负载均值最近呈上升态势，其中1分钟值高达30，而15分钟值仅有19。这种现象有许多种解释，很有可能是对CPU的争用；该系列的第3个和第4个命令——`vmstat` 和 `mpstat`——可以帮助我们进一步确定问题所在。

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, file-rss:0kB
[2320864.954447] TCP: Possible SYN flooding on port 7001. Dropping request. Check SNMP counters.
```

这个命令显示了最新的10个系统信息，如果有的话。注意会导致性能问题的错误信息。上面的例子里就包括对过多占用内存的某进程的死刑判决，还有丢弃TCP请求的公告。

不要漏了这一步！检查 `dmesg` 总是值得的。

6小时前



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”的简称，几十年前就已经包括在BSD套件之中，一直以来都是居家常备的工具。它会逐行输出服务器关键数据的统计结果。

通过指定1作为`vmstat`的输入参数，它会输出每一秒内的统计结果。（在我们当前使用的）`vmstat`输出的第一行数据是从启动到现在的平均数据，而不是前一秒的数据。所以我们可以跳过第一行，看看后面几行的情况。

检查下面各列：

r：等待CPU的进程数。该指标能更好地判定CPU是否饱和，因为它不包括I/O。简单地说，**r**值高于CPU数时就意味着饱和。

free：空闲的内存千字节数。如果你数不清有多少位，就说明系统内存是充足的。接下来要讲到的第7个命令，`free -m`，能够更清楚地说明空闲内存的状态。

si, so：Swap-ins和Swap-outs。如果它们不为零，意味着内存已经不足，开始动用交换空间的存粮了。

us, sy, id, wa, st: 它们是所有CPU的使用百分比。它们分别表示user time, system time (处于内核态的时间), idle, wait I/O和steal time (被其它租户, 或者是租户自己的Xen隔离设备驱动域 (isolated driver domain), 所占用的时间)。

通过相加us和sy的百分比, 你可以确定CPU是否处于忙碌状态。一个持续不变的wait I/O意味着瓶颈在硬盘上, 这种情况往往伴随着CPU的空闲, 因为任务都卡在磁盘I/O上了。你可以把wait I/O当作CPU空闲的另一种形式, 它额外给出了CPU空闲的线索。

I/O处理同样会消耗系统时间。一个高于20%的平均系统时间, 往往值得进一步发掘: 也许系统花在I/O的时间太长了。

在上面的例子中, CPU基本把时间花在用户态里面, 意味着跑在上面的应用占用了大部分时间。此外, CPU平均使用率在90%之上。这不一定是个问题; 检查下“r”列, 看看是否饱和了。

6小时前



实验楼管理员 (<https://www.shiyanlou.com/user/8490>) <https://www.shiyanlou.com/vip>

(<https://www.shiyanlou.com/user/8490>)

4. mpstat -P ALL 1

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

07:38:49 PM CPU      %usr   %nice    %sys %iowait    %irq   %soft  %steal  %guest  %gnice   %idle
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
[...]
```

这个命令显示每个CPU的时间使用百分比, 你可以用它来检查CPU是否存在负载不均衡。单个过于忙碌的CPU可能意味着整个应用只有单个线程在工作。

6小时前



实验楼管理员 (<https://www.shiyanlou.com/user/8490>) <https://www.shiyanlou.com/vip>

(<https://www.shiyanlou.com/user/8490>)

5. pidstat 1

```
$ pidstat 1
Linux 3.13.0-49-generic (titancusters-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 看上去就像 top, 不过 top 的输出会覆盖掉之前的输出, 而 pidstat 的输出则添加在之前的输出的后面。这有利于观察数据随时间的变动情况, 也便于把你看到的内容复制粘贴到调查报告中。

上面的例子表明, CPU主要消耗在两个java进程上。%CPU 列是在各个CPU上的使用量的总和; 1591% 意味着java进程消耗了将近16个CPU。

6小时前

<https://www.shiyanlou.com/user/8490>

6. iostat -xz 1

```
$ iostat -xz 1
Linux 3.13.0-49-generic (titancusters-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    kB/s    kB/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

这个命令可以弄清块设备（磁盘）的状况，包括工作负载和处理性能。注意以下各项：

r/s, w/s, kB/s, kB/s: 分别表示每秒设备读次数，写次数，读的KB数，写的KB数。它们描述了磁盘的工作负载。也许性能问题就是由过高的负载所造成的。

await: I/O平均时间，以毫秒作单位。它是应用中I/O处理所实际消耗的时间，因为其中既包括排队用时也包括处理用时。如果它比预期的大，就意味着设备饱和了，或者设备出了问题。

avgqu-sz: 分配给设备的平均请求数。大于1表示设备已经饱和了。（不过有些设备可以并行处理请求，比如由多个磁盘组成的虚拟设备）

%util: 设备使用率。这个值显示了设备每秒内工作时间的百分比，一般都处于高位。低于60%通常是低性能的表现（也可以从await中看出），不过这个得看设备的类型。接近100%通常意味着饱和。

如果某个存储设备是由多个物理磁盘组成的逻辑磁盘设备，100%的使用率可能只是意味着I/O占用

请牢记于心，disk I/O性能低不一定是个问题。应用的I/O往往是异步的（比如预读（read-ahead）和写缓冲（buffering for writes）），所以不一定会被阻塞并遭受延迟。

6小时前

<https://www.shiyanlou.com/user/8490>

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
```

右边的两列显示：

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

它们的值接近于0时，往往导致较高的磁盘I/O（可以通过iostat确认）和糟糕的性能。上面的例子里没有这个问题，每一列都有好几M呢。

比起第一行，`-/+ buffers/cache` 提供的内存使用量会更加准确些。Linux会把暂时用不上的内存用作缓存，一旦应用需要的时候立刻重新分配给它。所以部分被用作缓存的内存其实也算是空闲内存，第二行以此修订了实际的内存使用量。为了解释这一点，甚至有人专门建了个网站：[linuxatemyram \(http://www.linuxatemyram.com/\)](http://www.linuxatemyram.com/)。

如果你在Linux上安装了ZFS，正如我们在一些服务上所做的，这一点会变得更加迷惑，因为ZFS它自己的文件系统缓存不算入 `free -m`。有时系统看上去已经没有多少空闲内存可用了，其实内存都待在ZFS的缓存里呢。

6小时前



实验楼管理员 (<https://www.shiyanlou.com/user/8490>) [👑 \(https://www.shiyanlou.com/vip\)](https://www.shiyanlou.com/vip)

(<https://www.shiyanlou.com/user/8490>)

8. sar -n DEV 1

```
$ sar -n DEV 1
Linux 3.13.0-49-generic (titancusters-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    rxmcst/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    rxmcst/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
```

这个命令可以用于检查网络流量的工作负载：`rxkB/s`和`txkB/s`，以及它是否达到限额了。上面的例子中，`eth0` 接收的流量达到22Mbytes/s，也即176Mbits/sec（限额是1Gbit/sec）

我们用的版本中还提供了 `%ifutil` 作为设备使用率（接收和发送两者中的最大值）的指标。我们也可以用Brendan的`nicstat`计量这个值。一如 `nicstat`，`sar` 显示的这个值不一定是对的，在这个例子里面就没能正常工作（0.00）。

6小时前



实验楼管理员 (<https://www.shiyanlou.com/user/8490>) [👑 \(https://www.shiyanlou.com/vip\)](https://www.shiyanlou.com/vip)

(<https://www.shiyanlou.com/user/8490>)

9. sar -n TCP,ETCP 1

```
$ sar -n TCP,ETCP 1
Linux 3.13.0-49-generic (titancusters-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
```

这个命令显示一些关键TCP指标的汇总。其中包括：

- `active/s`：本地每秒创建的TCP连接数（比如`concept()`创建的）
- `passive/s`：远程每秒创建的TCP连接数（比如`accept()`创建的）
- `retrans/s`：每秒TCP重传次数

主动连接数（`active`）和被动连接数（`passive`）通常可以用来粗略地描述系统负载。可以认为主动连接是对外的，而被动连接是对内的，虽然严格来说不完全是这个样子。（比如，一个从`localhost`到`localhost`的连接）

重传是网络或系统问题的一个信号；它可能是不可靠的网络（比如公网）所造成的，也有可能是服务器已经过载并开始丢包。在上面的例子中，每秒只创建一个新的TCP连接。

6小时前



实验楼管理员 (<https://www.shiyanlou.com/user/8490>) (<https://www.shiyanlou.com/vip>)

(<https://www.shiyanlou.com/user/8490>)

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 titanc1+   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` 命令包括很多我们之前检查过的指标。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。

不能清晰显示数据随时间变动的情况，这是 `top` 的一个缺点。相较而言，`vmstat` 和 `pidstat` 的输出不会覆盖掉之前的结果，因此更适合查看数据随时间的变动情况。另外，如果你不能及时暂停 `top` 的输出（`Ctrl-s` 暂停，`Ctrl-q` 继续），也许某些关键线索会湮灭在新的输出中。

在这之后...

有很多工具和方法论有助于你深入地发掘问题。Brendan在2015年Velocity大会上的Linux Performance Tools tutorial (<http://techblog.netflix.com/2015/08/netflix-at-velocity-2015-linux.html>)中列出超过40个命令，覆盖了观测、基准测试、调优、静态性能调优、分析（profile），和追踪（tracing）多个方面。

作者：spacewander

文章地址：<http://segmentfault.com/a/1190000004104493>

6小时前

登录后才能回答问题哟~

我要提问

标签

Linux (<https://www.shiyanlou.com/questions/?tag=Linux>) Python (<https://www.shiyanlou.com/questions/?tag=Python>)

C/C++ (<https://www.shiyanlou.com/questions/?tag=C/C++>) 实验环境 (<https://www.shiyanlou.com/questions/?tag=实验环境>)

技术分享 (<https://www.shiyanlou.com/questions/?tag=技术分享>) 功能建议 (<https://www.shiyanlou.com/questions/?tag=功能建议>)

课程需求 (<https://www.shiyanlou.com/questions/?tag=课程需求>) Java (<https://www.shiyanlou.com/questions/?tag=Java>)

其他 (<https://www.shiyanlou.com/questions/?tag=其他>) SQL (<https://www.shiyanlou.com/questions/?tag=SQL>)

NodeJS (<https://www.shiyanlou.com/questions/?tag=NodeJS>) Hadoop (<https://www.shiyanlou.com/questions/?tag=Hadoop>)

Web (<https://www.shiyanlou.com/questions/?tag=Web>) 常见问题 (<https://www.shiyanlou.com/questions/?tag=常见问题>)

Shell (<https://www.shiyanlou.com/questions/?tag=Shell>) PHP (<https://www.shiyanlou.com/questions/?tag=PHP>)

Git (<https://www.shiyanlou.com/questions/?tag=Git>) HTML (<https://www.shiyanlou.com/questions/?tag=HTML>)

HTML5 (<https://www.shiyanlou.com/questions/?tag=HTML5>) 信息安全 (<https://www.shiyanlou.com/questions/?tag=信息安全>)

网络 (<https://www.shiyanlou.com/questions/?tag=网络>) GO (<https://www.shiyanlou.com/questions/?tag=GO>)

NoSQL (<https://www.shiyanlou.com/questions/?tag=NoSQL>) Android (<https://www.shiyanlou.com/questions/?tag=Android>)

训练营 (<https://www.shiyanlou.com/questions/?tag=训练营>) Ruby (<https://www.shiyanlou.com/questions/?tag=Ruby>)

Perl (<https://www.shiyanlou.com/questions/?tag=Perl>)

相关问题

Java虚拟机基础知识 (<https://www.shiyanlou.com/questions/3036>)

MySQL之终端（Terminal）管理数据库、数据表、数据的基本操作 (<https://www.shiyanlou.com/questions/3019>)

C++静态库与动态库 (<https://www.shiyanlou.com/questions/3017>)

谈Runtime机制和使用的整体化梳理 (<https://www.shiyanlou.com/questions/3010>)

JavaScript：彻底理解同步、异步和事件循环(Event Loop) (<https://www.shiyanlou.com/questions/3009>)

动手做实验，轻松学IT。

实验楼-通过动手实践的方式学会IT技术。

公司简介 (<https://www.shiyanlou.com/aboutus>) 联系我们 (<https://www.shiyanlou.com/contact>) 常见问题 (<https://www.shiyanlou.com/faq#howtostart>)
我要开课 (<https://www.shiyanlou.com/labs>) 隐私协议 (<https://www.shiyanlou.com/privacy>) 会员条款 (<https://www.shiyanlou.com/terms>)
友情链接 (<https://www.shiyanlou.com/friends>)
站长统计 (http://www.cnzz.com/stat/website.php?web_id=5902315) 蜀ICP备13019762号 (<http://www.miibeian.gov.cn/>)



QQ群



微信



微博
(<http://weibo.com/shiyanlou2013>)