

20 | 案例篇：为什么系统的Swap变高了？（下）

2019-01-04 倪朋飞

Linux性能优化实战

[进入课程 >](#)



讲述：冯永吉

时长 10:10 大小 9.32M



你好，我是倪朋飞。

上一节我们详细学习了 Linux 内存回收，特别是 Swap 的原理，先简单回顾一下。

在内存资源紧张时，Linux 通过直接内存回收和定期扫描的方式，来释放文件页和匿名页，以便把内存分配给更需要的进程使用。

文件页的回收比较容易理解，直接清空缓存，或者把脏数据写回磁盘后，再释放缓存就可以了。

而对不常访问的匿名页，则需要通过 Swap 换出到磁盘中，这样在下次访问的时候，再次从磁盘换入到内存中就可以了。

开启 Swap 后，你可以设置 `/proc/sys/vm/min_free_kbytes`，来调整系统定期回收内存的阈值，也可以设置 `/proc/sys/vm/swappiness`，来调整文件页和匿名页的回收倾向。

那么，当 Swap 使用升高时，要如何定位和分析呢？下面，我们就来看一个磁盘 I/O 的案例，实战分析和演练。

案例

下面案例基于 Ubuntu 18.04，同样适用于其他的 Linux 系统。

机器配置：2 CPU，8GB 内存


你需要预先安装 `sysstat` 等工具，如 `apt install sysstat`

首先，我们打开两个终端，分别 SSH 登录到两台机器上，并安装上面提到的这些工具。

同以前的案例一样，接下来的所有命令都默认以 root 用户运行，如果你是用普通用户身份登陆系统，请运行 `sudo su root` 命令切换到 root 用户。

如果安装过程中有什么问题，同样鼓励你先自己搜索解决，解决不了的，可以在留言区向我提问。

然后，在终端中运行 `free` 命令，查看 Swap 的使用情况。比如，在我的机器中，输出如下：


 复制代码

```
1 $ free
2              total        used        free      shared  buff/cache   available
3 Mem:        8169348       331668       6715972          696       1121708       7522896
4 Swap:           0           0           0
```

从这个 `free` 输出你可以看到，Swap 的大小是 0，这说明我的机器没有配置 Swap。

为了继续 Swap 的案例，就需要先配置、开启 Swap。如果你的环境中已经开启了 Swap，那你可以略过下面的开启步骤，继续往后走。

要开启 Swap，我们首先要清楚，Linux 本身支持两种类型的 Swap，即 Swap 分区和 Swap 文件。以 Swap 文件为例，在第一个终端中运行下面的命令开启 Swap，我这里配置 Swap 文件的大小为 8GB：

 复制代码

```
1 # 创建 Swap 文件
2 $ fallocate -l 8G /mnt/swapfile
3 # 修改权限只有根用户可以访问
4 $ chmod 600 /mnt/swapfile
5 # 配置 Swap 文件
6 $ mkswap /mnt/swapfile
7 # 开启 Swap
8 $ swapon /mnt/swapfile
```

然后，再执行 free 命令，确认 Swap 配置成功：

 复制代码

```
1 $ free
2      total        used        free      shared  buff/cache   available
3 Mem:    8169348    331668    6715972        696    1121708    7522896
4 Swap:    8388604         0    8388604
```

现在，free 输出中，Swap 空间以及剩余空间都从 0 变成了 8GB，说明 Swap 已经**正常开启**。

接下来，我们在第一个终端中，运行下面的 dd 命令，模拟大文件的读取：

 复制代码

```
1 # 写入空设备，实际上只有磁盘的读请求
2 $ dd if=/dev/sda1 of=/dev/null bs=1G count=2048
```

接着，在第二个终端中运行 sar 命令，查看内存各个指标的变化情况。你可以多观察一会儿，查看这些指标的变化情况。

```

1 # 间隔 1 秒输出一组数据
2 # -r 表示显示内存使用情况，-S 表示显示 Swap 使用情况
3 $ sar -r -S 1
4 04:39:56      kbmemfree      kbavail  kbmemused  %memused  kbbuffers  kbcached  kbcommit  %cor
5 04:39:57      6249676      6839824   1919632    23.50     740512     67316   1691736    16
6
7 04:39:56      kbswpfree  kbswpused  %swpused  kswpcad   %swpcad
8 04:39:57      8388604           0         0.00         0         0.00
9
10 04:39:57      kbmemfree      kbavail  kbmemused  %memused  kbbuffers  kbcached  kbcommit  %cor
11 04:39:58      6184472      6807064   1984836    24.30     772768     67380   1691736    16
12
13 04:39:57      kbswpfree  kbswpused  %swpused  kswpcad   %swpcad
14 04:39:58      8388604           0         0.00         0         0.00
15
16 ...
17
18
19 04:44:06      kbmemfree      kbavail  kbmemused  %memused  kbbuffers  kbcached  kbcommit  %cor
20 04:44:07      152780      6525716   8016528    98.13    6530440     51316   1691736    16
21
22 04:44:06      kbswpfree  kbswpused  %swpused  kswpcad   %swpcad
23 04:44:07      8384508      4096         0.05         52         1.27

```

我们可以看到，sar 的输出结果是两个表格，第一个表格表示内存的使用情况，第二个表格表示 Swap 的使用情况。其中，各个指标名称前面的 kb 前缀，表示这些指标的单位是 KB。

去掉前缀后，你会发现，大部分指标我们都已经见过了，剩下的几个新出现的指标，我来简单介绍一下。

kbcommit，表示当前系统负载需要的内存。它实际上是为了保证系统内存不溢出，对需要内存的估计值。**%commit**，就是这个值相对总内存的百分比。

kbactive，表示活跃内存，也就是最近使用过的内存，一般不会被系统回收。

kbinact，表示非活跃内存，也就是不常访问的内存，有可能会被系统回收。

清楚了界面指标的含义后，我们再结合具体数值，来分析相关的现象。你可以清楚地看到，总的内存使用率（**%memused**）在不断增长，从开始的 23% 一直长到了 98%，并且主要内存都被缓冲区（**kbbuffers**）占用。具体来说：


刚开始，剩余内存（kmemfree）不断减少，而缓冲区（kbbuffers）则不断增大，由此可知，剩余内存不断分配给了缓冲区。

一段时间后，剩余内存已经很小，而缓冲区占用了大部分内存。这时候，Swap 的使用开始逐渐增大，缓冲区和剩余内存则只在小范围内波动。

你可能困惑了，为什么缓冲区在不停增大？这又是哪些进程导致的呢？

显然，我们还得看看进程缓存的情况。在前面缓存的案例中我们学过，cachetop 正好能满足这一点。那我们就来 cachetop 一下。

在第二个终端中，按下 Ctrl+C 停止 sar 命令，然后运行下面的 cachetop 命令，观察缓存的使用情况：

 复制代码


```
1 $ cachetop 5
2 12:28:28 Buffers MB: 6349 / Cached MB: 87 / Sort: HITS / Order: ascending
3 PID      UID      CMD           HITS      MISSES    DIRTIES  READ_HIT%  WRITE_HIT%
4   18280  root     python         22         0         0       100.0%     0.0%
5   18279  root      dd          41088      41022         0        50.0%     50.0%
```

通过 cachetop 的输出，我们看到，dd 进程的读写请求只有 50% 的命中率，并且未命中的缓存页数（MISSES）为 41022（单位是页）。这说明，正是案例开始时运行的 dd，导致了缓冲区使用升高。

你可能接着会问，为什么 Swap 也跟着升高了呢？直观来说，缓冲区占了系统绝大部分内存，还属于可回收内存，内存不够用时，不应该先回收缓冲区吗？

这种情况，我们还得进一步通过 /proc/zoneinfo，观察剩余内存、内存阈值以及匿名页和文件页的活跃情况。

你可以在第二个终端中，按下 Ctrl+C，停止 cachetop 命令。然后运行下面的命令，观察 /proc/zoneinfo 中这几个指标的变化情况：

 复制代码

1 # -d 表示高亮变化的字段

```
2 # -A 表示仅显示 Normal 行以及之后的 15 行输出
3 $ watch -d grep -A 15 'Normal' /proc/zoneinfo
4 Node 0, zone      Normal
5   pages free      21328
6       min         14896
7       low         18620
8       high        22344
9       spanned     1835008
10      present     1835008
11      managed     1796710
12      protection: (0, 0, 0, 0, 0)
13      nr_free_pages 21328
14      nr_zone_inactive_anon 79776
15      nr_zone_active_anon 206854
16      nr_zone_inactive_file 918561
17      nr_zone_active_file 496695
18      nr_zone_unevictable 2251
19      nr_zone_write_pending 0
```

你可以发现，剩余内存（pages_free）在一个小范围内不停地波动。当它小于页低阈值（pages_low）时，又会突然增大到一个大于页高阈值（pages_high）的值。

再结合刚刚用 sar 看到的剩余内存和缓冲区的变化情况，我们可以推导出，剩余内存和缓冲区的波动变化，正是由于内存回收和缓存再次分配的循环往复。

当剩余内存小于页低阈值时，系统会回收一些缓存和匿名内存，使剩余内存增大。其中，缓存的回收导致 sar 中的缓冲区减小，而匿名内存的回收导致了 Swap 的使用增大。


紧接着，由于 dd 还在继续，剩余内存又会重新分配给缓存，导致剩余内存减少，缓冲区增大。

其实还有一个有趣的现象，如果多次运行 dd 和 sar，你可能会发现，在多次的循环重复中，有时候是 Swap 用得比较多，有时候 Swap 很少，反而缓冲区的波动更大。

换句话说，系统回收内存时，有时候会回收更多的文件页，有时候又回收了更多的匿名页。

显然，系统回收不同类型内存的倾向，似乎不那么明显。你应该想到了上节课提到的 swappiness，正是调整不同类型内存回收的配置选项。

还是在第二个终端中，按下 Ctrl+C 停止 watch 命令，然后运行下面的命令，查看 swappiness 的配置：

 复制代码


```
1 $ cat /proc/sys/vm/swappiness
2 60
```

swappiness 显示的是默认值 60，这是一个相对中和的配置，所以系统会根据实际运行情况，选择合适的回收类型，比如回收不活跃的匿名页，或者不活跃的文件页。

到这里，我们已经找出了 Swap 发生的根源。另一个问题就是，刚才的 Swap 到底影响了哪些应用程序呢？换句话说，Swap 换出的是哪些进程的内存？

这里我还是推荐 proc 文件系统，用来查看进程 Swap 换出的虚拟内存大小，它保存在 /proc/pid/status 中的 VmSwap 中（推荐你执行 man proc 来查询其他字段的含义）。

在第二个终端中运行下面的命令，就可以查看使用 Swap 最多的进程。注意 for、awk、sort 都是最常用的 Linux 命令，如果你还不熟悉，可以用 man 来查询它们的手册，或上网搜索教程来学习。


 复制代码

```
1 # 按 VmSwap 使用量对进程排序，输出进程名称、进程 ID 以及 SWAP 用量
2 $ for file in /proc/*/status ; do awk '/VmSwap|Name|^Pid/{printf $2 " " $3}END{ print "
3 dockerd 2226 10728 kB
4 docker-containe 2251 8516 kB
5 snapd 936 4020 kB
6 networkd-dispat 911 836 kB
7 polkitd 1004 44 kB
```

从这里你可以看到，使用 Swap 比较多的是 dockerd 和 docker-containe 进程，所以，当 dockerd 再次访问这些换出到磁盘的内存时，也会比较慢。


这也说明了一点，虽然缓存属于可回收内存，但在类似大文件拷贝这类场景下，系统还是会用 Swap 机制来回收匿名内存，而不仅仅是回收占用绝大部分内存的文件页。

最后，如果你在一开始配置了 Swap，不要忘记在案例结束后关闭。你可以运行下面的命令，关闭 Swap：

 复制代码

```
1 $ swapoff -a
```

实际上，关闭 Swap 后再重新打开，也是一种常用的 Swap 空间清理方法，比如：

 复制代码

```
1 $ swapoff -a && swapon -a
```

小结

在内存资源紧张时，Linux 会通过 Swap，把不常访问的匿名页换出到磁盘中，下次访问的时候再从磁盘换入到内存中来。你可以设置 `/proc/sys/vm/min_free_kbytes`，来调整系统定期回收内存的阈值；也可以设置 `/proc/sys/vm/swappiness`，来调整文件页和匿名页的回收倾向。

当 Swap 变高时，你可以用 `sar`、`/proc/zoneinfo`、`/proc/pid/status` 等方法，查看系统和进程的内存使用情况，进而找出 Swap 升高的根源和受影响的进程。

反过来说，通常，降低 Swap 的使用，可以提高系统的整体性能。要怎么做呢？这里，我也总结了几种常见的降低方法。

禁止 Swap，现在服务器的内存足够大，所以除非有必要，禁用 Swap 就可以了。随着云计算的普及，大部分云平台中的虚拟机都默认禁止 Swap。

如果实在需要用到 Swap，可以尝试降低 `swappiness` 的值，减少内存回收时 Swap 的使用倾向。

响应延迟敏感的应用，如果它们可能在开启 Swap 的服务器中运行，你还可以用库函数 `mlock()` 或者 `mlockall()` 锁定内存，阻止它们的内存换出。

思考

最后，给你留一个思考题。

今天的案例中，swappiness 使用的是默认配置的 60。如果把它配置成 0 的话，还会发生 Swap 吗？这又是为什么呢？

希望你可以实际操作一下，重点观察 sar 的输出，并结合今天的内容来记录、总结。

欢迎留言和我讨论，也欢迎把这篇文章分享给你的同事、朋友。我们一起在实战中演练，在交流中进步。

 极客时间

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，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 19 | 案例篇：为什么系统的Swap变高了（上）

下一篇 21 | 套路篇：如何“快准狠”找到系统内存的问题？

精选留言 (26)

 写留言

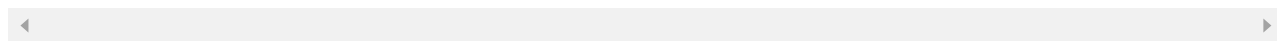


Free_fish
2019-01-04

 17

用smem --sort swap命令可以直接将进程按照swap使用量排序显示

作者回复: 谢谢分享



Geek_2b680...

2019-01-04

👍 9

希望老师在用工具的时候能够使用对内核版本要求不高的，毕竟生产环境用较新内核的还是比较少



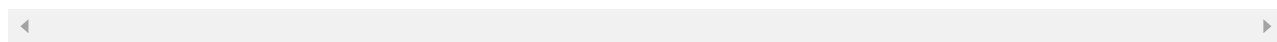
Scott

2019-01-04

👍 5

答案不是上一讲有提到吗，就算设置为0，如果空闲内存+文件页 < page_low，还是会发生swap，这个值是设置swap的积极程度，就算最不积极，被逼无奈还是得swap的。

作者回复: 是的



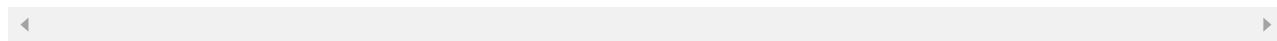
尘封

2019-01-04

👍 5

\$ swapoff -a && swapon -a，线上使用这个命令释放swap有什么风险吗？

作者回复: 有的，使用了swap说明内存可能有压力了，这么强制换入有可能导致内存问题



流转千回

2019-01-04

👍 2

打卡，老师这么晚还在更新专栏，致敬！

展开 ▾



夜空中最亮...

2019-01-04

👍 1

swappiness 使用的是默认配置的 60。如果把它配置为0 还会使用 swap 吗？
会使用的，为0 是最大限度不用，不是一定不用。

展开 ∨



ninuxer

2019-01-04

👍 1

打卡day21

按我的理解，swappiness只是几率，并不意味着一定，所以还是会发生匿名页交换，只是几率小点

展开 ∨



Adam

2019-03-06

👍

ES 可以通过设置锁定内存。

展开 ∨



如果

2019-01-31

👍

DAY20，打卡

展开 ∨



zshanjun

2019-01-15

👍

大文件读取瞬间完成，没有观察到内存的变化：

```
root@linux-1:~/go/bin# dd if=/dev/sda1 of=/dev/null bs=16G count=2048
```

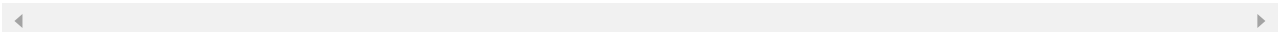
0+1 records in

0+1 records out

1048576 bytes (1.0 MB, 1.0 MiB) copied, 0.00693471 s, 151 MB/s

展开 ∨

作者回复: 仔细看看，1048576 bytes (1.0 MB, 1.0 MiB) copied



berryfl

👍



2019-01-10

/proc/*/status会匹配到self之类的特殊目录，把匹配的部分写复杂点可以限制到仅数字，例如+([0-9])

作者回复: 嗯，是的



仲鬼

2019-01-09



老师好，我研究man sar后还是没理解kbcommit，这个估计值具体指什么呢？就算单纯以不导致OOM的最小内存理解，为什么会小于kbmemused（已用物理内存）呢？我认为应该是kbcommit >= kbmemused。

展开 ∨

作者回复: kbcommit就是进程申请的内存之和，kbmemused还包括了buffer和cache



白华

2019-01-07



我写的dd参数就是大文件系统的数据，一共就20g存储的虚拟机，传的是/dev/root文件系统 有10g 需要写好久，实验现象和您描述的差不多，但是swap就是没变化

展开 ∨

作者回复: swap开了吗？swappiness配置的什么



AA

2019-01-07



请问min_free_kbytes一般设置多少？

展开 ∨

作者回复: 用系统默认值就可以，除非发现问题才建议调整





AA

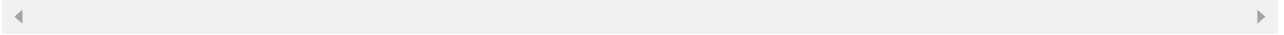
2019-01-07



请教下min_free_kbytes一般设置多少？

展开 ▾

作者回复: 系统默认值就可以



dexter

2019-01-06



当前全部进程VmSwap的使用量的总和是不是就是free里面swap中的used的使用量？



腾达

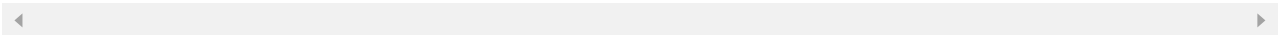
2019-01-06



第一次启动虚拟机按照步骤操作，swpused发现高起来好多，后来几次不知道为什么swpused高不起来了，始终维持2%上下。第一次和后面几次的差异就是第一次运行后，去安装了smem，然后重启了机器几次，系统的默认参数都没改过

展开 ▾

作者回复: 试试调整 dd 参数，可能是你的内存大得多，可以多读一些数据



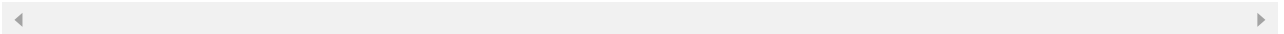
白华

2019-01-06



在centos系统操作还是会有区别，我的swap分的2G，但是一点都没有用到，kbswpfree总是不变，即使memused达到了98%以上，就看到了kbswpfree的波动。

作者回复: 详细的现象是什么？试试调大dd的参数读更多的数据？



风飘，吾独...

2019-01-06



打卡

展开 ▾



往事随风，...

2019-01-05



现在变化不大，按照比例操作

展开 ▾