01 基础篇（一）| 如何用数据观测Page Cache？

2020-08-17 邵亚方

Linux内核技术实战课 进入课程 >

讲述：邵亚方
时长 10:19　大小 9.46M

你好，我是邵亚方。今天我想和你聊一聊 Page Cache 的话题。

Page Cache 你应该不陌生了，如果你是一名应用开发者或者 Linux 运维人员，那么在工作中，你可能遇见过与 Page Cache 有关的场景，比如：

服务器的 load 飙高；

服务器的 I/O 吞吐飙高；

业务响应时延出现大的毛刺；

业务平均访问时延明显增加。

这些问题，很可能是由于 Page Cache 管理不到位引起的，因为 Page Cache 管理不当除了会增加系统 I/O 吞吐外，还会引起业务性能抖动，我在生产环境上处理过很多这类问题。

据我观察，这类问题出现后，业务开发人员以及运维人员往往会束手无策，究其原因在于他们对 Page Cache 的理解仅仅停留在概念上，并不清楚 Page Cache 如何和应用、系统关联起来，对它引发的问题自然会束手无策了。所以，要想不再踩 Page Cache 的坑，你必须对它有个清晰的认识。

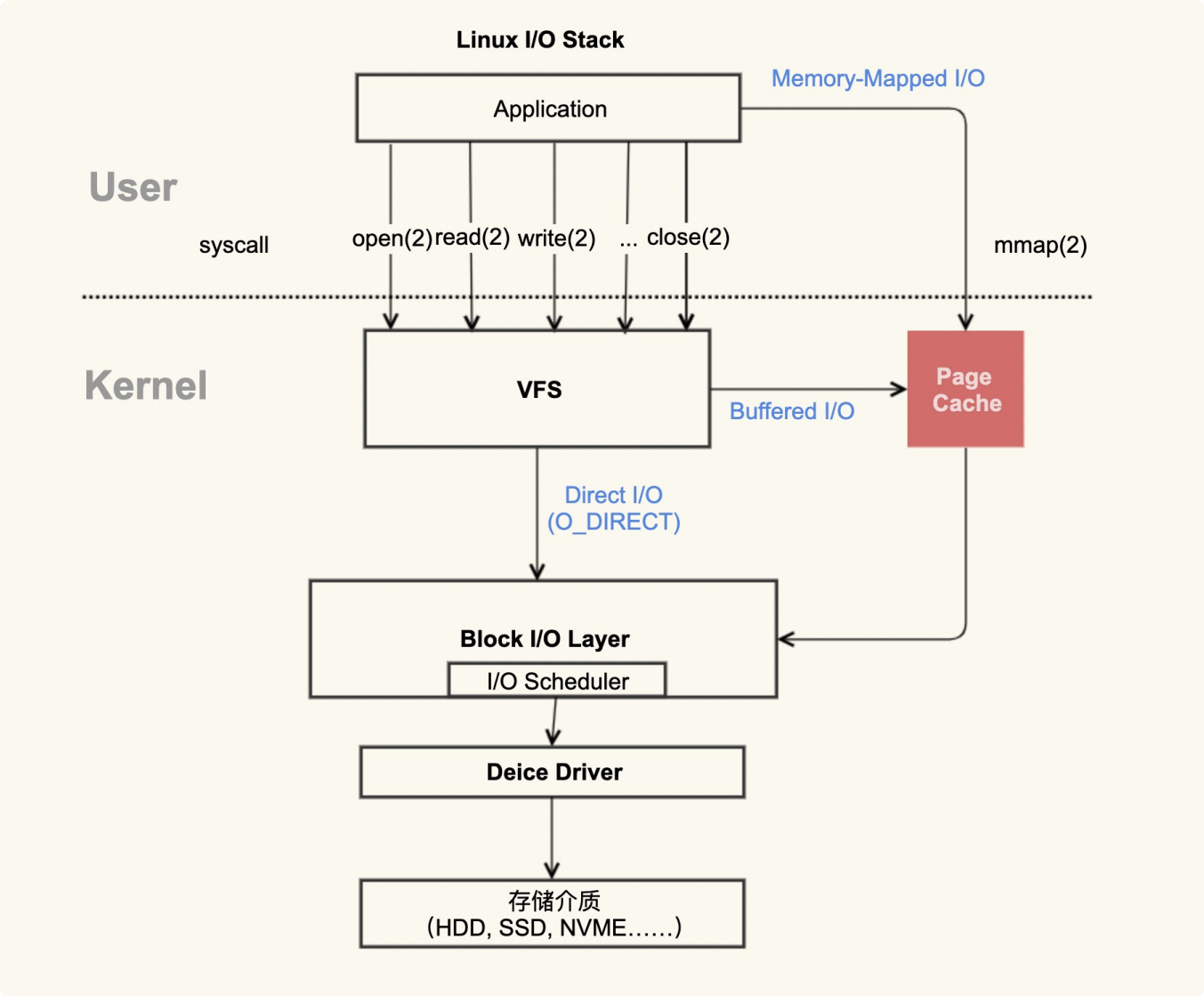
那么在我看来，认识 Page Cache 最简单的方式，就是用数据说话，通过具体的数据你会更加深入地理解 Page Cache 的本质。为了帮你消化和理解，我会用两节课的时间，用数据剖析什么是 Page Cache，为什么需要 Page Cache，Page Cache 的产生和回收是什么样的。这样一来，你会从本质到表象，透彻理解它，深切感受它和你的应用程序之间的关系，从而能更好地理解上面提到的四个问题。

不过，在这里我想给你提个醒，要学习今天的内容，你最好具备一些 Linux 编程的基础，比如，如何打开一个文件；如何读写一个文件；如何关闭一个文件等等。这样，你理解今天的内容会更加容易，当然了，不具备也没有关系，如果遇到你实在看不懂的地方，你可以查阅《UNIX 环境高级编程》这本书，它是每一位 Linux 开发者以及运维人员必看的入门书籍。

好了，话不多说，我们进入今天的学习。

什么是 Page Cache？

我记得很多应用开发者或者运维在向我寻求帮助，解决 Page Cache 引起的问题时，总是喜欢问我 Page Cache 到底是属于内核还是属于用户？针对这样的问题，我一般会让他们先看下面这张图：



应用程序产生Page Cache的逻辑示意图

通过这张图片你可以清楚地看到，红色的地方就是 Page Cache，**很明显，Page Cache 是内核管理的内存，也就是说，它属于内核不属于用户。**

那咱们怎么来观察 Page Cache 呢？其实，在 Linux 上直接查看 Page Cache 的方式有很多，包括 /proc/meminfo、free 、/proc/vmstat 命令等，它们的内容其实是一致的。

我们拿 /proc/meminfo 命令举例看一下（如果你想了解 /proc/meminfo 中每一项具体含义的话，可以去看Kernel Documentation的 meminfo 这一节，它详细解释了每一项的具体含义，Kernel Documentation 是应用开发者想要了解内核最简单、直接的方式）。

复制代码

```
$ cat /proc/meminfo
...
Buffers:            1224 kB
Cached:           111472 kB

```

```
SwapCached:         36364 kB
Active:           6224232 kB
Inactive:          979432 kB
Active(anon):     6173036 kB
Inactive(anon):    927932 kB
Active(file):       51196 kB
Inactive(file):     51500 kB
...
Shmem:              10000 kB
...
SReclaimable:       43532 kB
```

根据上面的数据，你可以简单得出这样的公式（等式两边之和都是 112696 KB）：

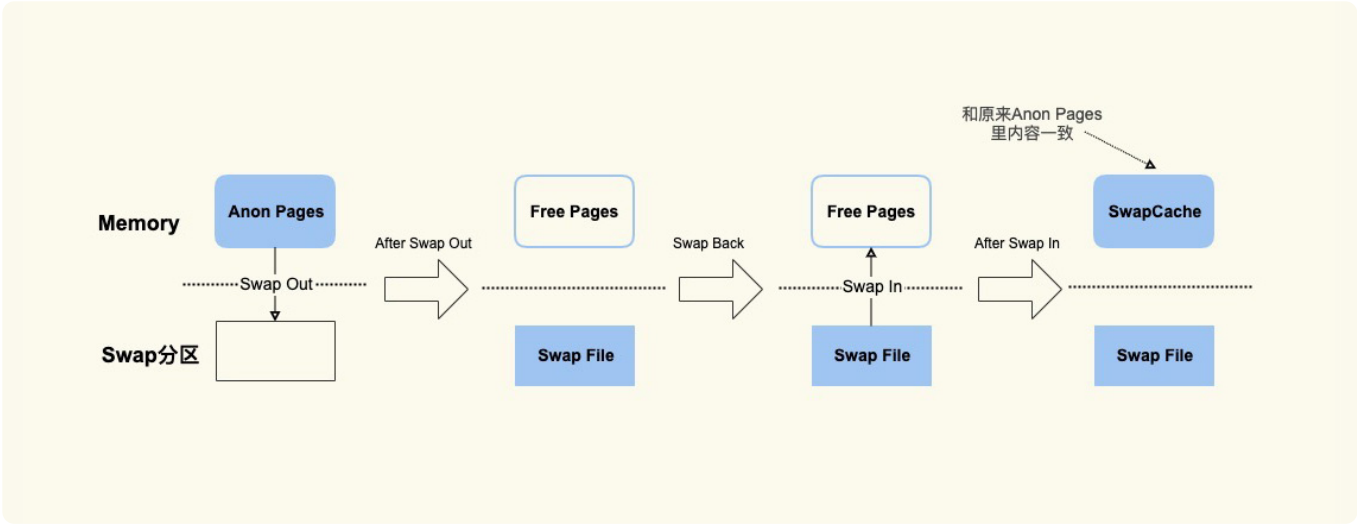
> Buffers + Cached + SwapCached = Active(file) + Inactive(file) + Shmem + SwapCached

那么等式两边的内容就是我们平时说的 Page Cache。请注意你没有看错，两边都有 SwapCached，之所以要把它放在等式里，就是说它也是 Page Cache 的一部分。

接下来，我带你分析一下这些项的具体含义。等式右边这些项把 Buffers 和 Cached 做了一下细分，分为了 Active(file)，Inactive(file) 和 Shmem，因为 Buffers 更加依赖于内核实现，在不同内核版本中它的含义可能有些不一致，而等式右边和应用程序的关系更加直接，所以我们从等式右边来分析。

在 Page Cache 中，Active(file)+Inactive(file) 是 File-backed page（与文件对应的内存页），是你最需要关注的部分。因为你平时用的 mmap() 内存映射方式和 buffered I/O 来消耗的内存就属于这部分，**最重要的是，这部分在真实的生产环境上也最容易产生问题，**我们在接下来的课程案例篇会重点分析它。

而 SwapCached 是在打开了 Swap 分区后，把 Inactive(anon)+Active(anon) 这两项里的匿名页给交换到磁盘（swap out），然后再读入到内存（swap in）后分配的内存。**由于读入到内存后原来的 Swap File 还在，所以 SwapCached 也可以认为是 File-backed page，即属于 Page Cache。**这样做的目的也是为了减少 I/O。你是不是觉得这个过程有些复杂？我们用一张图直观地看一下：



我希望你能通过这个简单的示意图明白 SwapCached 是怎么产生的。在这个过程中你要注意，SwapCached 只在 Swap 分区打开的情况下才会有，而我建议你在生产环境中关闭 Swap 分区，因为 Swap 过程产生的 I/O 会很容易引起性能抖动。

除了 SwapCached，Page Cache 中的 Shmem 是指匿名共享映射这种方式分配的内存（free 命令中 shared 这一项），比如 tmpfs（临时文件系统），这部分在真实的生产环境中产生的问题比较少，不是我们今天的重点内容，我们这节课不对它做过多关注，你知道有这回事就可以了。

当然了，很多同学也喜欢用 free 命令来查看系统中有多少 Page Cache，会根据 buff/cache 来判断存在多少 Page Cache。如果你对 free 命令有所了解的话，肯定知道 free 命令也是通过解析 /proc/meminfo 得出这些统计数据的，这些都可以通过 free 工具的源码来找到。free 命令的源码是开源，你可以去看下 procfs里的 free.c 文件，源码是最直接的理解方式，它会加深你对 free 命令的理解。

不过你是否好奇过，free 命令中的 buff/cache 究竟是指什么呢？我们在这里先简单地看一下：

复制代码

```
$ free -k
              total        used        free      shared  buff/cache   availabl
Mem:        7926580     7277960      492392       10000      156228      43068
Swap:       8224764      380748     7844016
```

通过 procfs 源码里面的proc/sysinfo.c这个文件，你可以发现 buff/cache 包括下面这几项：

> buff/cache = Buffers + Cached + SReclaimable

通过前面的数据我们也可以验证这个公式: 1224 + 111472 + 43532 的和是 156228。

另外，这里你要注意，你在做比较的过程中，一定要考虑到这些数据是动态变化的，而且执行命令本身也会带来内存开销，所以这个等式未必会严格相等，不过你不必怀疑它的正确性。

从这个公式中，你能看到 free 命令中的 buff/cache 是由 Buffers、Cached 和 SReclaimable 这三项组成的，它强调的是内存的可回收性，也就是说，可以被回收的内存会统计在这一项。

其中 SReclaimable 是指可以被回收的内核内存，包括 dentry 和 inode 等。而这部分内容是内核非常细节性的东西，对于应用开发者和运维人员理解起来相对有些难度，所以我们在这里不多说。

掌握了 Page Cache 具体由哪些部分构成之后，在它引发一些问题时，你就能够知道需要去观察什么。比如说，应用本身消耗内存（RSS）不多的情况下，整个系统的内存使用率还是很高，那不妨去排查下是不是 Shmem(共享内存) 消耗了太多内存导致的。

讲到这儿，我想你应该对 Page Cache 有了一些直观的认识了吧？当然了，有的人可能会说，内核的 Page Cache 这么复杂，我不要不可以么？

我相信有这样想法的人不在少数，如果不用内核管理的 Page Cache，那有两种思路来进行处理：

> 第一种，应用程序维护自己的 Cache 做更加细粒度的控制，比如 MySQL 就是这样做的，你可以参考MySQL Buffer Pool ，它的实现复杂度还是很高的。对于大多数应用而言，实现自己的 Cache 成本还是挺高的，不如内核的 Page Cache 来得简单高效。
>
> 第二种，直接使用 Direct I/O 来绕过 Page Cache，不使用 Cache 了，省的去管它了。这种方法可行么？那我们继续用数据说话，看看这种做法的问题在哪儿？

为什么需要 Page Cache？

通过第一张图你其实已经可以直观地看到，标准 I/O 和内存映射会先把数据写入到 Page Cache，这样做会通过减少 I/O 次数来提升读写效率。我们看一个具体的例子。首先，我们来生成一个 1G 大小的新文件，然后把 Page Cache 清空，确保文件内容不在内存中，以此来比较第一次读文件和第二次读文件耗时的差异。具体的流程如下。

先生成一个 1G 的文件：

> $ddif = /dev/zeroof = /home/yafang/test/dd.outbs = 4096count =$ ((1024*256))

其次，清空 Page Cache，需要先执行一下 sync 来将脏页（第二节课，我会解释一下什么是脏页）同步到磁盘再去 drop cache。

> $ sync && echo 3 > /proc/sys/vm/drop_caches

第一次读取文件的耗时如下：

复制代码

```
$ time cat /home/yafang/test/dd.out &> /dev/null
real  0m5.733s
user  0m0.003s
sys  0m0.213s
```

再次读取文件的耗时如下：

复制代码

```
$ time cat /home/yafang/test/dd.out &> /dev/null
real  0m0.132s
user  0m0.001s
sys  0m0.130s
```

通过这样详细的过程你可以看到，第二次读取文件的耗时远小于第一次的耗时，这是因为第一次是从磁盘来读取的内容，磁盘 I/O 是比较耗时的，而第二次读取的时候由于文件内

容已经在第一次读取时被读到内存了，所以是直接从内存读取的数据，内存相比磁盘速度是快很多的。**这就是 Page Cache 存在的意义：减少 I/O，提升应用的 I/O 速度。**

所以，如果你不想为了很细致地管理内存而增加应用程序的复杂度，那你还是乖乖使用内核管理的 Page Cache 吧，它是 ROI(投入产出比) 相对较高的一个方案。

你要知道，我们在做方案抉择时找到一个各方面都很完美的方案还是比较难的，大多数情况下都是经过权衡后来选择一个合适的方案。因为，我一直坚信，合适的就是最好的。

而我之所以说 Page Cache 是合适的，而不是说它是最好的，那是因为 Page Cache 的不足之处也是有的，这个不足之处主要体现在，它对应用程序太过于透明，以至于应用程序很难有好方法来控制它。

为什么这么说呢？要想知道这个答案，你就需要了解 Page Cache 的产生过程，这里卖个关子，我在下一讲会跟你讨论。

课堂总结

我们这节课主要是讲述了如何很好地理解 Page Cache，在我看来，要想很好的理解它，直观的方式就是从数据入手，所以，我从如何观测 Page Cache 出发来带你认识什么是 Page Cache；然后再从它为什么容易产生问题出发，带你回顾了它存在的意义，我希望通过这样的方式，帮你明确这样几个要点：

1. Page Cache 是属于内核的，不属于用户。
2. Page Cache 对应用提升 I/O 效率而言是一个投入产出比较高的方案，所以它的存在还是有必要的。

在我看来，如何管理好 Page Cache，最主要的是你要知道如何来观测它以及观测关于它的一些行为，有了这些数据做支撑，你才能够把它和你的业务更好地结合起来。而且，在我看来，当你对某一个概念很模糊、搞不清楚它到底是什么时，最好的认知方式就是先搞明白如何来观测它，然后动手去观测看看它究竟是如何变化的，正所谓纸上得来终觉浅，绝知此事要躬行！

这节课就讲到这里，下一节我们使用数据来观察 Page Cache 的产生和释放，这样一来，你就能了解 Page Cache 的整个生命周期，从而对于它引发的一些问题能有一个大概的判断。

课后作业

最后我给你留一道思考题，请你写一个程序来构造出来 Page Cache，然后观察 /proc/meminfo 和 /proc/vmstat 里面的数据是如何变化的， 欢迎在留言区分享你的看法。

感谢你的阅读，如果你认为这节课的内容有收获，也欢迎把它分享给你的朋友，我们下一讲见。

提建议

上一篇 开篇词 | 如何让Linux内核更好地服务应用程序？

下一篇 02 基础篇（二）| Page Cache是怎样产生和释放的？

精选留言

写留言

> 由作者筛选后的优质留言将会公开显示，欢迎踊跃留言。