

【译】替 swap 辩护：常见的误解



目录

目录

- 背景
 - 内存的类型
 - 可回收/不可回收内存
- 说到交换区的本质

- 考察有无交换区时会发生什么
 - 在无/低内存竞争的状态下
 - 在中/高内存竞争的状态下
 - 在临时内存占用高峰时
 - 好吧，所以我需要系统交换区，但是我该怎么为每个程序微调它的行为？
- 调优
 - 那么，我需要多少交换空间？
 - 我的 swappiness 应该如何设置？
 - 2019年07月更新：内核 4.20+ 中的内存压力指标
- 结论

这篇翻译自 Chris Down 的博文 [In defence of swap: common misconceptions](#)。原文的协议是 CC BY-SA 4.0，本文翻译同样也使用 CC BY-SA 4.0。其中加入了一些我自己的理解作为旁注，所有译注都在侧边栏中。

翻译这篇文章是因为经常看到朋友们（包括有经验的程序员和 Linux 管理员）对 swap 和 swappiness 有诸多误解，而这篇文章正好澄清了这些误解，也讲清楚了 Linux 中这两者的本质。值得一提的是本文讨论的 swap 针对 Linux 内核，在别的系统包括 macOS/WinNT 或者 Unix 系统中的交换文件可能有不同一样的行为，需要不同的调优方式。比如在 [FreeBSD handbook](#) 中明确建议了 swap 分区通常应该是两倍物理内存大小，这一点建议对 FreeBSD 系内核的内存管理可能非常合理，而不一

定适合 Linux 内核，FreeBSD 和 Linux 有不同的内存管理方式尤其是 swap 和 page cache 和 buffer cache 的处理方式有诸多不同。

经常有朋友看到系统卡顿之后看系统内存使用状况观察到大量 swap 占用，于是觉得卡顿是来源于 swap。就像文中所述，相关不蕴含因果，产生内存颠簸之后的确会造成大量 swap 占用，也会造成系统卡顿，但是 swap 不是导致卡顿的原因，关掉 swap 或者调低 swappiness 并不能阻止卡顿，只会将 swap 造成的 I/O 转化为加载文件缓存造成的 I/O。

以下是原文翻译：

这篇文章也有 [日文](#) 和 [俄文](#) 翻译。

太长不看：




1. 对维持系统的正常功能而言，有 swap 是相对挺重要的一部分。没有它的话会更难做到合理的内存管理。
2. swap 的目的通常并不是用作紧急内存，它的目的在于让内存回收能更平等和高效。事实上把它当作「紧急内存」来用的想法通常是有害的。
3. 禁用 swap 在内存压力下并不能避免磁盘 I/O 造成的性能问题，这么做只是让磁盘 I/O 颠簸的范围从匿名页面转化到文件页面。这不仅更低效，因为系统能回收的页面的选择范围更有限了，而且这种做法还可能是加重了内存压力的原因之一。

4. 内核 4.0 版本之前的交换进程 (swapper) 有一些问题，导致很多人对 swap 有负面印象，因为它太急于 (overeagerness) 把页面交换出去。在 4.0 之后的内核上这种情况已经改善了很多。
5. 在 SSD 上，交换出匿名页面的开销和回收文件页面的开销基本上在性能/延迟方面没有区别。在老式的磁盘上，读取交换文件因为属于随机访问读取所以会更慢，于是设置较低的 `vm.swappiness` 可能比较合理（继续读下面关于 `vm.swappiness` 的描述）。
6. 禁用 swap 并不能避免在接近 OOM 状态下最终表现出的症状，尽管的确有 swap 的情况下这种症状持续的时间可能会延长。在系统调用 OOM 杀手的时候无论有没有启用 swap，或者更早/更晚开始调用 OOM 杀手，结果都是一样的：整个系统留在了一种不可预知的状态下。有 swap 也不能避免这一点。
7. 可以用 cgroup v2 的 `memory.low` 相关机制来改善内存压力下 swap 的行为并且避免发生颠簸。


我的工作的一部分是改善内核中内存管理和 cgroup v2 相关，所以我和很多工程师讨论过看待内存管理的态度，尤其是在压力下应用程序的行为和操作系统在底层内存管理中用的基于经验的启发式决策逻辑。




在这种讨论中经常重复的话题是交换区（swap）。交换区的话题是非常有争议而且很少被理解的话题，甚至包括那些在 Linux 上工作过多年的人也是如此。很多人觉得它没什么用甚至是有害的：它是历史遗迹，从内存紧缺而磁盘读写是必要之恶的时代遗留到现在，为计算机提供在当年很必要的页面交换功能作为内存空间。最近几年我还经常能以一定频度看到这种论调，然后我和很多同事、朋友、业界同行们讨论过很多次，帮他们理解为什么在现代计算机系统中交换区仍是有用的概念，即便现在的电脑中物理内存已经远多于过去。

围绕交换区的目的还有很多误解——很多人觉得  它只是某种为了应对紧急情况的「慢速额外内存」，但是没能理解在整个操作系统健康运作的时候它也能改善普通负载的性能。


我们很多人也听说过描述内存时所用的常见说法：「Linux 用了太多内存」，「swap 应该设为物理内存的两倍大小」，或者类似的说法。虽然这些误解要么很容易化解，或者关于他们的讨论在最近几年已经逐渐变得琐碎，但是关于「无用」交换区的传言有更深的经验传承的根基，而不是一两个类比就能解释清楚的，并且要探讨这个先得对内存管理有一些基础认知。


本文主要目标是针对那些管理 Linux 系统并且有  兴趣理解「让系统运行于低/无交换区状态」或者「把 `vm.swappiness` 设为 0」这些做法的反论。


背景

如果没有基本理解 Linux 内存管理的底层机制是  如何运作的，就很难讨论为什么需要交换区以及交换出页面对正常运行的系统为什么是件好事，所以我们先确保大家有讨论的基础。

内存的类型

Linux 中内存分为好几种类型，每种都有各自的  属性。想理解为什么交换区很重要的关键一点在于理解这些的细微区别。

比如说，有种 **页面（「整块」的内存，通常  4K）** 是用来存放电脑里每个程序运行时各自的代码的。也有页面用来保存这些程序所需要读取的文件数据和元数据的缓存，以便加速随后的文件读写。这些内存页面构成 **页面缓存（page cache）**，后文中我称他们为文件内存。

还有一些页面是在代码执行过程中做的内存分配  得到的，比如说，当代码调用 `malloc` 能分配到新内存区，或者使用 `mmap` 的 `MAP_ANONYMOUS` 标志分配的内存。这些是「匿名(anonymous)」页面——之所以这么称呼它们是因为他们没有任何东西作后备——后文中我称他们为匿名内存。

还有其它类型的内存——共享内存、slab内存、内核栈内存、文件缓冲区（buffers），这种的——但是匿名内存和文件内存是最知名也最好理解的，所以后面的例子里我会用这两个说明，虽然后面的说明也同样适用于别的这些内存类型。

可回收/不可回收内存

考虑某种内存的类型时，一个非常基础的问题是这种内存是否能被回收。「回收（Reclaim）」在这里是指系统可以，在不丢失数据的前提下，从物理内存中释放这种内存的页面。

对一些内存类型而言，是否可回收通常可以直接判断。比如对于那些干净（未修改）的页面缓存内存，我们只是为了性能在用它们缓存一些磁盘上现有的数据，所以我们可以直接扔掉这些页面，不需要做什么特殊的操作。

对有些内存类型而言，回收是可能的，但是不是那么直接。比如对脏（修改过）的页面缓存内存，我们不能直接扔掉这些页面，因为磁盘上还没有写入我们所做的修改。这种情况下，我们可以选择拒绝回收，或者选择先等待我们的变更写入磁盘之后才能扔掉这些内存。



对还有些内存类型而言，是不能回收的。比如前面提到的匿名页面，它们只存在于内存中，没有任何后备存储，所以它们必须留在内存里。

说到交换区的本质

如果你去搜 Linux 上交换区的目的的描述，肯定会找到很多人说交换区只是在紧急时用来扩展物理内存的机制。比如下面这段是我在 google 中输入「什么是 swap」从前排结果中随机找到的一篇：

交换区本质上是紧急内存；是为了应对你的系统临时所需内存多余你现有物理内存时，专门分出一块额外空间。大家觉得交换区「不好」是因为它又慢又低效，并且如果你的系统一直需要使用交换区那说明它明显没有足够的内存。 [……] 如果你有足够内存覆盖所有你需要的情况，而且你觉得肯定不会用满内存，那么完全可以不用交换区 安全地运行系统。

事先说明，我不想因为这些文章的内容责怪这些文章的作者——这些内容被很多 Linux 系统管理员认为是「常识」，并且很可能你问他们什么是交换区的时候他们会给你这样的回答。但是也很不幸的是，这种认识是使用交换区的目的的一种普遍误解，尤其在现代系统上。

前文中我说过回收匿名页面的内存是「不可能的」，因为匿名内存的特点，把它们从内存中清除掉之后，没有别的存储区域能作为他们的备份——因此，要回收它们会造成数据丢失。但是，如果我们为这种内存页面创建一种后备存储呢？

嗯，这正是交换区存在的意义。交换区是一块存储空间，用来让这些看起来「不可回收」的内存页面在需要的时候可以交换到存储设备上。这意味着有了交换区之后，这些匿名页面也和别的那些可回收内存一样，可以作为内存回收的候选，就像干净文件页面，从而允许更有效地使用物理内存。

交换区主要是为了平等的回收机制，而不是为了紧急情况的「额外内存」。使用交换区不会让你的程序变慢——进入内存竞争的状态才是让程序变慢的元凶。

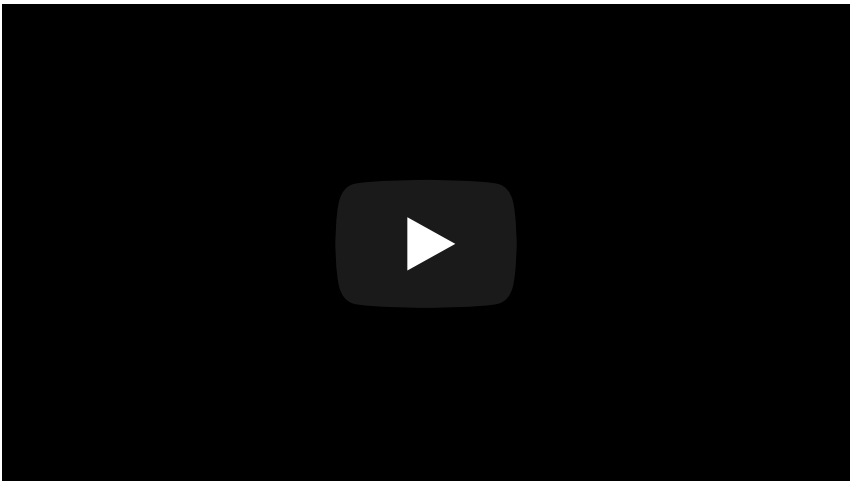
那么在这种「平等的可回收机遇」的情况下，让我们选择回收匿名页面的行为在何种场景中更合理呢？抽象地说，比如在下述不算罕见的场景中：

1. 程序初始化的时候，那些长期运行的程序可能要分配和使用很多页面。这些页面可能在最后


的关闭/清理的时候还需要使用，但是在程序「启动」之后（以具体的程序相关的方式）持续运行的时候不需要访问。对后台服务程序来说，很多后台程序要初始化不少依赖库，这种情况很常见。

2. 在程序的正常运行过程中，我们可能分配一些很少使用的内存。对整体系统性能而言可能比起让这些内存页一直留在内存中，只有在用到的时候才按需把它们用 **缺页异常 (major fault)** 换入内存，可以空出更多内存留给更重要的东西。


cgroupv2: Linux's new unified control group hierarchy (QCON London 2017)



考察有无交换区时会发生什么

我们来看一些在常见场景中，有无交换区时分别  会如何运行。在我的 [关于 cgroup v2 的演讲](#) 中探讨过「内存竞争」的指标。

在无/低内存竞争的状态下

- **有交换区:** 我们可以选择换出那些只有在进程  生存期内很小一部分时间会访问的匿名内存，这允许我们空出更多内存空间用来提升缓存命中率，或者做别的优化。
- **无交换区:** 我们不能换出那些很少使用的匿名内存，因为它们被锁在了内存中。虽然这通常不会直接表现出问题，但是在一些工作条件下这可能造成卡顿导致不平凡的性能下降，因为匿名内存占着空间不能给 更重要的需求使用。

译注：关于 **内存热度** 和 **内存颠簸 (thrash)**

讨论内核中内存管理的时候经常会说到内存页的 **冷热** 程度。这里冷热是指历史上内存页被访问到的频度，内存管理的经验在说，历史上在近期频繁

访问的**热**内存，在未来也可能被频繁访问，从而应该留在物理内存里；反之历史上不那么频繁访问的**冷**内存，在未来也可能很少被用到，从而可以考虑交换到磁盘或者丢弃文件缓存。

颠簸 (thrash) 这个词在文中出现多次但是似乎没有详细介绍，实际计算机科学专业的课程中应该有讲过。一段时间内，让程序继续运行所需的热内存总量被称作程序的工作集 (workset)，估算工作集大小，换句话说判断进程分配的内存页中哪些属于**热**内存哪些属于**冷**内存，是内核中内存管理的最重要的工作。当分配给程序的内存大于工作集的时候，程序可以不需要等待I/O全速运行；而当分配给程序的内存不足以放下整个工作集的时候，意味着程序每执行一小段就需要等待换页或者等待磁盘缓存读入所需内存页，产生这种情况的时候，从用户角度来看可以观察到程序肉眼可见的「卡顿」。当系统中所有程序都内存不足的时候，整个系统都处于颠簸的状态下，响应速度直接降至磁盘I/O的带宽。如本文所说，禁用交换区并不能防止颠簸，只是从等待换页变成了等待文件缓存，给程序分配超过工作集大小的内存才能防止颠簸。

在中/高内存竞争的状态下

- **有交换区:** 所有内存类型都有平等的被回收的



可能性。这意味着我们回收页面有更高的成功率——成功回收的意思是说被回收的那些页面不会在近期内被缺页异常换回内存中（颠簸）。


- **无交换区:** 匿名内存因为无处可去所以被锁在内存中。长期内存回收的成功率变低了，因为我们成体上能回收的页面总量少了。发生缺页颠簸的危险性更高了。缺乏经验的读者可能觉得这某时也是好事，因为这能避免进行磁盘I/O，但是实际上不是如此——我们只是把交换页面造成的磁盘I/O变成了扔掉热缓存页和扔掉代码段，这些页面很可能马上又要从文件中读回来。


在临时内存占用高峰时


- **有交换区:** 我们对内存使用激增的情况更有抵抗力，但是在严重的内存不足的情况下，从开始发生内存颠簸到OOM杀手开始工作的时间会被延长。内存压力造成的问题更容易观察到，从而可能更有效地应对，或者更有机会可控地干预。
- **无交换区:** 因为匿名内存被锁在内存中了不能被回收，所以OOM杀手会被更早触发。发生内存颠簸的可能性更大，但是发生颠簸之后到OOM解决问题的时间间隔被缩短了。基于你的程序，这可能更好或是更糟。比如说，基于队列的程序可能更希望这种从颠簸到杀进程的转换更快发生。即便如此，发生OOM的时机通常还是太迟于是没什么帮

助——只有在系统极度内存紧缺的情况下才会请出 OOM 杀手，如果想依赖这种行为模式，不如换成更早杀进程的方案，因为在这之前已经发生内存竞争了。

好吧，所以我需要系统交换区，但是我该怎么为每个程序微调它的行为？

你肯定想到了我写这篇文章一定会在哪儿插点  `cgroup v2` 的安利吧？;-)

显然，要设计一种对所有情况都有效的启发算法  会非常难，所以给内核提一些指引就很重要。历史上我们只能在整个系统层面做这方面微调，通过 `vm.swappiness` 。这有两方面问题：
`vm.swappiness` 的行为很难准确控制，因为它只是传递给一个更大的启发式算法中的一个小参数；并且它是一个系统级别的设置，没法针对一小部分进程微调。

你可以用 `mlock` 把页面锁在内存里，但是这要  么必须改程序代码，或者折腾 `LD_PRELOAD` ，或者在运行期用调试器做一些魔法操作。对基于虚拟机的语言来说这种方案也不能很好工作，因为通常你没法控制内存分配，最后得用上 `mlockall` ，而这个没有办法精确指定你实际上想锁住的页面。

cgroup v2 提供了一套可以每个 cgroup 微调的 `memory.low`，允许我们告诉内核说当使用的内存低于一定阈值之后优先回收别的程序的内存。这可以让我们不强硬禁止内核 换出程序的一部分内存，但是当发生内存竞争的时候让内核优先回收别的程序的内存。在正常条件下，内核的交换逻辑通常还是不错的，允许它有条件地换出一部分页面通常可以改善系统性能。在内存竞争的时候 发生交换颠簸虽然不理想，但是这更多地是单纯因为整体内存不够了，而不是因为交换进程（swapper）导致的问题。在这种场景下，你通常希望在内存压力开始积攒的时候通过自杀一些非关键的进程的方式来快速退出（fail fast）。

你不能依赖 OOM 杀手达成这个。OOM 杀手只有在非常急迫的情况下才会出动，那时系统已经处于极度不健康的状态了，而且很可能在这种状态下保持了一阵子了。需要在开始考虑 OOM 杀手之前，积极地自己处理这种情况。

不过，用传统的 Linux 内存统计数据还是挺难判断内存压力的。我们有一些看起来相关的系统指标，但是都只是支离破碎的——内存用量、页面扫描，这些——单纯从这些指标很难判断系统是处于高效的内存利用率还是在滑向内存竞争状态。我们在 Facebook 有个团队，由 Johannes 牵头，努力开发一些能评价内存压力的新指标，希望能在今后改善目前的现状。如果你对这方面感兴趣，在我的 cgroup v2 的演讲中介绍到一个被提议的指标。

调优

那么，我需要多少交换空间？

通常而言，最优内存管理所需的最小交换空间取决于程序固定在内存中而又很少访问到的匿名页面的数量，以及回收这些匿名页面换来的价值。后者大体上来说是问哪些页面不再会因为要保留这些很少访问的匿名页面而被回收掉腾出空间。

如果你有足够大的磁盘空间和比较新的内核版本（4.0+），越大的交换空间基本上总是越好的。更老的内核上 `kswapd`，内核中负责管理交换区的内核线程，在历史上倾向于有越多交换空间就急于交换越多内存出去。在最近一段时间，可用交换空间很大的时候的交换行为已经改善了很多。如果在运行 4.0+ 以后的内核，即便有很大的交换区在现代内核上也不会很激进地做交换。因此，如果你有足够的容量，现代内核上有个几个 GB 的交换空间大小能让你有更多选择。

如果你的磁盘空间有限，那么答案更多取决于你愿意做的取舍，以及运行的环境。理想上应该有足够的交换空间能高效应对正常负载和高峰（内存）负载。我建议先用 2-3GB 或者更多的交换空间搭个测试环境，然后监视在不同（内存）负载条件下持续一周左右的情况。只要在那一周里没有发生过严重的内存不足——发


生了的话说明测试结果没什么用——在测试结束的时候大概会留有多少 MB 交换区占用。作为结果说明你至少应该有那么多可用的交换空间，再多加一些以应对负载变化。用日志模式跑 atop 可以在 SWAPSZ 栏显示程序的页面被交换出去的情况，所以如果你还没用它记录服务器历史日志的话，这次测试中可以试试在测试机上用它记录日志。这也会告诉你什么时候你的程序开始换出页面，你可以用这个对照事件日志或者别的关键数据。

另一点值得考虑的是交换空间所在存储设备的媒介。读取交换区倾向于很随机，因为我们不能可靠预测什么时候什么页面会被再次访问。在 SSD 上这不是什么问题，但是在传统磁盘上，随机 I/O 操作会很昂贵，因为需要物理动作寻道。另一方面，重新加载文件缓存可能不那么随机，因为单一程序在运行期的文件读操作一般不会太碎片化。这可能意味着在传统磁盘上你想更多地回收文件页面而不是换出匿名页面，但仍就，你需要做测试评估在你的工作负载下如何取得平衡。


译注：关于休眠到磁盘时的交换空间大小

原文这里建议交换空间至少是物理内存大小，我觉得实际上不需要。休眠到磁盘的时候内核会写回并丢弃所有有文件作后备的可回收页面，交换区只需要能放下那些没有文件后备的页面就可以了。如果去掉文件缓存页面之后剩下的已用物理内存总量能完整放入交换区中，就可以正常休眠。对于桌面浏览器这种内存大户，通常有很多缓存页可以在休眠

的时候丢弃。

对笔记本/桌面用户如果想要休眠到交换区，这也需要考虑——这种情况下你的交换文件应该至少是物理内存大小。 

我的 swappiness 应该如何设置？

首先很重要的一点是，要理解 `vm.swappiness`  是做什么的。 `vm.swappiness` 是一个 `sysctl` 用来控制在内存回收的时候，是优先回收匿名页面，还是优先回收文件页面。内存回收的时候用两个属性：`file_prio`（回收文件页面的倾向）和 `anon_prio`（回收匿名页面的倾向）。 `vm.swappiness` 控制这两个值，因为它是 `anon_prio` 的默认值，然后也是默认 200 减去它之后 `file_prio` 的默认值。意味着如果我们设置 `vm.swappiness = 50` 那么结果是 `anon_prio` 是 50， `file_prio` 是 150（这里数值本身不是很重要，重要的是两者之间的权重比）。

译注：关于 SSD 上的 swappiness

原文这里说 SSD 上 swap 和 drop page cache 差不多开销所以 `vm.swappiness = 100`。我觉得实际上要考虑 swap out 的时候会产生写入操作，而 drop page cache 可能不需要写入（要看页面是否是脏页）。如果负载本身对 I/O 带宽比较敏感，稍微调低 swappiness 可能对性能更好，内核的默认值 60 是个不错的默认值。以及桌面用户可能对性能不那么关心，反而更关心 SSD 的写入寿命，虽然说 SSD 写入寿命一般也足够桌面用户，不过调低 swappiness 可能也能减少一部分不必要的写入（因为写回脏页是必然会发生的，而写 swap 可以避免）。当然太低的 swappiness 会对性能有负面影响（因为太多匿名页面留在物理内存里而降低了缓存命中率），这里的权衡也需要根据具体负载做测试。

另外澄清一点误解，swap 分区还是 swap 文件对系统运行时的性能而言没有差别。或许有人会觉得 swap 文件要经过文件系统所以会有性能损失，在译文之前译者说过 Linux 的内存管理子系统基本上独立于文件系统。实际上 Linux 上的 `swapon` 在设置 swap 文件作为交换空间的时候会读取一次文件系统元数据，确定 swap 文件在磁盘上的地址范围，随后运行的过程中做交换就和文件系统无关了。关于 swap 空间是否连续的影响，因为 swap 读写基本是页面单位的随机读写，所以即便连续的 swap 空间（swap 分区）也并不能改善 swap 的性能。稀疏文件的地址范围本身不连续，写入稀疏文件的空洞需要文件系统分配磁盘空间，所

以在 Linux 上交换文件不能是稀疏文件。只要不是稀疏文件，连续的文件内地址范围在磁盘上是否连续（是否有文件碎片）基本不影响能否 swapon 或者使用 swap 时的性能。

这意味着，通常来说 `vm.swappiness` 只是一个比例，用来衡量在你的硬件和工作负载下，回收和换回匿名内存还是文件内存哪种更昂贵。设定的值越低，你就是在告诉内核说换出那些不常访问的匿名页面在你的硬件上开销越昂贵；设定的值越高，你就是在告诉内核说在你的硬件上交换匿名页和文件缓存的开销越接近。内存管理子系统仍然还是会根据实际想要回收的内存的访问热度尝试自己决定具体是交换出文件还是匿名页面，只不过 `swappiness` 会在两种回收方式皆可的时候，在计算开销权重的过程中左右是该更多地做交换还是丢弃缓存。在 SSD 上这两种方式基本上是同等开销，所以设成 `vm.swappiness = 100`（同等比重）可能工作得不错。在传统磁盘上，交换页面可能会更昂贵，因为通常需要随机读取，所以你可能想要设低一些。

现实是大部分人对他们的硬件需求没有什么感受，所以根据直觉调整这个值可能挺困难的——你需要用不同的值做测试。你也可以花时间评估一下你的系统的内存分配情况和核心应用在大量回收内存的时候的行为表现。



讨论 `vm.swappiness` 的时候，一个极为重要需要考虑的修改是（相对）近期在 2012 年左右 Satoru Moriya 对 `vmscan` 行为的修改，它显著改变了代码对 `vm.swappiness = 0` 这个值的处理方式。

基本上来说这个补丁让我们在 `vm.swappiness = 0` 的时候会极度避免扫描（进而回收）匿名页面，除非我们已经在经历严重的内存抢占。就如本文前面所属，这种行为基本上不会是你想要的，因为这种行为会导致在发生内存抢占之前无法保证内存回收的公平性，这甚至可能是最初导致发生内存抢占的原因。想要避免这个补丁中对扫描匿名页面的特殊行为的话，`vm.swappiness = 1` 是你能设置的最低值。

内核在这里设置的默认值是 `vm.swappiness = 60`。这个值对大部分工作负载来说都不会太坏，但是很难有一个默认值能符合所有种类的工作负载。因此，对上面「那么，我需要多少交换空间？」那段讨论的一点重要扩展可以说，在测试系统中可以尝试使用不同的 `vm.swappiness`，然后监视你的程序和系统在重（内存）负载下的性能指标。在未来某天，如果我们在内核中有了合理的缺页检测，你也将能通过 `cgroup v2` 的页面缺页指标来以负载无关的方式决定这个。



2019年07月更新：内核 4.20+ 中的内存压力指标

前文中提到的开发中的内存缺页检测指标已经进入 4.20+ 以上版本的内核，可以通过 `CONFIG_PSI=y` 开启。详情参见我在 SREcon 大约 25:05 左右的讨论。

结论

- 交换区是允许公平地回收内存的有用工具，但是它的目的经常被人误解，导致它在业内这种负面声誉。如果你是按照原本的目的使用交换区的话——作为增加内存回收公平性的方式——你会

发现它是很有效的工具而不是阻碍。

- 禁用交换区并不能在内存竞争的时候防止磁盘I/O的问题，它只不过把匿名页面的磁盘I/O变成了文件页面的磁盘I/O。这不仅更低效，因为我们回收内存的时候能选择的页面范围更小了，而且它可能是导致高度内存竞争状态的元凶。
- 有交换区会导致系统更慢地使用 OOM 杀手，因为在缺少内存的情况下它提供了另一种更慢的内存，会持续地内存颠簸——内核调用 OOM 杀手只是最后手段，会晚于所有事情已经被搞得一团糟之后。解决方案取决于你的系统：

- 你可以预先更具每个 cgroup 的或者系统全局的内存压力改变系统负载。这能防止我们最初进入内存竞争的状态，但是 Unix 的历史中一直缺乏可靠的内存压力检测方式。希望不久之后在有了 缺页检测 这样的性能指标之后能改善这一点。
- 你可以使用 `memory.low` 让内核不倾向于回收（进而交换）特定一些 cgroup 中的进程，允许你在不禁用交换区的前提下保护关键后台服务。

感谢在撰写本文时 Rahul，Tejun 和 Johannes 提供的诸多建议和反馈。

