

【譯】替 swap 辯護：常見的誤解



目錄

目录

- 背景
 - 內存的類型
 - 可回收/不可回收內存
- 說到交換區的本質

- 考察有無交換區時會發生什麼
 - 在無/低內存競爭的狀態下
 - 在中/高內存競爭的狀態下
 - 在臨時內存佔用高峰時
 - 好吧，所以我需要系統交換區，但是我該怎麼為每個程序微調它的行爲？
- 調優
 - 那麼，我需要多少交換空間？
 - 我的 swappiness 應該如何設置？
 - 2019年07月更新：內核 4.20+ 中的內存壓力指標
- 結論

這篇翻譯自 Chris Down 的博文 [In defence of swap: common misconceptions](#)。原文的協議是 CC BY-SA 4.0，本文翻譯同樣也使用 CC BY-SA 4.0。其中加入了一些我自己的理解作為旁註，所有譯註都在側邊欄中。

翻譯這篇文章是因為經常看到朋友們（包括有經驗的程序員和 Linux 管理員）對 swap 和 swappiness 有諸多誤解，而這篇文章正好澄清了這些誤解，也講清楚了 Linux 中這兩者的本質。值得一提的是本文討論的 swap 針對 Linux 內核，在別的系統包括 macOS/WinNT 或者 Unix 系統中的交換文件可能有不同一樣的行爲，需要不同的調優方式。比如在 [FreeBSD handbook](#) 中明確建議了 swap 分區通常應該是兩倍物理內存大小，這一點建議對 FreeBSD 系內核的內存管理可能非常合理，而不一

定適合 Linux 內核，FreeBSD 和 Linux 有不同的內存管理方式尤其是 swap 和 page cache 和 buffer cache 的處理方式有諸多不同。

經常有朋友看到系統卡頓之後看系統內存使用狀況觀察到大量 swap 佔用，於是覺得卡頓是來源於 swap。就像文中所述，相關不蘊含因果，產生內存顛簸之後的確會造成大量 swap 佔用，也會造成系統卡頓，但是 swap 不是導致卡頓的原因，關掉 swap 或者調低 swappiness 並不能阻止卡頓，只會將 swap 造成的 I/O 轉化為加載文件緩存造成的 I/O。

以下是原文翻譯：

這篇文章也有 [日文](#) 和 [俄文](#) 翻譯。

太長不看：




1. 對維持系統的正常功能而言，有 swap 是相對挺重要的一部分。沒有它的話會更難做到合理的內存管理。
2. swap 的目的通常並不是用作緊急內存，它的目的在於讓內存回收能更平等和高效。事實上把它當作「緊急內存」來用的想法通常是有害的。
3. 禁用 swap 在內存壓力下並不能避免磁盤 I/O 造成的性能問題，這麼做只是讓磁盤 I/O 顛簸的範圍從匿名頁面轉化到文件頁面。這不僅更低效，因為系統能回收的頁面的選擇範圍更有限了，而且這種做法還可能是加重了內存壓力的原因之一。

4. 內核 4.0 版本之前的交換進程 (swapper) 有一些問題，導致很多人對 swap 有負面印象，因為它太急於 (overeagerness) 把頁面交換出去。在 4.0 之後的內核上這種情況已經改善了很多。
5. 在 SSD 上，交換出匿名頁面的開銷和回收文件頁面的開銷基本上在性能/延遲方面沒有區別。在老式的磁盤上，讀取交換文件因為屬於隨機訪問讀取所以會更慢，於是設置較低的 `vm.swappiness` 可能比較合理（繼續讀下面關於 `vm.swappiness` 的描述）。
6. 禁用 swap 並不能避免在接近 OOM 狀態下最終表現出的症狀，儘管的確有 swap 的情況下這種症狀持續的時間可能會延長。在系統調用 OOM 殺手的時候無論有沒有啓用 swap，或者更早/更晚開始調用 OOM 殺手，結果都是一樣的：整個系統留在了一種不可預知的狀態下。有 swap 也不能避免這一點。
7. 可以用 cgroup v2 的 `memory.low` 相關機制來改善內存壓力下 swap 的行為並且避免發生顛簸。


我的工作的一部分是改善內核中內存管理和 cgroup v2 相關，所以我和很多工程師討論過看待內存管理的態度，尤其是在壓力下應用程序的行為和操作系統在底層內存管理中用的基於經驗的啓發式決策邏輯。




在這種討論中經常重複的話題是交換區（swap）。交換區的話題是非常有爭議而且很少被理解的話題，甚至包括那些在 Linux 上工作過多年的人也是如此。很多人覺得它沒什麼用甚至是有害的：它是歷史遺蹟，從內存緊缺而磁盤讀寫是必要之惡的時代遺留到現在，為計算機提供在當年很必要的頁面交換功能作為內存空間。最近幾年我還經常能以一定頻度看到這種論調，然後我和很多同事、朋友、業界同行們討論過很多次，幫他們理解為什麼在現代計算機系統中交換區仍是有用的概念，即便現在的電腦中物理內存已經遠多於過去。

圍繞交換區的目的還有很多誤解——很多人覺得  它只是某種為了應對緊急情況的「慢速額外內存」，但是沒能理解在整個操作系統健康運作的時候它也能改善普通負載的性能。


我們很多人也聽說過描述內存時所用的常見說法：「Linux 用了太多內存」，「swap 應該設為物理內存的兩倍大小」，或者類似的說法。雖然這些誤解要麼很容易化解，或者關於他們的討論在最近幾年已經逐漸變得瑣碎，但是關於「無用」交換區的傳言有更深的經驗傳承的根基，而不是一兩個類比就能解釋清楚的，並且要探討這個先得對內存管理有一些基礎認知。


本文主要目標是針對那些管理 Linux 系統並且有  興趣理解「讓系統運行於低/無交換區狀態」或者「把 `vm.swappiness` 設為 0」這些做法的反論。


背景

如果沒有基本理解 Linux 內存管理的底層機制是  如何運作的，就很難討論為什麼需要交換區以及交換出頁面對正常運行的系統為什麼是件好事，所以我們先確保大家有討論的基礎。

內存的類型

Linux 中內存分爲好幾種類型，每種都有各自的  屬性。想理解為什麼交換區很重要的關鍵一點在於理解這些的細微區別。

比如說，有種 **頁面**（「**整塊**」的內存，通常  **4K**）是用來存放電腦裏每個程序運行時各自的代碼的。也有頁面用來保存這些程序所需要讀取的文件數據和元數據的緩存，以便加速隨後的文件讀寫。這些內存頁面構成 **頁面緩存**（**page cache**），後文中我稱他們爲文件內存。

還有一些頁面是在代碼執行過程中做的內存分配  得到的，比如說，當代碼調用 `malloc` 能分配到新內存區，或者使用 `mmap` 的 `MAP_ANONYMOUS` 標誌分配的內存。這些是「匿名(anonymous)」頁面——之所以這麼稱呼它們是因為他們沒有任何東西作後備——後文中我稱他們爲匿名內存。

還有其它類型的內存——共享內存、slab內存、內核棧內存、文件緩衝區（buffers），這種的——但是匿名內存和文件內存是最知名也最好理解的，所以後面的例子裏我會用這兩個說明，雖然後面的說明也同樣適用於別的這些內存類型。

可回收/不可回收內存

考慮某種內存的類型時，一個非常基礎的問題是這種內存是否能被回收。「回收（Reclaim）」在這裏是指系統可以，在不丟失數據的前提下，從物理內存中釋放這種內存的頁面。


對一些內存類型而言，是否可回收通常可以直接判斷。比如對於那些乾淨（未修改）的頁面緩存內存，我們只是爲了性能在用它們緩存一些磁盤上現有的數據，所以我們可以直接扔掉這些頁面，不需要做什麼特殊的操作。

對有些內存類型而言，回收是可能的，但是不是那麼直接。比如對髒（修改過）的頁面緩存內存，我們不能直接扔掉這些頁面，因爲磁盤上還沒有寫入我們所做的修改。這種情況下，我們可以選擇拒絕回收，或者選擇先等待我們的變更寫入磁盤之後才能扔掉這些內存。



對還有些內存類型而言，是不能回收的。比如前面提到的匿名頁面，它們只存在於內存中，沒有任何後備存儲，所以它們必須留在內存裏。

說到交換區的本質

如果你去搜 Linux 上交換區的目的的描述，肯定  會找到很多人說交換區只是在緊急時用來擴展物理內存的機制。比如下面這段是我在 google 中輸入「什麼是 swap」從前排結果中隨機找到的一篇：

交換區本質上是緊急內存；是爲了應對你的系統臨時所需內存多餘你現有物理內存時，專門分出一塊額外空間。大家覺得交換區「不好」是因爲它又慢又低效，並且如果你的系統一直需要使用交換區那說明它明顯沒有足夠的內存。 [……] 如果你有足夠內存覆蓋所有你需要的情況，而且你覺得肯定不會用滿內存，那麼完全可以不用交換區 安全地運行系統。

事先說明，我不想因為這些文章的內容責怪這些文章的作者——這些內容被很多 Linux 系統管理員認為是「常識」，並且很可能你問他們什麼是交換區的時候他們會給你這樣的回答。但是也很不幸的是，這種認識是使用交換區的目的的一種普遍誤解，尤其在現代系統上。

前文中我說過回收匿名頁面的內存是「不可能的」，因為匿名內存的特點，把它們從內存中清除掉之後，沒有別的存儲區域能作為他們的備份——因此，要回收它們會造成數據丟失。但是，如果我們為這種內存頁面創建一種後備存儲呢？

嗯，這正是交換區存在的意義。交換區是一塊存儲空間，用來讓這些看起來「不可回收」的內存頁面在需要的時候可以交換到存儲設備上。這意味着有了交換區之後，這些匿名頁面也和別的那些可回收內存一樣，可以作為內存回收的候選，就像乾淨文件頁面，從而允許更有效地使用物理內存。

交換區主要是為了平等的回收機制，而不是為了緊急情況的「額外內存」。使用交換區不會讓你的程序變慢——進入內存競爭的狀態才是讓程序變慢的元兇。

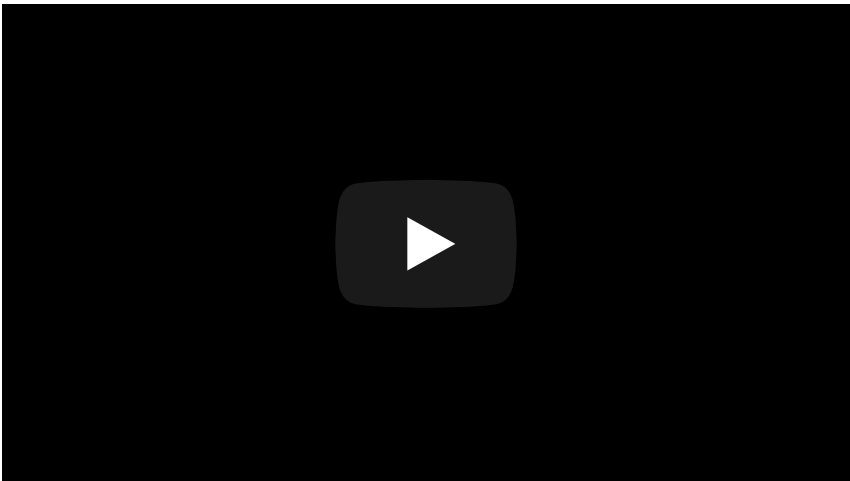
那麼在這種「平等的可回收機遇」的情況下，讓我們選擇回收匿名頁面的行為在何種場景中更合理呢？抽象地說，比如在下述不算罕見的場景中：

1. 程序初始化的時候，那些長期運行的程序可能要分配和使用很多頁面。這些頁面可能在最後

的關閉/清理的時候還需要使用，但是在程序「啓動」之後（以具體的程序相關的方式）持續運行的時候不需要訪問。對後臺服務程序來說，很多後臺程序要初始化不少依賴庫，這種情況很常見。

2. 在程序的正常運行過程中，我們可能分配一些很少使用的內存。對整體系統性能而言可能比起讓這些內存頁一直留在內存中，只有在用到的時候才按需把它們用 **缺頁異常 (major fault)** 換入內存，可以空出更多內存留給更重要的東西。

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



考察有無交換區時會發生什麼

我們來看一些在常見場景中，有無交換區時分別會如何運行。在我的關於 `cgroup v2` 的演講中探討過「內存競爭」的指標。

在無/低內存競爭的狀態下

- **有交換區:** 我們可以選擇換出那些只有在進程生存期內很小一部分時間會訪問的匿名內存，這允許我們空出更多內存空間用來提升緩存命中率，或者做別的優化。
- **無交換區:** 我們不能換出那些很少使用的匿名內存，因為它們被鎖在了內存中。雖然這通常不會直接表現出問題，但是在一些工作條件下這可能造成卡頓導致不平凡的性能下降，因為匿名內存佔着空間不能給更重要的需求使用。

譯註：關於 **內存熱度** 和 **內存顛簸 (thrash)**

討論內核中內存管理的時候經常會說到內存頁的**冷熱**程度。這裏冷熱是指歷史上內存頁被訪問到的頻度，內存管理的經驗在說，歷史上在近期頻繁

訪問的**熱**內存，在未來也可能被頻繁訪問，從而應該留在物理內存裏；反之歷史上不那麼頻繁訪問的**冷**內存，在未來也可能很少被用到，從而可以考慮交換到磁盤或者丟棄文件緩存。

顛簸 (thrash) 這個詞在文中出現多次但是似乎沒有詳細介紹，實際計算機科學專業的課程中應該有講過。一段時間內，讓程序繼續運行所需的熱內存總量被稱作程序的工作集 (workset)，估算工作集大小，換句話說判斷進程分配的內存頁中哪些屬於**熱**內存哪些屬於**冷**內存，是內核中內存管理的最重要的工作。當分配給程序的內存大於工作集的時候，程序可以不需要等待I/O全速運行；而當分配給程序的內存不足以放下整個工作集的時候，意味着程序每執行一小段就需要等待換頁或者等待磁盤緩存讀入所需內存頁，產生這種情況的時候，從用戶角度來看可以觀察到程序肉眼可見的「卡頓」。當系統中所有程序都內存不足的時候，整個系統都處於顛簸的狀態下，響應速度直接降至磁盤I/O的帶寬。如本文所說，禁用交換區並不能防止顛簸，只是從等待換頁變成了等待文件緩存，給程序分配超過工作集大小的內存才能防止顛簸。

在中/高內存競爭的狀態下

- **有交換區:** 所有內存類型都有平等的被回收的



可能性。這意味着我們回收頁面有更高的成功率——成功回收的意思是說被回收的那些頁面不會在近期內被缺頁異常換回內存中（顛簸）。


- **無交換區:** 匿名內存因為無處可去所以被鎖在內存中。長期內存回收的成功率變低了，因為我們成體上能回收的頁面總量少了。發生缺頁顛簸的危險性更高了。缺乏經驗的讀者可能覺得這某時也是好事，因為這能避免進行磁盤I/O，但是實際上不是如此——我們只是把交換頁面造成的磁盤I/O變成了扔掉熱緩存頁和扔掉代碼段，這些頁面很可能馬上又要從文件中讀回來。


在臨時內存佔用高峰時


- **有交換區:** 我們對內存使用激增的情況更有抵抗力，但是在嚴重的內存不足的情況下，從開始發生內存顛簸到OOM殺手開始工作的時間會被延長。內存壓力造成的問題更容易觀察到，從而可能更有效地應對，或者更有機會可控地干預。
- **無交換區:** 因為匿名內存被鎖在內存中了不能被回收，所以OOM殺手會被更早觸發。發生內存顛簸的可能性更大，但是發生顛簸之後到OOM解決問題的時間間隔被縮短了。基於你的程序，這可能更好或是更糟。比如說，基於隊列的程序可能更希望這種從顛簸到殺進程的轉換更快發生。即便如此，發生OOM的時機通常還是太遲於是沒什麼幫

助——只有在系統極度內存緊缺的情況下才會請出 OOM 殺手，如果想依賴這種行爲模式，不如換成更早殺進程的方案，因爲在這之前已經發生內存競爭了。

好吧，所以我需要系統交換區，但是我該怎麼爲每個程序微調它的行爲？

你肯定想到了我寫這篇文章一定會在哪兒插點  `cgroup v2` 的安利吧？;-)

顯然，要設計一種對所有情況都有效的啓發算法  會非常難，所以給內核提一些指引就很重要。歷史上我們只能在整個系統層面做這方面微調，通過 `vm.swappiness` 。這有兩方面問題：
`vm.swappiness` 的行爲很難準確控制，因爲它只是傳遞給一個更大的啓發式算法中的一個小參數；並且它是一個系統級別的設置，沒法針對一小部分進程微調。

你可以用 `mlock` 把頁面鎖在內存裏，但是這要  麼必須改程序代碼，或者折騰 `LD_PRELOAD` ，或者在運行期用調試器做一些魔法操作。對基於虛擬機的語言來說這種方案也不能很好工作，因爲通常你沒法控制內存分配，最後得用上 `mlockall` ，而這個沒有辦法精確指定你實際上想鎖住的頁面。

cgroup v2 提供了一套可以每個 cgroup 微調的 `memory.low`，允許我們告訴內核說當使用的內存低於一定閾值之後優先回收別的程序的內存。這可以讓我們不強硬禁止內核換出程序的一部分內存，但是當發生內存競爭的時候讓內核優先回收別的程序的內存。在正常條件下，內核的交換邏輯通常還是不錯的，允許它有條件地換出一部分頁面通常可以改善系統性能。在內存競爭的時候發生交換顛簸雖然不理想，但是這更多地是單純因為整體內存不夠了，而不是因為交換進程（swapper）導致的問題。在這種場景下，你通常希望在內存壓力開始積攢的時候通過自殺一些非關鍵的進程的方式來快速退出（fail fast）。

你不能依賴 OOM 殺手達成這個。OOM 殺手只有在非常急迫的情況下纔會出動，那時系統已經處於極度不健康的狀態了，而且很可能在這種狀態下保持了一陣子了。需要在開始考慮 OOM 殺手之前，積極地自己處理這種情況。

不過，用傳統的 Linux 內存統計數據還是挺難判斷內存壓力的。我們有一些看起來相關的系統指標，但是都只是支離破碎的——內存用量、頁面掃描，這些——單純從這些指標很難判斷系統是處於高效的內存利用率還是在滑向內存競爭狀態。我們在 Facebook 有個團隊，由 Johannes 牽頭，努力開發一些能評價內存壓力的新指標，希望能在今後改善目前的現狀。如果你對這方面感興趣，在我的 cgroup v2 的演講中介紹到一個被提議的指標。

調優

那麼，我需要多少交換空間？

通常而言，最優內存管理所需的最小交換空間取決於程序固定在內存中而又很少訪問到的匿名頁面的數量，以及回收這些匿名頁面換來的價值。後者大體上來說是問哪些頁面不再會因為要保留這些很少訪問的匿名頁面而被回收掉騰出空間。

如果你有足夠大的磁盤空間和比較新的內核版本（4.0+），越大的交換空間基本上總是越好的。更老的內核上 `kswapd`，內核中負責管理交換區的內核線程，在歷史上傾向於有越多交換空間就急於交換越多內存出去。在最近一段時間，可用交換空間很大的時候的交換行爲已經改善了很多。如果在運行 4.0+ 以後的內核，即便有很大的交換區在現代內核上也不會很激進地做交換。因此，如果你有足夠的容量，現代內核上有個幾個 GB 的交換空間大小能讓你有更多選擇。

如果你的磁盤空間有限，那麼答案更多取決於你願意做的取捨，以及運行的環境。理想上應該有足夠的交換空間能高效應對正常負載和高峰（內存）負載。我建議先用 2-3GB 或者更多的交換空間搭個測試環境，然後監視在不同（內存）負載條件下持續一週左右的情況。只要在那一週裏沒有發生過嚴重的內存不足——發

生了的話說明測試結果沒什麼用——在測試結束的時候大概會留有多少 MB 交換區佔用。作為結果說明你至少應該有那麼多可用的交換空間，再多加一些以應對負載變化。用日誌模式跑 atop 可以在 SWAPSZ 欄顯示程序的頁面被交換出去的情況，所以如果你還沒用它記錄服務器歷史日誌的話，這次測試中可以試試在測試機上用它記錄日誌。這也會告訴你什麼時候你的程序開始換出頁面，你可以用這個對照事件日誌或者別的關鍵數據。

另一點值得考慮的是交換空間所在存儲設備的媒介。讀取交換區傾向於很隨機，因為我們不能可靠預測什麼時候什麼頁面會被再次訪問。在 SSD 上這不是什麼問題，但是在傳統磁盤上，隨機 I/O 操作會很昂貴，因為需要物理動作尋道。另一方面，重新加載文件緩存可能不那麼隨機，因為單一程序在運行期的文件讀操作一般不會太碎片化。這可能意味着在傳統磁盤上你想更多地回收文件頁面而不是換出匿名頁面，但仍舊，你需要做測試評估在你的工作負載下如何取得平衡。

譯註：關於休眠到磁盤時的交換空間大小

原文這裏建議交換空間至少是物理內存大小，我覺得實際上不需要。休眠到磁盤的時候內核會寫回並丟棄所有有文件作後備的可回收頁面，交換區只需要能放下那些沒有文件後備的頁面就可以了。如果去掉文件緩存頁面之後剩下的已用物理內存總量能完整放入交換區中，就可以正常休眠。對於桌面瀏覽器這種內存大戶，通常有很多緩存頁可以在休眠

的時候丟棄。

對筆記本/桌面用戶如果想要休眠到交換區，這也需要考慮——這種情況下你的交換文件應該至少是物理內存大小。



我的 swappiness 應該如何設置？

首先很重要的一點是，要理解 `vm.swappiness` 是做什麼的。`vm.swappiness` 是一個 `sysctl` 用來控制在內存回收的時候，是優先回收匿名頁面，還是優先回收文件頁面。內存回收的時候用兩個屬性：`file_prio`（回收文件頁面的傾向）和 `anon_prio`（回收匿名頁面的傾向）。`vm.swappiness` 控制這兩個值，因為它是 `anon_prio` 的默認值，然後也是默認 200 減去它之後 `file_prio` 的默認值。意味着如果我們設置 `vm.swappiness = 50` 那麼結果是 `anon_prio` 是 50，`file_prio` 是 150（這裏數值本身不是很重要，重要的是兩者之間的權重比）。

譯註：關於 SSD 上的 swappiness

原文這裏說 SSD 上 swap 和 drop page cache 差不多開銷所以 `vm.swappiness = 100`。我覺得實際上要考慮 swap out 的時候會產生寫入操作，而 drop page cache 可能不需要寫入（要看頁面是否是髒頁）。如果負載本身對 I/O 帶寬比較敏感，稍微調低 swappiness 可能對性能更好，內核的默認值 60 是個不錯的默認值。以及桌面用戶可能對性能不那麼關心，反而更關心 SSD 的寫入壽命，雖然說 SSD 寫入壽命一般也足夠桌面用戶，不過調低 swappiness 可能也能減少一部分不必要的寫入（因為寫回髒頁是必然會發生的，而寫 swap 可以避免）。當然太低的 swappiness 會對性能有負面影響（因為太多匿名頁面留在物理內存裏而降低了緩存命中率），這裏的權衡也需要根據具體負載做測試。

另外澄清一點誤解，swap 分區還是 swap 文件對系統運行時的性能而言沒有差別。或許有人會覺得 swap 文件要經過文件系統所以會有性能損失，在譯文之前譯者說過 Linux 的內存管理子系統基本上獨立於文件系統。實際上 Linux 上的 `swapon` 在設置 swap 文件作為交換空間的時候會讀取一次文件系統元數據，確定 swap 文件在磁盤上的地址範圍，隨後運行的過程中做交換就和文件系統無關了。關於 swap 空間是否連續的影響，因為 swap 讀寫基本是頁面單位的隨機讀寫，所以即便連續的 swap 空間（swap 分區）也並不能改善 swap 的性能。稀疏文件的地址範圍本身不連續，寫入稀疏文件的空洞需要文件系統分配磁盤空間，所

以在 Linux 上交換文件不能是稀疏文件。只要不是稀疏文件，連續的文件內地址範圍在磁盤上是否連續（是否有文件碎片）基本不影響能否 swapon 或者使用 swap 時的性能。

這意味着，通常來說 `vm.swappiness` 只是一個比例，用來衡量在你的硬件和工作負載下，回收和換回匿名內存還是文件內存哪種更昂貴。設定的值越低，你就是在告訴內核說換出那些不常訪問的匿名頁面在你的硬件上開銷越昂貴；設定的值越高，你就是在告訴內核說在你的硬件上交換匿名頁和文件緩存的開銷越接近。內存管理子系統仍然還是會根據實際想要回收的內存的訪問熱度嘗試自己決定具體是交換出文件還是匿名頁面，只不過 `swappiness` 會在兩種回收方式皆可的時候，在計算開銷權重的過程中左右是該更多地做交換還是丟棄緩存。在 SSD 上這兩種方式基本上是同等開銷，所以設成 `vm.swappiness = 100`（同等比重）可能工作得不錯。在傳統磁盤上，交換頁面可能會更昂貴，因為通常需要隨機讀取，所以你可能想要設低一些。

現實是大部分人對他們的硬件需求沒有什麼感受，所以根據直覺調整這個值可能挺困難的——你需要用不同的值做測試。你也可以花時間評估一下你的系統的內存分配情況和核心應用在大量回收內存的時候的行為表現。

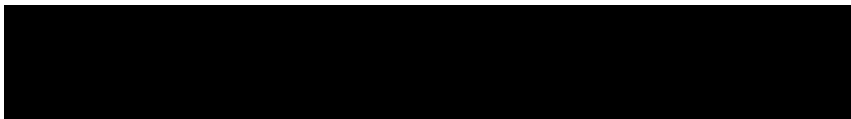


討論 `vm.swappiness` 的時候，一個極為重要需要考慮的修改是（相對）近期在 2012 年左右 Satoru Moriya 對 `vmscan` 行爲的修改，它顯著改變了代碼對 `vm.swappiness = 0` 這個值的處理方式。

基本上來說這個補丁讓我們在 `vm.swappiness = 0` 的時候會極度避免掃描（進而回收）匿名頁面，除非我們已經在經歷嚴重的內存搶佔。就如本文前面所屬，這種行爲基本上不會是你想要的，因為這種行爲會導致在發生內存搶佔之前無法保證內存回收的公平性，這甚至可能是最初導致發生內存搶佔的原因。想要避免這個補丁中對掃描匿名頁面的特殊行爲的話，`vm.swappiness = 1` 是你能設置的最低值。

內核在這裏設置的默認值是 `vm.swappiness = 60`。這個值對大部分工作負載來說都不會太壞，但是很難有一個默認值能符合所有種類的工作負載。因此，對上面「那麼，我需要多少交換空間？」那段討論的一點重要擴展可以說，在測試系統中可以嘗試使用不同的 `vm.swappiness`，然後監視你的程序和系統在重（內存）負載下的性能指標。在未來某天，如果我們在內核中有了合理的缺頁檢測，你也將能通過 `cgroup v2` 的頁面缺頁指標來以負載無關的方式決定這個。

SREcon19 Asia/Pacific - Linux Memory
Management at Scale: Under the Hood





2019年07月更新：內核 4.20+ 中的內存壓力指標

前文中提到的開發中的內存缺頁檢測指標已經進入 4.20+ 以上版本的內核，可以通過 `CONFIG_PSI=y` 開啓。詳情參見我在 SREcon 大約 25:05 左右的討論。

結論

- 交換區是允許公平地回收內存的有用工具，但是它的目的經常被人誤解，導致它在業內這種負面聲譽。如果你是按照原本的目的使用交換區的話——作為增加內存回收公平性的方式——你會

發現它是很有效的工具而不是阻礙。

- 禁用交換區並不能在內存競爭的時候防止磁盤I/O的問題，它只不過把匿名頁面的磁盤I/O變成了文件頁面的磁盤I/O。這不僅更低效，因為我們回收內存的時候能選擇的頁面範圍更小了，而且它可能是導致高度內存競爭狀態的元兇。
- 有交換區會導致系統更慢地使用 OOM 殺手，因為在缺少內存的情況下它提供了另一種更慢的內存，會持續地內存顛簸——內核調用 OOM 殺手只是最後手段，會晚於所有事情已經被搞得一團糟之後。解決方案取決於你的系統：

- 你可以預先更具每個 cgroup 的或者系統全局的內存壓力改變系統負載。這能防止我們最初進入內存競爭的狀態，但是 Unix 的歷史中一直缺乏可靠的內存壓力檢測方式。希望不久之後在有了缺頁檢測這樣的性能指標之後能改善這一點。
- 你可以使用 `memory.low` 讓內核不傾向於回收（進而交換）特定一些 cgroup 中的進程，允許你在不禁用交換區的前提下保護關鍵後臺服務。

感謝在撰寫本文時 [Rahul](#)，[Tejun](#) 和 [Johannes](#) 提供的諸多建議和反饋。

