



下载APP



10 | Swap: 容器可以使用Swap空间吗?

2020-12-07 李程远

容器实战高手课

[进入课程 >](#)**讲述: 李程远**

时长 15:36 大小 14.30M



你好, 我是程远。这一讲, 我们来看看容器中是否可以使用 Swap 空间。

用过 Linux 的同学应该都很熟悉 Swap 空间了, 简单来说它就是就是一块磁盘空间。

当内存写满的时候, 就可以把内存中不常用的数据暂时写到这个 Swap 空间上。这样一来, 内存空间就可以释放出来, 用来满足新的内存申请的需求。

它的好处是可以**应对一些瞬时突发的内存增大需求**, 不至于因为内存一时不够而触发 OOM Killer, 导致进程被杀死。



那么对于一个容器, 特别是容器被设置了 Memory Cgroup 之后, 它还可以使用 Swap 空间吗? 会不会出现什么问题呢?

问题再现

接下来，我们就结合一个小例子，一起来看看吧。

首先，我们在一个有 Swap 空间的节点上启动一个容器，设置好它的 Memory Cgroup 的限制，一起来看看接下来会发生什么。

如果你的节点上没有 Swap 分区，也没有关系，你可以用下面的 [这组命令](#) 来新建一个。

这个例子里，Swap 空间的大小是 20G，你可以根据自己磁盘空闲空间来决定这个 Swap 的大小。执行完这组命令之后，我们来运行 free 命令，就可以看到 Swap 空间有 20G。

输出的结果你可以参考下面的截图。

```
# cat create_swap.sh
#!/bin/bash
fallocate -l 20G ./swapfile
dd if=/dev/zero of=./swapfile bs=1024 count=20971520
chmod 600 ./swapfile
mkswap ./swapfile
swapon swapfile
# free -k
```

	total	used	free	shared	buff/cache	available
Mem:	89704584	942124	62010924	2496	26751536	87683764
Swap:	20971516	0	20971516			

然后我们再启动一个容器，和 OOM 那一讲里的 [例子](#) 差不多，容器的 Memory Cgroup 限制为 512MB，容器中的 mem_alloc 程序去申请 2GB 内存。

你会发现，这次和上次 OOM 那一讲里的情况不一样了，并没有发生 OOM 导致容器退出的情况，容器运行得好好的。

从下面的图中，我们可以看到，mem_alloc 进程的 RSS 内存一直在 512MB (RES: 515596) 左右。

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
1	root	20	0	2060324	515596	1348	S	0.0	0.6	0:04.96	mem_alloc
6	root	20	0	12020	3192	2776	S	0.0	0.0	0:00.19	bash
19	root	20	0	48432	3764	3152	R	0.0	0.0	0:00.02	top

那我们再看一下 Swap 空间，使用了 1.5GB (used 1542144KB)。输出的结果如下图，简单计算一下，1.5GB + 512MB，结果正好是 mem_alloc 这个程序申请的 2GB 内存。

```
[root@6782bd64fbc1 /]# free
```

	total	used	free	shared	buff/cache	available
Mem:	89704584	1495176	61504016	2500	26705392	87133816
Swap:	20971516	1542144	19429372			

通过刚刚的例子，你也许会这么想，因为有了 Swap 空间，本来会被 OOM Kill 的容器，可以好好地运行了。初看这样似乎也挺好的，不过你仔细想想，这样一来，Memory Cgroup 对内存的限制不就失去了作用么？

我们进一步分析，如果一个容器中的程序发生了内存泄漏（Memory leak），那么本来 Memory Cgroup 可以及时杀死这个进程，让它不影响整个节点中的其他应用程序。结果现在这个内存泄漏的进程没被杀死，还会不断地读写 Swap 磁盘，反而影响了整个节点的性能。

你看，这样一分析，对于运行容器的节点，你是不是又觉得应该禁止使用 Swap 了呢？

我想提醒你，不能一刀切地下结论，我们总是说，具体情况要具体分析，我们落地到具体的场景里，就会发现情况又没有原先我们想得那么简单。

比如说，某一类程序就是需要 Swap 空间，才能防止因为偶尔的内存突然增加而被 OOM Killer 杀死。因为这类程序重新启动的初始化时间会很长，这样程序重启的代价就很大了，也就是说，打开 Swap 对这类程序是有意义的。

这一类程序一旦放到容器中运行，就意味着它会和“别的容器”在同一个宿主机上共同运行，那如果这个“别的容器”不需要 Swap，而是希望 Memory Cgroup 的严格内存限制。

这样一来，在这一个宿主机上的两个容器就会有冲突了，我们应该怎么解决这个问题呢？要解决这个问题，我们先来看看 Linux 里的 Swappiness 这个概念，后面它可以帮到我们。

如何正确理解 swappiness 参数？

在普通 Linux 系统上, 如果你使用过 Swap 空间, 那么你可能配置过 `proc` 文件系统下的 `swappiness` 这个参数 (`/proc/sys/vm/swappiness`)。swappiness 的定义在 [Linux 内核文档](#)中可以找到, 就是下面这段话。

swappiness

This control is used to define how aggressive the kernel will swap memory pages. Higher values will increase aggressiveness, lower values decrease the amount of swap. A value of 0 instructs the kernel not to initiate swap until the amount of free and file-backed pages is less than the high water mark in a zone.

The default value is 60.

前面两句话大致翻译过来, 意思就是 **swappiness 可以决定系统将会有多频繁地使用交换分区。**

一个较高的值会使得内核更频繁地使用交换分区, 而一个较低的取值, 则代表着内核会尽量避免使用交换分区。swappiness 的取值范围是 0–100, 缺省值 60。

我第一次读到这个定义, 再知道了这个取值范围后, 我觉得这是一个百分比值, 也就是定义了使用 Swap 空间的频率。

当这个值是 100 的时候, 哪怕还有空闲内存, 也会去做内存交换, 尽量把内存数据写入到 Swap 空间里; 值是 0 的时候, 基本上就不做内存交换了, 也就不写 Swap 空间了。

后来再回顾的时候, 我发现这个想法不能说是完全错的, 但是想得简单了些。那这段 swappiness 的定义, 应该怎么正确地理解呢?

你还记得, 我们在上一讲里说过的两种内存类型 Page Cache 和 RSS 么?

在有磁盘文件访问的时候, Linux 会尽量把系统的空闲内存用作 Page Cache 来提高文件的读写性能。在没有打开 Swap 空间的情况下, 一旦内存不够, 这种情况下就只能把 Page Cache 释放了, 而 RSS 内存是不能释放的。

在 RSS 里的内存，大部分都是没有对应磁盘文件的内存，比如用 `malloc()` 申请得到的内存，这种内存也被称为**匿名内存 (Anonymous memory)**。那么当 Swap 空间打开后，可以写入 Swap 空间的，就是这些匿名内存。

所以在 Swap 空间打开的时候，问题也就来了，在内存紧张的时候，Linux 系统怎么决定是先释放 Page Cache，还是先把匿名内存释放并写入到 Swap 空间里呢？

我们一起来分析分析，都可能发生怎样的情况。最可能发生的是下面两种情况：

第一种情况是，如果系统先把 Page Cache 都释放了，那么一旦节点里有频繁的文件读写操作，系统的性能就会下降。

还有另一种情况，如果 Linux 系统先把匿名内存都释放并写入到 Swap，那么一旦这些被释放的匿名内存马上需要使用，又需要从 Swap 空间读回到内存中，这样又会让 Swap（其实也是磁盘）的读写频繁，导致系统性能下降。

显然，我们在释放内存的时候，需要平衡 Page Cache 的释放和匿名内存的释放，而 `swappiness`，就是用来定义这个平衡的参数。

那么 `swappiness` 具体是怎么来控制这个平衡的？我们看一下在 Linux 内核代码里是怎么用这个 `swappiness` 参数。

我们前面说了 `swappiness` 的这个值的范围是 0 到 100，但是请你一定要注意，**它不是一个百分比，更像是一个权重**。它是用来定义 Page Cache 内存和匿名内存的释放的一个比例。

我结合下面的这段代码具体给你讲一讲。

我们可以看到，这个比例是 `anon_prio: file_prio`，这里 `anon_prio` 的值就等于 `swappiness`。下面我们分三个情况做讨论：

第一种情况，当 `swappiness` 的值是 100 的时候，匿名内存和 Page Cache 内存的释放比例就是 100: 100，也就是等比例释放了。

第二种情况，就是 swappiness 缺省值是 60 的时候，匿名内存和 Page Cache 内存的释放比例就是 60 : 140，Page Cache 内存的释放要优先于匿名内存。

[复制代码](#)

```
1      /*
2      * With swappiness at 100, anonymous and file have the same priority.
3      * This scanning priority is essentially the inverse of IO cost.
4      */
5
6      anon_prio = swappiness;
7      file_prio = 200 - anon_prio;
```

还有一种情况，当 swappiness 的值是 0 的时候，会发生什么呢？这种情况下，Linux 系统是不允许匿名内存写入 Swap 空间了吗？

我们可以回到前面，再看一下那段 swappiness 的英文定义，里面特别强调了 swappiness 为 0 的情况。

当空闲内存少于内存一个 zone 的"high water mark"中的值的时候，Linux 还是会做内存交换，也就是把匿名内存写入到 Swap 空间后释放内存。

在这里 zone 是 Linux 划分物理内存的一个区域，里面有 3 个水位线（water mark），水位线可以用来警示空闲内存的紧张程度。

这里我们可以再做个试验来验证一下，先运行 `echo 0 > /proc/sys/vm/swappiness` 命令把 swappiness 设置为 0，然后用我们之前例子中的 mem_alloc 程序来申请内存。

比如我们的这个节点上内存有 12GB，同时有 2GB 的 Swap，用 mem_alloc 申请 12GB 的内存，我们可以看到 Swap 空间在 mem_alloc 调用之前，used=0，输出结果如下图所示。

```
-bash-4.2# free -m
              total        used         free       shared    buff/cache   available
Mem:          12008         210         9236           0         2562       11458
Swap:          2047           0         2047
-bash-4.2# cat /proc/sys/vm/swappiness
0
-bash-4.2# ./mem-alloc/mem_alloc 12000
Allocating,set to 12000 Mbytes
```

接下来, 调用 `mem_alloc` 之后, Swap 空间就被使用了。

```
-bash-4.2# free -m
              total        used         free       shared  buff/cache   available
Mem:          12008         11746           147            0           114            25
Swap:           2047           541          1506
```

因为 `mem_alloc` 申请 12GB 内存已经和节点最大内存差不多了, 我们如果查看 `cat /proc/zoneinfo`, 也可以看到 normal zone 里 high (water mark) 的值和 free 的值差不多, 这样在 `free < high` 的时候, 系统就会回收匿名内存页面并写入 Swap 空间。

```
Node 0, zone Normal
      pages free      20925
           min      12726
           low      15907
           high     19088
```

好了, 在这里我们介绍了 Linux 系统里 swappiness 的概念, 它是用来决定在内存紧张时候, 回收匿名内存和 Page Cache 内存的比例。

swappiness 的取值范围在 0 到 100, 值为 100 的时候系统平等回收匿名内存和 Page Cache 内存; 一般缺省值为 60, 就是优先回收 Page Cache; 即使 swappiness 为 0, 也不能完全禁止 Swap 分区的使用, 就是说在内存紧张的时候, 也会使用 Swap 来回收匿名内存。

解决问题

那么运行了容器, 使用了 Memory Cgroup 之后, swappiness 怎么工作呢?

如果你查看一下 Memory Cgroup 控制组下面的参数, 你会看到有一个 `memory.swappiness` 参数。这个参数是干啥的呢?

memory.swappiness 可以控制这个 Memroy Cgroup 控制组下面匿名内存和 page cache 的回收, 取值的范围和工作方式和全局的 swappiness 差不多。这里有一个优先顺序, 在 Memory Cgroup 的控制组里, 如果你设置了 memory.swappiness 参数, 它就会覆盖全局的 swappiness, 让全局的 swappiness 在这个控制组里不起作用。

不过, 这里有一点不同, 需要你留意: **当 memory.swappiness = 0 的时候, 对匿名页的回收是始终禁止的, 也就是始终都不会使用 Swap 空间。**

这时 Linux 系统不会再去比较 free 内存和 zone 里的 high water mark 的值, 再决定一个 Memory Cgroup 中的匿名内存要不要回收了。

请你注意, 当我们设置了"memory.swappiness=0 时, 在 Memory Cgroup 中的进程, 就不会再使用 Swap 空间, 知道这一点很重要。

我们可以跑个容器试一试, 还是在一个有 Swap 空间的节点上运行, 运行和这一讲开始一样的容器, 唯一不同的是把容器对应 Memory Cgroup 里的 memory.swappiness 设置为 0。

```
# free -m
total        used        free      shared  buff/cache   available
Mem:      87602        1510       58513         2       27577       84844
Swap:      20479          0       20479

# ./start_container.sh
3b7cc6882602066fce708031244cb2675517807ad4ec7f2e9ccd45d86637a59a
3b7cc6882602
/sys/fs/cgroup/memory/system.slice/docker-3b7cc6882602066fce708031244cb2675517807ad4ec7f2e9ccd45d86637a59a.scope/memory.swappiness
0
# free -m
total        used        free      shared  buff/cache   available
Mem:      87602        1527       58496         2       27578       84827
Swap:      20479          0       20479

# docker ps -a | grep mem_alloc
3b7cc6882602          registry/mem_alloc:v1                  "/mem_alloc 2000"          32 seconds ago          Exited (137) 1 second ago          mem_alloc
# docker inspect 3b7cc6882602 | grep -i oom
      "OOMKilled": true,
      "OomScoreAdj": 0,
      "OomKillDisable": false,
```

这次我们在容器中申请内存之后, Swap 空间就没有被使用了, 而当容器申请的内存超过 memory.limit_in_bytes 之后, 就发生了 OOM Kill。

好了, 有了"memory.swappiness = 0"的配置和功能, 就可以解决我们在这讲里最开始提出的问题了。

在同一个宿主机上, 假设同时存在容器 A 和其他容器, 容器 A 上运行着需要使用 Swap 空间的应用, 而别的容器不需要使用 Swap 空间。

那么，我们还是可以在宿主机节点上打开 Swap 空间，同时在其他容器对应的 Memory Cgroups 控制组里，把 `memory.swappiness` 这个参数设置为 0。这样一来，我们不但满足了容器 A 的需求，而且别的容器也不会受到影响，仍然可以严格按照 Memory Cgroups 里的 `memory.limit_in_bytes` 来限制内存的使用。

总之，`memory.swappiness` 这个参数很有用，通过它可以让需要使用 Swap 空间的容器和不需要 Swap 的容器，同时运行在同一个宿主机上。

重点总结

这一讲，我们主要讨论的问题是在容器中是否可以使用 Swap？

这个问题没有看起来那么简单。当然了，只要在宿主机节点上打开 Swap 空间，在容器中就是可以用到 Swap 的。但出现的问题是在同一个宿主机上，对于不需要使用 swap 的容器，它的 Memory Cgroups 的限制也失去了作用。

针对这个问题，我们学习了 Linux 中的 `swappiness` 这个参数。`swappiness` 参数值的作用是，在系统里有 Swap 空间之后，当系统需要回收内存的时候，是优先释放 Page Cache 中的内存，还是优先释放匿名内存（也就是写入 Swap）。

`swappiness` 的取值范围在 0 到 100 之间，我们可以记住下面三个值：

值为 100 的时候，释放 Page Cache 和匿名内存是同等优先级的。

值为 60，这是大多数 Linux 系统的缺省值，这时候 Page Cache 的释放优先级高于匿名内存的释放。

值为 0 的时候，当系统中空闲内存低于一个临界值的时候，仍然会释放匿名内存并把页面写入 Swap 空间。

swappiness 参数

值为 100

释放 Page Cache 和匿名内存是同等优先级的。

值为 60

Page Cache 的释放优先级高于匿名内存的释放。

值为 0

不能完全禁止 Swap 分区的使用。
如果内存紧张，仍会通过 Swap 回收匿名内存。

Swappiness 参数：缺省值为60，取值范围是0到100。

swappiness 参数除了在 proc 文件系统下有个全局的值外，在每个 Memory Cgroup 控制组里也有一个 memory.swappiness，那它们有什么不同呢？

不同就是每个 Memory Cgroup 控制组里的 swappiness 参数值为 0 的时候，就可以让控制组里的内存停止写入 Swap。这样一来，有了 memory.swappiness 这个参数后，需要使用 Swap 和不需要 Swap 的容器就可以在同一个宿主机上同时运行了，这样对于硬件资源的利用率也就更高了。

思考题

在一个有 Swap 分区的节点上用 Docker 启动一个容器，对它的 Memory Cgroup 控制组设置一个内存上限 N，并且将 memory.swappiness 设置为 0。这时，如果在容器中启动一个不断读写文件的程序，同时这个程序再申请 $1/2N$ 的内存，请你判断一下，Swap 分区中会有数据写入吗？

欢迎在留言区分享你的收获和疑问。如果这篇文章让你有所收获，也欢迎分享给你的朋友，一起交流和学习。

提建议

© 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 09 | Page Cache: 为什么我的容器内存使用量总是在临界点?

下一篇 11 | 容器文件系统: 我在容器中读写文件怎么变慢了?

精选留言 (8)

写留言



伟平

2020-12-07

k8s下容器貌似不能用swap

展开

作者回复: kubelet缺省不能在打开swap的节点上运行。

配置 "failSwapOn: false" 参数, kubelet可以在swap enabled的节点上运行。



3



Khirye

2020-12-11

想问下老师, 对于最近k8s宣布放弃docker有什么看法?

作者回复: 对于k8s来说, 的确没有必要再用docker了。

这是我们组之前做的从docker切换到containerd的技术分享:

https://www.infoq.cn/article/odslclsjvo8bnx*mbrbk



1



po

2020-12-15

老师, 文章中你说的这两句话好像是矛盾的, swappiness设置为0的时候, 到底会不会回收匿名内存呢?

1. 不过, 这里有一点不同, 需要你留意: 当 `memory.swappiness = 0` 的时候, 对匿名页的回收是始终禁止的, 也就是始终都不会使用 Swap 空间。这时 Linux 系统不会再去比较 free 内存和 zone 里的 high water mark 的值, 再决定一个 Memory Cgroup 中的匿...

展开 ▾

作者回复: 1. memory.swappiness 是指cgroup中的参数
2. 这里说的是/proc/sys/vm/swappiness

1



haha

2020-12-15

所以对于开启了swap，且swap空间够大的前提下，goup的mem limit没用咯？

作者回复: 是的，如果只是设置 memory.limit_in_bytes



谢哈哈

2020-12-07

已经设置memory.swappiness参数，全局参数swappiness参数失效，那么容器里就不能使用swap了

作者回复: @谢哈哈
是的



Geek4329

2020-12-07

感谢老师的分享，学习的越多，越感到自己的自信 🐱

学完这个课程，并完全吸收的话，是不是可以说自己接近精通容器技术了 😊

作者回复: @Geek4329
很高兴我的分享对你有帮助！



莫名

2020-12-07

Memory Cgroup 参数 `memory.swappiness` 起到局部控制的作用，前提是节点开启了 `swap` 功能，不然无论如何设置都无济于事。

展开 ▾

作者回复: @莫名, 没错



kimoti
2020-12-07

既然`memory.swappiness`设置为0了,Swap分区是不会有数据写入的。

作者回复: @kimoti
是的