基于回写机制的系统调优

1、内存的缓存机制

1.1 为什么要使用缓存

linux 引入缓存机制的主要目的是为了提高磁盘的 I/O 性能，即将一部分磁盘上的数据缓存到内存中。通过缓存能提高 I/O 性能是因为：

1、 CPU 访问内存的速度远远大于访问磁盘的速度

2、 数据一旦被访问，就有可能在短期内再次被访问（临时局部原理）

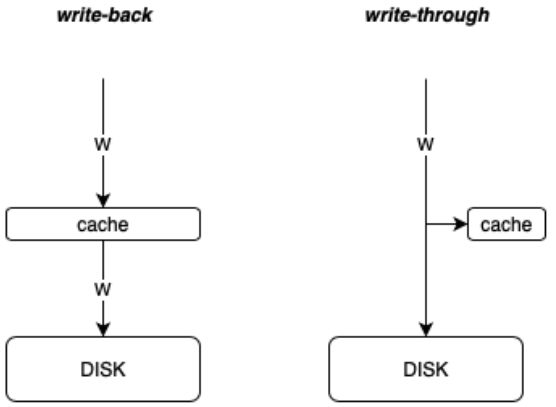
1.2 缓存机制（通常指写缓存）

一般写缓存有三种机制：

1.不缓存(nowrite)：当对缓存中的数据进行写操作时，直接写入磁盘，同时使此数据的缓存失效。

2.透写缓存(write-through cache)：当进行写操作时，会写入缓存中，并且马上再写入磁盘中（写数据时同时更新磁盘和缓存）。在透写缓存技术中，写操作的执行性能与无缓存系统的性能相仿，但读操作使用的时间短。

3.回写(copy-write or write-behind)：这种机制在写操作时，会先将数据写至缓存中，而不会立即写入磁盘中，在一些特定条件或操作，才会把缓存中的数据写入磁盘。



三种策略的优缺点如下：

策略	复杂度	性能
不缓存	简单	缓存只用于读，对于写操作较多的 I/O，性能反而会下降
透写缓存	简单	提升了读性能，写性能反而有些下降
回写	复杂	读写的性能都有提高(机制比较灵活，目前内核中采用的方法)

Linux 下 Buffered IO 默认使用的是 Write back 机制，即将要写入的磁盘数据写至 Page Cache 就返回（CPU 可以处理别的任务），之后 Page Cache 到磁盘的更新操作是异步

进行的。Page Cache 中被修改的内存页称之为脏页（Dirty Page），脏页在特定的时候被一个叫做 pdflush(Page Dirty Flush)的进程写入磁盘。下文会对 Write back 机制做详细阐述。

1.3 磁盘回写机制

当用户试图将数据写入文件的时候，Linux 系统不会立马将数据写入 disk 之中，而是先将它们临时存放在 memory 的一块区域，这块区域的名称是 page cache（页高速缓冲存储器），通过 cat /proc/meminfo 能够查看系统的相关信息。

/proc/meminfo 有两个重要指标，Cached 和 Dirty。Cached 指的是 page cache（大小固定值），Dirty 指的是有数据存于 page cache 中（大小动态变化），被标记的 dirty page 会被加入到一个链表中，等待写入 disk。在下文的几种情况下，dirty page 的数据会被写入至磁盘：

1. 当空闲的内存低于一个特定的阈值时，内核必须将脏页写回磁盘，以便释放内存。
2. 当 dirty page 超过 dirty_background_ratio 的设定值或 dirty_ratio 超过设定值，会进行回写操作
3. 当数据在 dirty page 中的时间超过 dirty_expire_centisecs 秒，内核必须将超时的脏页写回磁盘，以确保脏页不会无限期地驻留在内存。
4. 当用户程序调用了 sync()和 fsync()系统调用（建议在适当的时候进行同步操作，否则可能造成系统阻塞）。

正是因为有 pdflush 进程的服务，使得数据能够及时 flush 到 disk 中，避免了因 memory 吃紧导致系统挂掉的事情发生。

2、文件读取和写入的过程

2.1 文件读取的过程（以 read 为例）

1、 进程调用库函数向内核发起读文件请求；
2、 内核通过检查进程的文件描述符定位到虚拟文件系统的已打开文件列表表项；
3、 调用该文件可用的系统调用函数 read()
4、 read()函数通过文件表项链接到目录项模块，根据传入的文件路径，在目录项模块中检索，找到该文件的 inode；
5、 在 inode 中，通过文件内容偏移量计算出要读取的页；
6、 通过 inode 找到文件对应的 address_space；
7、 在 address_space 中访问该文件的页缓存树，查找对应的页缓存结点：

（1）如果页缓存命中，那么直接返回文件内容；

（2）如果页缓存缺失，那么产生一个页缺失异常，创建一个页缓存页，同时通过 inode 找到文件该页的磁盘地址，读取相应的页填充该缓存页；重新进行第 6 步查找页缓存；

8、 文件内容读取成功。

2.2 文件写入的过程（以 write 为例）

1、进程调用库函数向内核发起读文件请求；

2、内核通过检查进程的文件描述符定位到虚拟文件系统的已打开文件列表表项；

3、调用该文件可用的系统调用函数 write ()

3、write()函数通过文件表项链接到目录项模块，根据传入的文件路径，在目录项模块中检索，找到该文件的 inode；

4、在 inode 中，通过文件内容偏移量计算出要读取的页；

5、通过 inode 找到文件对应的 address_space；

6、在 address_space 中访问该文件的页缓存树，查找对应的页缓存结点：

7、如果页缓存缺失，那么产生一个页缺失异常，创建一个页缓存页，同时通过 inode 找到文件该页的磁盘地址，读取相应的页填充该缓存页。此时缓存页命中，进行第 6 步。

8、一个页缓存中的页如果被修改，那么会被标记成脏页。脏页需要写回到磁盘中的文件块。有两种方式可以把脏页写回磁盘：

（1）手动调用 sync()或者 fsync()系统调用把脏页写回

（2）pdflush 进程会定时把脏页写回到磁盘

同时注意，脏页不能被置换出内存，如果脏页正在被写回，那么会被设置写回标记，这时候该页就被上锁，其他写请求被阻塞直到锁释放。

2.3 相关概念

库函数：应用编程接口的 API，其实就是常见的函数定义，例如 open()和相关的函数。

文件描述符：内核（kernel）利用文件描述符（file descriptor）来访问文件。文件描述符是非负整数。打开现存文件或新建文件时，内核会返回一个文件描述符。读写文件也需要使用文件描述符来指定待读写的文件。文件描述符表中的指针指向 file 结构体。

文件列表表项：每个进程在 PCB（Process Control Block）中都保存着一份文件描述符表，文件描述符就是这个表的索引，每个表项都有一个指向已打开文件的指针，已打开的文件在内核中用 file 结构体表示，文件描述符表中的指针指向 file 结构体。每个 file 结构体都有一个指向 dentry 结构体的指针。而每个 dentry 结构体都有一个指针指向 inode 结构体

目录项：管理路径的目录项。比如一个路径 /home/foo/hello.txt，那么目录项有 home, foo, hello.txt。目录项的块，存储的是这个目录下的所有的文件的 inode 号和文件名等信息。其内部是树形结构，操作系统检索一个文件，都是从根目录开始，按层次解析路径中的所有目录，直到定位到文件。

address_space：address_space 是种数据结构，它是用于管理文件（struct inode）映射到内存的页面(struct page)，一个文件 inode 对应一个地址空间 address_space。而一个 address_space 对应一个页缓存基数树。

页缓存树：页缓存就是将一个文件在内存中的所有物理页所组成的一种树形结构

页缓存结点：树状图中具体的某个页缓存。

文件缓存：文件是指存储在外部存储介质上的、由文件名标识的一组相关信息的集合。由于 CPU 与 I/O 设备间速度不匹配。为了缓和 CPU 与 I/O 设备之间速度不匹配矛盾。文件缓冲区是用以暂时存放读写期间的文件数据而在内存区预留的一定空间。使用文件缓冲区可减少读取硬盘的次数。

Inode： 是文件的唯一标识，一个文件对应一个 inode。通过 inode 可以方便的找到文件在磁盘扇区的位置。同时 inode 模块可链接到 address_space 模块，方便查找自身文件数据是否已经缓存。UNIX 中创建文件系统时，同时将会创建大量的 inode 。通常，文件系统磁盘空间中大约百分之一空间分配给了 inode 表。

目录缓存：目录项高速缓存，是 Linux 为了提高目录项对象的处理效率而设计的；它记录了目录项到 inode 的映射关系。因此，当应用程序发起 stat 系统调用时，就会创建对应的 dentry_cache 项

Buffer Cache：Buffer Cache 用于缓存存储设备块（比如磁盘扇区）的数据，定时或手动的将数据同步到磁盘，避免频繁写磁盘。buffer cache 是对设备数据的缓存。

Page Cache：以页面形式缓存了文件系统的文件，给需要使用的程序读取，它是为了给读操作提供缓冲，避免频繁读硬盘，提高读取效率。page cache 是文件系统层面的对文件数据的缓存。

Dirty page：脏页－linux 内核中的概念，因为硬盘的读写速度远赶不上内存的速度，系统就把读写比较频繁的数据事先放到内存中，以提高读写速度，这就叫高速缓存，linux 是以页作为高速缓存的单位，当进程修改了高速缓存里的数据时，该页就被内核标记为脏页，内核将会在合适的时间把脏页的数据写到磁盘中去，以保持高速缓存中的数据和磁盘中的数据是一致的。

3、磁盘读写参数介绍

3.1 vm.dirty_ratio

vm.dirty_ratio 参数控制文件系统的同步写写缓冲区的大小，单位是百分比，表示当写缓冲使用到系统内存多少的时候（即当文件系统缓存脏页数量达到系统内存百分之多少时，如 10%），系统不得不开始处理缓存脏页（因为此时脏页数量已经比较多，为了避免数据丢失需要将一定脏页刷入外存），此时所有新的 I/O 块都会被阻塞，直到脏数据被写入磁盘。

增大该值会使用更多系统内存用于磁盘写缓冲，通常可以提高系统的写性能。但是，当你需要持续、恒定的写入场合时，应该降低其数值，一般启动上缺省是 10。参数的修改方法如下：

```
echo "3" > /proc/sys/vm/dirty_ratio
```

3.2 vm.dirty_background_ratio

vm.dirty_background_ratio 参数控制文件系统的后台进程在何时刷新磁盘，单位是百分比，表示系统内存的百分比，即当写缓冲使用到系统内存多少的时，会触发 pdflush/flush/kdmflush 等后台回写进程运行，将一定缓存的脏页异步地刷入外存。增大之会使用更多系统内存用于磁盘写缓冲，通常可以提高系统的写性能。但是在需要持续、恒定的写入场合时，应该降低其数值，一般启动上缺省是 5。参数的修改方法如下：

```
echo "5" > /proc/sys/vm/dirty_background_ratio
```

注意：如果 dirty_ratio 设置比 dirty_background_ratio 大，可能认为 dirty_ratio 的触发条件不可能达到，因为每次肯定会先达到
vm.dirty_background_ratio 的条件，然而，确实是先达到
vm.dirty_background_ratio 的条件然后触发 flush 进程进行异步的回写操作，但是这一过程中应用进程仍然可以进行写操作，如果多个应用进程写入的量大于 flush 进程刷出的量那自然会达到 vm.dirty_ratio 这个参数所设定的值，此时操作系统会转入同步地处理脏页的过程，阻塞应用进程。

3.3 vm.dirty_expire_centisecs

vm.dirty_expire_centisecs 参数声明 Linux 内核写缓冲区里面的数据停留多长时间后（指定脏数据能够存活的时间），pdflush 进程就开始考虑写到磁盘中去。单位是 1/100 秒。缺省是 3000（30 秒），系统认为脏数据在内存中停留的时间超过 30 秒，会将脏数据异步的写到磁盘中。对于特别重载的写操作来说，该值应适当缩小。建议设置为 1500，即 15 秒。若系统内存较大，写入模式是间歇式的，且每次写入的数据不大（比如几十 M），那么该值还是大些的好。参数的修改方法如下：

```
echo "1500" > /proc/sys/vm/dirty_expire_centisecs
```

3.4 vm.dirty_writeback_centisecs

vm.dirty_writeback_centisecs 该参数控制内核的脏数据刷新进程 pdflush 的运行间隔。单位是 1/100 秒。缺省数值是 500，也就是 5 秒。如果系统是持续地写入数据，该值应适当调低，通常 vm.dirty_writeback_centisecs 的数值要低于 vm.dirty_expire_centisecs，参数的修改方法如下：

```
echo "100" > /proc/sys/vm/dirty_writeback_centisecs
```

3.5 vm.zone_reclaim_mode

参数 vm.zone_reclaim_mode 只有在启用 CONFIG_NUMA 选项时才有效,

vm.zone_reclaim_mode 参数用来管理当一个内存区域(zone)内部的内存耗尽时，是从其内部进行内存回收还是可以从其他 zone 进行回收的选项

0：意味着关闭 zone_claim 模式，可以从其他 zone 或 NUMA 节点回收内存。

1：打开 zone_claim 模式，这样内存回收只会发生在本地节点内。

2：在本地回收内存时，可以将 cache 中的脏数据回写硬盘，以回收内存。

3：可以用 swap 方式回收内存。

该参数修改方法如下：

echo "2" > /proc/sys/vm/zone_reclaim_mode

4、调优方案

通过调整 vm.dirty_ratio、vm.dirty_background_ratio、vm.dirty_expire_centisecs、vm.dirty_writeback_centisecs 和 vm.zone_reclaim_mode 的参数值，实现对磁盘 IO 性能或数据写入磁盘可靠性的提升。

4.1 读写参数设置方案

根据数据写入磁盘的使用场景，将回写优化的参数从两个方向进行调整：高可靠性和高性能。高可靠性是指系统在收到数据写入请求的时候，能够快速将数据刷入至磁盘，避免在突然断电的场景下，丢失过多的数据。调优方法是降低 vm.dirty_ratio、vm.dirty_background_ratio、vm.dirty_expire_centisecs、vm.dirty_writeback_centisecs 和 vm.zone_reclaim_mode 的参数值,但是 vm.dirty_ratio 的数值不能下降过多，否则会因为 IO 阻塞而降低磁盘读写的性能。

高性能组是磁盘 IO 具有较高的读写速率。调优方法是提高 vm.dirty_ratio、vm.dirty_background_ratio、vm.dirty_expire_centisecs、vm.dirty_writeback_centisecs 和 vm.zone_reclaim_mode 的参数值，但可能在系统突然断电的场景下，数据因没有及时写入磁盘而导致数据大量流失。

默认组是指操作系统设定默认的磁盘读写参数值。

参数	默认组	高可靠组	高性能组
vm.dirty_ratio	10	20	40
vm.dirty_background_ratio	40	3	60
vm.dirty_expire_centisecs	3000	150	60000
vm.dirty_writeback_centisecs	500	100	12000
vm.zone_reclaim_mode	0	2	0

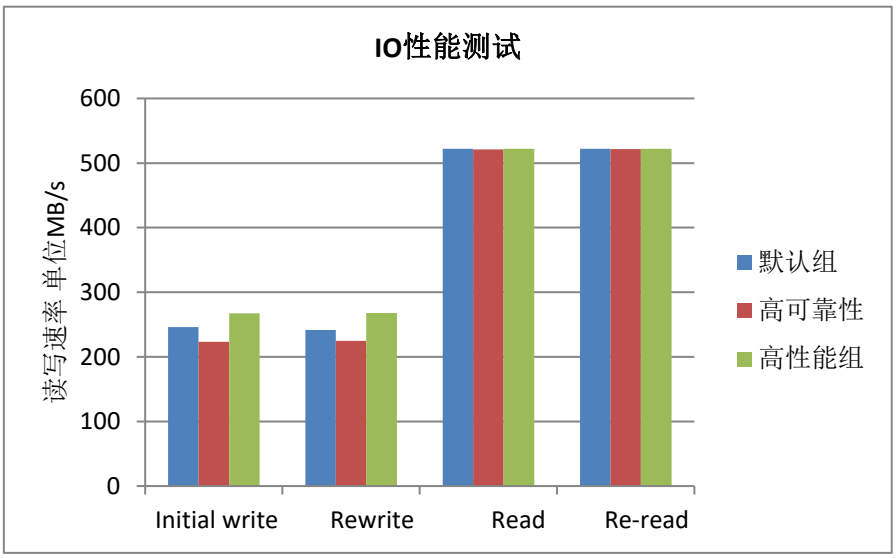
测试软硬件平台：中标麒麟服务器操作系统 5.0 版本，系统内存为 32G，磁盘记录块大小为 4K。

文件系统：xfs

4.2 性能测试

测试工具：iozone 测试工具

测试流程：利用 iozone 测试工具对 4.1 中的三组参数（默认组，高可靠性组，高性能组）进行磁盘读写速率的测试。设定测试文件大小为 64G（系统内存的两倍），测试前需要进行清缓存和 sync 同步操作（保证系统在相同的物理环境下进行性能测试）。测试结果如图一所示：



图一：IO 性能测试比较图

由图一的测试结果可以看出：通过调整磁盘的读写参数，能够提高/降低数据写入至磁盘的速率，但对读性能没有太大的影响。默认组 initial write 的速率为 246.1MB/s，高性能组 initial write 的速率为 267.41MB/s，initial write 的性能提高了约 8.6%；而高可靠性组 initial write 的速率为 223.37MB/s，与默认组进行比较，initial write 的性能下降了约 10.2%。而高性能组和高可靠性组 rewrite 的性能与默认组比较分别提升和下降了约 9%和 10.5%。

Write：测试向一个新文件写入的性能。

Re-write：测试向一个已存在的文件写入的性能。当一个已存在的文件被写入时，所需工作量较少，因为此时元数据已经存在。

Read：测试读一个已存在的文件的性能。

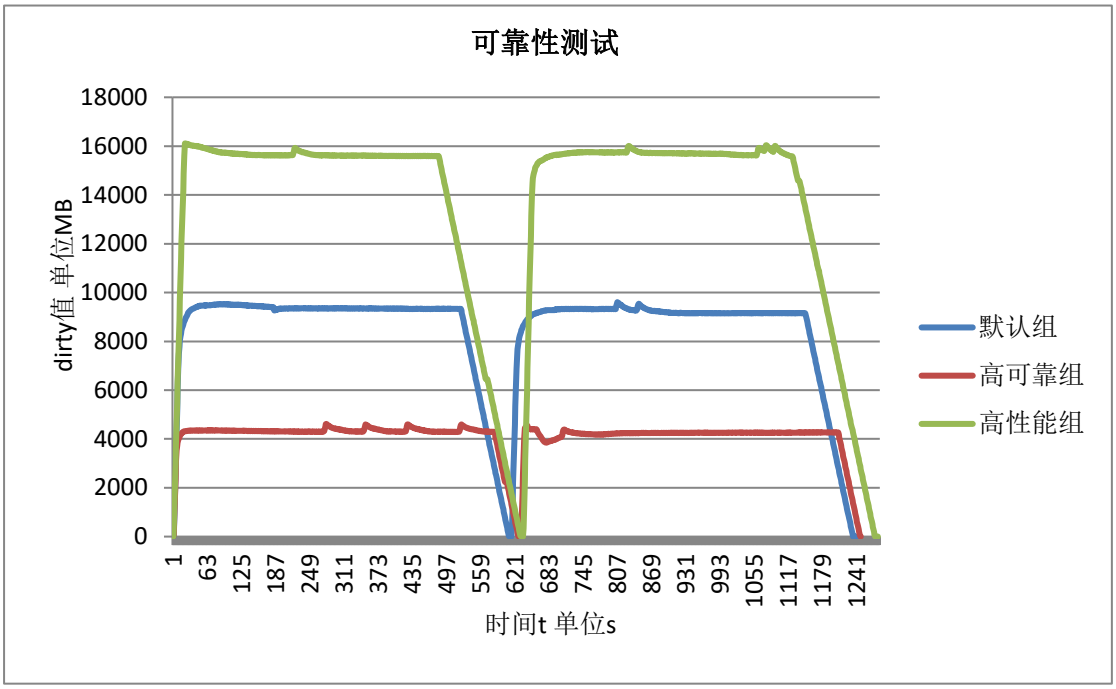
Re-Read： 测试读一个最近读过的文件的性能。Re-Read 性能会高些，因为操作系统通常会缓存最近读过的文件数据。

4.3 可靠性测试

测试工具：自写 shell 脚本

测试流程：测试流程：在运行 iozone 测试工具对 4.1 中的三组参数（默认组，高可靠性组，

高性能组）进行磁盘读写的同时，利用自写 shell 脚本监测在进行磁盘回写的时候，/proc/meminfo 文件下 Dirty 的参数值。图二为在一个 iozone 测试周期内，当前系统中存在的 dirty 值与时间的关系图：



图二：可靠性测试比较图

由图二的测试结果可以看出：通过调整磁盘的读写参数，能够提高/降低系统在当前时刻中，dirty 的数值。在 250s 和 990s 时刻，系统中默认组 dirty 的数值接近 9350MB，系统中高性能组 dirty 的数值接近 15630MB，dirty 的数值提高了约 67%；而系统中高可靠性组 dirty 的数值接近 4300MB，与默认组进行比较，dirty 的数值降低了约 117.4%。dirty 为当前要写入磁盘但没有写至磁盘的数据。如果在 250s 和 990s 时刻突然断电，对于高性能组而言，可能会有超过 15G 的数据因为没写入至磁盘而造成数据的丢失（要写入的磁盘数据量为 64G，即超过 23%的数据丢失），这在生产中是非常严重现象。而高可靠性组可能会丢失超过 4G，接近总量 6%的数据，能够通过磁盘修复的方式将数据进行恢复，在实际使用中是比较安全的。

结论：通过调整磁盘的读写参数，能够提高数据写入至磁盘的性能和可靠性。对于运行系统稳定的生产环境，建议使用高性能组的磁盘读写参数；对于运行环境不稳定或者可能发生突然断电的生产环境，建议使用高可靠性组的磁盘读写参数。