

推荐 热点 视频 图片 段子 社会 娱乐 科技 汽车 体育 财经 军事 国际 时尚 旅游 更多

3

今日头条

首页 / 技术 / 正文

大家

登录 问答 头条号

微博

Qzone

微信

Kafka-4614问题复盘

(MappedByteBuffer未关闭导致慢磁盘访问)

达人科技 2017-03-28 23:17

很早之前就想动笔就这个kafka bug总结一番了，只是这个问题既不是本人发现，也不是自己动手修复，终归是底气不足，故而一直耽搁下来。怎奈此问题实在是含金量十足，又恰逢最近有人询问Kafka 0.10.2都有哪些提升，我终究还是决定给这个bug写点东西了。

事先声明：这是一个日本人(下称Yuto)开的bug，其对问题的描述、定位、探查、分析、验证以及结论都堪称完美，令人印象深刻。自该issue报出的第一天起我便全程追踪其进度，整个过程甚觉受益良多，今总结出来希望对自己及各位读者都有所帮助。值得一提的是，我只会写出亲自验证了的结论，对于该issue中阐述的一些未经验证的观点，不会显式强调。有兴趣的读者可以查看issue原文自己去判断。附上bug地址：

https://issues.apache.org/jira/browse/KAFKA-4614

环境背景

操作系统：CentOS 6

内核版本：2.6.32-xx

文件系统：XFS

Java版本：1.8.0_66

GC: G1

Kafka版本：0.10.0.1

问题描述

生产环境中有一半左右的PRODUCE请求平均响应时间是1ms，99%请求的响应时间指标是10ms，但有时99%请求响应时间指标会飙升至100ms~500ms，且绝大多数情况下时间都花在了“Remote”部分，即意味着是由于备份机制慢造成的。另外，每天都会在不同的broker上稳定地观察到3~5次，同时伴随着Remote的增加，Total和RequestQueue也会相应地增加，就像下图这样：

上面提到的Remote、Total和RequestQueue分别表示RemoteTimeMs、TotalTimeMs和RequestQueueTimeMs这3个JMX指标，即副本备份时间(设置了acks=-1)、请求处理总时间以及请求在队列中的等待时间。具体来说，remote time就是leader broker等待所有follower副本拉取消息的时间。显然，只有在acks=-1下才会有remote time；RequestQueueTimeMs指客户端发过来的PRODUCE请求在broker端的请求队列中待的时间，由于请求队列是使用阻塞队列实现的，所以如果该值过大的话，说明broker端负载过大，需要增大请求队列的大小，即增加参数queued.max.requests的值。

另外，在上述三个指标飙升的同时还观测到broker所在磁盘的读操作指标也跟着飙升。

定位问题

¥57.00 ¥129 销量

学点Groovy来理解build.grac

java操作Access数据库文件工

如何在Ubuntu

vue.js展示AJAX数据简单示例

¥98 销

热门视频

马云台上开玩笑，台下跟说吓得他赶紧一本正经演讲

赌王儿子开直播发现粉丝送制止，呼吁孝敬父母做慈善

当裁判掏错牌！看到小罗和笑，我承认我也笑了

http://www.toutiao.com/i6402570140067561985/?tt_from=weixin&utm_campaign=client_share&from=timeline&app=news_article&utm_source=weixin&isa... 1/6



微博



Qzone



微信

接下来Yuto开始分析了：按说PRODUCE请求应该只会写磁盘，为什么磁盘的读操作指标会这么高呢？另外生产环境中的consumer几乎没有太多滞后，consumer请求的消息数据几乎全部命中操作系统的页缓存，为什么还会“狂”读磁盘呢？看来目前的主要问题就是要弄明白是哪个线程在读磁盘。于是Yuto使用了SystemTap，stap脚本如下

```
global target_pid = KAFKA_PID global target_dev = DATA_VOLUME probe
ioblock.request { if (rw == BIO_READ && pid == target_pid && devname ==
target_dev) { t_ms = gettimeofday_ms + 9 * 3600 * 1000 // timezone adjustment
printf("%s,%03d: tid = %d, device = %s, inode = %d, size = %d\n", ctime(t_ms / 1000),
t_ms % 1000, tid, devname, ino, size) print_backtrace print_ubacktrace } }
```

不了解SystemTap也没关系，上面的脚本就是打印Kafka进程读磁盘时内核态和用户态的栈trace，输出如下：

Thu Dec 22 17:21:39 2016,209: tid = 126123, device = sdb1, inode = -1, size = 4096 0xffffffff81275050 : generic_make_request+0x0/0x5a0 [kernel]

0xffffffff81275660 : submit_bio+0x70/0x120 [kernel]

0xffffffffa036bcaa : _xfs_buf_ioapply+0x16a/0x200 [xfs]

0xffffffffa036d95f : xfs_buf_iorequest+0x4f/0xe0 [xfs]

0xffffffffa036db46 : _xfs_buf_read+0x36/0x60 [xfs]

0xffffffffa036dc1b : xfs_buf_read+0xab/0x100 [xfs]

0xffffffffa0363477 : xfs_trans_read_buf+0x1f7/0x410 [xfs]

0xffffffffa033014e : xfs_btree_read_buf_block+0x5e/0xd0 [xfs]

0xffffffffa0330854 : xfs_btree_lookup_get_block+0x84/0xf0 [xfs]

0xffffffffa0330edf : xfs_btree_lookup+0xbf/0x470 [xfs]

0xffffffffa032456f : xfs_bmbt_lookup_eq+0x1f/0x30 [xfs]

0xffffffffa032628b : xfs_bmap_del_extent+0x12b/0xac0 [xfs]

0xffffffffa0326f34 : xfs_bunmap+0x314/0x850 [xfs]

0xffffffffa034ad79 : xfs_itruncate_extents+0xe9/0x280 [xfs]

0xffffffffa0366de5 : xfs_inactive+0x2f5/0x450 [xfs]

0xffffffffa0374620 : xfs_fs_clear_inode+0xa0/0xd0 [xfs]

0xffffffff811affbc : clear_inode+0xac/0x140 [kernel]

0xffffffff811b0776 : generic_delete_inode+0x196/0x1d0 [kernel]

0xffffffff811b0815 : generic_drop_inode+0x65/0x80 [kernel]

0xffffffff811af662 : iput+0x62/0x70 [kernel]

0x37ff2e5347 : munmap+0x7/0x30 [/lib64/libc-2.12.so]

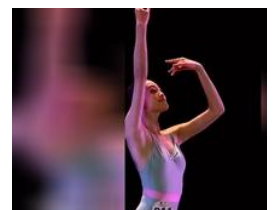
0x7ff169ba5d47 : Java_sun_nio_ch_FileChannelImpl_unmap0+0x17/0x50
[/usr/jdk1.8.0_66/jre/lib/amd64/libnio.so]

关于这段输出，Yuto并没有给出过多的解释。不过我特意标红了第一行和最后两行：

- 第一行中最重要的就是tid信息，即线程ID。我们需要记下这个ID：**126123**
- 最后这两行告诉我们目前Kafka进程下某个读磁盘的线程正在执行munmap系统调用试图删除某段地址空间的内存映射。虽然进程结束时该映射会被自动取消，但如果只是在程序中关闭了文件部署符(file descriptor)——比如调用File.close，那么这段映射是不会自动被移除的。被



来听听当地人是怎么读的，
 的正宗读音，最喜欢听玛莎



舞蹈之美，请用艺术的眼光



相关头条号



LaTeXila简介：Linux上的多
 编辑器

NMAP 常用扫描简介（一）



放弃编程的三个理由

程序员最喜欢的五大神器



微博



Qzone



微信

标红的第二行显示出了是哪段Java代码执行的这个系统调用。从输出中我们已知是由Java的FileChannelImpl的unmap方法发出的。

我们平时可能用到FileChannel的机会比较多，知道可以利用它来操作文件，但如果查询JDK官网API的话，你会发现它实际上是个抽象类，也就是说底层的实现是由子类提供的——这个子类就是这里的sun.nio.ch.FileChannelImpl。从包名来看，这并不是对外开放的Java类，应该是JVM内部的类。实际上，该类用到的一些其他类其实并不是所有JVM都有的，所以不鼓励用户直接使用这个类。

分析原因

Okay，既然我们已经知道Kafka进程下的某个线程正在执行大量读磁盘操作，并且是在运行mmap操作。那么下面该如何定位问题呢？Yuto想到了jstack dump，相关输出如下：

```
$ grep 0x1ecab /tmp/jstack.dump "Reference Handler" #2 daemon prio=10
os_prio=0 tid=0x00007ff278d0c800 nid=0x1ecab in Object.wait
[0x00007ff17da11000]
```

为什么是这一行？因为126123（还记得吧？就是刚才得到的线程ID）的16进制就是0x1ecab，即上面输出中的nid (native thread ID)。同时，还可以看到这个线程的名字是Reference Handler——这个线程就是Finalizer线程。当某个对象被标记为不可达之后，JVM会再给它一次机会让它重新关联上其他的对象从而“逃脱”被GC的命运，在这个过程中JVM会将该对象放入到一个特定的队列中，并由一个特定的Finalizer线程去查看这个对象是否真的不再被使用了。这个线程就是Reference Handler。在这里我们只需要知道这是个GC线程就可以了。

问题的定位似乎更进一步了！我们已经知道是因为GC线程导致的读操作从而引发的请求变慢。那么接下来查询一下GC日志便顺理成章了：

```
$ grep --text 'Total time for which application threads were stopped' kafkaServer-
gc.log | grep --text '11T01:4' | perl -ne '/^(.*T\d{2}:\d{2}).*stopped: ([0-9\.]+)/;
${$1} += $2 * 1000; END { print "$_ = ${$1}\n" for sort keys %h }'
2017-01-11T01:40 = 332.0601
2017-01-11T01:41 = 315.1805
2017-01-11T01:42 = 318.4317
2017-01-11T01:43 = 317.8821
2017-01-11T01:44 = 302.1132
2017-01-11T01:45 = 950.5807
2017-01-11T01:46 = 344.9449
2017-01-11T01:47 = 328.936
2017-01-11T01:48 = 296.3204
2017-01-11T01:49 = 299.2834
```

从上面的输出中可以看出GC STW的时间非常抖动，特别是有一段时间内的停顿耗时异常地高，再一次证明Kafka broker端请求处理能力的下降是因为GC的原因导致的。

其实问题梳理到这里，理论上已经可以丢给Kafka团队去让他们去解决了，但日本人的严谨实在令人佩服。这位Yuto先生并没有停下脚步，而是继续开始追查更深入的原因。还记得之前我们说过的munmap吗？我们可以沿着这条路继续追查下去。

首先，Kafka代码中哪部分用到了映射内存文件呢？答案就是索引文件！Kafka日志的索引文件都是使用MappedByteBuffer来实现的。后台日志删除线程会定期删除过期日志及其对应的索引文件。0.10.2之前的代码就是简单调用File.close把底层索引文件删除掉，但这样做其实并没有删除映射的内存区域。事实上，当调用完File.close之后，下面情况会依次发生：

1. 索引文件仅仅被VFS(虚拟文件系统)标记为“已删除”。由于对应的映射对象没有被清除，所以依然有引用指向该文件，故物理文件块仍然在磁盘上
2. Kafka的索引对象OffsetIndex在JVM中变为不可达，此时可以被GC收集器回收
3. GC线程开始工作清除MappedByteBuffer对象，并执行对应的删除映射操作

科锐广告



这样的飞机乘客让人讨厌，
仁的素质



镜头中的90年代武汉供电



13岁少年大肚子6年如怀孕
每天贴画为儿筹钱



武当这座房子靠“雷击”翻
灯500年不灭





微博



Qzone



微信

关于第三点，我们已经可以从之前的jstack dump中得到验证。jstack清晰地表明就是由GC线程来执行munmap的。okay，搞清楚了这些，现在的问题就要弄明白为什么munmap需要花费这么长的时间？以下就是Yuto的分析：

1. OffsetIndex对象通常都会在堆上保留很长时间(因为日志段文件通常都会保留很长时间)，所以该对象应该早就被提升到老年代了
2. OffsetIndex.delete操作将文件标记为“已删除”，VFS稍后会更新inode缓存
3. 由于G1收集器是以region为单位收集的，且它通常会挑选包含了最多“垃圾”的region进行收集，所以包含mmap对象的region通常都不属于这类region，从而有可能很长时间都不会被收集
4. 在没有被收集的这段时间内，inode缓存被更新，正式把标记为“已删除”的那个索引文件元数据信息“踢出”缓存
5. 最终当GC开始回收mmap对象并调用munmap时，内核就需要执行物理删除IO操作。但此时，该索引文件的元数据信息已经从缓存中被剔除，因此文件系统就必须要从磁盘中读取这些信息。而同一时刻Kafka磁盘正在用于高速写操作(producer还在飞速运行着)，所以读磁盘操作就遭遇竞争从而拉长整体GC时间。对Kafka broker来说，表现为所有Kafka线程全部暂停，极大地拉长了请求的平均响应时间

验证

不得不说，上面的这几点分析实在太精彩了。后面Yuto为了验证这一套分析结果还特意写了一段Java程序来模拟这个问题。由于代码太长，我就不贴了，具体思想就是使用dd命令先让磁盘忙起来，然后运行Java程序模拟mapped文件被删除并从VFS的inode缓存中被移除的情况，最后成功地复现了这个问题。SystemTap脚本输出和jstack结果都证明了GC线程执行这个逻辑花费了大量的时间。而原因就是VFS正在从磁盘中读取文件元数据信息。

问题解决

其实讲到这里该问题的解决办法已然清晰明了了——就是在删除索引文件的同时还取消对应的内存映射，删除mapped对象。不过令人遗憾的是，Java并没有特别好的解决方案——令人有些惊讶的是，Java没有为MappedByteBuffer提供unmap的方法，该方法甚至要等到Java 10才会被引入，详见JDK的这个bug：JDK-4724038不过Java倒是提供了内部的“临时”解决方案——DirectBuffer.cleaner.clean 切记这只是临时方法，毕竟该类在Java9中就正式被隐藏了，而且也不是所有JVM厂商都有这个类。

还有一个解决办法就是显式调用System.gc，让gc赶在cache失效前就进行回收。不过坦率地说，这个方法弊端更多：首先显式调用GC是强烈不被推荐使用的，其次很多生产环境甚至禁用了显式GC调用，所以这个办法最终没有被当做这个bug的解决方案。

结语

最后这个bug的修复几乎得到了所有Kafka主要committer的赞扬，其结果也非常喜人，以下是Yuto先生的反馈：

那条粗线代表bug的修复点。如图所示，在那之后请求的平均响应时间非常稳定，再也没有毛刺的现象出现，这足以说明这次修复是有效的。纵观整个bug的修复过程，Yuto对于Kafka索引文件机制、JVM GC和操作系统VFS都有很深的造诣，我自己也在这个过程中学到了很多。今总结出来以供大家讨论。最后贴一份Kafka committer Ijuma关于此bug的评论，足见此问题的分量：



Ismael Juma @ijuma · 2月3日

99th percentile latency improvement in the upcoming Kafka 0.10.2.0:

issues.apache.org/jira/browse/KA...







两轮自平衡小车套件 STM32控

¥398.00





淘广告

3 条评论



写下您的评论...

评论

 用户3569283524 5小时前

kafka0.10.1可以直接升级到0.10.2吗？

回复

0 个喜欢 1 个回复

 me低调131907534 5小时前

干货满满，赞

回复

4 个喜欢 1 个回复

 醉无吟 4小时前

牛笔

回复

0 个喜欢 1 个回复

相关推荐

推荐

热点

社会

娱乐

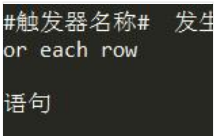
科技

体育



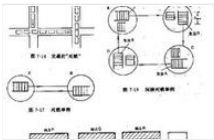
php安全编程——python测试实例编写

i春秋 · 1评论 · 刚刚 相关 不感兴趣 x



MySQL—触发器基础

小海dedede · 1评论 · 刚刚 相关 不感兴趣 x



Java多线程——死锁锁锁锁.....

拥慌洛 · 0评论 · 刚刚 相关 不感兴趣 x



全面的java入门学习笔记总结

Java学习 · 0评论 · 刚刚 相关 不感兴趣 x



一台电脑里，已经有几块硬盘从C分配到Z，再接一块硬盘怎样分配盘符？

头条问答 · 5评论 · 刚刚 相关 不感兴趣 x



Docker Compose搭建mysql主从复制

程序猿和程序媛的故事 · 1评论 · 刚刚 相关 不感兴趣 x

java操作Access数据库文件工具类

达人科技 · 0评论 · 刚刚 相关 不感兴趣 x




MySQL整体架构与内存结构


ITPUB · 3评论 · 刚刚 相关 不感兴趣 x


vue.js展示AJAX数据简单示例


达人科技 · 0评论 · 刚刚 相关 不感兴趣 x

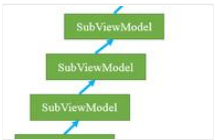


 3

 微博

 Qzone

 微信



Unity 3D Framework Designing(5)——ViewModel之间如何共享数据

达人科技 · 0评论 · 刚刚 [相关](#)

不感兴趣 x



C语言指针变量的运算

晨夕Sama · 2评论 · 刚刚 [相关](#)

不感兴趣 x



开源一个vue2的tree组件

科技优家 · 1评论 · 刚刚 [相关](#)

不感兴趣 x

如何在Ubuntu

达人科技 · 1评论 · 刚刚 [相关](#)

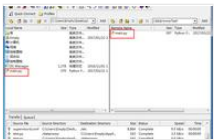
不感兴趣 x



排序算法之冒泡和快排

科技优家 · 1评论 · 刚刚 [相关](#)

不感兴趣 x



python服务器环境搭建（3）——参数配置

达人科技 · 1评论 · 刚刚 [相关](#)

不感兴趣 x

Vue.js 插件开发详解

科技优家 · 0评论 · 刚刚 [相关](#)

不感兴趣 x

清除Tomcat缓存

java探讨 · 0评论 · 刚刚 [相关](#)

不感兴趣 x



多线程之-并发任务间交换数据

风吹走了乌云 · 6评论 · 刚刚 [相关](#)

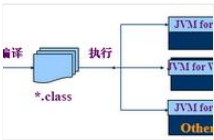
不感兴趣 x



多线程之不共享数据和共享数据

西城旧梦梦旧人 · 2评论 · 刚刚 [相关](#)

不感兴趣 x



java的特点跨平台原理以及JDK的安装

科技新鲜事 · 1评论 · 刚刚 [相关](#)

不感兴趣 x

