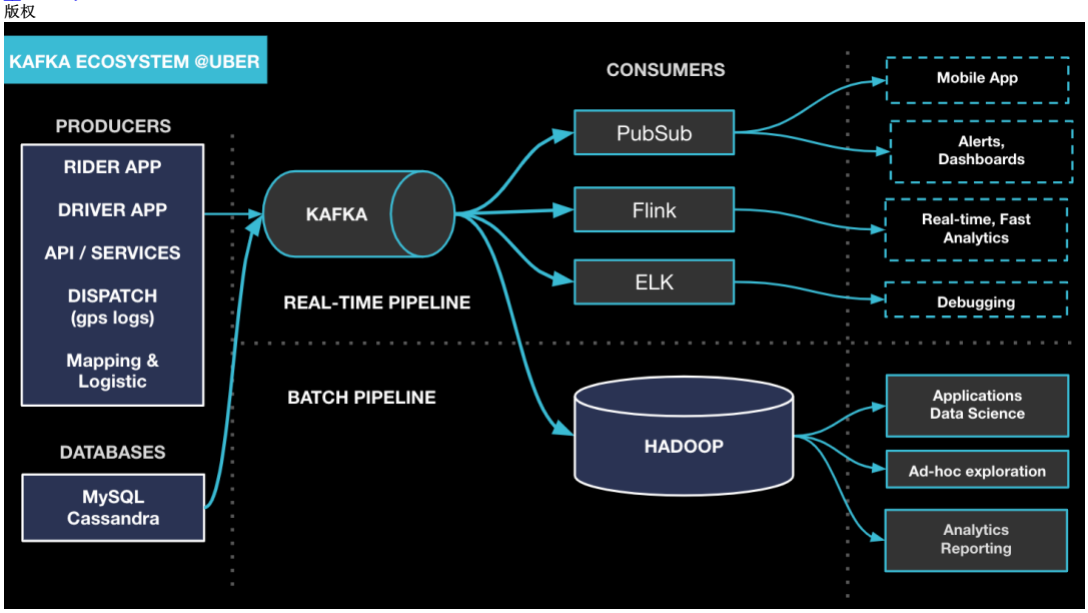
经典面试|为何Kafka这么"快"？

转载

公众号：极客重生 于 2022-01-16 21:54:57 发布 20 收藏
文章标签： 网络 队列 大数据 面试 java
原文链接：https://mp.weixin.qq.com/s?__biz=MzkyMTIzMTkzNA==&mid=2247568068&idx=1&sn=9946d0ded59161eee1598460702f620a&chksm=c1853d95f6f2b483090fc607ce23ac4e7fd7029acf2272a54d5f0f1756966ba7976e0959e4fc&scene=126&&sessionid=
版权



大家好，今天让我们一起来深入了解 **Kafka “快”的内部秘密**。你不仅可以学习到 Kafka 性能优化的各种手段，也可以提炼出各种**性能优化的方法论**，这些方法论也可以应用到我们自己的项目之中，助力我们写出**高性能的项目**。

Kafka 性能优化策略：

- 磁盘读写优化
- 零拷贝优化
- 基于NIO网络模型优化
- 文件数据结构设计优化
- 并行和可扩展优化
- 数据批量与压缩优化
- 无锁轻量级 offset优化

正文

关公战秦琼

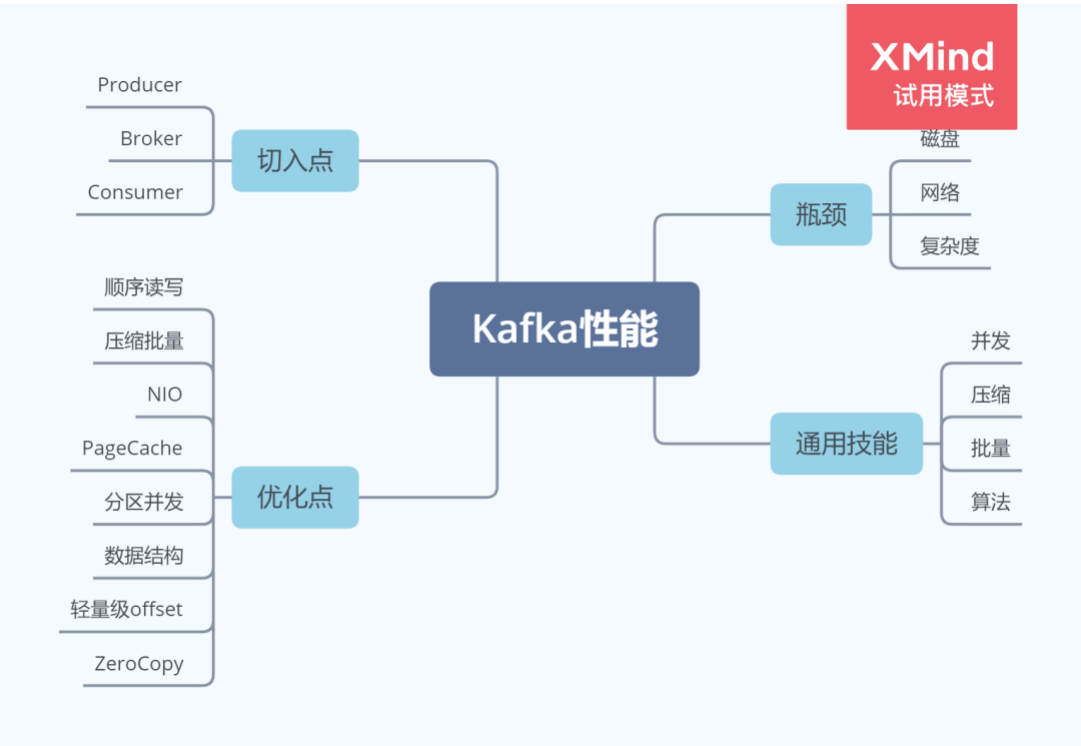
> “
> 65: Redis 和 Kafka 完全是不同作用的中间件，有比较性吗？
> ”

是的，所以此文讲的不是《分布式缓存的选型》，也不是《分布式中间件对比》。我们聚焦于这两个不同领域的项目对性能的优化，看一看优秀项目对性能优化的通用手段，以及在针对不同场景下的特色的优化方式。

很多人学习了很多东西，了解了很多框架，但在遇到实际问题时，却常常会感觉到知识不足。这就是没有将学习到的知识体系化，没有从具体的实现中抽象出可以行之有效的方法论。

学习开源项目很重要的一点就是归纳，将不同项目的优秀实现总结出方法论，然后演绎到自我的实践中去。

Kafka 性能全景



从高度抽象的角度来看，性能问题逃不出下面三个方面：

网络

磁盘

复杂度

对于 Kafka 这种网络分布式队列来说，网络和磁盘更是优化的重中之重。针对于上面提出的抽象问题，解决方案高度抽象出来也很简单：

并发

压缩

批量

缓存

算法

知道了问题和思路，我们再来看看，在 Kafka 中，有哪些角色，而这些角色就是可以优化的点：

Producer

Broker

Consumer

是的，所有的问题，思路，优化点都已经列出来了，我们可以尽可能的细化，三个方向都可以细化，如此，所有的实现便一目了然，即使不看 Kafka 的实现，我们自己也可以想到一二点可以优化的地方。

这就是思考方式。提出问题 > 列出问题点 > 列出优化方法 > 列出具体可切入的点 > `tradeoff`和细化实现。

现在，你也可以尝试自己想一想优化的点和方法，不用尽善尽美，不用管好不好实现，想一点是一点。

> “
> 65 哥：不行啊，我很笨，也很懒，你还是直接和我说吧，我白嫖比较行。
> ”

顺序写

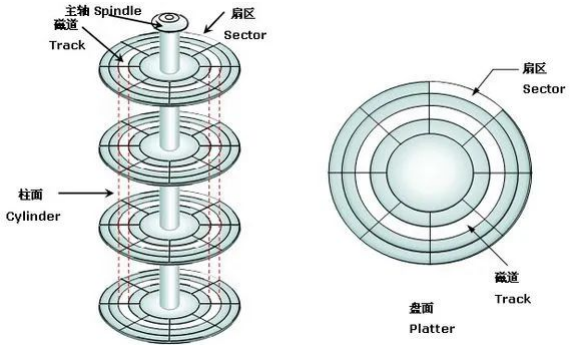
> “
> 65 哥：人家 Redis 是基于纯内存的系统，你 kafka 还要读写磁盘，能比？
> ”

为什么说写磁盘慢？

我们不能只知道结论，而不知其所以然。要回答这个问题，就得回到在校时我们学的操作系统课程了。65 哥还留着课本吗？来，翻到讲磁盘的章节，让我们回顾一下磁盘的运行原理。

> “
> 65 哥：鬼还留着哦，课程还没上到一半书就没了。要不是考试俺眼神好，估计现在还没毕业。
> ”

看经典大图：



完成一次磁盘 IO，需要经过寻道、旋转和数据传输三个步骤。

影响磁盘 IO 性能的因素也就发生在上面三个步骤上，因此主要花费的时间就是：

寻道时间：Tseek 是指将读写磁头移动至正确的磁道上所需要的时间。寻道时间越短，I/O 操作越快，目前磁盘的平均寻道时间一般在 3-15ms。

旋转延迟：Trotation 是指盘片旋转将请求数据所在的扇区移动到读写磁盘下方所需要的时间。旋转延迟取决于磁盘转速，通常用磁盘旋转一周所需时间的 1/2 表示。比如：7200rpm 的磁盘平均旋转延迟大约为 60*1000/7200/2 = 4.17ms，而转速为 15000rpm 的磁盘其平均旋转延迟为 2ms。

数据传输时间：Ttransfer 是指完成传输所请求的数据所需要的时间，它取决于数据传输率，其值等于数据大小除以数据传输率。目前 IDE/ATA 能达到 133MB/s，SATA II 可达到 300MB/s 的接口数据传输率，数据传输时间通常远小于前两部分消耗时间。简单计算时可忽略。

因此，如果在写磁盘的时候省去寻道、旋转可以极大地提高磁盘读写的性能。

Kafka 采用顺序写文件的方式来提高磁盘写入性能。顺序写文件，基本减少了磁盘寻道和旋转的次数。磁头再也不用在磁道上乱舞了，而是一路向前飞速前行。

Kafka 中每个分区是一个有序的，不可变的消息序列，新的消息不断追加到 Partition 的末尾，在 Kafka 中 Partition 只是一个逻辑概念，Kafka 将 Partition 划分为多个 Segment，每个 Segment 对应一个物理文件，Kafka 对 segment 文件追加写，这就是顺序写文件。

> “
> 65 哥：为什么 Kafka 可以使用追加写的方式呢？
> ”

这和 Kafka 的性质有关，我们来看看 Kafka 和 Redis，说白了，Kafka 就是一个`Queue`，而 Redis 就是一个`HashMap`。`Queue`和`Map`的区别是什么？

`Queue` 是 FIFO 的，数据是有序的；`HashMap`数据是无序的，是随机读写的。Kafka 的不可变性，有序性使得 Kafka 可以使用追加写的方式写文件。

其实很多符合以上特性的数据系统，都可以采用追加写的方式来优化磁盘性能。典型的有`Redis`的 AOF 文件，各种数据库的`WAL(Write ahead log)`机制等等。

> “
> 所以清楚明白自身业务的特点，就可以针对性地做出优化。
> ”

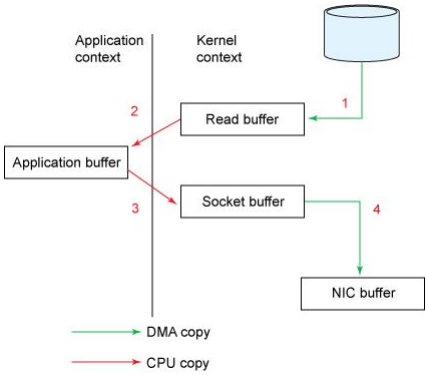
零拷贝

> “
> 65 哥：哈哈，这个我面试被问到过。可惜答得一般般，唉。
> ”

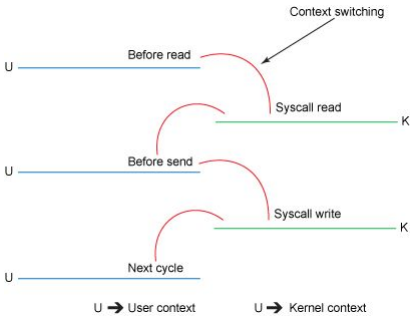
什么是零拷贝？

我们从 Kafka 的场景来看，Kafka Consumer 消费存储在 Broker 磁盘的数据，从读取 Broker 磁盘到网络传输给 Consumer，期间涉及哪些系统交互。Kafka Consumer 从 Broker 消费数据，Broker 读取 Log，就使用了 sendfile。如果使用传统的 IO 模型，伪代码逻辑就如下所示：

```
readFile(buffer)
send(buffer)
```



如图，如果采用传统的 IO 流程，先读取网络 IO，再写入磁盘 IO，实际需要将数据 Copy 四次。



第一次：读取磁盘文件到操作系统内核缓冲区；

第二次：将内核缓冲区的数据，copy 到应用程序的 buffer；

第三步：将应用程序 buffer 中的数据，copy 到 socket 网络发送缓冲区；

第四次：将 socket buffer 的数据，copy 到网卡，由网卡进行网络传输。

> “
> 65 哥：啊，操作系统这么傻吗？copy 来 copy 去的。
> ”

并不是操作系统傻，操作系统的设计就是每个应用程序都有自己的用户内存，用户内存和内核内存隔离，这是为了程序和系统安全考虑，否则的话每个应用程序内存满天飞，随意读写那还得了。

不过，还有零拷贝技术，英文——`Zero-Copy`。零拷贝就是尽量去减少上面数据的拷贝次数，从而减少拷贝的 CPU 开销，减少用户态内核态的上下文切换次数，从而优化数据传输的性能。

常见的零拷贝思路主要有三种：

直接 I/O：数据直接跨过内核，在用户地址空间与 I/O 设备之间传递，内核只是进行必要的虚拟存储配置等辅助工作；

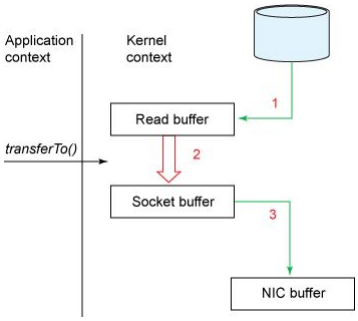
避免内核和用户空间之间的数据拷贝：当应用程序不需要对数据进行访问时，则可以避免将数据从内核空间拷贝到用户空间；

写时复制：数据不需要提前拷贝，而是当需要修改的时候再进行部分拷贝。

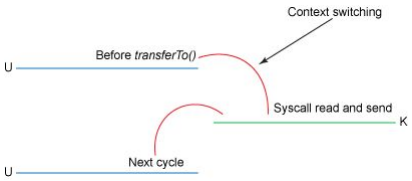
Kafka 使用到了 `mmap` 和 `sendfile` 的方式来实现零拷贝。分别对应 Java 的 `MappedByteBuffer` 和 `FileChannel.transferTo`。

使用 Java NIO 实现零拷贝，如下：

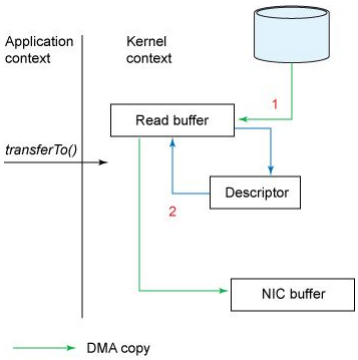
```
FileChannel.transferTo()
```



在此模型下，上下文切换的数量减少到一个。具体而言，`transferTo()`方法指示块设备通过 DMA 引擎将数据读取到读取缓冲区中。然后，将该缓冲区复制到另一个内核缓冲区以暂存到套接字。最后，套接字缓冲区通过 DMA 复制到 NIC 缓冲区。



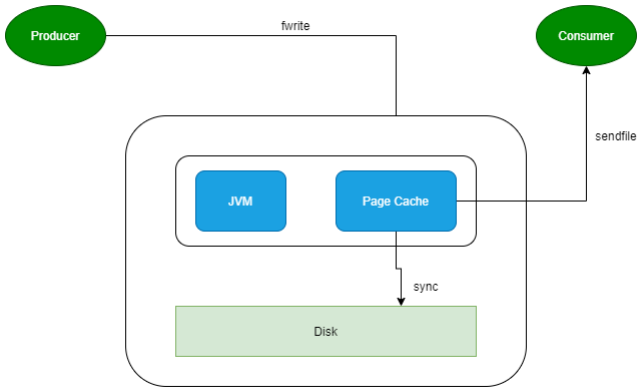
我们将副本数从四减少到三，并且这些副本中只有一个涉及 CPU。我们还将上下文切换的数量从四个减少到了两个。这是一个很大的改进，但是还没有查询零副本。当运行 Linux 内核 2.4 及更高版本以及支持收集操作的网络接口卡时，后者可以作为进一步的优化来实现。如下所示。



根据前面的示例，调用`transferTo()`方法会使设备通过 DMA 引擎将数据读取到内核读取缓冲区中。但是，使用`gather`操作时，读取缓冲区和套接字缓冲区之间没有复制。取而代之的是，给 NIC 一个指向读取缓冲区的指针以及偏移量和长度，该偏移量和长度由 DMA 消除。CPU 绝对不参与复制缓冲区。

关于零拷贝详情，可以详读这篇文章零拷贝 (Zero-copy) 浅析及其应用。

PageCache



producer 生产消息到 Broker 时，Broker 会使用 pwrite() 系统调用【对应到 Java NIO 的 FileChannel.write() API】按偏移量写入数据，此时数据都会先写入`page cache`。consumer 消费消息时，Broker 使用 sendfile() 系统调用【对应 FileChannel.transferTo() API】，零拷贝地将数据从 page cache 传输到 broker 的 Socket buffer，再通过网络传输。

leader 与 follower 之间的同步，与上面 consumer 消费数据的过程是同理的。

`page cache`中的数据会随着内核中 flusher 线程的调度以及对 sync()/fsync() 的调用写回到磁盘，就算进程崩溃，也不用担心数据丢失。另外，如果 consumer 要消费的消息不在`page cache`里，才会去磁盘读取，并且会顺便预读出一些相邻的块放入 page cache，以方便下一次读取。

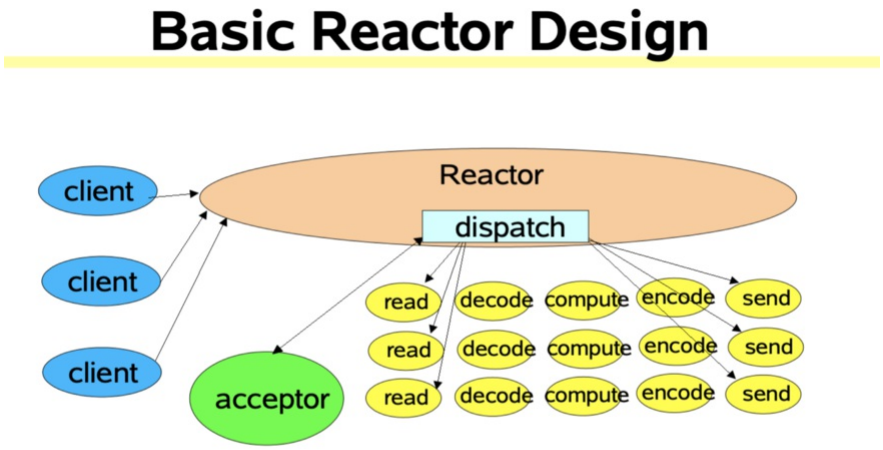
因此如果 Kafka producer 的生产速率与 consumer 的消费速率相差不大，那么就能几乎只靠对 broker page cache 的读写完成整个生产 - 消费过程，磁盘访问非常少。

网络模型

> “
> 65 哥：网络嘛，作为 Java 程序员，自然是 Netty
> ”

是的，Netty 是 JVM 领域一个优秀的网络框架，提供了高性能的网络服务。大多数 Java 程序员提到网络框架，首先想到的就是 Netty。Dubbo、Avro-RPC 等等优秀的框架都使用 Netty 作为底层的网络通信框架。

Kafka 自己实现了网络模型做 RPC。底层基于 Java NIO，采用和 Netty 一样的 Reactor 线程模型。



Single threaded version

Reacotr 模型主要分为三个角色

Reactor：把 IO 事件分配给对应的 handler 处理

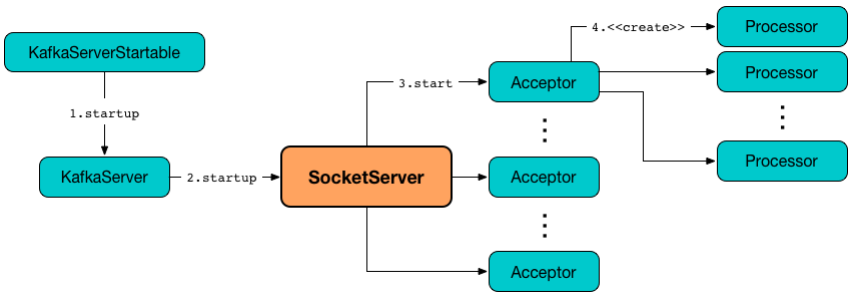
Acceptor：处理客户端连接事件

Handler：处理非阻塞的任务

在传统阻塞 IO 模型中，每个连接都需要独立线程处理，当并发数大时，创建线程数多，占用资源；采用阻塞 IO 模型，连接建立后，若当前线程没有数据可读，线程会阻塞在读操作上，造成资源浪费

针对传统阻塞 IO 模型的两个问题，Reactor 模型基于池化思想，避免为每个连接创建线程，连接完成后将业务处理交给线程池处理；基于 IO 复用模型，多个连接共用同一个阻塞对象，不用等待所有的连接。遍历到有新数据可以处理时，操作系统会通知程序，线程跳出阻塞状态，进行业务逻辑处理

Kafka 即基于 Reactor 模型实现了多路复用和处理线程池。其设计如下：



其中包含了一个`Acceptor`线程，用于处理新的连接，`Acceptor` 有 N 个 `Processor` 线程 select 和 read socket 请求，N 个 `Handler` 线程处理请求并相应，即处理业务逻辑。

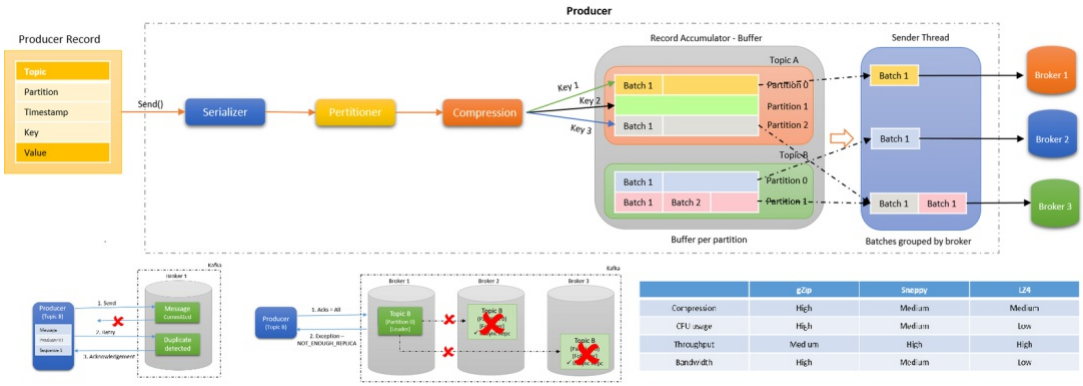
I/O 多路复用可以通过把多个 I/O 的阻塞复用到同一个 select 的阻塞上，从而使得系统在单线程的情况下可以同时处理多个客户端请求。它的最大优势是系统开销小，并且不需要创建新的进程或者线程，降低了系统的资源开销。

总结： Kafka Broker 的 `KafkaServer` 设计是一个优秀的网络架构，有想了解 Java 网络编程，或需要使用到这方面技术的同学不妨去读一读源码。后续『码哥』的 Kafka 系列文章也将涉及这块源码的解读。

批量与压缩

Kafka Producer 向 Broker 发送消息不是一条消息一条消息的发送。使用过 Kafka 的同学应该知道，Producer 有两个重要的参数：`batch.size`和`linger.ms`。这两个参数就和 Producer 的批量发送有关。

Kafka Producer 的执行流程如下图所示：



发送消息依次经过以下处理器：

Serialize：键和值都根据传递的序列化器进行序列化。优秀的序列化方式可以提高网络传输的效率。

Partition：决定将消息写入主题的哪个分区，默认情况下遵循 murmur2 算法。自定义分区程序也可以传递给生产者，以控制应将消息写入哪个分区。

Compress：默认情况下，在 Kafka 生产者中不启用压缩.Compression 不仅可以更快地从生产者传输到代理，还可以在复制过程中进行更快的传输。压缩有助于提高吞吐量，降低延迟并提高磁盘利用率。

Accumulate：`Accumulate`顾名思义，就是一个消息累计器。其内部为每个 Partition 维护一个`Deque`双端队列，队列保存将要发送的批次数据，`Accumulate`将数据累计到一定数量，或者在一定过期时间内，便将数据以批次的方式发送出去。记录被累积在主题每个分区的缓冲区中。根据生产者批次大小属性将记录分组。主题中的每个分区都有一个单独的累加器 / 缓冲区。

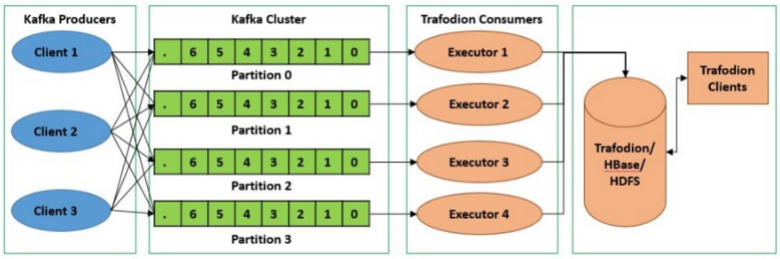
Group Send：记录累积器中分区的批次按将它们发送到的代理分组。批处理中的记录基于 batch.size 和 linger.ms 属性发送到代理。记录由生产者根据两个条件发送。当达到定义的批次大小或达到定义的延迟时间时。

Kafka 支持多种压缩算法：lz4、snappy、gzip。Kafka 2.1.0 正式支持 ZStandard —— ZStandard 是 Facebook 开源的压缩算法，旨在提供超高的压缩比 (compression ratio)，具体细节参见 zstd。

Producer、Broker 和 Consumer 使用相同的压缩算法，在 producer 向 Broker 写入数据，Consumer 向 Broker 读取数据时甚至可以不用解压缩，最终在 Consumer Poll 到消息时才解压，这样节省了大量的网络和磁盘开销。

分区并发

Kafka 的 Topic 可以分成多个 Partition，每个 Paritition 类似于一个队列，保证数据有序。同一个 Group 下的不同 Consumer 并发消费 Paritition，分区实际上是调优 Kafka 并行度的最小单元，因此，可以说，每增加一个 Paritition 就增加了一个消费并发。



Kafka 具有优秀的分区分配算法——StickyAssignor，可以保证分区的分配尽量地均衡，且每一次重分配的结果尽量与上一次分配结果保持一致。这样，整个集群的分区尽量地均衡，各个 Broker 和 Consumer 的处理不至于出现太大的倾斜。

> “
> 65 哥：那是不是分区数越多越好呢？
> ”

当然不是。

越多的分区需要打开更多的文件句柄

在 kafka 的 broker 中，每个分区都会对照着文件系统的一个目录。在 kafka 的数据日志文件目录中，每个日志数据段都会分配两个文件，一个索引文件和一个数据文件。因此，随着 partition 的增多，需要的文件句柄数急剧增加，必要时需要调整操作系统允许打开的文件句柄数。

客户端 / 服务器端需要使用的内存就越多

客户端 producer 有个参数 batch.size，默认是 16KB。它会为每个分区缓存消息，一旦满了就打包将消息批量发出。看上去这是个能够提升性能的设计。不过很显然，因为这个参数是分区级别的，如果分区数越多，这部分缓存所需的内存占用也会更多。

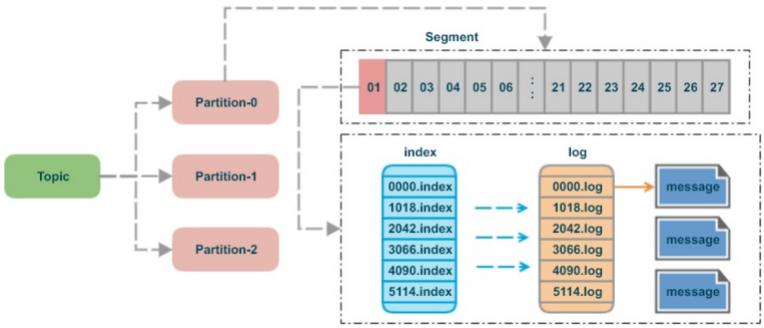
降低高可用性

分区越多，每个 Broker 上分配的分区也就越多，当一个发生 Broker 宕机，那么恢复时间将很长。

文件结构

Kafka 消息是以 Topic 为单位进行归类，各个 Topic 之间是彼此独立的，互不影响。每个 Topic 又可以分为一个或多个分区。每个分区各自存在一个记录消息数据的日志文件。

Kafka 每个分区日志在物理上实际按大小被分成多个 Segment。



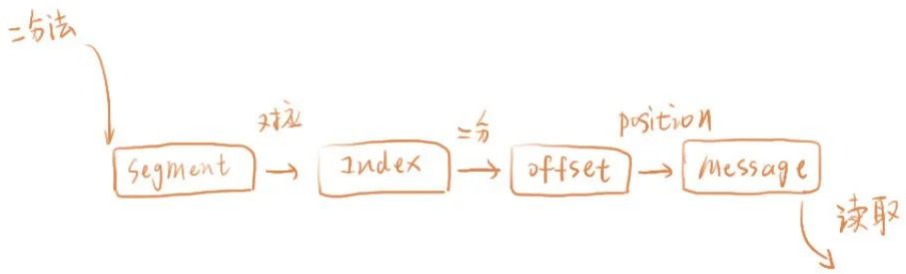
segment file 组成：由 2 大部分组成，分别为 index file 和 data file，此 2 个文件一一对应，成对出现，后缀”.index”和“.log”分别表示为 segment 索引文件、数据文件。

segment 文件命名规则：partion 全局的第一个 segment 从 0 开始，后续每个 segment 文件名为上一个 segment 文件最后一条消息的 offset 值。数值最大为 64 位 long 大小，19 位数字字符长度，没有数字用 0 填充。

index 采用稀疏索引，这样每个 index 文件大小有限，Kafka 采用`mmap`的方式，直接将 index 文件映射到内存，这样对 index 的操作就不需要操作磁盘 IO。`mmap`的 Java 实现对应`MappedByteBuffer`。

> “
> 65 哥笔记：mmap 是一种内存映射文件的方法。即将一个文件或者其它对象映射到进程的地址空间，实现文件磁盘地址和进程虚拟地址空间中一段虚拟地址的一一对映关系。实现这样的映射关系后，进程就可以采用指针的方式读写操作这一段内存，而系统会自动回写脏页面到对应的文件磁盘上，即完成了对文件的操作而不必再调用 read,write 等系统调用函数。相反，内核空间对这段区域的修改也直接反映用户空间，从而可以实现不同进程间的文件共享。
> ”

Kafka 充分利用二分法来查找对应 offset 的消息位置：



按照二分法找到小于 offset 的 segment 的.log 和.index

用目标 offset 减去文件名中的 offset 得到消息在这个 segment 中的偏移量。

再次用二分法在 index 文件中找到对应的索引。

到 log 文件中，顺序查找，直到找到 offset 对应的消息。

总结

Kafka 是一个优秀的开源项目。其在性能上面的优化做的淋漓尽致，是很值得我们深入学习的一个项目。无论是思想还是实现，我们都应该认真的去看一看，想一想。

Kafka 性能优化点：

磁盘读写优化

零拷贝优化

优化网络模型，基于Java NIO

高效的文件数据结构设计

并行和可扩展

数据批量与压缩

无锁轻量级 offset

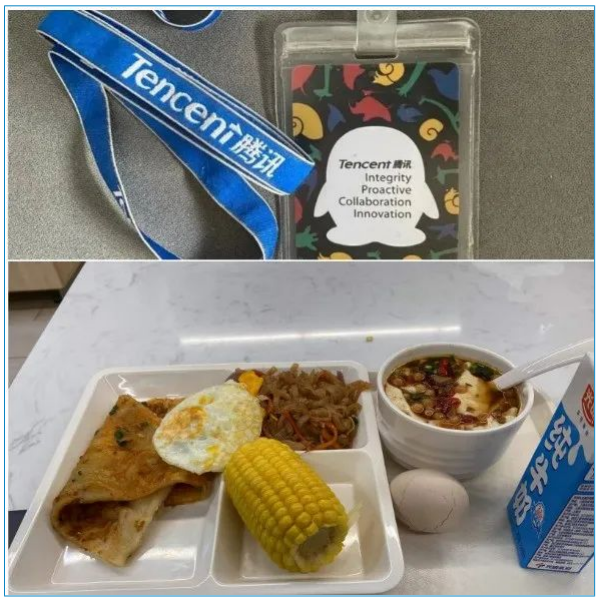
- END -

看完一键三连在看，转发，点赞

是对文章最大的赞赏，极客重生感谢你 

推荐阅读

深入理解Kafka的设计思想



大厂后台开发基本功修炼路线和经典资料

技术分享|深入理解Linux操作系统

你好，这里是极客重生，我是阿荣，大家都叫我荣哥，从华为->外企->到互联网大厂，目前是大厂资深工程师，多次获得五星员工，多年职场经验，技术扎实，专业后端开发和后台架构设计，热爱底层技术，丰富的实战经验，分享技术的本质原理，希望帮助更多人蜕变重生，拿BAT大厂offer，培养高级工程师能力，成为技术专家，实现高薪梦想，期待你的关注！**点击蓝字查看我的成长之路。**

校招/社招/简历/面试技巧/大厂技术栈分析/后端开发进阶/优秀开源项目/直播分享/技术视野/实战高手等, 极客星球希望成为最有技术价值星球，尽最大努力为星球的同学提供技术和成长帮助！详情查看->极客星球

求点赞，在看，分享三连