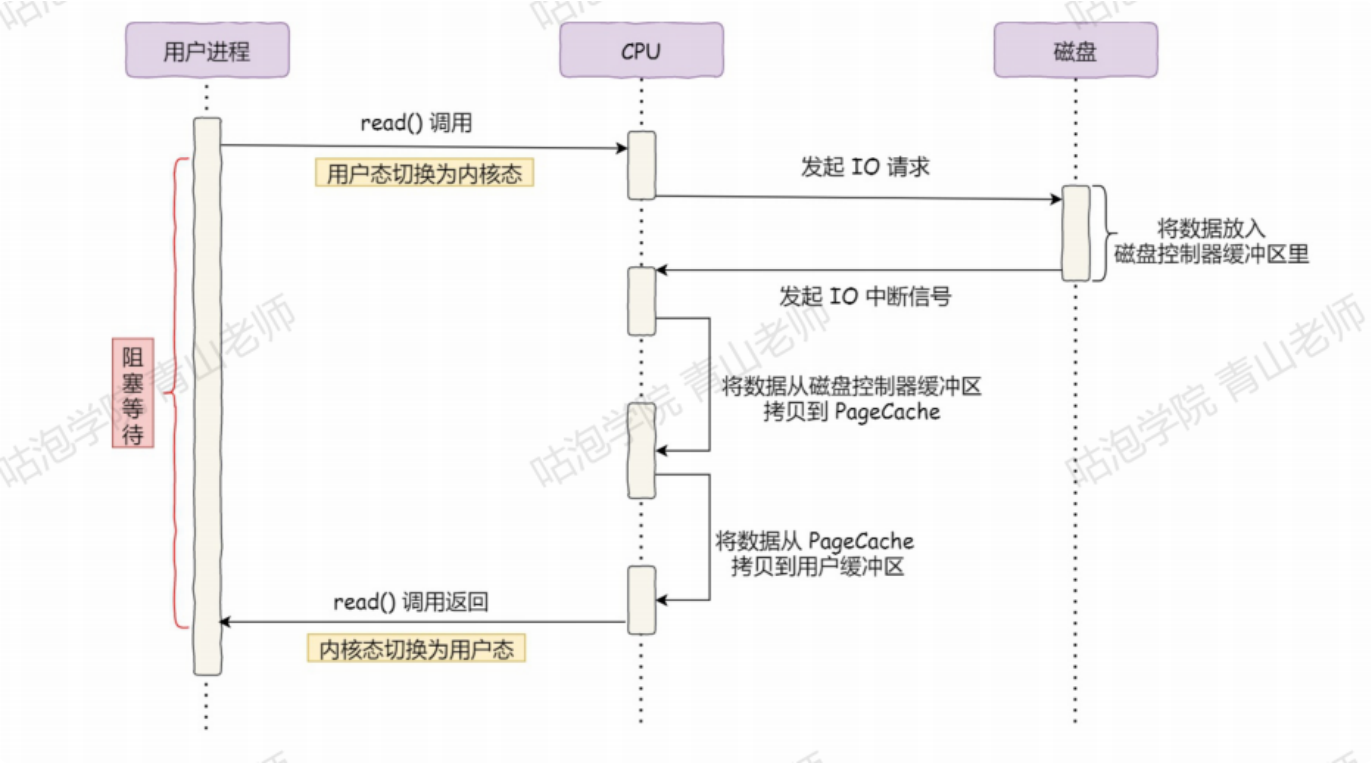
Kafka架构以及原理

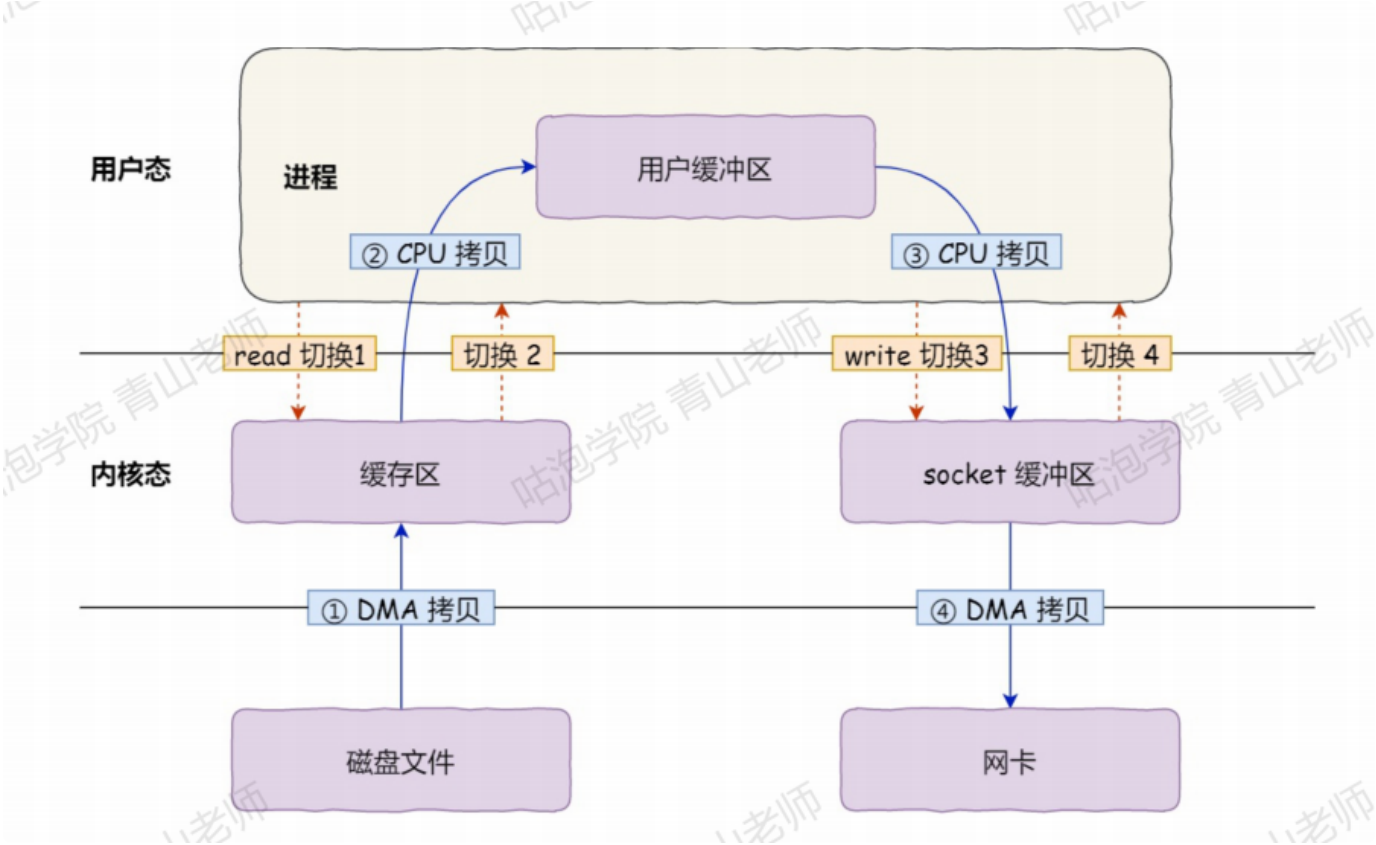
Kafka为什么那么快?

零拷贝:

> Kafka使用了零拷贝技术，也就是直接将数据从内核空间的读缓冲区直接拷贝到内核空间的socket 缓冲区，然后再写入到 NIC 缓冲区，避免了在内核空间和用户空间之间穿梭。
>
> 首先有两个名词要给大家解释一下。
>
> 第一个是操作系统虚拟内存的**内核空间**和**用户空间**。
>
> 操作系统的虚拟内存分成了两块，一部分是内核空间，一部分是用户空间。这样就可以避免用户进程直接操作内核，保证内核安全。
>
> 进程在内核空间可以执行任意命令，调用系统的一切资源；在用户空间必须要通过一些系统接口才能向内核发出指令。
>
> 如果用户要从磁盘读取数据（比如kafka消费消息），必须先把数据从磁盘拷贝到内核缓冲区，然后在从内核缓冲区到用户缓冲区，最后才能返回给用户。



> 第二个是DMA拷贝。没有DMA技术的时候，拷贝数据的事情需要CPU亲自去做，这个时候它没法干其他的事情，如果传输的数据量大那就有问题了。
>
> DMA技术叫做直接内存访问（Direct Memory Access），其实可以理解为CPU给自己找了一个小弟帮它做数据搬运的事情。在进行I/O设备和内存的数据传输的时候，数据搬运的工作全部交给 DMA 控制器，解放了CPU的双手。理解了这两个东西之后，我们来看下传统的I/O模型：



1.应用程序调用read函数，向操作系统发起IO调用，**上下文从用户态切换至内核态**

2.DMA控制器把数据从磁盘中读取到内核缓冲区

3.CPU把内核缓冲区数据拷贝到用户应用缓冲区，**上下文从内核态切换至用户态**，此时read函数返回

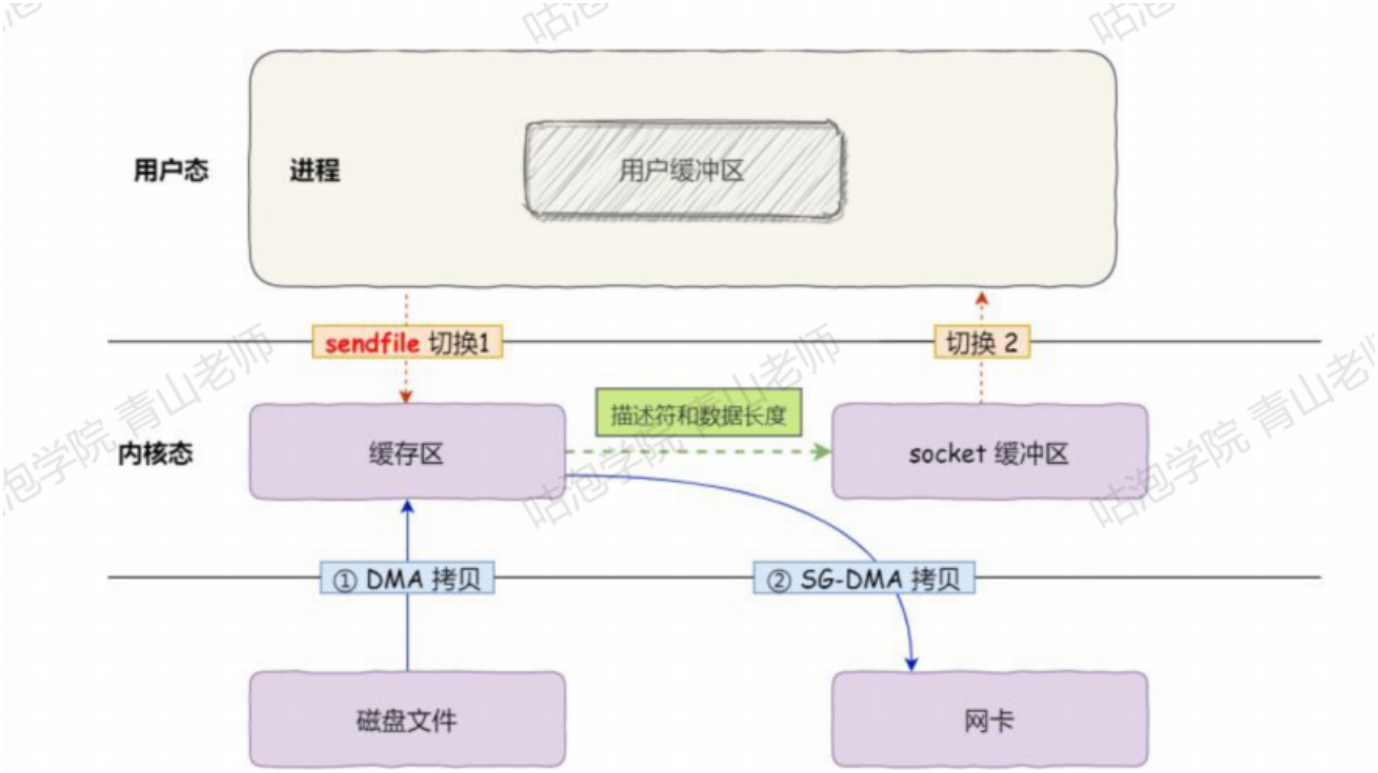
4.用户应用进程通过write函数，发起IO调用，**上下文从用户态切换至内核态**

5.CPU将缓冲区的数据拷贝到socket缓冲区

6.DMA控制器将数据从socket缓冲区拷贝到网卡设备，**上下文从内核态切换至用户态**，此时write函数返回

比如 kafka要消费消息，比如要先把数据从磁盘拷贝到内核缓冲区，然后拷贝到用户缓冲区，再拷贝到socket缓冲区，再拷贝到网卡设备。这里面发生了4次用户态和内核态的切换和4次数据拷贝，2次系统函数的调用（read、write），这个过程是非常耗费时间的。怎么优化呢?

> 在Linux操作系统里面提供了一个sendfile 函数，可以实现"零拷贝"。这个时候就不需要经过用户缓冲区了，直接把数据拷贝到网卡（这里画的是支持SG-DMA拷贝的情况）
>
> 因为这个只有DMA拷贝，没有CPU拷贝，所以叫做"零拷贝"。零拷贝至少可以提高一倍的性能。



Kafka 文件传输最终调用的是 Java NIO 库里的 transferTo 方法（PlaintextTransportLayer）：

```
@Overridepublic
long transferFrom(FileChannel fileChannel, long position, long count) throws IOException {
    return fileChannel.transferTo(position, count, socketChannel);
}
```

如果 Linux 系统支持 sendfile() 系统调用，那么 transferTo() 实际上最后就会使用到 sendfile() 系统调用函数。零拷贝技术可以大大地提升文件传输的性能。

顺序读写：

磁盘分为顺序读写与随机读写，基于磁盘的随机读写确实很慢，但磁盘的顺序读写性能却很高，kafka 这里采用的就是顺序读写。

批量读写：

生产者可以借助累加器，批量发送消息，消费者也可以批量拉取消费。Kafka 数据读写也是批量的而不是单条的,这样可以避免在网络上频繁传输单个消息带来的延迟和带宽开销。假设网络带宽为10MB/S，一次性传输10MB的消息比传输1KB的消息10000万次显然要快得多。

数据压缩：

Producer 可将数据压缩后发送给 broker，从而减少网络传输代价，目前支持的压缩算法有：Snappy、Gzip、LZ4。数据压缩一般都是和批处理配套使用来作为优化手段的。

分区分段+稀疏索引

Kafka 的 message 是按 topic分 类存储的，topic 中的数据又是按照一个一个的 partition 即分区存储到不同 broker 节点。每个 partition 对应了操作系统上的一个文件夹，partition 实际上又是按照segment分段存储的。通过这种分区分段的设计，Kafka 的 message 消息实际上是分布式存储在一个一个小的 segment 中的，每次文件操作也是直接操作的 segment。为了进一步的查询优化，Kafka 又默认为分段后的数据文件建立了索引文件，就是文件系统上的.index文件。这种分区分段+索引的设计，不仅提升了数据读取的效率，同时也提高了数据操作的并行度。

Page Cache

为了优化读写性能，Kafka 利用了操作系统本身的 Page Cache。数据直接写入page cache定时刷新脏页到磁盘即可。消费者拉取消息时，如果数据在page cache中，甚至能不需要去读磁盘io。读操作可直接在 Page Cache 内进行。如果消费和生产速度相当，甚至不需要通过物理磁盘（直接通过 Page Cache）交换数据

Broker 收到数据后，写磁盘时只是将数据写入 Page Cache，并不保证数据一定完全写入磁盘。从这一点看，可能会造成机器宕机时，Page Cache 内的数据未写入磁盘从而造成数据丢

> 失。但是这种丢失只发生在机器断电等造成操作系统不工作的场景，而这种场景完全可以由 Kafka 层面的 Replication 机制去解决。如果为了保证这种情况下数据不丢失而强制将 Page Cache 中的数据 Flush 到磁盘，反而会降低性能。也正因如此，Kafka 虽然提供了 flush.messages和flush.ms两个参数将 Page Cache 中的数据强制 Flush 到磁盘，但是 Kafka 并不建议使用。

如图：部署3个Broker，该Topic有3个分区，每个分区一共3个副本