# Kafka

题目难度：★★★

知识点标签：分布式消息中间件

主要应用场景：日志收集系统和消息系统

# 题目描述

Kafka为什么速度这么快?

## 解题思路

> Kafka作为MQ也好，作为存储层也罢，主要是用到两个功能:一个是Producer生产的数据存储到broker，二是Consumer从broker读取数据。那Kafka的速度快也就从写和读这两大方面考虑。

# 写数据分析

Kafka会把收到的消息都写入到硬盘中，它绝对不会丢失数据。为了优化写入速度Kafka采用了两个技术， 顺序写入和MMFile

# 1. 顺序写入

磁盘读写的快慢取决于你怎么使用它，也就是顺序读写或者随机读写。在顺序读写的情况下，磁盘的顺序读写速度和内存持平。

因为硬盘是机械结构，每次读写都会寻址->写入，其中寻址是一个“机械动作”，它是最耗时的。所以硬盘最讨厌随机I/O，最喜欢顺序I/O。为了提高读写硬盘的速度，Kafka就是使用顺序I/O。

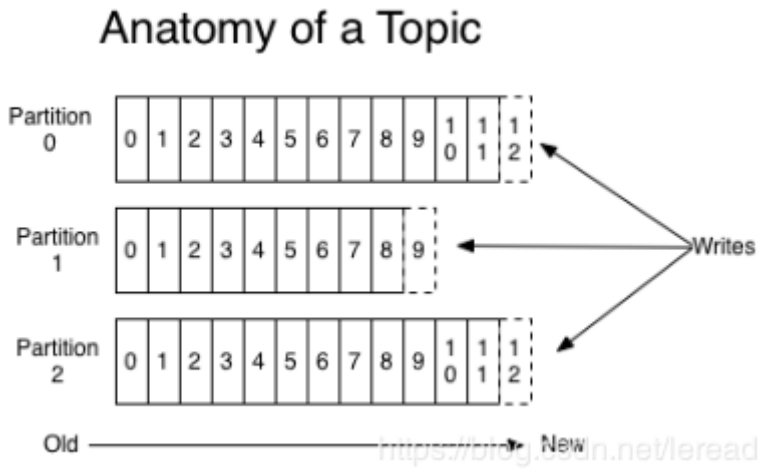
而且Linux对于磁盘的读写优化也比较多，包括read-ahead和write-behind，磁盘缓存等。如果在内存做这些操作的时候，一个是JAVA对象的内存开销很大，另一个是随着堆内存数据的增多，JAVA的GC时间会变得很长，使用磁盘操作有以下几个好处：

1、顺序写入磁盘顺序读写速度超过内存随机读写

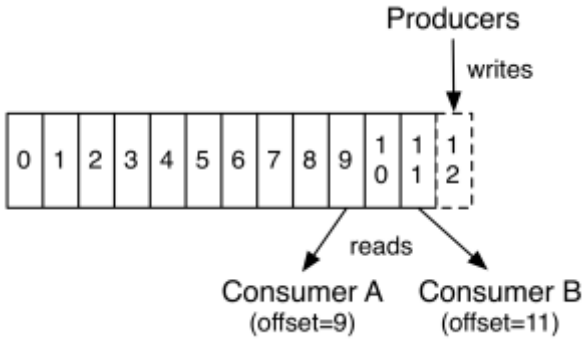
2、顺序写入JVM的GC效率低，内存占用大。使用磁盘可以避免这一问题

3、顺序写入系统冷启动后，磁盘缓存依然可用

下图就展示了Kafka是如何写入数据的， 每一个Partition其实都是一个文件 ， 收到消息后Kafka会把数据插入到文件末尾（虚框部分）：




这种方法有一个缺陷——没有办法删除数据 ，所以Kafka是不会删除数据的，它会把所有的数据都保留下来，每个消费者（Consumer）对每个Topic都有一个offset用来表示读取到了第几条数据 。




两个消费者

- 顺序写入Consumer1有两个offset分别对应Partition0、Partition1（假设每一个Topic一个Partition）；
- 顺序写入Consumer2有一个offset对应Partition2。

这个offset是由客户端SDK负责保存的，Kafka的Broker完全无视这个东西的存在；一般情况下SDK会把它保存到Zookeeper里面，所以需要给Consumer提供zookeeper的地址。

**如果不删除硬盘肯定会被撑满，所以Kakfa提供了两种策略来删除数据：**

- 顺序写入一是基于时间。
- 顺序写入二是基于partition文件大小。

具体配置可以参看它的配置文档

# 2、Memory Mapped Files

即便是顺序写入硬盘，硬盘的访问速度还是不可能追上内存。所以Kafka的数据并不是实时的写入硬盘 ，它充分利用了现代操作系统分页存储来利用内存提高I/O效率。

Memory Mapped Files(后面简称mmap)也被翻译成 内存映射文件 ，在64位操作系统中一般可以表示20G的数据文件，它的工作原理是直接利用操作系统的Page来实现文件到物理内存的直接映射。

完成映射之后你对物理内存的操作会被同步到硬盘上（操作系统在适当的时候）。

通过mmap，进程像读写硬盘一样读写内存（当然是虚拟机内存），也不必关心内存的大小有虚拟内存为我们兜底。

使用这种方式可以获取很大的I/O提升，省去了用户空间到内核空间复制的开销（调用文件的read会把数据先放到内核空间的内存中，然后再复制到用户空间的内存中。）

但也有一个很明显的缺陷——不可靠，写到mmap中的数据并没有被真正的写到硬盘，操作系统会在程序主动调用flush的时候才把数据真正的写到硬盘。

Kafka提供了一个参数——producer.type来控制是不是主动flush，如果Kafka写入到mmap之后就立即flush然后再返回Producer叫 同步 (sync)；写入mmap之后立即返回Producer不调用flush叫异步 (async)。

# 读取数据分析

Kafka在读取磁盘时做了哪些优化?

1、基于sendfile实现Zero Copy

传统模式下，当需要对一个文件进行传输的时候，其具体流程细节如下：

1.1、基于sendfile实现Zero Copy调用read函数，文件数据被copy到内核缓冲区

1.2、read函数返回，文件数据从内核缓冲区copy到用户缓冲区

1.3、write函数调用，将文件数据从用户缓冲区copy到内核与socket相关的缓冲区。

1.4、数据从socket缓冲区copy到相关协议引擎。

以上细节是传统read/write方式进行网络文件传输的方式，我们可以看到，在这个过程当中，文件数据实际上是经过了四次copy操作：

硬盘—>内核buf—>用户buf—>socket相关缓冲区—>协议引擎

而sendfile系统调用则提供了一种减少以上多次copy，提升文件传输性能的方法。

在内核版本2.1中，引入了sendfile系统调用，以简化网络上和两个本地文件之间的数据传输。sendfile的引入不仅减少了数据复制，还减少了上下文切换。

sendfile(socket, file, len);

**运行流程如下：**

1、sendfile系统调用，文件数据被copy至内核缓冲区

2、再从内核缓冲区copy至内核中socket相关的缓冲区

3、最后再socket相关的缓冲区copy到协议引擎

相较传统read/write方式，2.1版本内核引进的sendfile已经减少了内核缓冲区到user缓冲区，再由user缓冲区到socket相关缓冲区的文件copy，而在内核版本2.4之后，文件描述符结果被改变，sendfile实现了更简单的方式，再次减少了一次copy操作。

在Apache、Nginx、lighttpd等web服务器当中，都有一项sendfile相关的配置，使用sendfile可以大幅提升文件传输性能。

Kafka把所有的消息都存放在一个一个的文件中，当消费者需要数据的时候Kafka直接把文件发送给消费者，配合mmap作为文件读写方式，直接把它传给sendfile。

**2、批量压缩**

在很多情况下，系统的瓶颈不是CPU或磁盘，而是网络IO，对于需要在广域网上的数据中心之间发送消息的数据流水线尤其如此。进行数据压缩会消耗少量的CPU资源,不过对于kafka而言,网络IO更应该需要考虑。

2.1、如果每个消息都压缩，但是压缩率相对很低，所以Kafka使用了批量压缩，即将多个消息一起压缩而不是单个消息压缩

2.2、Kafka允许使用递归的消息集合，批量的消息可以通过压缩的形式传输并且在日志中也可以保持压缩格式，直到被消费者解压缩

2.3、Kafka支持多种压缩协议，包括Gzip和Snappy压缩协议

# 应用场景分析

消息队列

- 比起大多数的消息系统来说，Kafka有更好的吞吐量，内置的分区，冗余及容错性，这让Kafka成为了一个很好的大规模消息处理应用的解决方案。消息系统一般吞吐量相对较低，但是需要更小的端到端延时，并常常依赖于Kafka提供的强大的持久性保障。在这个领域，Kafka足以媲美传统消息系统，如ActiveMQ或RabbitMQ。
- 日志收集
    - 日志收集方面，其实开源产品有很多，包括Scribe、Apache Flume。很多人使用Kafka代替日志聚合（log aggregation）。日志聚合一般来说是从服务器上收集日志文件，然后放到一个集中的位置（文件服务器或HDFS）进行处理。然而Kafka忽略掉文件的细节，将其更清晰地抽象成一个个日志或事件的消息流。这就让Kafka处理过程延迟更低，更容易支持多数据源和分布式数据处理。比起以日志为中心的系统比如Scribe或者Flume来说，Kafka提供同样高效的性能和因为复制导致的更高的耐用性保证，以及更低的端到端延迟。