19-数据压缩：时间换空间的游戏

你好，我是李玥。

这节课我们一起来聊一聊数据压缩。我在前面文章中提到过，我曾经在一台配置比较高的服务器上，对Kafka做过一个极限的性能压测，想验证一下Kafka到底有多快。我使用的种子消息大小为1KB，只要是并发数量足够多，不开启压缩时，可以打满万兆网卡的全部带宽，TPS接近100万。开启压缩时，TPS可以达到2000万左右，吞吐量提升了大约20倍！

算术好的同学可能会立刻反驳我说，2000万TPS乘以1KB的消息大小，再把字节Byte转换成比特bit，换算成网络传输的带宽是200Gb/s，服务器网卡根本达不到这么大的传输带宽！

我们的测试服务器的网卡就是普通的万兆网卡，极限带宽也就是10Gb/s，压测时候的实际网络流量大概在7Gb/s左右。这里面，最重要的原因就是，我在测试的时候开启了Kafka的压缩功能。可以看到，对于Kafka来说，使用数据压缩，提升了大概几十倍的吞吐量。当然，在实际生产时，不太可能达到这么高的压缩率，但是合理地使用数据压缩，仍然可以做到提升数倍的吞吐量。

所以，**数据压缩不仅能节省存储空间，还可以用于提升网络传输性能。**这种使用压缩来提升系统性能的方法，不仅限于在消息队列中使用，我们日常开发的应用程序也可以使用。比如，我们的程序要传输大量的数据，或者要在磁盘、数据库中存储比较大的数据，这些情况下，都可以考虑使用数据压缩来提升性能，还能节省网络带宽和存储空间。

那如何在你的程序中使用压缩？应该选择什么样的压缩算法更适合我们的系统呢？这节课，我带你一起学习一下，使用数据压缩来提升系统性能的方法。

什么情况适合使用数据压缩？

在使用压缩之前，首先你需要考虑，当前这个场景是不是真的适合使用数据压缩。

比如，进程之间通过网络传输数据，这个数据是不是需要压缩呢？我和你一起来对比一下：

- 不压缩直接传输需要的时间是： 传输**未压缩**数据的耗时。
- 使用数据压缩需要的时间是： 压缩耗时 + 传输**压缩**数据耗时 + 解压耗时。

到底是压缩快，还是不压缩快呢？其实不好说。影响的因素非常多，比如数据的压缩率、网络带宽、收发两端服务器的繁忙程度等等。

压缩和解压的操作都是计算密集型的操作，非常耗费CPU资源。如果你的应用处理业务逻辑就需要耗费大量的CPU资源，就不太适合再进行压缩和解压。

又比如说，如果你的系统的瓶颈是磁盘的IO性能，CPU资源又很闲，这种情况就非常适合在把数据写入磁盘前先进行压缩。

但是，如果你的系统读写比严重不均衡，你还要考虑，每读一次数据就要解压一次是不是划算。

压缩它的本质是资源的置换，是一个时间换空间，或者说是CPU资源换存储资源的游戏。

就像木桶的那个短板一样，每一个系统它都有一个性能瓶颈资源，可能是磁盘IO，网络带宽，也可能是CPU。如果使用压缩，能用长板来换一些短板，那总体上就能提升性能，这样就是划算的。如果用了压缩之后，短板更短了，那就不划算了，不如不用。

如果通过权衡，使用数据压缩确实可以提升系统的性能，接下来就需要选择合适的压缩算法。

应该选择什么压缩算法？

压缩算法可以分为有损压缩和无损压缩。有损压缩主要是用来压缩音视频，它压缩之后是会丢失信息的。我们这里讨论的全都是无损压缩，也就是说，数据经过压缩和解压过程之后，与压缩之前相比，是100%相同的。

数据为什么可以被压缩呢？各种各样的压缩算法又是怎么去压缩数据的呢？我举个例子来简单说明一下。

比如说，下面这段数据：

> 00000000000000000000

我来给你人肉压缩一下：

> 20个0

20个字符就被压缩成了4个字符，并且是可以无损还原的。当然，我举的例子比较极端，我的压缩算法也几乎没什么实用性，但是，这确实是一个压缩算法，并且和其他的压缩算法本质是没什么区别的。

目前常用的压缩算法包括：ZIP，GZIP，SNAPPY，LZ4等等。选择压缩算法的时候，主要需要考虑数据的压缩率和压缩耗时。一般来说，压缩率越高的算法，压缩耗时也越高。如果是对性能要求高的系统，可以选择压缩速度快的算法，比如LZ4；如果需要更高的压缩比，可以考虑GZIP或者压缩率更高的XZ等算法。

压缩样本对压缩速度和压缩比的影响也是比较大的，同样大小的一段数字和一段新闻的文本，即使是使用相同的压缩算法，压缩率和压缩时间的差异也是比较大的。所以，有的时候在选择压缩算法的之前，用系统的样例业务数据做一个测试，可以帮助你找到最合适的压缩算法。

在这里，我不会去给你讲某一种压缩算法，因为压缩算法都很复杂，一般来说也不需要我们来实现某种压缩算法，如果你感兴趣的话，可以去学习一下最经典压缩算法：哈夫曼编码（也叫霍夫曼编码，Huffman Coding）。

如何选择合适的压缩分段？

大部分的压缩算法，他们的区别主要是，对数据进行编码的算法，压缩的流程和压缩包的结构大致一样的。而在压缩过程中，你最需要了解的就是如何选择合适的压缩分段大小。

在压缩时，给定的被压缩数据它必须有确定的长度，或者说，是有头有尾的，不能是一个无限的数据流，**如果要对流数据进行压缩，那必须把流数据划分成多个帧，一帧一帧的分段压缩。**

主要原因是，压缩算法在开始压缩之前，一般都需要对被压缩数据从头到尾进行一次扫描，扫描的目的是确定如何对数据进行划分和编码，一般的原则是重复次数多、占用空间大的内容，使用尽量短的编码，这样压

缩率会更高。

另外，被压缩的数据长度越大，重码率会更高，压缩比也就越高。这个很好理解，比如我们这篇文章，可能出现了几十次“压缩”这个词，如果将整篇文章压缩，这个词的重复率是几十次，但如果我们按照每个自然段来压缩，那每段中这个词的重复率只有二三次。显然全文压缩的压缩率肯定高于分段压缩。

当然，分段也不是越大越好，实际上分段大小超过一定长度之后，再增加长度对压缩率的贡献就不太大了，这是一个原因。另外，过大的分段长度，在解压缩的时候，会有更多的解压浪费。比如，一个1MB大小的压缩文件，即使你只是需要读其中很短的几个字节，也不得不把整个文件全部解压缩，造成很大的解压浪费。

所以，你需要根据你的业务，选择合适的压缩分段，在压缩率、压缩速度和解压浪费之间找到一个合适的平衡。

确定了如何对数据进行划分和压缩算法之后，就可以进行压缩了，压缩的过程就是用编码来替换原始数据的过程。压缩之后的压缩包就是由这个编码字典和用编码替换之后的数据组成的。

这就是数据压缩的过程。解压的时候，先读取编码字典，然后按照字典把压缩编码还原成原始的数据就可以了。

Kafka是如何处理消息压缩的?

回过头来，我们再看一下Kafka它是如何来处理数据压缩的。

首先，Kafka是否开启压缩，这是可以配置，它也支持配置使用哪一种压缩算法。原因我们在上面说过，不同的业务场景是否需要开启压缩，选择哪种压缩算法是不能一概而论的。所以，Kafka的设计者把这个选择权交给使用者。

在开启压缩时，Kafka选择一批消息一起压缩，每一个批消息就是一个压缩分段。使用者也可以通过参数来控制每批消息的大小。

我们之前讲过，在Kafka中，生产者生成一个批消息发给服务端，在服务端中是不会拆分批消息的。那按照批来压缩，意味着，在服务端也不用对这批消息进行解压，可以整批直接存储，然后整批发送给消费者。最后，批消息由消费者进行解压。

在服务端不用解压，就不会耗费服务端宝贵的CPU资源，同时还能获得压缩后，占用传输带宽小，占用存储空间小的这些好处，这是一个非常聪明的设计。

在使用Kafka时，如果生产者和消费者的CPU资源不是特别吃紧，开启压缩后，可以节省网络带宽和服务端的存储空间，提升总体的吞吐量，一般都是个不错的选择。

小结

数据压缩，它本质上是用CPU资源换取存储资源，或者说是用压缩解压的时间来换取存储的空间，这个买卖是不是划算，需要你根据自己的情况先衡量一下。

在选择压缩算法的时候，需要综合考虑压缩时间和压缩率两个因素，被压缩数据的内容也是影响压缩时间和压缩率的重要因素，必要的时候可以先用业务数据做一个压缩测试，这样有助于选择最合适的压缩算法。

另外一个影响压缩率的重要因素是压缩分段的大小，你需要根据业务情况选择一个合适的分段策略，在保证不错的压缩率的前提下，尽量减少解压浪费。

最后，我们讲了一下Kafka它是如何处理消息压缩的。Kafka在生产者上，对每批消息进行压缩，批消息在服务端不解压，消费者在收到消息之后再进行解压。简单地说，Kafka的压缩和解压都是在客户端完成的。

思考题

课后，你可以去看一下RocketMQ的文档或者源代码，看一下，RocketMQ是怎么处理消息压缩的。然后和Kafka的压缩方式对比一下，想一想哪种处理方式更适合你的系统？

欢迎你在留言区把你的思考分享出来，如果有任何问题也欢迎你在留言区提问。

感谢阅读，如果你觉得这篇文章对你有一些启发，也欢迎把它分享给你的朋友。

精选留言：

- 姜戈 2019-09-05 11:53:58
  RocketMq(rockmq-all 4.4.1-SNAPSHOT): 默认压缩大于4K的消息（可配置），压缩算法是zip，默认级别5（可配置），同样也是客户端做解压缩工作，服务端只存储；对于批量消息的压缩，目前并不支持。
  摘取DefaultMQProducerImpl.java 部分源码：
  private boolean tryToCompressMessage(final Message msg) {
  if (msg instanceof MessageBatch) {
  //batch dose not support compressing right now
  return false;
  }
  byte[] body = msg.getBody();
  if (body != null) {
  if (body.length >= this.defaultMQProducer.getCompressMsgBodyOverHowmuch()) {
  try {
  byte[] data = UtilAll.compress(body, zipCompressLevel);

```
if (data != null) {
msg.setBody(data);
return true;
}
} catch (IOException e) {
log.error("tryToCompressMessage exception", e);
log.warn(msg.toString());
}
}
}

return false;
} [4赞]
```

- leslie 2019-09-05 05:07:37
  老师提到了压缩：适当扩展一下，这个东西早期数据库备份其实经常会要去使用；个人的感受不是耗资源-是极度耗费资源，压缩比越大CPU越大，服务器做数据库备份压缩时基本上什么事情都做不了，尤其像oracle、sybase这种；sybase的备份策略更人性但是代价的权衡其实当时就、、、、、、
  大多数情况下其实我们在做这种事情时不太可能单独什么事情都不做：可能生产环境还是会去选择Rockmq吧，毕竟中间件不像数据库-单独的服务器，cpu的资源使用上相对宽裕，尤其是对于中小型企业，硬件资源没有那么宽裕，Rockmq是个不错的选择，计算机组成原理的课程还是对于数据备份做了一些比较好的讲解，胡老师的课对kafka的压缩机制有讲解；综合下来我应当会选择Rockmq。
  几门课程同时在跟-确实发现时间上蛮吃力，为了学习天天的业余生活活在闹钟之中：可能很多东西只能等完课了再进一步梳理；只能用刘超老师的学习法-第一遍先粗学了，第一遍还是希望明白是什么回事，什么场景怎么用；老师的消息队列课程与计算机组成原理、操作系统的关系更大些，同时跟这三门就已经挤压不出时间到kafka上了，完课的时候明白选什么，为什么选算是基本达到学习这门功课的基本目标吧，学习过程中为了明白原来而交叉去学习其它两门课所付出的时间代价其实远超与预期。
  感谢老师的分享：期待老师下节课的分享。 [3赞]

- 业余草 2019-09-05 08:42:54
  Kafka设计的很精妙，但是使用的场景局限性也很大。压缩是现代网络通信中必不可少的一环，也有不少专栏都写过了。 [1赞]

- 路途 2019-09-05 19:21:56
  kafka压缩后它的offset如何计算，假如刚好要回溯的数据就在压缩包中offset如何计算

- 青舟 2019-09-05 10:56:49
  RocketMQ 在DefaultMQProducerImpl.tryToCompressMessage中判断消息长度大于4Kb就进行压缩，压缩算法是zip（java.util.zip.Deflater）

  我注意到在发送时，会判断如果消息是一个批量消息（MessageBatch），就不开启压缩。这是为什么呢？

- DFighting 2019-09-05 10:21:07
  一直学习算法都是空间换时间，但是在消息中间件和一些IO密集型的应用中还会有CPU计算资源换网络带宽/磁盘IO，刚刚看了下RocketMQ源码，在DefaultMQPullConsumerImpl.pullSyncImpl中会调用PullAPIWrapper.processPullResult，在这里会为压缩的消息进行解压缩。Producer端没找到压缩的源码，只是在checkMessage中会对消息体的长度进行限制，超过4K(网上查的)会抛出来MQClientException，猜测应该也是会压缩。也就是RocketMQ的压缩机制也是Producer压缩，Broker传输，Consumer解压缩，不同的是kaffka的压缩是基于一批一批消息的，对于CPU空闲较多的场景下会有更大的吞吐提升。

- 一步 2019-09-05 09:48:40
  现在系统使用的是rabbitmq，在发送数据时候使用的是protobuffer进行序列化，这时候还有必要开启压缩吗？如果开启了压缩会不会效果更好的？