避坑指南：Kafka集群快速扩容的方案总结

什么是数据迁移

Apache Kafka 对于数据迁移的官方说法是分区重分配。即重新分配分区在集群的分布情况。官方提供了kafka-reassign-partitions.sh脚本来执行分区重分配操作。其底层实现主要有如下三步：

1. 通过副本复制的机制将老节点上的分区搬迁到新的节点上。
2. 然后再将Leader切换到新的节点。
3. 最后删除老节点上的分区。

重分配过程中最重要的一步是数据复制。故本文用数据迁移来形容这一行为，下面来看一下数据迁移的过程。

假设topicA有3个分区，2个副本，分区和副本分布在节点1和节点2。此时加了一个节点3，如果要让3个节点均分压力，就需要从节点1，2中迁移两个分区到节点3，如下所示:

图1：迁移前

图2：迁移后

扩容场景

了解了数据迁移，我们来看下哪些场景需要进行扩容，然后有哪些方法可以实现快速扩容的效果。通常有如下两种需要紧急扩容的场景：

- 集群所有节点负载都高，需要快速扩容。
- 集群内某几台节点负载很高，需要降低这些节点的压力。

首先，来谈谈什么是节点压力。从我们的运营经验来说，Kafka集群的压力通常体现在磁盘util、CPU、网卡三个指标上。正常来说，通过加节点都可以解决这三个指标带来的问题。但是，从精细化运维的角度来说，可以有针对性地解决负载问题，达到不扩容就可以快速降低集群压力的目的。比如通过参数调优，踢掉某台故障节点/某块坏盘等等。

关于精细化运维我们在后续的文章再展开，
本文主要是讨论如何能通过加节点实现快速实现集群扩容。

找到Top主题

根据二八法则和现网运营来看，在大多数集群中，头部效应一般都比较明显，即大部分压力都是由少量Topic带来的。所以一般只要解决导致问题的头部主题，就会事半功倍的解决问题。给大家看一下典型的现网集群的Topic流量排行示意图，集群的流量集中在下面的Top主题中：

Topic名称	分区数	副本数	Topic生产流量(MB/分钟)
[redacted]	500	2	65913.41
[redacted]	800	2	60334.97
[redacted]	600	2	47540.46
[redacted]	600	2	46322.11
[redacted]	600	2	39783.08
[redacted]	40	2	21436.91
[redacted]	400	2	13906.97
[redacted]	250	2	11125.51
[redacted]	100	2	1189.83
[redacted]	5	2	8.51
[redacted]	2	2	2.19
[redacted]	1	2	0.20
[redacted]	1	2	0.20
[redacted]	10	2	0.00
[redacted]	1	2	0.00
[redacted]	5	2	0.00
[redacted]	300	2	0.00
[redacted]	10	2	0.00
[redacted]	2	2	0.00

图3：现网某集群topic流量排序图

另外，kafka-reassign-partitions.sh 分区迁移工具支持分区粒度的迁移，也可以支持整个Topic的迁移。所以在进行集群扩容的时候，不需要迁移所有的Topic。可以迁移某几个Topic或者某几个Topic中的某些分区。这样尽量减少需要搬迁的数据量。

那怎么样找到Top主题呢？

1. 如果系统内部有通过Broker暴露的的Jmx接口采集Topic入流量指标，那么对这些流量做一个排序，可以快速的找到目标主题。

2. 也可以写一个shell脚本，使用cmdline-jmxclient.jar工具实时的取到所有topic的流量，然后再做排序。这个方法比较繁琐，仍需考虑Topic的分区是否分布在不同的Broker上，是否需要做汇总等。

3. 如果不想用2的办法，有一个简单的办法可以大概看出流量的分布。先进入broker上的数据目录，然后查看每个分区的堆积的数据量大小。比如执行如下命令：ll -h /data/kafka_data/。根据堆积的情况来判断哪些Topic的流量相对较大。

当然，如果集群中所有主题的流量都非常平均，那就对所有的Topic一起处理。接下来我们来讨论下当遇到紧急扩容的需求时，有哪些方案可以选择。

以下方案的核心思想：迁移尽量少的数据，或者不迁移实现压力的转移。

方案一：降低数据保留时间，精准迁移

这个方案是大家最常用的方法，也是操作起来最简单的。无论集群整体压力都高还是某些Broker的压力大，都可以通过这个方案来执行扩容。一般执行如下四步：

1. 找出需要操作的Topic

2. 调整这些Topic的数据保留时间

3. 对这些Topic进行迁移

4. 等待迁移完成，完成Leader切换，均分集群压力。

此时当整个集群压力都很大的时候，这个方案下最好的处理方式是：调整流量最大的分区的保留时间，然后针对每条broker迁移几个分区到新的节点上。比如，从每台Broker挑出3个分区，然后迁移到目标节点上。如果只有几台Broker压力大，也是一样的处理方式。

这种方式有两个缺点：

1. 业务数据的保留时间不一定能调短，因为调短了数据保留时间，这些数据就会被删除，无法恢复。所以，当业务不允许删除数据的时候，这个方案就不允许使用。
2. 当业务允许调整数据保留时间，比如调整到1个小时。此时还有一个小时的数据需要迁移，如果此时当前节点的负载已经很高了。此时副本拉取数据即会增加当前节点的负载，导致集群更加无法提供正常服务。当前节点压力大的话，可能导致新副本同步数据比较慢，会导致集群的压力没法快速降下来。

那有没有方案可以解决这个问题呢？我们再来看下一个方案。

方案二：往指定节点上添加分区，均分压力

如方案一所示，当整个集群压力都很大时，扩容节点后，因为数据迁移的方案无法使用，新节点无法承担压力，集群负载也降不下来。

此时可以通过扩容分区到新节点，将流量导到新的节点，让新的节点也可以承担流量。一般执行如下三步：

- 找出需要操作的Topic
- 评估这些Topic需要导多少流量到新节点。这一步比较好做，一般比如要把30%，50%的流量导入到新节点，大概评估一下即可。
- 扩容目标Topic的分区数，将这些新增的分区指定扩容到新的节点上。

因为扩容Topic的目标分区是秒级完成的，所以不需要等待。在没有分区变更的情况下客户端的metadata机制默认是每隔30s自动更新一次，所以客户端会很快感知到分区变化，默认的生产者写入策略是轮询的，此时新增的流量会自动流入到新节点，原先的节点的负载很快就降下来了。

方案二的核心点：新增扩容分区，比如指定添加到目标节点。如果只是扩容分区，而这些分区还是落到老的节点上，是解决不了问题的。因为一般情况下Topic的分区的流量都是均匀的，假设Topic当前分区是100，想让新的节点承担该Topic
50%的压力，可以将该Topic的分区数扩容到200。如下图所示：

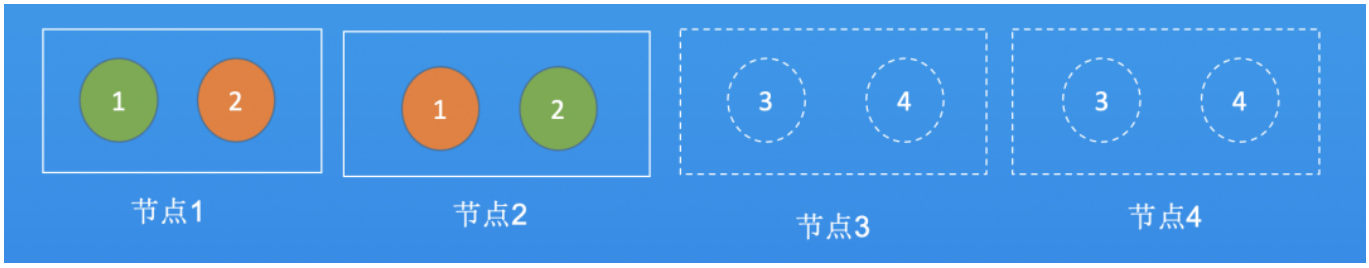

图4：分区扩容示意图

这种方式有如下几个缺点：

1. 如果业务逻辑限制了分区数不可变更，比如业务根据分区数做了哈希，根据哈希值往分区写数据，又需要要求数据保持有序。那么该方案就行不通了。但是一般情况下，有如上业务场景的Topic数据量都比较小，不会成为瓶颈Topic。反之，成为瓶颈的Topic，数据量都很大，一般都允许扩容分区。所以，这点通常情况下不会成为限制，但是在操作前最好和业务确认下。

2. 这种行为针对单个Topic不能多次使用。因为主题的分区一般不能删除，因为删除分区后，分区中的数据也会丢失。如果在单个Topic多次使用的策略下，该Topic的分区数就会膨胀到很大。比如每次都需要导出50%的流量，则需要再添加100个分区，此时总分区数量就需要为200。以此类推，则总分区数的变化趋势为：100，200，400，800，1600.....。

3. 如果客户端写入有倾斜，即客户端指定了写入策略，只针对某些分区写入，现在的方案就会失效。遇到这种情况，只能协调客户端处理。

虽然这个方案有以上几个缺点，但是整体方案可以应对90%以上的紧急情况。所以，在不用迁移就降低集群负载的情况下，这个方案是很好用的。

方案三：切换Leader，降低单机负载

方案二比较适合整个集群负载较高的场景，或者因为某些头部Topic的流量均匀的集中在部分节点的情况下。现在我们来看一下如下场景：

假设集群有20台节点，节点的机型和规格是一样的，从网卡进出流量上来看，流量是均衡的。但是因为节点间性能的差异(原因可能是机型年限不一样，不同节点上Topic的业务形态不一样等等)，导致某几台节点的负载很高。

此时，如果通过方案二处理，就会有种杀鸡用牛刀的感觉，而且效果并不好。为什么呢？

假设头部流量的Topic分区均匀分配在了20台节点上。如果要通过扩容分区降低其中几台的负载，因为生产端是均匀写入的，则需要扩容很多倍的分区到新的节点上，才能把这几台的流量降下来。这会导致该Topic的分区数急速膨胀。

在这种情况下，我们可以尝试切换分区 Leader的方案来降低单机的压力。这个操作比较简单，因为Leader切换的速度是秒级的，所以见效也很快。思路如下：

1. 找出负载高的节点上的所有Leader
2. 将节点上分区的Leader切换到负载较低的Follower节点上

这个操作的原理是：Kafka分区都在Leader进行读写，从单个分区看，即分区Leader所在节点的负载就会比Follower的节点负载高。所以，思路就是将负载高的节点上的Leader变为Fllower，降低单机压力，来看下图：

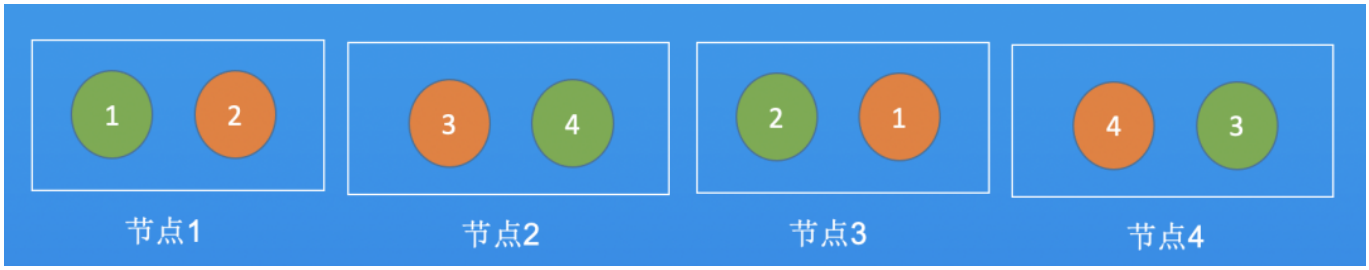

图5：切换Leader前

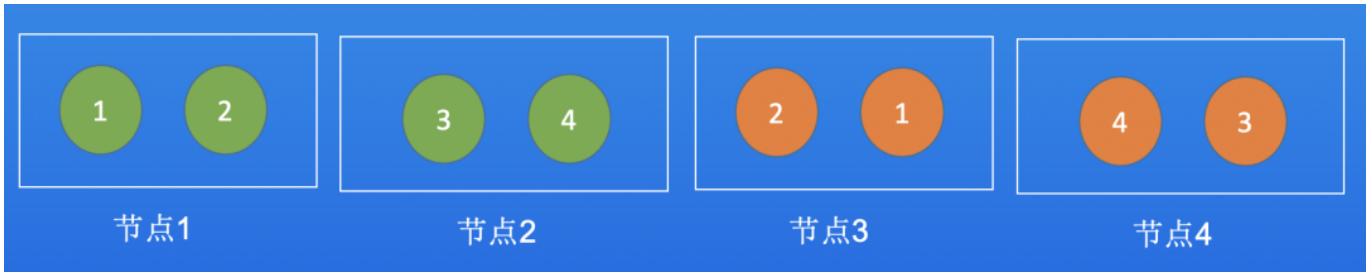

图6：切换Leader后

这种方式的缺点如下：

- 这种方式适用于部分节点负载较高的情况，因为负载会转移到Follower上，如果Follower

的负载本身就很高，则这种手段会加大Follower的压力。
- 当Leader出现负载较高的时候，副本可能会掉出ISR。因为Leader负载高，Follower拉取数据慢，导致副本跟不上Leader，掉出ISR。当出现这种情况，就不能切换Leader，强行切换的话，会出现数据截断，导致数据丢失。

我们继续来看一下，有没有其他方法可以解决这种问题。

方案四：单副本运行，降低单机负载

当出现方案三中的无法处理的情况时，即无法使用切换Leader的手段降低压力时。我们可以通过将高负载节点上的分区Follower剔除，将分区切换为单副本运行，来临时降低节点的压力，让集群暂时的快速回归正常。

思路如下：

1. 找出负载高节点上的所有Follower
2. 将节点上的Follower暂时移除

这个操作的原理：分区的Follower会不断的从Leader同步数据，即从Leader拉取数据到本地进行报错。这个行为会占用CPU，网卡，磁盘等资源，如果把这一部分流量去掉，则会把节点上这部分负载空出来。能快速地降低节点压力。

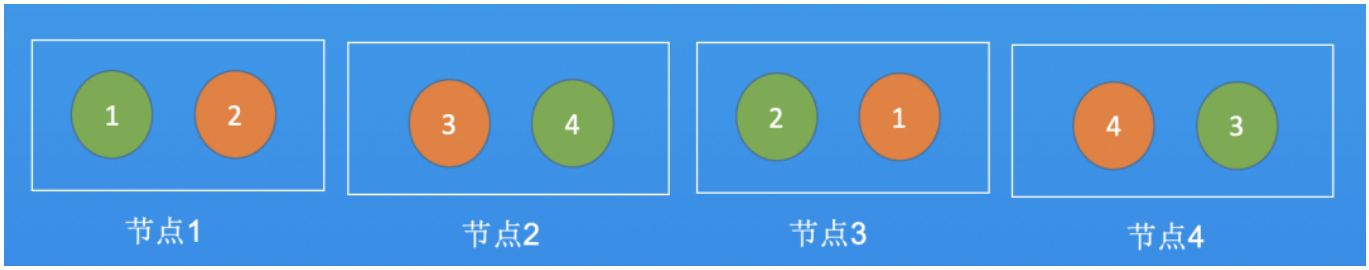

图7: 切掉Follower前

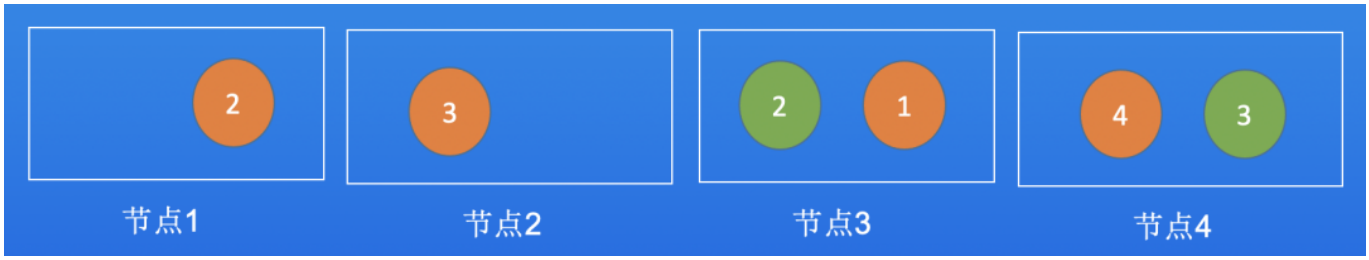

图8: 切掉Follower后

这种方式的缺点：切掉Follower后，分区就是单副本运行，没有备份。如果这台节点挂了，那服务就会出现异常，可能出现数据写入异常。

这个步骤还有一个执行风险：切掉Follower，节点负载降低后，一定要记得把停掉分区分区的Follower重新启动，否则单副本运行的数据，没有备份，很容易出现故障。

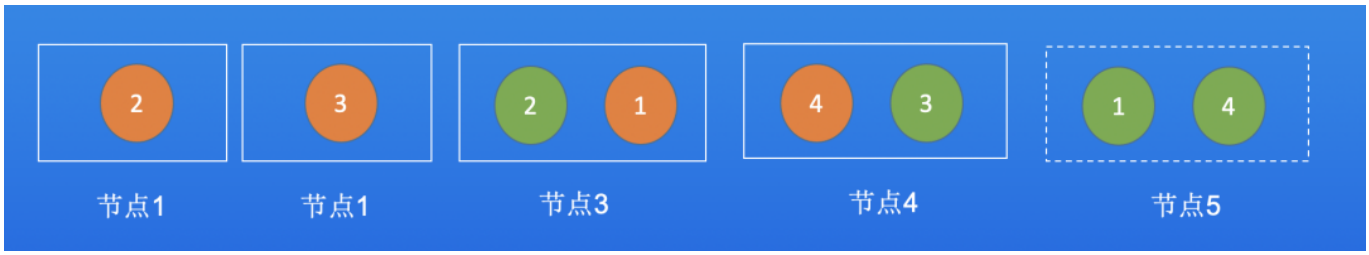

图9: 在其他节点拉起Follower

方案五：其他思路

除了上面提到的扩容方案。还有其他一些可选的解决方法。这些方案的缺点都比较明显，比如可能出现数据丢失，业务中断，或者需要客户端配合，但胜在见效快。

选择这种方案需要从业务来考虑：为了快速恢复业务，可以允许一定程度的数据丢失和服务中断。所以方案还是存在一定的风险，需要和业务侧讨论后，再决定可否执行。一起简单的来看下方案思路及其缺点：

1. 删除Topic重建

因为创建Topic的时候，Kafka默认的算法会将分区均匀的放到所有节点上。所以可以通过删除，重建Topic的形式快速扩容，均分压力。操作步骤如下：

1. 往集群添加新节点

2. 删除流量大的Topic

3. 重建Topic

当Topic创建完成后，流量就会均匀写入到所有节点。整个过程最大的优点就是恢复快，几分钟就完成了。缺点就是：

1. 如果删除Topic前，消费还有堆积，则这些数据就不会被消费到，会丢失。

2. 在重建Topic的过程，客户端会报UnknownTopicOrPartitionException错误

2. 垂直升配，替换为更高规格的节点

如果集群负载的压力是在Cpu或者网卡，并且是使用云上的虚拟机搭建的kafka集群，可以利用CVM的热迁移能力，垂直升配虚拟机的规格。

这个方法也是大家通常的做法。但是在做这个操作的时候，有一个注意点，需要关注这个节点上是否有未同步副本。也需要关注一下集群参数unclean.leader.election.enable 参数的值。

如果存在未同步副本，当 unclean.leader.election.enable=true 时，则表示允许选择不再ISR中的副本为Leader。此时如果垂直升配，则会出现未同步副本当选为Leader，出现数据阶段，出现数

据丢失。

如果存在未同步副本，当 unclean.leader.election.enable=false 时，则表示不允许选择不再ISR中的副本为Leader。此时在CVM升配过程，Broker重启的过程中，就会出现服务中断。但是不会数据截断导致的数据丢失。

3.客户端控制写入分区策略

这种方法严重依赖客户端。需要客户端有指定往哪些分区写入数据的能力。因为大部分业务的Producer客户端，基本都是直接调用官方SDK的Produce方法进行数据发送，数据会均匀的写入到所有分区。所以大部分客户端没有这个能力。而一旦业务的客户端可以动态的指定分区写入数据(官方SDK自带指定分区写入的功能)。降低负载就会变的很简单，不需要进行数据迁移。

即让客户端指定数据写入到负载较低的分区，就可以降低高负载节点的压力。

4.从最新的数据迁移，丢弃老数据

这个思路是之前网上看到的一个方案，适用于某些允许数据丢失的场景。因为Kakfa默认的迁移机制，新副本都是从分区最早的可用的位置拉取数据，然后进行同步。

在某些场景，如数据量很大，数据保留时间却很短的场景中，可能出现当副本同步完数据后，这些数据其实已经过期了，同步完数据就需要立即删除。则这个迁移的行为即是多余的。所以还不如直接从最新的数据进行同步，这样在新添加的节点上的新副本立即就可以加入ISR。则立即可以Leader提供服务。

这个方案的缺点是：

1. 如果新加入的副本立即进入ISR，同时又成为Leader，则会出现数据丢失，即从加入ISR那一刻起，往前推这个Topic的数据保留时间这段时间的数据都会丢失。

2. 需要修改Broker默认的数据复制机制，对研发能力要求较高。

关于架构的一些想法

看到这里，会发现整体下来，Kafka的扩容还是不够灵活，快捷和方便。操作起来比较麻烦。有没有更好的方案呢？

首先来看下造成扩容问题的原因，是受Kafka本身架构的限制。Kafka 是以分区为读写单位，分区是和节点绑定的，这些数据会写入到元数据存储中。此时一旦计算层(CPU/网卡)或存储层(util)出现瓶颈，是没办法让其他节点承载压力的。如果要解决这个问题，Kafka在架构上要做很大的改动。

从架构的角度出发，我个人理解，解决的思路就是：计算存储分离 + 存储分段。这一点Apache Pulsar就做的很好。我们来简单看一下Pulsar的做法。来看下图：

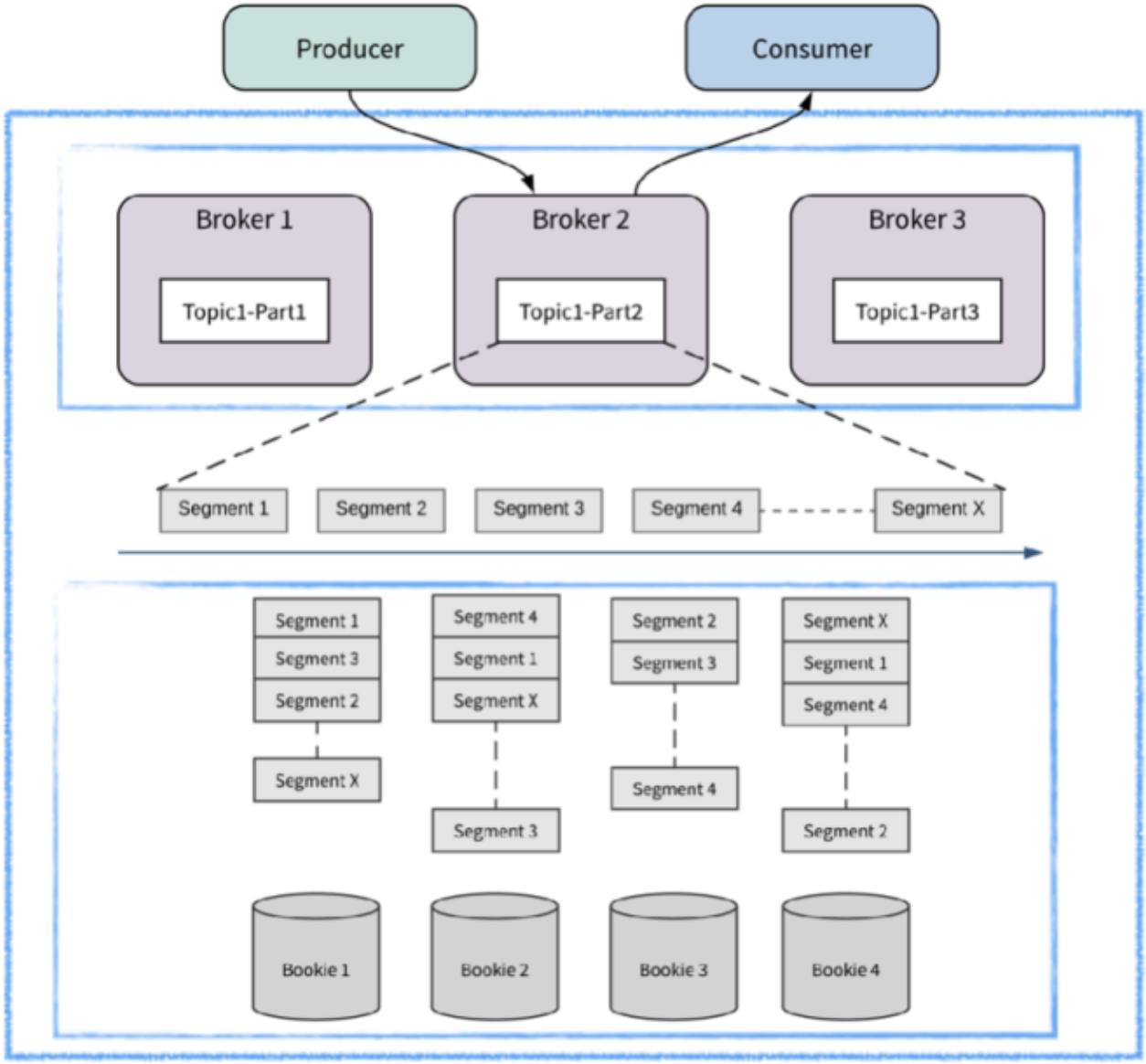

图10：Pulsar 简要架构图

计算存储分离：解决的是计算压力的快速转移
。计算节点和存储节点是分开的。计算节点只负责计算逻辑的处理，是无状态的节点。当节点出现瓶颈，可以快速横向扩容。

存储分段
：解决的主要是存储层IO压力的快速转移。Pulsar使用Bookeeper作为存储层，Pulsar将逻辑上的分区，在实际存储层面，分为多个段(segment)进行管理和存储。如果出现某个存储的机器有瓶颈，直接禁用该机器上segment，在新的机器上拉起新的Segment即可。

总结一下，一旦Pulsar集群遇到上面说的Kafka集群类似的瓶颈，从扩容的角度来说，会更优雅和便捷。这是架构自身带来的优势。

总结

本
文列
举了一些
快速扩容的手段和
方案，帮助大家尽量避免迁移、尽量
降低需要迁移的数据量
。大家遇到问题的时候，可以根据实际的业务场景选择适合的方案进行处理。

Apache Kafka 的扩容复杂度源于Kakfa存算一体的架构。即数据生产和消费都是以分区为单位的，而分区从创建开始就会和某一个节点进行绑定。如果没有进行分区迁移，则分区和节点的绑定关系不会发生改变。当遇到节点出现性能问题时，这个分区也会受到影响，从而产生本文讨论的问题。

在云原生架构，存算分离成为了一种趋势。在消息队列领域，最近新兴的开源消息队列Apache Pulsar在架构上实现了存储计算分离，通过Apache Bookeeper
实现对分区数据的分段分节点存储，从而避免了Kakfa遇到的扩容迁移问题。