简单理解 Kafka 的原理框架

作者：Dadong-Zhang
原文链接：https://ld246.com/article/1567769128331
来源网站：链滴

<h2 id="简单理解Kafka的原理框架">简单理解 Kafka 的原理框架</h2>
<h2 id="Kafka原理简介">Kafka 原理简介</h2>
<h3 id="一句话简介">一句话简介</h3>
Kafka 是一个高吞吐量的，在分布式集群中负责消息管理的组件，如消息的分发保存接收等操作使用 java 编写。
<h3 id="相关术语-1-">相关术语[1]</h3>
<h4 id="Producer">Producer</h4>
消息生产者，发布消息到 Kafka 的某个 Topic。
<h4 id="Consumer">Consumer</h4>
消息消费者，在 Kafka 的 Topic 中接收消息。
<h4 id="Consumer-Group">Consumer Group</h4>
消费者的分组，未分组的消费者属于默认组。消费者以组为单位消费 Topic 消息，因此一个组中阅的 Topic 相同，已订阅的 Topic 中的每个 Partition 会平均分给这个组中的消费者，Consumer 比artition 多时会产生空闲的 Consumer。
<h4 id="Message">Message</h4>
Kafka 的消息，拥有 Topic，key，Value 三种内容，Topic 必须是字符串而 KeyValue 可以是意数据类型，但在同一 Topic 下必须统一。
<h4 id="Topic">Topic</h4>
逻辑上的分区概念，每一条消息和每一个生产者消费者都要有归属的 Topic，在物理上不同的 Toic 的消息会独立保存。
<h4 id="Partition">Partition</h4>
物理上对 Topic 的细化分区，也是分布式系统的共有特点，每个 Partition 可以有多个副本，Kafa 使用哈希算法将拥有相同 key 的消息映射到同一个 Partition 中，分区保存消息提高吞吐量。哈希法的生成也叫 Balance。
<h4 id="Replica">Replica</h4>
Partition 的副本就是 Replica，属于分布式 Kafka 的容灾机制，消息存储的最小物理分区的副，拥有一个 leader 和若干 follower，之间的数据同步由 leader 完成，
<h4 id="Balance">Balance</h4>
Kafka 生成将同一 Topic 下不同 Key 的消息映射到某一 Partition 下的哈希算法，初始化时触发当增加新的 Partition、Consumer 增加关闭宕机，Coordinator 宕机时会产生 Rebalance。Balance 和 Rebalance 期间 Kafka 暂停消息收发。
<h4 id="Broker">Broker</h4>
Kafka 消息集群中的服务器就是 Broker。
<h4 id="Controller">Controller</h4>
Kafka 对于生产阶段的控制者，自动分配给某一个 Broker，负责 partition 的分配，leader 的举。
<h4 id="Coordinator">Coordinator</h4>
Kafka 对于消费阶段的协调者，自动选择一个 Broker，用于给 Consumer 分配 Partition
<h3 id="容灾机制">容灾机制</h3>
当 broker 宕机后，controller 就会给受到影响的 partition 中读取对应 partition 的 ISR（in-syc replica 已同步的副本）列表，选一个出来做 leader。[4]
<h3 id="消息可靠性">消息可靠性</h3>
这里的策略，服务端这边的处理是 follower 从 leader 批量拉取数据来同步。但是具体的可靠性是由生产者来决定的。
生产者生产消息的时候，通过 request.required.acks 参数来设置数据的可靠性。
<table>
<thead>
<tr>
<th>acks</th>
<th>What happen</th>
<th></th>
</tr>
</thead>
<tbody>

<tr>
<td>0</td>
<td>发过去就完事了，不关心 broker 是否处理成功，可能丢数据。</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>当写 Leader 成功后就返回,其他的 replica 都是通过 fetcher 去同步的,所以 kafka 是异步写，备切换可能丢数据。</td>
<td></td>
</tr>
<tr>
<td>-1</td>
<td>要等到 isr 里所有机器同步成功，才能返回成功，延时取决于最慢的机器。强一致，不会丢数据</td>
<td></td>
</tr>
</tbody>
</table>
在 acks=-1 的时候，如果 ISR 少于 min.insync.replicas 指定的数目，那么就会返回不可用。
这里 ISR 列表中的机器是会变化的，根据配置 replica.lag.time.max.ms，多久没同步，就会从 ISR 列表中剔除。以前还有根据落后多少条消息就踢出 ISR，在 1.0 版本后就去掉了，因为这个值很难，在高峰的时候很容易出现节点不断的进出 ISR 列表。
从 ISA 中选出 leader 后，follower 会从把自己日志中上一个高水位后面的记录去掉，然后去和 leader 拿新的数据。因为新的 leader 选出来后，follower 上面的数据，可能比新 leader 多，所以要取。这里高水位的意思，对于 partition 和 leader，就是所有 ISR 中都有的最新一条记录。消费者最只能读到高水位。[4]
kafka 支持 3 种消息投递语义
At most once：最多一次，消息可能会丢失，但不会重复
At least once：最少一次，消息不会丢失，可能会重复
Exactly once：只且一次，消息不丢失不重复，只且消费一次（0.11 中实现，仅限于下游也是 kafka
在业务中，常常都是使用 At least once 的模型，如果需要不重复的话，往往是业务自己实现。
At least once
先获取数据，再进行业务处理，业务处理成功后 commit offset。
1、生产者生产消息异常，消息是否成功写入不确定，重做，可能写入重复的消息
2、消费者处理消息，业务处理成功后，更新 offset 失败，消费者重启的话，会重复消费
At most once
先获取数据，再 commit offset，最后进行业务处理。
1、生产者生产消息异常，不管，生产下一个消息，消息就丢了
2、消费者处理消息，先更新 offset，再做业务处理，做业务处理失败，消费者重启，消息就丢了

Exactly once
思路是这样的，首先要保证消息不丢，再去保证不重复。所以盯着 At least once 的原因来搞。先想出来的：

生产者重做导致重复写入消息——生产保证幂等性
消费者重复消费——消灭重复消费，或者业务接口保证幂等性重复消费也没问题

由于业务接口是否幂等，不是 Kafka 能保证的，所以 Kafka 这里提供的 exactly once 是有限制，消费者的下游也必须是 Kafka。所以一下讨论的，没特殊说明，消费者的下游系统都是 Kafka（注:用 Kafka connect，它对部分系统做了适配，实现了 exactly once）。

<h4 id="生产幂等性">生产幂等性</h4>
思路是这样的，为每个 producer 分配一个 pid，作为该 producer 的唯一标识。producer 会每一个 <topic,partition> 维护一个单调递增的 seq。类似的，broker 也会为每个 <pid,topic partition> 记录下最新的 seq。当 req_seq == broker_seq+1 时，broker 才会接受该消息。因：

消息的 seq 比 broker 的 seq 大超过时，说明中间有数据还没写入，即乱序了。
消息的 seq 不比 broker 的 seq 小，那么说明该消息已被保存。

<h3 id="安全性">安全性</h3>
Kafka 的安全性体现在三点，认证机制、加密和权限管理。通过给集群主机分发签名证书，认证制可以保证连接在 Kafka 集群中的机器都是合法的，若没有认证机制，任意一台得到某一台服务器地的非法服务器都可以加入集群，这非常影响整个集群主机业务的稳定和数据的安全性。加密是消息数以非明文方式传输，使截取到的数据难以破译。限制 Client 的权限也可以使 Kafka 更加安全。
Kafka 的认证机制包括 SSL 协议和 SASL 协议，而加密机制只有 SSL 协议，加密范围包含 Broker 与 Tools 之间、Broker 与 Client（Consumer 和 Producer）之间、Broker 之间三个方面，可以分决定是否加密传输。
<h2 id="Kafka-MirrorMaker">Kafka MirrorMaker</h2>
<h2 id="一句话简介-">一句话简介</h2>
在 Kafka 集群之间镜像复制传递某 Topic 消息的 Kafka 组件。对于迁移的 topic 而言，topic 字一样，partition 数量可以不一样，消息的 offset 会不一样。[2]
因为 MirrorMaker 是 Kafka 的组件，其中相当多的概念都是 Kafka 本身的，因此相关信息都在一模块中讨论。本章只讨论 MirrorMaker 所特有的。
<h2 id="配置说明-2-">配置说明[2]</h2>
--consumer.config # 消费者配置。
--producer.config # 生产者配置。
--whitelist #需要 mirror 的 topic，支持 Java 正则表达式，例如'ABTestMsg|AppColdStartMs’，或使用逗号分隔。
--blacklist #不需要拷贝的 topic，支持 Java 正则表达式
--num.producers #producer 数量，默认为 1
--num.streams #consumer 数量，默认为 1
--queue.size #consumer 和 producer 之间缓存的 queue size，默认 10000
<h2 id="相关术语">相关术语</h2>
<h3 id="Source-Kafka-Cluster">Source Kafka Cluster</h3>
Kafka MirrorMaker 监听的源集群
<h3 id="Target-Kafka-Cluster">Target Kafka Cluster</h3>
Kafka MirrorMaker 复制的镜像消息的目标集群
<h2 id="相关特性">相关特性</h2>
在 Target Cluster 没有对应的 Topic 的时候，Kafka MirrorMaker 会自动为我们在 Target Cluser 上创建一个一模一样(Topic Name、分区数量、副本数量)一模一样的 topic。如果 Target Cluster 存在相同的 Topic 则不进行创建，并且，MirrorMaker 运行 Source Cluster 和 Target Cluster 的 Tpic 的分区数量和副本数量不同。[3]
可以将 MirrorMaker 当作一个 Consumer 和 Producer 结合的管道，当 Consumer 接收到源群相关 Topic 的消息时，传递给 Producer，让它发送给目标集群的同名 Topic。
<h2 id="可靠性-吞吐量-安全性">可靠性、吞吐量、安全性</h2>
因为这三点某一项的改变都会影响其他两项，所以我在同一标题下讲述它们的相关性。
数据可靠性，即指数据不重复，不遗漏，可靠性的提升必然降低数据的吞吐量。数据安全性，指据不易被截取、破译，Kafka 集群不会被攻击控制，数据的加密和 Kafka 组件间的认证机制因为操作多计算量大也会影响数据的吞吐量。
<h2 id="数据可靠性">数据可靠性</h2>
根据 MirrorMaker 是部署在源服务器或目标服务器的情况，产生了拉模式和推模式的概念。拉式比推模式更有可靠性保障，因为一旦两个集群失去连接，拉模式只是暂时无法获取到数据，而推模有可能会将数据推丢（即自己认为发送成功而没有收到响应）。因为消费组只会消费一条消息一次这特点，将多个 MirrorMaker 部署在不同的 Broker，而他们又归属于同一个消费者组下，可以简单的

高可靠性容错率和数据吞吐量。
<h2 id="数据吞吐量">数据吞吐量</h2>
提高数据吞吐量的方法有很多，基于 Kafka 的消费组特性的吞吐量优化在上一节已经提到，这其属于增加 MirrorMaker 的进程数量。根据 Partition 的特性，可以通过增加 MirrorMaker 的 Consuer 来增加数据的接收能力，这相当于增加了线程。Consumer 的数量可以是所有监听的 Topic 的 Partition 数量总和，这样在理想的分区指派策略下（如：org.apache.kafka.clients.consumer.RoundRobnAssignor）每一个 Consumer 负责监听一个 Partition 的数据。
而对于数据吞吐量的进一步提高，需要使用 Monitor 对 Kafka 的进一步监控得到数据和对实际况的考虑，才能有更详细的策略。
<h2 id="数据安全性">数据安全性</h2>
然而和提升可靠性的推拉模式相反，当对于 MirrorMaker 使用 SSL 加密时， 消费者相比生产者说性能受到的影响更大[5]。因此增加可靠性机制，如 acks=all 和足够的重试次数，然后再将 Mirroraker 放到源集群也是可以的，这样数据从 Broker 到 MirrorMaker 的消费者是可以不需要 SSL 加密，只需要生产者将消息发送给目标集群时加密就可以了。
官方文档[8]将数据加密和认证混在一起讲述，且内容简单只有配置方法，因此没有找到只针对 MirorMaker 的加密方式，只有将其视作 Client 的数据加密方式和认证配置，详细的配置信息需要解读码。结合各 IT 公司已有的 Kafka 集群资料[7]，加密模块可能需要特殊定制，才能实现只在外网传输使用数据加密，而内网集群只做身份认证的方案（即只对 MirrorMaker 的数据传输进行加密），这方案可以在保证安全性的同时减小加密对服务器性能的消耗而影响吞吐量与延迟度。
<h2 id="参考文献链接">参考文献链接</h2>
[1] https://baike.aidu.com/item/Kafka/17930165?fr=aladdin
[2] https://cloud.tencent.com/developer/article/1358933
[3] https://blog.csn.net/u010003835/article/details/86611070
[4] https://www.jianshu.com/p/d3e963ff870
[5] http://www.dengshenyu.com/分布式系统/2017/12/23/kafka-data-mirror.html
[6] https://blog.csdn.ne/mnasd/article/details/82760508
[7] https://www.jianshu.com/p/ec8954ec785
[8] http://kafka.apache.org/documentation/#security