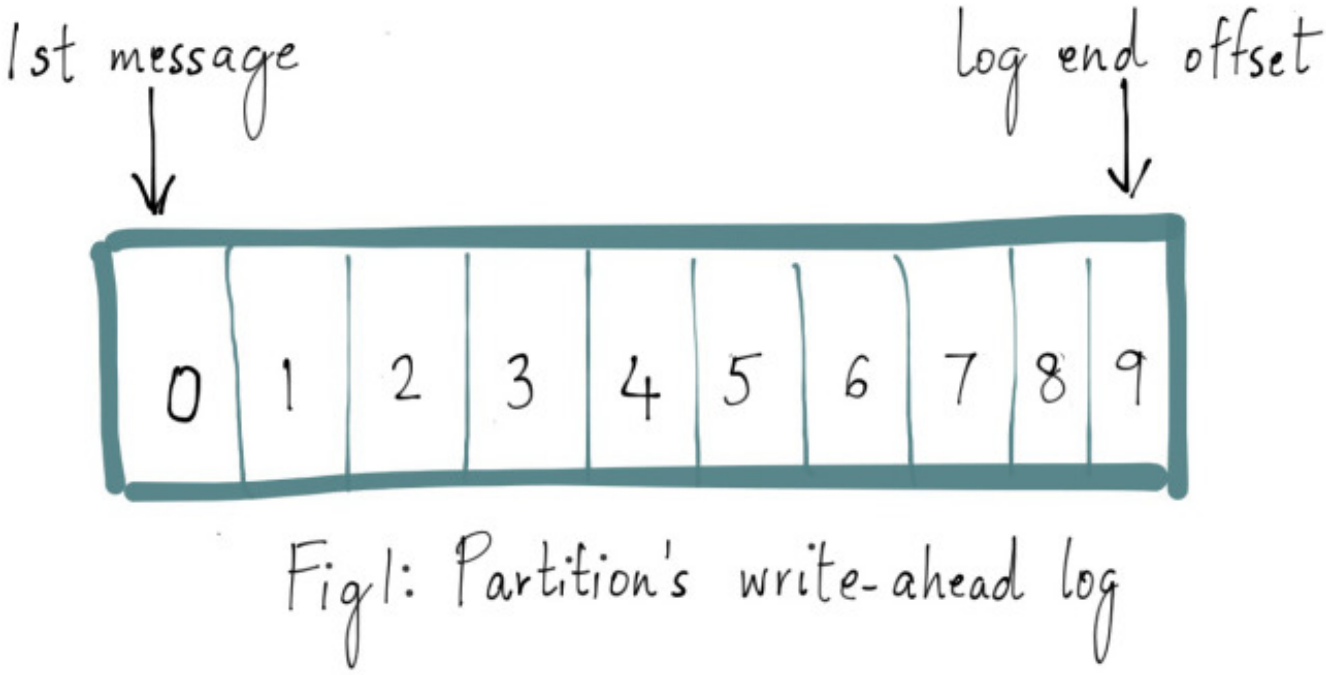
一文了解 Kafka 的副本复制机制

让分布式系统的操作变得简单，在某种程度上是一种艺术，通常这种实现都是从大量的实践中总结得到的。Apache Kafka 的受欢迎程度在很大程度上归功于其设计和操作简单性。随着社区添加更多功能，开发者们会回过头来重新思考简化复杂行为的方法。

Apache Kafka 中一个更细微的功能是它的复制协议（replication protocol）。对于单个集群上不同大小的工作负载，调整 Kafka replication 以让它适用不同情况在今天来看是有点棘手的。使这点特别困难的挑战之一是如何防止副本从同步副本列表（也称为ISR）加入和退出。从用户的角度来看，这意味着如果生产者（producer）发送一批“足够大”的消息，那么这可能会导致 Kafka brokers 发出多个警报。这些警报表明某些主题“未被复制”（under replicated），这意味着数据未被复制到足够多的 brokers 上，从而增加数据丢失的可能性。因此，Kafka cluster 密切监控“未复制的”分区总数非常重要。在这篇文章中，我将讨论导致这种行为的根本原因以及我们如何解决这个问题。

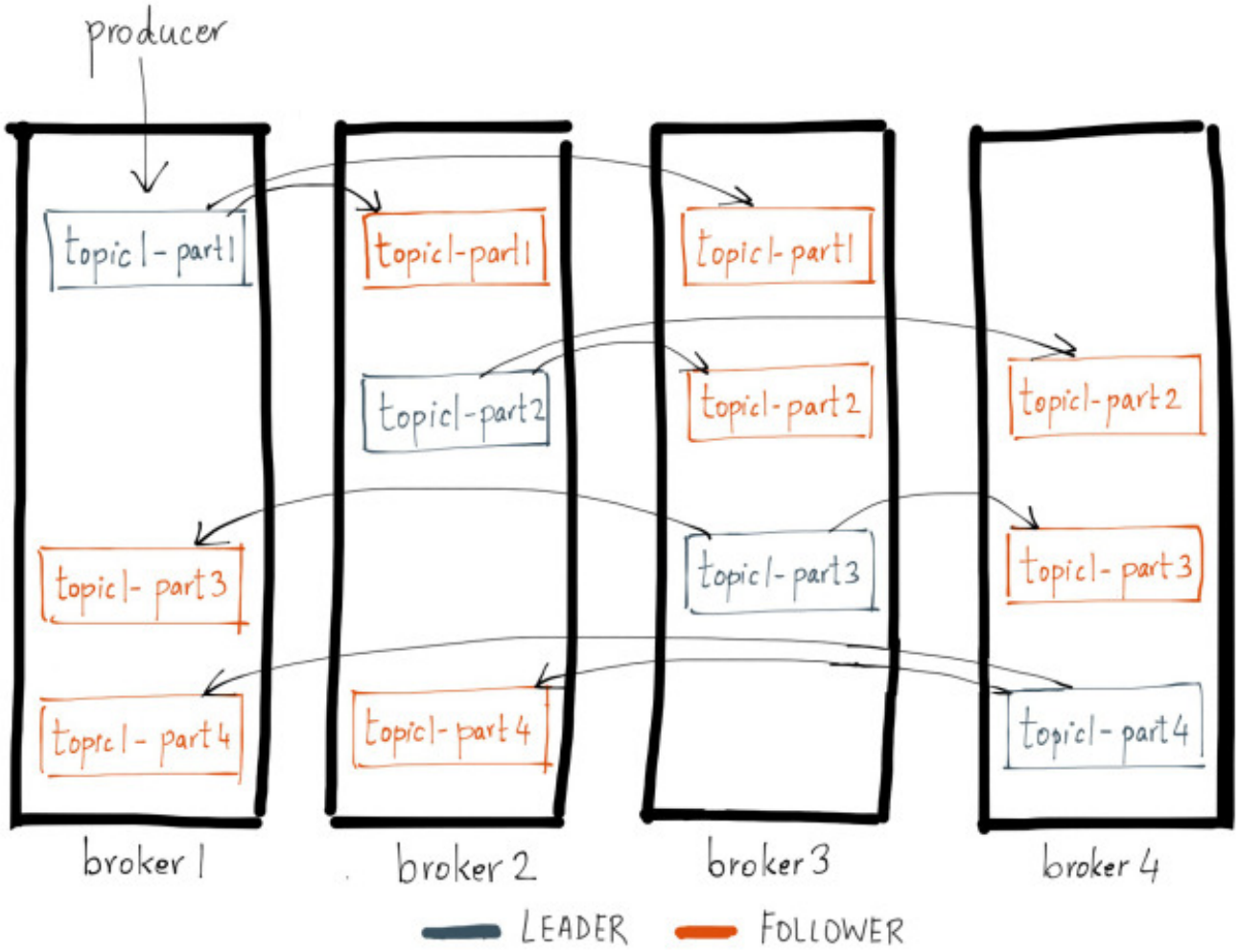
一分钟了解 Kafka 复制机制

Kafka 主题中的每个分区都有一个预写日志（write-ahead log），我们写入 Kafka 的消息就存储在这里面。这里面的每条消息都有一个唯一的偏移量，用于标识它在当前分区日志中的位置。如下图所示：



如果想及时了解Spark、Hadoop或者Hbase相关的文章，欢迎关注微信公众号：iteblog_hadoop

Kafka 中的每个主题分区都被复制了 n 次，其中的 n 是主题的复制因子（replication factor）。这允许 Kafka 在集群服务器发生故障时自动切换到这些副本，以便在出现故障时消息仍然可用。Kafka 的复制是以分区为粒度的，分区的预写日志被复制到 n 个服务器。 在 n 个副本中，一个副本作为 leader，其他副本成为 followers。顾名思义，producer 只能往 leader 分区上写数据（读也只能从 leader 分区上进行），followers 只按顺序从 leader 上复制日志。



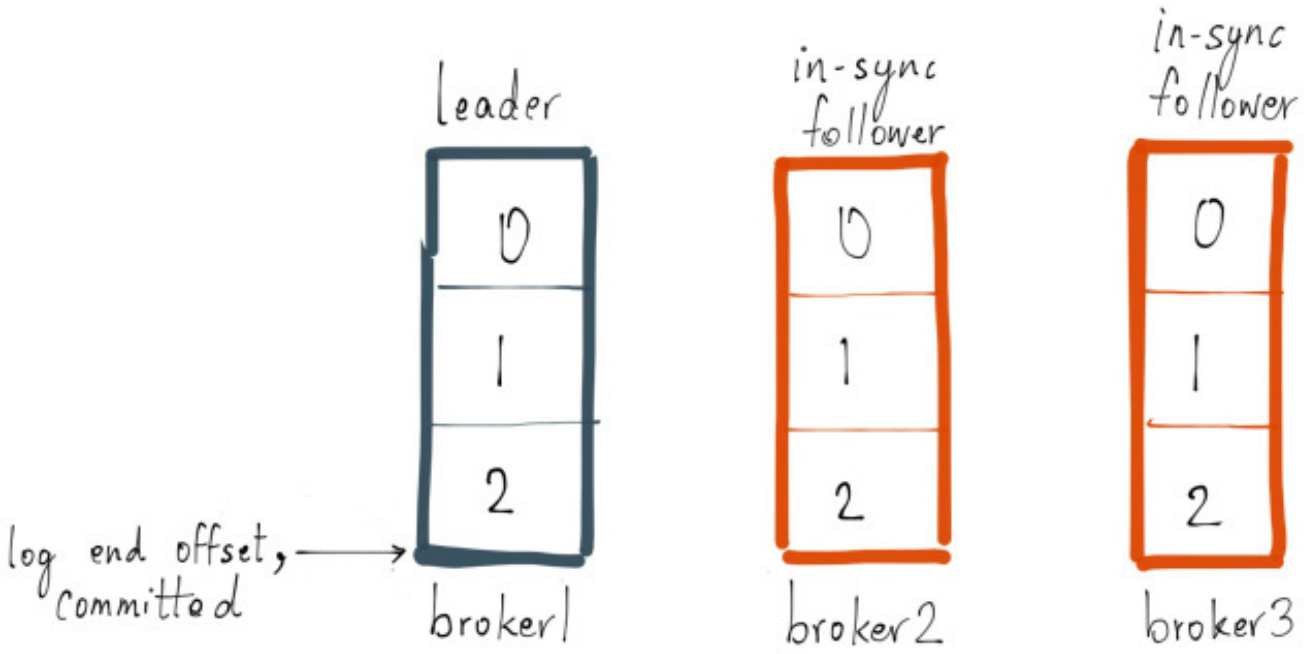
日志复制算法（log replication algorithm）必须提供的基本保证是，如果它告诉客户端消息已被提交，而当前 leader 出现故障，新选出的 leader 也必须具有该消息。在出现故障时，Kafka 会从挂掉 leader 的 ISR 里面选择一个 follower 作为这个分区新的 leader ；换句话说，是因为这个 follower 是跟上 leader 写进度的。

每个分区的 leader 会维护一个 in-sync replica（同步副本列表，又称 ISR）。当 producer 往 broker 发送消息，消息先写入到对应 leader 分区上，然后复制到这个分区的所有副本中。只有将消息成功复制到所有同步副本（ISR）后，这条消息才算被提交。由于消息复制延迟受到最慢同步副本的限制，因此快速检测慢副本并将其从 ISR 中删除非常重要。 Kafka 复制协议的细节会有

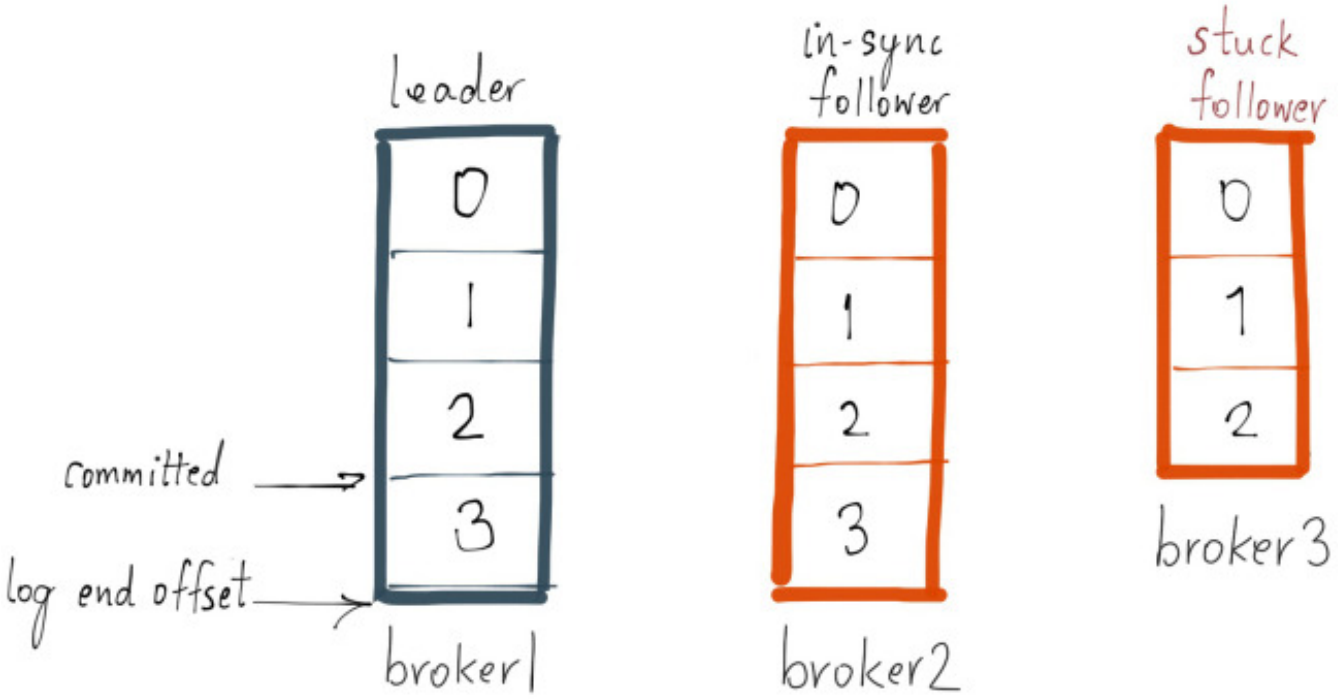
些细微差别，本博客并不打算对该主题进行详尽的讨论。感兴趣的同学可以到这里详细了解 Kafka 复制的工作原理。

副本在什么情况下才算跟上 leader

一个副本如果它没有跟上 leader 的日志进度，那么它可能会被标记为不同步的副本。我通过一个例子来解释跟上（caught up）的含义。假设我们有一个名为 foo 的主题，并且只有一个分区，同时复制因子为 3。假设此分区的副本分别在 brokers 1，2和3上，并且我们已经在主题 foo 上提交了3条消息。brokers 1上的副本是 leader，副本2和3是 followers，所有副本都是 ISR 的一部分。假设 replica.lag.max.messages 设置为4，这意味着只要 follower 落后 leader 的消息不超过3条，它就不会从 ISR 中删除。我们把 replica.lag.time.max.ms 设置为500毫秒，这意味着只要 follower 每隔500毫秒或更早地向 leader 发送一个 fetch 请求，它们就不会被标记为死亡并且不会从 ISR 中删除。

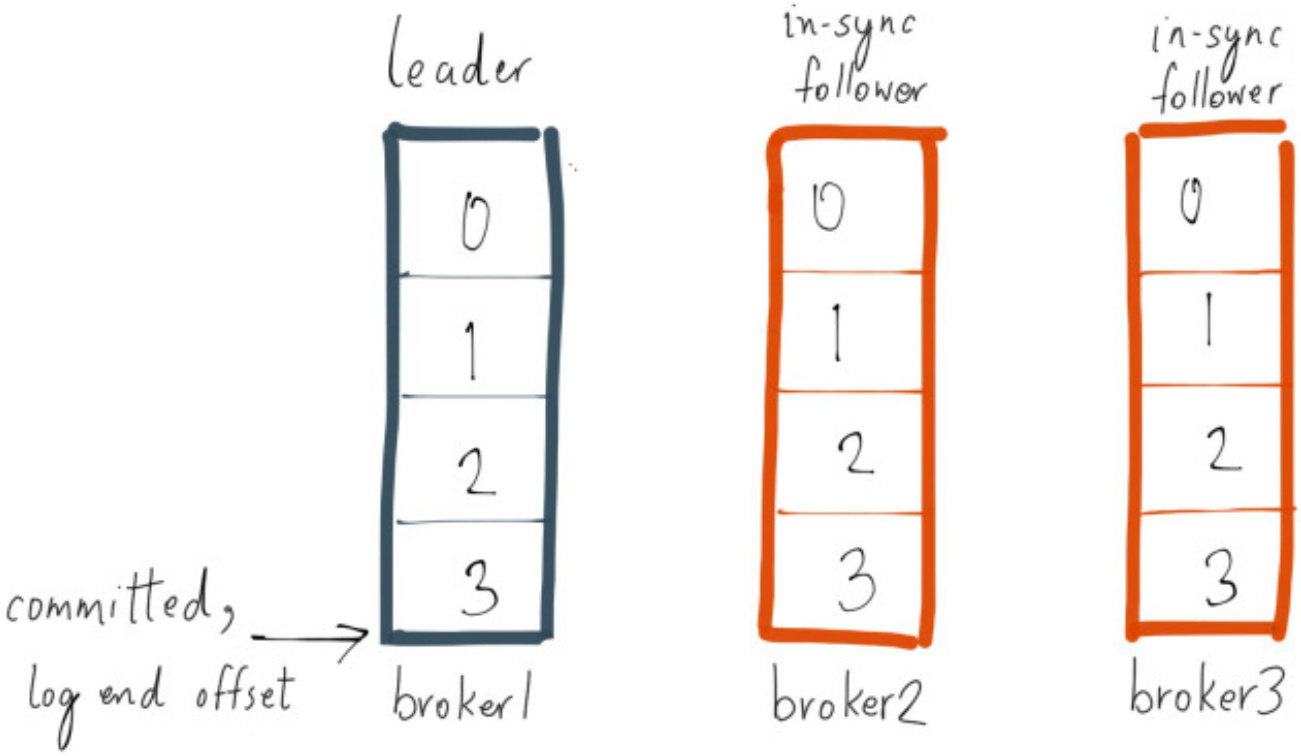


现在假设 producer 往 leader 上发送下一条消息，与此同时，broker 3 上发生了 GC 停顿，现在每个 broker 上的分区情况如下所示：



如果想及时了解Spark、Hadoop或者Hbase相关的文章，欢迎关注微信公众号：iteblog_hadoop

由于 broker 3 在 ISR中，因此在将 broker 3从 ISR 中移除或 broker 3 上的分区跟上 leader 的日志结束偏移之前，最新消息都是不认为被提交的。注意，由于 border 3 落后 leader 的消息比 replica.lag.max.messages = 4 要小，因此不符合从 ISR 中删除的条件。这意味着 broker 3 上的分区需要从 leader 上同步 offset 为 3
的消息，如果它做到了，那么这个副本就是跟上 leader 的。假设 broker 3 在 100ms 内 GC 完成了，并且跟上了 leader 的日志结束偏移，那么最新的情况如下图：



如果想及时了解Spark、Hadoop或者Hbase相关的文章，欢迎关注微信公众号：iteblog_hadoop

什么情况下会导致一个副本与 leader 失去同步

一个副本与 leader 失去同步的原因有很多，主要包括：

- 慢副本（Slow replica）：follower replica 在一段时间内一直无法赶上 leader 的写进度。造成这种情况的最常见原因之一是 follower replica 上的 I/O瓶颈，导致它持久化日志的时间比它从 leader 消费消息的时间要长；
- 卡住副本（Stuck replica）：follower replica 在很长一段时间内停止从 leader 获取消息。这可能是以为 GC 停顿，或者副本出现故障；
- 刚启动副本（Bootstrapping replica）：当用户给某个主题增加副本因子时，新的 follower replicas 是不同步的，直到它跟上 leader 的日志。

当副本落后于 leader 分区时，这个副本被认为是不同步或滞后的。在 Kafka 0.8.2 中，副本的滞后于 leader 是根据 replica.lag.max.messages 或 replica.lag.time.max.ms 来衡量的； 前者用于检测慢副本（Slow replica），而后者用于检测卡住副本（Stuck replica）。

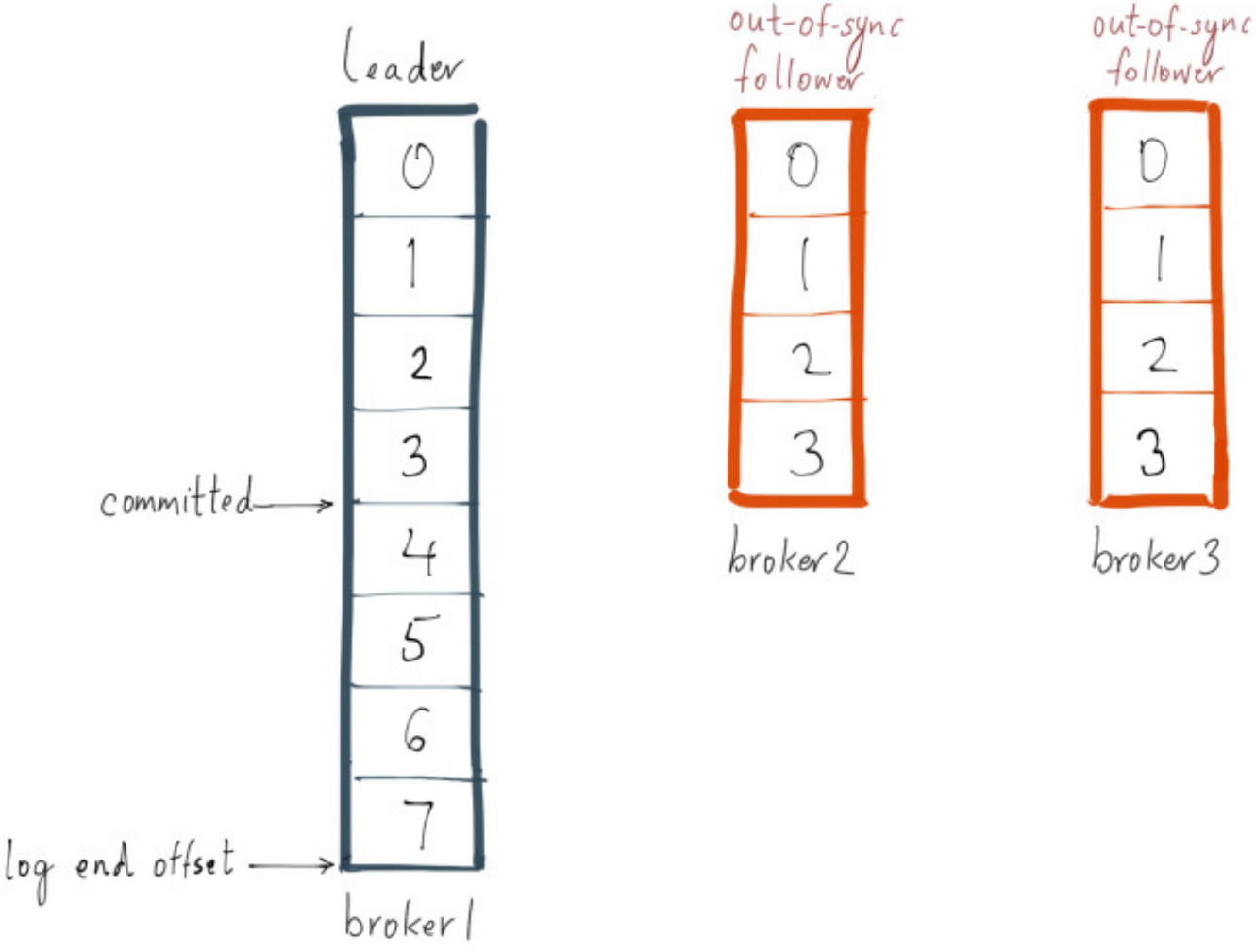
如何确认某个副本处于滞后状态

通过 replica.lag.time.max.ms 来检测卡住副本（Stuck replica）在所有情况下都能很好地工作。它跟踪 follower 副本没有向 leader 发送获取请求的时间，通过这个可以推断 follower 是否正常。另一方面，使用消息数量检测不同步慢副本（Slow replica）的模型只有在为单个主题

或具有同类流量模式的多个主题设置这些参数时才能很好地工作，但我们发现它不能扩展到生产集群中所有主题。

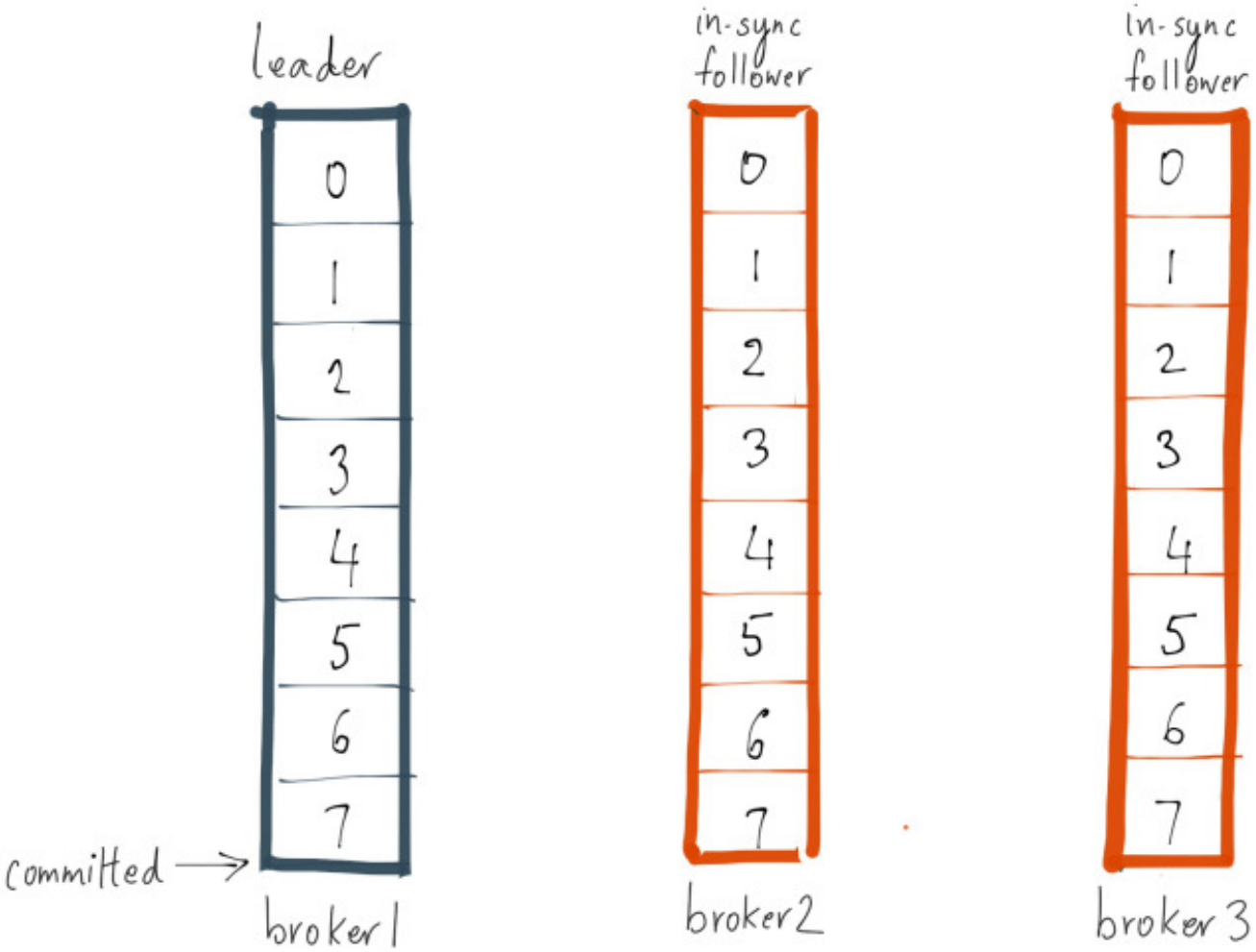
在我之前的示例的基础上，如果主题 foo 以 2 msg/sec 的速率写入数据，其中 leader 收到的单个批次通常永远不会超过3条消息，那么我们知道这个主题的 replica.lag.max.messages 参数可以设置为 4。为什么？ 因为我们以最大速度往 leader 写数据并且在 follower 副本复制这些消息之前，follower 的日志落后于 leader 不超过3条消息。同时，如果主题 foo 的 follower 副本始终落后于 leader 超过3条消息，则我们希望 leader 删除慢速 follower 副本以防止消息写入延迟增加。

这本质上是 replica.lag.max.messages 的目标 - 能够检测始终与 leader 不同步的副本。假设现在这个主题的流量由于峰值而增加，生产者最终往 foo 发送了一批包含4条消息，等于 replica.lag.max.messages = 4 的配置值。此时，两个 follower 副本将被视为与 leader 不同步，并被移除 ISR。



但是，由于两个 follower 副本都处于活动状态，因此它们将在下一个 fetch 请求中赶上 leader 的日志结束偏移量并被添加回 ISR。如果生产者继续向 leader

发送大量的消息，则将重复上述相同的过程。这证明了 follower 副本进出 ISR 时触发不必要的错误警报的情况。



replica.lag.max.messages 参数的核心问题是，用户必须猜测如何配置这个值，因为我们不知道 Kafka 的传入流量到底会到多少，特别是在网络峰值的情况下。

一个参数搞定一切

我们意识到，检测卡住或慢速副本真正重要的事情，是副本与 leader 不同步的时间。我们删除了通过猜测来设置的 replica.lag.max.messages 参数。现在，我们只需要在服务器上配置 replica.lag.time.max.ms 参数即可；这个参数的含义为副本与 leader 不同步的时间。

- 检测卡住副本（Stuck replica）的方式与以前相同 - 如果副本未能在 replica.lag.time.max.ms 时间内发送 fetch 请求，则会将其视为已死的副本并从 ISR 中删除；
- 检测慢副本的机制已经改变 - 如果副本落后于 leader 的时间超过

replica.lag.time.max.ms，则认为它太慢并且从 ISR 中删除。

因此，即使在峰值流量下，生产者往 leader 发送大量的消息，除非副本始终和 leader 保持 replica.lag.time.max.ms 时间的落后，否则它不会随机进出 ISR。

本文翻译自：Hands-free Kafka Replication: A lesson in operational simplicity