如何为Kafka集群选择合适的Topics/Partitions数量？

这是许多kafka使用者经常会问到的一个问题。本文的目的是介绍与本问题相关的一些重要决策因素，并提供一些简单的计算公式。

越多的分区可以提供更高的吞吐量

首先我们需要明白以下事实：在kafka中，单个patition是kafka并行操作的最小单元。在producer和broker端，向每一个分区写入数据是可以完全并行化的，此时，可以通过加大硬件资源的利用率来提升系统的吞吐量，例如对数据进行压缩。在consumer段，kafka只允许单个partition的数据被一个consumer线程消费。因此，在consumer端，每一个Consumer Group内部的consumer并行度完全依赖于被消费的分区数量。综上所述，通常情况下，在一个Kafka集群中，partition的数量越多，意味着可以到达的吞吐量越大。

我们可以粗略地通过吞吐量来计算kafka集群的分区数量。假设对于单个partition，producer端的可达吞吐量为p，Consumer端的可达吞吐量为c，期望的目标吞吐量为t，那么集群所需要的partition数量至少为max(t/p,t/c)。在producer端，单个分区的吞吐量大小会受到批量大小、数据压缩方法、 确认类型（同步/异步）、复制因子等配置参数的影响。经过测试，在producer端，单个partition的吞吐量通常是在10MB/s左右。在consumer端，单个partition的吞吐量依赖于consumer端每个消息的应用逻辑处理速度。因此，我们需要对consumer端的吞吐量进行测量。

虽然随着时间的推移，我们能够对分区的数量进行添加，但是对于基于Key来生成的这一类消息需要我们重点关注。当producer向kafka写入基于key的消息时，kafka通过key的hash值来确定消息需要写入哪个具体的分区。通过这样的方案，kafka能够确保相同key值的数据可以写入同一个partition。kafka的这一能力对于一部分应用是极为重要的，例如对于同一个key的所有消息，consumer需要按消息的顺序进行有序消费。如果partition的数量发生改变，那么上面的有序性保证将不复存在。为了避免上述情况发生，通常的解决办法是多分配一些分区，以满足未来的需求。通常情况下，我们需要根据未来1到2年的目标吞吐量来设计kafka的分区数量。

一开始，我们可以基于当前的业务吞吐量为kafka集群分配较小的broker数量，随着时间的推移，我们可以向集群中增加更多的broker，然后在线方式将适当比例的partition转移到新增加的broker中去。通过这样的方法，我们可以在满足各种应用场景（包括基于key消息的场景）的情况下，保持业务吞吐量的扩展性。

在设计分区数时，除了吞吐量，还有一些其他因素值得考虑。正如我们后面即将看到的，对于一些应用场景，集群拥有过的分区将会带来负面的影响。

越多的分区需要打开更多地文件句柄

在kafka的broker中，每个分区都会对照着文件系统的一个目录。在kafka的数据日志文件目录中，每个日志数据段都会分配两个文件，一个索引文件和一个数据文件。当前版本的kafka，每个broker会为每个日志段文件打开一个index文件句柄和一个数据文件句柄。因此，随着partition的增多，需要底层操作系统配置更高的文件句柄数量限制。这更多的是一个配置问题。我们曾经

见到过，在生产环境Kafka集群中，每个broker打开的文件句柄数量超过30,000。

更多地分区会导致更高的不可用性

Kafka通过多副本复制技术，实现kafka集群的高可用和稳定性。每个partition都会有多个数据副本，每个副本分别存在于不同的broker。所有的数据副本中，有一个数据副本为Leader，其他的数据副本为follower。在kafka集群内部，所有的数据副本皆采用自动化的方式进行管理，并且确保所有的数据副本的数据皆保持同步状态。不论是producer端还是consumer端发往partition的请求，皆通过leader数据副本所在的broker进行处理。当broker发生故障时，对于leader数据副本在该broker的所有partition将会变得暂时不可用。Kafka将会自动在其他数据副本中选择出一个leader，用于接收客户端的请求。这个过程由kafka controller节点broker自动完成，主要是从Zookeeper读取和修改受影响partition的一些元数据信息。在当前的kafka版本实现中，对于zookeeper的所有操作都是由kafka controller来完成的（serially的方式）。

在通常情况下，当一个broker有计划地停止服务时，那么controller会在服务停止之前，将该broker上的所有leader一个个地移走。由于单个leader的移动时间大约只需要花费几毫秒，因此从客户层面看，有计划的服务停机只会导致系统在很小时间窗口中不可用。（注：在有计划地停机时，系统每一个时间窗口只会转移一个leader，其他leader皆处于可用状态。）

然而，当broker非计划地停止服务时（例如，kill -9方式)，系统的不可用时间窗口将会与受影响的partition数量有关。假如，一个2节点的kafka集群中存在2000个partition，每个partition拥有2个数据副本。当其中一个broker非计划地宕机，所有1000个partition同时变得不可用。假设每一个partition恢复时间是5ms，那么1000个partition的恢复时间将会花费5秒钟。因此，在这种情况下，用户将会观察到系统存在5秒钟的不可用时间窗口。

更不幸的情况发生在宕机的broker恰好是controller节点时。在这种情况下，新leader节点的选举过程在controller节点恢复到新的broker之前不会启动。Controller节点的错误恢复将会自动地进行，但是新的controller节点需要从zookeeper中读取每一个partition的元数据信息用于初始化数据。例如，假设一个kafka集群存在10,000个partition，从zookeeper中恢复元数据时每个partition大约花费2ms，则controller的恢复将会增加约20秒的不可用时间窗口。

通常情况下，非计划的宕机事件发生的情况是很少的。如果系统可用性无法容忍这些少数情况的场景，我们最好是将每个broker的partition数量限制在2,000到4,000，每个kafka集群中partition的数量限制在10,000以内。

越多的分区可能增加端对端的延迟

Kafka端对端延迟定义为producer端发布消息到consumer端接收消息所需要的时间。即consumer接收消息的时间减去producer发布消息的时间。Kafka只有在消息提交之后，才会将消息暴露给消费者。例如，消息在所有in-sync副本列表同步复制完成之后才暴露。因此，in-sync副本复制所花时间将是kafka端对端延迟的最主要部分。在默认情况下，每个broker从其他broker节点进行数据副本复制时，该broker节点只会为此工作分配一个线程，该线程需要完成该broker所有partition数据的复制。经验显示，将1000个partition从一个broker到另一个broker所带来的时间延迟约为20ms，这意味着端对端的延迟至少是20ms。这样的延迟对于一些实时应用需求来说显得过长。

注意，上述问题可以通过增大kafka集群来进行缓解。例如，将1000个分区leader放到一个broker节点和放到10个broker节点，他们之间的延迟是存在差异的。在10个broker节点的集群中，每个broker节点平均需要处理100个分区的数据复制。此时，端对端的延迟将会从原来的数十毫秒变为仅仅需要几毫秒。

根据经验，如果你十分关心消息延迟问题，限制每个broker节点的partition数量是一个很好的主意：对于b各broker节点和复制因子为r的kafka集群，整个kafka集群的partition数量最好不超过100*b*r个，即单个partition的leader数量不超过100.

越多的partition意味着需要客户端需要更多的内存

在最新发布的0.8.2版本的kafka中，我们开发了一个更加高效的Java producer。新版producer拥有一个比较好的特征，他允许用户为待接入消息存储空间设置内存大小上限。在内部实现层面，producer按照每一个partition来缓存消息。在数据积累到一定大小或者足够的时间时，积累的消息将会从缓存中移除并发往broker节点。

如果partition的数量增加，消息将会在producer端按更多的partition进行积累。众多的partition所消耗的内存汇集起来，有可能会超过设置的内容大小限制。当这种情况发生时，producer必须通过消息堵塞或者丢失一些新消息的方式解决上述问题，但是这两种做法都不理想。为了避免这种情况发生，我们必须重新将produder的内存设置得更大一些。

根据经验，为了达到较好的吞吐量，我们必须在producer端为每个分区分配至少几十KB的内存，并且在分区数量显著增加时调整可以使用的内存数量。

类似的事情对于consumer端依然有效。Consumer端每次从kafka按每个分区取出一批消息进行消费。消费的分区数越多，需要的内存数量越大。尽管如此，上述方式主要运用于非实时的应用场景。

总结

通常情况下，kafka集群中越多的partition会带来越高的吞吐量。但是，我们必须意识到集群的partition总量过大或者单个broker节点partition过多，都会对系统的可用性和消息延迟带来潜在的影响。未来，我们计划对这些限制进行一些改进，让kafka在分区数量方面变得更加可扩展。

英文原文：http://www.confluent.io/blog/how-to-choose-the-number-of-topicspartitions-in-a-kafka-cluster/