分区数的上限

一味地增加分区数并不能使吞吐量一直得到提升，并且分区数也并不能一直增加，如果超过默认的配置值，还会引起 Kafka 进程的崩溃。读者可以试着在一台普通的 Linux 机器上创建包含10000个分区的主题，比如在下面示例中创建一个主题 topic-bomb：

```
[root@node1 kafka_2.11-2.0.0]# bin/kafka-
topics.sh --zookeeper localhost:2181/ kafka --
create --topic topic-bomb --replication-factor 1
--partitions 10000
Created topic "topic-bomb".
```

执行完成后可以检查 Kafka 的进程是否还存在（比如通过 jps 命令或 ps -aux|grep kafka 命令）。一般情况下，会发现原本运行完好的 Kafka 服务已经崩溃。此时或许会想到，创建这么多分区，是不是因为内存不够而引起的进程崩溃？我们在启动 Kafka 进程的时候将 JVM 堆设置得大一点是不是就可以解决问题了。其实不然，创建这些分区而引起的内存增长完全不足以让 Kafka“畏惧”。

为了分析真实的原因，我们可以打开 Kafka 的服务日志文件（$KAFKA_HOME/logs/ server.log）来一探究竟，会发现服务日志中出现大量的异常：

```
[2018-09-13 00:36:40,019] ERROR Error while
creating log for topic-bomb-xxx in dir
/tmp/kafka-logs
(kafka.server.LogDirFailureChannel)
java.io.IOException: Too many open files
     at
java.io.UnixFileSystem.createFileExclusively(Nati
ve Method)
     at
java.io.File.createNewFile(File.java:1012)
     at kafka.log.AbstractIndex.<init>
(AbstractIndex.scala:54)
     at kafka.log.OffsetIndex.<init>
(OffsetIndex.scala:53)
     at
kafka.log.LogSegment$.open(LogSegment.scala:634)
     at kafka.log.Log.loadSegments(Log.scala:503)
     at kafka.log.Log.<init>(Log.scala:237)
```

异常中最关键的信息是“Too many open flies”，这是一种常见的 Linux 系统错误，通常意味着文件描述符不足，它一般发生在创建线程、创建 Socket、打开文件这些场景下。在 Linux 系统的默认设置下，这个文件描述符的个数不是很多，通过 ulimit 命令可以查看：

```
[root@node1 kafka_2.11-2.0.0]# ulimit -n
1024
[root@node1 kafka_2.11-2.0.0]# ulimit -Sn
1024
[root@node1 kafka_2.11-2.0.0]# ulimit -Hn
4096
```

ulimit 是在系统允许的情况下，提供对特定 shell 可利用的资源的控制。-H 和 -S 选项指定资源的硬限制和软限制。硬限制设定之后不能再添加，而软限制则可以增加到硬限制规定的值。如果 -H 和 -S 选项都没有指定，则软限制和硬限制同时设定。限制值可以是指定资源的数值或 hard、soft、unlimited 这些特殊值，其中 hard 代表当前硬限制，soft 代表当前软件限制，unlimited 代表不限制。如果不指定限制值，则打印指定资源的软限制值，除非指定了 -H 选项。硬限制可以在任何时候、任何进程中设置，但硬限制只能由超级用户设置。软限制是内核实际执行的限制，任何进程都可以将软限制设置为任意小于等于硬限制的值。

我们可以通过测试来验证本案例中的 Kafka 的崩溃是否是由于文件描述符的限制而引起的。下面我们在一个包含3个节点的 Kafka 集群中挑选一个节点进行具体的分析。首先通过 jps 命令查看 Kafka 进程 pid 的值：

```
[root@node1 kafka_2.11-2.0.0]# jps -l
31796 kafka.Kafka
```

查看当前 Kafka 进程所占用的文件描述符的个数（注意这个值并不是Kafka第一次启动时就需要占用的文件描述符的个数，示例中的 Kafka 环境下已经存在了若干主题）：

```
[root@node1 kafka_2.11-2.0.0]# ls /proc/31796/fd
| wc -l
194
```

我们再新建一个只有一个分区的主题，并查看 Kafka 进程所占用的文件描述符的个数：

```
[root@node1 kafka_2.11-2.0.0]# bin/kafka-
topics.sh --zookeeper localhost:2181/ kafka --
create --topic topic-bomb-1 --replication-factor
1 --partitions 1
Created topic "topic-bomb-1".

[root@node1 kafka_2.11-2.0.0]# ls /proc/31796/fd
| wc -l
195
```

可以看到增加了一个分区，对应的也只增加了一个文件描述符。之前我们通过 ulimit命令可以看到软限制是1024，我们创建一个具有829（1024-195=829）个分区的主题：

```
[root@node1 kafka_2.11-2.0.0]# bin/kafka-
topics.sh --zookeeper localhost:2181/ kafka --
create --topic topic-bomb-2 --replication-factor
1 --partitions 829
Created topic "topic-bomb-2".

[root@node1 kafka_2.11-2.0.0]# ls /proc/31796/fd
| wc -l
1024
```

可以看到 Kafka 进程此时占用了1024个文件描述符，并且运行完好。这时我们还可以联想到硬限制4096这个关键数字，我们再创建一个包含3071（4096-1024=3072，这里特地少创建1个分区）个分区的主题，示例如下：

```
[root@node1 kafka_2.11-2.0.0]# bin/kafka-
topics.sh --zookeeper localhost:2181/ kafka --
create --topic topic-bomb-3 --replication-factor
1 --partitions 3071
Created topic "topic-bomb-3".

[root@node1 kafka_2.11-2.0.0]# ls /proc/31796/fd
| wc -l
4095
```

Kafka 进程依旧完好，文件描述符占用为4095，逼近最高值4096。最后我们再次创建一个只有一个分区的主题：

```
[root@node1 kafka_2.11-2.0.0]# bin/kafka-
topics.sh --zookeeper localhost:2181/ kafka --
create --topic topic-bomb-4 --replication-factor
1 --partitions 1
Created topic "topic-bomb-4".

[root@node1 kafka_2.11-2.0.0]# ls /proc/31796/fd
| wc -l
ls: cannot access /proc/31796/fd: No such file or
directory
0
```

此时 Kafka 已经崩溃，查看进程号时已没有相关信息。查看 Kafka 中的日志，还会发现报出前面提及的异常“java.io.IOException: Too many open files”，表明已达到上限。

如何避免这种异常情况？对于一个高并发、高性能的应用来说，1024或4096的文件描述符限制未免太少，可以适当调大这个参数。比如使用 ulimit -n 65535命令将上限提高到65535，这样足以应对大多数的应用情况，再高也完全没有必要了。

```
[root@node1 kafka_2.11-2.0.0]# ulimit -n 65535
#可以再次查看相应的软硬限制数
[root@node1 kafka_2.11-2.0.0]# ulimit -Hn
65535
[root@node1 kafka_2.11-2.0.0]# ulimit -Sn
65535
```

也可以在/etc/security/limits.conf 文件中设置，参考如下：

```
#nofile - max number of open file descriptors
root soft nofile 65535
root hard nofile 65535
```

limits.conf 文件修改之后需要重启才能生效。limits.conf 文件与 ulimit 命令的区别在于前者是针对所有用户的，而且在任何 shell 中都是生效的，即与 shell 无关，而后者只是针对特定用户的当前 shell 的设定。在修改最大文件打开数时，最好使用 limits.conf 文件来修改，通过这个文件，可以定义用户、资源类型、软硬限制等。也可以通过在/etc/profile 文件中添加 ulimit 的设置语句来使全局生效。

设置之后可以再次尝试创建10000个分区的主题，检查一下 Kafka 是否还会再次崩溃。

考量因素

如何选择合适的分区数？一个“恰如其分”的答案就是视具体情况而定。

从吞吐量方面考虑，增加合适的分区数可以在一定程度上提升整体吞吐量，但超过对应的阈值之后吞吐量不升反降。如果应用对吞吐量有一定程度上的要求，则建议在投入生产环境之前对同款硬件资源做一个完备的吞吐量相关的测试，以找到合适的分区数阈值区间。

在创建主题之后，虽然我们还能够增加分区的个数，但基于 key 计算的主题需要严谨对待。当生产者向 Kafka 中写入基于 key 的消息时，Kafka 通过消息的 key 来计算出消息将要写入哪个具体的分区，这样具有相同 key 的数据可以写入同一个分区。Kafka 的这一功能对于一部分应用是极为重要的，比如日志压缩（Log Compaction），详细可以参考[《图解Kafka之核心原理》(https://juejin.im/book/5c7d270ff265da2d89634e9e)](https://juejin.im/book/5c7d270ff265da2d89634e9e)；再比如对于同一个 key 的所有消息，消费者需要按消息的顺序进行有序的消费，如果分区的数量发生变化，那么有序性就得不到保证。

在创建主题时，最好能确定好分区数，这样也可以省去后期增加分区所带来的多余操作。尤其对于与 key 高关联的应用，在创建主题时可以适当地多创建一些分区，以满足未来的需求。通常情况下，可以根据未来2年内的目标吞吐量来设定分区数。当然如果应用与 key 弱关联，并且具备便捷的增加分区数的操作接口，那么也可以不用考虑那么长远的目标。

有些应用场景会要求主题中的消息都能保证顺序性，这种情况下在创建主题时可以设定分区数为1，通过分区有序性的这一特性来达到主题有序性的目的。

当然分区数也不能一味地增加，参考前面的内容，分区数会占用文件描述符，而一个进程所能支配的文件描述符是有限的，这也是通常所说的文件句柄的开销。虽然我们可以通过修改配置来增加可用文件描述符的个数，但凡事总有一个上限，在选择合适的分区数之前，最好再考量一下当前 Kafka 进程中已经使用的文件描述符的个数。

分区数的多少还会影响系统的可用性。在前面章节中，我们了解到 Kafka 通过多副本机制来实现集群的高可用和高可靠，每个分区都会有一至多个副本，每个副本分别存在于不同的 broker 节点上，并且只有 leader 副本对外提供服务。在 Kafka 集群的内部，所有的副本都采用自动化的方式进行管理，并确保所有副本中的数据都能保持一定程度上的同步。当 broker 发生故障时，leader 副本所属宿主的 broker 节点上的所有分区将暂时处于不可用的状态，此时 Kafka 会

自动在其他的 follower 副本中选举出新的 leader 用于接收外部客户端的请求，整个过程由 Kafka 控制器负责完成。分区在进行 leader 角色切换的过程中会变得不可用，不过对于单个分区来说这个过程非常短暂，对用户而言可以忽略不计。如果集群中的某个 broker 节点宕机，那么就会有大量的分区需要同时进行 leader 角色切换，这个切换的过程会耗费一笔可观的时间，并且在这个时间窗口内这些分区也会变得不可用。

分区数越多也会让 Kafka 的正常启动和关闭的耗时变得越长，与此同时，主题的分区数越多不仅会增加日志清理的耗时，而且在被删除时也会耗费更多的时间。对旧版的生产者和消费者客户端而言，分区数越多，也会增加它们的开销，不过这一点在新版的生产者和消费者客户端中有效地得到了抑制。

如何选择合适的分区数？从某种意思来说，考验的是决策者的实战经验，更透彻地说，是对 Kafka 本身、业务应用、硬件资源、环境配置等多方面的考量而做出的选择。在设定完分区数，或者更确切地说是创建主题之后，还要对其追踪、监控、调优以求更好地利用它。读者看到本节的内容之前或许没有对分区数有太大的困扰，而看完本节的内容之后反而困惑了起来，其实大可不必太过惊慌，一般情况下，根据预估的吞吐量及是否与 key 相关的规则来设定分区数即可，后期可以通过增加分区数、增加 broker 或分区重分配等手段来进行改进。如果一定要给一个准则，则建议将分区数设定为集群中 broker 的倍数，即假定集群中有3个 broker 节点，可以设定分区数为3、6、9等，至于倍数的选定可以参考预估的吞吐量。不过，如果集群中的 broker 节点数有很多，比如大几十或上百、上千，那么这种准则也不太适用，在选定分区数时进一步可以引入基架等参考因素。

总结

从16节开始到这里笔者主要讲述了 Kafka 概念中的两大核心—主题和分区。通过对主题的增删查改、配置管理等内容来了解主题相关的知识点。通过对分区副本的一系列操作及分区数设定的考量因素来理解分区相关的概念，比如优先副本、限流、分区重分配等。还介绍了KafkaAdminClient、kafka-topics.sh、kafka-configs.sh、kafka-perferred-replica-election.sh、kafka-reassign-partitions.sh、kafka-producer-perf-test.sh和kafka-consumer-perf-test.sh等脚本的具体使用，读者可以通过实地操作来加深对这些内容的理解。