Kafka Producer是如何动态感知Topic分区数变化

我们都知道，使用Kafka Producer往Kafka的Broker发送消息的时候，Kafka会根据消息的key计算出这条消息应该发送到哪个分区。默认的分区计算类是HashPartitioner，其实现如下：

```
class HashPartitioner(props: VerifiableProperties = null) extends Partitioner {
  def partition(data: Any, numPartitions: Int): Int = {
    (data.hashCode % numPartitions)
  }
}
```

其中numPartitions就是Tpoic的分区总数。partition函数会在kafka的getPartition函数中被调用，计算消息的分区ID。而numPartitions的数量是通过

```
val topicPartitionsList = getPartitionListForTopic(message)
val numPartitions = topicPartitionList.size
```

计算得到的，getPartitionListForTopic实现如下：

```
private def getPartitionListForTopic(m: KeyedMessage[K,Message]):
                                      Seq[PartitionAndLeader] = {
  val topicPartitionsList =
brokerPartitionInfo.getBrokerPartitionInfo(m.topic, correlationId.getAndIncrement)
  debug("Broker partitions registered for topic: %s are %s"
    .format(m.topic, topicPartitionsList.map(p => p.partitionId).mkString(",")))
  val totalNumPartitions = topicPartitionsList.length
  if(totalNumPartitions == 0)
    throw new NoBrokersForPartitionException("Partition key = " + m.key)
  topicPartitionsList
}
```

那么问题是，如果在Kafka Producer往Kafka的Broker发送消息的时候用户通过命令修改了改主题的分区数，Kafka Producer能动态感知吗？答案是可以的。那是立刻就感知吗？不是，是过一定的时间(topic.metadata.refr

esh.interval.ms参数决定)才知道分区数改变的，我们来看看代码实现。

上面代码中的topicPartitionsList是通过getBrokerPartitionInfo函数获取的，其实现如下：

```
def getBrokerPartitionInfo(topic: String, correlationId: Int): Seq[PartitionAndLeader] = {
  debug("Getting broker partition info for topic %s".format(topic))
  // check if the cache has metadata for this topic
  val topicMetadata = topicPartitionInfo.get(topic)
  val metadata: TopicMetadata =
    topicMetadata match {
      case Some(m) => m
      case None =>
        // refresh the topic metadata cache
        updateInfo(Set(topic), correlationId)
        val topicMetadata = topicPartitionInfo.get(topic)
        topicMetadata match {
          case Some(m) => m
          case None =>
throw new KafkaException("Failed to fetch topic metadata for topic: " + topic)
        }
    }
  val partitionMetadata = metadata.partitionsMetadata
  if(partitionMetadata.size == 0) {
    if(metadata.errorCode != ErrorMapping.NoError) {
      throw
        new KafkaException(ErrorMapping.exceptionFor(metadata.errorCode))
    } else {
      throw new KafkaException("Topic metadata %s has empty
                    partition metadata and no error code".format(metadata))
    }
  }
  partitionMetadata.map { m =>
    m.leader match {
      case Some(leader) =>
        debug("Partition [%s,%d] has leader %d".format(topic, m.partitionId, leader.id))
        new PartitionAndLeader(topic, m.partitionId, Some(leader.id))
      case None =>
        debug("Partition [%s,%d] does not have a leader yet".format(topic, m.partitionId))
        new PartitionAndLeader(topic, m.partitionId, None)
    }
  }.sortWith((s, t) => s.partitionId < t.partitionId)
}
```

topicPartitionInfo的数据类型是HashMap[String, TopicMetadata]，程序先通过topic名从topicPartitionInfo中查找是否有这个topic的元数据，如果有则直接返回；如果没有呢？则调用updateInfo(Set(topic), correlationId)函数获取该topic的元数据：

```
def updateInfo(topics: Set[String], correlationId: Int) {
  var topicsMetadata: Seq[TopicMetadata] = Nil
  val topicMetadataResponse =
        ClientUtils.fetchTopicMetadata(topics, brokers, producerConfig, correlationId)
  topicsMetadata = topicMetadataResponse.topicsMetadata
  // throw partition specific exception
  topicsMetadata.foreach(tmd =>{
    trace("Metadata for topic %s is %s".format(tmd.topic, tmd))
    if(tmd.errorCode == ErrorMapping.NoError) {
      topicPartitionInfo.put(tmd.topic, tmd)
    } else
      warn("Error while fetching metadata [%s] for topic [%s]: %s
        ".format(tmd, tmd.topic, ErrorMapping.exceptionFor(tmd.errorCode).getClass))
    tmd.partitionsMetadata.foreach(pmd =>{
      if (pmd.errorCode != ErrorMapping.NoError
        && pmd.errorCode == ErrorMapping.LeaderNotAvailableCode) {
        warn("Error while fetching metadata %s for topic partition
              [%s,%d]: [%s]".format(pmd, tmd.topic, pmd.partitionId,
          ErrorMapping.exceptionFor(pmd.errorCode).getClass))
      } // any other error code (e.g. ReplicaNotAvailable)
        //can be ignored since the producer does not need to
        //access the replica and isr metadata
    })
  })
  producerPool.updateProducer(topicsMetadata)
}
```

上面程序通过调用ClientUtils.fetchTopicMetadata函数获取topicsMetadata。如果没有遇到错误，那么就将获取到的topicsMetadata存放到topicPartitionInfo中。那么程序是如何感知当前topic的元数据发生变化，比如前面说的分区数增加了？其实主要逻辑在这里：

```
if (topicMetadataRefreshInterval >= 0 &&
        SystemTime.milliseconds - lastTopicMetadataRefreshTime >
            topicMetadataRefreshInterval) {
      Utils.swallowError(brokerPartitionInfo.updateInfo(topicMetadataToRefresh.toSet,
            correlationId.getAndIncrement))

      ......
}
```

这个逻辑在处理消息的时候就会被调用，如果topicMetadataRefreshInterval>=0并且当前时间减去上一次元数据更新的时间间隙大于topicMetadataRefreshInterval，则会再一次更新Tpoic的元数据，而topicMetadataRefreshInterval的值就是通过topic.metadata.refresh.interval.ms配置的。

结论：在启动Kafka Producer往Kafka的Broker发送消息的时候，用户修改了该Topic的分区数，Producer可以在最多topic.metadata.refresh.interval.ms的时间之后感知到，此感知同时适用于async和sync模式，并且可以将数据发送到新添加的分区中。下面附上topic.metadata.refresh.interval.ms参数的解释：

> The producer generally refreshes the topic metadata from brokers when there is a failure (partition missing, leader not available...). It will also poll regularly (default: every 10min so 600000ms). If you set this to a negative value, metadata will only get refreshed on failure. If you set this to zero, the metadata will get refreshed after each message sent (not recommended) Important note: the refresh happen only AFTER the message is sent, so if the producer never sends a message the metadata is never refreshed