Kafka创建Topic时如何将分区放置到不同的Broker中

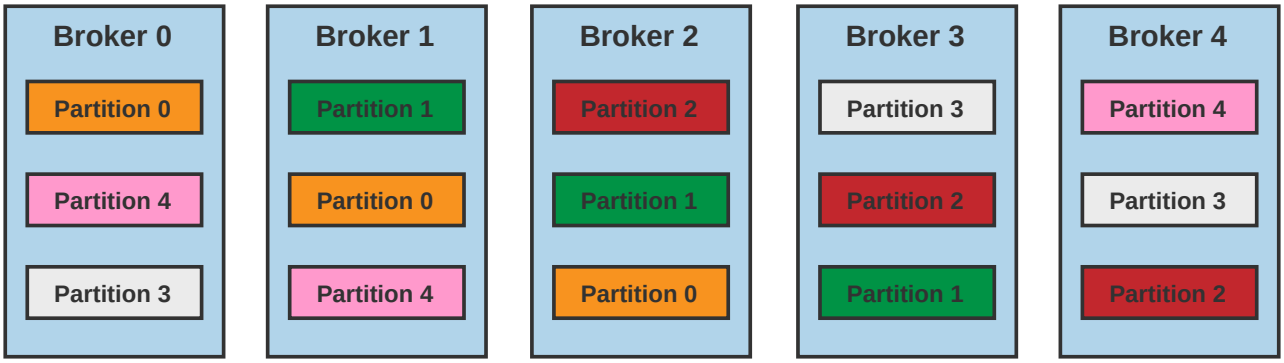
熟悉 Kafka 的同学肯定知道，每个主题有多个分区，每个分区会存在多个副本，本文今天要讨论的是这些副本是怎么样放置在 Kafka 集群的 Broker 中的。

大家可能在网上看过这方面的知识，网上对这方面的知识是千变一律，都是如下说明的：

为了更好的做负载均衡，Kafka尽量将所有的Partition均匀分配到整个集群上。Kafka分配Replica的算法如下：

- 将所有存活的N个Brokers和待分配的Partition排序
- 将第i个Partition分配到第(i mod n)个Broker上，这个Partition的第一个Replica存在于这个分配的Broker上，并且会作为partition的优先副本
- 将第i个Partition的第j个Replica分配到第((i + j) mod n)个Broker上

假设现在有5个
Broker，分区数为5，副本为3的主题，按照上面的说法，主题最终分配在整个集群的样子如下：



???????Spark?Hadoop??Hbase??????????????????iteblog_hadoop

但事实真的是这样的吗？实际上如果真按照这种算法，会存在以下明显几个问题：

- 所有主题的第一个分区都是存放在第一个Broker上，这样会造成第一个Broker上的分区总数多于其他的Broker，这样就失去了负载均衡的目的；
- 如果主题的分区数多于Broker的个数，多于的分区都是倾向于将分区发放置在前几个Broker上，同样导致负载不均衡。

所以其实上面的算法不准确。严格来说，上面的算法只是Kafka分配分区的一种特例（下面介绍算法部分会说明）。下面我们来看看 Kafka 内部到底是如何将分区分配到各个 Broker

中的，其具体算法实现函数就是 assignReplicasToBrokers，如下：

```
//////////////////////////////////////////////////////////////////////
 User: 过往记忆
 Date: 2017年08月08日
 Time: 07:57:32
 bolg: https://www.iteblog.com
 本文地址：https://www.iteblog.com/archives/2219
 过往记忆博客，专注于hadoop、hive、spark、shark、flume的技术博客，大量的干货
 过往记忆博客微信公共帐号：iteblog_hadoop
//////////////////////////////////////////////////////////////////////
def assignReplicasToBrokers(brokerList: Seq[Int],
                            nPartitions: Int,
                            replicationFactor: Int,
                            fixedStartIndex: Int = -1,
                            startPartitionId: Int = -1)
: Map[Int, Seq[Int]] = {
  if (nPartitions <= 0)
    throw new AdminOperationException("number of partitions must be larger than 0")
  if (replicationFactor <= 0)
    throw new AdminOperationException("replication factor must be larger than 0")
  if (replicationFactor > brokerList.size)
    throw new AdminOperationException("replication factor: " + replicationFactor +
      " larger than available brokers: " + brokerList.size)
  val ret = new mutable.HashMap[Int, List[Int]]()
  val startIndex = if (fixedStartIndex >= 0) fixedStartIndex else rand.nextInt(brokerList.size)
  var currentPartitionId = if (startPartitionId >= 0) startPartitionId else 0

  var nextReplicaShift = if (fixedStartIndex >= 0) fixedStartIndex else rand.nextInt(brokerList.siz
e)
  for (i <- 0 until nPartitions) {
    if (currentPartitionId > 0 && (currentPartitionId % brokerList.size == 0))
      nextReplicaShift += 1
    val firstReplicaIndex = (currentPartitionId + startIndex) % brokerList.size
    var replicaList = List(brokerList(firstReplicaIndex))
    for (j <- 0 until replicationFactor - 1)
      replicaList ::= brokerList(replicaIndex(firstReplicaIndex, nextReplicaShift, j, brokerList.size))
    ret.put(currentPartitionId, replicaList.reverse)
    currentPartitionId = currentPartitionId + 1
  }
  ret.toMap
}

private def replicaIndex(firstReplicaIndex: Int, secondReplicaShift: Int, replicaIndex: Int, nBrok
ers: Int): Int = {
  val shift = 1 + (secondReplicaShift + replicaIndex) % (nBrokers - 1)
```

```
  (firstReplicaIndex + shift) % nBrokers
}
```

从上面的算法可以看出：

- 副本因子不能大于 Broker 的个数；
- 第一个分区（编号为0）的第一个副本放置位置是随机从 brokerList 选择的；
- 其他分区的第一个副本放置位置相对于第0个分区依次往后移。也就是如果我们有5个 Broker，5个分区，假设第一个分区放在第四个 Broker 上，那么第二个分区将会放在第五个 Broker 上；第三个分区将会放在第一个 Broker 上；第四个分区将会放在第二个 Broker 上，依次类推；
- 剩余的副本相对于第一个副本放置位置其实是由 nextReplicaShift 决定的，而这个数也是随机产生的；

所以如果我们依次如下调用上面的程序，ret 变量的输出结果会如下：

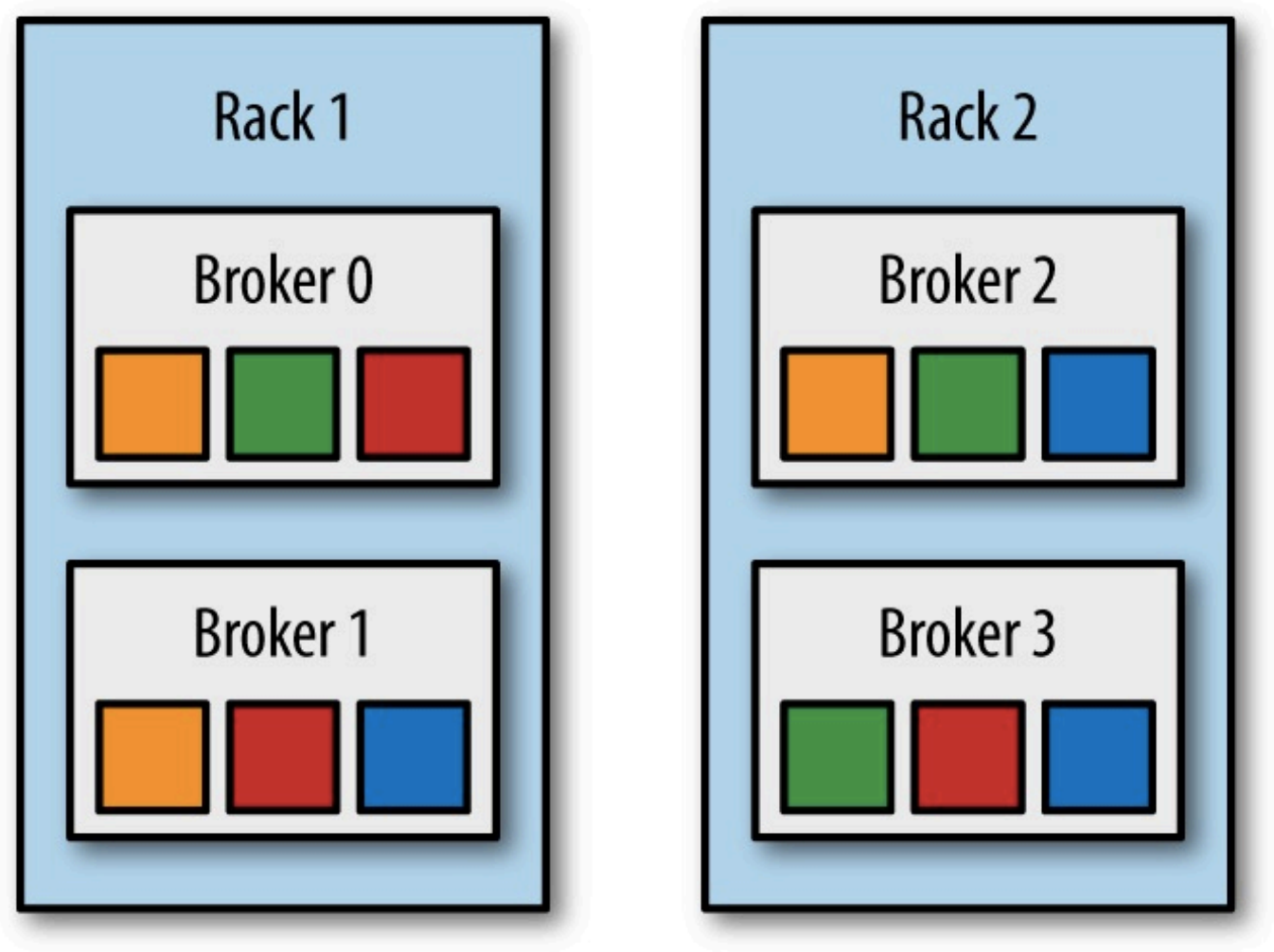
```
assignReplicasToBrokers(Seq(0, 1, 2, 3), 3, 3)
(0,List(3, 2, 0))
(1,List(0, 3, 1))
(2,List(1, 0, 2))

assignReplicasToBrokers(Seq(0, 1, 2, 3), 5, 3)
(0,List(3, 1, 2))
(1,List(0, 2, 3))
(2,List(1, 3, 0))
(3,List(2, 0, 1))
(4,List(3, 2, 0))

assignReplicasToBrokers(Seq(0, 1, 2, 3), 9, 3)
(0,List(1, 0, 2))
(1,List(2, 1, 3))
(2,List(3, 2, 0))
(3,List(0, 3, 1))
(4,List(1, 2, 3))
(5,List(2, 3, 0))
(6,List(3, 0, 1))
(7,List(0, 1, 2))
(8,List(1, 3, 0))
```

注意，你运行上面的程序结果可能和我的不一样，因为上面算法中的 startIndex 和

nextReplicaShift 变量都是随机生成的。其实 Kafka
创建主题就是这么调用算法的（ fixedStartIndex
和 startPartitionId都是使用默认值）。另外，第一个放置的分区副本一般都是
Leader，其余的都是 Follow 副本，也就是说，上面输出的List第一个元素就是 Leader
副本所在的 Broker 编号。

到这里我们应该知道，网上其他博客介绍的 Kafka 分区是如何分配到各个 Broker 上其实是将
startIndex 设置成 0， 同时 fixedStartIndex 设置成
1，这样本文最开头介绍的算法就对了。但其实 Kafka 内部并不是这样调用的，大家注意。

如果我们还考虑机架的话，情况就更复杂了。这里为了简便起见，我们假设 startIndex =
4，fixedStartIndex = 1。现在如果我们有两个机架的 Kafka 集群，brokers 0,
1 和 2 同属于一个机架；brokers 3, 4 和 5 属于另外一个机架。现在我们对这些 Broker
进行排序：0, 3, 1, 4, 2, 5（每个机架依次选择一个Broker进行排序）。按照机架的 Kafka
分区放置算法，如果分区0的第一个副本放置到broker 4上面，那么其第二个副本将会放到broker
2上面，第三个副本将会放到 broker 5上面；同理，分区1的第一个副本放置到broker
2上面，其第二个副本将会放到broker 5上面，第三个副本将会放到 broker
0上面。这就保证了这两个副本放置到不同的机架上面，即使其中一个机架出现了问题，我们的
Kafka
集群还是可以正常运行的。现在把机架因素考虑进去的话，我们的分区看起来像下面一样：



如果想及时了
解Spark、Hadoop或者Hbase相关的文章，欢迎关注微信公共帐号：iteblog_hadoop

从上图可以看出，只要上面其中一个机架没有问题，我们的数据仍然可以对外提供服务。这就大大提高了集群的可用性。