Spark+Kafka的Direct方式将偏移量发送到Zookeeper实现

Apache Spark 1.3.0引入了Direct API，利用Kafka的低层次API从Kafka集群中读取数据，并且在Spark Streaming系统里面维护偏移量相关的信息，并且通过这种方式去实现零数据丢失(zero data loss)相比使用基于Receiver的方法要高效。但是因为是Spark Streaming系统自己维护Kafka的读偏移量，而Spark Streaming系统并没有将这个消费的偏移量发送到Zookeeper中，这将导致那些基于偏移量的Kafka集群监控软件（比如：Apache Kafka监控之Kafka Web Console、Apache Kafka监控之KafkaOffsetMonitor等）失效。本文就是基于为了解决这个问题，使得我们编写的Spark Streaming程序能够在每次接收到数据之后自动地更新Zookeeper中Kafka的偏移量。

我们从Spark的官方文档可以知道，维护Spark内部维护Kafka便宜了信息是存储在HasOffsetRanges类的offsetRanges中，我们可以在Spark Streaming程序里面获取这些信息：

```
val offsetsList = rdd.asInstanceOf[HasOffsetRanges].offsetRanges
```

这样我们就可以获取所以分区消费信息，只需要遍历offsetsList，然后将这些信息发送到Zookeeper即可更新Kafka消费的偏移量。完整的代码片段如下：

```
val messages = KafkaUtils.createDirectStream[String, String, StringDecoder, StringDecoder](ssc, kafkaParams, topicsSet)
    messages.foreachRDD(rdd => {
      val offsetsList = rdd.asInstanceOf[HasOffsetRanges].offsetRanges
      val kc = new KafkaCluster(kafkaParams)
      for (offsets < - offsetsList) {
        val topicAndPartition = TopicAndPartition("iteblog", offsets.partition)
        val o = kc.setConsumerOffsets(args(0), Map((topicAndPartition, offsets.untilOffset)))
        if (o.isLeft) {
          println(s"Error updating the offset to Kafka cluster: ${o.left.get}")
        }
      }
})
```

KafkaCluster类用于建立和Kafka集群的链接相关的操作工具类，我们可以对Kafka中Topic的每个分区设置其相应的偏移量Map((topicAndPartition, offsets.untilOffset)),然后调用KafkaCluster类的setConsumerOffsets方法去更新Zookeeper里面的信息，这样我们就可以更新Kafka的偏移

量，最后我们就可以通过KafkaOffsetMonitor之类软件去监控Kafka中相应Topic的消费信息，下图是KafkaOffsetMonitor的监控情况：

Topic	Partition	Offset	logSize	Lag	Owner	Created	Last Seen
iteblog		390378001	390382382	4381			
	0	43375369	43375851	482		5 minutes ago	a few seconds ago
	1	43375352	43375834	482		5 minutes ago	a few seconds ago
	2	43375324	43375811	487		5 minutes ago	a few seconds ago
	3	43375320	43375806	486		5 minutes ago	a few seconds ago
	4	43375331	43375819	488		5 minutes ago	a few seconds ago
	5	43375314	43375802	488		5 minutes ago	a few seconds ago

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

从图中我们可以看到KafkaOffsetMonitor监控软件已经可以监控到Kafka相关分区的消费情况，这对监控我们整个Spark Streaming程序来非常重要，因为我们可以任意时刻了解Spark读取速度。另外，KafkaCluster工具类的完整代码如下：

```
package org.apache.spark.streaming.kafka

import kafka.api.OffsetCommitRequest
import kafka.common.{ErrorMapping, OffsetMetadataAndError, TopicAndPartition}
import kafka.consumer.SimpleConsumer
import org.apache.spark.SparkException
import org.apache.spark.streaming.kafka.KafkaCluster.SimpleConsumerConfig

import scala.collection.mutable.ArrayBuffer
import scala.util.Random
import scala.util.control.NonFatal

/**
```

```
* User: 过往记忆
* Date: 2015-06-02
* Time: 下午23:46
* bolg: https://www.iteblog.com
* 本文地址：https://www.iteblog.com/archives/1381.html
* 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
*/

class KafkaCluster(val kafkaParams: Map[String, String]) extends Serializable {
  type Err = ArrayBuffer[Throwable]

  @transient private var _config: SimpleConsumerConfig = null

  def config: SimpleConsumerConfig = this.synchronized {
    if (_config == null) {
      _config = SimpleConsumerConfig(kafkaParams)
    }
    _config
  }

  def setConsumerOffsets(groupId: String,
                         offsets: Map[TopicAndPartition, Long]
                         ): Either[Err, Map[TopicAndPartition, Short]] = {
    setConsumerOffsetMetadata(groupId, offsets.map { kv =>
      kv._1 -> OffsetMetadataAndError(kv._2)
    })
  }

  def setConsumerOffsetMetadata(groupId: String,
                                metadata: Map[TopicAndPartition, OffsetMetadataAndError]
                                ): Either[Err, Map[TopicAndPartition, Short]] = {
    var result = Map[TopicAndPartition, Short]()
    val req = OffsetCommitRequest(groupId, metadata)
    val errs = new Err
    val topicAndPartitions = metadata.keySet
    withBrokers(Random.shuffle(config.seedBrokers), errs) { consumer =>
      val resp = consumer.commitOffsets(req)
      val respMap = resp.requestInfo
      val needed = topicAndPartitions.diff(result.keySet)
      needed.foreach { tp: TopicAndPartition =>
        respMap.get(tp).foreach { err: Short =>
          if (err == ErrorMapping.NoError) {
            result += tp -> err
          } else {
            errs.append(ErrorMapping.exceptionFor(err))
```

```
        }
       }
      }
      if (result.keys.size == topicAndPartitions.size) {
       return Right(result)
      }
     }
     val missing = topicAndPartitions.diff(result.keySet)
     errs.append(new SparkException(s"Couldn't set offsets for ${missing}"))
     Left(errs)
   }

   private def withBrokers(brokers: Iterable[(String, Int)], errs: Err)
                          (fn: SimpleConsumer => Any): Unit = {
     brokers.foreach { hp =>
       var consumer: SimpleConsumer = null
       try {
         consumer = connect(hp._1, hp._2)
         fn(consumer)
       } catch {
         case NonFatal(e) =>
           errs.append(e)
       } finally {
         if (consumer != null) {
           consumer.close()
         }
       }
     }
   }

   def connect(host: String, port: Int): SimpleConsumer =
     new SimpleConsumer(host, port, config.socketTimeoutMs,
       config.socketReceiveBufferBytes, config.clientId)
}
```

完整代码工程下载