Spark Streaming和Kafka整合开发指南(一)

Apache Kafka是一个分布式的消息发布-订阅系统。可以说，任何实时大数据处理工具缺少与Kafka整合都是不完整的。本文将介绍如何使用Spark Streaming从Kafka中接收数据，这里将会介绍两种方法：（1）、使用Receivers和Kafka高层次的API；（2）、使用Direct API，这是使用低层次的KafkaAPI，并没有使用到Receivers，是Spark 1.3.0中开始引入的。这两种方法有不同的编程模型，性能特点和语义担保。下文将会一一介绍。

基于Receivers的方法

这个方法使用了Receivers来接收数据。Receivers的实现使用到Kafka高层次的消费者API。对于所有的Receivers，接收到的数据将会保存在Spark executors中，然后由Spark Streaming启动的Job来处理这些数据。

然而，在默认的配置下，这种方法在失败的情况下会丢失数据，为了保证零数据丢失，你可以在Spark Streaming中使用WAL日志，这是在Spark 1.2.0才引入的功能，这使得我们可以将接收到的数据保存到WAL中（WAL日志可以存储在HDFS上），所以在失败的时候，我们可以从WAL中恢复，而不至于丢失数据。

下面，我将介绍如何使用这种方法来接收数据。

1、引入依赖。

对于Scala和Java项目，你可以在你的pom.xml文件引入以下依赖：

```
<dependency>
  <groupId>org.apache.spark</groupId>
  <artifactId>spark-streaming-kafka_2.10</artifactId>
  <version>1.3.0</version>
</dependency>
```

如果你是使用SBT，可以这么引入：

```
libraryDependencies += "org.apache.spark" % "spark-streaming-kafka_2.10" % "1.3.0"
```

2、编程

在Streaming程序中，引入KafkaUtils，并创建一个输入DStream：

```
import org.apache.spark.streaming.kafka._

val kafkaStream = KafkaUtils.createStream(streamingContext,
    [ZK quorum], [consumer group id], [per-topic number of Kafka partitions to consume])
```

在创建DStream的时候，你也可以指定数据的Key和Value类型，并指定相应的解码类。

需要注意的是：
1、Kafka中Topic的分区和Spark Streaming生成的RDD中分区不是一个概念。所以，在 KafkaUtils.createStream()增加特定主题分区数仅仅是增加一个receiver中消费Topic的线程数。并不增加Spark并行处理数据的数量；

2、对于不同的Group和tpoic我们可以使用多个receivers创建不同的DStreams来并行接收数据；

3、如果你启用了WAL，这些接收到的数据将会被持久化到日志中，因此，我们需要将storage level 设置为StorageLevel.MEMORY_AND_DISK_SER ,也就是：

```
KafkaUtils.createStream(..., StorageLevel.MEMORY_AND_DISK_SER)
```

3、部署

对应任何的Spark 应用，我们都是用spark-submit来启动你的应用程序，对于Scala和Java用户，如果你使用的是SBT或者是Maven，你可以将spark-streaming-kafka_2.10及其依赖打包进应用程序的Jar文件中，并确保spark-core_2.10和 spark-streaming_2.10标记为provided，因为它们在Spark 安装包中已经存在：

```
<dependency>
        <groupId>org.apache.spark</groupId>
        <artifactId>spark-streaming_2.10</artifactId>
        <version>1.3.0</version>
        <scope>provided</scope>
</dependency>

<dependency>
```

```
        <groupId>org.apache.spark</groupId>
        <artifactId>spark-core_2.10</artifactId>
        <version>1.3.0</version>
        <scope>provided</scope>
</dependency>
```

然后使用spark-submit来启动你的应用程序。

当然，你也可以不在应用程序Jar文件中打包spark-streaming-kafka_2.10及其依赖，我们可以在spark-submit后面加上--jars参数也可以运行你的程序：

```
[iteblog@ spark]$ spark-1.3.0-bin-2.6.0/bin/spark-submit  --master yarn-cluster
            --class iteblog.KafkaTest
            --jars lib/spark-streaming-kafka_2.10-1.3.0.jar,
            lib/spark-streaming_2.10-1.3.0.jar,
            lib/kafka_2.10-0.8.1.1.jar,lib/zkclient-0.3.jar,
            lib/metrics-core-2.2.0.jar ./iteblog-1.0-SNAPSHOT.jar
```

下面是一个完整的例子：

```
object KafkaWordCount {
  def main(args: Array[String]) {
    if (args.length < 4) {
      System.err.println("Usage: KafkaWordCount <zkQuorum> <group> <topics> <numThreads>
")
      System.exit(1)
    }

    StreamingExamples.setStreamingLogLevels()

    val Array(zkQuorum, group, topics, numThreads) = args
    val sparkConf = new SparkConf().setAppName("KafkaWordCount")
    val ssc =  new StreamingContext(sparkConf, Seconds(2))
    ssc.checkpoint("checkpoint")

    val topicMap = topics.split(",").map((_,numThreads.toInt)).toMap
    val lines = KafkaUtils.createStream(ssc, zkQuorum, group, topicMap).map(_._2)
    val words = lines.flatMap(_.split(" "))
    val wordCounts = words.map(x => (x, 1L))
```

```
    .reduceByKeyAndWindow(_ + _, _ - _, Minutes(10), Seconds(2), 2)
  wordCounts.print()

  ssc.start()
  ssc.awaitTermination()
 }
}
```