# Spark Streaming 1.3对Kafka整合的提升详解

Apache Kafka近年来迅速地成为开源社区流行的流输入平台。同时我们也看到了Spark Streaming的使用趋势和它类似。因此，在Spark 1.3中，社区对Kafka和Spark Streaming的整合做了很多重要的提升。主要修改如下：

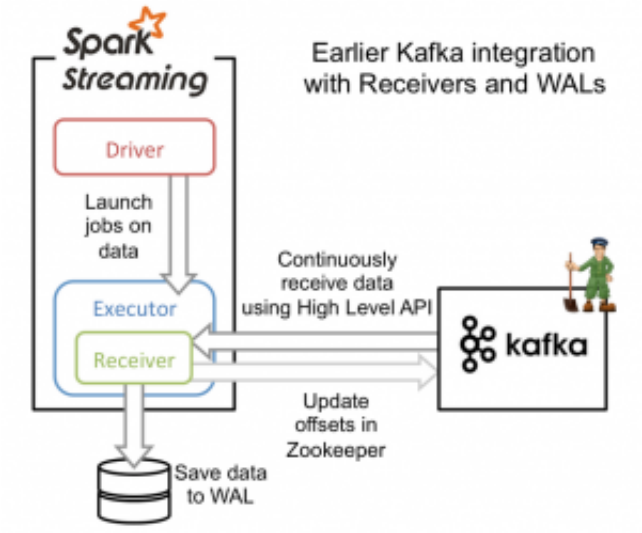
1、为Kafka新增了新的Direct API。这个API可以使得每个Kafka记录仅且被处理一次(processed exactly once)，即使读取过程中出现了失败，这个保证并没有使用到WAL(Write Ahead Logs)。这个使得Spark Streaming和Kafka的流处理跟高效，同时提供更强的容错性担保。
2、为Python新增了Kafka相关API，这样你可以在Python中处理Kafka中的数据。
这篇文章中，我将详细地介绍这些提升。

从Spark Streaming诞生之日起，它就支持与Kafka进行整合，而且在生产环境将Spark Streaming和Kafka整合的公司也越来越多，因为使用的地方很多，所以Spark社区要求能够提供更好的容错性保证和更加可靠的语义保证；为了满足这个需求，Spark 1.2中引入了WAL，他保证从任何可靠的数据源（比如提供事务的数据源：Flume, Kafka和Kinesis）接收到的数据在失败中都不会被丢失；如果数据源不可靠(不提供事务),比如从sockets接收数据，WAL也可以尽量减少数据的丢失。

然而对于那些可以从任意位置读取数据的数据源(比如Kafka)，我们可以实现功能更强的容错，因为Spark Streaming对这些数据源有更强的控制能力。Spark 1.3引入了 Direct API概念，这个可以在不使用WAL的情况下实现仅处理一次的语义(exactly-once semantics)。让我们来详细地讨论一下Apache Spark中Direct API。

## 我们如何构建

从高层次的角度看，之前的和Kafka集成方案使用WAL工作方式如下：
1、运行在Spark workers/executors上的Kafka Receivers连续不断地从Kafka中读取数据，其中用到了Kafka中高层次的消费者API。
2、接收到的数据被存储在Spark workers/executors中的内存，同时也被写入到WAL中。只有接收到的数据被持久化到log中，Kafka Receivers才会去更新Zookeeper中Kafka的偏移量。
3、接收到的数据和WAL存储位置信息被可靠地存储，如果期间出现故障，这些信息被用来从错误中恢复，并继续处理数据。



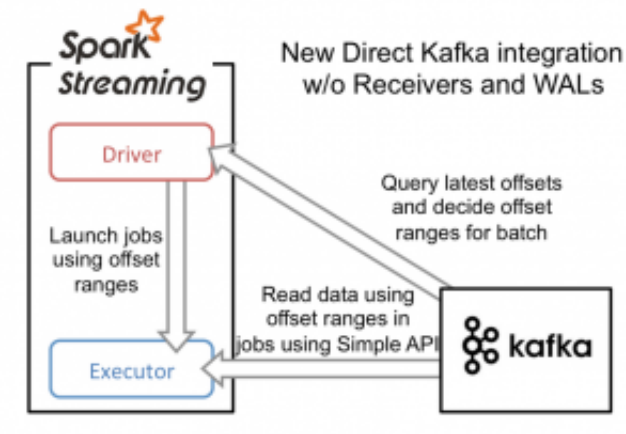

这个方法可以保证从Kafka接收的数据不被丢失。但是在失败的情况下，有些数据很有可能会被处理不止一次！这种情况在一些接收到的数据被可靠地保存到WAL中，但是还没有来得及更新Zookeeper中Kafka偏移量，系统出现故障的情况下发生。这导致数据出现不一致性：Spark Streaming知道数据被接收，但是Kafka那边认为数据还没有被接收，这样在系统恢复正常时，Kafka会再一次发送这些数据。



这种不一致产生的原因是因为两个系统无法对那些已经接收到的数据信息保存进行原子操作。为了解决这个问题，只需要一个系统来维护那些已经发送或接收的一致性视图，而且，这个系统需要拥有从失败中恢复的一切控制权利。基于这些考虑，社区决定将所有的消费偏移量信息只存储在Spark Streaming中，并且使用Kafka的低层次消费者API来从任意位置恢复数据。

为了构建这个系统，新引入的Direct API采用完全不同于Receivers和WALs的处理方式。它不是启动一个Receivers来连续不断地从Kafka中接收数据并写入到WAL中，而且简单地给出每个bat

ch区间需要读取的偏移量位置，最后，每个batch的Job被运行，那些对应偏移量的数据在Kafka中已经准备好了。这些偏移量信息也被可靠地存储（checkpoint），在从失败中恢复可以直接读取这些偏移量信息。




需要注意的是，Spark Streaming可以在失败以后重新从Kafka中读取并处理那些数据段。然而，由于仅处理一次的语义，最后重新处理的结果和没有失败处理的结果是一致的。

因此，Direct API消除了需要使用WAL和Receivers的情况，而且确保每个Kafka记录仅被接收一次并被高效地接收。这就使得我们可以将Spark Streaming和Kafka很好地整合在一起。总体来说，这些特性使得流处理管道拥有高容错性，高效性，而且很容易地被使用。

## 如何来使用

新的API相比之前的更加容易使用：

```
// Define the Kafka parameters, broker list must be specified
val kafkaParams = Map("metadata.broker.list" -> "www.iteblog.com:9092,anotherhost:9092")

// Define which topics to read from
val topics = Set("sometopic", "iteblog")

// Create the direct stream with the Kafka parameters and topics
val kafkaStream = KafkaUtils.createDirectStream[String, String, StringDecoder,
    StringDecoder](streamingContext, kafkaParams, topics)
```

因为新的API没有任何receivers，所以你不需要担心如何创建多个DStreams输入流以便创建多个receivers，而且你也不需要考虑每个receivers需要处理Kafka的分区数量。每个Kafka分区将会被自动地并行处理。而且，每个Kafka分区将会对应RDD中的分区，这使得并行模型变得更加简单。

除了新的Streaming API，社区同时也引入了KafkaUtils.createRDD()，这个API可以对Kafka中的数据运行多个Job。

```
// Define the offset ranges to read in the batch job
val offsetRanges = Array(
  OffsetRange("some-topic", 0, 110, 220),
  OffsetRange("some-topic", 1, 100, 313),
  OffsetRange("another-topic", 0, 456, 789)
)

// Create the RDD based on the offset ranges
val rdd = KafkaUtils.createRDD[String, String, StringDecoder,
    StringDecoder](sparkContext, kafkaParams, offsetRanges)
```

## Python 中的Kafka API

在Spark 1.2中，基本的用于操作Spark Streaming中数据的Python API被引入，所以开发者可以使用纯Python来开发分布式地流处理应用程序。在Spark 1.3中，Spark Streaming流处理引入了Kafka操作API，有了这个API，在Python处理Kafka数据变得非常地容易。下面是简单的代码片段：

```
kafkaStream = KafkaUtils.createStream(streamingContext,
    "www.iteblog.com:2181", "consumer-group", {"some-topic": 1})

lines = kafkaStream.map(lambda x: x[1])
```

需要注意的是，这个实现还只是使用了旧的Kafka API，在Python中使用Direct API正在开发中，很有可能在Spark 1.4版本中可用。

本文翻译自：https://databricks.com/blog/2015/03/30/improvements-to-kafka-integration-of-spark-streaming.html