Spark Streaming的概要与应用

作者：费英林

目录

[1. 系统概要 2](#_Toc428794270)

[2. 基本概念 3](#_Toc428794271)

[2.1. Linking 3](#_Toc428794272)

[2.2. 初始化StreamingContext 4](#_Toc428794273)

[2.3. 离散数据流DStream 4](#_Toc428794274)

[2.4. Input DStream和Receiver 5](#_Toc428794275)

[2.5. DStream的转换 5](#_Toc428794276)

[2.6. DStream的输出操作 6](#_Toc428794277)

[2.7. DataFrame和SQL操作 6](#_Toc428794278)

[2.8. MLlib操作 6](#_Toc428794279)

[2.9. DataFrame和SQL操作 6](#_Toc428794280)

[2.10. 缓存与持久化 6](#_Toc428794281)

[2.11. Checkpointing 6](#_Toc428794282)

[3. 实例 7](#_Toc428794283)

[3.1. Spark Streaming程序 7](#_Toc428794284)

[3.2. Flume配置 9](#_Toc428794285)

[3.3. 打包 10](#_Toc428794286)

[3.4. 启动应用 10](#_Toc428794287)

[3.5. 启动Flume Agent 10](#_Toc428794288)

[3.6. 状态监控 10](#_Toc428794289)

[3.7. 结果确认 11](#_Toc428794290)

# 系统概要

Spark Streaming是Spark API的扩展，提供了可扩展的、高吞吐量的、可容错的实时数据处理。数据源可以是Kafka、Flume、Twitter、ZeroMQ、Kinesis和TCP sockets等，这些数据可以通过map、reduce、join和window等高级函数进行处理，处理后的数据可以存储至文件系统、数据库和仪表板等。我们甚至可以将机器学习和图处理应用于数据流处理的过程中。



它的内部工作机制如下图所示。Spark Streaming接收输入数据流并将接收到的数据划分为多个小的batch，Spark引擎依次处理这些batch，生成结果。



Spark Streaming提供了一个高级抽象 – 离散数据流DStream，它代表一个持续的数据流。DStream可从输入流中创建，如Kafka、Flume和Kinesis，也可以从其它DStream中创建。实际上，DStream是一个RDD的序列。

我们可以使用Scala、Java或Python(Spark 1.2中引入)来实现Spark Streaming。其中针对Python语言提供的API是不完善的，具体细节可参照API文档。

# 基本概念

## Linking

以下是基于Maven的配置。

 <dependency>

 <groupId>org.apache.spark</groupId>

 <artifactId>spark-streaming\_2.10</artifactId>

 <version>1.3.0</version>

 </dependency>

对于Kafka、Flume和Kinesis等，我们需要添加相应的artifact 到Maven里，列表如下：

Kafka - spark-streaming-kafka\_2.10

Flume - spark-streaming-flume\_2.10

Kinesis - spark-streaming-kinesis-asl\_2.10 [Amazon Software License]

Twitter - spark-streaming-twitter\_2.10

ZeroMQ - spark-streaming-zeromq\_2.10

MQTT - spark-streaming-mqtt\_2.10

## 初始化StreamingContext

StreamingContext是Spark Streaming的主入口，初始化代码如下：

import org.apache.spark.\*;

import org.apache.spark.streaming.api.java.\*;

SparkConf conf = new SparkConf().setAppName(appName).setMaster(master);

JavaStreamingContext ssc = new JavaStreamingContext(conf, Duration(1000));

appName是在cluster UI上可看到的应用名称。 Master可以是Spark、Mesos、YARN cluster URL或者local[\*]（本地模式）。在实际应用中，如果是运行在一个集群上，我们不需要设置master参数，可以通过spark-submit提交作业。Batch区间要依据具体应用的实时需求和集群的资源情况进行设定。

JavaStreamingContext也可以从一个已有的JavaSparkContext中创建，如：

import org.apache.spark.streaming.api.java.\*;

JavaSparkContext sc = ... //existing JavaSparkContext

JavaStreamingContext ssc = new JavaStreamingContext(sc, Durations.seconds(1));

在定义了上下文之后，我们需要完成以下工作：

* + - 通过DStream定义输入源；
		- 定义流计算过程 – 在DStream定义转换和输出；
		- 调用streamingContext.start()开始接收、处理数据；
		- 调用streamingContext.awaitTermination()等待处理完成；
		- 可调用amingContext.stop()来手工停止处理过程。

## 离散数据流DStream

DStream表示一个连续数据流，或者是从数据源获得的输入数据流，或者输入数据流处理后的数据流。DStream在内部表示为一个持续的RDD序列。



在DStream上的操作都翻译为底层RDD上的操作。以wordcount应用为例，

flatMap 操作是应用于lines DStream中的每个RDD的，然后生成对应的words DStream。如下图所示：



## Input DStream和Receiver

Input DStreams代表从源数据接收到的输入数据流。每个Input DStream（除了文件流）都与一个Receiver对象关联，Receiver对象从数据源接收数据并存储在Spark的内存中供后续使用。Spark Streaming提供了两类内置的流数据源：

基本源：可从StreamingContext API中直接访问的源，如文件系统，socket连接，和Akka角色。

高级源：如Kafka、Flume、Kinesis和Twitter等，可通过附加的工具类获得。

## DStream的转换

类似于RDD上的操作，transformation可以改变输入流中的数据，具体的transformation说明请参阅API。

Spark Streaming支持窗口操作，即我们可以定义一个滑动的时间窗口，这个时间窗口可以包含一个或多个batch区间。



## DStream的输出操作

DStream的数据可以输出到外部系统，如文件系统或数据库。DStream把持foreachRDD操作，类似于对输入流中的RDD做for循环。

## DataFrame和SQL操作

Spark Streaming支持DataFrames和SQL操作。类似的，我们也可以在其它线程（与当前StreamingContext异步）里定义的表上运行SQL查询。

## MLlib操作

Spark Streaming支持MLlib的包含的机器学习算法。

## DataFrame和SQL操作

Spark Streaming支持DataFrames和SQL操作。类似的，我们也可以在其它线程（与当前StreamingContext异步）里定义的表上运行SQL查询。

## 缓存与持久化

类似于RDD，DStreams允许开发者在内存中持久化数据。对于需要进行多次运算的数据，这个功能很有用。前面提到的时间窗口操作也依赖于这个功能。

## Checkpointing

Spark Streaming支持两类数据做checkpoint：

* 元数据checkpointing - 保存流计算的定义到一个可空错存储，如HDFS。元数据包括配置、DStream操作和未完成的batch。
* 数据checkpointing - 保存产生的RDD到一个可靠存储。

具体的启用Checkpointing及恢复请参考官方文档。

# 实例

以下实例通过Flume接入一个实时增加的文件，Flume通过事件的形式将增加的文件数据加到Spark流处理的接收器，然后对这些数据进行wordcount操作，统计结果存储到一个HBase的表里面。

## Spark Streaming程序

相关的Java代码如下：

package org.apache.spark.examples.streaming;

import java.util.Arrays;

import org.apache.spark.SparkConf;

import org.apache.spark.api.java.JavaPairRDD;

import org.apache.spark.api.java.function.FlatMapFunction;

import org.apache.spark.api.java.function.Function2;

import org.apache.spark.api.java.function.PairFunction;

import org.apache.spark.api.java.function.VoidFunction;

import org.apache.spark.streaming.Duration;

import org.apache.spark.streaming.Time;

import org.apache.spark.streaming.api.java.JavaPairDStream;

import org.apache.spark.streaming.api.java.JavaReceiverInputDStream;

import org.apache.spark.streaming.api.java.JavaStreamingContext;

import org.apache.spark.streaming.flume.FlumeUtils;

import org.apache.spark.streaming.flume.SparkFlumeEvent;

import scala.Tuple2;

public final class JavaFlumeEventCount2 {

 private JavaFlumeEventCount2() {

 }

 public static void main(String[] args) {

 if (args.length != 2) {

 System.err.println("Usage: JavaFlumeEventCount <host> <port>");

 System.exit(1);

 }

 String host = args[0];

 int port = Integer.parseInt(args[1]);

 Duration batchInterval = new Duration(2000);

 SparkConf sparkConf = new SparkConf().setAppName("JavaFlumeEventCount");

 JavaStreamingContext ssc = new JavaStreamingContext(sparkConf,

 batchInterval);

 JavaReceiverInputDStream<SparkFlumeEvent> flumeStream = FlumeUtils

 .createStream(ssc, host, port);

 JavaPairDStream<String, Integer> lastCounts = flumeStream

 .flatMap(new FlatMapFunction<SparkFlumeEvent, String>() {

 // @Override

 public Iterable<String> call(SparkFlumeEvent event)

 throws Exception {

 String bodyString = new String(event.event().getBody()

 .array(), "UTF-8");

 return Arrays.asList(bodyString.split(" "));

 }

 }).mapToPair(new PairFunction<String, String, Integer>() {

 // @Override

 public Tuple2<String, Integer> call(String s) {

 return new Tuple2<String, Integer>(s, 1);

 }

 }).reduceByKey(new Function2<Integer, Integer, Integer>() {

 // @Override

 public Integer call(Integer x, Integer y) throws Exception {

 // TODO Auto-generated method stub

 return x.intValue() + y.intValue();

 }

 });

 lastCounts

 .foreach(new Function2<JavaPairRDD<String, Integer>, Time, Void>() {

 // @Override

 public Void call(JavaPairRDD<String, Integer> values,

 Time time) throws Exception {

 values.foreach(new VoidFunction<Tuple2<String, Integer>>() {

 // @Override

 public void call(Tuple2<String, Integer> tuple)

 throws Exception {

 HBaseCounterIncrementor incrementor = HBaseCounterIncrementor

 .getInstance("spark\_flume", "cf1");

 incrementor.increment("Counter", tuple.\_1(),

 tuple.\_2());

 System.out.println("Counter:" + tuple.\_1()

 + "," + tuple.\_2());

 }

 });

 return null;

 }

 });

 ssc.start();

 ssc.awaitTermination();

 }

}

## Flume配置

创建一个配置文件：/etc/flume-ng/conf/flume-spark-streaming-tail.conf，内容如下：

a3.sources = r1

a3.sinks = k1

a3.channels = c1

# Describe/configure the source

a3.sources.r1.type = exec

a3.sources.r1.command = tail -F /var/log/hadoop-yarn/yarn/yarn-yarn-nodemanager-udh-yf-dev-12.log

# Describe the sink

a3.sinks.k1.type = avro

a3.sinks.k1.hostname = udh-yf-dev-12.yonyou.com

a3.sinks.k1.port = 44123

# Use a channel which buffers events in memory

a3.channels.c1.type = memory

a3.channels.c1.capacity = 1000

a3.channels.c1.transactionCapacity = 100

# Bind the source and sink to the channel

a3.sources.r1.channels = c1

a3.sinks.k1.channel = c1

## 打包

需要注意的是，如果我们的应用中使用到了Spark之外的包，那么需要将这些包一起打包到应用包中，也就是将依赖的包包含进来。通过在Maven中配置Build选项实现这一点。

## 启动应用

启动命令如下：

spark-submit --class org.apache.spark.examples.streaming.JavaFlumeEventCount2 \

--master yarn-cluster \

--deploy-mode cluster \

--num-executors 3 \

--driver-memory 500m \

--executor-memory 500m \

--executor-cores 1 \

/tmp/SparkTest.jar \

udh-yf-dev-12.yonyou.com \

44123

其中/tmp/SparkTest.jar是应用包，udh-yf-dev-12.yonyou.com是Flume sink的目标主机，44123是Flume sink的目标端口，org.apache.spark.examples.streaming.JavaFlumeEventCount2是具体的应用。

## 启动Flume Agent

启动命令如下：

flume-ng agent -n a3 -c /etc/flume-ng/conf -f /etc/flume-ng/conf/flume-spark-streaming-tail.conf

其中阿a3是定义的Agent名称，/etc/flume-ng/conf/flume-spark-streaming-tail.conf是前面提到的Flume配置文件。

启动后，Flume将传输新增的文件数据至指定的主机和端口，我们的Spark Streaming程序将监听那个端口并处理接收到的数据。

## 状态监控

可打开Yarn的UI查看作业运行状态，我们可以看到有一个Spark应用在持续运行。

## 结果确认

打开HBase Shell，查询指定表，可看到数据在不断变化。