《Spark编程基础（Scala版）》

教材官网： http://dblab.xmu.edu.cn/post/spark/

温馨提示：编辑幻灯片母版，可以修改每页PPT的厦大校徽和底部文字

第7章 Spark Streaming

（PPT版本号：2018年7月版本）

扫一扫访问教材官网

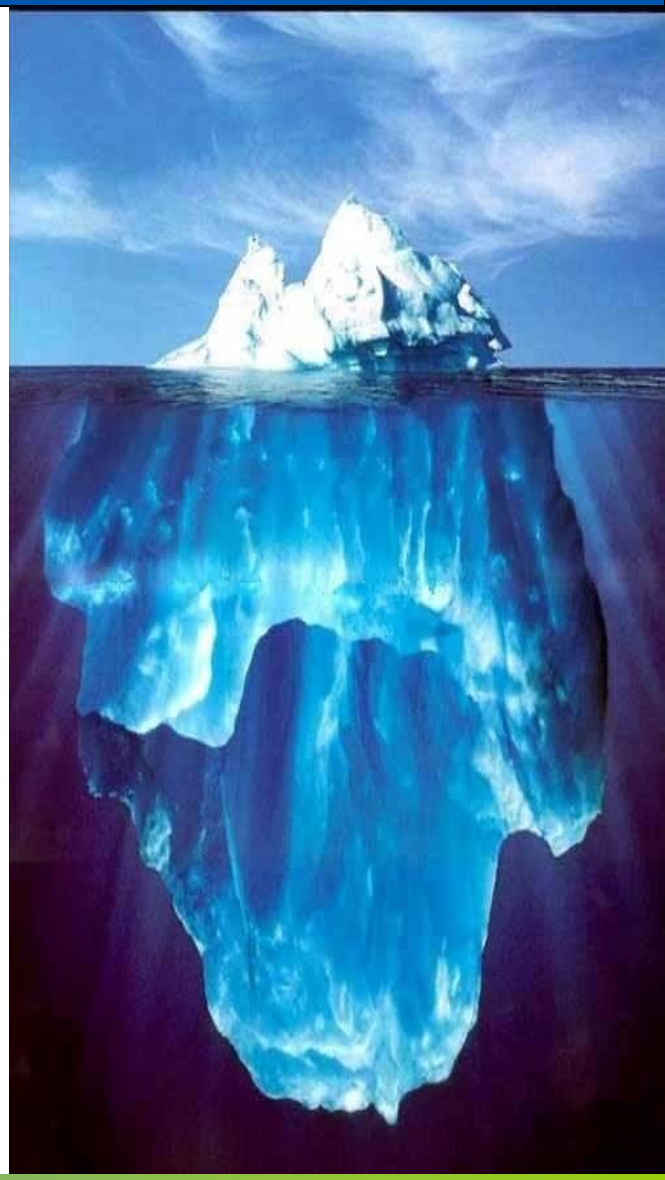
林子雨

厦门大学计算机科学系

E-mail: ziyulin@xmu.edu.cn

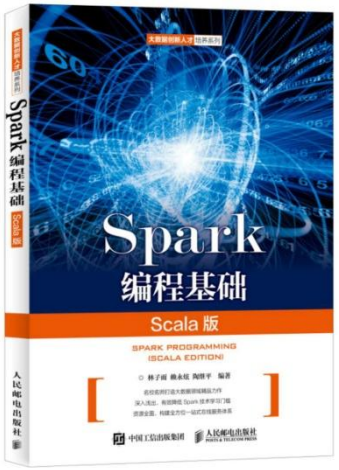
主页：**http://www.cs.xmu.edu.cn/linziyu**

提纲



课程教材

本套讲义PPT属于以下教材的配套材料



《Spark编程基础（Scala版）》

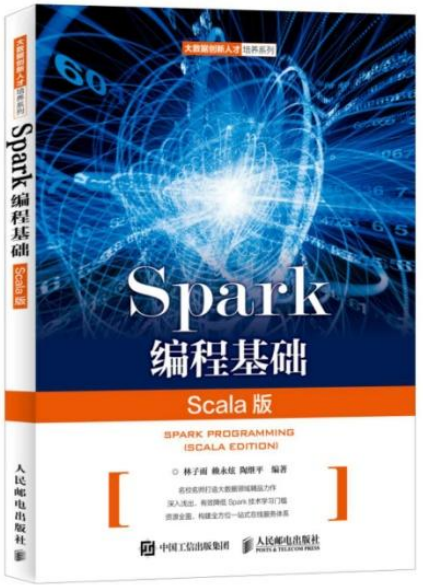
厦门大学 林子雨，赖永炫，陶继平 编著

披荆斩棘，在大数据丛林中开辟学习捷径
填沟削坎，为快速学习Spark技术铺平道路
深入浅出，有效降低Spark技术学习门槛
资源全面，构建全方位一站式在线服务体系

人民邮电出版社出版发行，ISBN:978-7-115-48816-9
教材官网：http://dblab.xmu.edu.cn/post/spark/

本书以Scala作为开发Spark应用程序的编程语言，系统介绍了Spark编程的基础知识。全书共8章，内容包括大数据技术概述、Scala语言基础、Spark的设计与运行原理、Spark环境搭建和使用方法、RDD编程、Spark SQL、Spark Streaming、Spark MLlib等。本书每个章节都安排了入门级的编程实践操作，以便读者更好地学习和掌握Spark编程方法。本书官网免费提供了全套的在线教学资源，包括讲义PPT、习题、源代码、软件、数据集、授课视频、上机实验指南等。

课程配套授课视频



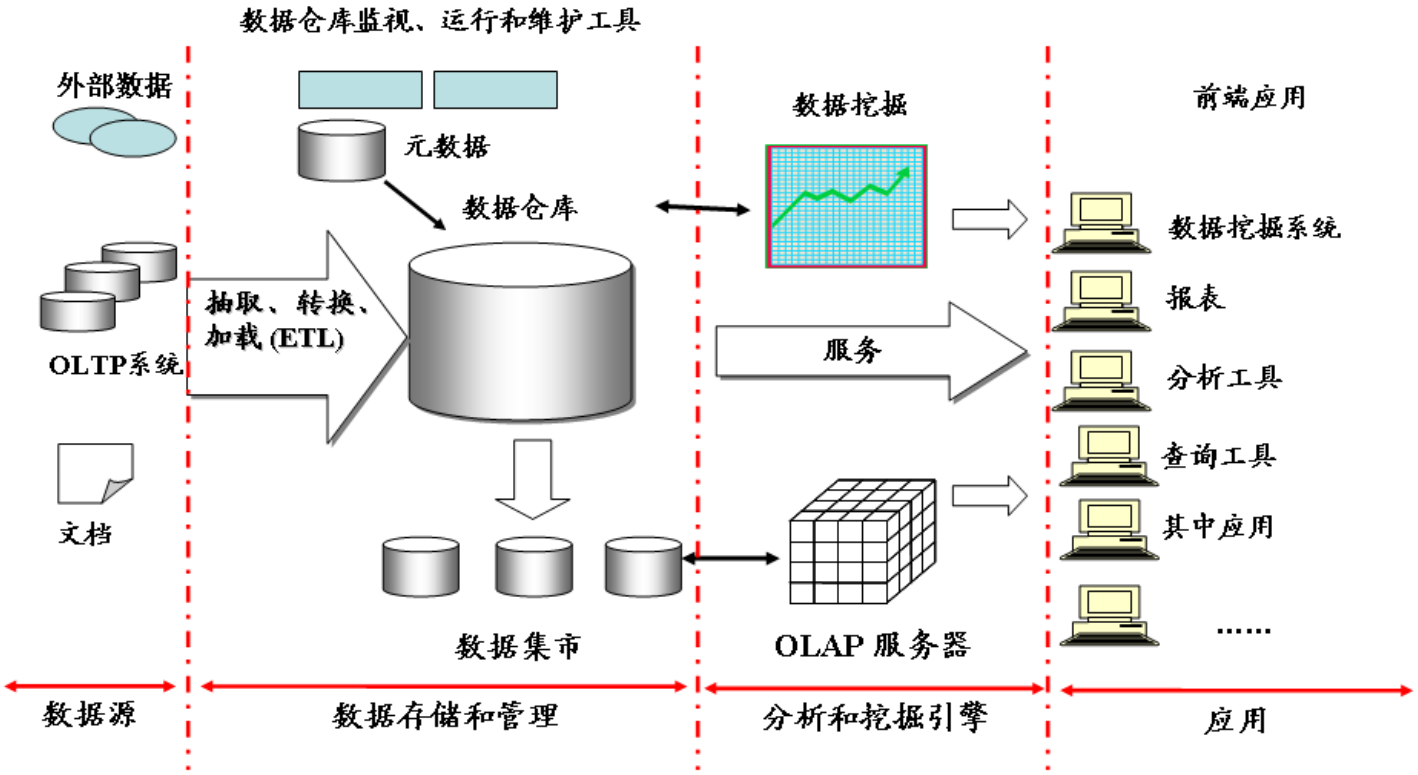
课程在线视频地址：http://dblab.xmu.edu.cn/post/10482/

7.1 流计算概述

- 7.1.1 静态数据和流数据
- 7.1.2 批量计算和实时计算
- 7.1.3 流计算概念
- 7.1.4 流计算框架
- 7.1.5 流计算处理流程

7.1.1 静态数据和流数据

- 很多企业为了支持决策分析而构建的数据仓库系统，其中存放的大量历史数据就是静态数据。技术人员可以利用数据挖掘和OLAP（On-Line Analytical Processing）分析工具从静态数据中找到对企业有价值的信息



7.1.1 静态数据和流数据

- 近年来，在Web应用、网络监控、传感监测等领域，兴起了一种新的数据密集型应用——流数据，即数据以大量、快速、时变的流形式持续到达
- 实例：PM2.5检测、电子商务网站用户点击流

流数据具有如下特征：

- 数据快速持续到达，潜在大小也许是无穷无尽的
- 数据来源众多，格式复杂
- 数据量大，但是不十分关注存储，一旦经过处理，要么被丢弃，要么被归档存储
- 注重数据的整体价值，不过分关注个别数据
- 数据顺序颠倒，或者不完整，系统无法控制将要处理的新到达的数据元素的顺序

7.1.2 批量计算和实时计算

- 对静态数据和流数据的处理，对应着两种截然不同的计算模式：批量计算和实时计算

•批量计算：充裕时间处理静态数据，如Hadoop

•流数据不适合采用批量计算，因为流数据不适合用传统的关系模型建模

•流数据必须采用实时计算，响应时间为秒级

•数据量少时，不是问题，但是，在大数据时代，数据格式复杂、来源众多、数据量巨大，对实时计算提出了很大的挑战。因此，针对流数据的实时计算——流计算，应运而生

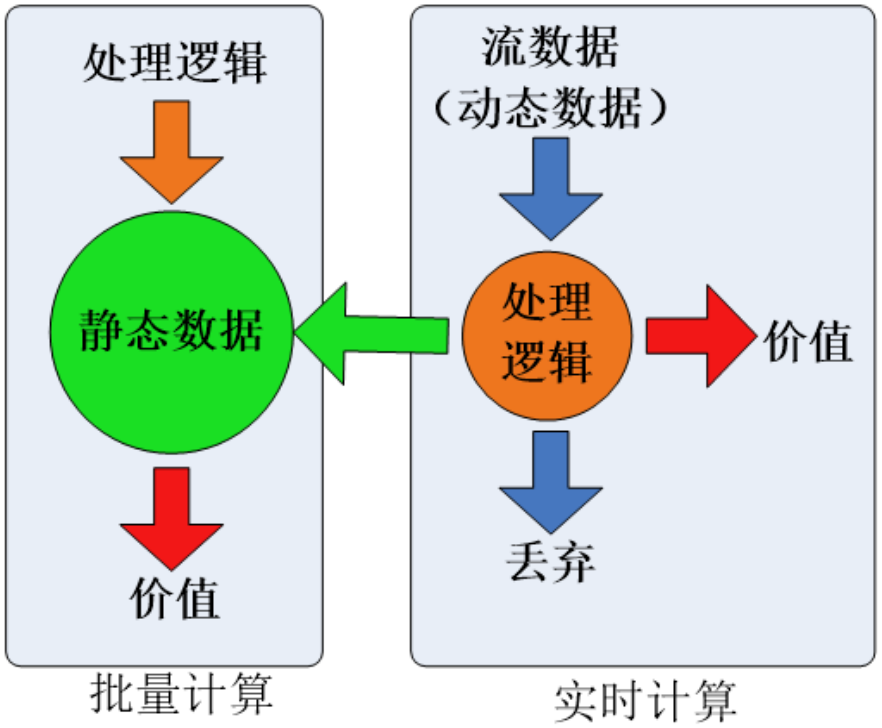


图 数据的两种处理模型

7.1.3 流计算概念

- 流计算：实时获取来自不同数据源的海量数据，经过实时分析处理，获得有价值的信息

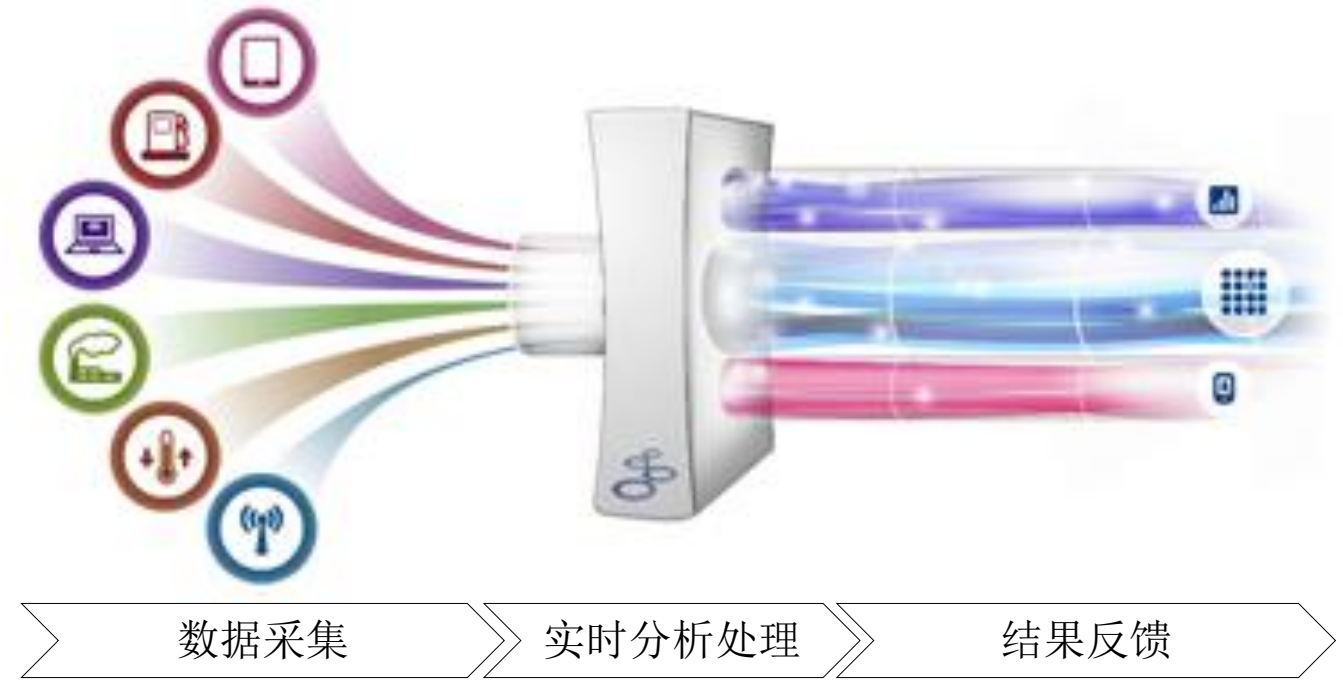


图 流计算示意图

7.1.3 流计算概念

- 流计算秉承一个基本理念，即**数据的价值随着时间的流逝而降低**，如用户点击流。因此，当事件出现时就应该立即进行处理，而不是缓存起来进行批量处理。为了及时处理流数据，就需要一个低延迟、可扩展、高可靠的处理引擎

对于一个流计算系统来说，它应达到如下需求：

- 高性能：处理大数据的基本要求，如每秒处理几十万条数据
- 海量式：支持TB级甚至是PB级的数据规模
- 实时性：保证较低的延迟时间，达到秒级别，甚至是毫秒级别
- 分布式：支持大数据的基本架构，必须能够平滑扩展
- 易用性：能够快速进行开发和部署
- 可靠性：能可靠地处理流数据

7.1.4 流计算框架

- 当前业界诞生了许多专门的流数据实时计算系统来满足各自需求
- 目前有三类常见的流计算框架和平台：商业级的流计算平台、开源流计算框架、公司为支持自身业务开发的流计算框架
- 商业级：IBM InfoSphere Streams和IBM StreamBase
- 较为常见的是开源流计算框架，代表如下：
  - Twitter Storm：免费、开源的分布式实时计算系统，可简单、高效、可靠地处理大量的流数据
  - Yahoo! S4（Simple Scalable Streaming System）：开源流计算平台，是通用的、分布式的、可扩展的、分区容错的、可插拔的流式系统
- 公司为支持自身业务开发的流计算框架：
  - Facebook Puma
  - Dstream（百度）
  - 银河流数据处理平台（淘宝）

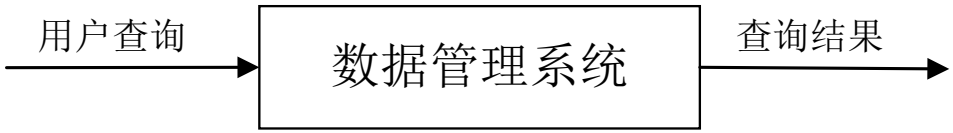
7.1.5 流计算处理流程

- 1. 概述
- 2. 数据实时采集
- 3. 数据实时计算
- 4. 实时查询服务

7.1.5 流计算处理流程

1. 概述

- 传统的数据处理流程，需要先采集数据并存储在关系数据库等数据管理系统中，之后由用户通过查询操作和数据管理系统进行交互

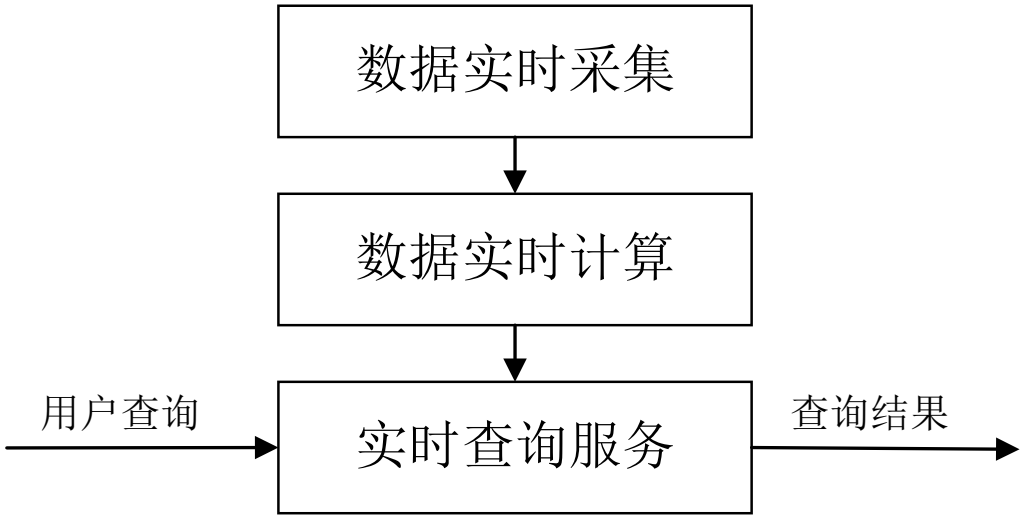


传统的数据处理流程示意图

- 传统的数据处理流程隐含了两个前提：
  - **存储的数据是旧的**。存储的静态数据是过去某一时刻的快照，这些数据在查询时可能已不具备时效性了
  - **需要用户主动发出查询来获取结果**

7.1.5 流计算处理流程

- 流计算的处理流程一般包含三个阶段：数据实时采集、数据实时计算、实时查询服务

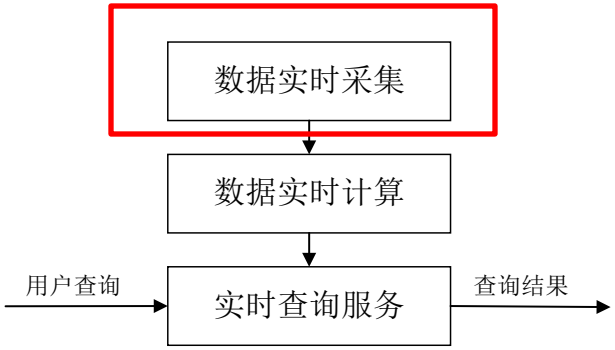


流计算处理流程示意图

7.1.5 流计算处理流程

2. 数据实时采集

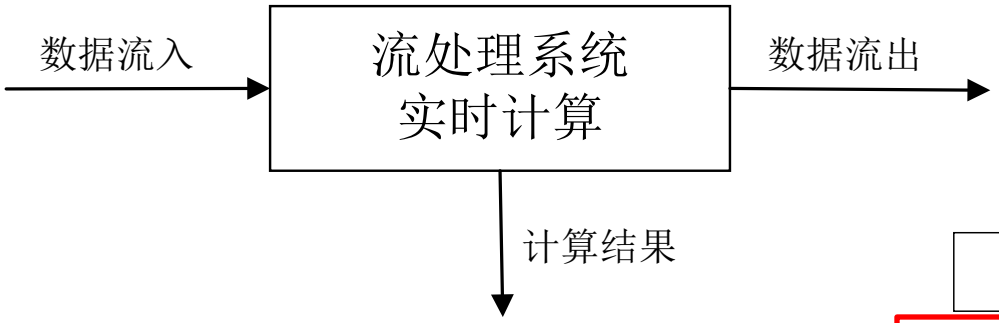
- 数据实时采集阶段通常采集多个数据源的海量数据，需要保证实时性、低延迟与稳定可靠
- 以日志数据为例，由于分布式集群的广泛应用，数据分散存储在不同的机器上，因此需要实时汇总来自不同机器上的日志数据
- 目前有许多互联网公司发布的开源分布式日志采集系统均可满足每秒数百MB的数据采集和传输需求，如：
  - Facebook的Scribe
  - LinkedIn的Kafka
  - 淘宝的Time Tunnel
  - 基于Hadoop的Chukwa和Flume



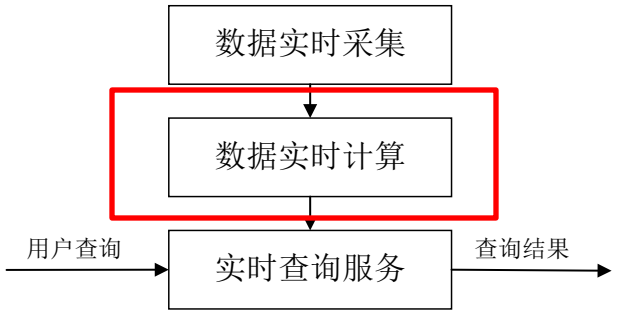
7.1.5 流计算处理流程

3. 数据实时计算

- 数据实时计算阶段对采集的数据进行实时的分析和计算，并反馈实时结果
- 经流处理系统处理后的数据，可视情况进行存储，以便之后再进行分析计算。在时效性要求较高的场景中，处理之后的数据也可以直接丢弃



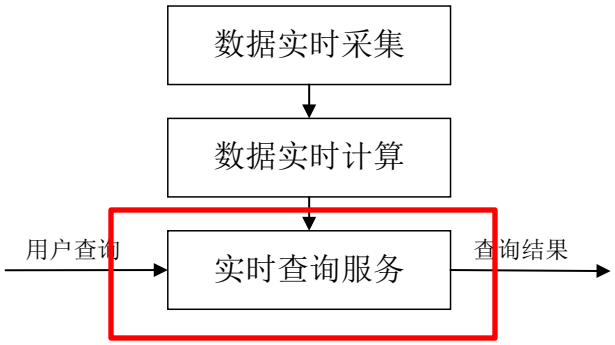
数据实时计算流程



7.1.5 流计算处理流程

4.实时查询服务

- 实时查询服务：经由流计算框架得出的结果可供用户进行实时查询、展示或储存
- 传统的数据处理流程，用户需要主动发出查询才能获得想要的结果。而在流处理流程中，实时查询服务可以不断更新结果，并将用户所需的结果实时推送给用户
- 虽然通过对传统的数据处理系统进行**定时**查询，也可以实现不断地更新结果和结果推送，但通过这样的方式获取的结果，仍然是根据过去某一时刻的数据得到的结果，与实时结果有着本质的区别



7.1.5 流计算处理流程

- 可见，流处理系统与传统的数据处理系统有如下不同：
  - 流处理系统处理的是实时的数据，而传统的数据处理系统处理的是预先存储好的静态数据
  - 用户通过流处理系统获取的是实时结果，而通过传统的数据处理系统，获取的是过去某一时刻的结果
  - 流处理系统无需用户主动发出查询，实时查询服务可以主动将实时结果推送给用户

7.2 Spark Streaming

7.2.1 Spark Streaming设计
7.2.2 Spark Streaming与Storm的对比
7.2.3 从“Hadoop+Storm”架构转向Spark架构

7.2.1 Spark Streaming设计

•Spark Streaming可整合多种输入数据源，如Kafka、Flume、HDFS，甚至是普通的TCP套接字。经处理后的数据可存储至文件系统、数据库，或显示在仪表盘里

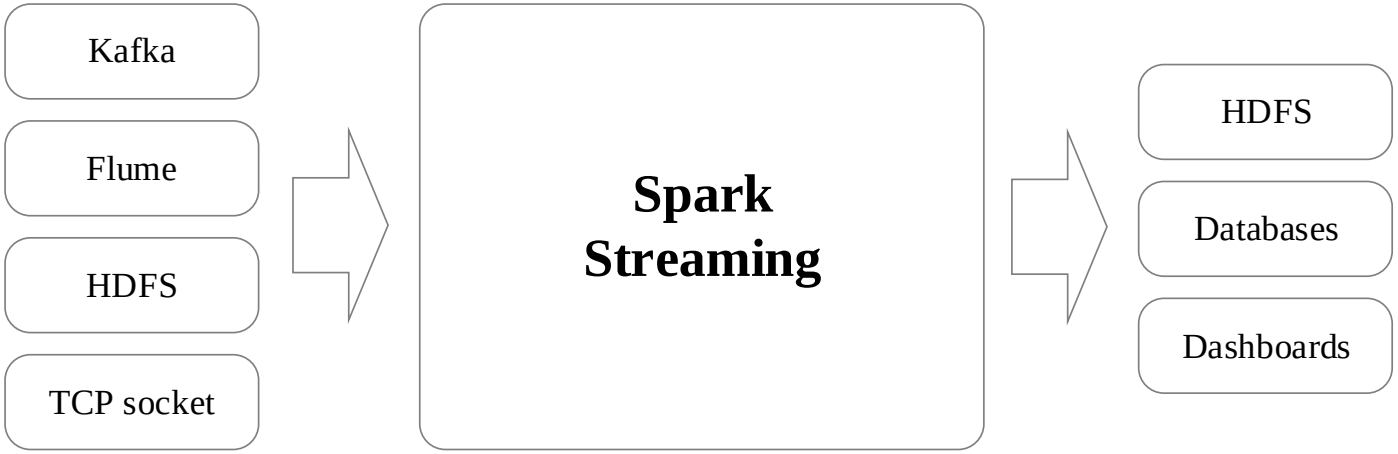


图 Spark Streaming支持的输入、输出数据源

7.2.1 Spark Streaming设计

Spark Streaming的基本原理是将实时输入数据流以时间片（秒级）为单位进行拆分，然后经Spark引擎以类似批处理的方式处理每个时间片数据

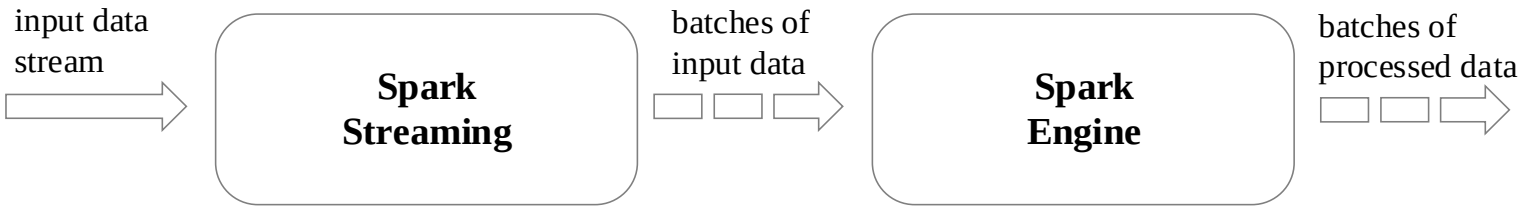


图 Spark Streaming执行流程

7.2.1 Spark Streaming设计

Spark Streaming最主要的抽象是DStream（Discretized Stream，离散化数据流），表示连续不断的数据流。在内部实现上，Spark Streaming的输入数据按照时间片（如1秒）分成一段一段，每一段数据转换为Spark中的RDD，这些分段就是Dstream，并且对DStream的操作都最终转变为对相应的RDD的操作

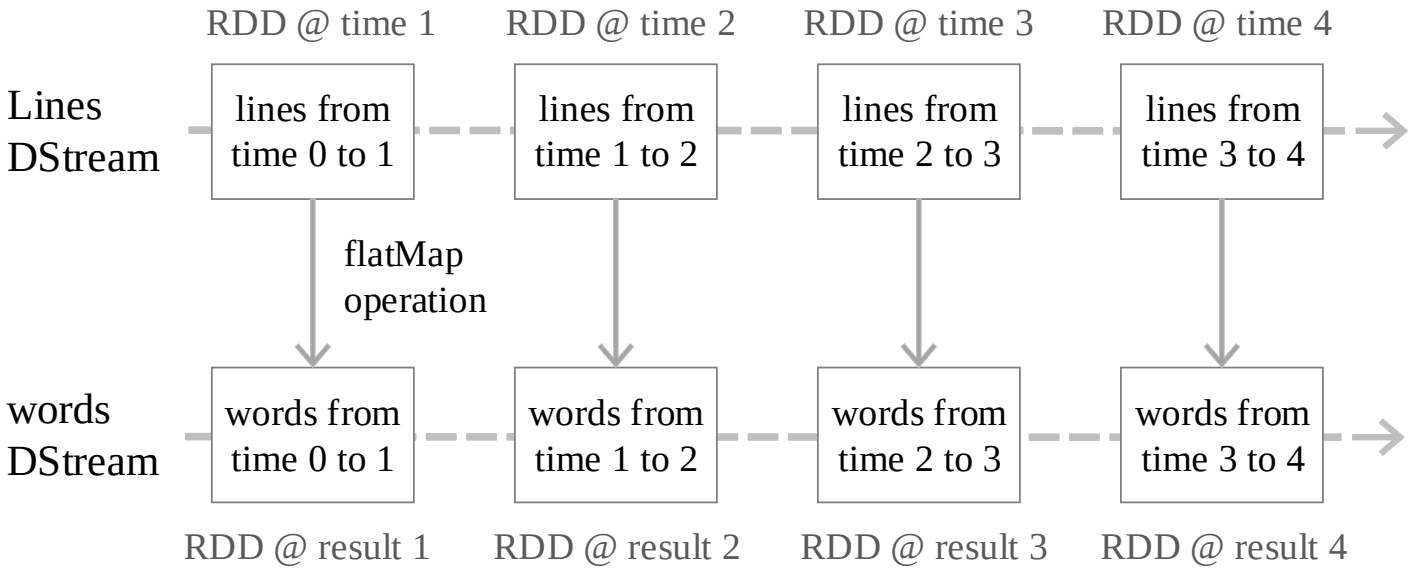


图 DStream操作示意图

7.2.2 Spark Streaming与Storm的对比

•Spark Streaming和Storm最大的区别在于，Spark Streaming无法实现毫秒级的流计算，而Storm可以实现毫秒级响应

•Spark Streaming构建在Spark上，一方面是因为Spark的低延迟执行引擎（100ms+）可以用于实时计算，另一方面，相比于Storm，RDD数据集更容易做高效的容错处理

•Spark Streaming采用的小批量处理的方式使得它可以同时兼容批量和实时数据处理的逻辑和算法，因此，方便了一些需要历史数据和实时数据联合分析的特定应用场合

7.2.3 从“Hadoop+Storm”架构转向Spark架构

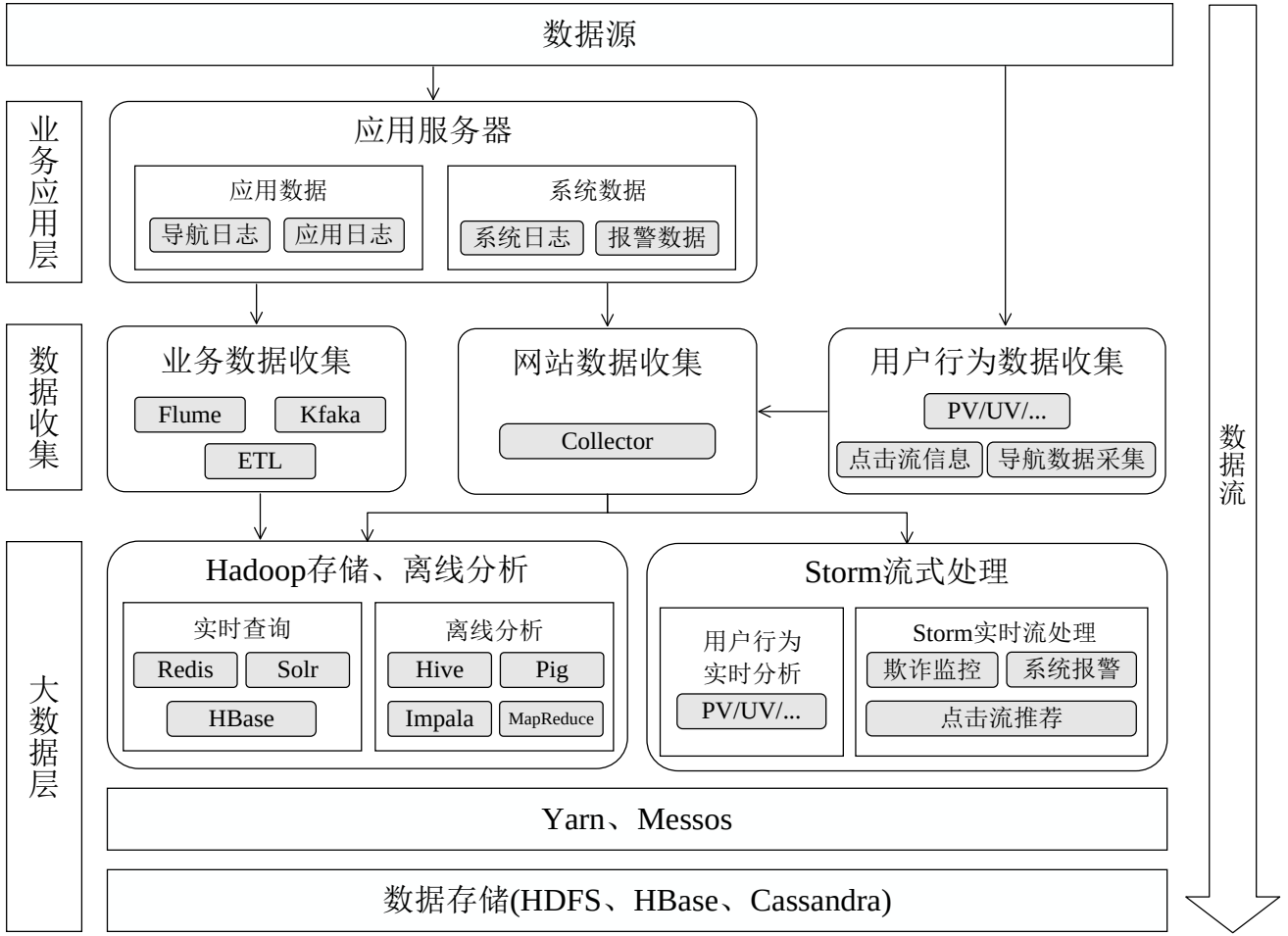


图 采用Hadoop+Storm部署方式的一个案例

7.2.3 从“Hadoop+Storm”架构转向Spark架构

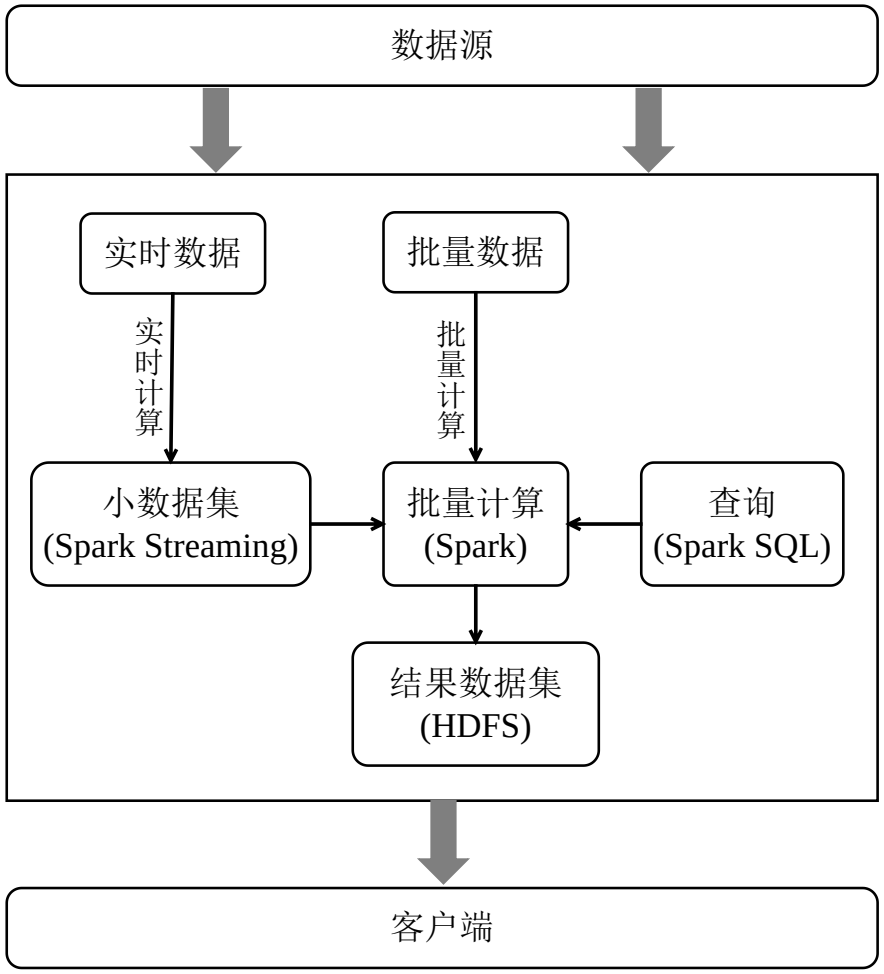


图　用Spark架构满足批处理和流处理需求

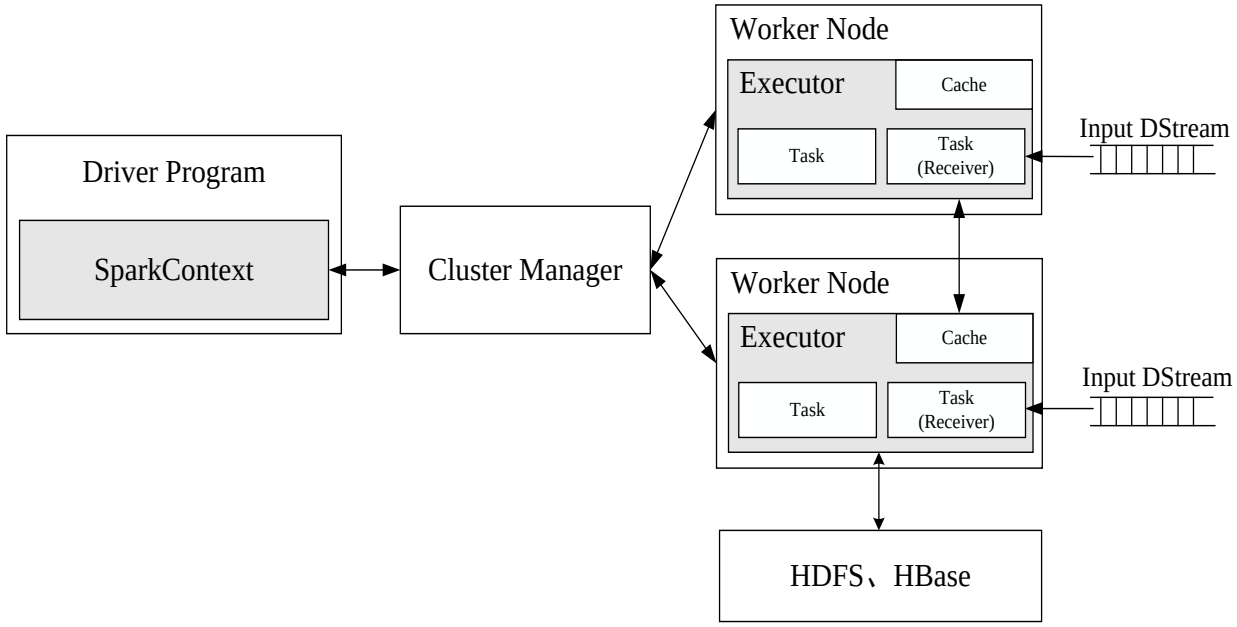
采用Spark架构具有如下优点：

- 实现一键式安装和配置、线程级别的任务监控和告警；
- 降低硬件集群、软件维护、任务监控和应用开发的难度；
- 便于做成统一的硬件、计算平台资源池。

7.3 DStream操作概述

7.3.1 Spark Streaming工作机制
7.3.2 Spark Streaming程序的基本步骤
7.3.3 创建StreamingContext对象

7.3.1 Spark Streaming工作机制



- 在Spark Streaming中，会有一个组件Receiver，作为一个长期运行的task跑在一个Executor上
- 每个Receiver都会负责一个input DStream（比如从文件中读取数据的文件流，比如套接字流，或者从Kafka中读取的一个输入流等等）
- Spark Streaming通过input DStream与外部数据源进行连接，读取相关数据

7.3.2 Spark Streaming程序的基本步骤

编写Spark Streaming程序的基本步骤是：

1.通过创建输入DStream来定义输入源

2.通过对DStream应用转换操作和输出操作来定义流计算

3.用streamingContext.start()来开始接收数据和处理流程

4.通过streamingContext.awaitTermination()方法来等待处理结束（手动结束或因为错误而结束）

5.可以通过streamingContext.stop()来手动结束流计算进程

7.3.3 创建StreamingContext对象

•如果要运行一个Spark Streaming程序，就需要首先生成一个StreamingContext对象，它是Spark Streaming程序的主入口

•可以从一个SparkConf对象创建一个StreamingContext对象
•登录Linux系统后，启动spark-shell。进入spark-shell以后，就已经获得了一个默认的SparkConext，也就是sc。因此，可以采用如下方式来创建StreamingContext对象：

```
scala> import org.apache.spark.streaming._
scala> val ssc = new StreamingContext(sc, Seconds(1))
```

7.3.3 创建StreamingContext对象

如果是编写一个独立的Spark Streaming程序，而不是在spark-shell中运行，则需要通过如下方式创建StreamingContext对象：

```
import org.apache.spark._
import org.apache.spark.streaming._
val conf = new
SparkConf().setAppName("TestDStream").setMaster("local[2]")
val ssc = new StreamingContext(conf, Seconds(1))
```

7.4 基本输入源

7.4.1 文件流
7.4.2 套接字流
7.4.3 RDD队列流

7.4.1 文件流

1.在spark-shell中创建文件流

```
$ cd /usr/local/spark/mycode
$ mkdir streaming
$ cd streaming
$ mkdir logfile
$ cd logfile
```

7.4.1 文件流

进入spark-shell创建文件流。请另外打开一个终端窗口，启动进入spark-shell

```
scala> import org.apache.spark.streaming._
scala> val ssc = new StreamingContext(sc, Seconds(20))
scala> val lines =
ssc.textFileStream("file:///usr/local/spark/mycode/streaming/logfile")
scala> val words = lines.flatMap(_.split(" "))
scala> val wordCounts = words.map(x => (x, 1)).reduceByKey(_ +
_)
scala> wordCounts.print()
scala> ssc.start()
scala> ssc.awaitTermination()
```

7.4.1 文件流

上面在spark-shell中执行的程序，一旦你输入ssc.start()以后，程序就开始自动进入循环监听状态，屏幕上会显示一堆的信息，如下：

```
//这里省略若干屏幕信息
-------------------------------------------
Time: 1479431100000 ms
-------------------------------------------
//这里省略若干屏幕信息
-------------------------------------------
Time: 1479431120000 ms
-------------------------------------------
//这里省略若干屏幕信息
-------------------------------------------
Time: 1479431140000 ms
-------------------------------------------
```

在“/usr/local/spark/mycode/streaming/logfile”目录下新建一个log.txt文件，就可以在监听窗口中显示词频统计结果

7.4.1 文件流

2. 采用独立应用程序方式创建文件流

```
$ cd /usr/local/spark/mycode
$ mkdir streaming
$ cd streaming
$ mkdir -p src/main/scala
$ cd src/main/scala
$ vim TestStreaming.scala
```

7.4.1 文件流

用vim编辑器新建一个TestStreaming.scala代码文件，请在里面输入以下代码：

```
import org.apache.spark._
import org.apache.spark.streaming._
object WordCountStreaming {
  def main(args: Array[String]) {
    val sparkConf = new
SparkConf().setAppName("WordCountStreaming").setMaster("local[2]")//设置为本地运行模式，2个线程，一个监听，另一个处理数据
    val ssc = new StreamingContext(sparkConf, Seconds(2))// 时间间隔为2秒
    val lines = ssc.textFileStream("file:///usr/local/spark/mycode/streaming/logfile")  //这里采用本地文件，当然你也可以采用HDFS文件
    val words = lines.flatMap(_.split(" "))
    val wordCounts = words.map(x => (x, 1)).reduceByKey(_ + _)
    wordCounts.print()
    ssc.start()
    ssc.awaitTermination()
  }
}
```

7.4.1 文件流

```
$ cd /usr/local/spark/mycode/streaming
$ vim simple.sbt
```

在simple.sbt文件中输入以下代码：

```
name := "Simple Project"
version := "1.0"
scalaVersion := "2.11.8"
libraryDependencies += "org.apache.spark" % "spark-streaming_2.11" % "2.1.0"
```

执行sbt打包编译的命令如下：

```
$ cd /usr/local/spark/mycode/streaming
$ /usr/local/sbt/sbt package
```

7.4.1 文件流

打包成功以后，就可以输入以下命令启动这个程序：

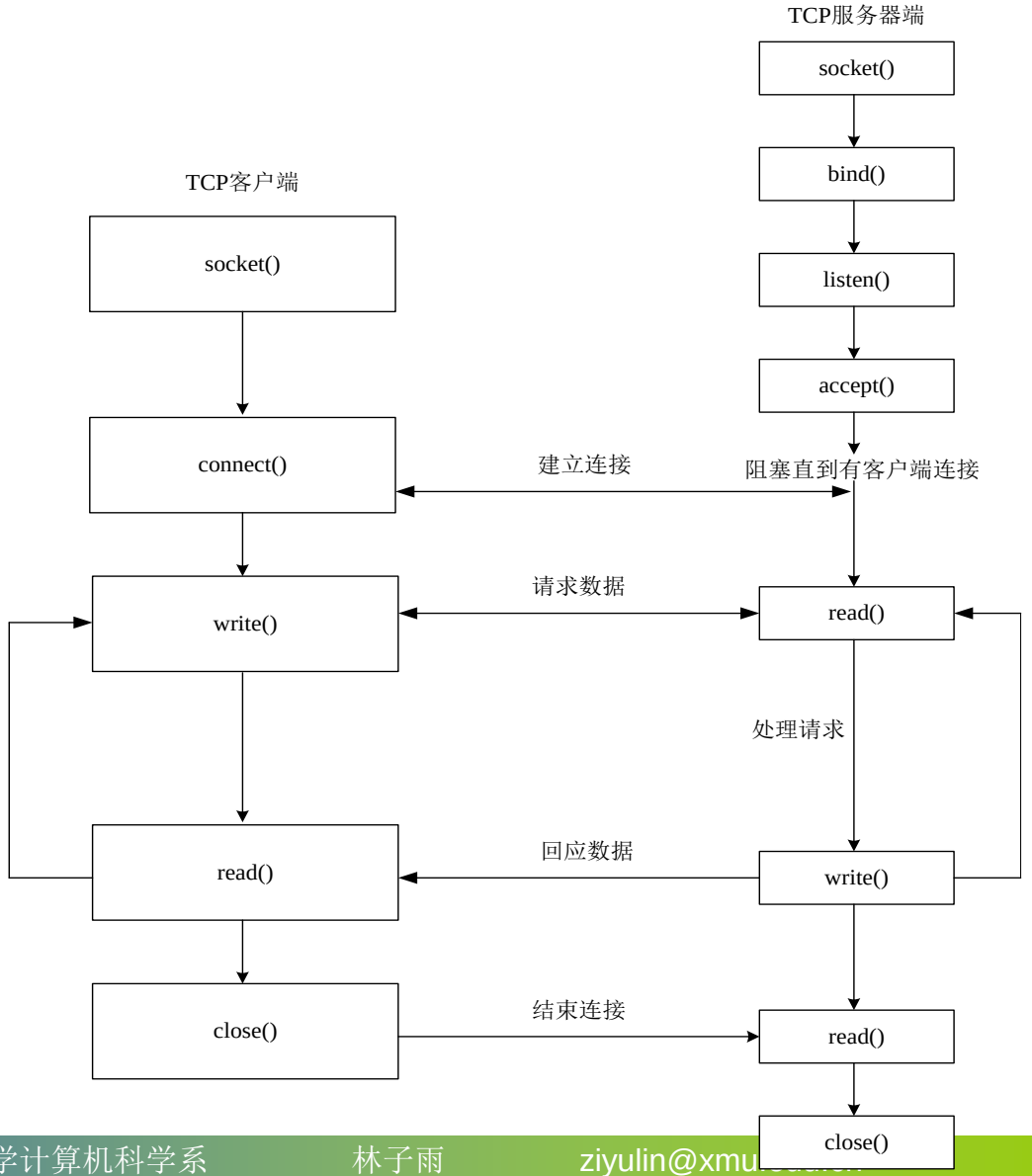
```
$ cd /usr/local/spark/mycode/streaming
$ /usr/local/spark/bin/spark-submit --class
"WordCountStreaming“  /usr/local/spark/mycode/streaming/target/scala-
2.11/simple-project_2.11-1.0.jar
```

- 执行上面命令后，就进入了监听状态（我们把运行这个监听程序的窗口称为监听窗口）
- 切换到另外一个Shell窗口，在"/usr/local/spark/mycode/streaming/logfile"目录下再新建一个log2.txt文件，文件里面随便输入一些单词，保存好文件退出vim编辑器
- 再次切换回“监听窗口”，等待20秒以后，按键盘Ctrl+C或者Ctrl+D停止监听程序，就可以看到监听窗口的屏幕上会打印出单词统计信息

7.4.2 套接字流

•Spark Streaming可以通过Socket端口监听并接收数据，然后进行相应处理

1.Socket工作原理



7.4.2 套接字流

2.使用套接字流作为数据源

```
$ cd /usr/local/spark/mycode
$ mkdir streaming #如果已经存在该目录，则不用创建
$ mkdir -p /src/main/scala #如果已经存在该目录，则不用创建
$ cd /usr/local/spark/mycode/streaming/src/main/scala
$ vim NetworkWordCount.scala
```

请在NetworkWordCount.scala文件中输入如下内容：

```
package org.apache.spark.examples.streaming
import org.apache.spark._
import org.apache.spark.streaming._
import org.apache.spark.storage.StorageLevel
```

剩余代码在下一页

```
object NetworkWordCount {
  def main(args: Array[String]) {
    if (args.length < 2) {
      System.err.println("Usage: NetworkWordCount <hostname> <port>")
      System.exit(1)
    }
    StreamingExamples.setStreamingLogLevels()
    val sparkConf = new
SparkConf().setAppName("NetworkWordCount").setMaster("local[2]")
    val ssc = new StreamingContext(sparkConf, Seconds(1))
    val lines = ssc.socketTextStream(args(0), args(1).toInt,
StorageLevel.MEMORY_AND_DISK_SER)
    val words = lines.flatMap(_.split(" "))
    val wordCounts = words.map(x => (x, 1)).reduceByKey(_ + _)
    wordCounts.print()
    ssc.start()
    ssc.awaitTermination()
  }
}
```

7.4.2 套接字流

在相同目录下再新建另外一个代码文件StreamingExamples.scala，文件内容如下：

```
package org.apache.spark.examples.streaming
import org.apache.spark.internal.Logging
import org.apache.log4j.{Level, Logger}
/** Utility functions for Spark Streaming examples. */
object StreamingExamples extends Logging {
  /** Set reasonable logging levels for streaming if the user has not configured log4j. */
  def setStreamingLogLevels() {
    val log4jInitialized = Logger.getRootLogger.getAllAppenders.hasMoreElements
    if (!log4jInitialized) {
      // We first log something to initialize Spark's default logging, then we override the
      // logging level.
      logInfo("Setting log level to [WARN] for streaming example." +
        " To override add a custom log4j.properties to the classpath.")
      Logger.getRootLogger.setLevel(Level.WARN)
    }
  }
}
```

7.4.2 套接字流

```
$ cd /usr/local/spark/mycode/streaming/
$ vim simple.sbt
```

```
name := "Simple Project"
version := "1.0"
scalaVersion := "2.11.8"
libraryDependencies += "org.apache.spark" % "spark-streaming_2.11" % "2.1.0"
```

```
$ cd /usr/local/spark/mycode/streaming
$ /usr/local/sbt/sbt package
```

```
$ cd /usr/local/spark/mycode/streaming
$ /usr/local/spark/bin/spark-submit --class "
org.apache.spark.examples.streaming.NetworkWordCount"
/usr/local/spark/mycode/streaming/target/scala-2.11/simple-project_2.11-
1.0.jar localhost 9999
```

7.4.2 套接字流

新打开一个窗口作为nc窗口，启动nc程序：

```
$ nc -lk 9999
```

•可以在nc窗口中随意输入一些单词，监听窗口就会自动获得单词数据流信息，在监听窗口每隔1秒就会打印出词频统计信息，大概会在屏幕上出现类似如下的结果：

```
-------------------------------------------
Time: 1479431100000 ms
-------------------------------------------
(hello,1)
(world,1)
-------------------------------------------
Time: 1479431120000 ms
-------------------------------------------
(hadoop,1)
-------------------------------------------
Time: 1479431140000 ms
-------------------------------------------
(spark,1)
```

7.4.2 套接字流

3. 使用Socket编程实现自定义数据源

•下面我们再前进一步，把数据源头的产生方式修改一下，不要使用nc程序，而是采用自己编写的程序产生Socket数据源

```
$ cd /usr/local/spark/mycode/streaming/src/main/scala
$ vim DataSourceSocket.scala
```

```
package org.apache.spark.examples.streaming
Import  java.io.{PrintWriter}
Import  java.net.ServerSocket
Import  scala.io.Source
```

剩余代码见下一页

7.4.2 套接字流

```
object DataSourceSocket {
  def index(length: Int) = {
    val rdm = new java.util.Random
    rdm.nextInt(length)
  }
  def main(args: Array[String]) {
    if (args.length != 3) {
      System.err.println("Usage: <filename> <port>
<millisecond>")
      System.exit(1)
    }
    val fileName = args(0)
    val lines = Source.fromFile(fileName).getLines.toList
    val rowCount = lines.length
```

剩余代码见下一页

7.4.2 套接字流

```
val listener = new ServerSocket(args(1).toInt)
  while (true) {
    val socket = listener.accept()
    new Thread() {
      override def run = {
        println("Got client connected from: " + socket.getInetAddress)
        val out = new PrintWriter(socket.getOutputStream(), true)
        while (true) {
          Thread.sleep(args(2).toLong)
          val content = lines(index(rowCount))
          println(content)
          out.write(content + '\n')
          out.flush()
        }
        socket.close()
      }
    }.start()
  }
}
```

7.4.2 套接字流

执行sbt打包编译：

```
$ cd /usr/local/spark/mycode/streaming
$ /usr/local/sbt/sbt package
```

DataSourceSocket程序需要把一个文本文件作为输入参数，所以，在启动这个程序之前，需要首先创建一个文本文件word.txt并随便输入几行内容：/usr/local/spark/mycode/streaming/word.txt

启动DataSourceSocket程序：

```
$ /usr/local/spark/bin/spark-submit   \
> --class "org.apache.spark.examples.streaming.DataSourceSocket"   \
>/usr/local/spark/mycode/streaming/target/scala-2.11/simple-project_2.11-1.0.jar  \
> /usr/local/spark/mycode/streaming/word.txt  9999 1000
```

这个窗口会不断打印出一些随机读取到的文本信息，这些信息也是Socket数据源，会被监听程序捕捉到

7.4.2 套接字流

在另外一个窗口启动监听程序：

```
$ /usr/local/spark/bin/spark-submit --class
"org.apache.spark.examples.streaming.NetworkWordCount"
/usr/local/spark/mycode/streaming/target/scala-2.11/simple-project_2.11-
1.0.jar localhost 9999
```

启动成功后，你就会看到，屏幕上不断打印出词频统计信息

7.4.3 RDD队列流

•在调试Spark Streaming应用程序的时候，我们可以使用streamingContext.queueStream(queueOfRDD)创建基于RDD队列的DStream

•新建一个TestRDDQueueStream.scala代码文件，功能是：每隔1秒创建一个RDD，Streaming每隔2秒就对数据进行处理

```
package org.apache.spark.examples.streaming
import org.apache.spark.SparkConf
import org.apache.spark.rdd.RDD
import org.apache.spark.streaming.StreamingContext._
import org.apache.spark.streaming.{Seconds,
StreamingContext}
```

剩余代码在下一页

7.4.3 RDD队列流

```
object QueueStream {
  def main(args: Array[String]) {
    val sparkConf = new
SparkConf().setAppName("TestRDDQueue").setMaster("loc
al[2]")
    val ssc = new StreamingContext(sparkConf, Seconds(2))
    val rddQueue =new
scala.collection.mutable.SynchronizedQueue[RDD[Int]]()
    val queueStream = ssc.queueStream(rddQueue)
    val mappedStream = queueStream.map(r => (r % 10, 1))
    val reducedStream = mappedStream.reduceByKey(_ + _)
    reducedStream.print()
    ssc.start()
```

剩余代码见下一页

7.4.3 RDD队列流

```
for (i <- 1 to 10){
        rddQueue += ssc.sparkContext.makeRDD(1 to
100,2)
        Thread.sleep(1000)
    }
    ssc.stop()
  }
}
```

7.4.3 RDD队列流

sbt打包成功后，执行下面命令运行程序：

```
$ cd /usr/local/spark/mycode/streaming
$ /usr/local/spark/bin/spark-submit   \
>--class "org.apache.spark.examples.streaming.QueueStream"   \
>/usr/local/spark/mycode/streaming/target/scala-2.11/simple-project_2.11-1.0.jar
```

执行上面命令以后，程序就开始运行，就可以看到类似下面的结果：

```
-------------------------------------------
Time: 1479522100000 ms
-------------------------------------------
(4,10)
(0,10)
(6,10)
(8,10)
(2,10)
(1,10)
(3,10)
(7,10)
(9,10)
(5,10)
```

7.5 高级数据源

7.5.1 Kafka简介
7.5.2 Kafka准备工作
7.5.3 Spark准备工作
7.5.4 编写Spark Streaming程序使用Kafka数据源

7.5.1 Kafka简介

- Kafka是一种高吞吐量的分布式发布订阅消息系统，用户通过Kafka系统可以发布大量的消息，同时也能实时订阅消费消息
- Kafka可以同时满足在线实时处理和批量离线处理

- 在公司的大数据生态系统中，可以把Kafka作为数据交换枢纽，不同类型的分布式系统（关系数据库、NoSQL数据库、流处理系统、批处理系统等），可以统一接入到Kafka，实现和Hadoop各个组件之间的不同类型数据的实时高效交换

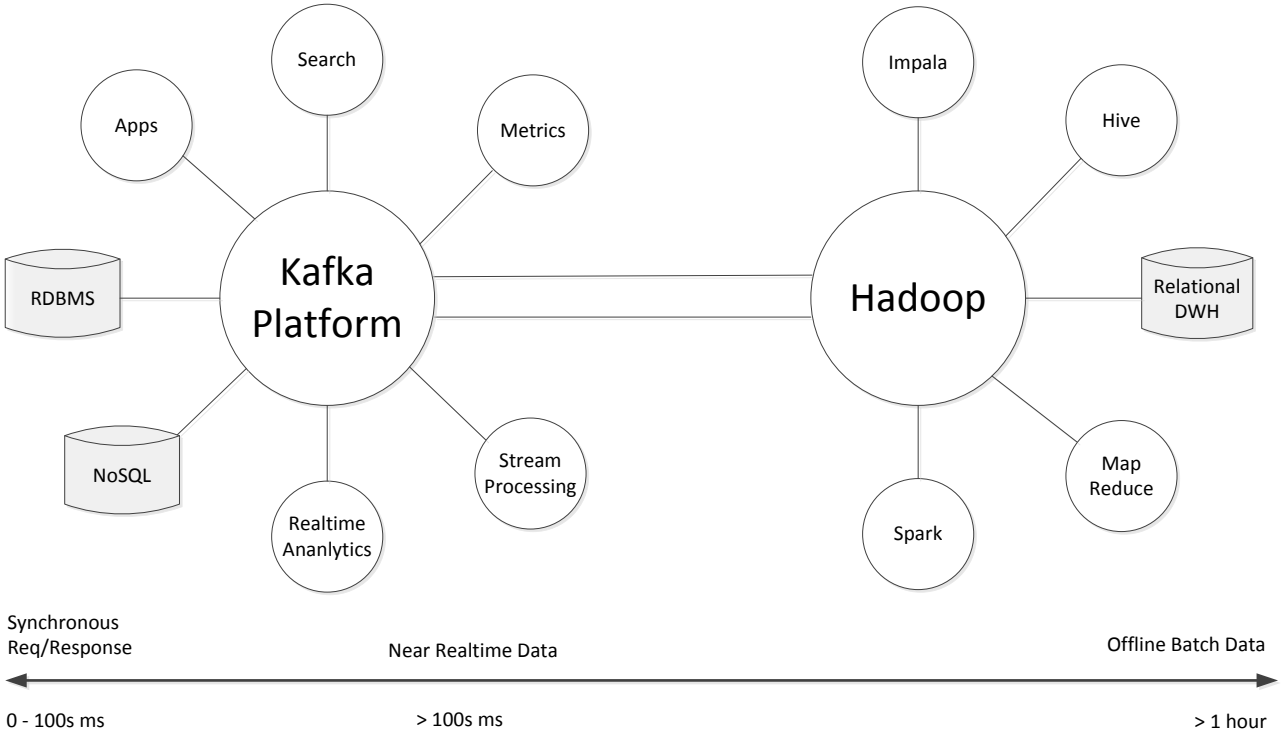


图 Kafka作为数据交换枢纽

7.5.1 Kafka简介

•Broker
Kafka集群包含一个或多个服务器，这种服务器被称为broker

•Topic
每条发布到Kafka集群的消息都有一个类别，这个类别被称为Topic。（物理上不同Topic的消息分开存储，逻辑上一个Topic的消息虽然保存于一个或多个broker上，但用户只需指定消息的Topic即可生产或消费数据而不必关心数据存于何处）

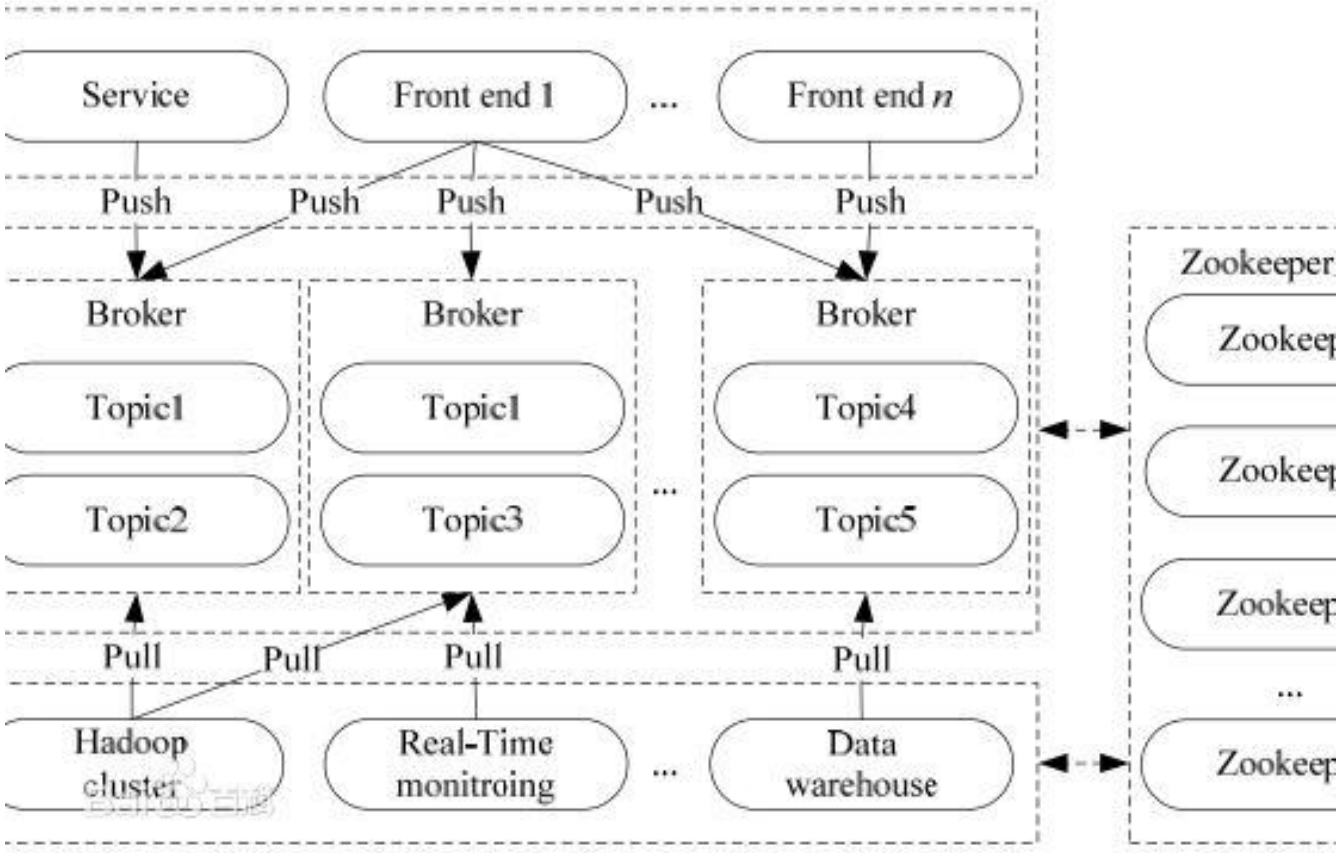
•Partition
Partition是物理上的概念，每个Topic包含一个或多个Partition.

•Producer
负责发布消息到Kafka broker

•Consumer
消息消费者，向Kafka broker读取消息的客户端。

•Consumer Group
每个Consumer属于一个特定的Consumer Group（可为每个Consumer指定group name，若不指定group name则属于默认的group）

7.5.1 Kafka简介


Service
Front end 1
...
Front end n
Push
Push
Push
Push
Push
Broker
Broker
Broker
Topic1
Topic1
Topic4
Topic2
Topic3
Topic5
...
Pull
Pull
Pull
Pull
Hadoop cluster
Real-Time monitroing
...
Data warehouse
Zookeeper
Zookeep
Zookeep
Zookeep
...
Zookeep

7.5.2 Kafka准备工作

1.安装Kafka
2.启动Kafka
3.测试Kafka是否正常工作

7.5.2 Kafka准备工作

1.安装Kafka

- 关于Kafka的安装方法，请参考厦门大学数据库实验室建设的高校大数据课程公共服务平台的技术博客文章《Kafka的安装和简单实例测试》
- http://dblab.xmu.edu.cn/blog/1096-2/
- 这里假设已经成功安装Kafka到“/usr/local/kafka”目录下

高校大数据课程

公 共 服 务 平 台

平台每年访问量超过100万次

7.5.2 Kafka准备工作

2.启动Kafka

说明：本课程下载的安装文件为Kafka_2.11-0.10.2.0.tgz，前面的2.11就是该Kafka所支持的Scala版本号，后面的0.10.2.0是Kafka自身的版本号

打开一个终端，输入下面命令启动Zookeeper服务：

```
$ cd /usr/local/kafka
$ ./bin/zookeeper-server-start.sh
config/zookeeper.properties
```

千万不要关闭这个终端窗口，一旦关闭，Zookeeper服务就停止了

7.5.2 Kafka准备工作

打开第二个终端，然后输入下面命令启动Kafka服务：

```
$ cd /usr/local/kafka
$ bin/kafka-server-start.sh config/server.properties
```

千万不要关闭这个终端窗口，一旦关闭，Kafka服务就停止了

7.5.2 Kafka准备工作

3.测试Kafka是否正常工作

再打开第三个终端，然后输入下面命令创建一个自定义名称为“wordsendertest”的Topic：

```
$ cd  /usr/local/kafka
$ ./bin/kafka-topics.sh  --create  --zookeeper  localhost:2181  \
>--replication-factor  1  --partitions  1  --topic  wordsendertest
#可以用list列出所有创建的Topic，验证是否创建成功
$ ./bin/kafka-topics.sh  --list  --zookeeper  localhost:2181
```

7.5.2 Kafka准备工作

下面用生产者（Producer）来产生一些数据，请在当前终端内继续输入下面命令：

```
$ ./bin/kafka-console-producer.sh  --broker-list  localhost:9092 \
>  --topic  wordsendertest
```

上面命令执行后，就可以在当前终端内用键盘输入一些英文单词，比如可以输入：

hello hadoop
hello spark

7.5.2 Kafka准备工作

现在可以启动一个消费者，来查看刚才生产者产生的数据。请另外打开第四个终端，输入下面命令：

```
$ cd /usr/local/kafka
$./bin/kafka-console-consumer.sh  --zookeeper  localhost:2181  \
> --topic  wordsendertest  --from-beginning
```

可以看到，屏幕上会显示出如下结果，也就是刚才在另外一个终端里面输入的内容：

hello hadoop
hello spark

7.5.3 Spark准备工作

1.添加相关jar包
2.启动spark-shell

7.5.3 Spark准备工作

1.添加相关jar包

Kafka和Flume等高级输入源，需要依赖独立的库（jar文件）

在spark-shell中执行下面import语句进行测试：

```
scala> import org.apache.spark.streaming.kafka._
<console>:25: error: object kafka is not a member of
package org.apache.spark.streaming
       import org.apache.spark.streaming.kafka._
                                          ^
```

对于Spark2.1.0版本，如果要使用Kafka，则需要下载spark-streaming-kafka-0-8_2.11相关jar包，下载地址：

http://mvnrepository.com/artifact/org.apache.spark/spark-streaming-kafka-0-8_2.11/2.1.0

7.5.3 Spark准备工作

把jar文件复制到Spark目录的jars目录下

```
$ cd /usr/local/spark/jars
$ mkdir kafka
$ cd ~
$ cd 下载
$ cp ./spark-streaming-kafka-0-8_2.11-2.1.0.jar
/usr/local/spark/jars/kafka
```

继续把Kafka安装目录的libs目录下的所有jar文件复制到“/usr/local/spark/jars/kafka”目录下，请在终端中执行下面命令：

```
$ cd /usr/local/kafka/libs
$ cp ./* /usr/local/spark/jars/kafka
```

2.启动spark-shell

执行如下命令启动spark-shell：

```
$ cd  /usr/local/spark
$ ./bin/spark-shell  \
>--jars /usr/local/spark/jars/*:/usr/local/spark/jars/kafka/*
```

启动成功后，再次执行如下命令：

```
scala> import  org.apache.spark.streaming.kafka._
//会显示下面信息
import org.apache.spark.streaming.kafka._
```

7.5.4 编写Spark Streaming程序使用Kafka数据源

1.编写生产者（producer）程序
2.编写消费者（consumer）程序
3.编写日志格式设置程序
4.编译打包程序
5.运行程序

7.5.4 编写Spark Streaming程序使用Kafka数据源

1.编写生产者（producer）程序

编写KafkaWordProducer程序。执行命令创建代码目录：

```
$ cd /usr/local/spark/mycode
$ mkdir kafka
$ cd kafka
$ mkdir -p src/main/scala
$ cd src/main/scala
$ vim KafkaWordProducer.scala
```

在KafkaWordProducer.scala中输入以下代码：

```
package org.apache.spark.examples.streaming
import java.util.HashMap
import org.apache.kafka.clients.producer.{KafkaProducer, ProducerConfig,
ProducerRecord}
import org.apache.spark.SparkConf
import org.apache.spark.streaming._
import org.apache.spark.streaming.kafka._
```

剩余代码见下一页

7.5.4 编写Spark Streaming程序使用Kafka数据源

```
object KafkaWordProducer {
  def main(args: Array[String]) {
    if (args.length < 4) {
      System.err.println("Usage: KafkaWordCountProducer <metadataBrokerList>
<topic> " +
        "<messagesPerSec> <wordsPerMessage>")
      System.exit(1)
    }
    val Array(brokers, topic, messagesPerSec, wordsPerMessage) = args
    // Zookeeper connection properties
    val props = new HashMap[String, Object]()
    props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, brokers)
    props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,
      "org.apache.kafka.common.serialization.StringSerializer")
    props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,
      "org.apache.kafka.common.serialization.StringSerializer")
    val producer = new KafkaProducer[String, String](props)
```

剩余代码见下一页

7.5.4 编写Spark Streaming程序使用Kafka数据源

```
// Send some messages
    while(true) {
      (1 to messagesPerSec.toInt).foreach { messageNum =>
        val str = (1 to wordsPerMessage.toInt).map(x =>
scala.util.Random.nextInt(10).toString)
          .mkString(" ")
                print(str)
                println()
        val message = new ProducerRecord[String, String](topic, null, str)
        producer.send(message)
      }
     Thread.sleep(1000)
    }
  }
}
```

7.5.4 编写Spark Streaming程序使用Kafka数据源

2.编写消费者（consumer）程序

继续在当前目录下创建KafkaWordCount.scala代码文件，它会把KafkaWordProducer发送过来的单词进行词频统计，代码内容如下：

```
package org.apache.spark.examples.streaming
import org.apache.spark._
import org.apache.spark.SparkConf
import org.apache.spark.streaming._
import org.apache.spark.streaming.kafka._
import org.apache.spark.streaming.StreamingContext._
import org.apache.spark.streaming.kafka.KafkaUtils
```

剩余代码见下一页

7.5.4 编写Spark Streaming程序使用Kafka数据源

```
object KafkaWordCount{
def main(args:Array[String]){
StreamingExamples.setStreamingLogLevels()
val sc = new
SparkConf().setAppName("KafkaWordCount").setMaster("local[2]")
val ssc = new StreamingContext(sc,Seconds(10))
ssc.checkpoint("file:///usr/local/spark/mycode/kafka/checkpoint") //设置检查点，如果存放在HDFS上面，则写成类似ssc.checkpoint("/user/hadoop/checkpoint")这种形式，但是，要启动hadoop
val zkQuorum = "localhost:2181" //Zookeeper服务器地址
val group = "1"  //topic所在的group，可以设置为自己想要的名称，比如不用1，而是val group = "test-consumer-group“
```

剩余代码见下一页

7.5.4 编写Spark Streaming程序使用Kafka数据源

```
val topics = "wordsender"  //topics的名称
val numThreads = 1  //每个topic的分区数
val topicMap =topics.split(",").map((_,numThreads.toInt)).toMap
val lineMap = KafkaUtils.createStream(ssc,zkQuorum,group,topicMap)
val lines = lineMap.map(_._2)
val words = lines.flatMap(_.split(" "))
val pair = words.map(x => (x,1))
val wordCounts = pair.reduceByKeyAndWindow(_ + _,_ -
_,Minutes(2),Seconds(10),2) //这行代码的含义在下一节的窗口转换操作中会有介绍
wordCounts.print
ssc.start
ssc.awaitTermination
}
}
```

7.5.4 编写Spark Streaming程序使用Kafka数据源

3.编写日志格式设置程序

继续在当前目录下创建StreamingExamples.scala代码文件，，用于设置log4j：

```
package org.apache.spark.examples.streaming
import org.apache.spark.Logging
import org.apache.log4j.{Level, Logger}
/** Utility functions for Spark Streaming examples. */
object StreamingExamples extends Logging {
  /** Set reasonable logging levels for streaming if the user has not configured log4j.
*/
  def setStreamingLogLevels() {
    val log4jInitialized = Logger.getRootLogger.getAllAppenders.hasMoreElements
    if (!log4jInitialized) {
      // We first log something to initialize Spark's default logging, then we override the
      // logging level.
      logInfo("Setting log level to [WARN] for streaming example." +
        " To override add a custom log4j.properties to the classpath.")
      Logger.getRootLogger.setLevel(Level.WARN)
    }
  }
}
```

7.5.4 编写Spark Streaming程序使用Kafka数据源

4.编译打包程序

创建simple.sbt文件：

```
$ cd /usr/local/spark/mycode/kafka/
$ vim simple.sbt
```

在simple.sbt中输入以下代码：

```
name := "Simple Project"
version := "1.0"
scalaVersion := "2.11.8"
libraryDependencies += "org.apache.spark" %% "spark-core" % "2.1.0"
libraryDependencies += "org.apache.spark" % "spark-streaming_2.11" % "2.1.0"
libraryDependencies += "org.apache.spark" % "spark-streaming-kafka-0-8_2.11" % "2.1.0"
```

进行打包编译：

```
$ cd /usr/local/spark/mycode/kafka/
$ /usr/local/sbt/sbt package
```

5.运行程序

打开一个终端，执行如下命令，运行“KafkaWordProducer”程序，生成一些单词（是一堆整数形式的单词）：

```
$ cd /usr/local/spark
$ /usr/local/spark/bin/spark-submit  \
>--driver-class-path /usr/local/spark/jars/*:/usr/local/spark/jars/kafka/*  \
>--class "org.apache.spark.examples.streaming.KafkaWordProducer"  \
>/usr/local/spark/mycode/kafka/target/scala-2.11/simple-project_2.11-1.0.jar  \
> localhost:9092 wordsender 3 5
```

7.5.4 编写Spark Streaming程序使用Kafka数据源

执行上面命令后，屏幕上会不断滚动出现新的单词，如下：

```
75073
28213
01292
80909
90068
66165
83677
......
```

不要关闭这个终端窗口，就让它一直不断发送单词

7.5.4 编写Spark Streaming程序使用Kafka数据源

请新打开一个终端，执行下面命令，运行KafkaWordCount程序，执行词频统计：

```
$ cd /usr/local/spark
$ /usr/local/spark/bin/spark-submit   \
>--driver-class-path /usr/local/spark/jars/*:/usr/local/spark/jars/kafka/* \
>--class "org.apache.spark.examples.streaming.KafkaWordCount"   \
>/usr/local/spark/mycode/kafka/target/scala-2.11/simple-project_2.11-1.0.jar
```

运行上面命令以后，就启动了词频统计功能，屏幕上就会显示如下类似信息：

```
-------------------------------------------
Time: 1488156500000 ms
-------------------------------------------
(4,5)
(8,12)
(6,14)
(0,19)
(2,11)
(7,20)
(5,10)
(9,9)
(3,9)
(1,11)
```

7.6 转换操作

7.6.1 DStream无状态转换操作
7.6.2 DStream有状态转换操作

7.6.1 DStream无状态转换操作

•map(func) ：对源DStream的每个元素，采用func函数进行转换，得到一个新的Dstream

•flatMap(func)： 与map相似，但是每个输入项可用被映射为0个或者多个输出项

•filter(func)： 返回一个新的DStream，仅包含源DStream中满足函数func的项

•repartition(numPartitions)： 通过创建更多或者更少的分区改变DStream的并行程度

•reduce(func)：利用函数func聚集源DStream中每个RDD的元素，返回一个包含单元素RDDs的新DStream

•count()：统计源DStream中每个RDD的元素数量

•union(otherStream)： 返回一个新的DStream，包含源DStream和其他DStream的元素

7.6.1 DStream无状态转换操作

•countByValue()：应用于元素类型为K的DStream上，返回一个（K，V）键值对类型的新DStream，每个键的值是在原DStream的每个RDD中的出现次数

•reduceByKey(func, [numTasks])：当在一个由(K,V)键值对组成的DStream上执行该操作时，返回一个新的由(K,V)键值对组成的DStream，每一个key的值均由给定的recuce函数（func）聚集起来

•join(otherStream, [numTasks])：当应用于两个DStream（一个包含（K,V）键值对,一个包含(K,W)键值对），返回一个包含(K, (V, W))键值对的新Dstream

•cogroup(otherStream, [numTasks])：当应用于两个DStream（一个包含（K,V）键值对,一个包含(K,W)键值对），返回一个包含(K, Seq[V], Seq[W])的元组

•transform(func)：通过对源DStream的每个RDD应用RDD-to-RDD函数，创建一个新的DStream。支持在新的DStream中做任何RDD操作

7.6.1 DStream无状态转换操作

无状态转换操作实例：
之前“套接字流”部分介绍的词频统计，就是采用无状态转换，每次统计，都是只统计当前批次到达的单词的词频，和之前批次无关，不会进行累计

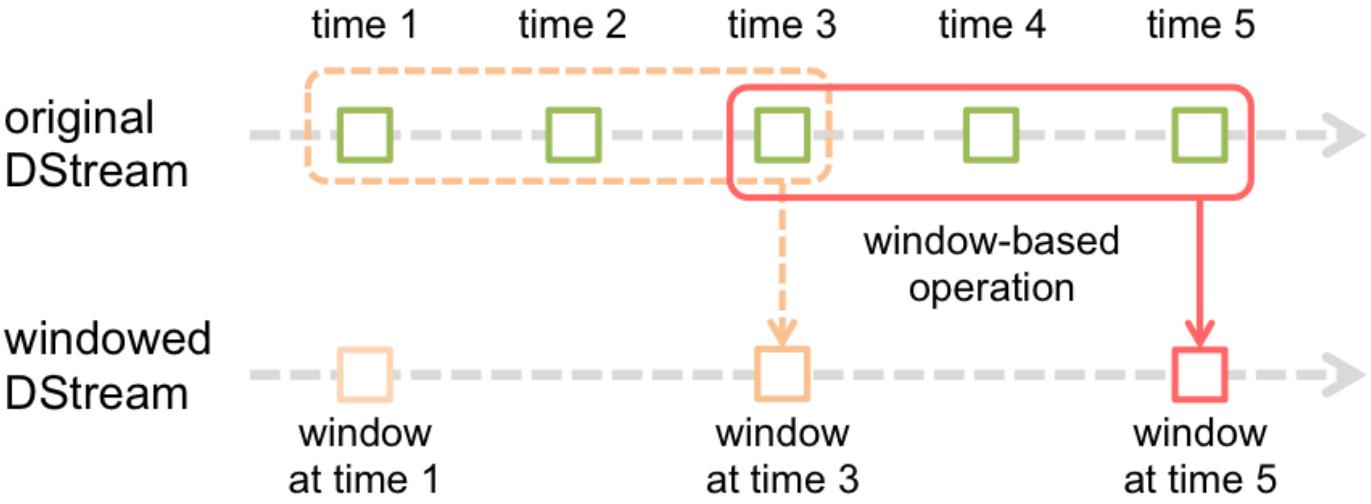
7.6.2 DStream有状态转换操作

1. 滑动窗口转换操作
2. updateStateByKey操作

7.6.2 DStream有状态转换操作

1. 滑动窗口转换操作

- 事先设定一个滑动窗口的长度（也就是窗口的持续时间）
- 设定滑动窗口的时间间隔（每隔多长时间执行一次计算），让窗口按照指定时间间隔在源DStream上滑动
- 每次窗口停放的位置上，都会有一部分Dstream（或者一部分RDD）被框入窗口内，形成一个小段的Dstream
- 可以启动对这个小段DStream的计算



7.6.2 DStream有状态转换操作

一些窗口转换操作的含义：

•window(windowLength, slideInterval) 基于源DStream产生的窗口化的批数据，计算得到一个新的Dstream

•countByWindow(windowLength, slideInterval) 返回流中元素的一个滑动窗口数

•reduceByWindow(func, windowLength, slideInterval) 返回一个单元素流。利用函数func聚集滑动时间间隔的流的元素创建这个单元素流。函数func必须满足结合律，从而可以支持并行计算

•reduceByKeyAndWindow(func, windowLength, slideInterval, [numTasks]) 应用到一个(K,V)键值对组成的DStream上时，会返回一个由(K,V)键值对组成的新的DStream。每一个key的值均由给定的reduce函数(func函数)进行聚合计算。注意：在默认情况下，这个算子利用了Spark默认的并发任务数去分组。可以通过numTasks参数的设置来指定不同的任务数

7.6.2 DStream有状态转换操作

一些窗口转换操作的含义：

•reduceByKeyAndWindow(func, invFunc, windowLength, slideInterval, [numTasks]) 更加高效的reduceByKeyAndWindow，每个窗口的reduce值，是基于先前窗口的reduce值进行增量计算得到的；它会对进入滑动窗口的新数据进行reduce操作，并对离开窗口的老数据进行“逆向reduce”操作。但是，只能用于“可逆reduce函数”，即那些reduce函数都有一个对应的“逆向reduce函数”（以InvFunc参数传入）

•countByValueAndWindow(windowLength, slideInterval, [numTasks]) 当应用到一个(K,V)键值对组成的DStream上，返回一个由(K,V)键值对组成的新的DStream。每个key的值都是它们在滑动窗口中出现的频率

7.6.2 DStream有状态转换操作

窗口转换操作实例：

在上一节的“Apache Kafka作为DStream数据源”内容中，已经使用了窗口转换操作，在KafkaWordCount.scala代码中，可以找到下面这一行：

```
val wordCounts =
pair.reduceByKeyAndWindow(_ + _,_ -
_,Minutes(2),Seconds(10),2)
```

7.6.2 DStream有状态转换操作

```
val wordCounts = pair.reduceByKeyAndWindow(_ + _,_ - _,Minutes(2),Seconds(10),2)
```

实现增量计算

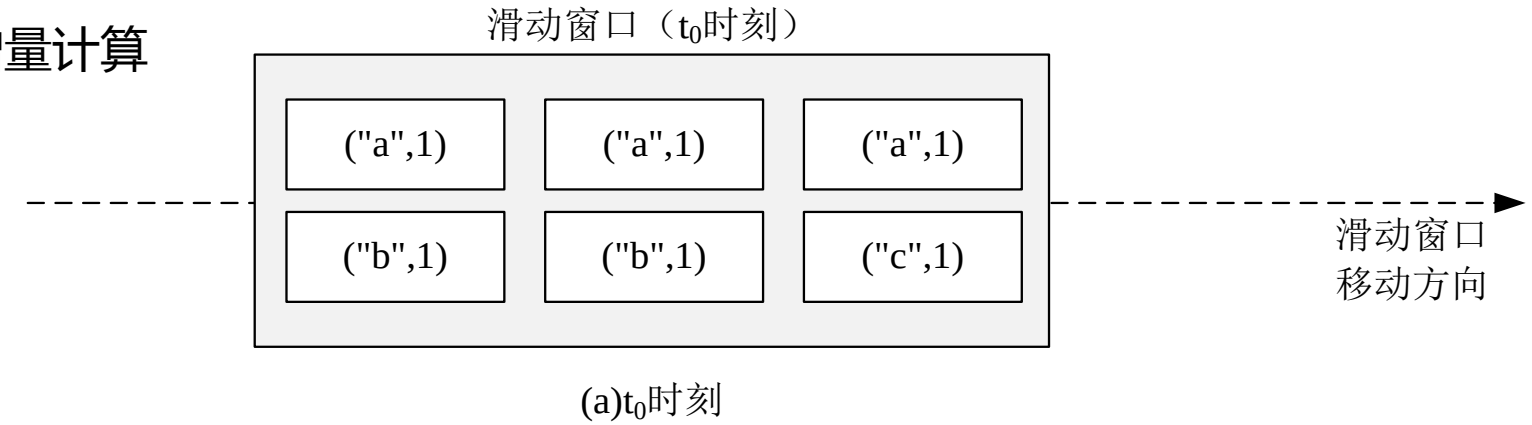


图 reduceByKeyAndWindow操作过程示意图

7.6.2 DStream有状态转换操作

2. updateStateByKey操作

需要在跨批次之间维护状态时，就必须使用updateStateByKey操作

词频统计实例：

对于有状态转换操作而言，本批次的词频统计，会在之前批次的词频统计结果的基础上进行不断累加，所以，最终统计得到的词频，是所有批次的单词的总的词频统计结果

新建一个NetworkWordCountStateful.scala代码文件，输入以下代码：

```
package org.apache.spark.examples.streaming
import org.apache.spark._
import org.apache.spark.streaming._
import org.apache.spark.storage.StorageLevel
```

剩余代码见下一页

7.6.2 DStream有状态转换操作

```
object NetworkWordCountStateful {
  def main(args: Array[String]) {
    //定义状态更新函数
    val updateFunc = (values: Seq[Int], state: Option[Int]) => {
      val currentCount = values.foldLeft(0)(_ + _)
      val previousCount = state.getOrElse(0)
      Some(currentCount + previousCount)
    }
        StreamingExamples.setStreamingLogLevels()  //设置log4j
日志级别
```

剩余代码见下一页

7.6.2 DStream有状态转换操作

```
val conf = new
SparkConf().setMaster("local[2]").setAppName("NetworkWordCountStateful
")
    val sc = new StreamingContext(conf, Seconds(5))
    sc.checkpoint("file:///usr/local/spark/mycode/streaming/stateful/")    //设置检查点，检查点具有容错机制
    val lines = sc.socketTextStream("localhost", 9999)
    val words = lines.flatMap(_.split(" "))
    val wordDstream = words.map(x => (x, 1))
    val stateDstream = wordDstream.updateStateByKey[Int](updateFunc)
    stateDstream.print()
    sc.start()
    sc.awaitTermination()
  }
}
```

7.6.2 DStream有状态转换操作

新建一个StreamingExamples.scala文件，用于设置log4j日志级别，代码如下：

```
package org.apache.spark.examples.streaming
import org.apache.spark.Logging
import org.apache.log4j.{Level, Logger}
/** Utility functions for Spark Streaming examples. */
object StreamingExamples extends Logging {
  /** Set reasonable logging levels for streaming if the user has not configured log4j. */
  def setStreamingLogLevels() {
    val log4jInitialized = Logger.getRootLogger.getAllAppenders.hasMoreElements
    if (!log4jInitialized) {
      // We first log something to initialize Spark's default logging, then we override the
      // logging level.
      logInfo("Setting log level to [WARN] for streaming example." +
        " To override add a custom log4j.properties to the classpath.")
      Logger.getRootLogger.setLevel(Level.WARN)
    }
  }
}
```

7.6.2 DStream有状态转换操作

创建simple.sbt文件：

```
name := "Simple Project"
version := "1.0"
scalaVersion := "2.11.8"
libraryDependencies += "org.apache.spark" %% "spark-streaming" % "2.1.0"
```

执行sbt打包编译

输入以下命令启动这个程序：

```
$ /usr/local/spark/bin/spark-submit   \
>--class "org.apache.spark.examples.streaming.NetworkWordCountStateful"   \
>/usr/local/spark/mycode/streaming/stateful/target/scala-2.11/simple-project_2.11-1.0.jar
```

执行上面命令后，就进入了监听状态（称为监听窗口）

7.6.2 DStream有状态转换操作

新打开一个窗口作为nc窗口，启动nc程序：

```
$ nc -lk 9999
//在这个窗口中手动输入一些单词
hadoop
spark
hadoop
spark
hadoop
spark
```

切换到刚才的监听窗口，会发现，已经输出了词频统计信息：

```
-------------------------------------------
Time: 1479890485000 ms
-------------------------------------------
(spark,1)
(hadoop,1)
-------------------------------------------
Time: 1479890490000 ms
-------------------------------------------
(spark,2)
(hadoop,3)
```

7.7 输出操作

在Spark应用中，外部系统经常需要使用到Spark DStream处理后的数据，因此，需要采用输出操作把DStream的数据输出到数据库或者文件系统中

7.7.1 把DStream输出到文本文件中
7.7.2 把DStream写入到MySQL数据库中

7.7.1 把DStream输出到文本文件中

请在NetworkWordCountStateful.scala代码文件中输入以下内容：

```
package org.apache.spark.examples.streaming
import org.apache.spark._
import org.apache.spark.streaming._
import org.apache.spark.storage.StorageLevel
object NetworkWordCountStateful {
  def main(args: Array[String]) {
    //定义状态更新函数
    val updateFunc = (values: Seq[Int], state: Option[Int]) => {
      val currentCount = values.foldLeft(0)(_ + _)
      val previousCount = state.getOrElse(0)
      Some(currentCount + previousCount)
    }
        StreamingExamples.setStreamingLogLevels()  //设置log4j日志级别
```

剩余代码见下一页

7.7.1 把DStream输出到文本文件中

```
val conf = new
SparkConf().setMaster("local[2]").setAppName("NetworkWordCountStateful")
    val sc = new StreamingContext(conf, Seconds(5))
    sc.checkpoint("file:///usr/local/spark/mycode/streaming/dstreamoutput/")    //设置
检查点，检查点具有容错机制
    val lines = sc.socketTextStream("localhost", 9999)
    val words = lines.flatMap(_.split(" "))
    val wordDstream = words.map(x => (x, 1))
    val stateDstream = wordDstream.updateStateByKey[Int](updateFunc)
    stateDstream.print()
        //下面是新增的语句，把DStream保存到文本文件中

stateDstream.saveAsTextFiles("file:///usr/local/spark/mycode/streaming/dstreamou
tput/output.txt")
    sc.start()
    sc.awaitTermination()
  }
}
```

7.7.1 把DStream输出到文本文件中

sbt打包编译后，使用如下命令运行程序：

```
$ /usr/local/spark/bin/spark-submit --class
"org.apache.spark.examples.streaming.NetworkWordCountStateful"
/usr/local/spark/mycode/streaming/dstreamoutput/target/scala-
2.11/simple-project_2.11-1.0.jar
```

程序运行以后，屏幕上就会显示类似下面的程序运行信息：

```
-------------------------------------------
Time: 1479890485000 ms
-------------------------------------------


-------------------------------------------
Time: 1479890490000 ms
-------------------------------------------
```

7.7.1 把DStream输出到文本文件中

打开另外一个终端，作为单词产生的源头，提供给NetworkWordCountStateful程序进行词频统计：

```
$ nc -lk 9999
//请手动输入一些单词，可以随便输入，比如下面是笔者输入的单词
hadoop
spark
hadoop
spark
hadoop
spark
```

7.7.1 把DStream输出到文本文件中

运行NetworkWordCountStateful程序的监听窗口，就可以看到类似下面的词频统计结果：

```
-------------------------------------------
Time: 1479890485000 ms
-------------------------------------------
(spark,1)
(hadoop,1)


-------------------------------------------
Time: 1479890490000 ms
-------------------------------------------
(spark,2)
(hadoop,3)
```

7.7.1 把DStream输出到文本文件中

这些词频结果被成功地输出到
“/usr/local/spark/mycode/streaming/dstreamoutput/output.txt”文件中

```
$ cd /usr/local/spark/mycode/streaming/dstreamoutput/
$ ls
```

可以发现，在这个目录下，生成了很多文本文件，如下：

```
output.txt-1479951955000
output.txt-1479951960000
output.txt-1479951965000
output.txt-1479951970000
output.txt-1479951975000
output.txt-1479951980000
output.txt-1479951985000
```

output.txt的命名看起来像一个文件，但是，实际上，spark会生成名称为output.txt的目录，而不是文件

7.7.2 把DStream写入到MySQL数据库中

启动MySQL数据库，并完成数据库和表的创建：

```
$ service mysql start
$ mysql -u root -p
$ #屏幕会提示你输入密码
```

在此前已经创建好的“spark”数据库中创建一个名称为“wordcount”的表：

```
mysql> use spark
mysql> create table wordcount (word char(20), count int(4));
```

7.7.2 把DStream写入到MySQL数据库中

在NetworkWordCountStateful.scala文件中加入下面代码：

```
package org.apache.spark.examples.streaming
import java.sql.{PreparedStatement, Connection,
DriverManager}
import java.util.concurrent.atomic.AtomicInteger
import org.apache.spark.SparkConf
import org.apache.spark.streaming.{Seconds,
StreamingContext}
import org.apache.spark.streaming.StreamingContext._
import org.apache.spark.storage.StorageLevel
```

剩余代码见下一页

7.7.2 把DStream写入到MySQL数据库中

```
object NetworkWordCountStateful {
  def main(args: Array[String]) {
    //定义状态更新函数
    val updateFunc = (values: Seq[Int], state: Option[Int]) => {
      val currentCount = values.foldLeft(0)(_ + _)
      val previousCount = state.getOrElse(0)
      Some(currentCount + previousCount)
    }
    StreamingExamples.setStreamingLogLevels()  //设置log4j
日志级别
```

剩余代码见下一页

7.7.2 把DStream写入到MySQL数据库中

```
val conf = new
SparkConf().setMaster("local[2]").setAppName("NetworkWor
dCountStateful")
    val sc = new StreamingContext(conf, Seconds(5))

sc.checkpoint("file:///usr/local/spark/mycode/streaming/dstrea
moutput/")   //设置检查点，检查点具有容错机制
    val lines = sc.socketTextStream("localhost", 9999)
    val words = lines.flatMap(_.split(" "))
    val wordDstream = words.map(x => (x, 1))
    val stateDstream =
wordDstream.updateStateByKey[Int](updateFunc)
    stateDstream.print()
```

剩余代码见下一页

7.7.2 把DStream写入到MySQL数据库中

```
//下面是新增的语句，把DStream保存到MySQL数据库中
    stateDstream.foreachRDD(rdd => {
     //内部函数
     def func(records: Iterator[(String,Int)]) {
       var conn: Connection = null
       var stmt: PreparedStatement = null
       try {
         val url = "jdbc:mysql://localhost:3306/spark"
         val user = "root"
         val password = "hadoop"  //数据库密码是hadoop
         conn = DriverManager.getConnection(url, user, password)
         records.foreach(p => {
           val sql = "insert into wordcount(word,count) values (?,?)"
           stmt = conn.prepareStatement(sql);
           stmt.setString(1, p._1.trim)
           stmt.setInt(2,p._2.toInt)
           stmt.executeUpdate()
         })
```

剩余代码见下一页

7.7.2 把DStream写入到MySQL数据库中

```
} catch {
        case e: Exception => e.printStackTrace()
      } finally {
        if (stmt != null) {
          stmt.close()
        }
        if (conn != null) {
          conn.close()
        }
      }
    }
    val repartitionedRDD = rdd.repartition(3)
    repartitionedRDD.foreachPartition(func)
  })
  sc.start()
  sc.awaitTermination()
 }
}
```

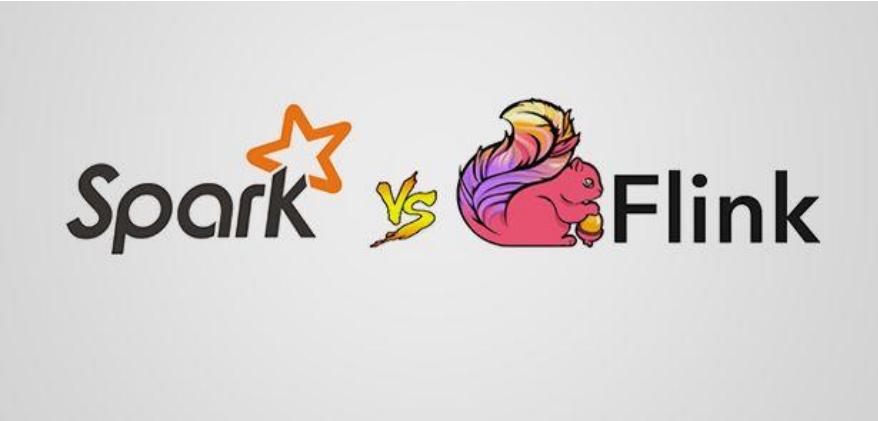
7.8 Structured Streaming

7.8.1 Spark流计算组件的演进
7.8.2 Structured Streaming的基本原理
7.8.3 为什么设计Structured Streaming

7.8.1 Spark流计算组件的演进

Spark流计算组件的演进

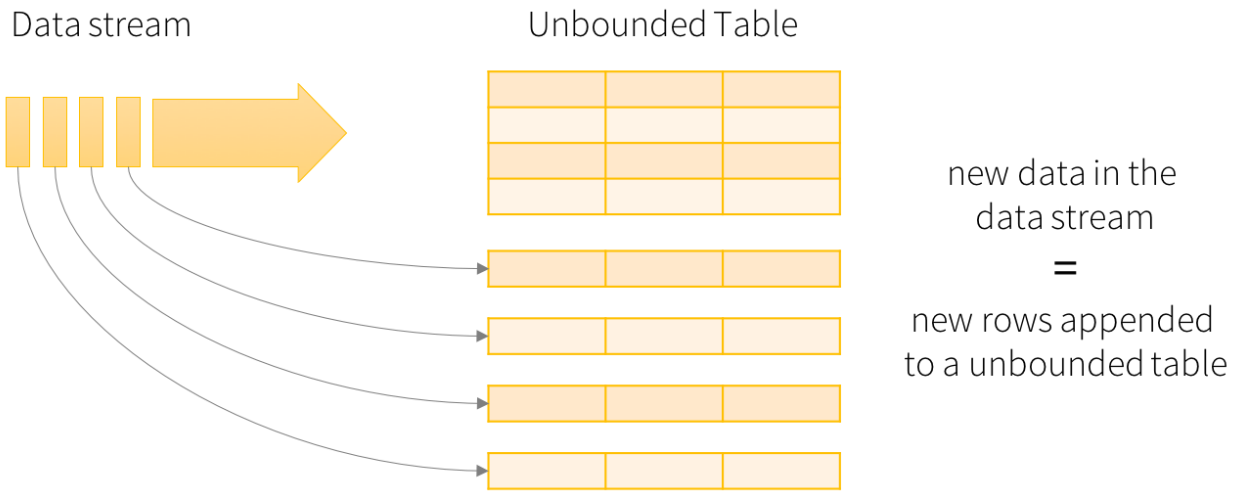
- Spark2.0之前，使用Spark Streaming，基于RDD的数据抽象
- Spark2.0之后，新增了Structured Streaming，基于DataFrame的数据抽象，采用“微批次模式”
- Structured Streaming在Spark2.0中只是测试版本，2.2版本才正式发布
- 2018年2月28日，Spark2.3重磅发布，新版本Structured Streaming引入了持续流式处理模式，可将流处理延迟降低至毫秒级别，与Flink一较高下



2018年Spark2.3版本引入持续流式处理模型与Flink一较高下

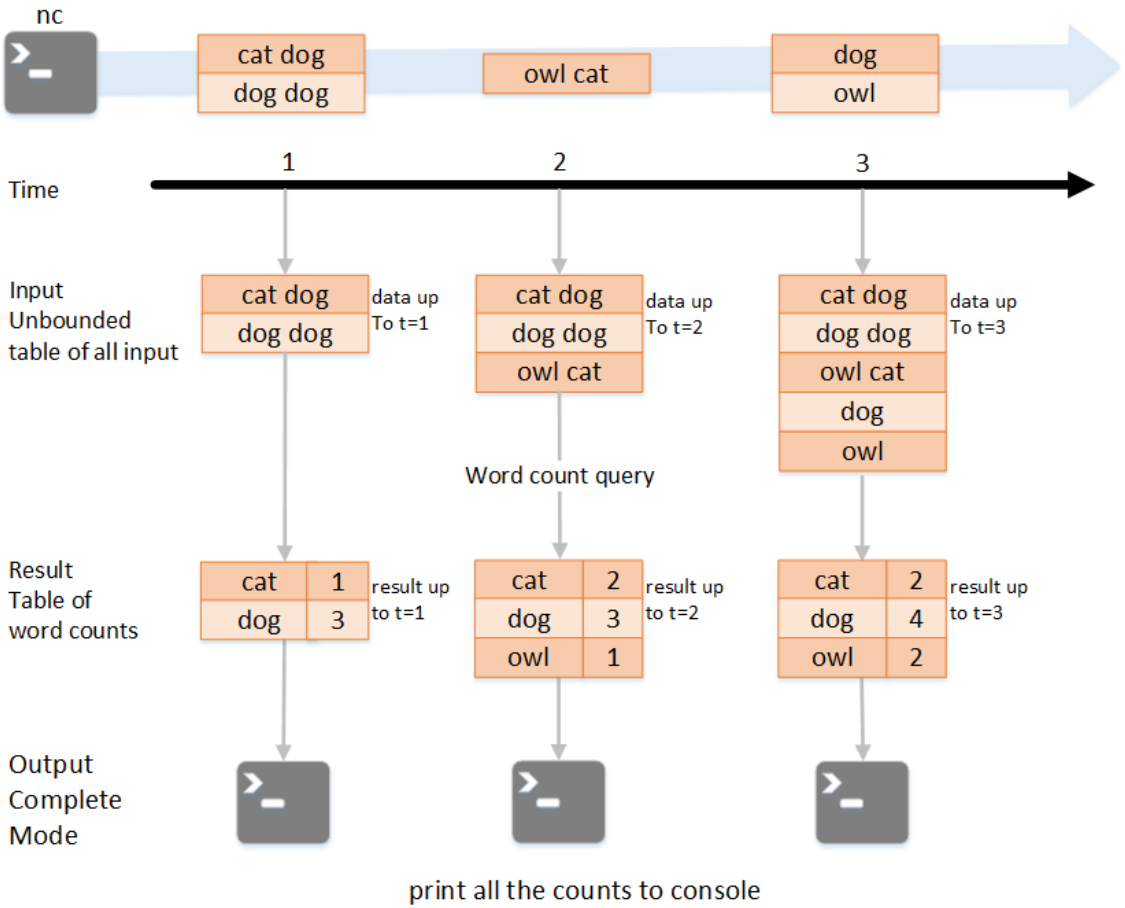
7.8.2 Structured Streaming的基本原理

- 把不断输入的流式数据加载为内存中一张没有边界的数据库表，每一条新来的数据都会作为一行数据新增到这张表中
- 每一条查询的操作都会产生一个结果集-Result Table。每一个触发间隔（比如说1秒），当新的数据新增到表中，都会最终更新Result Table



Data stream as an unbounded table

7.8.2 Structured Streaming的基本原理



Model of the Quick Example

Structured Streaming编程模型示例

7.8.3 为什么设计Structured Streaming

问：此前已经有了Spark Streaming，为什么还要设计Structured Streaming？

答：
第一，重新抽象了流式计算
第二，易于实现数据的exactly-once。2.0之前的Spark Streaming只能做到at-least once，框架层次很难帮你做到exactly-once。 现在通过重新设计流式计算框架，使得实现exactly-once 变得容易了

附录A：主讲教师林子雨简介

主讲教师：林子雨

单位：厦门大学计算机科学系
E-mail: ziyulin@xmu.edu.cn
个人网页：http://www.cs.xmu.edu.cn/linziyu
数据库实验室网站：http://dblab.xmu.edu.cn

扫一扫访问个人主页

林子雨，男，1978年出生，博士（毕业于北京大学），现为厦门大学计算机科学系助理教授（讲师），曾任厦门大学信息科学与技术学院院长助理、晋江市发展和改革局副局长。中国计算机学会数据库专业委员会委员，中国计算机学会信息系统专业委员会委员。国内高校首个“数字教师”提出者和建设者，厦门大学数据库实验室负责人，厦门大学云计算与大数据研究中心主要建设者和骨干成员，2013年度和2017年度厦门大学教学类奖教金获得者，荣获2017年福建省精品在线开放课程和2017年厦门大学高等教育成果二等奖。主要研究方向为数据库、数据仓库、数据挖掘、大数据、云计算和物联网，并以第一作者身份在《软件学报》《计算机学报》和《计算机研究与发展》等国家重点期刊以及国际学术会议上发表多篇学术论文。作为项目负责人主持的科研项目包括1项国家自然科学青年基金项目(No.61303004)、1项福建省自然科学青年基金项目(No.2013J05099)和1项中央高校基本科研业务费项目(No.2011121049)，主持的教改课题包括1项2016年福建省教改课题和1项2016年教育部产学协作育人项目，同时，作为课题负责人完成了国家发改委城市信息化重大课题、国家物联网重大应用示范工程区域试点泉州市工作方案、2015泉州市互联网经济调研等课题。中国高校首个“数字教师”提出者和建设者，2009年至今，“数字教师”大平台累计向网络免费发布超过500万字高价值的研究和教学资料，累计网络访问量超过500万次。打造了中国高校大数据教学知名品牌，编著出版了中国高校第一本系统介绍大数据知识的专业教材《大数据技术原理与应用》，并成为京东、当当网等网店畅销书籍；建设了国内高校首个大数据课程公共服务平台，为教师教学和学生学习大数据课程提供全方位、一站式服务，年访问量超过100万次。

附录B：大数据学习路线图

大数据学习路线图访问地址：http://dblab.xmu.edu.cn/post/10164/

附录C：《大数据技术原理与应用》教材

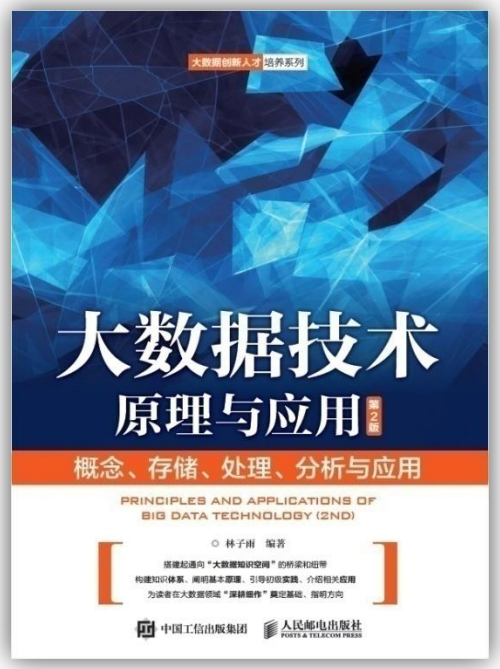
《大数据技术原理与应用——概念、存储、处理、分析与应用（第2版）》，由厦门大学计算机科学系林子雨博士编著，是国内高校第一本系统介绍大数据知识的专业教材。人民邮电出版社 ISBN:978-7-115-44330-4 定价：49.80元

全书共有15章，系统地论述了大数据的基本概念、大数据处理架构Hadoop、分布式文件系统HDFS、分布式数据 库HBase、NoSQL数据库、云数据库、分布式并行编程模型MapReduce、Spark、流计算、图计算、数据可视化以及大数据在互联网、生物医学和物流等各个领域的应用。在Hadoop、HDFS、HBase和MapReduce等重要章节，安排了入门级的实践操作，让读者更好地学习和掌握大数据关键技术。

本书可以作为高等院校计算机专业、信息管理等相关专业的大数据课程教材，也可供相关技术人员参考、学习、培训之用。

欢迎访问《大数据技术原理与应用——概念、存储、处理、分析与应用》教材官方网站：
http://dblab.xmu.edu.cn/post/bigdata

扫一扫访问教材官网



附录D：《大数据基础编程、实验和案例教程》

本书是与《大数据技术原理与应用（第2版）》教材配套的唯一指定实验指导书

大数据教材

1+1黄金组合

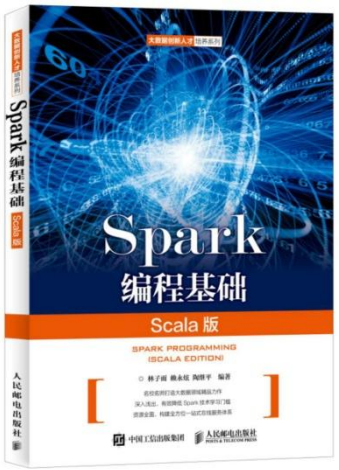
厦门大学林子雨编著

配套实验指导书

清华大学出版社 ISBN:978-7-302-47209-4 定价：59元

- 步步引导，循序渐进，详尽的安装指南为顺利搭建大数据实验环境铺平道路
- 深入浅出，去粗取精，丰富的代码实例帮助快速掌握大数据基础编程方法
- 精心设计，巧妙融合，五套大数据实验题目促进理论与编程知识的消化和吸收
- 结合理论，联系实际，大数据课程综合实验案例精彩呈现大数据分析全流程

附录E：《Spark编程基础（Scala版）》



《Spark编程基础（Scala版）》

厦门大学 林子雨，赖永炫，陶继平 编著

披荆斩棘，在大数据丛林中开辟学习捷径
填沟削坎，为快速学习Spark技术铺平道路
深入浅出，有效降低Spark技术学习门槛
资源全面，构建全方位一站式在线服务体系

人民邮电出版社出版发行，ISBN:978-7-115-48816-9
教材官网：http://dblab.xmu.edu.cn/post/spark/

本书以Scala作为开发Spark应用程序的编程语言，系统介绍了Spark编程的基础知识。全书共8章，内容包括大数据技术概述、Scala语言基础、Spark的设计与运行原理、Spark环境搭建和使用方法、RDD编程、Spark SQL、Spark Streaming、Spark MLlib等。本书每个章节都安排了入门级的编程实践操作，以便读者更好地学习和掌握Spark编程方法。本书官网免费提供了全套的在线教学资源，包括讲义PPT、习题、源代码、软件、数据集、授课视频、上机实验指南等。

附录F：高校大数据课程公共服务平台

高校大数据课程

公 共 服 务 平 台

http://dblab.xmu.edu.cn/post/bigdata-teaching-platform/

扫一扫访问平台主页

扫一扫观看3分钟FLASH动画宣传片

附录G：高校大数据实训课程系列案例教材

为了更好满足高校开设大数据实训课程的教材需求，厦门大学数据库实验室林子雨老师团队联合企业共同开发了《高校大数据实训课程系列案例》，目前已经完成开发的系列案例包括：

《基于协同过滤算法的电影推荐》
《电信用户行为分析》
《实时日志流处理分析》
《微博用户情感分析》
《互联网广告预测分析》
《网站日志处理分析》

部分教材书稿已经完成写作，将于2019年陆续出版发行，教材相关信息，敬请关注网页后续更新！http://dblab.xmu.edu.cn/post/shixunkecheng/

扫一扫访问大数据实训课程系列案例教材主页

Department of Computer Science, Xiamen University, 2018