# 

**Spark Streaming的分析与应用**

**需求规格说明书**

Version 1.4.0

小组成员：

陈少杰

姜鑫

蒲彦均

邹嘉欣

**版本变更记录**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 版本 | 变更时间 | 修改人 | 审核人 | 备注 |
| 1.0 | 2017/03/22 | 邹嘉欣、陈少杰 | 姜鑫、蒲彦均 | 初稿 |
| 1.1 | 2017/03/28 | 邹嘉欣、陈少杰、姜鑫、蒲彦均 | 邹嘉欣、陈少杰、姜鑫、蒲彦均 | 二稿 |
| 1.2 | 2017/04/04 | 姜鑫 | 邹嘉欣、陈少杰、蒲彦均 | 修改了业务需求模型 |
| 1.2.1 | 2017/04/06 | 陈少杰 | 邹嘉欣、蒲彦均、姜鑫 | 修改了用例图，调整章节顺序 |
| 1.3.0 | 2017/04/20 | 姜鑫 | 邹嘉欣、蒲彦均、陈少杰 | 增加了设计实现部分的需求 |
| 1.3.1 | 2017/04/21 | 邹嘉欣 | 陈少杰、姜鑫、蒲彦均 | 增加性能指标 |
| 1.4.0 | 2017/04/23 | 姜鑫 | 陈少杰、邹嘉欣、蒲彦均 | 1.增加实现方案、难点、可行性等  2.针对刘超老师的批注做语言规范性等方面的调整（仅修改了原来姜鑫负责的章节） |

目录

[图片目录 iii](#_Toc480537403)

[表格目录 iv](#_Toc480537404)

[1. 前言 1](#_Toc480537405)

[1.1 目的 1](#_Toc480537406)

[1.2 系统概述 1](#_Toc480537407)

[1.3 文档概述 1](#_Toc480537408)

[1.4 术语和缩略语 2](#_Toc480537409)

[2. 引用文档 2](#_Toc480537410)

[3. 业务需求模型 3](#_Toc480537411)

[3.1社交媒体数据实时分析 3](#_Toc480537412)

[3.2网站流量统计 4](#_Toc480537413)

[4. 功能性需求 5](#_Toc480537414)

[4.1 数据的产生与输入 5](#_Toc480537415)

[4.1.1输入流接收器 5](#_Toc480537416)

[4.1.2 数据源输入流 6](#_Toc480537417)

[4.2 数据流抽象DStream 6](#_Toc480537418)

[4.2.1 DStream的离散化 6](#_Toc480537419)

[4.2.2 基本数据集抽象——RDD 7](#_Toc480537420)

[4.2.3 DStream的操作 7](#_Toc480537421)

[3.3 作业调度 10](#_Toc480537422)

[4.3.1 Spark应用之间的资源调度 10](#_Toc480537423)

[4.3.2 同一应用程序内的资源调度 11](#_Toc480537424)

[4.4 长时间容错 11](#_Toc480537425)

[4.4.1 数据导入导出模块长时间容错 12](#_Toc480537426)

[4.4.2 数据抽象模块及作业调度模块的长时间容错 12](#_Toc480537427)

[4.5 窗口支持 12](#_Toc480537428)

[4.6 用例图 13](#_Toc480537429)

[4.7 用例说明 13](#_Toc480537430)

[4.7.1 产生与输入数据 14](#_Toc480537431)

[4.7.2 数据流抽象 15](#_Toc480537432)

[4.7.3 系统容错 16](#_Toc480537433)

[4.7.4 作业调度 16](#_Toc480537434)

[5. 非功能性需求 17](#_Toc480537435)

[5.1 实时性 17](#_Toc480537436)

[5.2 扩展性与吞吐量 17](#_Toc480537437)

[5.3 持久化 17](#_Toc480537438)

[5.4性能调优 18](#_Toc480537439)

[5.4.1合理的批处理时间 18](#_Toc480537440)

[5.4.2 增加并行化 18](#_Toc480537441)

[5.4.3 及时清除过期数据 18](#_Toc480537442)

[5. 设计实现 18](#_Toc480537443)

[5.1工作目标 18](#_Toc480537444)

[5.2软件基本功能描述 18](#_Toc480537445)

[5.3技术路线 19](#_Toc480537446)

[6.运行环境 20](#_Toc480537447)

[6.1硬件环境 20](#_Toc480537448)

[6.2软件环境 20](#_Toc480537449)

[7.参考资料 20](#_Toc480537450)

图片目录

图 1 Spark生态系统 1

图 10 社交数据分析的数据流 4

图 2 DStream的离散化 6

图 3 DStream与RDD 7

图 4 Spark工作流程 8

图 5 用例图 13

图 6产生与输入数据RUCM模型 14

图 7数据流抽象RUCM模型 15

图 8系统容错RUCM模型 16

图 9作业调度RUCM模型 16

表格目录

[表1 术语和缩略语表 2](#_Toc479327579)

[表2 transformation方法表 8](#_Toc479327580)

[表3 action方法表 9](#_Toc479327581)

[表4 软件环境表 18](#_Toc479327582)

# 前言

## 目的

以Spark Streaming开源项目及相关资料为输入，分析软件设计需求，结合软件工程综合实验具体要求，输出软件需求规格说明书，作为设计开发的依据并指导后续的开发工作。

## 系统概述

Apache Spark是一个围绕速度、易用性和复杂分析构建的大数据处理框架。最初在2009年由加州大学伯克利分校的AMP-Lab开发，并于2010年成为Apache的开源项目之一。

Spark是一个类似于Hadoop的MapReduce的分布式计算框架，其核心是弹性分布式数据集，提供了比MapReduce更丰富的模型，可以快速在内存中对数据集进行多次迭代，以支持复杂的数据挖掘算法和图形计算算法。

Spark Streaming是建立在Spark上的实时计算框架，通过它提供的丰富的API、基于内存的高速执行引擎，用户可以结合流式、批处理和交互式查询应用，扩展了Spark处理大规模流式数据的能力。图1展示了Spark与其配套的Spark SQL, Spark Streaming, MLib和GraphX等上层模块构成的生态系统。



图 1 Spark生态系统

## 文档概述

文档用途：本文档主要是介绍Spark Streaming系统需求及规格说明。

主要内容：

* 以用例图、状态图的形式给出Spark Streaming系统功能需求的分解结构，并对用例模型中的参与者和用例进行详细的描述，其中主要包括软件系统的用例模型、系统的核心流程等；
* 使用 RUCM 模型对功能需求进行建模；
* 描述了与此次系统实施相关的硬件环境的一些要求；
* 描述了与此系统实施相关的软件环境的要求。

## 术语和缩略语

术语和缩略语表如表1所示。

表1 术语和缩略语表

|  |  |  |  |
| --- | --- | --- | --- |
| 编号 | 术语 | 英文 | 说明 |
| 1 | UCM | UCM | 用例建模 |
| 2 | RUCM | RUCM | 限制性用例模型 |
| 3 | SparkConf | SparkConf | 包含了集群配置的一系列参数，包括master节点，app名称等。 |
| 4 | SparkContext | SparkContext | 所有Spark程序的入口，相当于main函数。可以完成变量广播，RDD的操作等。SparkContext的构建需要SparkConf。 |
| 5 | DStream | DStream | DStream是Spark的一个基本抽象概念，它代表了一个连续的数据流，数据流按照时间间隔进行切分，会产生一系列的RDD。所以，DStream也可以看作是RDD的序列。 |
| 6 | RDD | RDD | RDD(Resilient Distributed Datasets)弹性分布式数据集，是分布式内存的一个抽象概念，本质上是一个只读的分区记录集合。 |
| 7 | filter | filter | Spark Streaming主要适用于流式的数据处理，而输入流中的数据并不是全部都需要处理，这时便可以使用filter进行数据过滤，减少集群计算压力。 |
| 8 | map/mapPartition/flatMap | map/mapPartition/flatMap | map系列的函数是基于一个RDD中的元素来产生一个新的RDD的方式。其具体的用法，区别，性能等也是本项目研究的方向之一。 |

# 引用文档

* 《Spark Streaming的分析与应用项目计划》

# 业务需求模型

现在，几乎所有的大型公司在某种程度上来说都是软件公司，它们实时地监控传感器、物联网设备、社交网络和在线事务系统产生的数据，然后通过大规模、实时的流处理系统对其进行分析从而实现快速响应。此外，公司还会使用这些数据生成日报和业务模型。也就是说，现代流处理框架不仅需要应对实时场景，还需要处理预处理和后处理等非实时场景。Spark Streaming正在变为实现实时数据处理与分析方案的首选平台。文本分类，数据统计……它被用于各种应用场景。下面是一些典型的Spark Streaming用例：

* 文本简单处理类：这种需求是指仅将输入文本进行独立的处理分析，做出相应的修改后返回，不存在文本之间的交互作用，具体的应用场景有文本分类，文本标注，数据的格式化等。
* 数据“浓缩”类：将实时数据浓缩成更为精炼的数据以用于实时分析。比如简单的示例程序WordCount就属于这一类操作——将海量数据进行某种形式的统计，返回更加丰富的统计信息，从而实现数据的精炼。具体的应用场景可以是网站流量统计，舆情监测等。
* 更加复杂的流式，分布式学习：随着机器学习，深度学习的快速发展，数据量的不断扩张，单机机器学习越来越受到速度，内存等一系列瓶颈的限制。Spark的分布式特点正迎合了这类问题。基于不断流入的数据，分布式训练模型。通过数据并行和模型并行真正实现分布式机器学习。

下面具体介绍社交媒体数据实时分析和网站流量统计这两种应用场景

## 3.1社交媒体数据实时分析

现在很多社交网络公司，比如Twitter，都在利用 Spark Streaming 实时分析社交数据。以下是一个小例子，利用一个不停运转的流程序，实时获取 Twitter 数据，根据文本和位置来收集推文，并使用了 k 均值算法。程序中的数据流如下图4所示：Twitter 数据被提供给 Spark Streaming，Spark Streaming 将它们收集在一起，并在地图上实时标绘出推文。

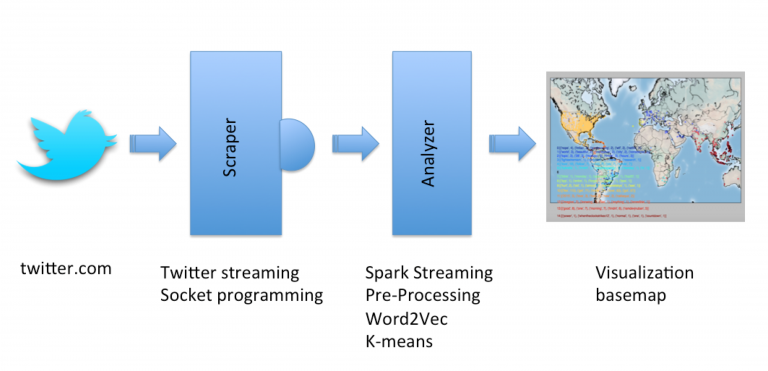


图 2 社交数据分析的数据流

首先，需要读取 Twitter 数据，并将它们写入一个套接字中，然后实时地清理、分析和可视化这些 Twitter 数据。具体来讲，包括三个任务：

* 打开一个套接字，监听并接受来自 Analyzer 的连接请求
* 从 Twitter Streaming API 中读取流推文数据
* 采用 JSON 格式将推文写入套接字中

之后，使用 Spark streaming API 从套接字中读取数据，预处理Twitter 数据，在数据到达某个流时收集，并将结果可视化。其中利用了 k 均值算法，根据分组数据的相似性分别收集。但 k 均值算法只能处理数值向量数据。因此，必须将原始数据转换为特征或向量。这部分采用Word2Vec实现。Word2Vec 计算词语的分布式向量的表示形式。分布式表示形式的主要优点是将类似的词语收集到向量空间中。Word2Vec 使用文档的语料库来计算构建矢量。

最后，分析出来的结果将被发送到一个函数，该函数使用 Python 的地图可视化工具 Basemap 来显示推文。类似推文会划分到同一个集群中，这里的相似性是根据推文的地理位置以及推文中的关键字的类似性来定义的。所以，如果有两篇推文具有相同的颜色，那么它们文本中的术语是相同的，或者是位置相近。接下来，在找到每个集群后，我们会寻找每个集群中最常用的关键字，最常用关键字的列表会揭示该区域中的事件/新闻/趋势。

## 3.2网站流量统计

在互联网应用中，网站流量统计作为一种常用的应用模式，需要在不同粒度上对不同数据进行统计，既有实时性的需求，又需要涉及到聚合、去重、连接等较为复杂的统计需求。传统上，若是使用Hadoop MapReduce框架，虽然可以容易地实现较为复杂的统计需求，但实时性却无法得到保证；反之若是采用Storm这样的流式框架，实时性虽可以得到保证，但需求的实现复杂度也大大提高了。Spark Streaming在两者之间找到了一个平衡点，能够以准实时的方式容易地实现较为复杂的统计需求。

下面介绍一下使用Kafka和Spark Streaming搭建实时流量统计框架。

* 数据暂存：Kafka作为分布式消息队列，既有非常优秀的吞吐量，又有较高的可靠性和扩展性，在这里采用Kafka作为日志传递中间件来接收日志，抓取客户端发送的流量日志，同时接受Spark Streaming的请求，将流量日志按序发送给Spark Streaming集群。
* 数据处理：将Spark Streaming集群与Kafka集群对接，Spark Streaming从Kafka集群中获取流量日志并进行处理。Spark Streaming会实时地从Kafka集群中获取数据并将其存储在内部的可用内存空间中。当每一个batch窗口到来时，便对这些数据进行处理。
* 结果存储：为了便于前端展示和页面请求，处理得到的结果将写入到数据库中。

相比于传统的处理框架，Kafka+Spark Streaming的架构有以下几个优点。

* Spark框架的高效和低延迟保证了Spark Streaming操作的准实时性。
* 利用Spark框架提供的丰富API和高灵活性，可以精简地写出较为复杂的算法。
* 编程模型的高度一致使得上手Spark Streaming相当容易，同时也可以保证业务逻辑在实时处理和批处理上的复用。

# 功能性需求

## 4.1 数据的产生与输入

### 4.1.1输入流接收器

输入接收器Receiver用来持续产生或持续接收系统外的数据，比如 TwitterReceiver 可以实时爬取 twitter 数据。为了支持多种数据源，需要定义多种Receiver，对不同的数据源进行数据获取。

每个ReceiverSupervisor启动后将马上生成一个用户提供的 Receiver 实现的实例，并在 Receiver 实例生成后开始持续不断地接收外界数据，并持续交给ReceiverSupervisor。ReceiverSupervisor将数据成块存储。存储完成后，ReceiverSupervisor及时上报数据的mata信息给driver 端的ReceiverTracker。

### 4.1.2 数据源输入流

在driver端，数据源输入流（ReceiverInputDStream）在每个batch检查ReceiverTracker 收到的数据块的meta信息，界定哪些新数据需要在本batch内处理，然后生成相应的 RDD 实例去处理这些块数据。

## 4.2 数据流抽象DStream

有了数据的来源，就需要有数据的处理与存储。DStream是Spark Streaming中所有流数据的抽象。

### 4.2.1 DStream的离散化

DStream（Discretized Stream）是Spark Streaming对内部持续的实时数据流的抽象描述。而要处理的实时数据流，在Spark Streaming中对应于一个DStream 实例；另外，RDD（Resilient Distributed Datasets）是Spark最基本也是最根本的数据抽象，是只读的分区记录集合。 DStream 代表了一系列连续的RDD，DStream中每个RDD包含特定时间间隔的数据。



图 3 DStream的离散化

对DStream应用的任何操作都会转换为RDD上的操作。例如，在将文本流转换为词流的示例中，flatMap操作应用于文本的DStream中的每个RDD，以生成词的DStream的RDD 。



图 4 DStream与RDD

### 4.2.2 基本数据集抽象——RDD

RDD是一个分布式计算抽象模型，它提供了通用的数据处理方法和高效的分布式容错机制，Spark是它的一种实现。

当初设计RDD主要是为了解决三个问题：

Fast: Spark之前的[Hadoop](http://lib.csdn.net/base/hadoop" \t "_blank" \o "Hadoop知识库)用的是MapReduce的编程模型，没有很好的利用分布式内存系统，中间结果都需要保存到external disk，运行效率很低。RDD模型是in-memory computing的，中间结果不需要被物化（materialized），它的 persistence机制，可以保存中间结果重复使用，对需要迭代运算的[机器学习](http://lib.csdn.net/base/machinelearning)应用和交互式数据挖掘应用，加速显著。Spark快还有一个原因是开头提到过的 Delay Scheduling机制，它得益于RDD的Dependency设计。

General: MapReduce编程模型只能提供有限的运算种类（Map和Reduce），RDD希望支持更广泛更多样的operators（map，flatMap，filter等等），然后用户可以任意地组合他们。

Fault tolerance: 其他的in-memory storage on clusters，基本单元是可变的，用细粒度更新（ fine-grained updates ）方式改变状态，如改变table/cell里面的值，这种模型的容错只能通过复制多个数据copy，需要传输大量的数据，容错效率低下。而RDD是 不可变的（immutable） ，通过粗粒度变换（ coarse-grained transformations ），比如map，filter和join，可以把相同的运算同时作用在许多数据单元上，这样的变换只会产生新的RDD而不改变旧的RDD。这种模型可以让Spark用 Lineage 很高效地容错。

### 4.2.3 DStream的操作

Spark的API运算函数分为两大类，Transformation 和 Action ：Transformations是 lazy evaluation 的，调用他们只会被记录而不会被真正执行，只有遇到Actions，之前的Transformations才会被依次执行

Spark的基本工作流程是，用户提交程序给cluster，用户的main函数会在 Driver 上面运行，根据用户的程序Spark会产生很多的 Jobs，原则是遇到一个 Action 就产生一个 Job ，以DAG图的方式记录RDD之间的依赖关系，每一个Job又会根据这些依赖关系被DAGScheduler分成不同的 Stages ，每一个Stage是一个 TaskSet ，以TaskSet为单位，TaskScheduler通过Cluster Manager一批一批地调度到不同node上运行，同一个TaskSet里面的Task都做同样的运算，一个Partition对应一个Task。

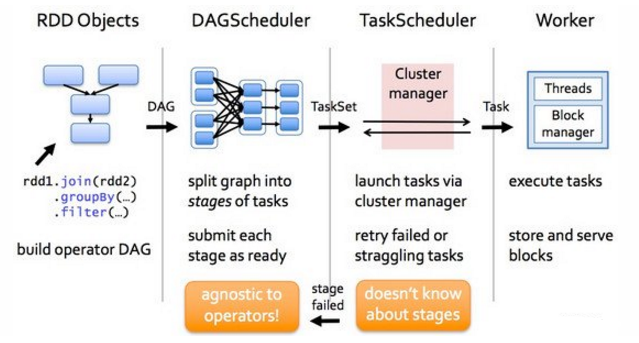


图 5 Spark工作流程

为了适应不同的数据处理业务需求，SparkStreaming（Spark）实现了丰富的数据处理算子。其中，属于Transformation 的方法如表2所示

表2 transformation方法表

|  |  |
| --- | --- |
| **用例** | **描述** |
| **map**(*func*) | 源DStream的每个元素通过函数func返回一个新的DStream。 |
| **flatMap**(*func*) | 类似与map操作，不同的是每个输入元素可以被映射出0或者更多的输出元素。 |
| **filter**(*func*) | 在源DSTREAM上选择Func函数返回仅为true的元素,最终返回一个新的DSTREAM 。 |
| **sample(**withReplacement,fraction,seed**)** | 以指定的随机种子随机抽样出数量为fraction的数据 |
| **union**(*another Stream*) | 返回一个包含源DStream与其他DStream的元素合并后的新DStream。 |
| **reduceByKey**(*func*, [*numTasks*]) | 当一个类型为（K，V）键值对的DStream被调用的时候,返回类型为（K，V）键值对的新 DStream,其中每个键的值V都是使用聚合函数func汇总。注意：默认情况下，使用 Spark的默认并行度提交任务（本地模式下并行度为2，集群模式下位8），可以通过配置numTasks设置不同的并行任务数。 |
| **join**(*another Stream*, [*numTasks*]) | 当被调用类型分别为（K，V）和（K，W）键值对的2个DStream时，返回类型为（K，（V，W））键值对的一个新 DSTREAM。 |
| **cogroup**(*another Stream*, [*numTasks*]) | 当被调用的两个DStream分别含有(K, V) 和(K, W)键值对时,返回一个(K, Seq[V], Seq[W])类型的新的DStream。 |
| **transform**(*func*) | 通过对源DStream的每RDD应用RDD-to-RDD函数返回一个新的DStream，这可以用来在DStream做任意RDD操作。 |
| **updateStateByKey**(*func*) | 返回一个新状态的DStream,其中每个键的状态是根据键的前一个状态和键的新值应用给定函数func后的更新。这个方法可以被用来维持每个键的任何状态数据。 |
| **…** | … |

属于action的方法如表3所示：

表3 action方法表

|  |  |
| --- | --- |
| **用例** | **描述** |
| **saveAsTextFiles(prefix, [suffix])** | 将DStream中的内容以文本的形式保存为文本文件，其中每次批处理间隔内产生的文件以prefix-TIME\_IN\_MS[.suffix]的方式命名。 |
| **saveAsObjectFiles(prefix, [suffix])** | 将DStream中的内容按对象序列化并且以SequenceFile的格式保存。其中每次批处理间隔内产生的文件以prefix-TIME\_IN\_MS[.suffix]的方式命名。 |
| **saveAsHadoopFiles(prefix, [suffix])** | 将DStream中的内容以文本的形式保存为Hadoop文件，其中每次批处理间隔内产生的文件以prefix-TIME\_IN\_MS[.suffix]的方式命名。 |
| **reduce(func)** | 使用函数func（有两个参数并返回一个结果）将源DStream 中每个RDD的元素进行聚 合操作,返回一个内部所包含的RDD只有一个元素的新DStream。 |
| **countByValue**() | 计算DStream中每个RDD内的元素出现的频次并返回新的DStream[(K,V)]，其中K是RDD中元素的类型，V是元素出现的频次。 |
| **count**() | 对源DStream内部的所含有的RDD的元素数量进行计数，返回一个内部的RDD只包含一个元素的DStream。 |
| **…** | … |

## 3.3 作业调度

在每个Spark应用（包含一个SparkContext实例）中运行了一些其独占的执行器（executor）进程。集群管理器提供了Spark应用之间的资源调度。在各个Spark应用内部，各个线程可能并发地通过action操作提交多个Spark作业。在Spark应用内部（对应同一个SparkContext）各个作业之间，Spark默认FIFO调度，同时也可以支持公平调度。下面将分别介绍Spark应用之间的资源调度以及Spark应用内部的资源调度。

### 4.3.1 Spark应用之间的资源调度

如果在集群上运行，每个Spark应用都会获得一批独占的执行器JVM，来运行其任务并存储数据。如果有多个用户共享集群，那么会有很多资源分配相关的选项，如何设置还取决于具体的集群管理器。

对Spark所支持的各个集群管理器而言，最简单的资源分配，就是对资源静态划分。这种方式就意味着，每个Spark应用都是设定一个最大可用资源总量，并且该应用在整个生命周期内都会占住这些资源。这种方式在Spark独立部署（standalone）和YARN调度，以及Mesos粗粒度模式（coarse-grained Mesos mode）下都可用。

* Standalone mode: 默认情况下，Spark应用在独立部署的集群中都会以FIFO（first-in-first-out）模式顺序提交运行，并且每个Spark应用都会占用集群中所有可用节点。不过可以通过设置spark.cores.max或者spark.deploy.defaultCores 来限制单个应用所占用的节点个数。最后，除了可以控制对CPU的使用数量之外，还可以通过 spark.executor.memory 来控制各个应用的内存占用量。
* Mesos: 在Mesos中要使用静态划分的话，需要将 spark.mesos.coarse 设为true，同样，也需要设置 spark.cores.max 来控制各个应用的CPU总数，以及 spark.executor.memory 来控制各个应用的内存占用。
* YARN: 在YARN中需要使用 –num-executors 选项来控制Spark应用在集群中分配的执行器的个数，对于单个执行器（executor）所占用的资源，可以使用 –executor-memory 和 –executor-cores 来控制。

### 4.3.2 同一应用程序内的资源调度

在Spark应用程序（即SparkContext实例）内部，如果多个作业分别由单独的线程提交，那么它们就可以同时并行执行。所谓的作业指的是Spark action（如save、collect等）以及需要执行该action的任何任务。Spark调度器是完全线程安全的，而且支持这种特性能让应用程序响应多个服务请求（比如为多个用户提供查询服务）。

默认情况下，spark调度器以FIFO方式运行作业。每个作业被分成若干个“stage”（比如map和reduce阶段），同时，当第一个作业中的stage有任务要启动时，那么该作业优先获得所有可用资源，接着是第二个作业获得优先权，依次类推。如果队列首位的作业不需要占用集群上的所有资源，那么后续的作业就可以立即启动，但是如果队列首位的作业规模很大，那么后续作业的执行会显著地延迟。

从Spark 0.8版开始，通过配置可以让各个作业公平地共享资源。在公平资源共享模式下，spark以轮转方式在各个作业之间分配任务，以便所有作业都能大致平等地共享集群资源。这就意味着，在长期作业正在运行时提交的短期作业可以立即获得资源并得到良好的响应时间，而无需等待长期作业执行完毕。该模式对于多用户环境是最佳的。

## 4.4 长时间容错

由于Spark的数据导入导出、数据抽象、作业调度3个模块都是分布式的并且长时间运行的，所以需要一个错误响应与处理模块来应对前3个模块运行过程中出现的错误。其中保障数据的产生与输入模块需要进行数据的备份。保障数据流抽象和作业调度这两个模块需要对这两个模块的过程数据设置检查点，在出错时读取检查点信息然后恢复即可。

### 4.4.1 数据导入导出模块长时间容错

在数据导入导出模块，数据接收器如果出错只需要直接重启就可以了。核心是需要保障接收到的数据的安全，保障了源数据，就可以保障后续工作的重现。在这里我们可以按照需求场景的不同来灵活设定对源数据的保障。

* 热备份**：**热备份是指在存储块数据时，将其存储到本执行器、并同时复制到另外一个执行器上去。这样在一个执行器失效后，可以立刻无感知切换到另一份 执行器进行计算。实现方式是，在实现自己的接收器时，设置存储级别。将块数据存储在多个执行器上。
* 冷备份：冷备份是每次存储块数据前，可以先把块数据作为 log 写出到log日志里，再存储到所在的执行器。执行器失效时，就由另外的执行器去读日志信息，再重做 log 来恢复块数据。日志信息通常写到可靠存储如 HDFS上，所以恢复时可能需要花费一段重现时间。
* 数据重放：如果源数据的发出者支持数据重放，例如Apache Kafka，那就可以选择不在数据接收端来备份数据而是出现问题时让源数据的发出者进行数据重发即可。

### 4.4.2 数据抽象模块及作业调度模块的长时间容错

由于数据抽象模块及作业调度模块都是对输入信息进行处理，产生块数据进行执行，所以可以采用上一节提到的冷备份的方式进行备份。当出现问题时，读取到log日志然后重现操作即可。此外，由于这两个模块会产生大量的过程数据。所以可以对整个流程处理设置检查点。来记录两个模块的完成情况。这样可以在出现问题后选择最近一次的检查点进行恢复，最大化的减少损失时间。

## 4.5 窗口支持

为了更有效率的再一组RDD进行计算，spark streaming提供了窗口计算，可以转换滑动窗口内的数据，即可以更灵活的控制DStream的大小（时间间隔大小、数据元素个数）。在设置好窗口的长度，滑动的间隔长度后，窗口在源DStream上滑动。每次滑动，窗口内的RDD将被合并生成窗口DStream内的RDD。

## 4.6 用例图

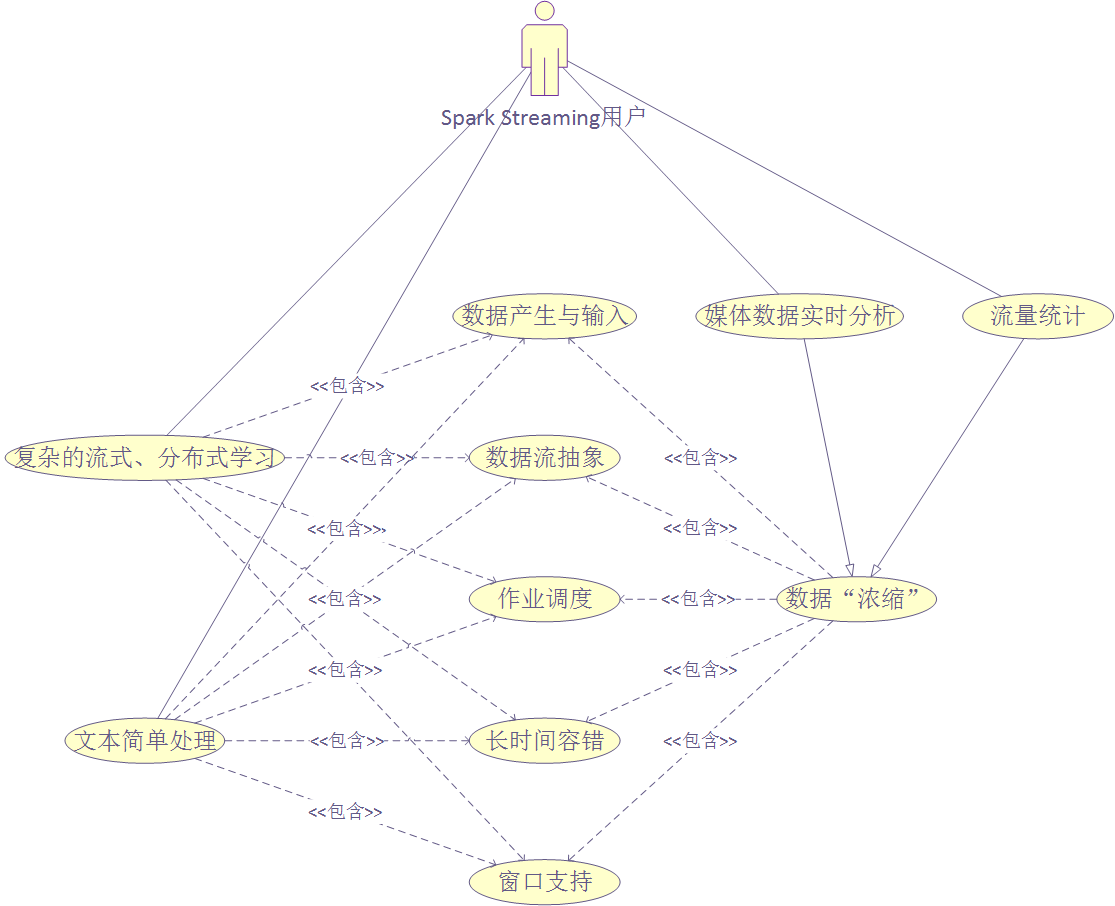


图 6 用例图

## 4.7 用例说明

下面使用RUCM模型描述用例。

### 4.7.1 产生与输入数据



图 7产生与输入数据RUCM模型

### 4.7.2 数据流抽象



图 8数据流抽象RUCM模型

### 4.7.3 系统容错



图 9系统容错RUCM模型

### 4.7.4 作业调度

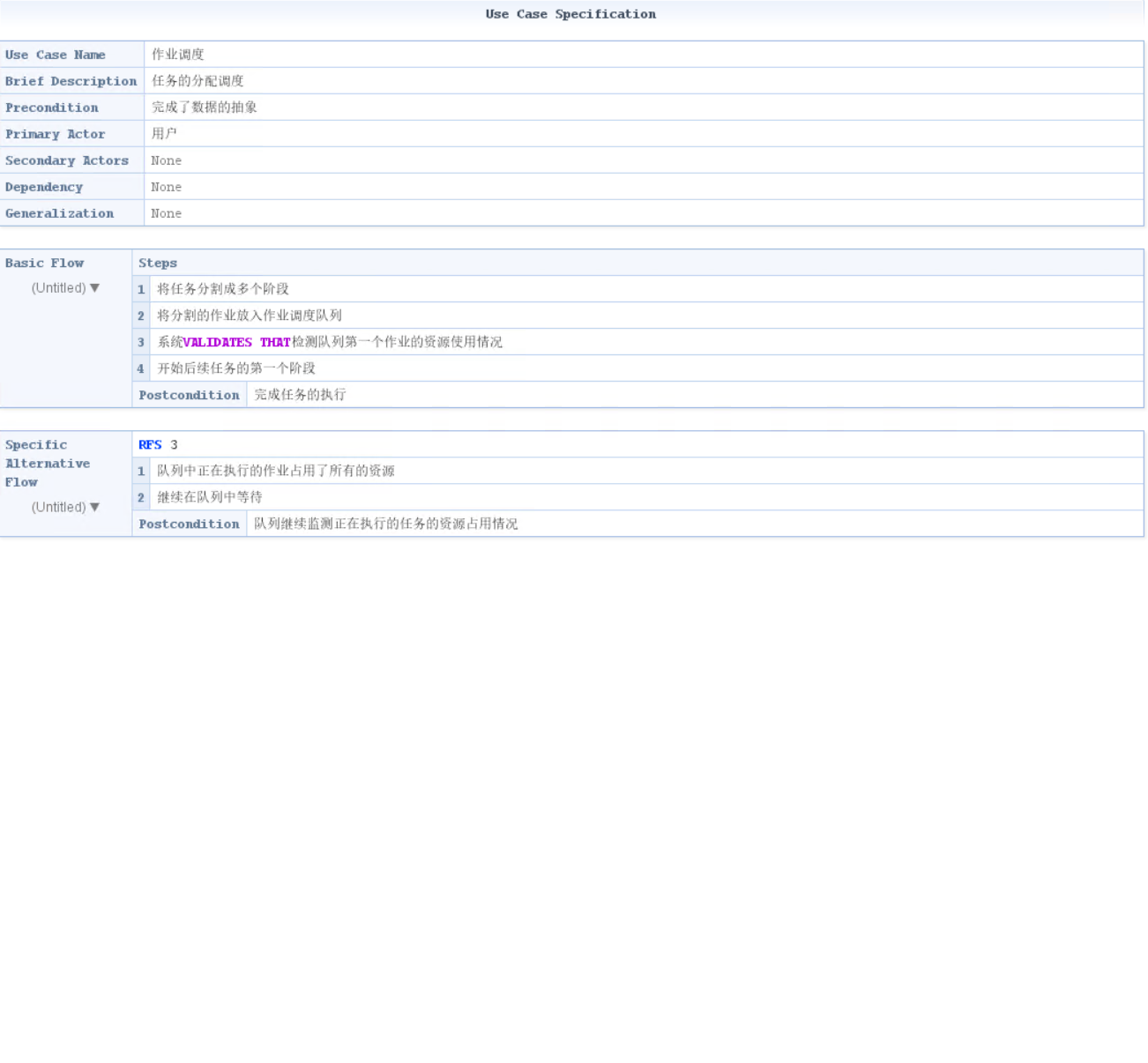


图 10作业调度RUCM模型

# 5. 非功能性需求

## 5.1 实时性

由于Spark Streaming需要进行大型流数据处理，其系统的实时性有很高要求。也就是说，Spark Streaming对外来事件要在规定的时间内作出反应。Spark Streaming将流式计算分解成多个Spark Job，对于每一段数据的处理都会经过Spark DAG图分解以及Spark的任务集的调度过程。所以这就要求Spark Streaming的批处理时间不能太大，以保证到来的数据能在零点几或几秒内完成。

## 5.2 扩展性与吞吐量

作为大型分布式系统，其数据的处理能力是至关重要的。对于数据的处理能力，涉及到两方面的内容，一是可扩展性，二是吞吐量。

可扩展性是软件系统计算处理能力的设计指标，高可伸缩性代表一种弹性，在系统扩展成长过程中，软件能够保证旺盛的生命力，通过很少的改动甚至只是硬件设备的添置，就能实现整个系统处理能力的线性增长，实现高吞吐量和低延迟高性能。对于这里而言，就是Spark Streaming对节点数量的容纳能力。

吞吐量是指系统在单位时间内处理请求的数量，在这里，吞吐量指的是Spark Streaming每个节点对数据的处理能力。

## 5.3 持久化

持久化（Persistence），即把数据（如内存中的对象）保存到可永久保存的存储设备中（如磁盘）。持久化的主要应用是将内存中的对象存储在数据库中，或者存储在磁盘文件中、XML数据文件中等。因为内存运行速度快，而磁盘等可以永久存储，所以对于不同的数据，Spark Streaming需要采用不同的持久化方式，来满足各部分数据不同的需求。

对于一些基本的操作的数据，如窗口操作，其持久化方式就是保存到内存中，这样可以明显提升计算速度。对于一些来自网络的数据源，为了容错能力等，可以采用将数据保存到两台机器上的持久化方式进行。

## 5.4性能调优

### 5.4.1合理的批处理时间

在Spark Streaming中需要设置批处理的时间间隔，每隔此时间间隔会提交一个Job。Job之间可能存在依赖关系，后面的Job必须确保前面的Job执行结束才能提交。若前面Job执行时间超过了批处理时间间隔，后面的Job无法按时提交，就会进一步拖延接下来的Job，造成后续Job的阻塞。所以必须设置合理的批处理时间间隔。

### 5.4.2 增加并行化

数据处理并行化，有助于充分利用资源，以提高性能。增加并行化程度，可以通过增加Job的并行度，或者从接收端来增加并行度。

### 5.4.3 及时清除过期数据

Spark Streaming会将接收到的数据全部存储，随着时间推移，有些数据不需要了，这些数据的存储会浪费宝贵的内存资源。所以需要定期清理超时的无用数据。

# 设计实现

## 5.1工作目标

项目计划基于Spark Streaming框架，基于机器学习、自然语言处理等技术实现指定主题短文本的自动识别功能。本软件产品是一项独立的软件，用户根据软件提供的调用命令实现各种指定功能。

期望达到的性能指标是：在十台机器的spark集群上达到每分钟5万条的处理速度。

## 5.2软件基本功能描述

数据生产模块读取一批固定的，未标注的短文本集合，将其不断发送至kafka的指定topic。

基于Spark Streaming的短文本分类主系统从kafka的上述topic中不断获取实时流数据，根据用户信息等字段可以对短文本进行初步的过滤，而后调用分类模型对短文本流进行分类，打分类标签后存入HDFS和kafka的另一个topic。

后处理模块从kafka结果topic中读取分类结果，存入mysql数据库。

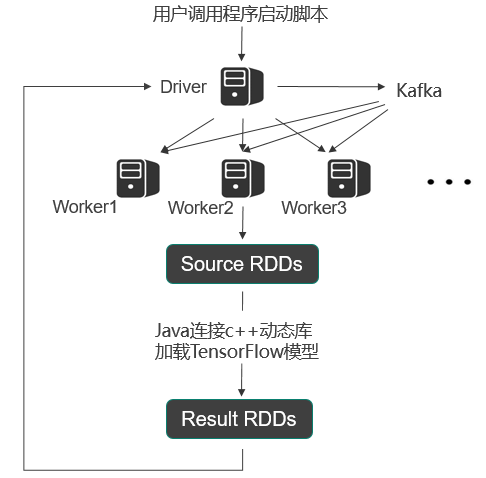


图 11 系统架构图

## 5.3技术路线

利用已标注的训练样本，进行分词、关键词抽取、词语泛化等预处理后，基于tensorFlow/libsvm训练短文本分类模型。

编写Kafka发送数据程序，读取现有的有限的短文本，产生重复的数据流输入。

编写Spark应用程序，利用Spark API中的filter、map等RDD操作。调用分类模型进行实时流数据短文本分类。

编写kafka消费程序，读取结果topic写入数据库完成短文本的分类任务。

## 5.4详细实现方案

### 模型训练

由于本项目的重点在于Spark Streaming的使用，模型训练并不是我们要着重关心的问题。这里，我们将使用python的sk-learn工具作为文本分类的模型训练工具。

### 数据的产生

由于项目能获取的样本数有限，我们采用对同一批数据进行无限复制从而产生流式数据的解决办法。数据的产生由单独的程序控制，程度每隔一段时间对源数据进行读取，并按照数据格式标准将源数据发送到kafka指定主题下，供给分类程序使用。

### 文本预处理

对于输入的文本，首先要进行分词、去停用词这样的基本操作，而为了忽略掉一些对文本分类无意义的词，从而达到更好的分类效果，还需将文本中的量词、人名等进行泛化处理，如将“张小明”泛化为“<人名>”，将“一百只”泛化为“<量词>”等。

### 文本的分类

在spark的各个计算节点上，加载训练好的模型。在文本预处理的基础上，对处理后的文本进行向量化，输入给训练好的模型，得出分类结果。将分类结果与源数据进行拼接，返回。

### 分类结果的导出

在spark程序将输入的文本进行分类，写入kafka消息队列后，从消息队列中读取分类结果，按照协议切分分类结果，将各个字段写入mysql数据库，完成整个分类系统。

## 5.5技术难点

本课题使用Spark作为分布式计算框架，使用java或python作为开发语言。没有大的困难点，但是会存在一些诸如分布式模型调用等方面的工程问题。因此项目风险也属于可控范围内，项目是可行的。

# 6.运行环境

## 6.1硬件环境

当前实验室拥有12台服务器可供使用，硬件为core i7处理器，16G内存，1TB硬盘。

## 6.2软件环境

软件环境如表4所示。

表4 软件环境表

|  |  |
| --- | --- |
| 操作系统 | centos 6.5或Ubuntu16.04及以上 |
| 开发环境 | eclipse |
| 项目管理 | Github |
| Hadoop | 2.2及以上 |
| Spark | 1.6及以上 |

# 7.参考资料

[1]<https://www.ibm.com/developerworks/cn/analytics/blog/analyze-social-media-data-real-time/index.html>

[2] <https://academy.datastax.com/resources/apache-spark-streaming>

[3] <http://www.infoq.com/cn/articles/apache-spark-streaming>