

第十一讲 流计算系统 Spark Streaming



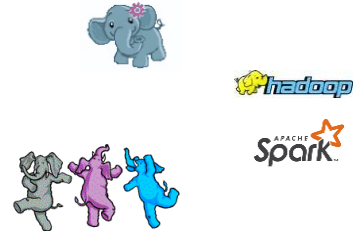
徐辰
cxu@dase.ecnu.edu.cn

华东师范大学

DaSE
Data Science
& Engineering

Making an elephant dancing

2



大纲

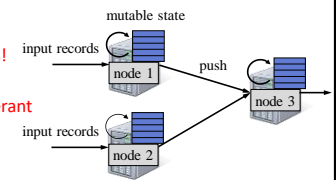
3

- 设计思想
 - ✦ 连续处理 vs. 微批处理
 - ✦ 数据模型
 - ✦ 计算模型
- 体系架构
- 工作原理
- 容错机制
- 编程实例

连续处理(Continuous processing)

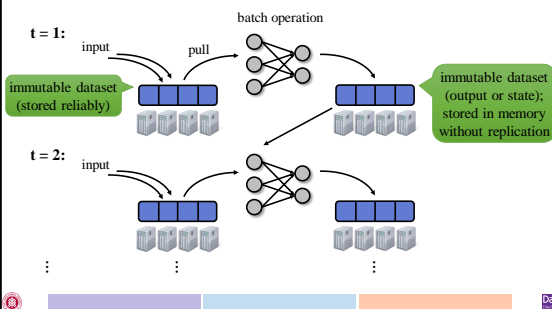
4

- Existing streaming systems have a record-at-a-time processing model
 - ✦ Each node has **mutable state**
 - ✦ For each record, update state and send new records
- State is lost if node dies!
- Making stateful stream processing be fault-tolerant is challenging



微批处理(Micro-batch processing)

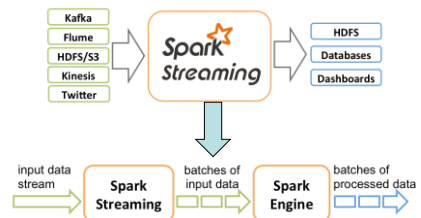
5



What is Spark Streaming?

6

- Extends Spark for doing large scale stream processing



Basic Idea

7

- Break input data streams into **batches**
- Derive an **RDD** per batch
- Emulate (模拟)** a **continuous** execution by scheduling each consecutive **RDD** operation (as its own application).

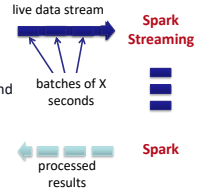


Spark + MicroBatch

8

Run a streaming computation as a **series of very small, deterministic batch jobs**

- Chop up the live stream into batches of X seconds
- Spark treats each batch of data as RDDs and processes them using RDD operations
- Finally, the processed results of the RDD operations are returned in batches



大纲

9

- 设计思想**
 - 连续处理 vs. 微批处理
 - 数据模型**
 - 计算模型**
- 体系架构**
- 工作原理**
- 容错机制**
- 编程实例**

流数据的离散化

10

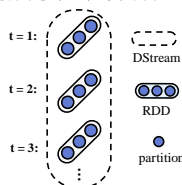
- 在Storm的数据模型中，我们将流数据看作一系列连续的元组
- Spark Streaming将连续的流数据切片，即离散化，生成一系列小块数据



DStream数据模型

11

- 按照数据到来的时间间隔将连续的数据流离散化
- 得到的每一小批数据都是独立的RDD，一组RDD序列抽象为流数据的DStream



DStream数据模型

12

- 对输入数据流进行切割
 - 根据用户指定的时间间隔对数据进行切割
 - DStream维护一系列RDD的信息



大纲

13

□ 设计思想

✚ 连续处理 vs. 微批处理

✚ 数据模型

✚ 计算模型

□ 体系架构

□ 工作原理

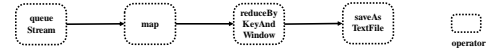
□ 容错机制

□ 编程实例

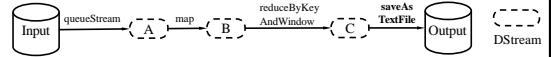
逻辑计算模型

14

□ Operator DAG: 与RDD的operator类似

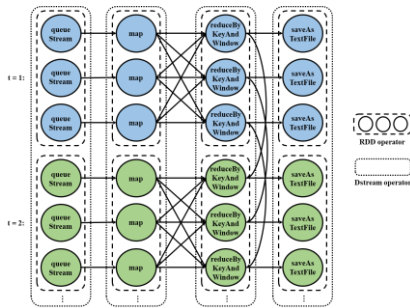


□ DStream Lineage: 借用RDD Lineage的概念



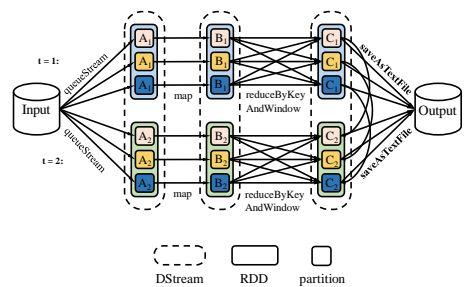
物理计算模型: Operator DAG

15



物理计算模型: DStream Lineage

16



大纲

17

□ 设计思想

□ 体系架构

✚ 架构图

✚ 应用程序执行流程

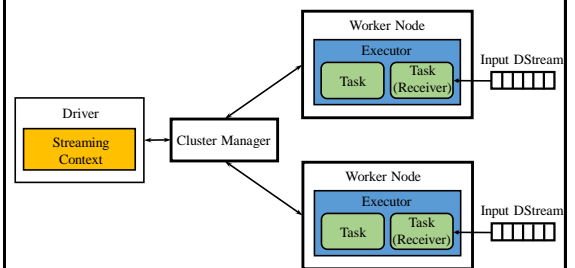
□ 工作原理

□ 容错机制

□ 编程实例

架构图

18



工作部件

19

- Driver: Spark Streaming对SparkContext进行了扩充, 构造了StreamingContext, 用于管理流计算的元信息。
- Executor: Executor中作为Receiver的某些task, 负责从外部数据源源不断的获取流数据, 这和spark批处理读取数据的方式是不同的。

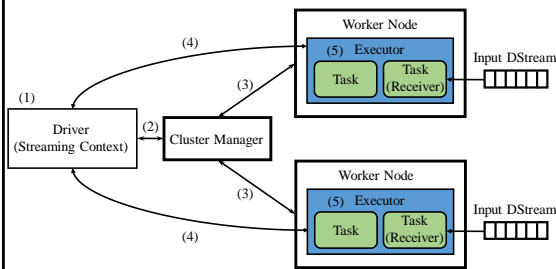
大纲

20

- 设计思想
- 体系架构
 - ✦ 架构图
 - ✦ 应用程序执行流程
- 工作原理
- 容错机制
- 编程实例

应用程序执行流程

21



应用程序执行流程

22

1. 启动Driver, 以Standalone模式为例
 - ✦ 如果使用Client部署方式, 客户端直接启动Driver, 并向Master注册。
 - ✦ 如果使用Cluster部署方式, 客户端将应用程序提交给Master, 由Master选择一个Worker启动Driver进程(DriverWrapper)。
2. 构建基本运行环境, 即由Driver创建StreamingContext, 向Cluster Manager进行资源申请, 并由Driver进行任务分配和监控

应用程序执行流程 (续)

23

3. Cluster Manager通知工作节点启动Executor进程, 该进程内部以多线程方式运行任务
4. Executor进程向Driver注册
5. StreamingContext构建关于RDD转换的DAG, 从而交给Executor进程中的线程来执行任务。

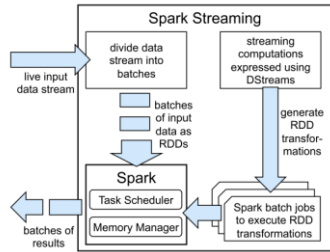
大纲

24

- 设计思想
- 体系架构
- 工作原理
- 容错机制
- 编程实例

工作过程

25



大纲

26

- 设计思想
- 体系架构
- 工作原理
 - ✚ 数据输入
 - ✚ 数据转换
 - ✚ 数据输出
- 容错机制
- 编程实例

流数据读取

27

- 从外部数据源直接获取数据
 - ✚ 从socket端口获取网络数据
 - ✚ 接收外部传感器产生的数据
 - ✚
- 从外部存储系统周期性地读取数据
 - ✚ 数据源将数据存入HDFS
 - ✚ 数据源将数据存入Kafaka
 - ✚

常用数据输入接口

28

- socketTextStream
- fileStream
- textFileStream
- queueStream

<https://spark.apache.org/docs/latest/streaming-programming-guide.html#input-dstreams-and-receivers>

大纲

29

- 设计思想
- 体系架构
- 工作原理
 - ✚ 数据输入
 - ✚ 数据转换
 - ✚ 数据输出
- 容错机制
- 编程实例

数据转换

30

- DStream转换操作
 - ✚ map(func)
 - ✚ flatMap(func)
 - ✚ filter(func)
 - ✚ repartition(numPartitions)
 - ✚ union(otherStream)
 - ✚ count()
 - ✚ reduce(func)
 - ✚ countByValue()
 - ✚ reduceByKey(func, [numTasks])
 - ✚ join(otherStream, [numTasks])
 - ✚ cogroup(otherStream, [numTasks])
- 直接使用RDD转换操作
 - ✚ Transform(func)

<https://spark.apache.org/docs/latest/streaming-programming-guide.html#transformations-on-dstreams>

Transformation

Transformation

- 允许用户在DStream的每个RDD上执行DStream为支持的RDD操作
- 例如：DStream与RDD之间的join

```
val spamInfoRDD = ssc.sparkContext.newAPIHadoopRDD(...)
// RDD containing spam information

val cleanedDStream = wordsContext.transform { rdd =>
  rdd.join(spamInfoRDD).filter(...)
// join data stream with spam information to do data cleaning
  ...
}
```

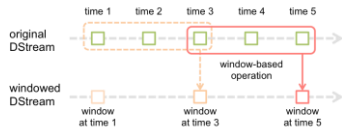
数据转换

- window(windowLength, slideInterval)
- countByWindow(windowLength, slideInterval)
- reduceByWindow(func, windowLength, slideInterval)
- reduceByKeyAndWindow(func, windowLength, slideInterval, [numTasks])
- reduceByKeyAndWindow(func, invFunc, windowLength, slideInterval, [numTasks])
- countByValueAndWindow(windowLength, slideInterval, [numTasks])
- updateStateByKey(func)

窗口操作

滑动窗口

用户可以指定窗口时间间隔，窗口大小



```
val tweets = ssc.twitterStream()
val hashTags = tweets.flatMap(status => getTags(status))
val tagCounts = hashTags.window(Minutes(1), Seconds(5)).countByValue()
```

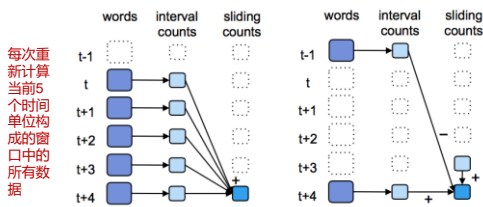


DStream窗口操作

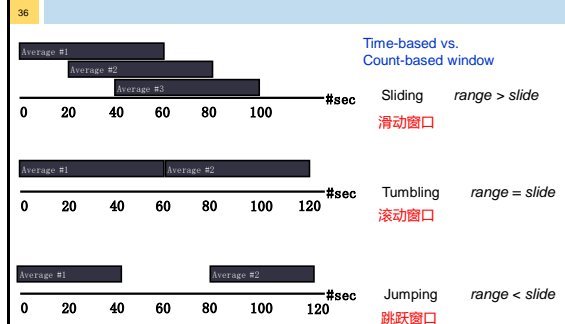
- window(windowLength, slideInterval)
- countByWindow(windowLength, slideInterval)
- countByValueAndWindow(windowLength, slideInterval, [numTasks])
- reduceByKeyAndWindow(func, windowLength, slideInterval, [numTasks])
- reduceByKeyAndWindow(func, invFunc, windowLength, slideInterval, [numTasks])

窗口逆函数

增量式窗口操作：t+3的sliding counts按照左边方式得出，t+4的sliding counts右边方式得出



Window Types



数据转换

37

- window(windowLength, slideInterval)
- countByWindow(windowLength, slideInterval)
- reduceByWindow(func, windowLength, slideInterval)
- reduceByKeyAndWindow(func, windowLength, slideInterval, [numTasks])
- reduceByKeyAndWindow(func, invFunc, windowLength, slideInterval, [numTasks])
- countByValueAndWindow(windowLength, slideInterval, [numTasks])
- **updateStateByKey(func)** 状态操作

状态操作

38

- 允许用户维护任意的状态信息，并根据新数据更新状态
 - ✦ 定义状态类型
 - ✦ 状态更新函数
 - UpdateStateByKey
- Example
 - ✦ Maintain per-user mood as state
 - ✦ update it with their tweets


```
updateMood(newTweets, lastMood) => newMood
moods = tweets.updateStateByKey(updateMood _)
```

Stateful vs. Stateless

39

- Stateful transformations: 有状态的转换
 - ✦ 计算跨越批次
- Stateless transformations: 无状态的转换
 - ✦ 计算不跨批次

大纲

40

- 设计思想
- 体系架构
- 工作原理
 - ✦ 数据输入
 - ✦ 数据转换
 - ✦ 数据输出
- 容错机制
- 编程实例

数据输出

41

- print
- saveAsTextFiles(prefix,[suffix])
- saveAsObjectFiles(prefix,[suffix])
- saveAsHadoopFiles(prefix,[suffix])
- foreachRDD(func)
 - ✦ 通用接口，通过func可以对DStream的每个RDD执行操作

大纲

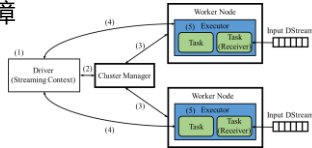
42

- 设计思想
- 体系架构
- 工作原理
- 容错机制
- 编程实例

故障类型

43

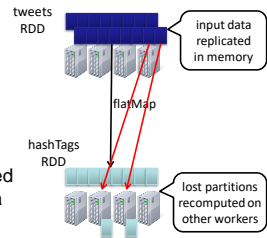
- Cluster Manager故障：不考虑
- Executor(Worker)故障
 - 不含Receiver
 - 含有Receiver
- Driver故障



不含Receiver的Executor故障

44

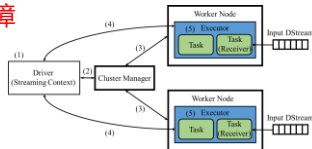
- RDDs remember the operations that created them
- Batches of input data are replicated in memory for fault-tolerance
- Data lost due to worker failure, can be recomputed from replicated input data
- All transformed data is fault-tolerant, and exactly-once transformations



故障类型

45

- Cluster Manager故障：不考虑
- Executor(Worker)故障
 - 不含Receiver
 - 含有Receiver
- Driver故障



大纲

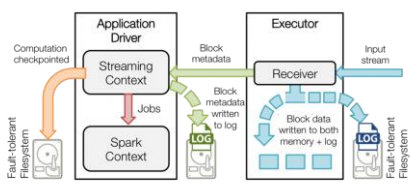
46

- 设计思想
- 体系架构
- 工作原理
- 容错机制
 - 日志与检查点
 - 故障恢复
- 编程实例

日志

47

- 日志：标记哪些接收到数据已经做了备份
 - Receiver日志
 - Driver日志



检查点

48

- 检查点：计算的结果
 - 数据检查点
 - 将有状态转换操作生成的RDD周期性地写入检查点
 - 元数据检查点
 - 配置信息：创建spark streaming应用程序的配置信息
 - DStream操作信息：定义了应用程序计算逻辑的DStream操作的信息
 - 未处理的batch信息：那些正在排队的作业中还没处理的batch信息

大纲

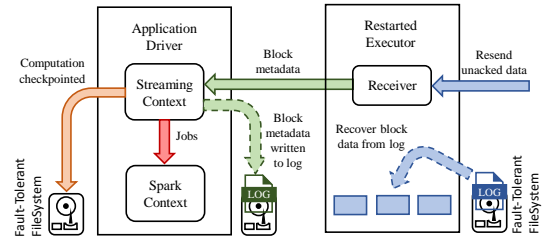
49

- 设计思想
- 体系架构
- 工作原理
- 容错机制
 - ✦ 日志与检查点
 - ✦ 故障恢复
- 编程实例

故障恢复

50

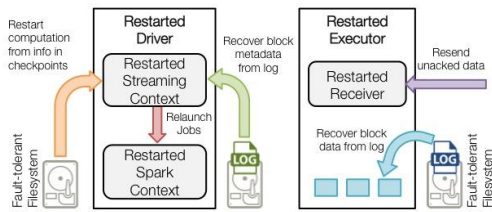
Executor(Worker)故障恢复



故障恢复

51

Driver故障恢复



容错语义

52

- Receiving the data: *at-least or exactly once*
 - ✦ The data is received from sources using Receivers or others.
- Transforming the data: *exactly once*
 - ✦ The received data is transformed using DStream and RDD transformations.
- Pushing out the data: *at-least or exactly once*
 - ✦ The final transformed data is pushed out to external systems like file systems, databases, dashboards, etc.

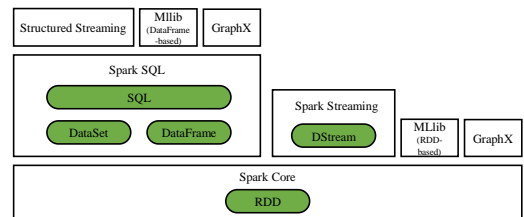
大纲

53

- 设计思想
- 体系架构
- 工作原理
- 容错机制
- 编程实例

Spark API

54



Spark Streaming程序框架

55

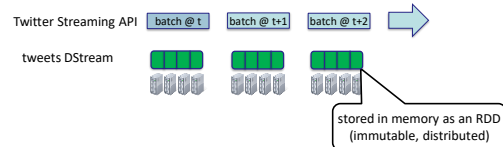
- 1.通过创建输入DStream来定义输入源
- 2.通过对DStream应用转换操作和输出操作来定义流计算
- 3.用streamingContext.start()来开始接收数据和处理流程
- 4.通过streamingContext.awaitTermination()方法来等待处理结束（手动结束或因为错误而结束）
- 5.可以通过streamingContext.stop()来手动结束流计算进程

Example – Get hashtags from Twitter

56

```
val tweets = ssc.twitterStream()
```

DStream: a sequence of RDDs representing a stream of data

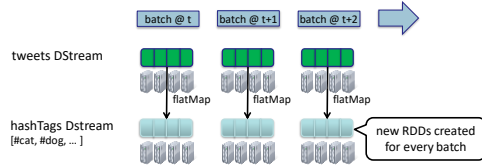


Example – Get hashtags from Twitter

57

```
val tweets = ssc.twitterStream()
val hashTags = tweets.flatMap(status => getTags(status))
```

new DStream transformation: modify data in one DStream to create another DStream

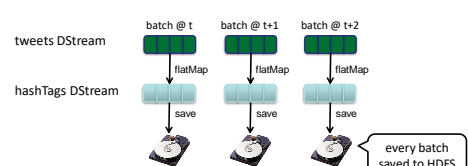


Example – Get hashtags from Twitter

58

```
val tweets = ssc.twitterStream()
val hashTags = tweets.flatMap(status => getTags(status))
                        .saveAsHadoopFiles("hdfs://...")
```

output operation: to push data to external storage

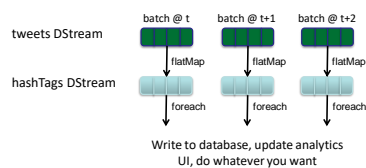


Example – Get hashtags from Twitter

59

```
val tweets = ssc.twitterStream()
val hashTags = tweets.flatMap(status => getTags(status))
                        .foreach(hashTagRDD => { ... })
```

foreach: do whatever you want with the processed data



Java Example

60

Scala

```
val tweets = ssc.twitterStream()
val hashTags = tweets.flatMap(status => getTags(status))
hashTags.saveAsHadoopFiles("hdfs://...")
```

Java

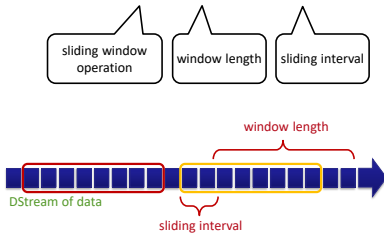
```
JavaDStream<Status> tweets = ssc.twitterStream()
JavaDStream<String> hashTags = tweets.flatMap(new
Function<Status, Iterable<String>>() { ... })
hashTags.saveAsHadoopFiles("hdfs://...")
```

Function object

Window-based Transformations

61

```
val tweets = ssc.twitterStream()
val hashTags = tweets.flatMap(status => getTags(status))
val tagCounts = hashTags.window(Minutes(1), Seconds(5)).countByValue()
```



课后阅读

62

论文

- ✦ Zaharia, M., Das, T., Li, H., Hunter, T., Shenker, S., & Stoica, I. (2013). Discretized Streams: Fault-Tolerant Streaming Computation at Scale. In SOSP (pp. 423–438).

本讲小结

63

- 设计思想
- 体系架构
- 工作原理
- 容错机制
- 编程实例

谢谢! Q&A

Spark
Streaming



Reminder: 本科生

64

实验报告提交

- ✦ Spark部署与编程: 2019年11月17日 23:59
- ✦ Storm部署与编程: 2019年11月24日 23:59

作业提交

- ✦ Storm: 2019年11月20日 23:59

Reminder: 研究生

65

- 选择上课所讲述之外的系统, 从设计思想、体系结构、工作原理、容错机制等角度与所学系统进行对比分析, 谈谈你的理解
- 格式:
 - ✦ 标题四号加粗、正文小四、中文宋体、英文 Times New Roman、单倍行距)
 - ✦ 建议1-2页, 可超过
- 提交方式:
 - ✦ 2019.12.31前发送到 dasebigdata@163.com