第十三讲 批流融合系统Flink

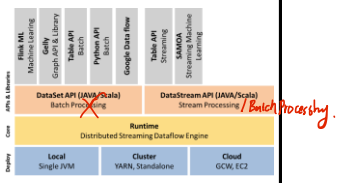
徐 辰
cxu@dase.ecnu.edu.cn
華東師範大學

What is Apache Flink?

2

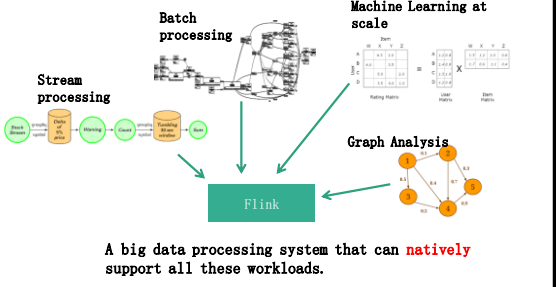
Apache Flink is an open source platform for scalable batch and stream data processing.

- The core of Flink is a distributed streaming dataflow engine.
  - Executing dataflows in parallel on clusters
  - Providing a reliable foundation for various workloads
- **DataSet** and **DataStream** programming abstractions are the foundation for user programs and higher layers



What can I do with it?

3

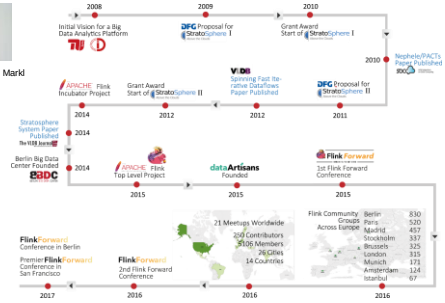


A big data processing system that can natively support all these workloads.

Flink历史

4

Prof. Dr. Volker Markl



MapReduce & Database ++

5

Taken from database Technology

- Declarativity
- Query optimization
- Robust out-of-core

- Iterations
- Advanced Dataflows
- General APIs

Taken from MapReduce technology

- Scalability
- User-defined functions
- Complex data types
- Schema on read

大纲

6

- 设计思想
  - 数据模型
  - 计算模型
  - 迭代模型
- 体系架构
- 工作原理
- 容错机制

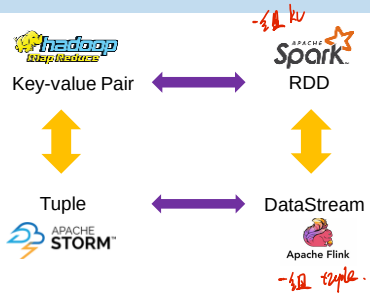
数据模型

7

- 与Storm类似，Flink将输入数据看作是一个不间断的无界的连续数据项序列
- 有所不同的是，Flink将这一系列的数据项抽象成DataStream

数据模型的比较

8



大纲

9

- 设计思想
  - 数据模型
  - 计算模型
  - 迭代模型
- 体系架构
- 工作原理
- 容错机制

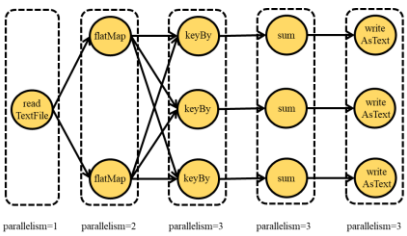
逻辑计算模型

10

```
DataStream<String> source = env.readTextFile("xxx")
DataStream<Tuple2<String,Integer>> count = source
        .flatMap(new FlatMapFunction<>())
        .keyBy(0)
        .sum(1);
count.writeAsText("yyy");
```

物理计算模型

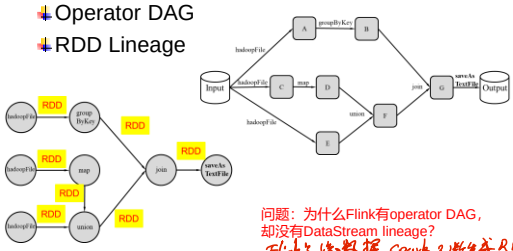
11



回顾：Spark逻辑计算模型

12

- 对于spark，有两种方式描述逻辑计算模型
  - Operator DAG
  - RDD Lineage



问题：为什么Flink有operator DAG，却没有DataStream lineage？

大纲

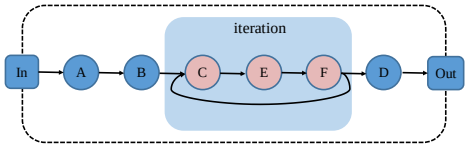
13

- 设计思想
  - 数据模型
  - 计算模型
  - 迭代模型
- 体系架构
- 工作原理
- 容错机制

迭代模型

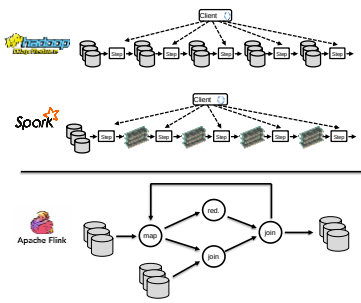
14

- 迭代过程内部必然存在环路
- 将迭代部分整体视为一个算子，计算的过程仍然是DAG（有向无环图）



迭代比较

15



每一迭代步骤(Step)结束时将结果写入HDFS，下一步将该结果再次从HDFS读出迭代次数决定了MapReduce作业的个数

每一迭代步骤 (Step)结束时生成一个RDD，下一步使用RDD再次计算，整个迭代是一个作业

系统提供迭代算子，用户编程时不需要使用for、while等循环，整个迭代是一个作业

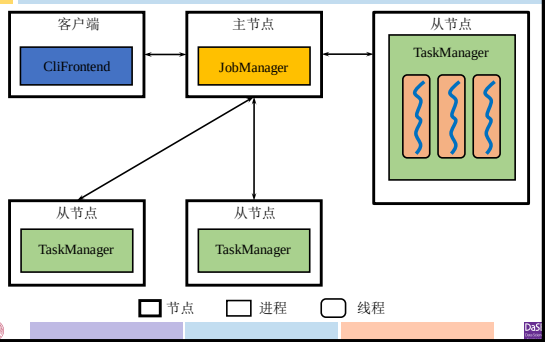
大纲

16

- 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

Flink架构图

17



系统角色

18

- JobManager：作业管理器
  - 用于协调系统的作业执行，包括任务调度，状态快照的协调和故障恢复等
  - Standalone部署方式下，JobManager的进程名为StandaloneSessionClusterEntrypoint

系统角色

19

- TaskManager：任务管理器
  - 将内存抽象成多个TaskSlot，用来执行JobManager分配的任务（Task）
  - 并且负责读取数据、缓存数据以及与其它TaskManager之间进行数据交换
  - Standalone部署方式下，TaskManager的进程名为TaskManagerRunner

系统角色

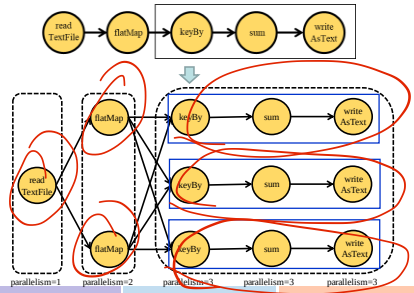
20

- Client: 客户端
  - 用户程序的翻译为逻辑执行图并进行优化
    - 将用户编写的DataStream程序翻译为逻辑执行图，并进行chaining优化
    - 将用户编写的DataSet程序翻译为逻辑执行图，并进行chaining和基于代价的优化
  - 将优化后的逻辑执行图提交到JobManager

Chaining优化

21

- 将某些算子合并到一个Task中执行



与MapReduce/Spark/Storm比较

22

	MapReduce	Storm	Spark	Flink
系统进程	JobTracker	Nimbus	Master	JobManager
	TaskTracker	Supervisor	Worker	TaskManager
	Child	Worker	CoarseGrained ExecutorBackend	
工作线程	/	Executor	Task	Task
任务代码	Task	Task		
基础接口	Map/Reduce	Spout/Bolt	RDD	DataStream/ DataSet

大纲

23

- 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

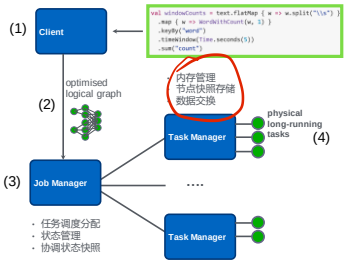
应用程序执行流程

24

1. Client端根据用户编写的Dataflow(包括source、operator和sink等)程序，将其翻译为逻辑执行图并进行优化
2. Client将逻辑执行图提交给JobManager，之后可以保持连接以获取任务的状态信息或者断开连接（detected模式）
3. JobMangaer将逻辑执行图转换为物理执行图，即考虑算子的并行度、进行Task划分等，并将任务分配到各个TaskManager中
4. TaskManager用于执行分配到的Task

应用程序执行流程

25



大纲

26

- 设计思想
- 体系架构
- 工作原理
- 容错机制

工作过程

27

- 数据输入、数据转换、数据输出
- 讨论主题
  - 执行引擎不同Task（运行某个或某些算子）之间如何进行数据交换？
    - 问题：MapReduce/Spark/Storm分别如何进行数据交换？
  - 如何基于执行引擎来实现Dataflow编程模型？

大纲

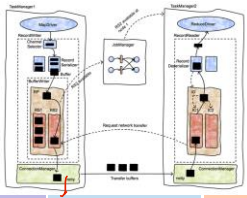
28

- 设计思想
- 体系架构
- 工作原理
  - 非迭代任务间的数据交换
  - 迭代任务内部的数据交换
  - Dataflow模型的实现
  - 关系化Dataflow模型的实现
- 容错机制

Pipeline数据传输

29

- 对于DataStream程序，Flink采用流水线(pipeline)方式进行数据交换，上游的Task将数据存储在buffer中，一旦buffer满了或者超时，就向下游Task进行发送



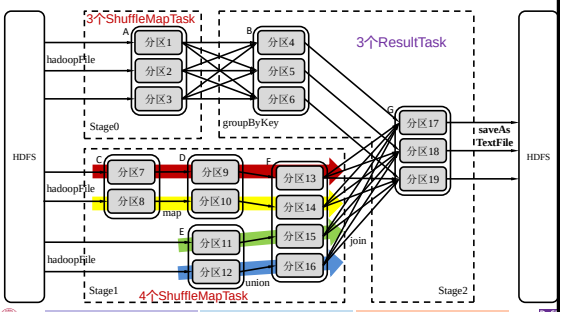
Pipeline in Flink vs. Spark

30

- Flink Pipeline数据交换方式是不同Task之间的数据传输，而Spark Pipeline是指Stage内部同一个Task实现多个不同算子间的数据交换方式
- Spark Pipeline和Flink Chaining类似

回顾：Spark Pipeline

31



Task间的数据交换方式

32

- 阻塞式数据传输
  - 一个Task（运行某个或某些算子）要将所有需要处理的数据计算完，甚至要将结果写入磁盘，才会发送给位于下游Task或被其读取
- 非阻塞式数据传输
  - 一个Task处理一条或一部分数据，通常将计算结果放在缓存里，就会发送给位于下游Task或被其读取

Task间的数据交换方式

33

引擎	阻塞式数据传输	非阻塞式数据传输
Hadoop MapReduce	Map和Reduce任务之间的Shuffle	
Spark	不同Stage中任务之间的Shuffle	
Apache Storm		一次一记录的消息传递机制
Apache Flink		一次一缓冲块的流水线机制

大纲

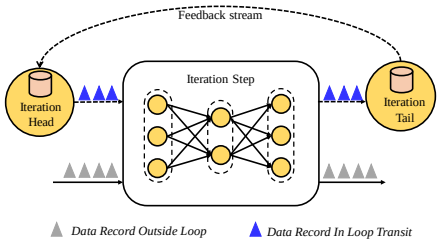
34

- 设计思想
- 体系架构
- 工作原理
  - 非迭代算子间的数据交换
  - 迭代算子内部的数据交换
  - Dataflow模型的实现
  - 关系化Dataflow模型的实现
- 容错机制

流式迭代的数据交换

35

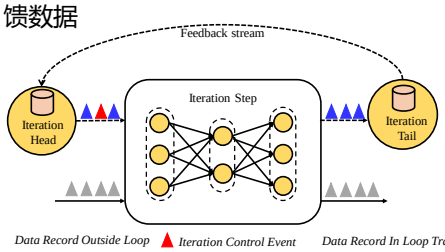
- iterationHead和iterationTail两个特殊的task来实现数据的“循环反馈”



批式迭代的数据交换

36

- 当满足迭代停止条件，IterationHead发出特殊的控制事件，之后IterationTail不再反馈数据



大纲

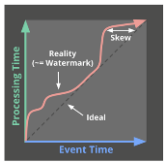
37

- 设计思想
- 体系架构
- 工作原理
  - 非迭代算子间的数据交换
  - 迭代算子内部的数据交换
  - Dataflow模型的实现
  - 关系化Dataflow模型的实现
- 容错机制

Dataflow模型中的重要机制

38

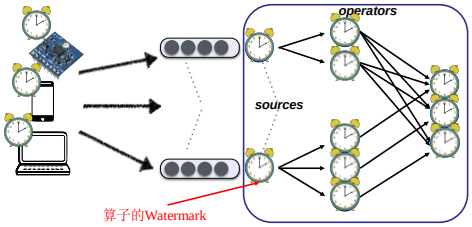
- 事件时间与水位线
  - 水位线所指示的事件时间表示早于该事件时间的记录已经完全被系统观察到
  - 系统根据水位线"认定"当前事件时间域的所处时间
- 窗口与触发器
  - 基于事件时间的窗口
  - 迟到记录的对结果的修正



算子的Watermaker

39

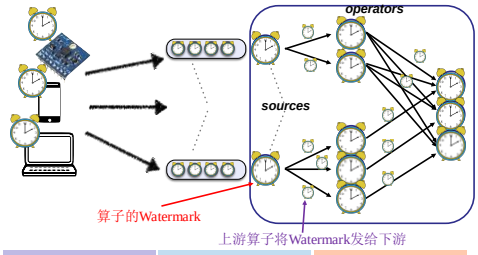
- 每个算子都要维护watermarker，由此确定到底收到了哪个event time之前的记录



算子的Watermaker

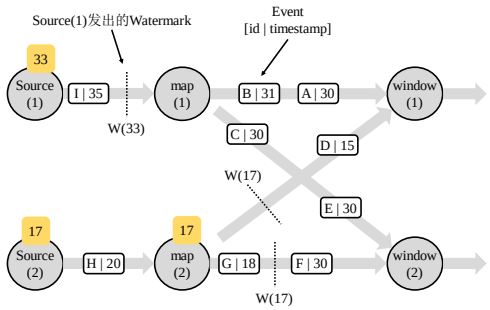
40

- 上游算子将自己的watermarker告诉下游算子，下游算子据此进行watermaker计算



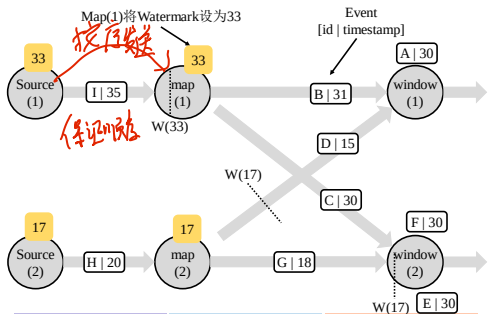
示例：算子的Watermark

41



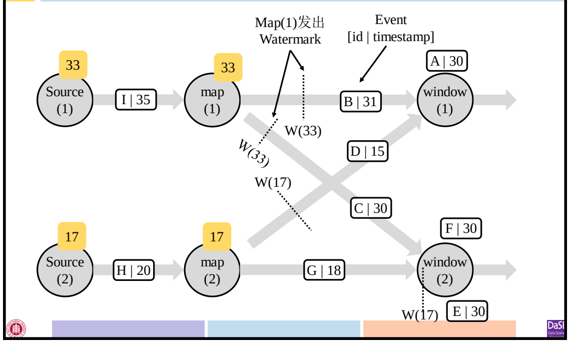
示例：算子的Watermark

42



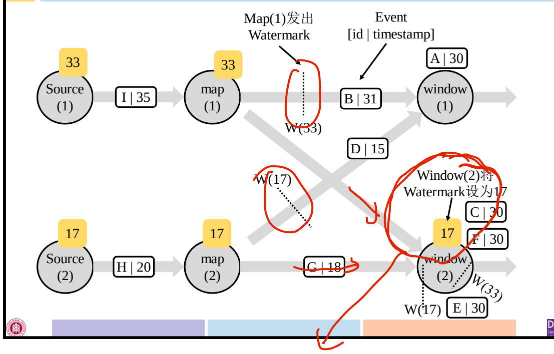
示例：算子的Watermark

43



示例：算子的Watermark

44



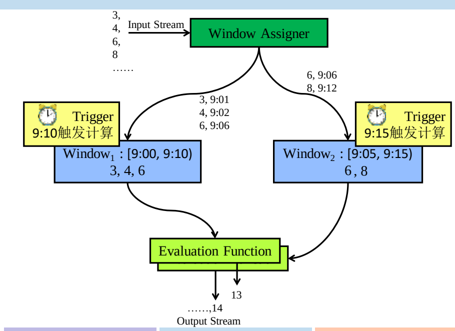
Window操作

45

- Window操作的实现
  - Window Assigner: 负责将元素分配到不同的window
  - Trigger: 触发器，定义什么情况触发计算
- 基于时间的窗口
  - Processing Time Window
  - Event Time Window
- 基于计数的窗口

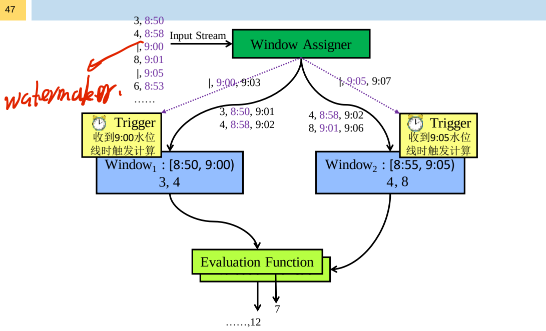
Processing Time Window

46



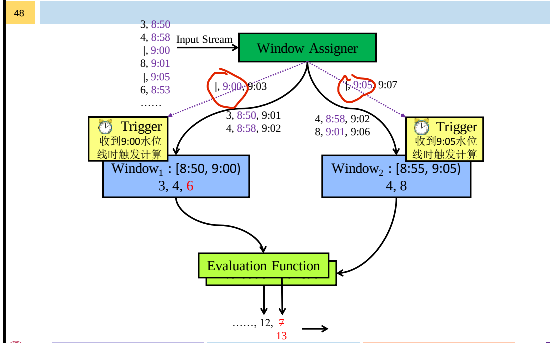
Event Time Window

47



Event Time Window

48



大纲

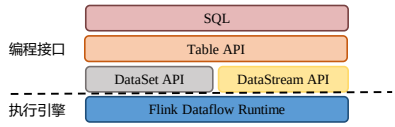
49

- 设计思想
- 体系架构
- 工作原理
  - 非迭代算子间的数据交换
  - 迭代算子内部的数据交换
  - Dataflow模型的实现
  - 关系化Dataflow模型的实现
- 容错机制

Flink API

50

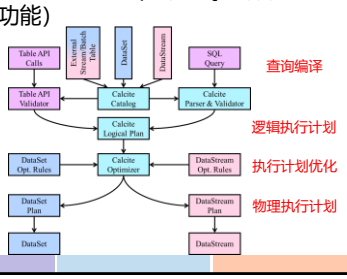
- SQL/Table API是关系型的声明式接口
- DataStream是Dataflow的命令式接口



Relational APIs → DataStream API

51

- 类似于关系数据库的查询处理
  - 利用Apache Calcite（提供SQL解析、翻译、优化等功能）



大纲

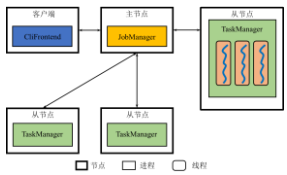
52

- 设计思想
- 体系架构
- 工作原理
- 容错机制

故障类型

53

- JobManager故障
  - 怎么办?
- TaskManager故障
  - 运行了非迭代Task
  - 运行了迭代Task



大纲

54

- 设计思想
- 体系架构
- 工作原理
- 容错机制
  - 状态管理
  - 非迭代计算过程的容错
  - 迭代计算过程的容错

为什么需要状态？

55

- 假如我们进行单词计数操作，需使用聚合操作，并设置触发器根据条件触发计算
- 在触发计算前，聚合算子需要保存<单词，数目>的信息　回忆一下Storm WordCount例子中的CountBolt
  - 聚合算子中编写一个HashMap，一旦该算子所在task发生故障，内存中的HashMap就丢失了
  - 为了支持容错，需要编写程序将HashMap写入磁盘等可靠的存储设备，故障恢复后读取
  - 不同的数据结构都需要编写相应的保存、读取代码☹

状态定义

56

- 状态：系统定义的特殊的数据结构，用于记录需要保存的算子计算结果
  - ValueState<T>：状态保存的是每个key的一个值，可以通过update(T)来更新，T.value()获取
  - ListState<T>：状态保存的是每个key的一个列表，通过add(T)添加数据，Iterable.get()获取
  - ReducingState<T>：状态保存的是关于每个key的经过聚合之后的值列表，通过add(T)添加数据，通过指定的聚合方法获取
  - ……

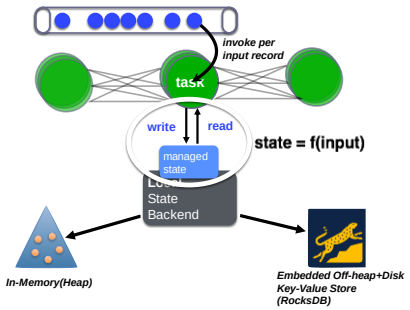
有状态算子 vs. 无状态算子

57

- 状态可以看作算子上的记忆能力，可以保留和已经处理完的输入相关的信息，并对后续输入的处理造成影响。我们将具备记忆能力的算子称为有状态算子，例如聚合操作。
- 与之相反，不具备记忆能力的无状态算子只会考虑到当前处理的元素，不会受到处理完毕的元素的影响，也不会影响到后续待处理的元素。例如，Map就是一种典型的无状态算子。

状态存储

58



状态存储

59

- 状态保存、发生故障后的恢复由系统来负责，而不是由用户程序来负责
  - MemoryStateBackend：将状态存储于本地JVM堆中，因而所能存储状态的大小受限于可用内存的容量（调试程序）
  - RocksDBStateBackend：将状态存储到本地的RocksDB
  - ……

状态快照

60

- 一个DAG中有若干有状态算子，系统为了支持容错需要向一时刻保存所有有状态算子的状态，即状态快照（snapshot）
- 然而，算子可能运行在不同物理机器上TaskManager进程中的task线程，各个物理机器的时钟不可能达到绝对同步

快照与恢复

61

- 为了支持故障恢复，系统可以将数据分为
  - 进入系统并已经成功处理的数据：有状态算子处理这些数据后的状态保存到HDFS或其它可靠存储，作为快照
  - 进入系统而未成功处理的数据
  - 未进入系统的数据
- 故障恢复的方法
  - 将系统中所有算子的状态从快照中读出并重置
  - 重放进入而未成功处理的数据
  - 接着进行处理尚未进入系统的数据

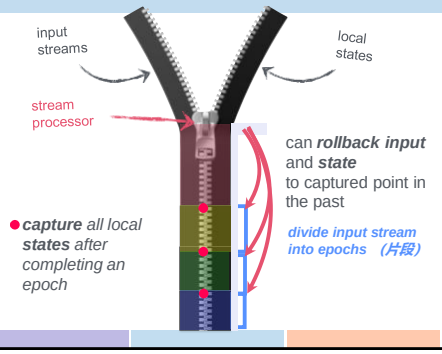
大纲

62

- 设计思想
- 体系架构
- 工作原理
- 容错机制
  - 状态管理
  - 非迭代计算过程的容错
  - 迭代计算过程的容错

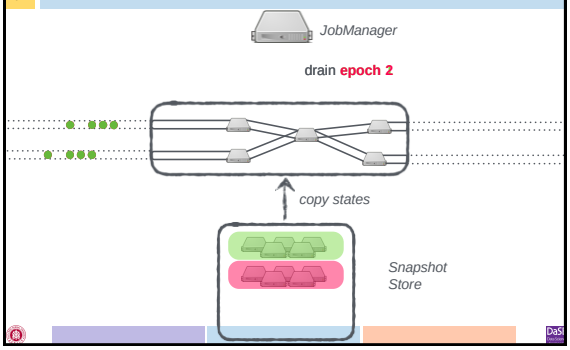
直观的方法

63



Synchronous Snapshots

64



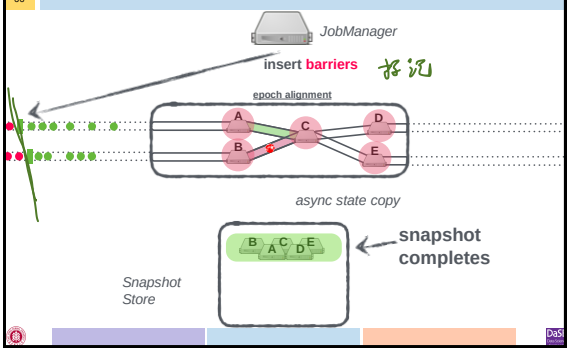
Synchronous Snapshots

65

- 将输入数据划分为一系列的片度，处理一个片段时记录所有算子的状态，处理完一个片段后接着处理下一个片度，实现简单
- 片段1的数据未处理完，片段2的数据无法进入系统，造成延迟高
- 此时实际上已经不是连续处理方式了，而是退化为微批处理方式

Asynchronous Snapshots

66



做到了连续处理.

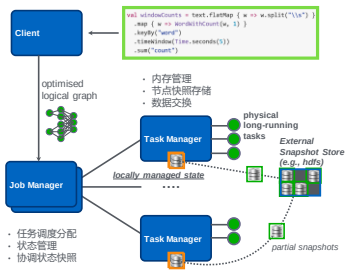
Asynchronous Snapshots

67

- 在输入数据中加入barrier，作为划分片度的标记
- 处理片段1时容许片段2的数据进入系统，与Synchronous Snapshots相比延迟低
- 当某一算子接收多路输入时，需要进行片段对齐(epoch alignment)，避免该算子保存的状态处理到下一个片段的数据，因此快照中所存储的状态与Synchronous Snapshots语义上等价

全局的视角

68



大纲

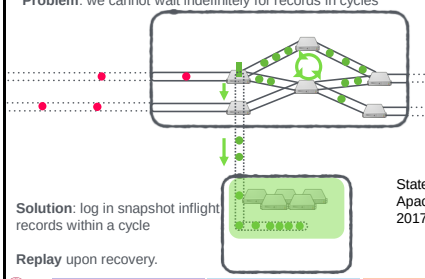
69

- 设计思想
- 体系架构
- 工作原理
- 容错机制
  - 状态管理
  - 非迭代计算过程的容错
  - 迭代计算过程的容错

Asynchronous Snapshots (cycles)

70

Problem: we cannot wait indefinitely for records in cycles



Solution: log in snapshot inflight records within a cycle

Replay upon recovery.

State Management in Apache Flink. *PVLDB*, 2017.

Chandy-Lamport 算法

71

- Chandy-Lamport 算法: 记录分布式系统全局一致状态的算法
- Flink实现的Asynchronous Snapshots是Chandy-Lamport 算法一个变种
- 目前，迭代计算的Asynchronous Snapshots (cycles)实现代码尚未并入正式版本

课后阅读

72

- 分布式系统概念与设计，George Coulouris等著，金蓓弘等译
  - 第14章 14.5
- 论文
  - Carbone, P., Ewen, S., Haridi, S., Katsifodimos, A., Markl, V., & Tzoumas, K. (2015). Apache Flink: Unified Stream and Batch Processing in a Single Engine. IEEE Data Eng. Bull., 38(4), 28–38.
  - Carbone, P., Ewen, S., Richter, S., & Gyula, F. (2017). State Management in Apache Flink. PVLDB, 10(20), 1718–1729.

本讲小结

73

- 设计思想
- 体系架构
- 工作原理
- 容错机制

谢谢！Q&A

Apache Flink

Credits

74

- Some slides from the presenter
  - Fabian Hueske
  - Ufuc Celebi
  - Paris Carbone
  - Volker Markl