第十二讲 批流融合概述

徐 辰
cxu@dase.ecnu.edu.cn
華東師範大學

大纲

2

- 批流融合的背景
  - 融合的需求
  - Lambda架构及其局限性
- 批流处理的统一性
- Dataflow统一编程模型
- 关系化Dataflow编程模型
- 一体化执行引擎

批处理 vs. 流计算

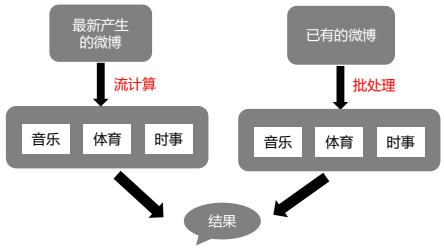
3

- 批处理系统适合处理大批量数据、实时性要求不高的场景
- 流计算系统适合处理快速产生的数据、实时性要求高的场景
- 但是，同一场景可能既有大批量数据、又有快速产生的数据，某些模块实时性要求高，某些模块实时性要求低

热点话题统计

4

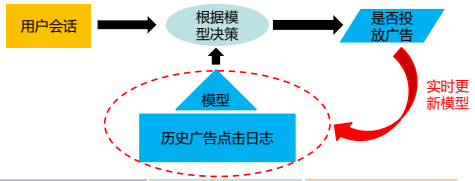
- 根据已有的微博（如近一周）和最新产生的微博统计不同话题数量来判断热门程度



广告投放

5

- 模型训练：根据大量历史数据离线训练
- 模型预测：根据已有模型和用户数据决策，之后又变成历史数据更新模型
- 问题：模型怎样实时或近实时更新?



Streaming & Batch Processing

6

- Many environments require processing same data in live streaming as well as batch post processing
- Existing framework cannot do both
  - Either do stream processing of 100s of MB/s with low latency
  - Or do batch processing of TBs / PBs of data with high latency

Volume和Velocity

7

- 批处理系统：应对数据量大volume
- 流计算系统：应对数据产生速度快velocity
- 问题：如何同时应对volume和velocity?

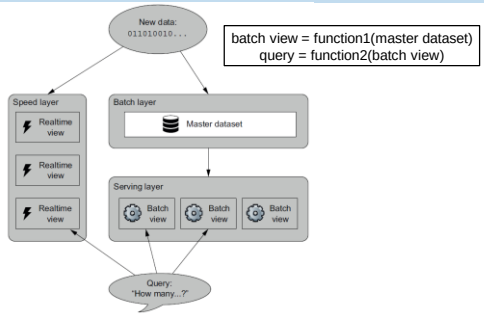
大纲

8

- 批流融合的背景
  - 融合的需求
  - Lambda架构及其局限性
- 批流处理的统一性
- Dataflow统一编程模型
- 关系化Dataflow编程模型
- 一体化执行引擎

Lambda架构

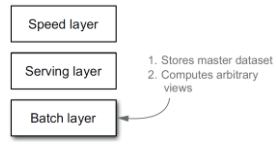
9



批处理层Batch Layer

10

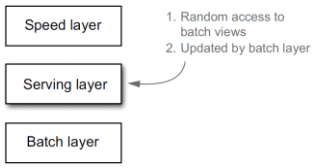
- 功能：全量计算
  - 存储MasterDataset
    - 不可变、持续增长的数据集
  - 针对这个MasterDataset进行预运算
- 实现batch view = function1(master dataset)



服务层Serving Layer

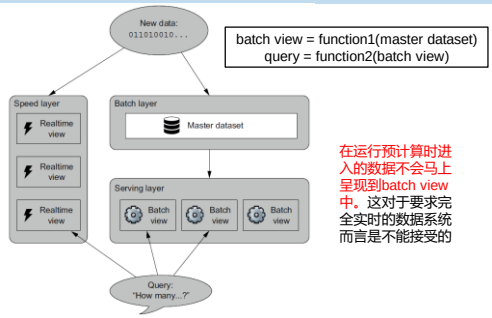
11

- 功能：对batch view进行操作，从而为最终的实时查询提供支撑
  - 支持对batch view的随机访问
  - 根据batch layer的计算结果更新batch view



Lambda架构

12

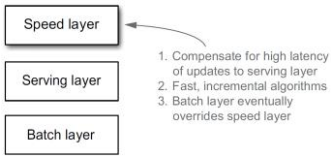


在运行预计算时进入的数据不会马上呈现到batch view中。这对于要求完全实时的数据系统而言是不能接受的

加速层Speed Layer

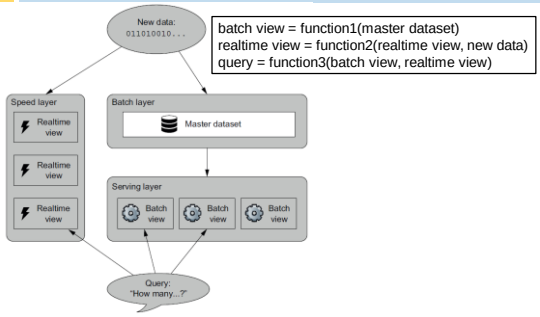
13

- 功能：增量计算
  - 只处理最近的数据，而不是所有数据
  - 更新realtime view
    = function3(realtime view, new data)



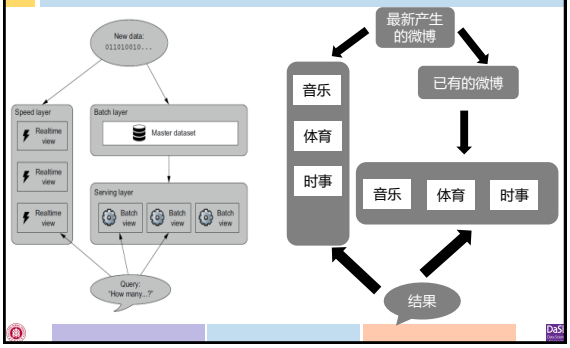
Lambda架构

14



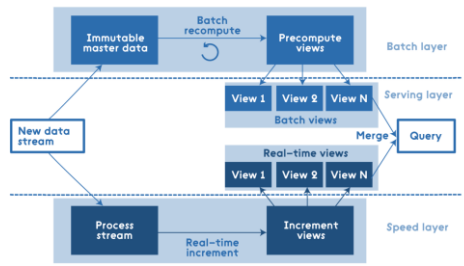
热点话题统计

15



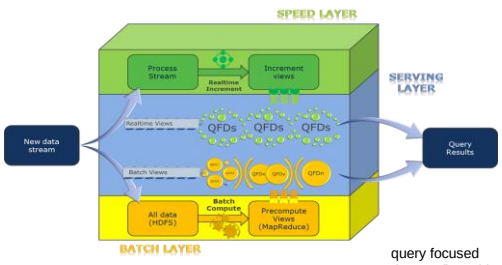
Lambda架构:其它表示

16



Lambda架构:其它表示

17

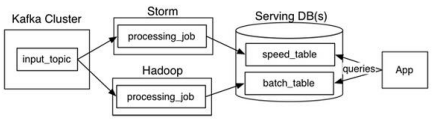


query focused datasets" or qfds

系统选型

18

- Lambda架构只是个架构
- 批处理层：批处理系统
- 加速层：流计算系统
- 服务层：数据库



Lambda架构优缺点

19

- 优点：
  - 平衡了重新计算高开销和需求低延迟的矛盾
- 缺点：
  - 开发复杂：需要将所有的算法实现两次，批处理系统和实时系统分开编程，还要求查询得到的是两个系统结果的合并
    - Different programming models
    - Double the implementation effort
    - Double the number of bugs
  - 运维复杂：同时维护两套执行引擎

融合的思路

20

- 如何将批流系统分开编程变为统一编程？
  - Google Dataflow
  - Apache Beam
- 如何将两套执行引擎变为一体化的批流融合执行引擎并支持统一编程？
  - Spark Structured Streaming
  - Apache Flink

大纲

21

- 批流融合的背景
- 批流处理的统一性
  - 有界与无界数据集
  - 窗口操作
  - 时间语义
- Dataflow统一编程模型
- 关系化Dataflow编程模型
- 一体化执行引擎

批处理 vs. 流计算引擎

22

- 问题：批处理引擎仅能处理批数据，流计算引擎仅能处理流数据，对吗？

执行引擎 \ 数据	批数据	流数据
Batch	✓	?
Streaming	?	✓

有界 vs. 无界数据集

23

- 数据：bounded/unbounded
  - 有界数据集意味着系统处理的数据是有限的，对应于我们之前所说的批数据
  - 无界数据集意味着系统处理的是无限的数据，对应于我们之前所说的流数据
- 处理引擎：Batch/Streaming

执行引擎 \ 数据	bounded	unbounded
Batch	✓	?
Streaming	?	✓

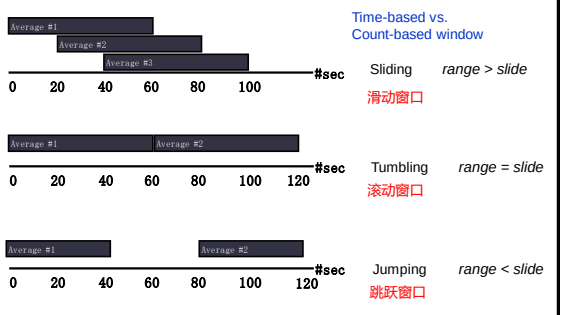
大纲

24

- 批流融合的背景
- 批流处理的统一性
  - 有界与无界数据集
  - 窗口操作
  - 时间语义
- Dataflow统一编程模型
- 关系化Dataflow编程模型
- 一体化执行引擎

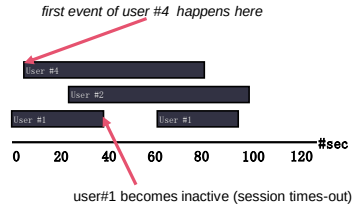
Sliding Windows

25



Session Windows

26



窗口与数据集

27

- 窗口操作可以把数据集切分为有限的数据片以便于针对该数据片进行处理
  - 对于无界数据集来说，诸如聚合（aggregate）等操作需要窗口来定义边界，例如最近1分钟等；另一些则不需要（如filter，map等）
  - 对于有界数据集来说，窗口是可选的，或者我们认为是有一个全局窗口（Global Window）涵盖有界数据集中的所有数据

大纲

28

- 批流融合的背景
- 批流处理的统一性
  - 有界与无界数据集
  - 窗口操作
  - 时间语义
- Dataflow统一编程模型
- 关系化Dataflow编程模型
- 一体化执行引擎

时间域

29

- 我们将一条数据记录视为一个事件
  - 事件时间 Event Time：该事件实际发生的时间
    - 当该事件发生时，其所在系统（可能是传感器等数据源）的当前时间
  - 处理时间 Processing Time：，一个事件在数据处理的过程中被数据处理系统观察到的时间
    - 当该事件被数据处理系统处理时，数据处理系统的当前时间
  - 一个记录的事件时间是永远不变的，但是处理时间随着该记录在系统中被各个节点处理时而持续变化

有界数据集的时间域

30

- 例子：系统日志分析
  - 假设Hadoop平台从学期开始时一直运行，产生了大量的日志文件
    - 日志中的每一行开头的时间（事件时间）
    - 现在（处理时间）统计error的数量
- 需求：统计每周出现各类故障及其数量
  - 目前为止，我们所学的窗口是基于处理时间的，那么如何设置该需求的窗口？
    - 需要根据事件时间来确定窗口

无界数据集的时间域

31

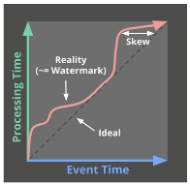
- 例子：传感器温度值统计
  - 传感器周期性(1-2s)采集温度信息，产生(time, temperature)的记录并向数据处理系统发送
  - 半小时统计一次最近1小时内平均值：系统当前时间9:00:00，统计范围(8:00:00-8:59:59)
- 假如：当前时间9:20:00(处理时间),传感器8:45:01(事件时间)产生的记录由于网络阻塞等原因才到达
  - 9:00:00窗口中记录不完整，统计结果不准确
  - 到底还有没有其它记录没有到达?

水位线(watermark)

32

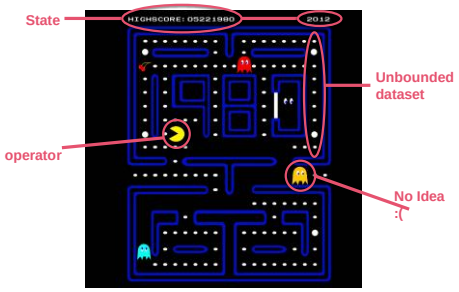
- 水位线所指示的事件时间表示早于该事件时间的记录已经完全被系统观察到
- 系统根据水位线“认定”当前事件时间域的所处时间

水位线是由系统根据预定义或用户指定的启发式规则生成的，因此“认定”实际上只是一种猜测



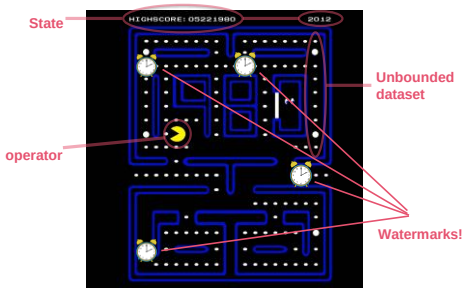
举个栗子

33



还是这个栗子

34



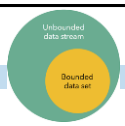
大纲

35

- 批流融合的背景
- 批流处理的统一性
- Dataflow统一编程模型
- 关系化Dataflow编程模型
- 一体化执行引擎

输入数据特点

36



- 无界(Unbounded)：
  - 输入数据视为无界数据集，认为数据记录在系统执行计算过程中不断地动态到达并永无止境
  - 有界数据集仅仅是无界数据集的一个特例
- 乱序(Out-of-order)：
  - 由于数据记录从产生到进入系统通常会产生延迟(Delay)，而不同记录产生的延迟可能不同，所以数据记录的原始顺序和进入系统的处理顺序可能不一致

WWWH模型

37

- *What* results are calculated?
  - 计算什么结果? (read, map, reduce) 操作原语
- *Where* in event time are results calculated?
  - 在哪儿切分数据? (event time window) 窗口表达
- *When* in processing time are results materialized?
  - 什么时候计算数据? (triggers) 触发器
- *How* do refinements of results relate?
  - 如何修正相关的数据? (Accumulation) 结果修正

大纲

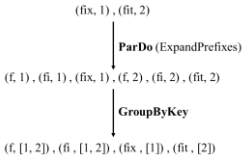
38

- 批流融合的背景
- 批流处理的统一性
- Dataflow统一编程模型
  - 操作原语
  - 窗口表达
  - 触发器
  - 结果修正
- 关系化Dataflow编程模型
- 一体化执行引擎

操作原语

39

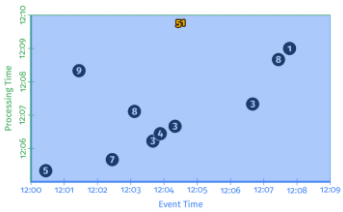
- ParDo: 用于进行通用的并行化处理
  - 该操作将每个输入元素（这个元素本身有可能是一个有限的集合）都使用一个用户定义函数（UDF）进行处理，获得0或多个输出元素
- GroupByKey: 用来按键把元素重新分组



What results are calculated?

40

```
PCollection<KV<String, Integer>> scores = input
    .apply(Sum.integersPerKey());
```



大纲

41

- 批流融合的背景
- 批流处理的统一性
- Dataflow统一编程模型
  - 操作原语
  - 窗口表达
  - 触发器
  - 结果修正
- 关系化Dataflow编程模型
- 一体化执行引擎

无界数据集与窗口

42

- 为了支持无界数据集的处理，我们需要结合窗口操作重新定义GroupByKey操作，即表达GroupByKeyAndWindow操作
  - 窗口分配AssignWindows ()：把元素分配到0或多个窗口中去
  - 窗口合并MergeWindows()：这个操作在分组时合并窗口
- 元组(key, value, event_time, window)

滑动窗口的表达

43

(k, v_1, 12:00, [0, ∞)) , (k, v_2, 12:01, [0, ∞))

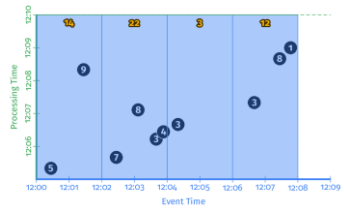
↓ **AssignWindows** (Sliding(2m, 1m))

(k, v_1, 12:00, [11:59, 12:01)),
(k, v_1, 12:00, [12:00, 12:02)),
(k, v_2, 12:01, [12:00, 12:02)),
(k, v_2, 12:01, [12:01, 12:03))

Where in event time?

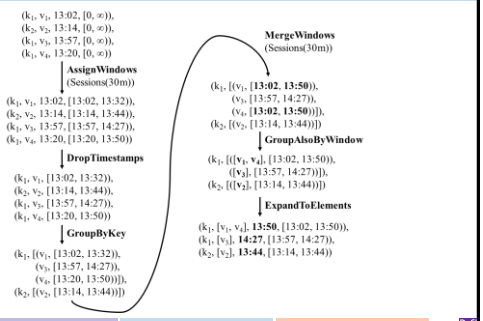
44

```
PCollection<KV<String, Integer>> scores = input
    .apply(Window.into(FixedWindows.of(Duration.standardMinutes(2))))
    .apply(Sum.integersPerKey());
```



会话窗口的表达

45



大纲

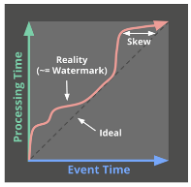
46

- 批流融合的背景
- 批流处理的统一性
- Dataflow统一编程模型
  - 操作原语
  - 窗口表达
  - 触发器
  - 结果修正
- 关系化Dataflow编程模型
- 一体化执行引擎

触发器

47

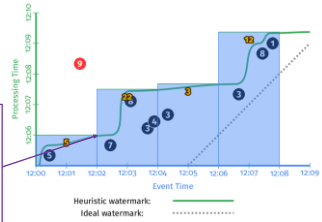
- 在某一处理时间决定处理窗口的聚合结果并输出
- 窗口的语义要求根据事件时间
- 触发器需要利用水位线



When in processing time?

48

```
PCollection<KV<String, Integer>> scores = input
    .apply(Window.into(FixedWindows.of(Duration.standardMinutes(2))
        .triggering(AtWatermark()))
    .apply(Sum.integersPerKey());
```



系统处理时间为12:06时，水位线指示的事件时间为12:02，“认定”12:02之前的记录都已经收到了，决定触发窗口计算

水位线过慢

49

- 水位线本质上是对事件时间的猜测，可能
  - 过慢：窗口操作结束前，需要提前触发窗口操作并输出结果
  - 例如：再定义一个触发器，每隔1min（处理时间）触发窗口计算

When in processing time?

50

```
PCollection<KV<String, Integer>> scores = input
    .apply(Window.into(FixedWindows.of(Duration.standardMinutes(2))
        .triggering(AtWatermark()
            .withEarlyFirings(AtPeriod(Duration.standardMinutes(1)))
            .withLateFirings(AtCount(1)))
        .accumulatingFiredPanes())
    .apply(Sum.integersPerKey());
```

系统处理时间12:06和12:07，窗口[12:02-12:04]尚不满足水位线触发条件，但是定义了earlyFiring触发器，表示"提前"触发窗口计算

大纲

51

- 批流融合的背景
- 批流处理的统一性
- Dataflow统一编程模型
  - 操作原语
  - 窗口表达
  - 触发器
  - 结果修正
- 关系化Dataflow编程模型
- 一体化执行引擎

水位线过快

52

- 水位线本质上是对事件时间的猜测，可能
  - 过快：存在"迟到"的数据，需要之后修正原有窗口计算结果，再次触发窗口操作并输出结果
    - 抛弃(Discarding)：触发器一旦触发后，窗口内容即被抛弃，之后窗口计算的结果和之前的结果不存在任何相关性
    - 累积(Accumulating)：触发器触发后，窗口内容被完整并持久化保留到系统状态，而之后的计算结果成为对之前结果的一个修正版本

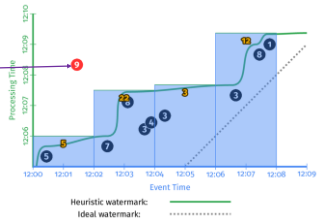
How do refinements relate?

53

```
PCollection<KV<String, Integer>> scores = input
    .apply(Window.into(FixedWindows.of(Duration.standardMinutes(2))
        .triggering(AtWatermark()))
    .apply(Sum.integersPerKey());
```

Discarding

窗口12:00-12:02的结果是5，与迟到的9没有任何相关性

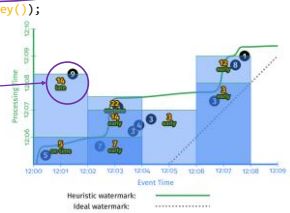


How do refinements relate?

54

```
PCollection<KV<String, Integer>> scores = input
    .apply(Window.into(FixedWindows.of(Duration.standardMinutes(2))
        .triggering(AtWatermark()
            .withEarlyFirings(AtPeriod(Duration.standardMinutes(1)))
            .withLateFirings(AtCount(1)))
        .accumulatingFiredPanes())
    .apply(Sum.integersPerKey());
```

属于窗口[12:00-12:02]的记录9迟到了，accumulate表示要将其与窗口原来的值进行累加，此外定义了lateFiring触发器每遇到一个迟到的记录就输出一次新的结果



小故事：dataflow machine

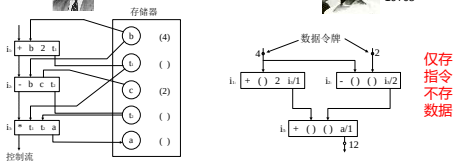
55

- Dataflow machine：指令级
- Dataflow编程模型：算子级

John von Neumann 1946

表达式
a = (b + 2) * (b - c)

Jack Dennis 1970s



控制驱动方式
Control flow

数据驱动方式
Data flow

大纲

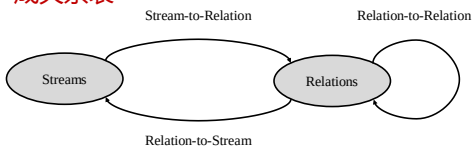
56

- 批流融合的背景
- 批流处理的统一性
- Dataflow统一编程模型
- 关系化Dataflow编程模型
- 一体化执行引擎

Dataflow编程模型的关系化

57

- Dataflow编程模型认为输入数据是一系列元组组成的序列
- 从关系模型的角度看，这一系列的元组构成关系表



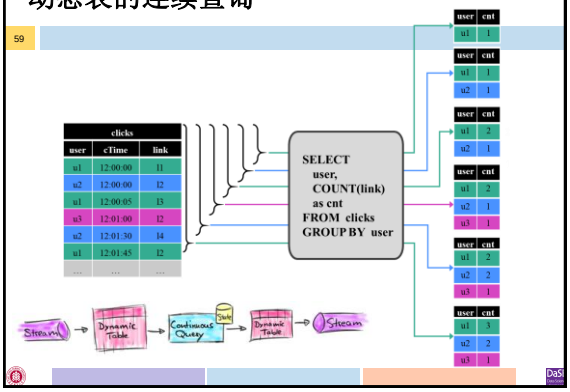
The CQL continuous query language: semantic foundations and query execution. VLDB J 2006.

无界数据集→动态表

58

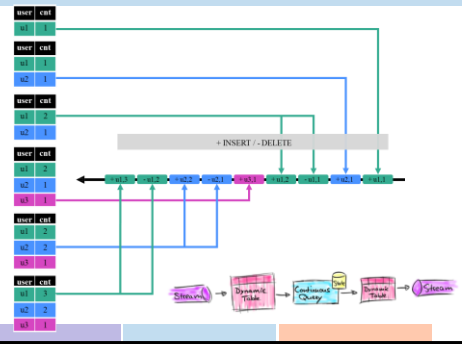
动态表的连续查询

59



动态表→无界数据集

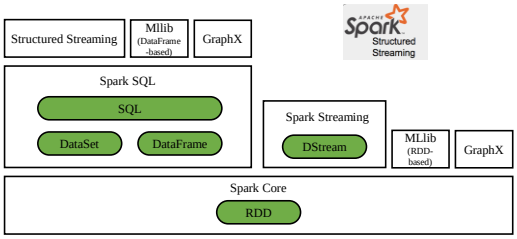
60



Structured Streaming

61

- Structured Streaming实现了关系化的Dataflow编程模型



大纲

62

- 批流融合的背景
- 批流处理的统一性
- Dataflow统一编程模型
- 关系化Dataflow编程模型
- 一体化执行引擎

融合的思路

63

- 如何将批流系统分开编程变为统一编程?
  - Google Dataflow
  - Apache Beam
- 如何将两套执行引擎变为一体化的批流融合执行引擎并支持统一编程?
  - Spark Structured Streaming
  - Apache Flink

执行引擎一体化思路

64

- 当前通用的核心执行引擎是批处理引擎或流计算引擎，因此一体化执行引擎需要选择其中一种作为核心
  - 以批处理为核心：该引擎利于处理有界数据集，需要解决的问题是如何基于批处理来处理无界数据集
  - 以流计算为核心：该引擎利于处理无界数据集，需要解决的问题是如何基于流计算处理有界数据集

大纲

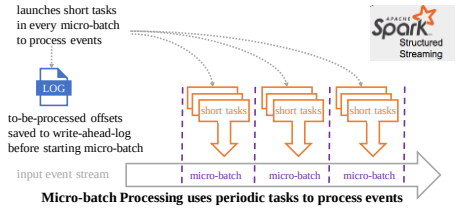
65

- 批流融合的背景
- 批流处理的统一性
- Dataflow统一编程模型
- 关系化Dataflow编程模型
- 一体化执行引擎
  - 以批处理为核心
  - 以流计算为核心

以批处理为核心：微批处理

66

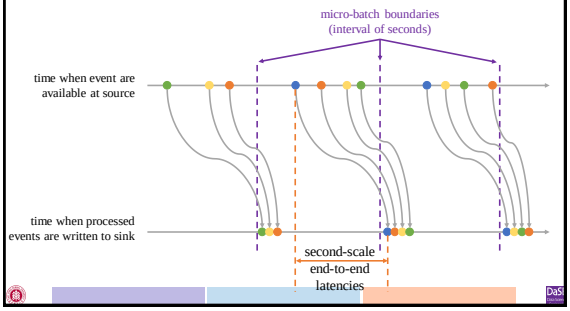
- 将无界数据集划分为小批量数据，不断地启动短作业来处理这些小批量数据
- 先启动的作业必须执行完，才启动新作业



微批处理的延迟

67

- 串行执行短作业的方式带来延迟在秒级



大纲

68

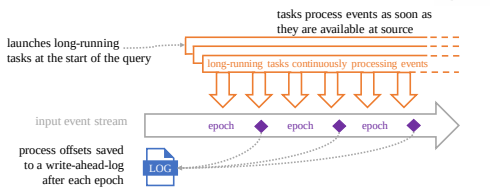
- 批流融合的背景
- 批流处理的统一性
- Dataflow统一编程模型
- 关系化Dataflow编程模型
- 一体化执行引擎
  - 以批处理为核心：微批处理
  - 以流计算为核心：连续处理

以流计算为核心：连续处理

69

- 对于有界数据集来说，相当于系统接收一定量的记录之后就不再接收新的记录了
- 启动一个长期运行的作业

Apache Flink

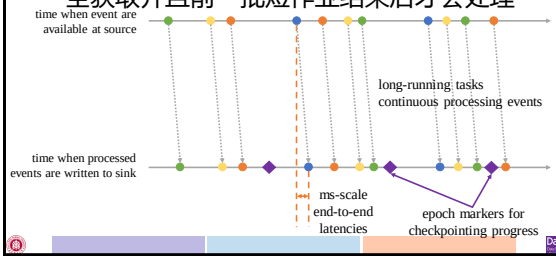


Continuous Processing uses long-running tasks to continuously process events

连续处理的延迟

70

- 记录一旦进入系统将立即得到处理，而不是像微批处理方式中那样等到一批数据完全获取并且前一批短作业结束后才会处理



课后阅读

71

- 论文
  - Akidau, T., Bradshaw, R., Chambers, C., Chernyak, S., Fer Andez-Moctezuma, R. J., Lax, R., …, S. W. (2015). The Dataflow Model: A Practical Approach to Balancing Correctness, Latency, and Cost in Massive-Scale, Unbounded, Out-of-Order Data Processing. PVLDB, 8(12), 1792–1803.
  - Armbrust, M., Das, T., & Torres, J. (2018). Structured Streaming: A Declarative API for Real-Time Applications in Apache Spark. In SIGMOD Conference (pp. 465–476).

本讲小结

72

- 批流融合的背景
- 批流处理的统一性
- Dataflow统一编程模型
- 关系化Dataflow编程模型
- 一体化执行引擎

Apache Beam

Spark Structured Streaming

Apache Flink

谢谢！Q&A