Chapter 2. Stream Processing Fundamentals
（流处理的基本原理）

介绍流处理的基本概念及其框架的要求

1. **一、Introduction to Dataflow Programming**

   (数据流编程介绍)

   前言

   先介绍一下数据流背景和术语

   1.1. Dataflow Graphs

   （数据流模型）

   1.至少一个 data source 和一个 data sink

   2.逻辑图：节点称之为算子，边表示依赖性

物理执行图：node 代表 task，逻辑图中一个，算子可以代表多个并行 task，如下两张图所示

1.1.1. 图 2-1 data flow 模型逻辑执行图

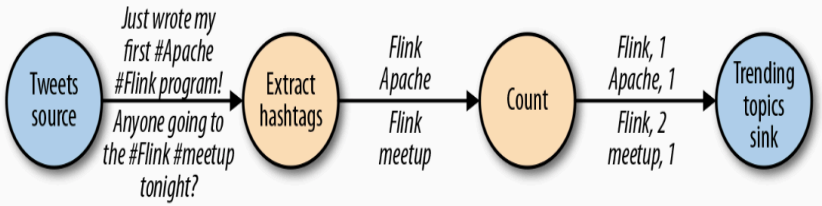

Figure 2-1. A logical dataflow graph to continuously count hashtags (nodes represent operators and edges denote data dependencies)

1.1.2. 图 2-2 data flow 模型物理执行

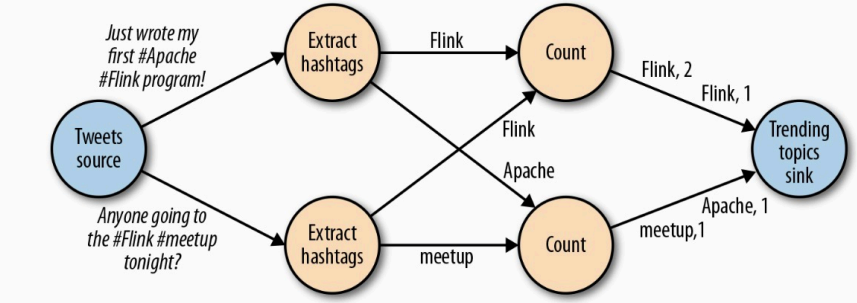

Figure 2-2. A physical dataflow plan for counting hashtags (nodes represent tasks)

1.2. Data Parallelism and Task Parallelism

（数据并行性和任务并行性）

1.数据并行性：可以对数据分区达到并行处理，将负载分散在 多个节点

2.任务并行性：可以通过算子的 task 并行，这样可以更好的使用集群资源

1.3. Data Exchange Strategies

（数据交换策略：定义了如何将数据分配给物理执行计划中 task）

1.forward strategy

前向策略：发送 task 和接收 task 在同一台机器，可避免网络传输

2.broadcast strategy

广播策略：把每条数据项发送给 operator 的所有并行 task，涉及数据复制和网络传输，代价昂贵

3.key-based strategy

key 分区策略：根据 key 将相同的数据分配给同一个任务

4.random strategy

随机策略：将数据随机分配给 operator 的并行 task，目的在于均分分散数据

1.3.1. 图 2-3 数据交换策略

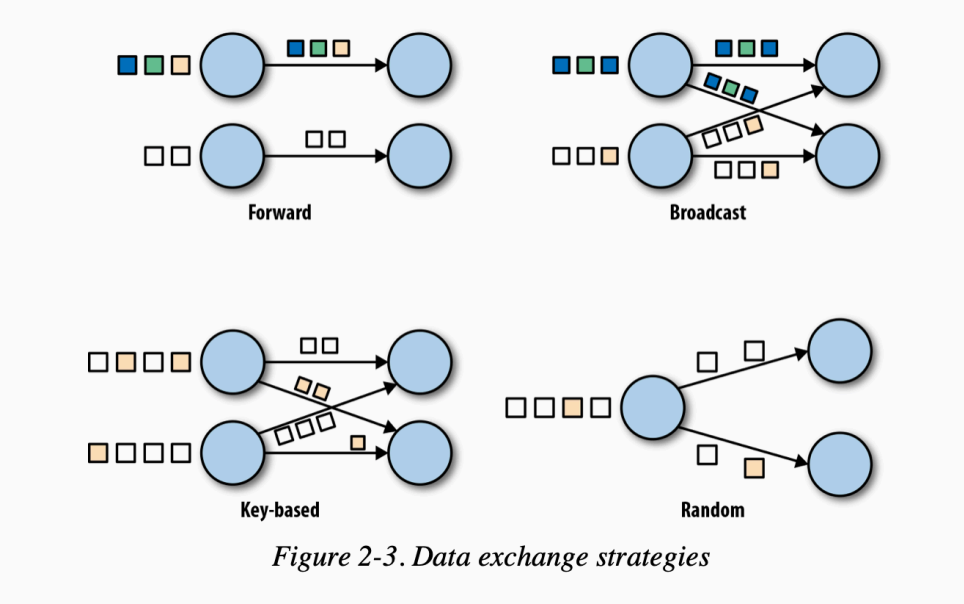

Figure 2-3. Data exchange strategies

2. **二、Processing Streams in Parallel**

（并行处理数据流）

前言

数据流：是一个潜在无限的事件序列

本节将学习并行处理无界流

2.1. Latency and Throughput (延迟和吞吐)

1.Latency 的含义：

接收数据到看到处理结果所花的时间，咖啡店排队例子，从排队到喝上咖啡的时间，延迟分为平均延迟、最大延迟、百分位数延迟，平均延迟会掩盖延迟的分布。

2.Throughput

a. 含义：衡量系统的处理能力，也就是每个时间单位可以处理多少事件,处理速度取决于到达速度，吞吐量低并不一定是性能低，流处理系统中更关心峰值处理能力

b.峰值吞吐量：系统处于最大负载时的性能极限，也就是摄入数据的速率使系统的资源充分被利用，当系统资源被充分利用，就需要缓冲事件，这种情况叫做背压，有不同的不理策略

3.Latency Versus Throughput

延迟和吞吐量不是独立指标，事件在管道中处理的时间过长，就无法轻易的确保吞吐量，相反如果系统的容量很小，事件将会被缓冲，需要等待事件被处理，影响延迟并进而影响吞吐量的一个因素是事件被处理时所

花费的时间，举例来说，圣诞节咖啡师需要在杯子上画一个圣诞老人，这样单个事件所花费的事件比平时就会有延长，

降低延迟提高吞吐量的方法：降低延迟可以提高吞吐量，增加并行性也是降低延迟提高吞吐量的一种方式

2.2. Operations on Data Streams（数据流操作）

分为有状态的算子和无状态的算子

1.Data ingestion and data egress

数据源（source）：实现数据提取逻辑的运算符，例如 tcp、文件、传感器、kafka topic

数据接收器（sink）：实现数据输出的算子，例如文件、数据库、消息对列

2.Transformation operations

可以将流合并及拆分修改逻辑流程图

3.Rolling aggregations

滚动聚合是针对每个输入事件不断更新的聚合， 聚合操作是有状态的，并将 当前状态 和 传入事件组合在一起以生成更新的聚合值。 请注意，为了能够有效地将当前状态与事件组合并产生单个值，聚合函数必须是关联的和可交换的。 否则，操作员将必须存储完整的流历史记录

例如：max、sum、min

2.3. Window operations

前言：

a.window 的特点：

Transformations and rolling aggregations 是每一次处理一个事件，针对当前的单个事件进行转换或者更新状态，除此之外还有一些操作必须收集和缓冲记录才可以进行计算，例如：streaming join operation、holistic aggregate(整体聚合)，为了在无界流上有效的评估这些函数，需要限制这些操作所维护的数据量

b.window 中的桶

窗口操作不断地从无限制的事件流中创建称为桶的有限事件集，让我们对这些有限集进行计算。通常会根据数据属性或时间将事件分配给存储桶。

c.window 中的分桶测略和计算频率

为了正确定义窗口运算符的语义，我们需要确定事件如何分配给存储桶以及窗口产生结果的频率。 Windows 的行为由一组策略定义

窗口策略：决定何时创建桶以及如何将数据分配

触发条件：决定将数据分发到计算函数，计算函数的策略可以基于时间、计数、数据属性

1.Tumbling windows（翻转窗口）

将数据分配到固定大小的非重叠桶，当窗口边界经过时，将数据发送给计算函数处理，可以基于条数也可以基于时间，例如：每 10 条计算一次或者每 10min 计算一次

2.Sliding windows

滑动窗口，将事件分配给固定大小的重叠桶中，一条记录可能属于多个桶重叠桶

3.Session windows（会话窗口）

a.概念：

Session windows 根据 Session gap 的值对事件进行分组，会话间隔值定义了不活动的时间以认为会话已关闭，仅定义了时间差，数据产生连续性是取决于数据源的

b.例如：

分析一个用户购物操作，目的在于将源于同一时期的数据分组到一起，指定 session gap 的值（类似超时时间，是前一条数据和后一条数据的时间差，如果小于 seession gap 就会被分入到相同的桶中），小于的被分为一个桶，大于的被分在另外的桶，整个流被切为多个 session（桶），

c.session windows 的内部合并机制

session window 是为每一个进入的事件创建一个窗口，窗口的边界是 [timestamp, timestamp+sessionGap) 对于每一个窗口在出发前都会判断是否有重叠，如果有重叠就会把窗口合并最终形成一个 session window

4. 分区窗口

当一个流被分区为多个逻辑流时，对于每一个分区中的 window 策略，不同分区是互不影响的

2.3.1. 图 2-7 基于计数和基于时间的 tubling window

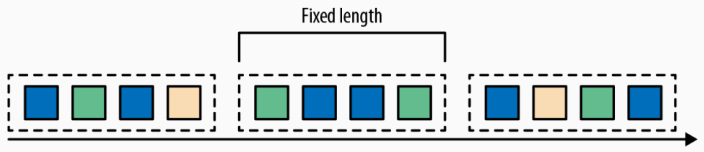

Figure 2-6. Count-based tumbling window

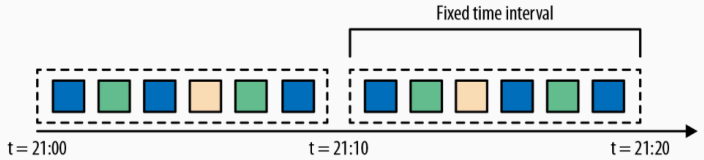

Figure 2-7. Time-based tumbling window

2.3.2. 图 2-8 基于计数的 sliding window　　4 个固定长度-3 个滑动长度

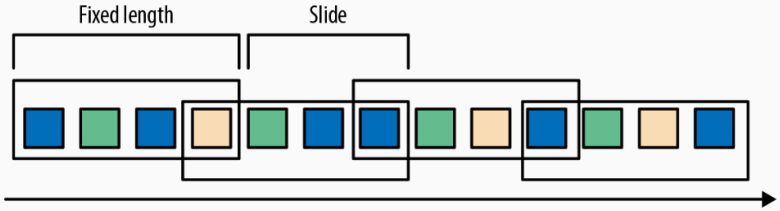

Figure 2-8. Sliding count-based window with a length of four events and a slide of three events

2.3.3. 图 2-9 session window

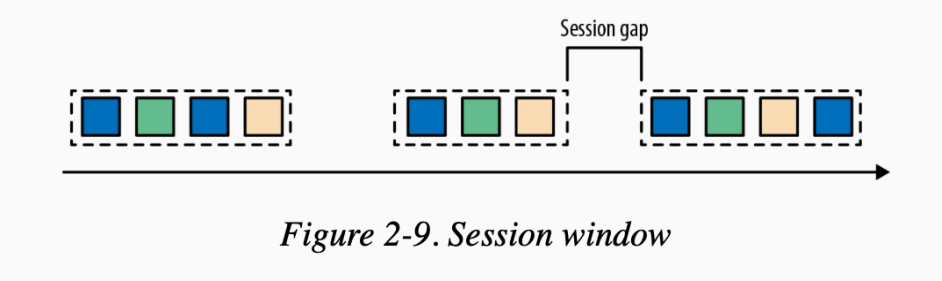

Figure 2-9. Session window

2.3.4. 图 2-10 不同分区的 window

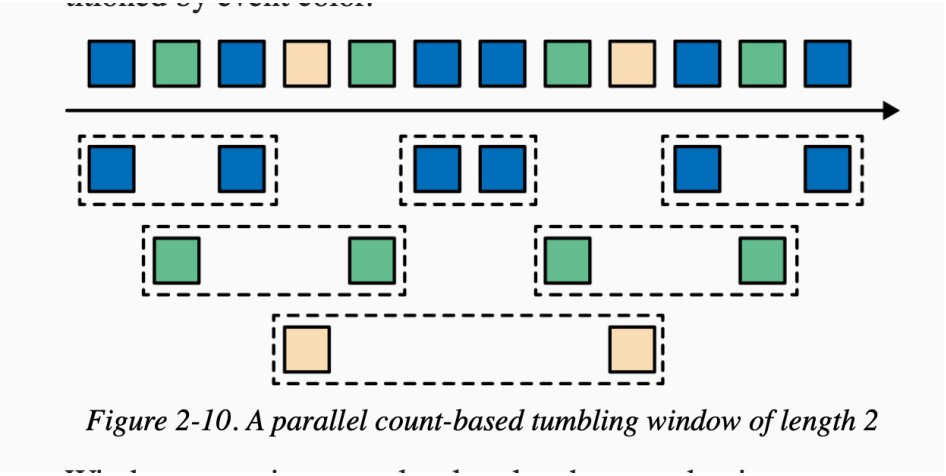

Figure 2-10. A parallel count-based tumbling window of length 2

3. 三、Time Semantics

（时间语义）

介绍流中的时间概念

3.1. What Does One Minute Mean in Stream Processing

在流处理中 1 分钟的定义是什么

前言：以手游经过地铁没信号为例，抛出问题

1. Processing Time

程序侧时间，不考虑客户端带上来的时间

2.Event time

附加在数据流中的时间戳，跟随事件上报

3.Watermarks

*它是一个全局的处理指标，表示确信不会再有事件到来的时间点

*它提供了一个可信度和延迟之间的权衡，实现用户配置

*水印并不是万能的而且也不好权衡，所以当希望丢掉水印外的事件时，可以存储事件之后用作纠正结果

4.Processing Time Versus Event Time

*处理时间适合实时汇报数据的应用或者反映真实情况的监控应用

*事件时间提供了结果保障

4. 四、State and Consistency Models

（状态和一致性模型）

分布式流处理中对于状态的三个挑战

***状态管理——不受并发更新的影响**

***状态分区——可以将事件分区独立维护每一类事件的状态**

***状态恢复——在失败情况下可以恢复正确的状态数据**

4.1. Task Failures

前言：在构建的 logic data flow 转化为 physical data flow 时每一个任务都有可能失败

1.What is a task failure?

task 处理数据的流程

本地缓存数据——>可能会更新内部状态——>产生一个输出记录，每个步骤都可能失败，下面介绍现代流处理系统的保证类型

4.2. Result Guarantees

前言：结果保证含义是流处理系统中的内部状态的一致性，并不是输出的保证，因为结果一旦输出很难再去改变除非 sink 支持事务

1.At-most-once（最多一次）

不关心丢失的数据，也称为不保证，适合关心延迟而不关心结果的应用程序

2.At-least-once

a.含义：至少一次，适用于检测某个事件的场景，但存在多次消费提供错误结果的可能

b.保证的手段:(1)持久存储实现事件重放 (2)缓存在缓冲区直到所有 task 确认处理过后即可以抛弃该条事件

3.Exactly-once（flink 系统内）

a.目的：包含至少一次的场景，所以流量重放依然是被需要的，除此 之外，还需要知道事件是否已经反应的状态上。

b.手段：

*事务更新是一种方法但是开销很大

*flink 使用的是轻量级快照的技术，会在后面讨论这个算法

4.End-to-end exactly-once

在这种情况下可以通过 At-least-once 幂等操作来实现 End-to-end exactly-once ， 幂等操作就是多次操作和一次操作返回的结果相同

5. **五、Summary**

（总结）

***前面介绍了独立于 flink 的流处理概念**

***后面将介绍 flink 如何使用这些概念以及 data streaming API 的使用**