大数据处理引擎 Apache Flink 全梳理

2017-08-24 中兴大数据

文 | 郭进良

基本原理

Apache Flink是一个面向分布式数据流处理和批量数据处理的开源计算平台，它能够基于同一个Flink运行时，提供支持流处理和批处理两种类型应用的功能。

现有的开源计算方案，会把流处理和批处理作为两种不同的应用类型，因为它们所提供的SLA（Service-Level-Aggreement）是完全不相同的：流处理一般需要支持低延迟、Exactly-once保证，而批处理需要支持高吞吐、高效处理。

Flink从另一个视角看待流处理和批处理，将二者统一起来：Flink是完全支持流处理，也就是说作为流处理看待时输入数据流是无界的；批处理被作为一种特殊的流处理，只是它的输入数据流被定义为有界的。

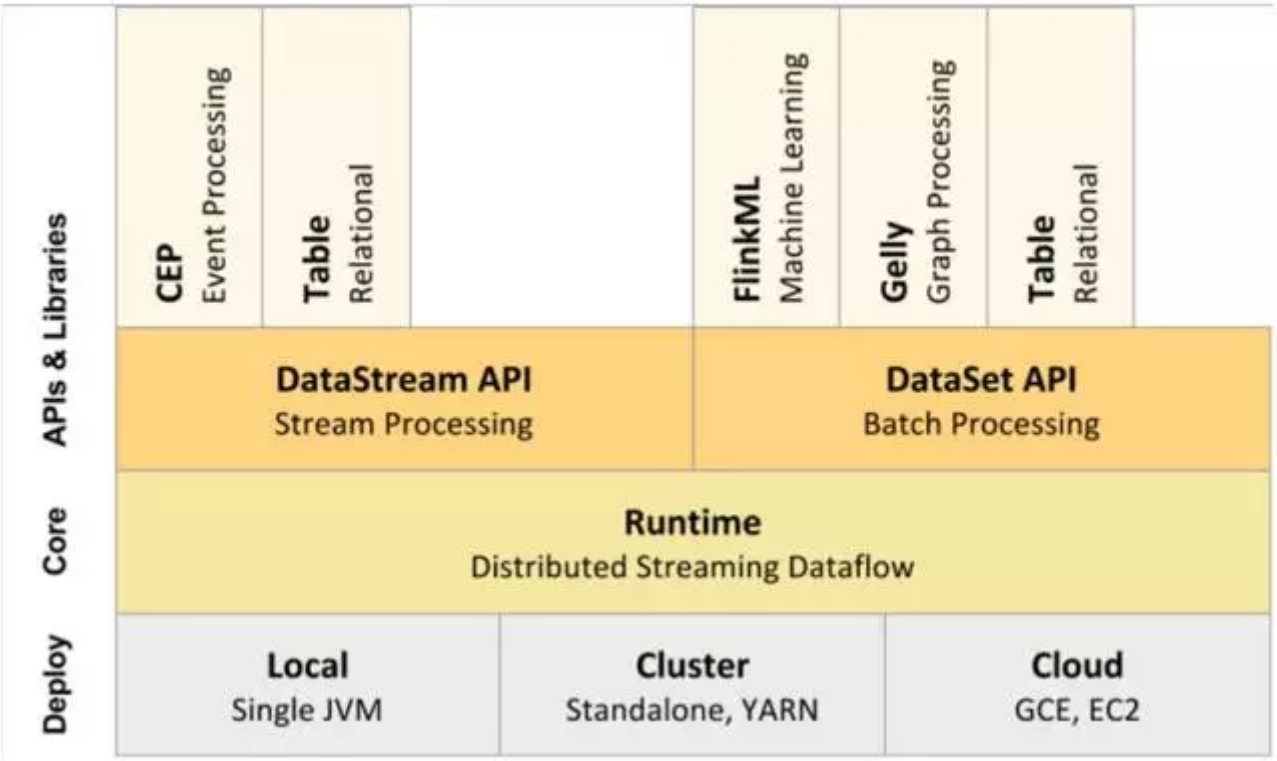


图1 Flink技术栈

Flink流处理特性：

- 支持高吞吐、低延迟、高性能的流处理
- 支持带有事件时间的窗口（Window）操作
- 支持有状态计算的Exactly-once语义
- 支持高度灵活的窗口（Window）操作，支持基于time、count、session，以及data-driven的窗口操作
- 支持具有Backpressure功能的持续流模型
- 支持基于轻量级分布式快照（Snapshot）实现的容错
- 一个运行时同时支持Batch on Streaming处理和Streaming处理
- Flink在JVM内部实现了自己的内存管理
- 支持迭代计算
- 支持程序自动优化：避免特定情况下Shuffle、排序等昂贵操作，中间结果有必要进行缓存

架构

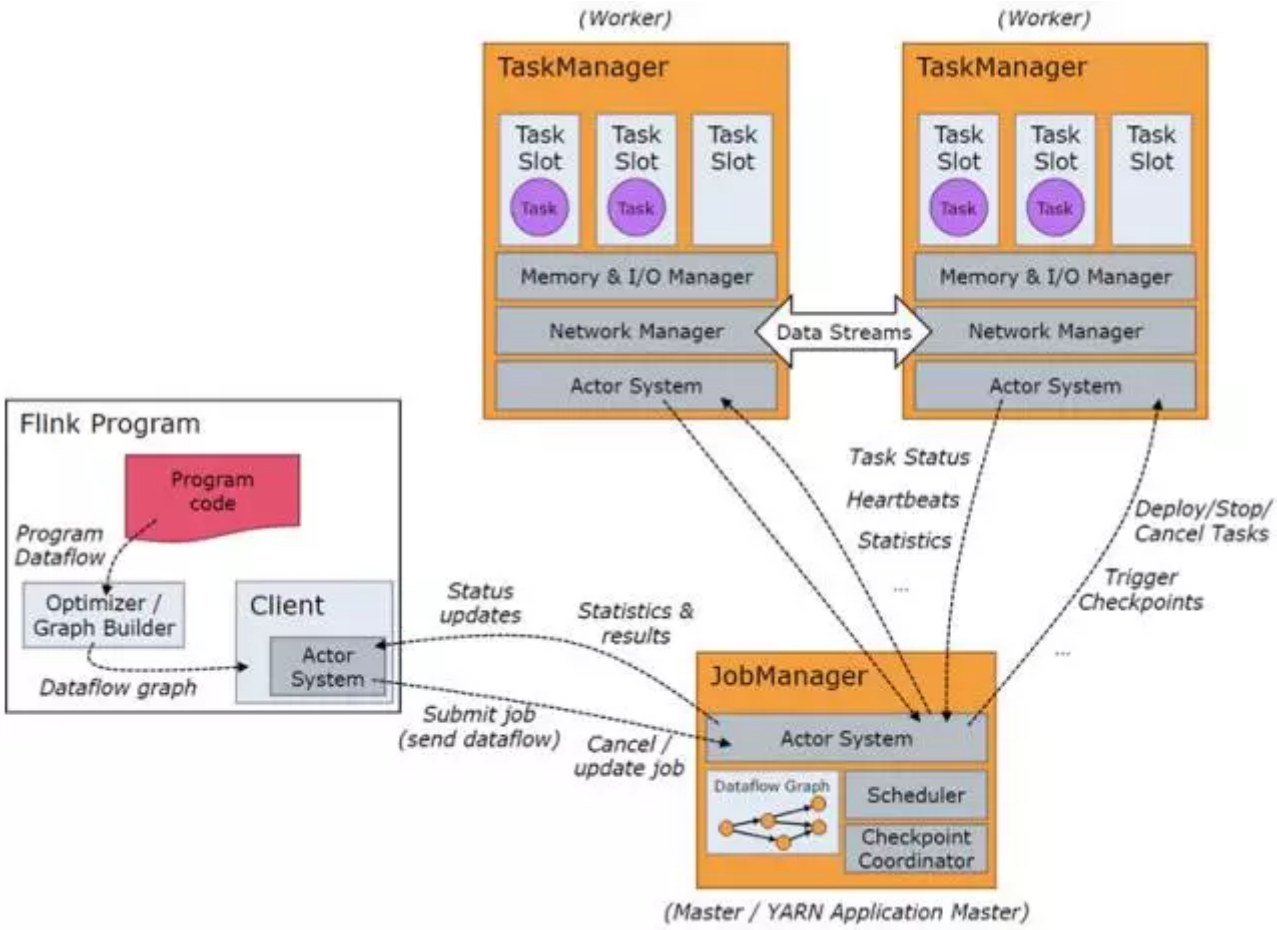


图2 Flink架构

Flink整个系统包含三个部分：

- **Client**：Flink Client主要给用户提供向Flink系统提交用户任务（流式作业）的能力。
- **TaskManager**：Flink系统的业务执行节点，执行具体的用户任务。TaskManager可以有多个，各个TaskManager都平等。
- **JobManager**：Flink系统的管理节点，管理所有的TaskManager，并决策用户任务在哪些Taskmanager执行。JobManager在HA模式下可以有多个，但只有一个主JobManager。

Flink系统提供的关键能力：

- **低时延**：提供ms级时延的处理能力。
- **ExactlyOnce**：提供异步快照机制，保证所有数据真正只处理一次。
- **HA**：JobManager支持主备模式，保证无单点故障。
- **水平扩展能力**：TaskManager支持手动水平扩展。

原理

- **流、转换、操作符**

用户实现的Flink程序是由Stream和Transformation这两个基本构建块组成。

a）Stream是一个中间结果数据，而Transformation是一个操作，它对一个或多个

输入Stream进行计算处理，输出一个或多个结果Stream。如图3所示。

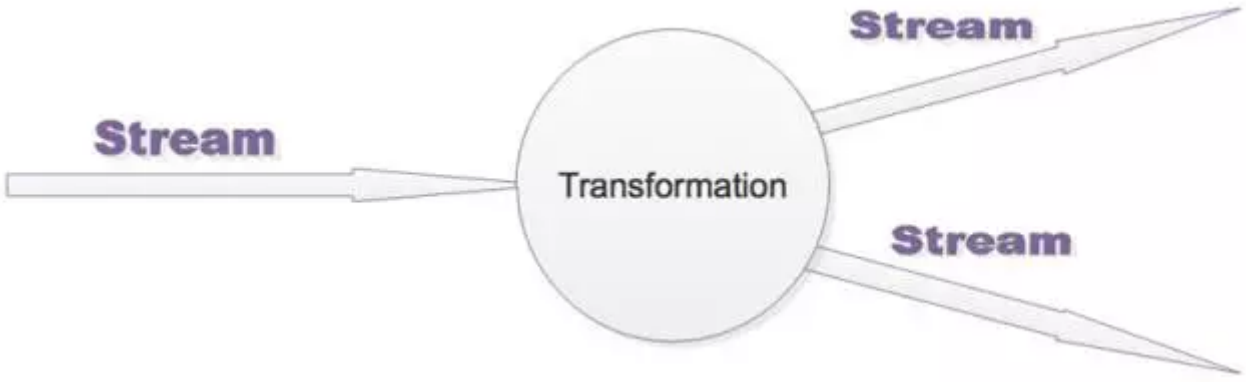


图3

b）当一个Flink程序被执行的时候，它会被映射为Streaming Dataflow。一个

Streaming Dataflow是由一组Stream和Transformation Operator组成，它类似于

一个DAG图，在启动的时候从一个或多个SourceOperator开始，结束于一个

或多个Sink Operator。如图4所示。

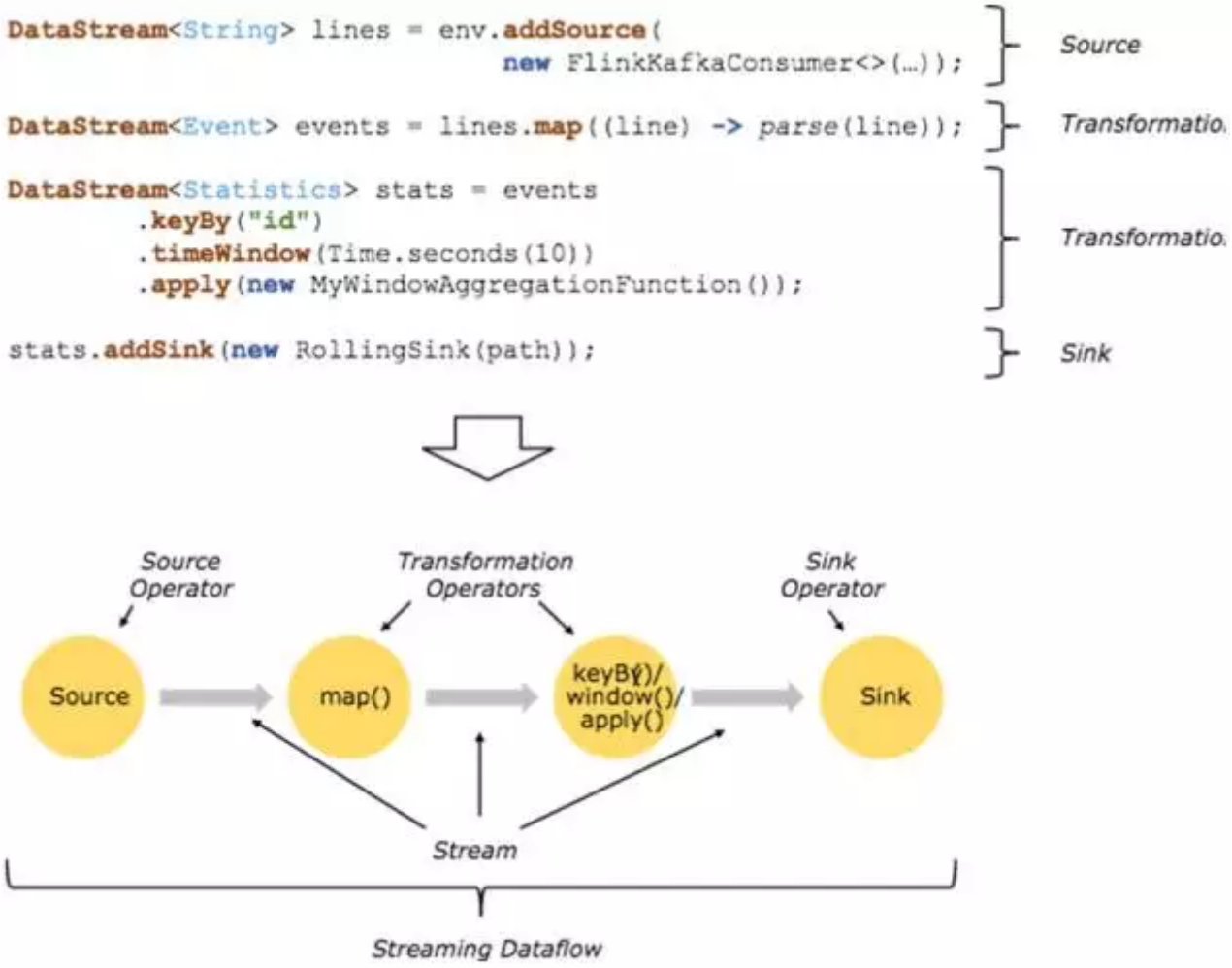


图4

- **并行数据流**

一个Stream可以被分成多个Stream分区（Stream Partitions），一个Operator可以被分成多个Operator Subtask，每一个Operator Subtask是在不同的线程中独立执行的。一个Operator的并行度，等于Operator Subtask的个数，一个Stream的并行度总是等于生成它的Operator的并行度。

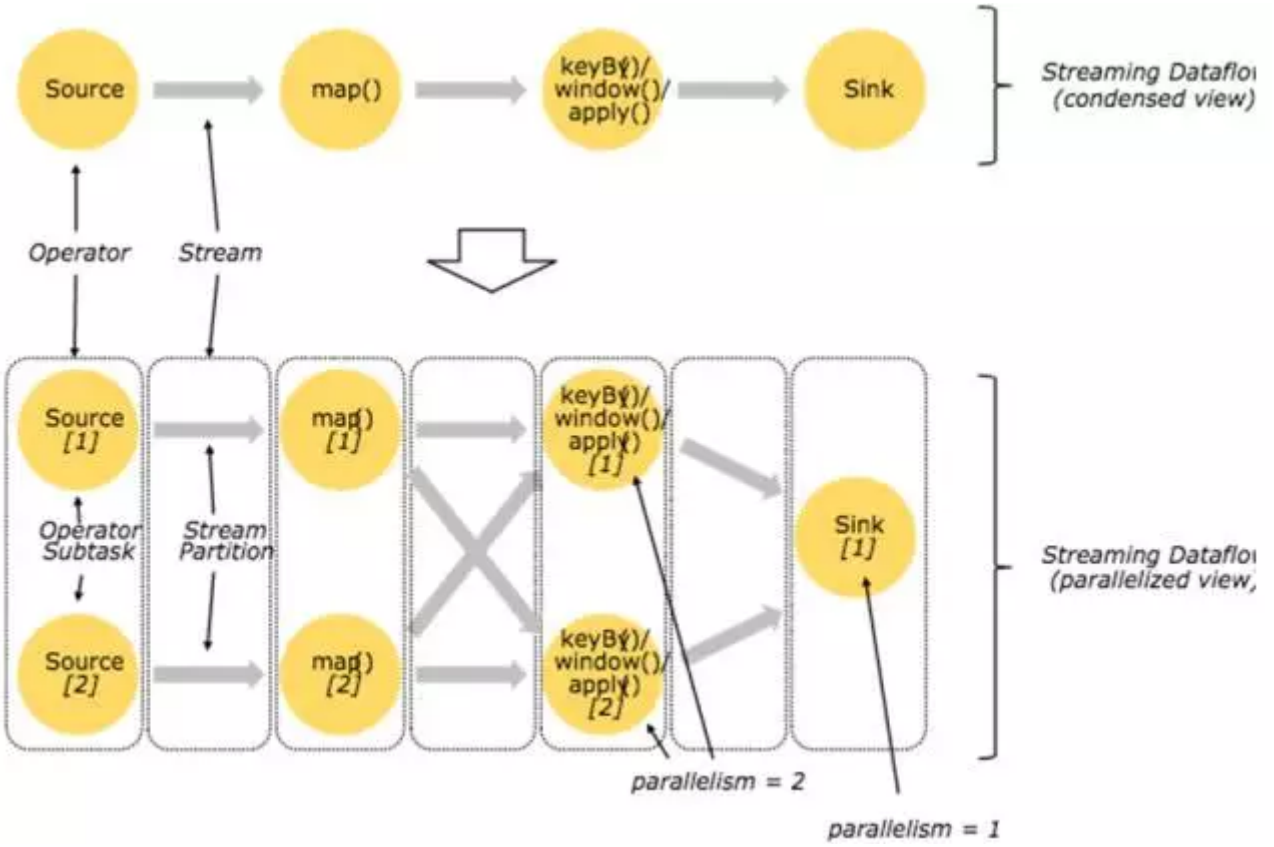


图5 Operator

紧密度低的算子则不能进行优化，而是将每一个Operator Subtask放在不同的线程中独立执行。一个Operator的并行度，等于Operator Subtask的个数，一个Stream的并行度(分区总数)等于生成它的Operator的并行度。

One-to-one模式

比如从Source[1]到map()[1]，它保持了Source的分区特性（Partitioning）和分区内元素处理的有序性，也就是说map()[1]的Subtask看到数据流中记录的顺序，与Source[1]中看到的记录顺序是一致的。

Redistribution模式

这种模式改变了输入数据流的分区，比如从map()[1]、map()[2]到keyBy()/window()/apply()[1]、keyBy()/window()/apply()[2]，上游的Subtask向下游的多个不同的Subtask发送数据，改变了数据流的分区，这与实际应用所选择的Operator有关系。

- **任务、操作符链**

紧密度高的算子可以进行优化，优化后可以将多个Operator Subtask串起来组成一个Operator Chain，实际上就是一个执行链，每个执行链会在TaskManager上一个独立的线程中执行。

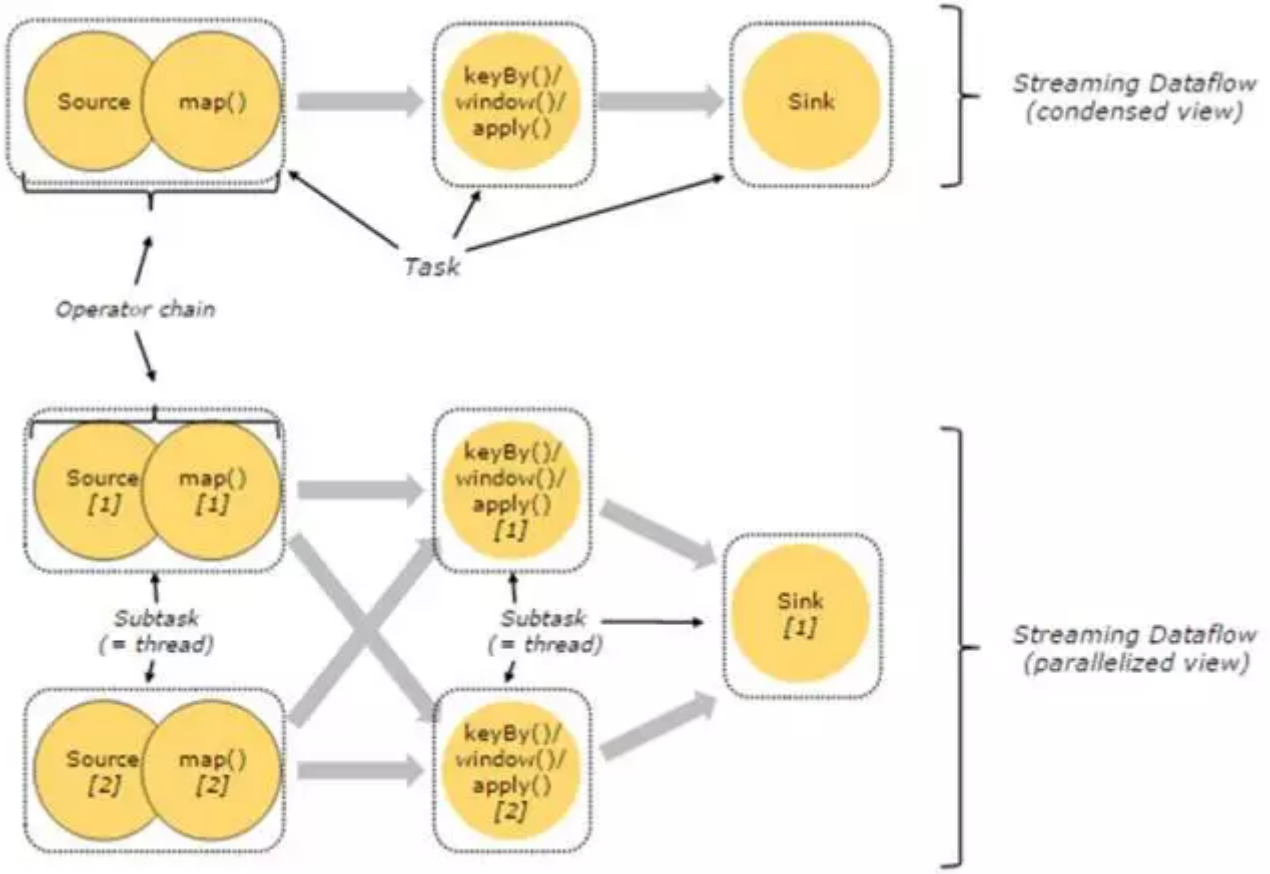


图6 Operator Chain

图6上半部分表示的是将Source和map两个紧密度高的算子优化后串成一个Operator Chain，实际上一个Operator Chain就是一个大的Operator的概念。图中的Operator Chain表示一个Operator，keyBy表示一个Operator，Sink表示一个Operator，它们通过Stream连接，而每个OperatorFusionInsight Flink在运行时对应一个Task，也就是说图中的上半部分有3个Operator对应的是3个Task。下半部分是上半部分的一个并行版本，对每一个Task都并行化为多个Subtask，这里只是演示了2个并行度，sink算子是1个并行度。

时间

处理Stream中的记录时，记录中通常会包含各种典型的时间字段：

- **Event Time**：表示事件创建时间
- **Ingestion Time**：表示事件进入到Flink Dataflow的时间
- **Processing Time**：表示某个Operator对事件进行处理的本地系统时间

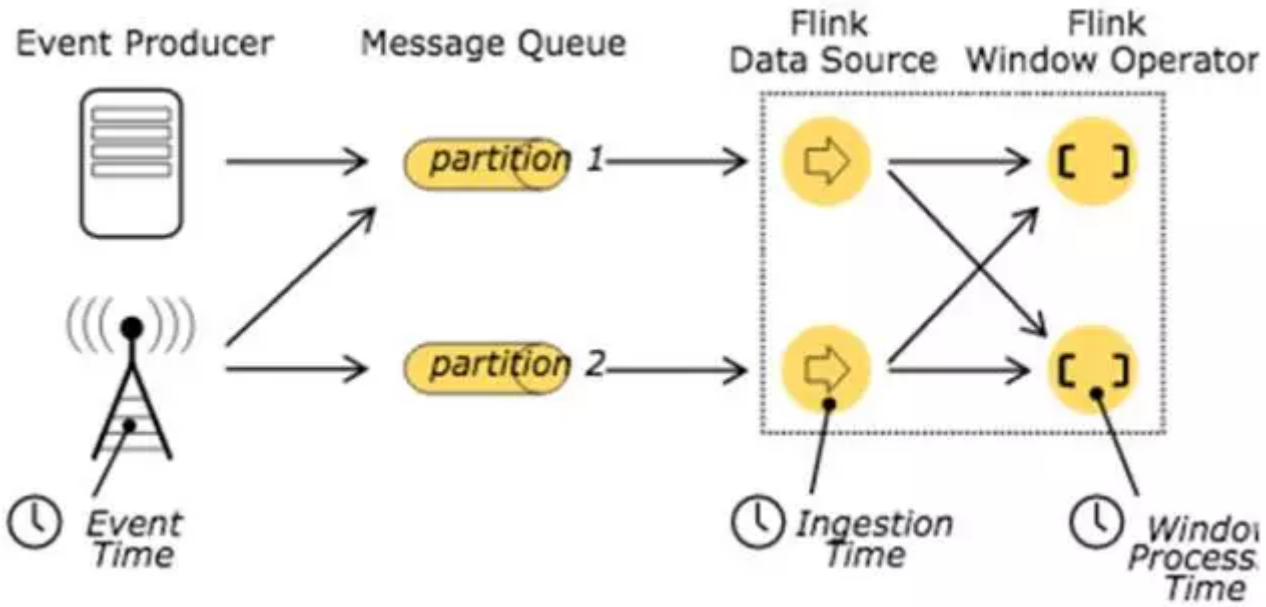


图7 time

Flink使用WaterMark衡量时间的时间，WaterMark携带时间戳t，并被插入到stream中：

- WaterMark的含义是所有时间t' < t的事件都已经发生。
- 针对乱序的的流，WaterMark至关重要，这样可以允许一些事件到达延迟，而不至于过于影响window窗口的计算。
- 并行数据流中，当Operator有多个输入流时，Operator的event time以最小流event time为准。

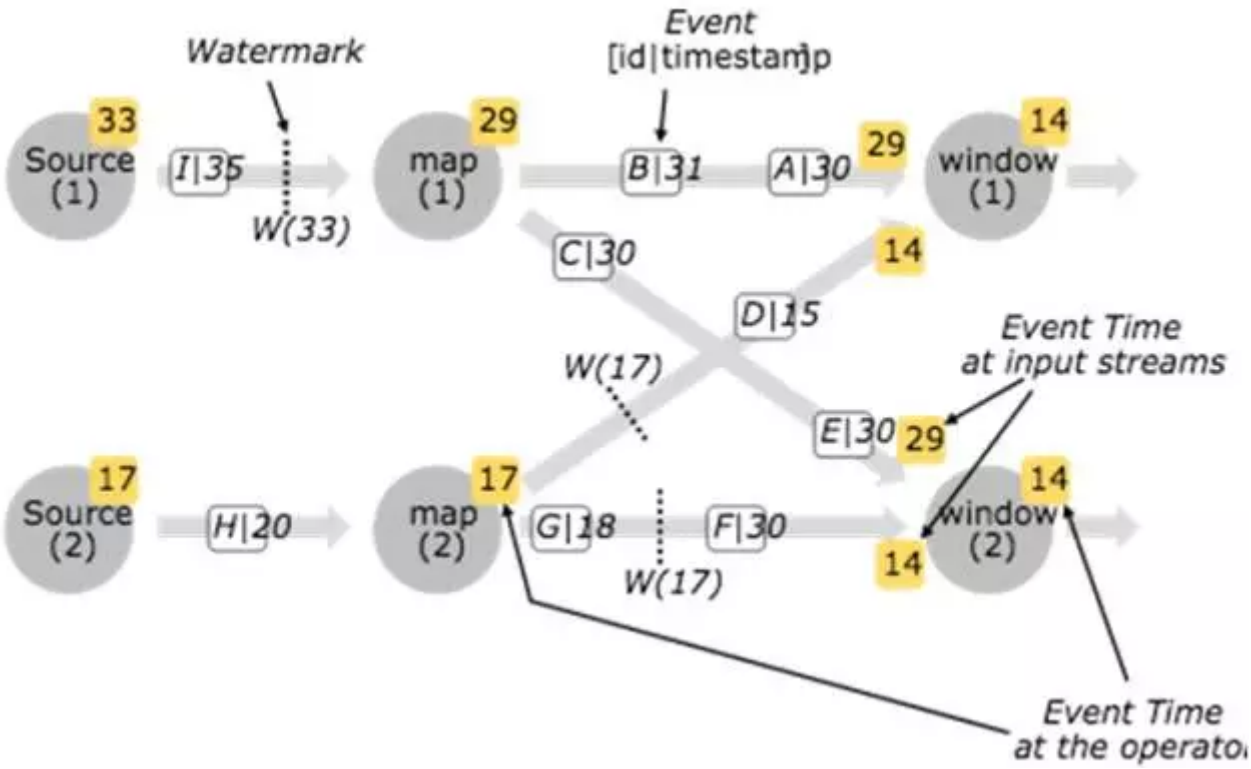


图8 WaterMark

窗口

在流处理应用中，数据是连续不断的，因此我们不可能等到所有数据都到了才开始处理。当然我们可以每来一个消息就处理一次，但是有时我们需要做一些聚合类的处理，例如：在过去的1分钟内有多少用户点击了我们的网页。在这种情况下，我们必须定义一个窗口，用来收集最近一分钟内的数据，并对这个窗口内的数据进行计算。

窗口可以是时间驱动的（Time Window，例如：每30秒钟），也可以是数据驱动的（Count Window，例如：每一百个元素）。

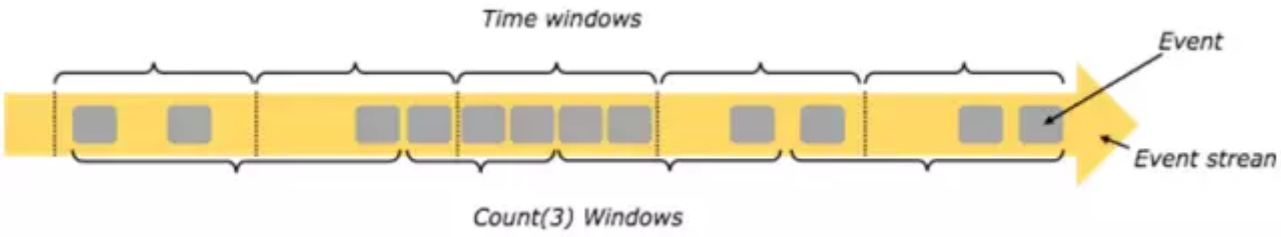


图9 窗口

一种经典的窗口分类可以分成：翻滚窗口（Tumbling Window，无重叠），滚动窗口（Sliding Window，有重叠），和会话窗口（Session Window，活动间隙）。

我们举个具体的场景来形象地理解不同窗口的概念。假设，淘宝网会记录每个用户每次购买的商品个数，我们要做的是统计不同窗口中用户购买商品的总数。下图给出了几种经典的窗口切分概述图：

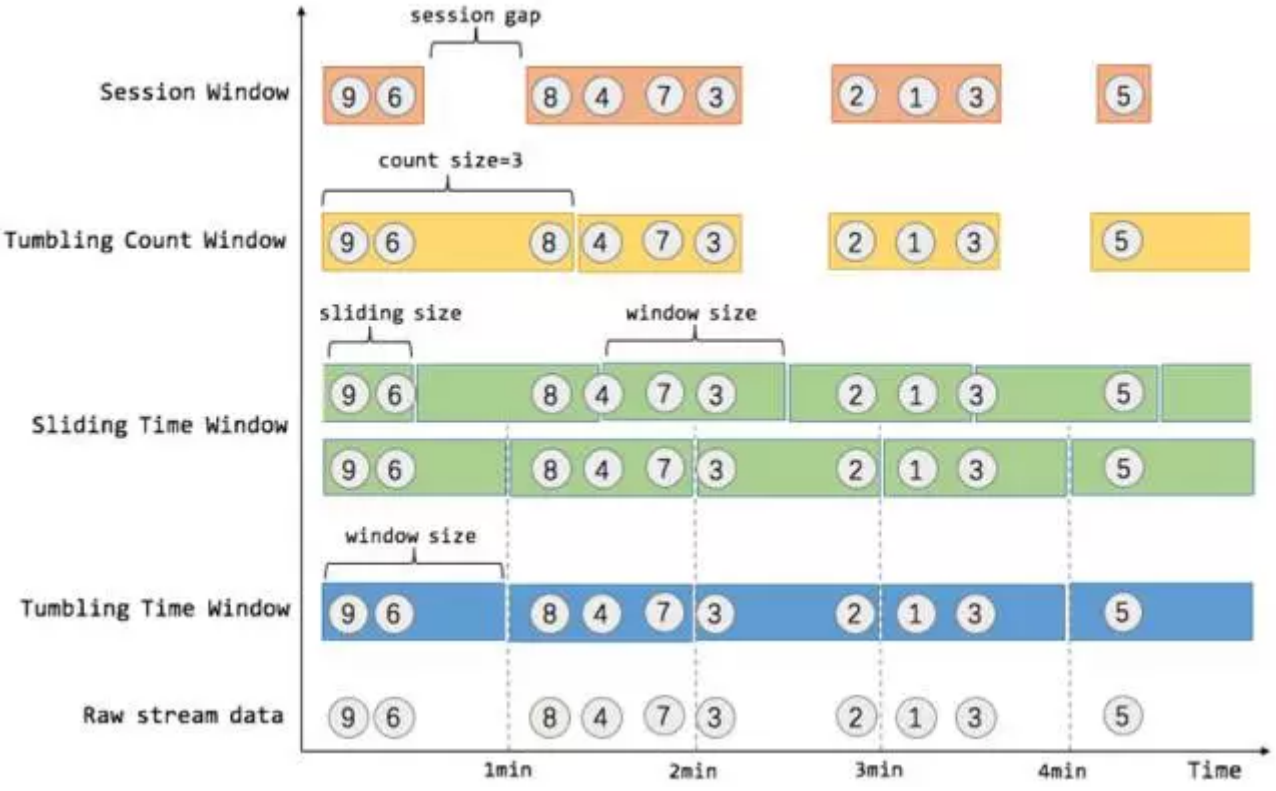


图10 窗口切分概述

上图中，raw data stream 代表用户的购买行为流，圈中的数字代表该用户本次购买的商品个数，事件是按时间分布的，所以可以看出事件之间是有time gap的。

容错

Flink分布式快照机制的核心是barriers，这些barriers周期性插入到数据流中，并作为数据流的一部分随之流动。在同一条流中barriers并不会超越其前面的数据，严格的按照线性流动。一个barrier将属于本周期快照的数据与下一个周期快照的数据分隔开来。每个barrier均携带所属快照周期的ID，barrier并不会阻断数据流，因此十分轻量。

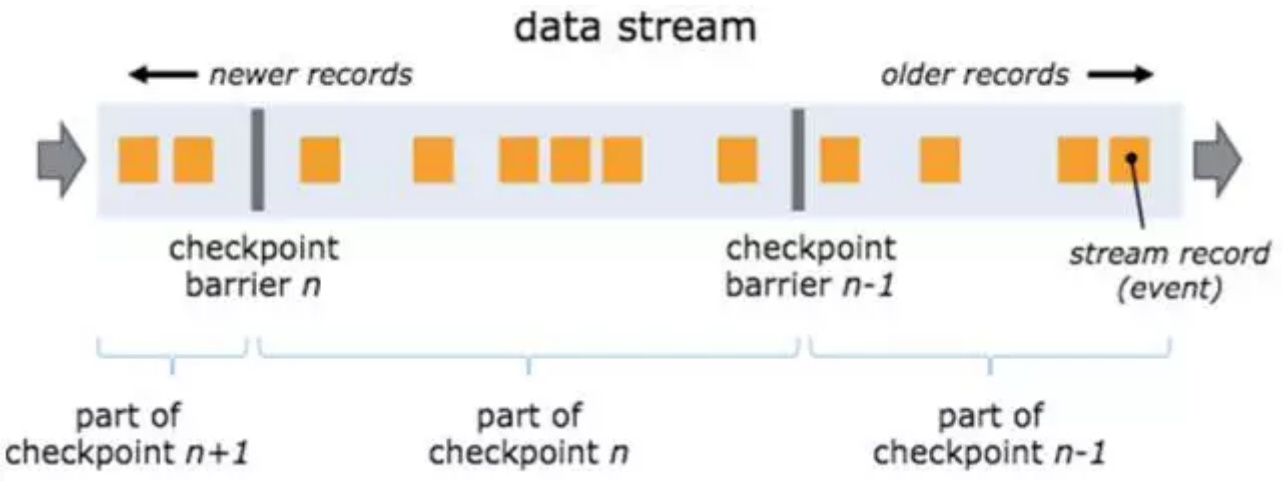
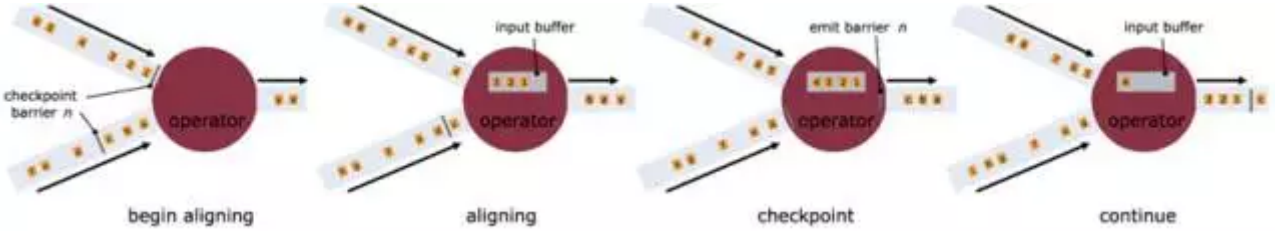


图11 Checkpoint

1. 出现一个Barrier，在该Barrier之前出现的记录都属于该Barrier对应的Snapshot，在该Barrier之后出现的记录属于下一个Snapshot。
2. 来自不同Snapshot多个Barrier可能同时出现在数据流中，也就是说同一个时刻可能并发生成多个Snapshot。
3. 当一个中间（Intermediate）Operator接收到一个Barrier后，它会发送Barrier到属于该Barrier的Snapshot的数据流中，等到Sink Operator接收到该Barrier后会向Checkpoint Coordinator确认该Snapshot，直到所有的Sink Operator都确认了该Snapshot，才被认为完成了该Snapshot。

如果一个算子有两个输入源，则暂时阻塞先收到barrier的输入源，等到第二个输入源相同编号的barrier到来时，再制作自身快照并向下游广播该barrier。



1. Operator从一个incoming Stream接收到Snapshot Barrier n，然后暂停处理，直到其它的incoming Stream的Barrier n（否则属于2个Snapshot的记录就混在一起了）到达该Operator。
2. 接收到Barrier n的Stream被临时搁置，来自这些Stream的记录不会被处理，而是被放在一个Buffer中。
3. 一旦最后一个Stream接收到Barrier n，Operator会emit所有暂存在Buffer中的记录，然后向Checkpoint Coordinator发送Snapshot n。
4. 继续处理来自多个Stream的记录。

checkpoint机制是Flink可靠性的基石，可以保证Flink集群在某个算子因为某些原因（如异常退出）出现故障时，能够将整个应用流图的状态恢复到故障之前的某一状态，保证应用流图状态的一致性。Flink的checkpoint机制原理来自“Chandy-Lamportalgorithm”算法。

每个需要checkpoint的应用在启动时，Flink的JobManager为其创建一个CheckpointCoordinator，CheckpointCoordinator全权负责本应用的快照制作。用户通过CheckpointConfig中的setCheckpointInterval()接口设置checkpoint的周期。