Flink中的一些核心概念

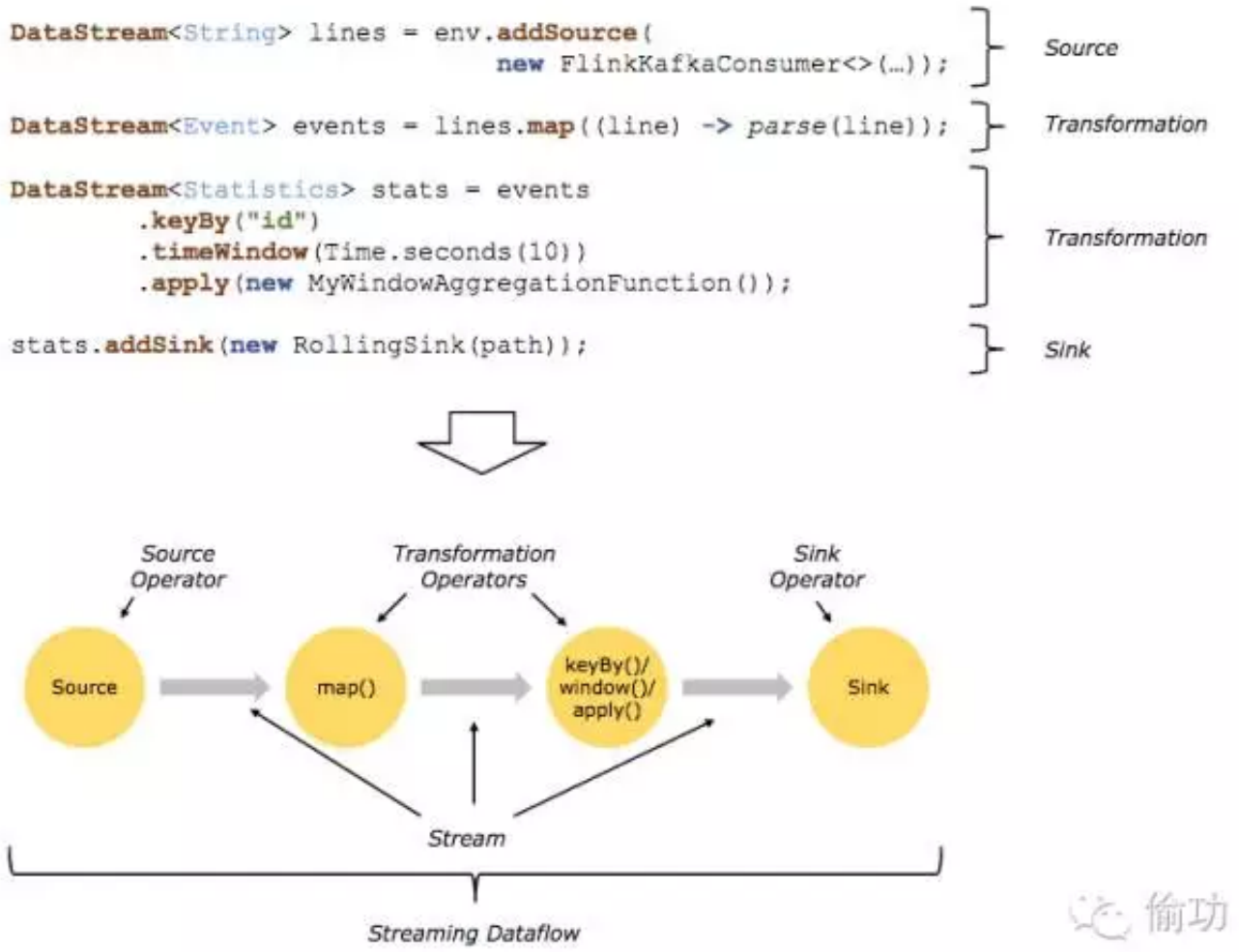
2016-06-20 vinoYang（译者） 偷功

> 在源码解读前我们有必要先了解一下Flink的一些基本的但却很关键的概念。这有助于帮助我们理解整个**架构**。在翻译文档的同时，对于有争议的或者不是非常适合用中文表达的地方，我尽量保留原始英文单词。

1 程序和数据流

Flink程序的基本构建块是streams和transformations（注意，DataSet在内部也是一个stream）。一个stream可以看成一个中间结果，而一个transformations是以一个或多个stream作为输入的某种operation，该operation利用这些stream进行计算从而产生一个或多个result stream。

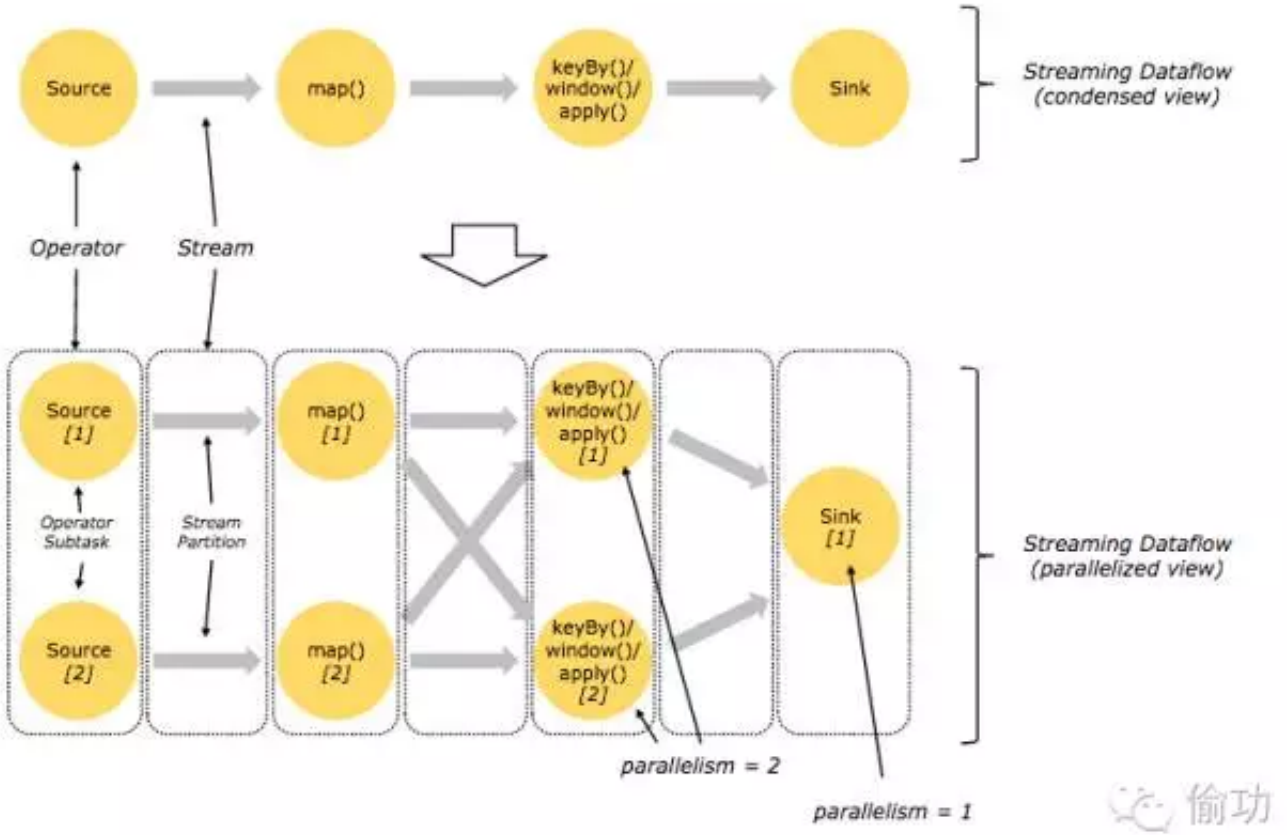
在运行时，Flink上运行的程序会被映射成streaming dataflows，它包含了streams和transformations operators。每一个dataflow以一个或多个**sources**开始以一个或多个**sinks**结束。dataflow类似于任意的有向无环图（DAG），当然特定形式的**环**可以通过**iteration**构建。在大部分情况下，程序中的transformations跟dataflow中的operator是一一对应的关系。但有时候，一个transformation可能对应多个operator。



2 并行数据流

程序在Flink内部的执行具有并行、分布式的特性。stream被分割成stream partition，operator被分割成operator subtask，这些operator subtasks在不同的线程、不同的物理机或不同的容器中彼此互不依赖得执行。

一个特定operator的subtask的个数被称之为其parallelism(并行度)。一个stream的并行度总是等同于其producing operator的并行度。一个程序中，不同的operator可能具有不同的并行度。



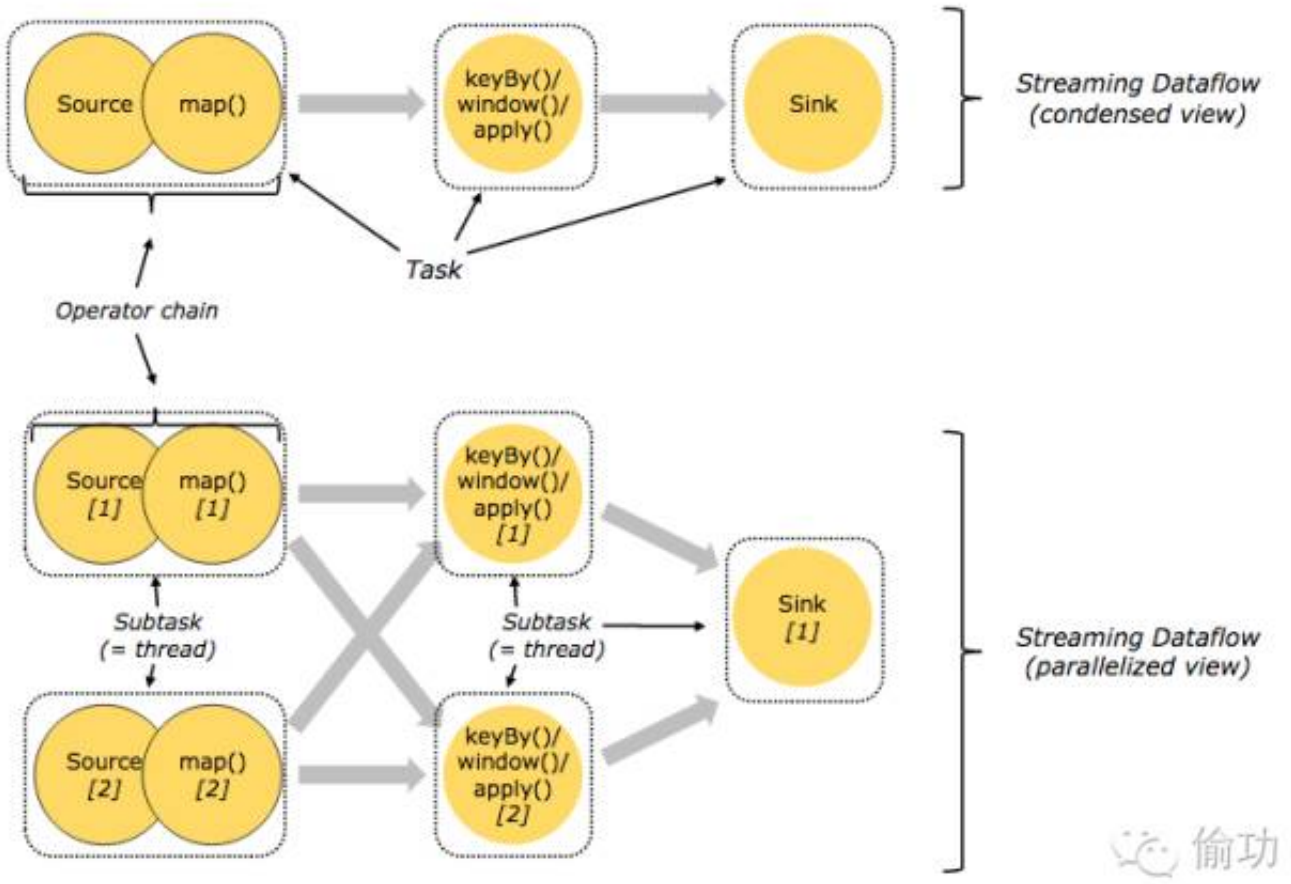
Stream在operator之间传输数据的形式可以是**one-to-one**(forwarding)的模式也可以是**redistributing**的模式。

- **One-to-one :** strem(比如在source和map operator之间)维护着分区以及元素的顺序。那意味着map operator的subtask看到的元素的个数以及顺序跟source operator的subtask生产的元素的个数、顺序相同。
- **Redistributing :** stream(map()跟keyBy/window之间或者keyBy/window跟sink之间)的分区会发生改变。每一个operator subtask依据所选择的transformation发送数据到不同的目标subtask。例如，keyBy()（基于hash码重分区），broadcast()或者rebalance()（随机redistribution）。在一个redistribution的交换中，只有每一个发送、接收task对的顺序才会被维持（比如map()的subtask和keyBy/window的subtask）。

3 task & operator chains

出于分布式执行的目的，Flink将operator的subtask链接在一起形成task。每个task在一个线程中执行。将operators链接成task是非常有效的优化：它能减少线程之间的切换和基于缓存区的数据交换，在减少时延的同时提升吞吐量。链接的行为可以在编程API中进行指定。

下面这幅图，展示了5个subtask以5个并行的线程来执行。



4 分布式执行

Master,Worker,Client

Flink运行时包含了两种类型的处理器：

- **master处理器：**也称之为**JobManagers**用于协调分布式执行。它们用来调度task，协调检查点，协调失败时恢复等。

Flink运行时至少存在一个master处理器。一个高可用的运行模式会存在多个master处理器，它们其中有一个是leader，而其他的都是standby。

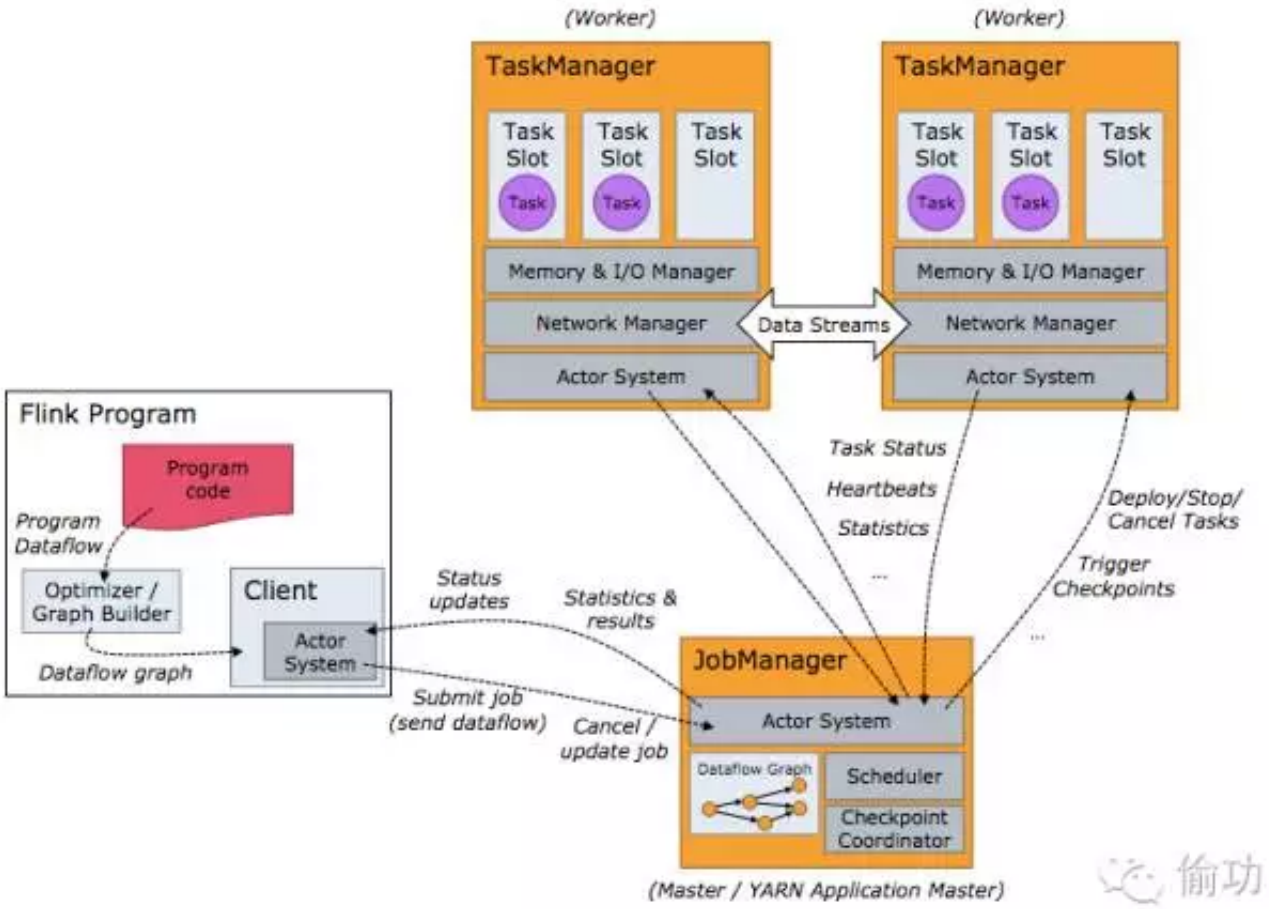
- **worker处理器：**也称之为**TaskManagers**用于执行一个dataflow的task(或者特殊的subtask)、数据缓冲和data stream的交换。

Flink运行时至少会存在一个worker处理器。

master和worker处理器可以以如下方式中的任意一种启动：直接在物理机上启动，通过容器，或者通过像YARN这样的资源调度框架。worker连接到master，告知自身的可用性进

而获得任务分配。

客户端不是运行时和程序执行的一部分。但它用于准备并发送dataflow给master。然后，客户端断开连接或者维持连接以等待接收计算结果。客户端可以以两种方式运行：要么作为**Java**/Scala程序的一部分被程序触发执行，要么以命令行./bin/flink run的方式执行。



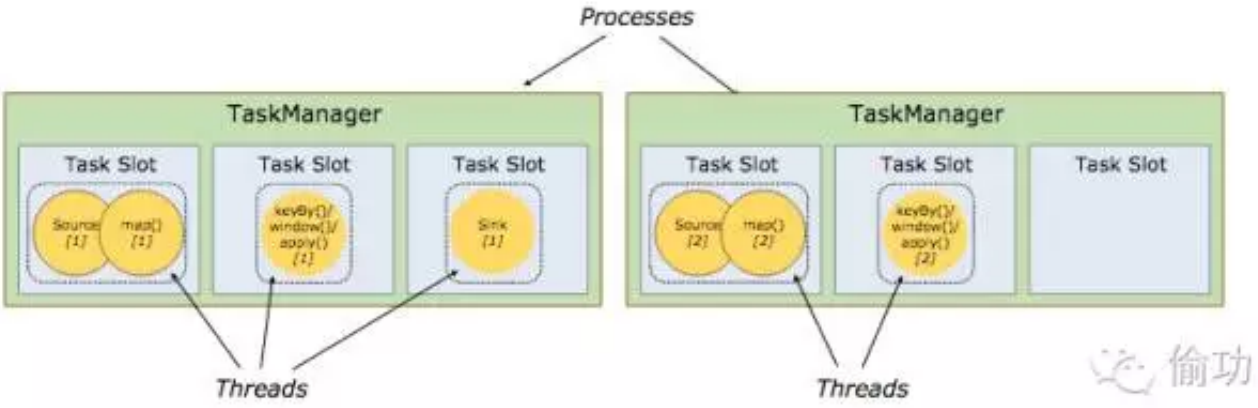
5 Workers,Slots,Resources

每一个worker(TaskManager)是一个JVM进程，它可能会在独立的线程上执行一个或多个subtask。为了控制一个worker能接收多少个task。worker通过**task slot**来进行控制（一个worker至少有一个task slot）。

每个task slot表示TaskManager拥有资源的一个固定大小的子集。假如一个TaskManager有三个slot，那么它会将其管理的内存分成三份给各个slot。资源slot化意味着一个subtask将不需要跟来自其他job的subtask竞争被管理的内存，取而代之的是它将拥有一定数量的内存储备。需要注意的是，这里不会涉及到CPU的隔离，slot目前仅仅用来隔离task的受管理的内存。

通过调整task slot的数量，允许用户定义subtask之间如何互相隔离。如果一个

TaskManager一个slot，那将意味着每个task group运行在独立的JVM中（该JVM可能是通过一个特定的容器启动的）。而一个TaskManager多个slot意味着更多的subtask可以共享同一个JVM。而在同一个JVM进程中的task将共享TCP连接（基于多路复用）和心跳消息。它们也可能共享数据集和数据结构，因此这减少了每个task的负载。

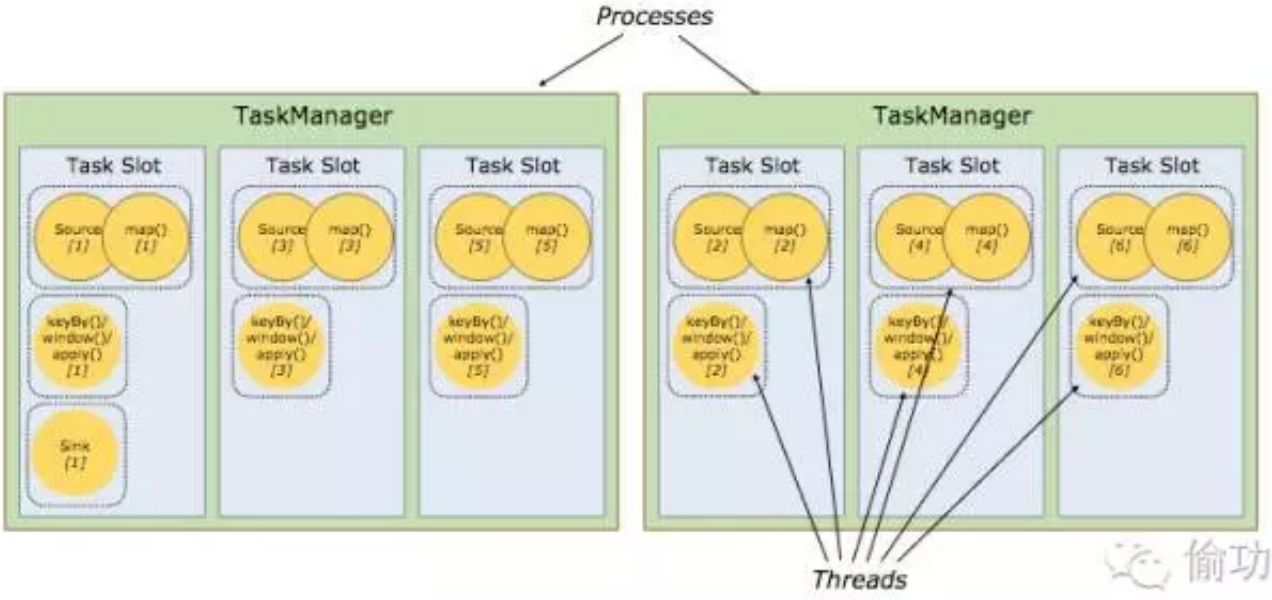


默认，如果subtask是来自相同job，但不是相同的task，Flink允许subtask共享slot。结果是，一个slot可能hold住该job的整个pipeline。允许slot共享有两个好处：

- Flink集群确实需要许多task slots来让Job达到最高的并行度。不需要计算一个程序总共包含多少个task。
- 更容易获得更好的资源利用。如果没有slot共享，非密集型的source/map()的subtask将阻塞跟密集型的window的subtask一样多的占用资源。而如果有slot共享，基本的并发度通过完整地利用共享的slot资源将获得2到6倍的提升，同时仍然保证每一个TaskManager会在任务繁重的subtask之间进行合理的slot共享。

slot共享行为可以通过API来控制，以防止不合理的共享。这个机制称之为**resource groups**，它定义了subtask可能共享的slot是什么资源。

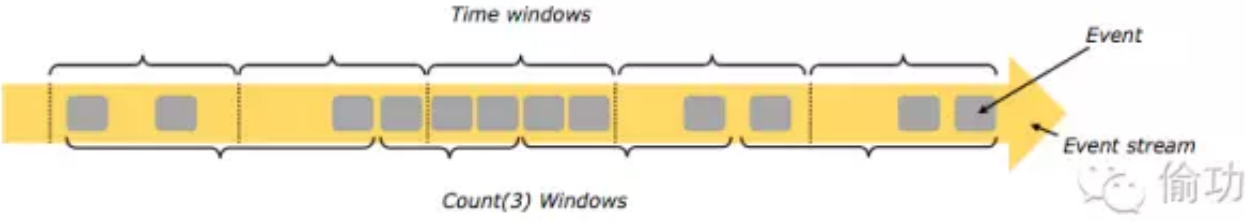
作为一个约定俗成的规则，task slot推荐的默认值是CPU的核数。基于超线程技术，每个slot占用两个或者更多的实际线程上下文。



6 时间窗口

聚合事件（比如count,sum）工作起来比起批处理略微有些不同。例如，它不能一次完成对流中所有元素的数量统计，然后返回结果。因为流通常都是无限的（无边界）。取而代之的是，在流上的聚合（count，sum等）被隔离到window域中，比如，“统计最近5分钟的数量”或“对最近100个元素求和”。

窗口可以是时间驱动的（比如，每30秒）也可以是数据驱动的（比如，每100个元素）。通常我们将窗口划分为：tumbing windows(不重叠)，sliding windows（有重叠）和session windows(有空隙的活动)。

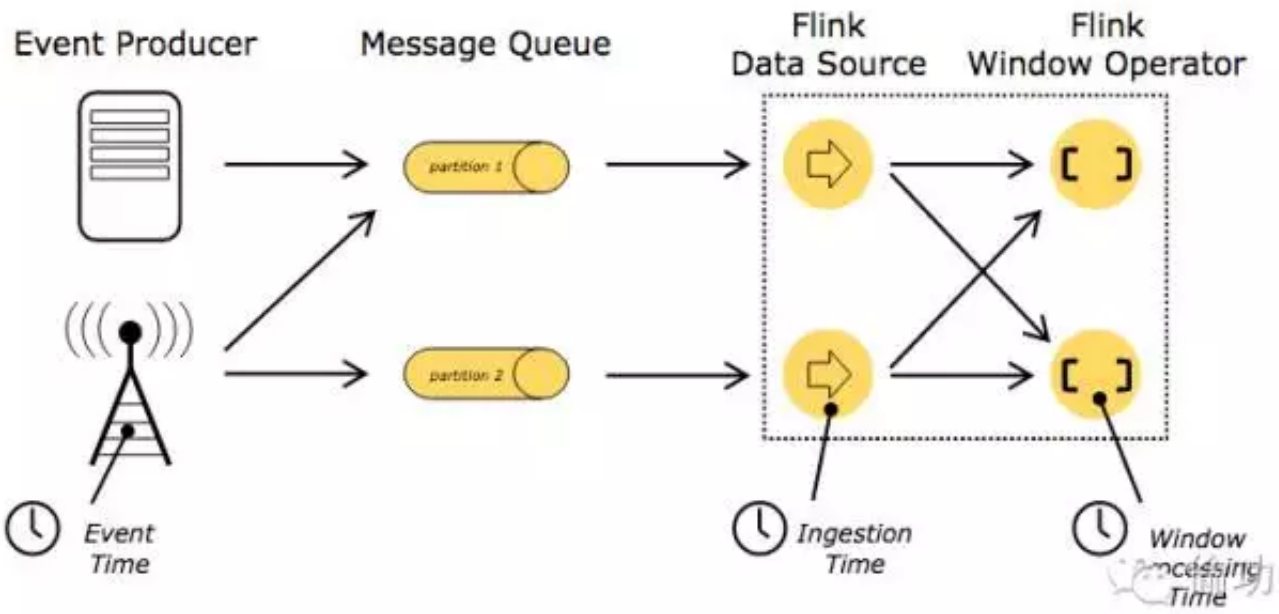


7 时间

当在流式编程中涉及到时间的（比如定义一个窗口），可能会牵扯到时间的不同定义：

- **Event Time：**指一个事件的创建时间。通常在event中用时间戳来描述，比如，可能是由生产事件的传感器或生产服务来附加。Flink访问事件时间戳通过时间戳分配器。

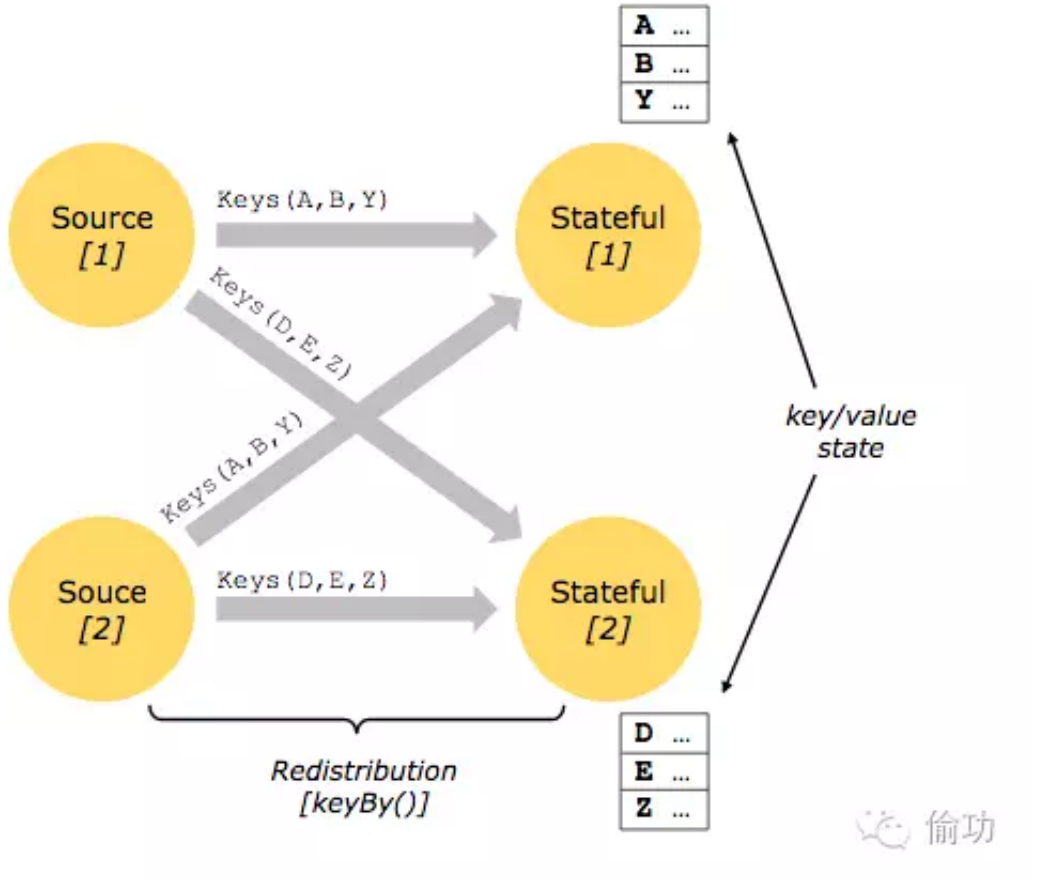
- **Ingestion time：**指一个事件从source operator进入Flink dataflow的时间。
- **Processing time：**每一个执行一个基于时间操作的operator的本地时间。



9 状态和失败容忍

在dataflow中的许多操作一次只关注一个独立的事件（比如一个事件解析器），还有一些操作能记住多个独立事件的信息（比如，window operator），而这些操作被称为**stateful**（有状态的）。

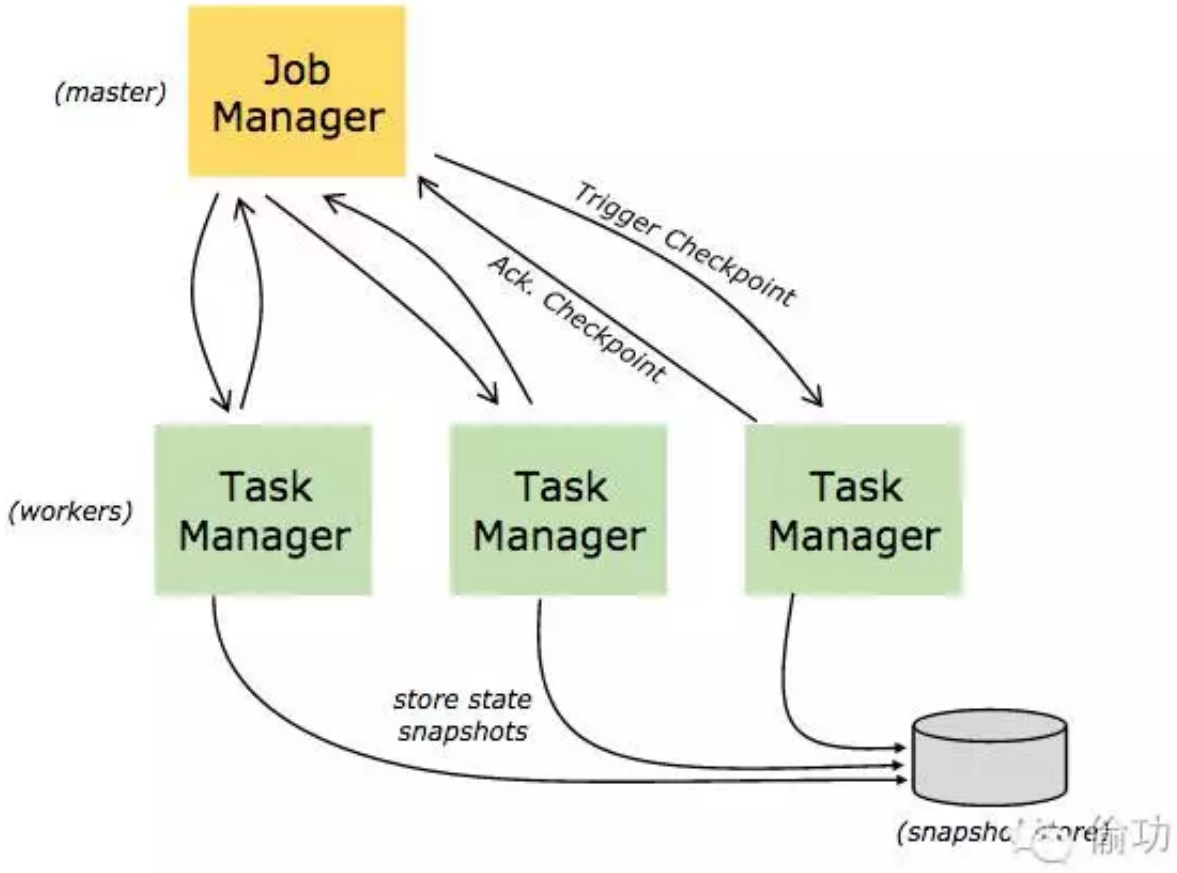
有状态的操作，其状态被维护的地方，可以将其看作是一个内嵌的key/value存储器。状态和流一起被严格得分区和分布以供有状态的operator读取。因此，访问key/value的状态仅能在**keyed streams**中（在执行keyBy()函数之后产生keyed stream），并且只能根据当前事件的键来访问其值。对齐stream的键和状态可以确保所有的状态更新都是本地操作，在不需要事务开销的情况下保证一致性。这个对齐机制也允许Flink重新分布状态并显式调整stream的分区。



用于失败容忍的检查点

Flink实现失败容忍使用了**流重放**和**检查点**的混合机制。一个检查点会在流和状态中定义一个一致点，在该一致点streaming dataflow可以恢复并维持一致性（exactly-once的处理语义）。在最新的检查点之后的事件或状态更新将在input stream中被重放。

检查点的设置间隔意味着在执行时对失败容忍产生的额外开销以及恢复时间（也决定了需要被重放的事件数）。



状态的最终存储

给key/value构建索引的数据结构最终被存储的地方取决于状态最终存储的选择。其中一个选择是在内存中基于hash map，另一个是RocksDB。另外用来定义Hold住这些状态的数据结构，状态的最终存储也实现了基于时间点的快照机制，给key/value做快照，并将快照作为检查点的一部分来存储。

10 基于流的批处理

Flink执行批处理程序是将其作为流处理程序的一个特例来看待。它将其看作有界的流（有限数量的元素）。DataSet在内部被当作一个流数据，因此上面的这些适用于流处理的这些概念在批处理中同样适用，只有很少的几个例外：

- DataSet的编程API不适用检查点。恢复机制是通过重放完整的流数据来进行。那是合理的，因为输入时有界的。它将开销更多地引入到恢复操作上，但另一方面也使得运行时的常规流程代价更低，因为它规避了检查点机制。
- DataSet的有状态的operation API简单地使用**in-memory/out-of-core**的数据结构，而不是基于key/value的索引机制
- DataSet的API引进了独特的同步迭代机制（基于superstep），它仅在有界的流中存在。

如果想在PC端阅读作者原文，可以点击本文左下角的“阅读原文”。

阅读原文