Flink之一 Flink基本原理介绍

Flink介绍：

很多人可能都是在 2015 年才听到 Flink 这个词，其实早在 2008 年，Flink 的前身已经是柏林理工大学一个研究性项目， 在 2014 被 Apache 孵化器所接受，然后迅速地成为了 ASF（Apache Software Foundation）的顶级项目之一。Flink 的最新版本目前已经更新到了 0.10.0 了，在很多人感慨 Spark 的快速发展的同时，或许我们也该为 Flink 的发展速度点个赞。

Flink 是一个针对流数据和批数据的分布式处理引擎。它主要是由 Java 代码实现。目前主要还是依靠开源社区的贡献而发展。对 Flink 而言，其所要处理的主要场景就是流数据，批数据只是流数据的一个极限特例而已。再换句话说，Flink 会把所有任务当成流来处理，这也是其最大的特点。Flink 可以支持本地的快速迭代，以及一些环形的迭代任务。

Flink的特性：

Flink是个分布式流处理开源框架：

1： 即使数据源是无序的或者晚到达的数据，也能保持结果准确性

2：有状态并且容错，可以无缝的从失败中恢复，并可以保持exactly-once

3：大规模分布式

Flink可以确保仅一次语义状态计算；Flink有状态意味着，程序可以保持已经处理过的数据；

Flink支持流处理和窗口事件时间语义，Flink支持灵活的基于时间窗口,计数,或会话数据驱动的窗户；

Flink容错是轻量级和在同一时间允许系统维持高吞吐率和提供仅一次的一致性保证，Flink从失败中恢复，零数据丢失；

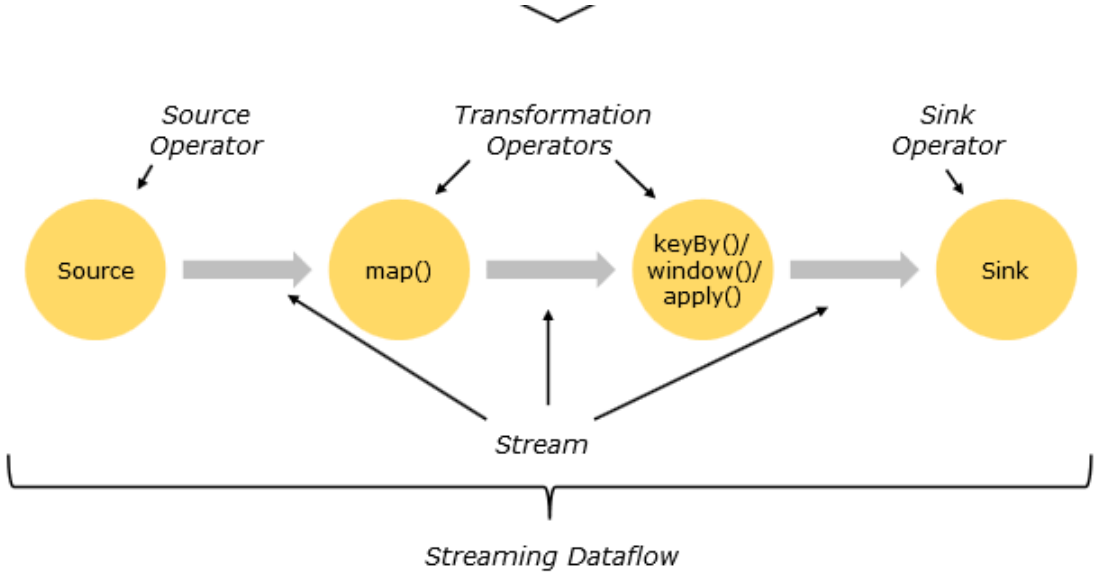
Flink能够高吞吐量和低延迟；

Flink保存点提供版本控制机制,从而能够更新应用程序或再加工历史数据没有丢失并在最小的停机时间。

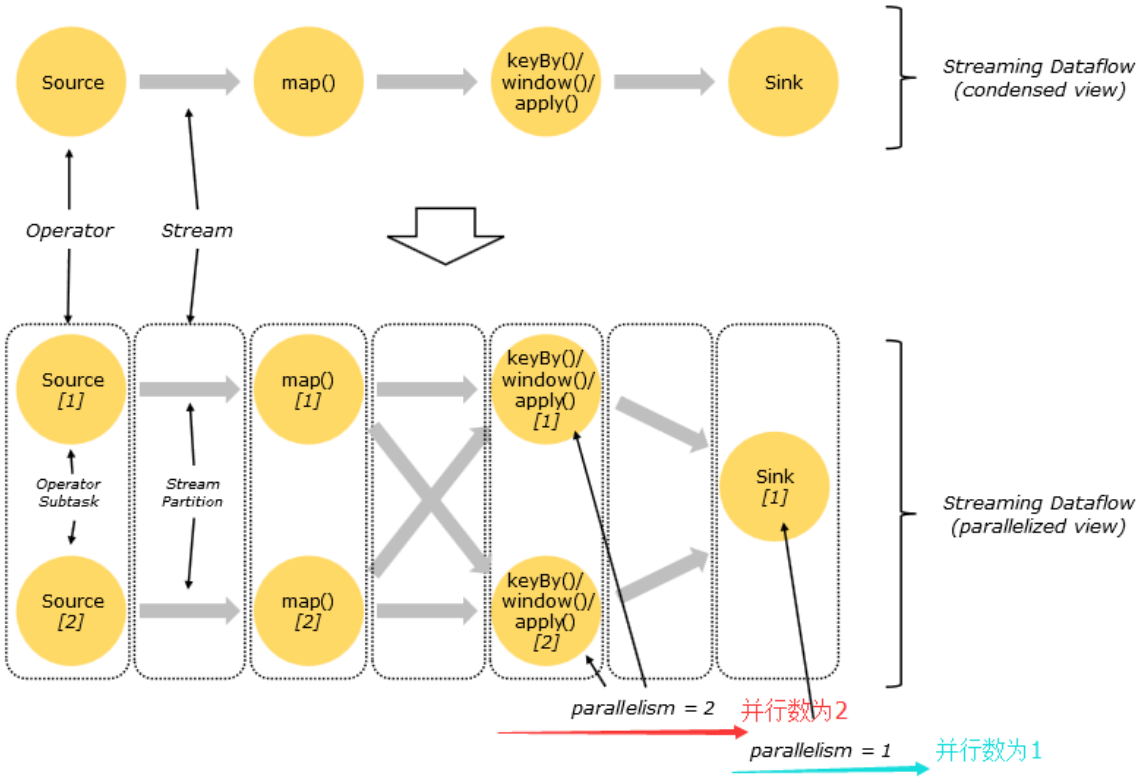
Flink相关概念：

Parallel Dataflows

Flink中把整个流处理过程叫做Stream Dataflow,从数据源提取数据的操作叫做Source Operator,中间的map(),聚合、统计等操作可以统称为Tranformation Operators,最后结果数据的流出被称为sink operators,具体可以见下方图示:



Flink的程序内在是并行和分布式的，数据流可以被分区成**stream partitions**，operators被划分为operator subtasks;这些subtasks在不同的机器或容器中分不同的线程独立运行；operator subtasks的数量在具体的operator就是并行计算数，程序不同的operator阶段可能有不同的并行数；如下图所示，source operator的并行数为2，但最后的sink operator为1；



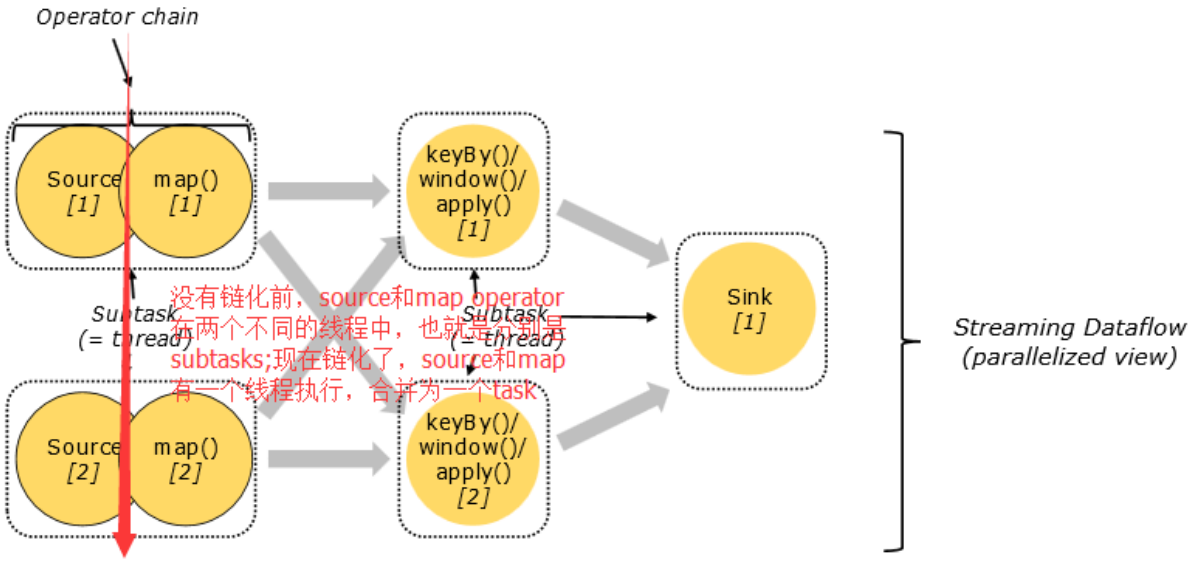
数据在两个operator之间传递的时候有两种模式：

一：one-to-one 模式：两个operator用此模式传递的时候，会保持数据的分区数和数据的排序；

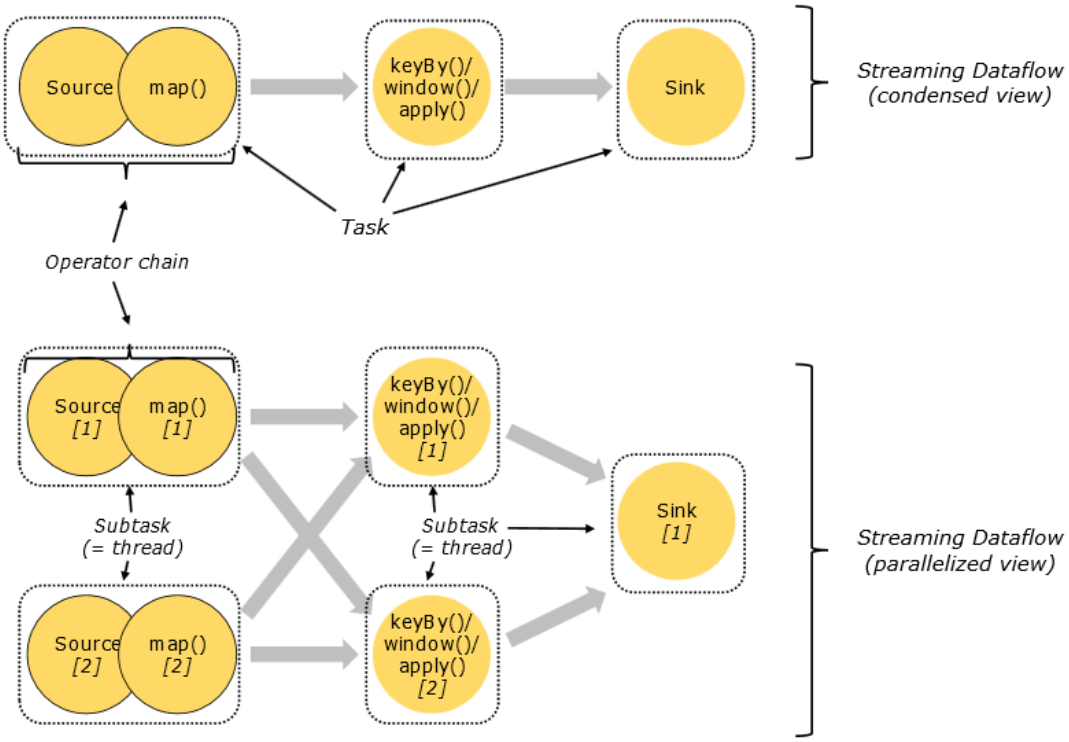
二：Redistributing 模式：这种模式会改变数据的分区数；每个一个operator subtask会根据选择transformation把数据发送到不同的目标subtasks,比如keyBy()会通过hashcode重新分区,broadcast()和rebalance()方法会随机重新分区；

Tasks & Operator Chains

对于分布式计算，Flink封装operator subtasks 链化为tasks;每个task由一个线程执行；把tasks链化有助于优化，它减少了开销线程和线程之间的交接和缓冲；增加了吞吐量和减少延迟时间；链化的作用可以见下图：在没有链化之前，source operator和map operator 是两个线程运行的两个task，也就是说下面的dataflow 最初应该有7个subtasks;



但经过优化链化后，source和map合并为一个task，有一个线程执行，这样优化可以减少source operator 和map operator两个线程之间的交接和缓存开销；链化后只有5个task；对于链化这个优化，笔者也有疑问：是否是operator之间数据传递模式相同才能链化?



Distributed Execution

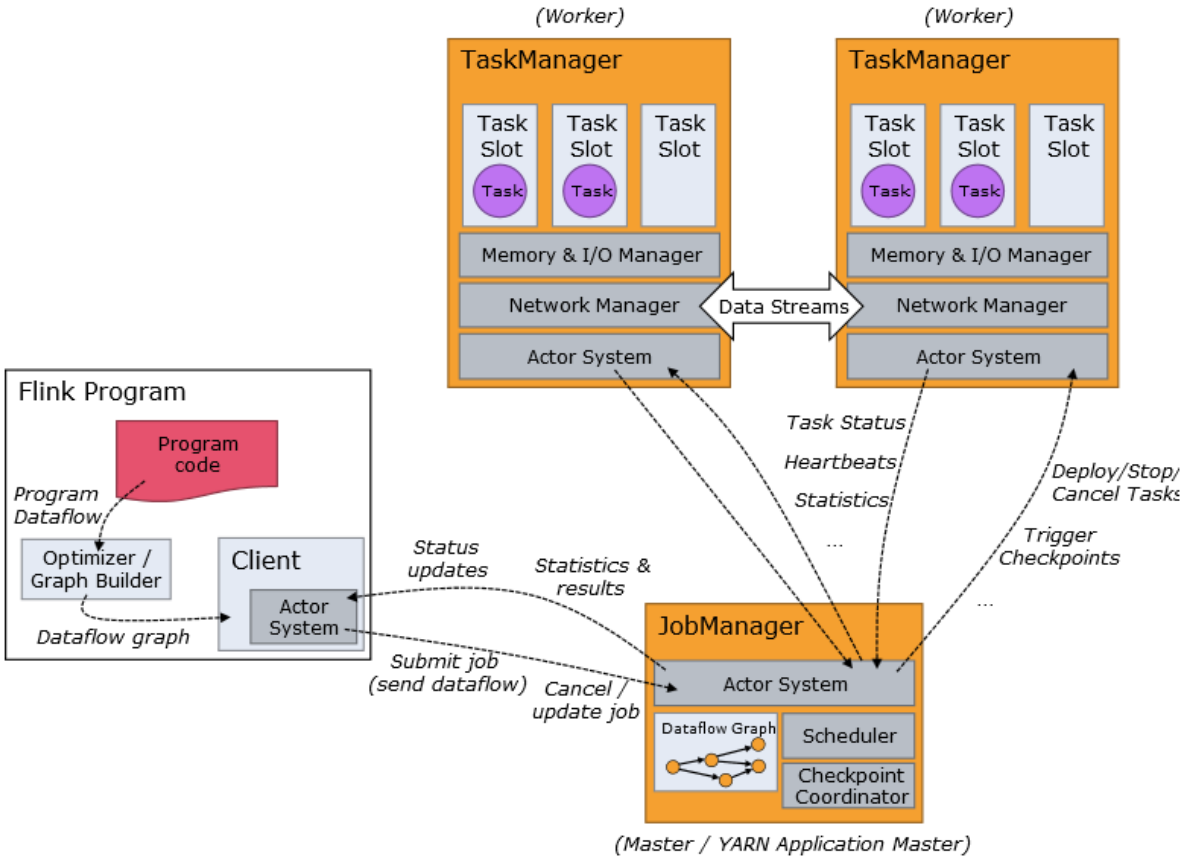
Flink runTime 包括两种类型进程（类似于第一代hadoop架构）：

master进程；也叫作JobManager,协调各个节点工作；master调度task，协调checkpoints和容灾；机器群中至少有一个master，高可用机器中可以有多个master，但要保证一个是leader,其他是standby;

work 类型进程；也叫taskManagers;具体执行tasks;

client 虽然不是运行和程序的一部分，但是客户端常被用作准备和发送dataflow给master；

flink作业提交架构流程可见下图：



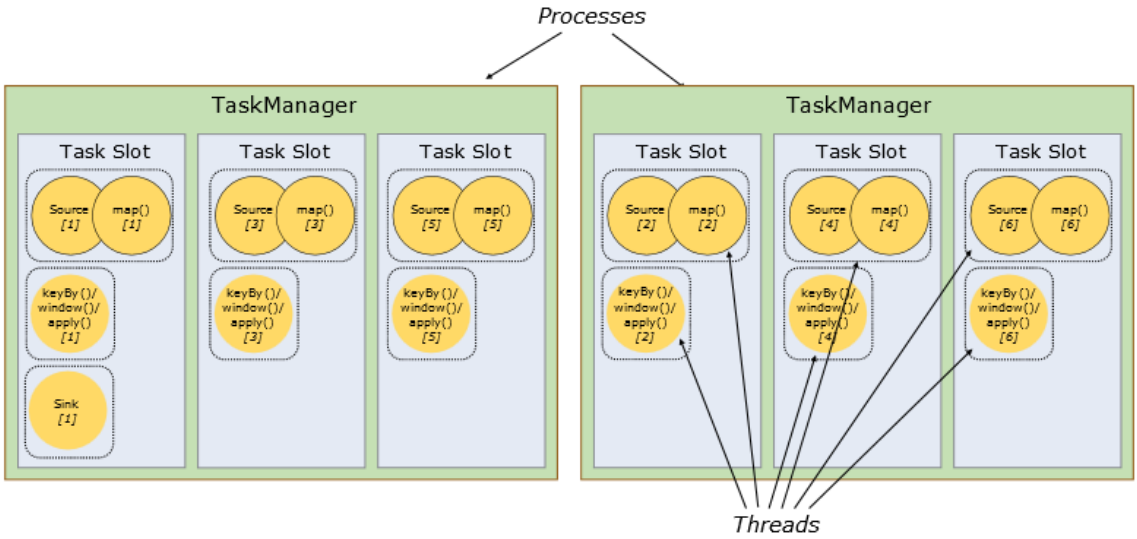
Workers, Slots, Resources

每一个TaskManager是个jvm,每个jvm中可以执行一个或者多个subtasks,jvm中taskSlot的数量决定了接受多少个task;每个tasksolt都有固定的资源，比如TaskManager有三个task solts,taskManager把平均把管理的内存分配到三个task slot中，这样solt中的task不会跟其他的job竞争资源；默认上Flink许可subtasks去分享同一个是slots;但要保证这些subtask是不同的task，并且来自相同的Job；极端情况下，一个slot中执行整个job的task；solt分享有两个重要的好处：

1：flink 机群中可以用到的最高的并行计算数，就是taskSolt的数量

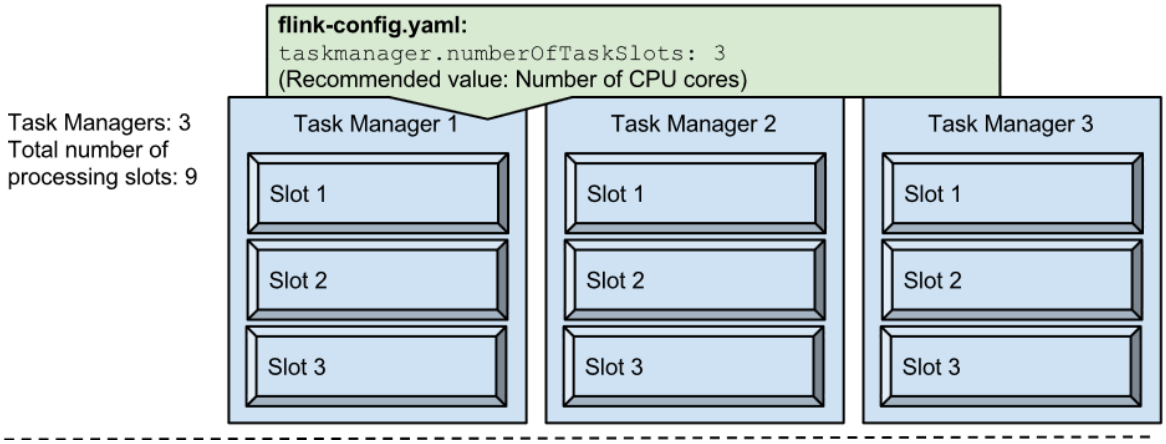
2：可以容易的达到资源利用；

solt 资源共享是可以在api中设置种控制；resource group机制可以设置哪些tasks 共享slots;

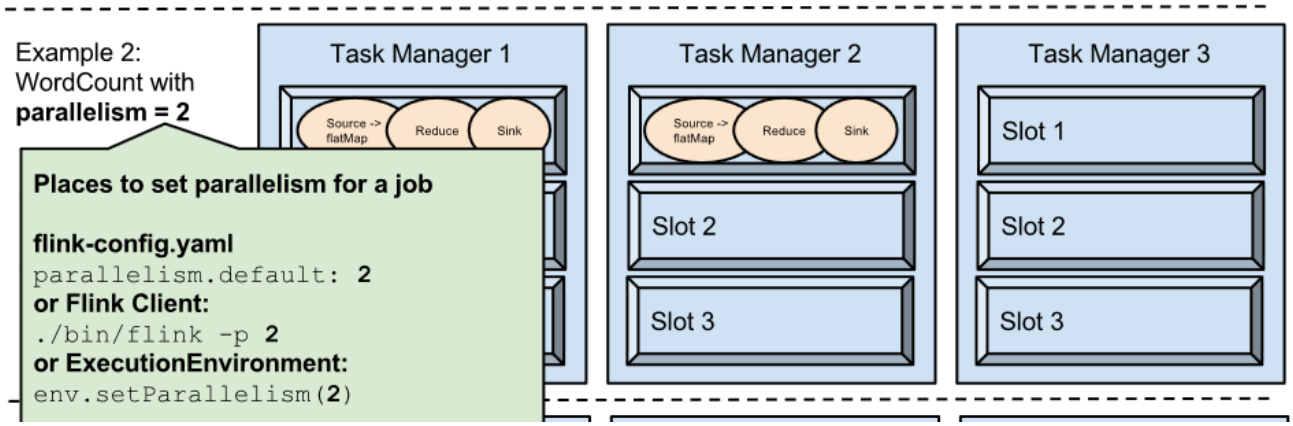


Configuring TaskManager processing slots

slot的数量通常设置是成正比的每个TaskManager可用CPU核的数量；一般建议,可用CPU核的数量正好是taskmanager.numberOfTaskSlots的数量，当开始Flink应用程序中,用户可以提供slot的数量，可以在命令中加入-p(for paralleism)参数指定；另外也可以在API中设置；例如taskManager有三台机器，并在flink-config.yaml中设置taskmanager.numberOfTaskSlots:3(建议是cpu的核数)；这样每台机器有3个slot,机器中共有9个processing taskslots,见下图所示：



当设置parallelism.default:2 或者启动的时候指定-p参数-./bin/flink -p2 或者代码中设置env.setParallelism(2),那边task分配如下图所示：



当然也可以把某个operator的并行度另外设置，比如把sink的并行度设置为1，那多task分配就会如下图：