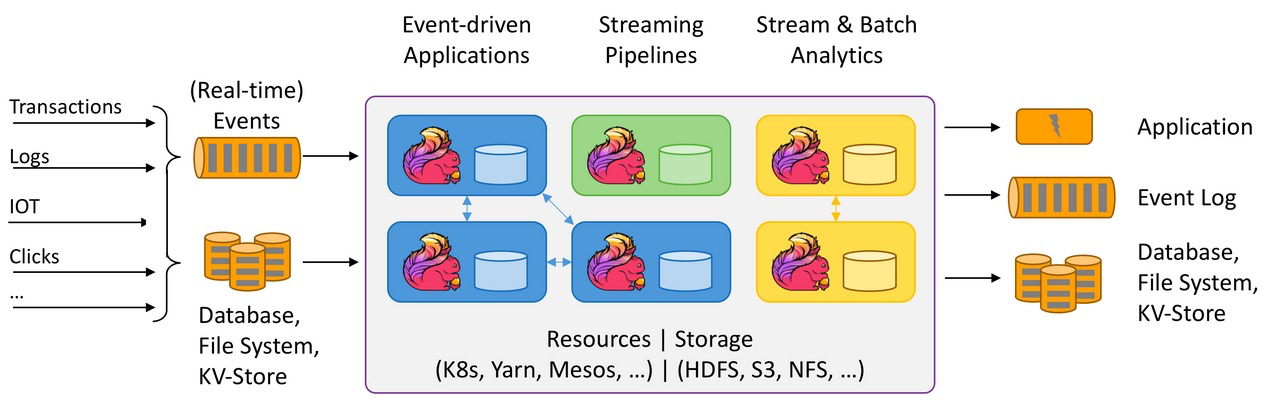
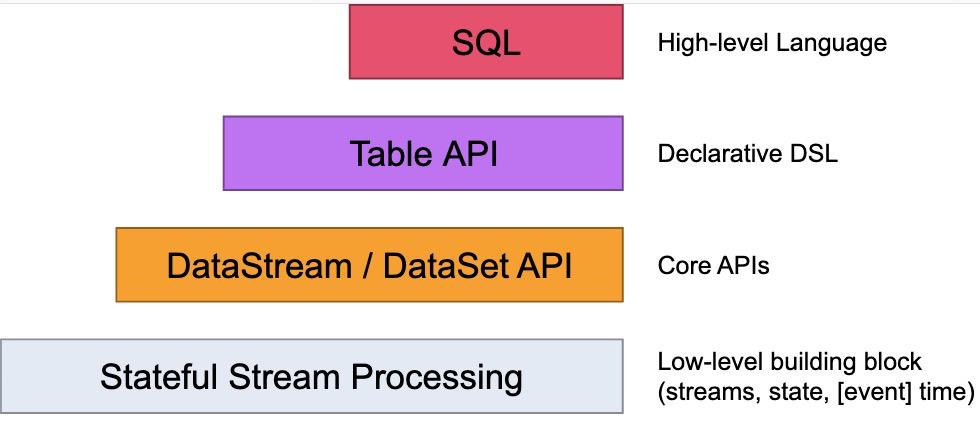
# Apache Flink简介

## 概念[1][2]

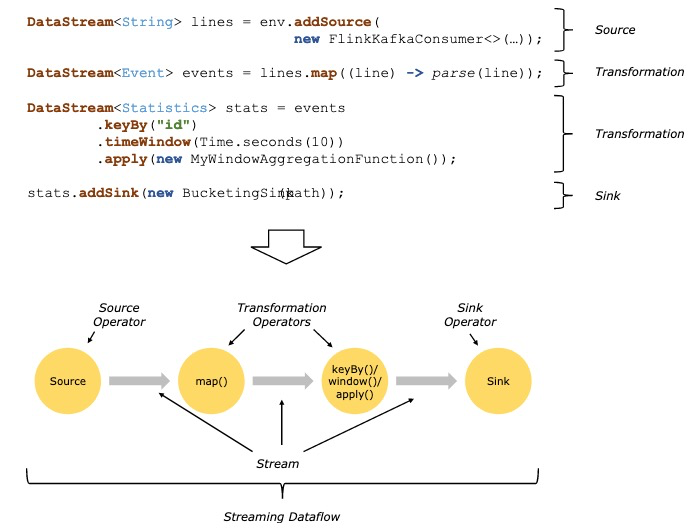
### 基础



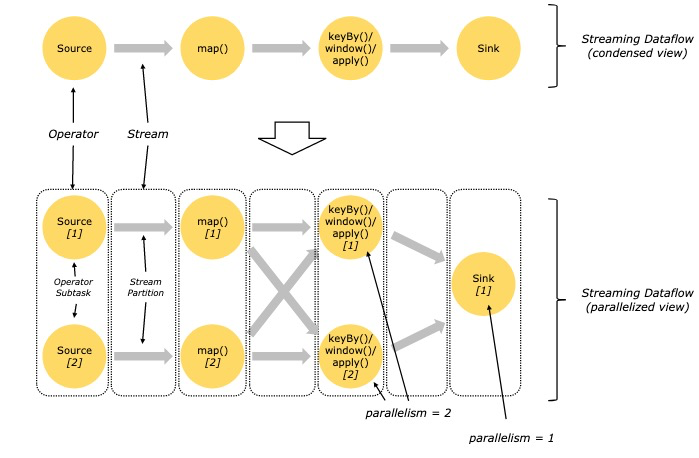
1. 数据流的有状态计算



1. 流/批处理应用的抽象层级



1. 程序与数据流

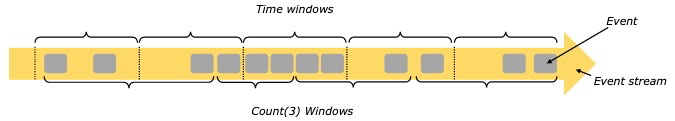


1. 并行数据流

### Windows

汇聚事件（如计数和求和）在流处理和批处理场景计算方式不同。流通常是无界的。因此流计算常以windows界定范围。如过去10秒的访问计数等

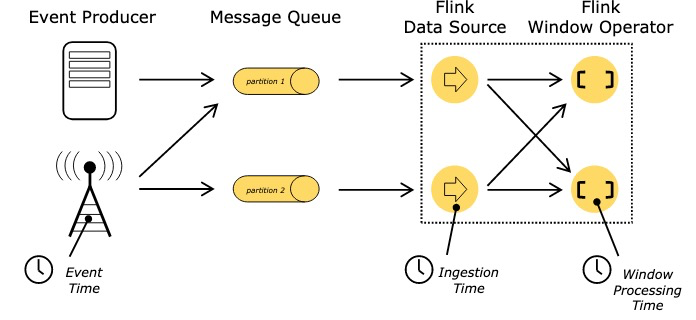
Windows可以是时间驱动（每10秒）或者数据驱动（每100个elements）。Windows可以分为tumbling windows (无重叠), 滑动windows (有)和 session windows (通过非活跃区间截断)。



1. Windows

### Time

事件事件-->聚集时间-->处理时间



1. 时间

### 有状态运算(operations)

不仅仅处理一个事件，而是考虑多个（如时间窗口等）。

### 检验点(checkpoints)

Flink通过流重放(stream replay)和检验点(checkpointing)实现容错(fault tolerance)

### 批处理

Flink将批处理作为一种特定流应用的方式，即作为有界的流应用。批处理容错不使用校验点方式。虽然避免了校验点的负担，但是需要重点考虑数据恢复（容错处理）。

## 数据流容错

Flink提供一种容错机制，可以一致性恢复数据流应用的状态。这种机制甚至可以保证出现错误的情况下，应用的状态仍旧可以最终精确性唯一地从数据流反映每一个数据记录。注意我们有切换选项降级到至少一次。

这种容错机制持续性的生成分布式流数据的快照。对于少量状态的流应用，这些快照是极其轻量的，可以频繁生成而不需要考虑对性能的影响。流应用的状态保存位置是可配置的（如集群的主节点或者HDFS）。

发生程序失败时（源于主机、网络或者软件失败），Flink停止分布式流处理。系统会重新启动算子并将它们重置为最近成功的检验点。输入流重置为最近的状态快照。重启并行数据流过程中处理的记录确保不属于之前检验点状态。

校验点默认关闭。

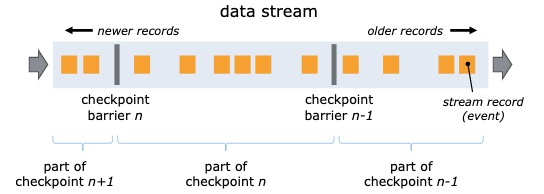
为了实现校验点机制，数据流原必须支持可以将流恢复到所设定的点。如Kafka。

### 检验点

Flink容错机制的核心即是生成分布式数据流和算子状态的一致性快照。这些快照作为系统失败时可以依赖的一致性校验点。生成快照的机制和算法可以参考[3][4]。

Stream Barriers

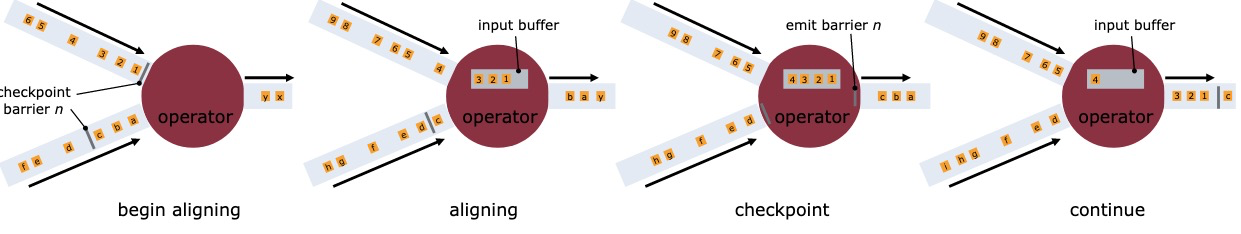
Stream Barriers注入到数据流中并作为数据流的一部分随记录流动。Barriers不会超越记录，而是严格排队处理。Barriers将数据流中的记录划分为不同的集合，确定进入不同的快照。每个Barrier携带前面记录的快照ID。Barriers不会中断流因此是轻量级的。在同一时间流中肯存在不同快照的多个barriers，这意味着不同快照可以并发产生。



1. Stream Barriers

Flink的JobManager是检测点协调者。当中间算子收到所有输入流快照n的barrier。它会将快照n的barrier发送所有输出流。一旦sink算子收到所有输入流的barrier n，它会发确认消息到检测点协调者。当所有sink确认了这个快照，快照就确定完成了。

一旦快照n完成，job便不会向数据源请求快照n之前的数据记录。这时，所有检测点之前的数据记录已经通过了计算拓扑。



1. 快照对齐

状态

状态包含用户自定义状态和系统状态。

当运算子收到所有输入流的快照barrier之后，在将barrier发送到输出流之前，它将当前点的状态取一个快照保存。此时，当前快照的所有状态修改都应已经完成。状态可以报存在可配置的存储后端，如HDFS。当状态保存完成后，算子确认检测点，并将快照barrier发送到输出流，然后计算继续进行。

生成的快照包含：

对每个并行流数据源，当前快照启动的偏移/位置。

对每个算子，指向其状态的指针。

精准一次性和至少一次性

快照对齐将引入延迟。Flink可以在检测点时跳过对齐。检测点快照仍旧在算子收到所有输入源的检测barrier时生成。

忽略对齐时，算子将持续处理所有输入，即使是收到部分检测点n的barrier后。这样，检测点n和n+1的数据记录将同时处理。恢复时，检测点n+1的数据将会重复，它们同时存在于检测点n和检测点n之后的数据重放中。

异步状态快照

上述描述的机制意味着当算子存储状态快照时它会停止处理输入记录。这种同步状态快照在生成快照时引入了延迟。如果存储快照时不停止处理数据，那么状态快照可以异步进行。异步模式要求数据的一致性，因此要考虑copy-on-write这样的数据结构（RocksDB中使用）

恢复

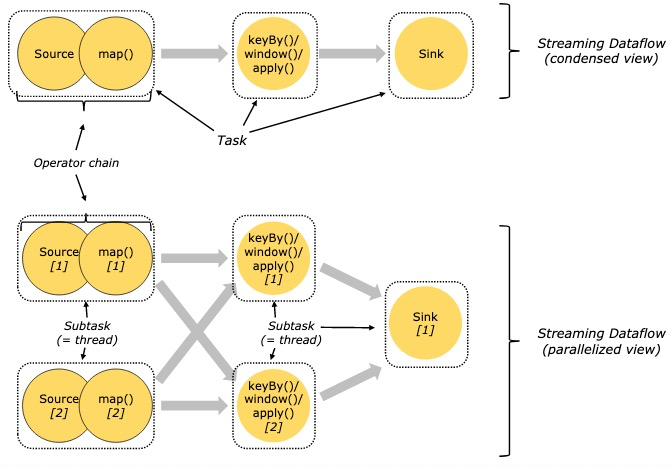
选择最近完成的检测点恢复计算

Operators and state backends provide their snapshots as a Java FutureTask.

## 分布式运行环境

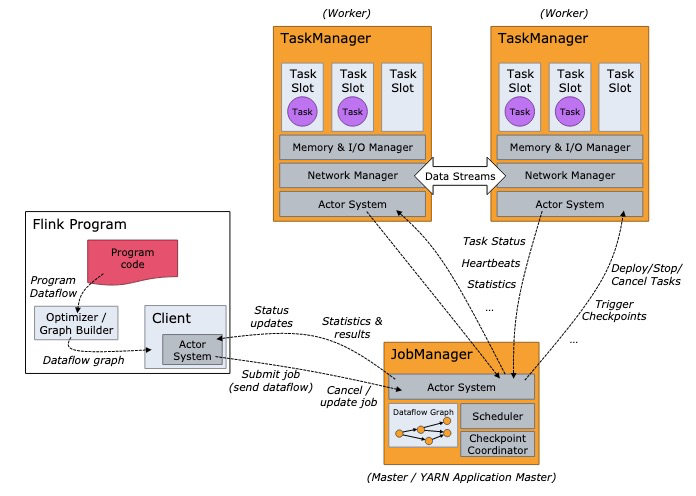
任务和算子链

Flink将算子子任务链接起来形成任务在分布式环境执行。每个任务由一个线程执行。将算子链接成任务减少了线程间切换的负担和缓存，增加吞吐量的同时降低了延迟。



1. 链接子任务

Job管理器，任务管理器和客户端



1. 运行时环境

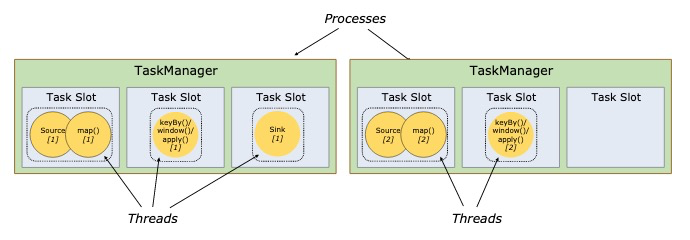
工作管理器：也叫做master，可以高可靠部署，其中一个为leader。

任务管理器：也叫做worker，执行任务（子任务），缓存，交换数据流。

任务槽和资源

每个worker是一个JVM进程。可以在不同线程执行一个或者多个子任务。任务槽控制一个worker可以执行多少任务。

每个任务槽表示任务管理器的一个固定的资源子集。例如，如果一个任务管理器有三个槽，那么将分配每个槽位1/3的内存。分槽意味着子任务间不会竞争内存资源。当前只管理内存。

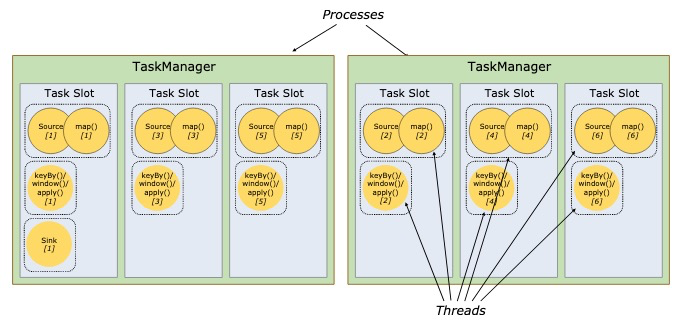


1. 任务管理器与任务槽

默认情况，Flink允许来自同一工作的子任务共享槽位。结果是，一个槽位可能持有工作的整个流水线。槽位共享有两个主要优点：

只需要使用槽位确定任务并行级别，而不用考虑一个程序包含多少任务。

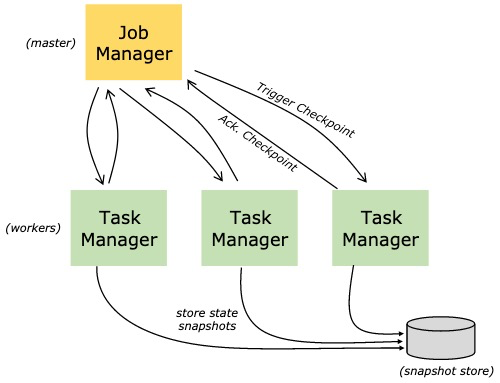
易于优化资源配置。



1. 槽位共享

此外可以使用API禁止槽位共享。

状态存储后端



1. 状态存储后端

存储点（HDFS/NFS...）

存储点是手动触发的检测点。

1. <http://research.microsoft.com/en-us/um/people/lamport/pubs/chandy.pdf>
2. <https://ci.apache.org/projects/flink/flink-docs-release-1.9/internals/stream_checkpointing.html>
3. <http://arxiv.org/abs/1506.08603>
4. <http://research.microsoft.com/en-us/um/people/lamport/pubs/chandy.pdf>
5. <https://ci.apache.org/projects/flink/flink-docs-release-1.9/concepts/runtime.html>