Apache Flink数据流容错机制

Introduce

Apache Flink 提供了可以恢复数据流应用到一致状态的容错机制。确保在发生故障时，程序的每条记录只会作用于状态一次（exactly-once），当然也可以降级为至少一次（at-least-once）。

容错机制通过持续创建分布式数据流的快照来实现。对于状态占用空间小的流应用，这些快照非常轻量，可以高频率创建而对性能影响很小。流计算应用的状态保存在一个可配置的环境，如：master 节点或者 HDFS上。

在遇到程序故障时（如机器、网络、软件等故障），Flink 停止分布式数据流。系统重启所有 operator ，重置其到最近成功的
checkpoint。输入重置到相应的状态快照位置。保证被重启的并行数据流中处理的任何一个 record 都不是 checkpoint 状态之前的一部分。

注意：为了容错机制生效，数据源（例如 消息队列 或者
broker）需要能重放数据流。Apache Kafka 有这个特性，Flink 中 Kafka 的 connector
利用了这个功能。

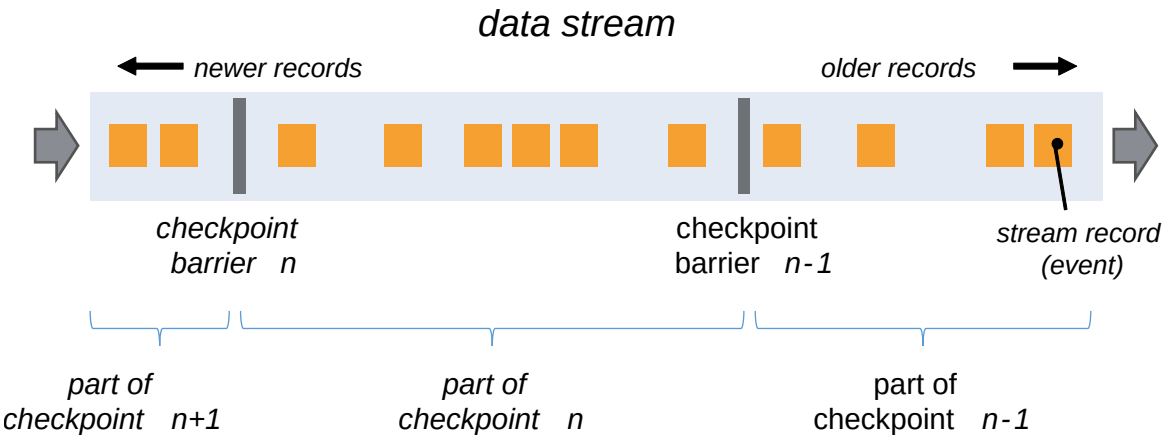
注意：由于 Flink 的 checkpoint 是通过分布式快照实现的，接下来我们将 snapshot 和 checkpoint 这两个词交替使用。

Checkpointing

Flink 容错机制的核心就是持续创建分布式数据流及其状态的一致快照。这些快照在系统遇到故障时，充当可以回退的一致性检查点（checkpoint）。Lightweight Asynchronous Snapshots for Distributed Dataflows 描述了Flink 创建快照的机制。此论文是受分布式快照算法 Chandy-Lamport 启发，并针对 Flink 执行模型量身定制。

Barriers

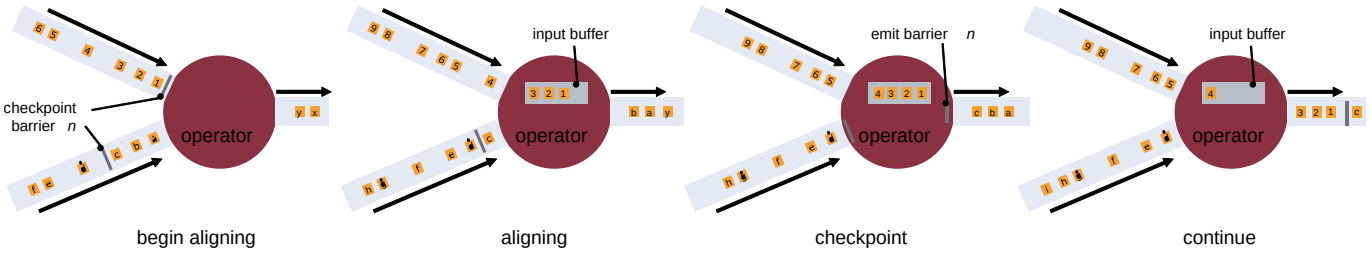
Flink 分布式快照的核心概念之一就是数据栅栏（barrier）。这些 barrier
被插入到数据流中，作为数据流的一部分和数据一起向下流动。Barrier
不会干扰正常数据，数据流严格有序。一个 barrier
把数据流分割成两部分：一部分进入到当前快照，另一部分进入下一个快照。每一个 barrier
都带有快照 ID，并且 barrier 之前的数据都进入了此快照。Barrier
不会干扰数据流处理，所以非常轻量。多个不同快照的多个 barrier
会在流中同时出现，即多个快照可能同时创建。



如果想及时了
解Spark、Hadoop或者Hbase相关的文章，欢迎关注微信公共帐号：iteblog_hadoop

Barrier 在数据源端插入，当 snapshot n 的 barrier 插入后，系统会记录当前 snapshot 位置值 n (用Sn表示)。例如，在 Apache Kafka
中，这个变量表示某个分区中最后一条数据的偏移量。这个位置值 Sn 会被发送到一个称为 checkpoint coordinator 的模块。(即 Flink 的 JobManager).

然后 barrier 继续往下流动，当一个 operator 从其输入流接收到所有标识 snapshot n 的 barrier 时，它会向其所有输出流插入一个标识 snapshot n 的 barrier。当 sink operator （DAG 流的终点）从其输入流接收到所有 barrier n 时，它向 the checkpoint coordinator 确认 snapshot n 已完成。当所有 sink 都确认了这个快照，快照就被标识为完成。



如果想及时了
解Spark、Hadoop或者Hbase相关的文章，欢迎关注微信公共帐号：iteblog_hadoop

接收超过一个输入流的 operator 需要基于 barrier 对齐（align）输入。参见上图：

1、operator 只要一接收到某个输入流的 barrier
n，它就不能继续处理此数据流后续的数据，直到 operator 接收到其余流的 barrier n。否则会将属于 snapshot n 的数据和 snapshot n+1的搞混

2、barrier n
所属的数据流先不处理，从这些数据流中接收到的数据被放入接收缓存里（input buffer）

3、当从最后一个流中提取到 barrier n 时，operator

会发射出所有等待向后发送的数据，然后发射snapshot n 所属的 barrier

4、经过以上步骤，operator 恢复所有输入流数据的处理，优先处理输入缓存中的数据

State

operator 包含任何形式的状态，这些状态都必须包含在快照中。状态有很多种形式

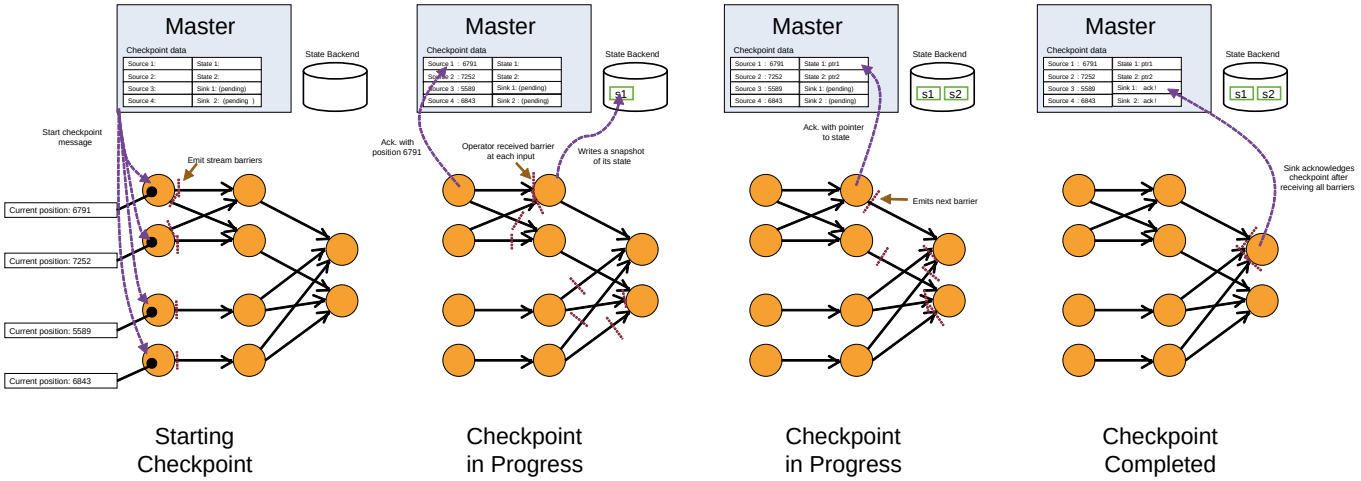
1、用户自定义状态：由 transformation 函数例如（ map() 或者 filter())直接创建或者修改的状态。用户自定义状态可以是：转换函数中的 Java 对象的一个简单变量或者函数关联的 key/value 状态。

2、系统状态：这种状态是指作为 operator 计算中一部分缓存数据。典型例子就是：窗口缓存（window buffers），系统收集窗口对应数据到其中，直到窗口计算和发射。

operator 在收到所有输入数据流中的 barrier 之后，在发射 barrier 到其输出流之前对其状态进行快照。此时，在 barrier 之前的数据对状态的更新已经完成，不会再依赖 barrier 之前数据。由于快照可能非常大，所以后端存储系统可配置。默认是存储到 JobManager 的内存中，但是对于生产系统，需要配置成一个可靠的分布式存储系统（例如 HDFS）。状态存储完成后，operator 会确认其 checkpoint 完成，发射出 barrier 到后续输出流。

快照现在包含了：

1、对于并行输入数据源：快照创建时数据流中的位置偏移
2、对于 operator：存储在快照中的状态指针



如果想及时了解Spark、Hadoop或者Hbase相关的文章，欢迎关注微信公共帐号：iteblog_hadoop

Exactly Once vs. At Least Once

对齐操作可能会对流程序增加延迟。通常，这种额外的延迟在几毫秒的数量级，但是我们也遇到过延迟显著增加的异常情况。针对那些需要对所有输入都保持毫秒级的应用，Flink 提供了在 checkpoint 时关闭对齐的方法。当 operator 接收到一个 barrier 时，就会打一个快照，而不会等待其他 barrier。

跳过对齐操作使得即使在 barrier 到达时，Operator 依然继续处理输入。这就是说：operator 在 checkpoint n 创建之前，继续处理属于 checkpoint n+1 的数据。所以当异常恢复时，这部分数据就会重复，因为它们被包含在了 checkpoint n 中，同时也会在之后再次被处理。

注意：对齐操作只会发生在拥有多输入运算（join)或者多个输出的 operator（重分区、分流）的场景下。所以，对于自由 map(), flatmap(), fliter() 等的并行操作即使在至少一次的模式中仍然会保证严格一次。

Asynchronous State Snapshots

我们注意到上面描述的机制意味着当 operator 向后端存储快照时，会停止处理输入的数据。这种同步操作会在每次快照创建时引入延迟。

我们完全可以在存储快照时，让 operator 继续处理数据，让快照存储在后台异步运行。为了做到这一点，operator 必须能够生成一个后续修改不影响之前状态的状态对象。例如 RocksDB 中使用的写时复制（copy-on-write ）类型的数据结构。

接收到输入的 barrier 时，operator 异步快照复制出的状态。然后立即发射 barrier 到输出流，继续正常的流处理。一旦后台异步快照完成，它就会向 checkpoint coordinator（JobManager）确认 checkpoint 完成。现在 checkpoint 完成的充分条件是：所有 sink 接收到了 barrier，所有有状态 operator 都确认完成了状态备份（可能会比 sink 接收到 barrier 晚）。

Recovery

在这种容错机制下的错误回复很明显：一旦遇到故障，Flink 选择最近一个完成的 checkpoint k。系统重新部署整个分布式数据流，重置所有 operator 的状态到 checkpoint k 。数据源被置为从 S_k 位置读取。例如在 Apache Kafka 中，意味着让消费者从 S_k 处偏移开始读取。

如果是增量快照，operator 需要从最新的全量快照回复，然后对此状态进行一系列增量更新。

Operator Snapshot Implementation

当 operator 快照创建时有两部分操作：同步操作和异步操作。

operator 和后端存储将快照以 Java FutureTask 的方式提供。这个 task

包含了同步操作已经完成，异步操作还在等待的状态（state）。异步操作在后台线程中被执行。

完全同步的 operator 返回一个已经完成的 FutureTask
。如果异步操作需要执行，FutureTask 中的 run() 方法会被调用。

为了释放流和其他资源的消耗，可以取消这些 task。

本文转载自：Apache Flink 容错机制
英文原文：Data Streaming Fault Tolerance