Flink 原理（六）——异步I/O（asynchronous I/O）

目录

回到顶部

1、前言

本文是基于Flink官网上Asynchronous I/O的介绍结合自己的理解写成的，**若有不正确的欢迎大伙留言交流，谢谢**！

回到顶部

2、Asynchronous I/O简介

将Flink用于流计算时，若涉及到和外部系统进行交互，如利用Flink从数据库中读取数据，这种需要获取I/O的场景时，我们需要考虑交互所带来的时延问题。

为分析如何减少时延，我们先来分析一下，Flink以同步的形式方法外部系统（以MapFunction中和数据库交互为例）的过程，若图1虚线左侧所示，请求a发送到database后，MapFunction等待回复后才进行下发送下一个请求b，期间，I/O处于空闲状态，请求b又开始重复此过程，这样在两个来回的时间内（发送请求-收到结果为一个来回），只处理两个请求。如图1虚线右侧所示，同样是在两个来回的时间内，以异步的形式进行交互，请求a发出去后，在等待回复时，请求b,c,d依次发出，这样既可以处理4个请求了。

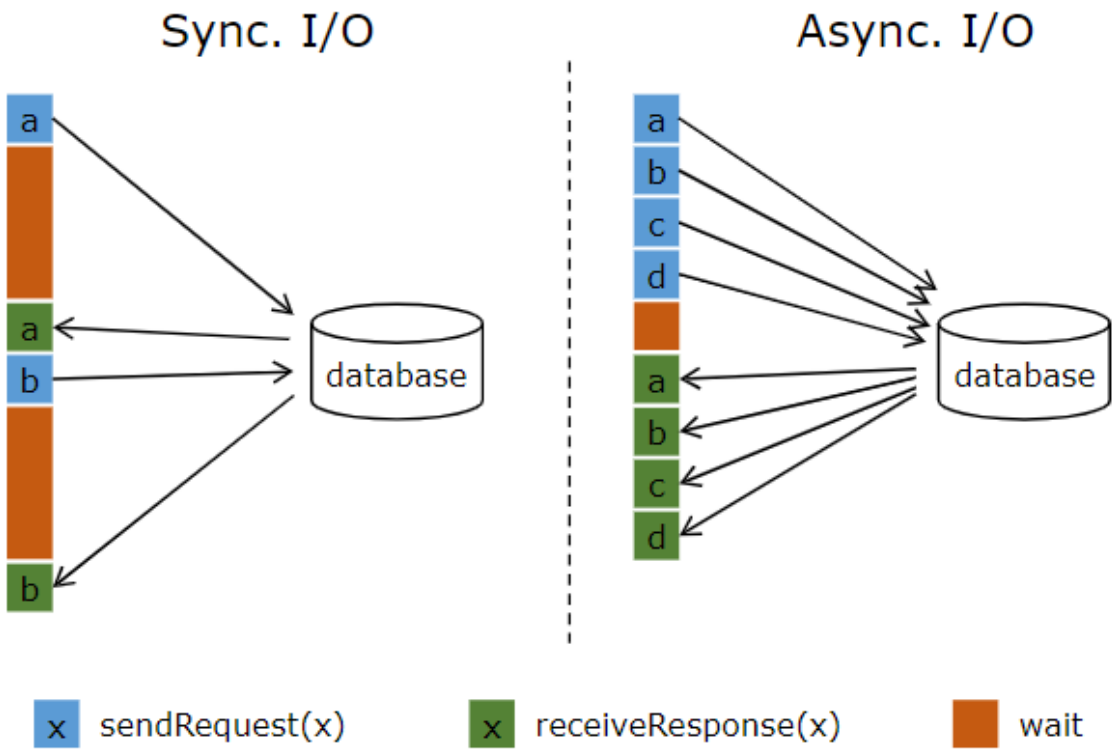

图1 同/异访问数据库方式对比图(**Ref[1]**)

在某些场景下，为了提高系统的吞吐能力，可以仅通过增大MapFunction的并发度以达目的，但是随之而来是资源的大量消耗。

【重要事项】

1）为了实现以异步I/O访问数据库或K/V存储，数据库等需要有能支持异步请求的client；若是没有，可以通过创建多个同步的client并使用线程池处理同步call的方式实现类似并发的client，但是这方式没有异步I/O的性能好。

2）AsyncFunction不是以多线程方式调用的，一个AsyncFunction实例按顺序为每个独立消息发送请求；

3）目前（Flink 1.9），使用AsyncWaitOperator时要打断operator chain（默认也是不使用），原因见`FLINK-13063`。

回到顶部

3、结果的顺序

由于请求响应的快慢可能不一样，AsyncFunction的"并发"请求可能导致结果的乱序 。如图1中虚线右侧所示，若请求b发出之后，其结果在请求a的之前返回，这样异步I/O算子前后的消息顺序就不一致了。为了控制结果的返回顺序，Flink提供了两种模式：

1）**Unordered**：当异步的请求完成时，其结果立马返回，不考虑结果顺序即乱序模式。当以processing time作为时间属性时，该模式可以获得最小的延时和最小的开销，使用方式：AsyncDataStream.unorderedWait(...)；

2）**Ordered**：该模式下，消息在异步I/O算子前后的顺序一致，先请求的先返回，即有序模式。为实现有序模式，算子将请求返回的结果放入缓存，直到该请求之前的结果全部返回或超时。该模式通常情况下回引入额外的时延以及在checkpoint过程中会带来开销，这是因为，和无序模式相比，消息和请求返回的结果都会在checkpoint的状态中维持更长时间。使用方式：AsyncDataStream.orderedWait(...)；

在此，我们需要针对流任务和event time相结合的情况进行补充说明。为什么？是因为watermark和消息的整体相对位置是不会变的，什么意思了？发生在某个watermark之后的消息，只能在watermark被发出之后发出，其请求结果也是。换句话说，两个watermark之间的消息整体与watermark的有序的。当然这个区间内消息之间是否有序这得根据使用的模式来分析。

1）对Ordered模式，因为消息本身是有序的，所以watermark和消息之间也是有序的，和processing time相比，其不需要引入额外的开销；

2）对Unordered模式，其模式是先响应先返回，但在与event time结合的情况里，消息或结果都需在特定watermark发出之后才能发出，此时，就会引入延时和开销，其开销的大小取决于watermark的频率，其原因参加下文原理部分。

回到顶部

4、原理

4.1 Terms

为更加详细的说明异步I/O的实现过程，先说明几个term，其中也会涉及其基本用法，若分析原理只看其含义即可。

1)AsyncFunction：异步I/O的触发接口

AsyncFunction在AsyncWaitOperator中作为一个用户函数，类似FlatMap，有open()/processElement(StreamRecord< in > record)/processWatermark(Watermark mark)方法。
对于用户自己实现的AsyncFunction，必须重写asyncInvoke(IN input, AsyncCollector collector)来提供调用异步操作的代码。

2）AsyncWaitOperator：调用AsyncFunction的流算子，是个抽象的概念，具体算子是unorderedWait(...)或orderedWait(...)

3）AsyncCollector:

AsyncCollector由AsyncWaitOperator创建，并传递给AsyncFunction，在这里它应该被添加到用户的回调函数中。它充当从用户代码中获取结果或错误的角色，并通知AsyncCollectorBuffer发出结果。

4）AsyncCollectorBuffer：AsyncCollectorBuffer保存所有的AsyncCollector，并将结果发送给下一个节点。

上述概念是工作示意图可参见**Ref[2]**。

4.2 架构图

在流式计算中，涉及异步I/O的整体过程图如下：

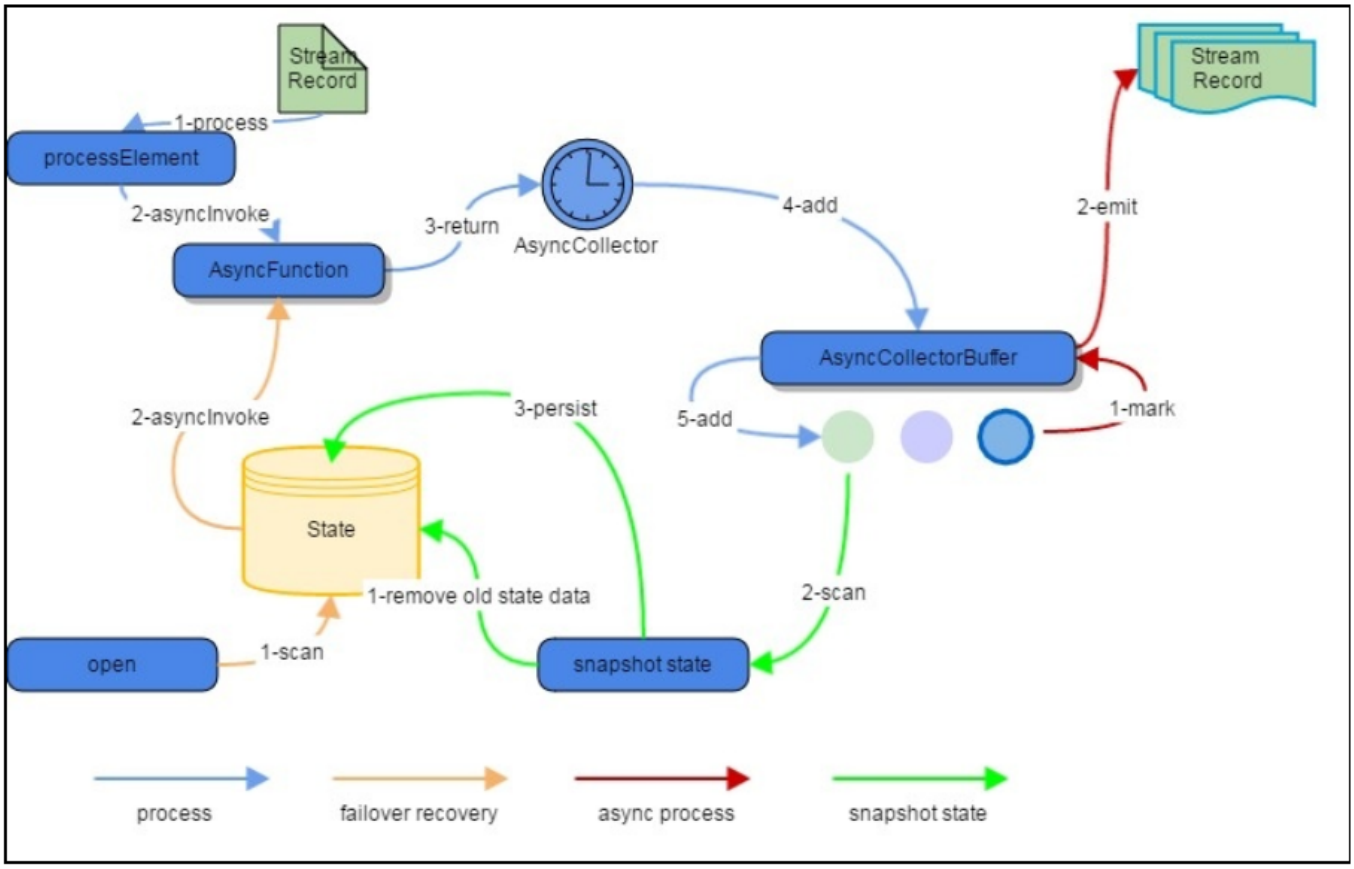

图2 异步I/O架构图(**Ref[2]**)

1）消息达到AsyncWaitOperator后正常处理过程如下：

AsyncWaitOperator调用AsyncFunction，并创建AsyncCollector传递给AsyncFunction。AsyncCollector等待获取到返回结果（异常）之后将入到AsyncCollectorBuffer保存时，会将一条mark消息放入AsyncCollectorBuffer中，然后一个signal信息将会发送到Emitter 线程，若此时是将消息发送出去的signal，则会将消息发送出去并通知task thread加消息到collector buffer中。至于怎么发要依据代码中设置的模式是有序还是无序，若是有序则发head，删head。该过程的更详细过程如下图：

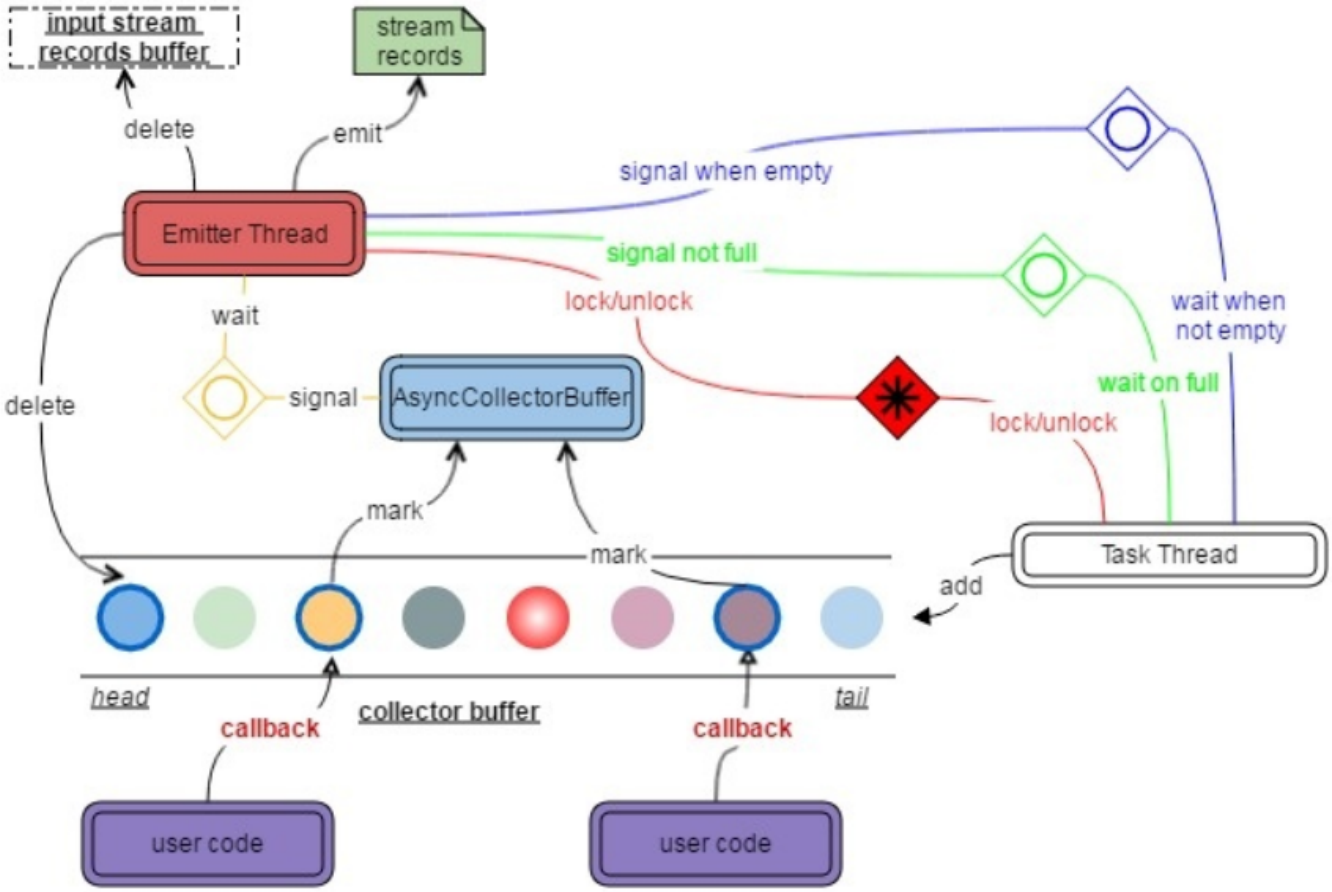

图3 异步I/O正常处理消息图(**Ref[2]**)

2）checkpoint过程

AsyncWaitOperator先是对AsyncCollectorBuffer中所有的输入流数据进行扫描，完成后就删除state中老的数据，然后将AsyncCollectorBuffer中数据存入到state中，而不是在处理时对单个输入流一个接一个的存入state，具体过程图见图2或图4。

3）故障恢复

在恢复AsyncWaitOperator的状态时，AsyncWaitOperator将scan状态中的所有元素，获取AsyncCollectors，调用AsyncFunction.asyncInvoke()并将它们插入AsyncCollectorBuffer中，具体的如下：

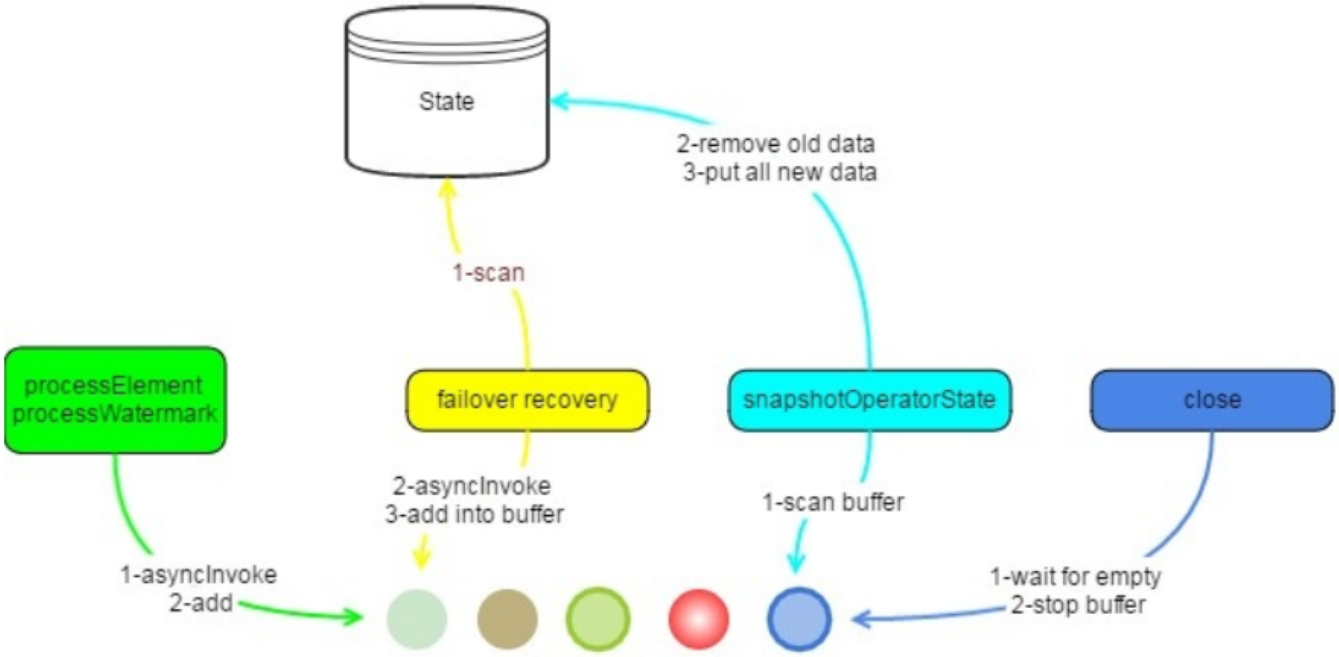

图4 故障恢复和checkpoint流程图(**Ref[2]**)

总结:

关于具体使用的方法见后期的博客，建议大伙看看原文，一千个读者就有一千个哈姆雷特!

回到顶部

Ref

[1]https://ci.apache.org/projects/flink/flink-docs-release-1.9/dev/stream/operators/asyncio.html

[2]https://cwiki.apache.org/confluence/pages/viewpage.action?pageId=65870673

[3]https://blog.icocoro.me/2019/05/26/1905-apache-flinkv2-asyncio/