Flink 在大数据平台建构中的实践与踩过的坑*

刘河 刘娅

重庆市教育科学研究院，重庆 400015 重庆市石柱民族中学校，重庆 409100

摘 要 为了建构同时处理流数据和批量数据实时计算平台的目的,采用实验验证、综合分析方法,做了Flink和Spark Structured Streaming 对比,在 Yarn 上部署、Flink 结合 Spring、Flink 异步不支持 KeyedState、Flink CEP 的使用的实验,获得Flink能将流处理和批处理二者有机融合起来的结果,得到Flink同时满足低延迟,Exactly-once保证、高吞吐、高效处理的结论。

关键字 Flink，流数据，批量数据，计算引擎

Practice and Pitfalls of Flink in the Construction of Big Data Platforms

LIU He LIU Ya

Chongqing Academy of Educational Sciences Chongqing Shizhu Ethnic Middle Schoo
Chongqing 400015，China Chongqing Shizhu 409100, China
bjxiexintong@126.com

Abstract—In order to construct a real-time computing platform for processing stream data and batch data simultaneously; Using experimental verification and comprehensive analysis methods; Experiments were conducted to compare Flink with spark structured streaming, deploy it on Yan, combine Flink with spring, and Flink asynchronously does not support the use of keyedstate and Flink CEP; Get the result that Flink can organically integrate stream processing and batch processing; It is concluded that Flink meets the requirements of low latency, exactly once guarantee, high throughput and efficient processing.

Keywords—Flink, Stream data, Batch data, Computing engine

1 引 言

Apache Flink是面向分布式数据流处理和批量数据处理的开源计算引擎，能基于同一 Flink 运行时（Flink Runtime）提供流处理和批处理API。Flink在实现流处理和批处理时与传统方案完全不同，它将流处理和批处理二者统一起来：Flink 完全支持流处理，作为流处理时输入数据流是无界的；批处理作为一种特殊的流处理，输入数据流被定义为有界的。流处理需要支持低延迟、Exactly-once保证，而批处理需要支持高吞吐、高效处理。实现批处理的开源方案有 MapReduce、Spark 等，实现流处理的开源方案有Storm等。从四个方面研究了 Flink 在平台建构中一些实践应用：实时平台架构、Flink 和 spark 的structured streaming 对比、一些踩过的坑和具体的经验以及对 Flink 未来的展望。大数据有其自身的数据特性,平台需要同时处理大量流数据和批量数据,研究同时支持两种数据类型处理的计算框架对大数据平台建构具有重要现实意义。

*基金资助：本文得到重庆市教委科学技术研究计划重点项目“面向智慧教育的类脑网络理论与关键技术研究”（KJZD-K202114401）；重庆市科研机构绩效激励引导专项“面向智慧教育的 Spiking 神经网络理论与方法研究” （受理编号：cstc2022jxjl0214）；重庆市教育科学规划课题一般课题“网络谣言治理及机器生成文本检测研究”(k23YG6020127)资助。

2 大数据实时平台架构

大数据的实时平台架构主要分为数据源、计算引擎、存储引擎、实时 OLAP 和作业平台，如图1所示。

2.1 数据源

数据源主要是数据中间件,除了采用 kafka 以外还使用了NSQ。NSQ 是 go 语言编写的数据中间件，我们的大数据平台建构的消息团队封了一层 java 的Client，这个 Client 是使用 push 和 ack 模式，所以在做 Connector 的时候跟 kafka 也会有比较大的差距，我们主要参考的是 flink 官方提供的rabbit mq connector 的实现[1],然后做了一些改造。

2.2 计算引擎

在计算引擎方面，我们最早是采用 storm，现在还是运行在standalone 集群上。因为 storm 的开发

成本相对会比较高，并且它的吞吐量也不是很大，它的中间状态的管理也没有什么优势，另一方面因为它的 SQL 支持不是很好，所以第二个实时计算引擎就采用 spark Streaming。因为它是 spark 生态中的一环，所以对大多数的开发者来说相对更加熟悉一些。但 spark steaming 它本身是基于微批引擎，它的延时很难达到这种纯流式引擎的效果。最后我们第三个计算引擎引入的是 flink。flink 的调研是从 2018 年 5 月份开始的，在文章后面我会介绍，我们为什么还要再引入一个新的计算引擎。

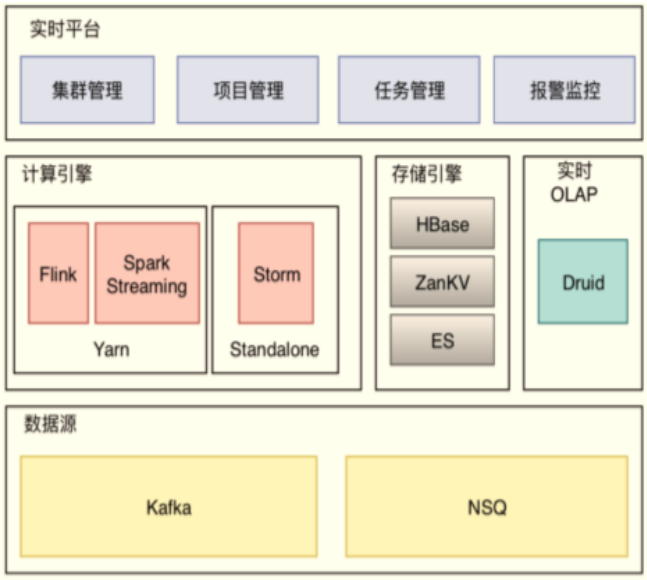

图 1 大数据实时平台架构

2.3 存储引擎

存储引擎除了 HBase 和 ES 以外，我们还有一个 ZanKV[2]。ZanKV 兼容 Redis 协议，是一个分布式的 KV 数据库。

2.4 实时 OLAP

实时 OLAP 采用 Druid。

2.5 作业平台

作业平台主要是集群的管理、项目的管理、任务的管理和报警监控[3]的功能。

3 Flink 探索阶段

在探索阶段，我们从四个角度去和 Spark Structured Streaming 对比。为什么要和 Spark Structured Streaming 对比？因为 SQL 化是一个大方向，比较有代表性的引擎就是 Flink 和 Spark Structured Streaming 应用。

3.1 性能

（1）延迟

在性能上，从延迟的角度，因为 Spark Structured Streaming 依旧是基于它的微批引擎，从延迟上肯定是没有办法跟 Flink 竞争的，但是它现在也在开发一个它们自己的一个流式计算引擎，是基于流式的，但总体来说不是很成熟，另外对语义也没有什么保证。

（2）吞吐

在吞吐方面，在一些场景里面 Flink 其实略逊于 Spark Structured Streaming，在一些有状态的场景下可能更优，因为 Spark Structured Streaming 它的中间状态全部驻管在内存，在一些中间状态比较大，吞吐量比较大的任务中，Flink 可以借助 rocksDB 的缓存和持久化，相对来说吞吐量还能更大。

（3） State

至于 State 管理，我们觉得简单的理解，rocksDB 就是一个带缓存的嵌入式的数据库，借助 rocksDB 的磁盘持久化的能力可以去保存更多的状态。

3.2 SQL 支持

（1）一个 query 包含多个聚合

在 SQL 支持上面，我们刚开始使用 Spark Structured Streaming 的时候，写了一个 query，这个 query 里有多个聚合，然后它给我抛了个异常，一个 query 里只能包含一个聚合操作[4]，我去查阅官方文档，它也是这么说的。所以这也是我们放弃 Spark Structured Streaming 的一个原因之一。

（2）Distinct 去重

还有 Distinct 去重，Spark Structured Streaming 是不支持去重，Flink 做得更好。

（3）窗口

Flink 的窗口机制相对来说会更加灵活，同样是参考对它所有的模型，它的输出机制覆盖的场景更多一些。

3.3 灵活性

（1）抽象层次的转换

灵活性主要是抽象层次的转换，首先在 Spark 里，Spark Structured Streaming 里面有一套 API 用 Table[5]，一套API 用的是 DStream，但是它的 Table 和 DStream 是没有办法转换的，就是你一旦选用了 Spark Structured Streaming 只能选用它的 SQL 模块，你什么内容只能写在 UDF 里。在一些复杂场景，

其实这相对来说不会太实用。对 Flink 而言,你使用了 Flink table API,你后续还可以把它转化成 Dataset Streaming,对它做其它的处理。

3.4 复杂事件处理

（1）CEP

最后是复杂事件的处理，就是 CEP 模块，在文章最后会有一个例子去介绍这方面的应用。

4 Flink 在大数据平台建构中的实践

4.1 在 Yarn 上部署

因为本身的技术栈的原因，我们选择了部署在 Yarn 上，我们选择用 Single Job [6]的模式，虽然每个任务都会起一个 Application Master，但是隔离性也会更好一些。刚开始部署 Flink 的时候，我们遇到了一个比较奇怪的问题，就是一个很简单的任务，这个任务也有一些中间状态的管理，我们当时想运行一下，看它是否稳定，然后我上午提交了这个任务，我提交的时候申请了5个 Container,但是到下午的时候我就收到了我们同事的消息，说有个 Flink 任务占了一百多个 Container，然后我们就查找原因，发现它还是只有5个 Container,5个Task Executor 在工作，但是剩下的全部都处于一种 Free 的状态[7]，但是 Single Job 的模式你没有办法在这上面提交一个任务,他这个任务明显就是实际收到的 Container 的数量超过了我配置的数量，并且他没有被回收。

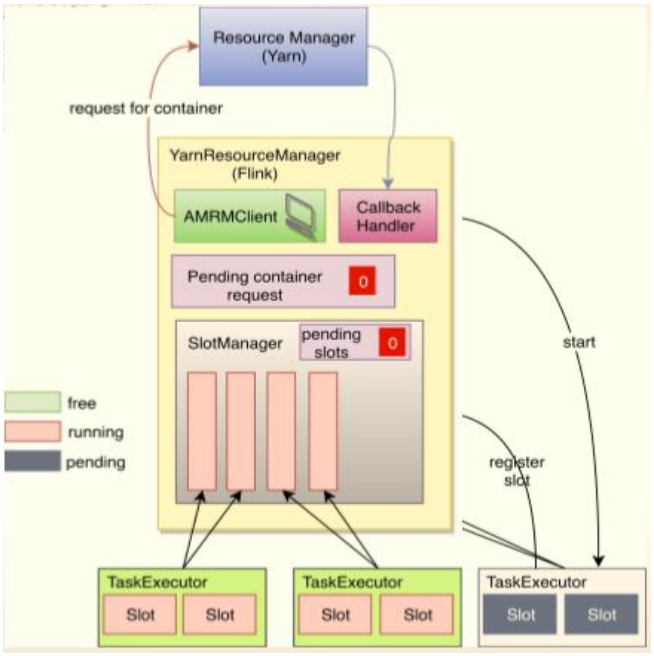

图 2 Flink 整合 Yarn

（1）Flink 如何部署在 Yarn 上

我们先研究Flink 是如何在跟 Yarn 整合到一块的。在图2中，我们从下往上看，TaskExecuter 它是实际任务执行者,它会启动 Slot[8]，然后每一个 Slot 会执行一个具体的子任务，那它会将自己的状态汇报给 SlotManager[9],SlotManager 它既是 Slot 的管理者，它也是负责运行中的任务，向它申请 Slot 的时候,它要给这些运行中的任务提供一个运行的 Slot。

它只管理这些 Slot，它并不会去增加资源，它会向 Flink 的 ResourceManager去申请新的容器，然后在申请的过程中，但是还没有拿到这些 Slots 的时候，它会记录在 Pending slots 里。接下来就是 Flink ResourceManager，它接到了 SlotManager 的请求之后，它会向 Yarn 去申请容器，它也会有一个计数器，就是计数有多少个请求积压在这里，就是我们申请出去了但是 Container 它还没有分配过来。最后它跟 Yarn 的 ResourceManager 交互的时候，通过 AMRMClient 发起申请容器，然后当 Yarn ResourceManager 把资源准备好了之后，Callback Handler 会通知 Flink ResourceManager，或者当容器异常退出的时候，它也会通知 Flink ResourceManager。

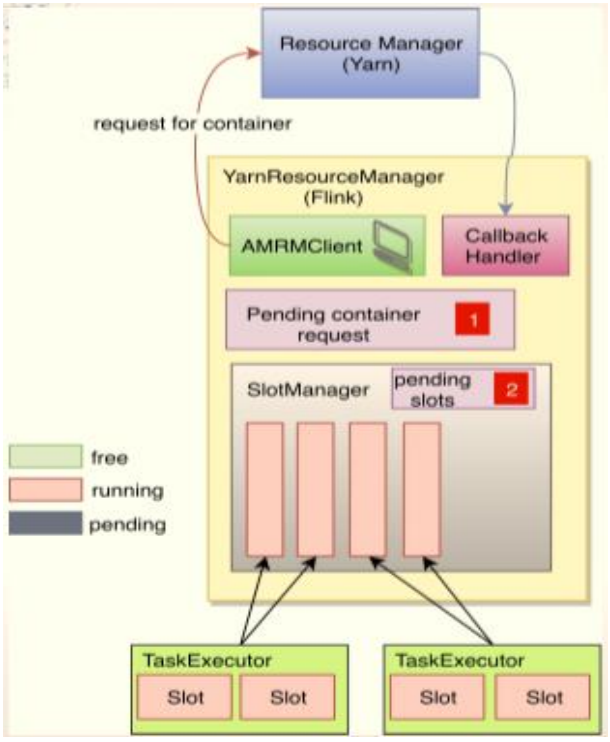

图 3 申请3个 Container

（2）正常流程

比如有这样一个任务，如图3，我们申请了3个 Container,每个 Container 有2个 Slot，然后并行

度是6,我们申请过程还没有完成,现在是有4个 Slot,然后这时候运行的任务就会向 SlotManager 请求,我们还需要2个 Slot 执行我的子任务。

这个 SlotManager 接到请求之后,它发现当前并没有可用的 Slot,就会把这个 Slot 请求积压在 Pending slots 里,并且向 Flink ResourceManager 去申请一个 Worker[10]。Flink ResourceManager 接到了申请之后,它就会通过 AMRMClient 向 Yarn 申请一个 Container,并且将积压的 Container 的请求置为1。

当 Yarn 准备好 Container 之后,它就会通过 Callback Handler 通知 Flink ResourceManager。Flink ResourceManager 接到通知,它就会将 Pending Container Request 减1,现在就置为0,并且它会在 Container 中启动一个 TaskExcutor。但现在 Pending slots 这一块,其实当时还是2,当 TaskExcutor 启动完成之后,它就会向 Slotmanager 注册,注册2个空的 Slot,这2个空的 Slot 可以满足刚刚积压的所有申请。

这样整个流程,任务就正常地运行起来了。

接下来,出问题的是什么情况?如图4所示,就是正常运行时,6个 Slot,然后3个 Container。

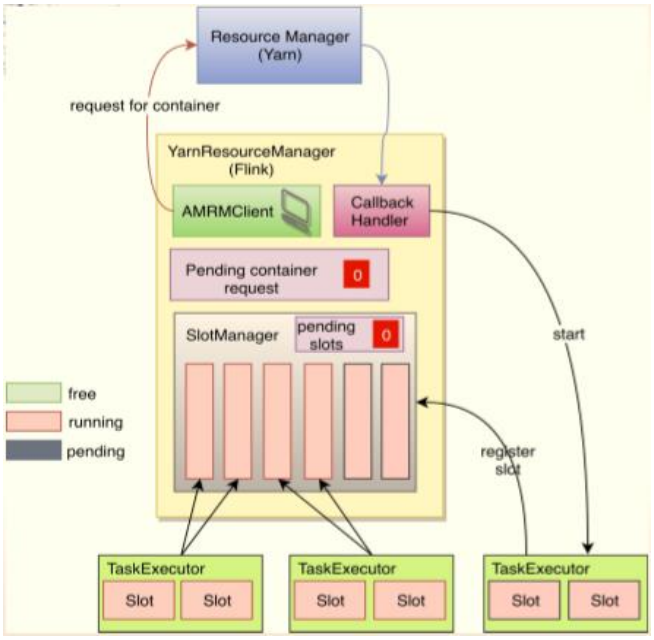

图 4 2个空的 Slot正常运行

(3)第一个坑:TaskExecutor 挂掉

这个时候,比如瞬间 TaskExecutor 都挂掉,我们当时还在测试调研的阶段,我使用的是 Yarn 的 default 队列,当时它的资源一下子就被抢占了。

这个时候 Flink ResourceManager 的 Callback Handler 就3次会接到这个回调,发现这个 Container 异常退出的。它就会立即去新申请3个新的 Container,并且把积压的 Pending container requerst[11] 的计数置为3。

但是在这个 Container 还没有拿到,还没有启动 TaskExecutor [12]的时候,事实上它的 SlotManager 里 Slot 是空的。这个时候任务重启,他又重新申请了6个 Slot,SlotManager 又重新向 Flink ResourceManager 申请3个 Container,这个时候 Pending Container request 就变成了6。

最后的结果就是出现了6个 Container,如图5,并且比配置的多了一倍,其中6个 Container 提供了12个 Slot,但是实际运行的只有6个,另外6个是被闲置的,它也不会去回收。

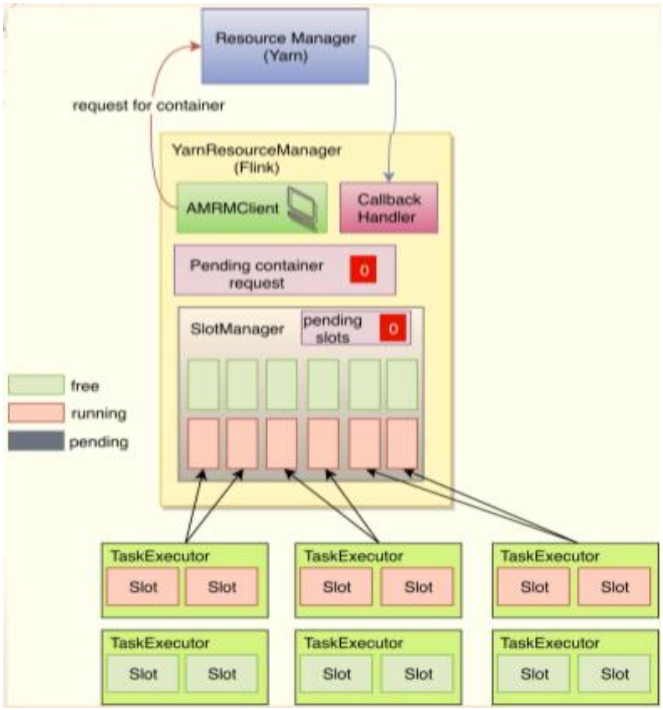

图 5 6个 Container 提供了12个 Slot

(4)异常流程的解决方案

第一个比较简单的解决方法,在接收到容器退出通知的时候,我们不去启动这个申请信,也不会去申请一个新容器,我们等它任务重启之后,它向 SlotManager 申请 Slot 再去申请新容器[13]。但这又会带来另一个问题,当我们一个 TaskExecutor 在启动的过程中,它报错退出了,事实上,它从来都没有注册 Slot,它通知了 Callback Handler,Callback Handler 就是没有任何作为。此时,就有2个 Slot Request 一直积压在那,永远都得不到满足。

第二个解决方案，在 FlinkResourceManager 申请 Container 之前，先去检查一下 Pending slots，以上述例子为例，比如Pending slots 是6，Pending Container Request 是3，正好就可以满足6个 Slot 的请求，如果超过3，我就可以认为它已经超出了当前的需求。

这个问题在1.5.1和1.6.0中得到了修复，如果还有 Patch 需要人为处理，是和 FlinkResouceManager 相关的Patch，这个问题在1.5.5和1.6.2中得到了完全的修复。

（5）第二个坑：延时监控

我们遇到的第二个坑，是和延时监控[14]相关的。如图6，这是一个很简单的任务，2个 Source，2个 operator，然后并行度是2。但在这个任务中，延时监控的数据它不是这么计算的，它会测量从每一个 Source 子任务到每一个算子的子任务之间两个算子之间的延时。

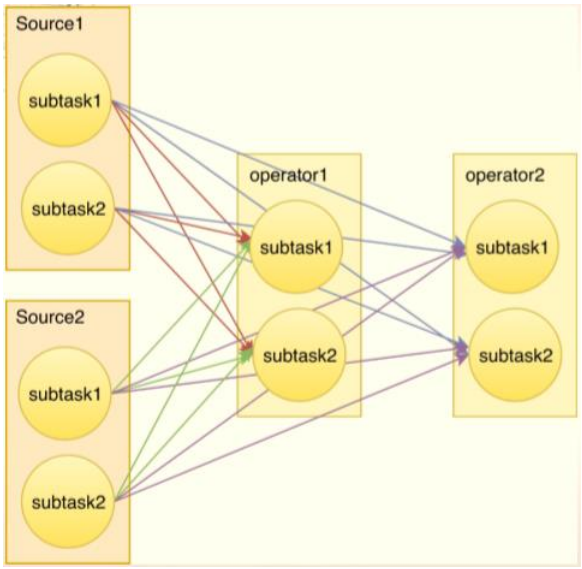

图 6 延时数据增长随着并行度呈平方的速度增长

从图6我们可以得到一个公式，N=n(subtasks per source)n(sources)n(subtasks per operator)*n(operator)，假设 source 和算子的并行度都为 p ，可以简化为：N=p2n(sources)n(operator)。我们简化，如果算子和 Source 的并行度都是2，我们延时数据的增长随着并行度呈平方的速度在增长。

以上述任务为例，把并行度改到10，那你就收到了400条延时相关的消息，这里还包括了它的平均延时、它的99%延时、它的中位数延时、它的最大延时，就会收到非常多的消息。如果仅仅是消息多，影响不大。但是在 Flink 开发的 Mailing list 里，有一个用户就反馈了，他开启了 latency tracking[15] 之后，JobMaster 很快就挂掉，他当时投了两个数据，第一个24000，他收到了24000多条跟延时相关的消息。1.6G是它存储这些监控数据的，存储数据的哈希表里就占用了1.6G的内存。

（6）延时监控的解决方案

我们安装Application Master配的内存一般是配2G，仅延时相关的监控就占到1.6G，肯定是不能容忍的。关于这个问题的解决方案，如果它影响到 JobMaster 正常工作的话，推荐关掉。但关掉，你一方面没有办法测量到自己任务的延时，另一方面你也没有办法针对延时去做一些监控。

所以，另一个解决方案相对来说要复杂一些，在 Flink-10243 里，它引入了监控配置粒度，从源头上减少监控的数量，它不再测从每一个 Source 子任务到每一个算子的子任务之间延时，它只测每个算子所有消息进来的延时是多少，上例10个并行度，2个 Source,2个 Operator，它延时数据从400减到20，从根源上解决了这个问题，因为它已经和并行度呈线性增长。

还有一个解决方法是改进它内部 MetricQueryService，这里有3个 Patch 是为监控任务启动一个低优先级的 ActorSystem。ActorSystem 是 Actor 里的概念，可以简单理解为：提供低优先级线程的线程池，防止了 Metric影响它主要组件的运行。

Flink-10252 仍然处于 reviewing 和修改当中，就是去控制每条监控消息的大小。这里相信在处理完 Flink-10243 和 Flink-10246 的前三个 patch，这个问题应该也会得到解决。

4. 2 Flink 结合 Spring

在大数据平台，很多服务仅开通了 dubbo 接口，要使用 dobbu 服务只能通过 spring 去启动 dubbo 的Client。一些工程师是从 java 工程师转过来的，他们非常希望在 Flink 中也能用 spring 去解决应用开发。

先来研究系统应用中的一些错误。

（1）错误一

第一个错误，用户在代码里启动了一个 SpringContext，如图7，最后 SpringContext 启动在客户端 Client ，实际运行的时候，任务是在 Worker 的 Task 端，然后在 Task 里去获取 Bean 环境，这是不可能的，因为它们本身都已经不在一个 JVM 中。

出现了这个问题之后，我们讨论如何解决这个问题？在实际执行 Function 的 open 方法中，我们通过配置文件新建一个 Spring Context。这个问题又带来其它新问题。

（2）错误二

如图8，一个 TaskManager 有3个 Slot，每个 TaskSlot 都可以启动一个 Spring Context,若仅是资源浪费还好,在实验时一个 JVM 启动了两个 Sping Context，系统会出问题。最简单的解决方法，一个 TaskManager 只设一个 slot，但是还是不行，因为 Flink 的优化机制 OperatorChain会把窄依赖的算子嵌到一个 operator 、一个 TaskSlot 中去执行，把这个关掉，还是不行。它还有另外一个机制，就是 CoLocationGroup，它会把几个 sub_task 放到一个 TaskSlot 中去执行，这样可以去优化它的并行度，这肯定不是很好的解决方法。

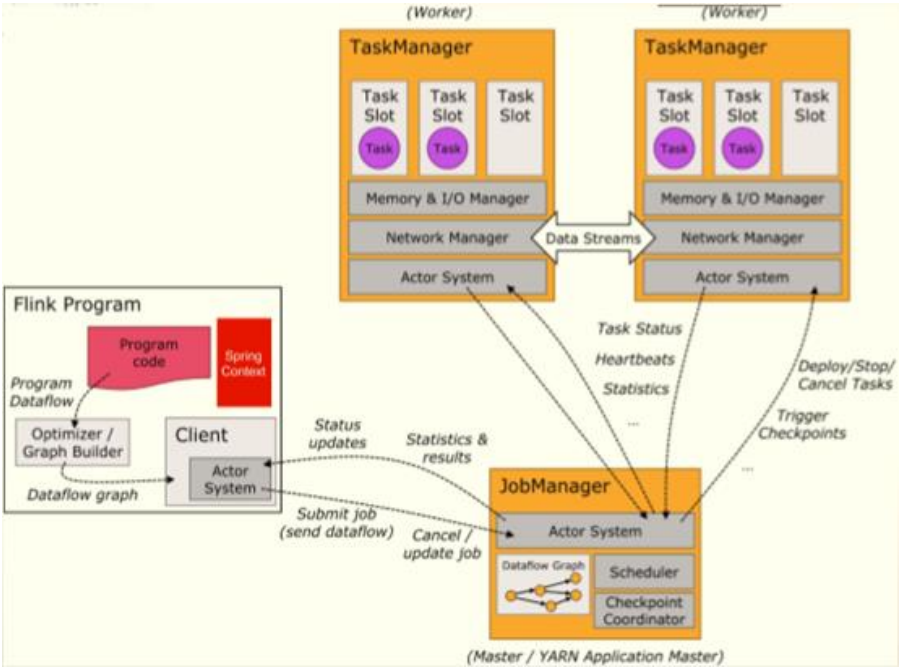

图 7 启动了一个 SpringContext

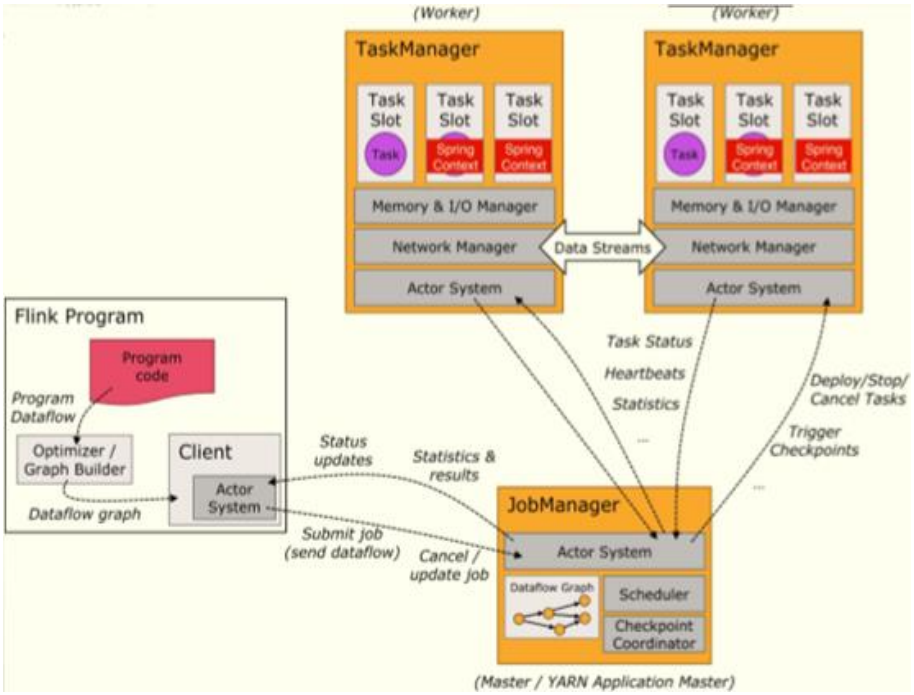

图 8 通过配置文件新建一个 Spring Context

（3）解决方案

最后的解决方案很简单，封装一个单例的SpringContext,在算子的 open 方法中获取单例来获取相应的 Bean,调用 dubbo 服务，调用过程可能是十毫秒到百毫秒，至少吞吐量就限制在一百毫秒。

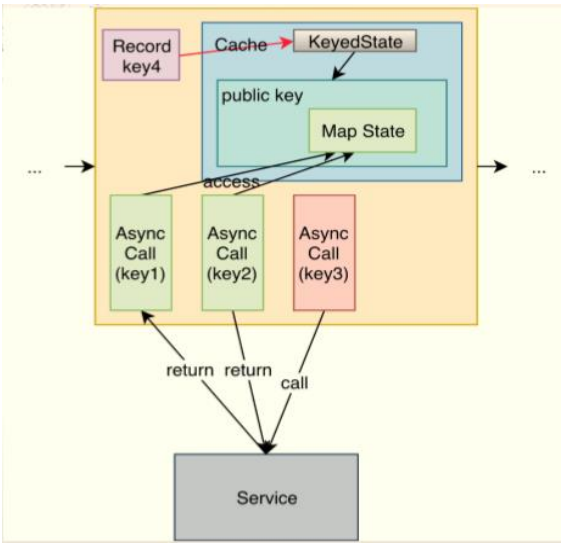

图 9 在 public Key 用一个 Map State 存储 Key 和它的缓存内容

4.3 Flink 异步不支持 KeyedState

对于实时任务，如何解决资源浪费、性能不好的问题？解决办法是做缓存。经过多次实验，发现返回的是同一个值，我就考虑去做缓存，通过异步方式提升吞吐量。我希望用 Flink 的 KeyedState 去做缓存，但在实际使用的过程中，RichAsyncfunction不支持使用 Flink 的 KeyedState。

我仔细分析代码,这个原因其实也是很有道理的，当一个消息进入一个算子的时候,它会将 KeyedState 指向了跟这个消息的 Key 绑定的内存空间。比如 Record Key4 进来的时候，KeyedState 已经指向了 Key4 绑定的存储空间，这时和 Key1 相关的异步请求完成了，这时访问KeyedState实际是访问的 Key4 绑定的地址空间,到下一个 Key1 的消息到来的时候，它这个 KeyedState 又会指到 Key1 上面，然而我之前做的缓存是在 Key4 里，我就没有办法访问到Key4 资源了。

（1）解决方法一

第一个简单的方法就是将缓存放到堆里，做一个类似于 LRUCache，但是这就没有办法利用 Flink 对这种中间状态的管理能力。

（1）解决方法二

然后我们再去研究，发现可以使用一种比较奇怪的方式，如图9，避开了KeyedState机制。我新建一个算子，然后覆盖了KeyedState 指向 Key 的方法，让它永远都只存在一个 public Key ，每一个消息进来，每一个回调，我都是去访问同一块存储空间，并且我可以用一个 Map State 去存储这些 Key 和它的缓存的内容。对用户来说，实际就是继承 AbstractRichFunction，然后实现 AsyncFunction ，就可以在使用异步的时候也可以去获取它的 keyedState。

4.4 Flink CEP 的使用

Flink 的 CEP能够实现跨事件的处理。比如当一个“下单事件”发生后的5分钟内，它出现了一个“付款事件”，然后“付款事件”发生后的5分钟内收款方又立刻去“提现”了，我们觉得这可能是一种比较恶意的套现或其它的一些形式，然后我们可能需要一个报警。在 Flink CEP 中，我们可以配置一套规则，begin...followedBy 然后 within。begin,事件命名；where，把这个消息流中和这个事件相关的事件 filter 出来;followedBy,接下来会发生的那个事件；Within，两个事件发生的时间差。将这个规则应用在一个事件流上面，就得到我们想要的效果。

5 结束语

关于Flink的未来，从大数据平台建构角度来讲，之前我们是通过 sdk 加配置化的方法来实现实时 SQL 化的，接下来我们希望在基于 Flink 的 SQLClient 再做一些二次开发，做一些改造。

然后对于 Flink 的批处理和 ML 模块，现在依旧处于一个调研阶段，会将它跟 Spark 做一些比较。

对 Flink 的新特性,我们关注的是调度和资源管理。现在 Flink 的调度模式，它的调度器和任务执行图是耦合在一块的，下一步首先把调度器从 ExecutionGraph 中解耦,变成一个独立的可插拔式的模块，它现在的调度模式很简单。我们有了可插拔式的模块之后，可以尝试更多更好的调度模式，然后基于一个新的调度器,可以去改造这种资源管理的模式，可以去动态调节资源量，甚至去实现 AutoScaling。这是我们接下来比较期待的一个特性。

参 考 文 献

[1] Apache Flink official website [EB/OL]. https://flink.apache.org/.

[2] 付雯 聂强 Spark 大数据实时分析实战[M]，北京理工大学出版社，2020.12

[3] 刘河 杨艺. 智能系统[M]，电子工业出版社,2020.4

[4] Yang, M.; Ma, J.; Wang, P.; Huang, Z.; Li, Y.; Liu, H.; Hameed, Z. Hierarchical Boosting Dual-Stage Feature Reduction Ensemble Model for

Parkinson's Disease Speech Data. Diagnostics 2021, 11, 2312. https://doi.org/10.3390/diagnostics11122312

[5] 聂强 付雯 Hadoop 离线分析实战[M]，北京理工大学出版社，2021.08

[6] (德)雷扎尔·卡里姆, (美)斯里达尔·阿拉著 Scala 和 Spark 大数据分析[M],清华大学出版社，2020.06

[7] 曹洁著 Spark 大数据分析技术[M]，北京航空航天大学出版社，2021

[8] (美)朱尔斯·J. 伯曼(Jules J. Berman)著 大数据原理与实践[M]，机械工业出版社，2020

[9] (挪)拉金德拉·阿卡拉卡(Rajendra Akerkar), (印)普里蒂·斯里尼瓦斯·萨加(Priti Srinivas Sajja)著 大数据分析与算法[M]，机械工业出版社，2018.10

[10] Miao Y, Tang H, Pan G. A Supervised Multi-Spike Learning Algorithm for Spiking Neural Networks[C]//2018 International Joint Conference on Neural Networks (IJCNN). IEEE, 2018: 1-7.

[11] Zhang T, Zeng Y, Zhao D, et al. A plasticity-centric approach to train the non-differential Spiking neural networks[C]//Thirty-Second AAAI Conference on Artificial Intelligence. 2018.

[12] Zhang T, Zeng Y, Zhao D, et al. Brain-inspired Balanced Tuning for Spiking Neural Networks[C]//IJCAI. 2018: 1653-1659.

[13] Zhang M, Qu H, Li J, et al. Feedforward computational model for pattern recognition with Spiking neurons[J]. International Journal of Robotics and Automation, 2018, 33: 206-5044.

[14] Zhang M, Qu H, Belatreche A, et al. EMPD: An efficient membrane potential driven supervised learning algorithm for Spiking neurons[J]. IEEE Transactions on Cognitive and Developmental Systems, 2018, 10(2): 151-162.

[15] 陈志毅,隋杰.基于 DeepFM 和卷积神经网络的集成式多模态谣言检测方法[J].计算机科学,2022, 49(01): 101-107.

.....

国际学术会议 IEEE ICCSE 2023 简讯

第十八届国际计算机科学与教育学术会议（IEEE ICCSE 2023）将于 2023 年 12 月 1-3 日在马来西亚吉隆坡召开。该会议由全国高等学校计算机教育研究会主办，厦门大学马来西亚分校承办。会议论文集将由斯普林格出版社出版，并由其提交到 EI 等检索数据库。历年会议论文集、会议情况及最新征文通知见会议网站：www.ieee-iccse.org。 欢迎投稿！

咨询与联系：ieee.iccse@gmail.com。

.....

《计算机技术与教育学报》征文通知

《计算机技术与教育学报》是全国高等学校计算机教育研究会会刊，国际刊号为：ISSN: 2325-0208。 期刊网址为： http://www.csteic.org 。现面向全国高校的教师，学生；企业从事计算机技术应用及教育的工作者征文。

联系邮箱：csteic3@163.com, csteic@gmail.com。

.....

数字化能力水平认证（简称"DCLC"）

数字化能力水平认证（简称"DCLC"）是由全国高等学校计算机教育研究会主办，面向社会，用于考查个人的数字素养与技能、数据科学知识以及组织的数字化转型能力全国性的认证体系。欢迎个人或团体参加DCLC认证，诚邀符合条件的单位加入合作伙伴。更多信息详见官方网站：http://www.dclc.org.cn。

欢迎咨询！联系电话:15960240768，联系邮箱：support@dclc.org.cn。