第3章

Flink 的设计与运行原理

近年来，流处理这种应用场景在企业中变得越来越频繁，由此带动了企业数据架构开始由传统数据处理架构、大数据 Lambda 架构向流处理架构演变。Flink 就是一种具有代表性的开源流处理架构，具有十分强大的功能，它实现了 Google Dataflow 流计算模型，是一种兼具高吞吐、低延迟和高性能的实时流计算框架，并且同时支持批处理和流处理。Flink 的主要特性包括批流一体化、精密的状态管理、事件时间支持以及“精确一次”的状态一致性保障等。Flink 不仅可以运行在包括 YARN、Mesos、Kubernetes 等在内的多种资源管理框架上，还支持在裸机集群上独立部署。Flink 目前已经在全球范围内得到了广泛的应用，大量企业已经开始大规模使用 Flink 作为企业的分布式大数据处理引擎。

本章首先给出 Flink 简介，并探讨选择 Flink 的原因以及 Flink 的典型应用场景；其次介绍 Flink 的统一数据处理、技术栈、工作原理、编程模型和应用程序结构；最后介绍 Flink 中的数据一致性。

3.1 Flink 简介

Flink 是 Apache 软件基金会的一个顶级项目，是为分布式、高性能、随时可用以及准确的流处理应用程序打造的开源流处理框架，并且可以同时支持实时计算和批量计算。Flink 起源于 Stratosphere 项目，该项目是 2010—2014 年由柏林工业大学、柏林洪堡大学和哈索·普拉特纳研究所联合开展的，开始是做批处理，后来转向了流计算。2014 年 4 月，Stratosphere 代码被贡献给 Apache 软件基金会，成为 Apache 软件基金会孵化器项目，并开始在开源大数据行业内崭露头角。之后，团队的大部分创始成员离开大学，共同创办了一家名为 Data Artisans 的公司，该公司于 2019 年 1 月被我国的阿里巴巴公司收购。在项目孵化期间，为了避免与另外一个项目发生重名，Stratosphere 被重新命名为 Flink。在德语中，Flink 是“快速和灵巧”的意思，使用这个词作为项目名称，可以彰显流计算框架的速度快和灵活性强的特点。项目使用一只棕红色的松鼠图案作为标志（见图 3-1），因为松

图 3-1 Flink 的标志

鼠具有灵活、快速的特点。

2014 年 12 月，Flink 成为 Apache 软件基金会顶级项目。目前，Flink 是 Apache 软件基金会的 5 个最大的大数据项目之一，在全球范围内拥有 350 多位开发人员，并在越来越多的企业中得到了应用。在国外，优步、网飞、微软和亚马逊等公司已经开始使用 Flink。在国内，包括阿里巴巴、美团、滴滴等在内的知名互联网企业，都已经开始大规模使用 Flink 作为企业的分布式大数据处理引擎。在阿里巴巴，基于 Flink 搭建的平台于 2016 年正式上线，并从阿里巴巴的“搜索和推荐”这两大场景开始实现。目前，阿里巴巴所有的业务，包括阿里巴巴所有子公司都采用了基于 Flink 搭建的实时计算平台，服务器规模已经达到数万台，这种规模等级在全球范围内也是屈指可数的。阿里巴巴的 Flink 平台内部积累起来的状态数据，已经达到 PB 级别规模，每天在平台上处理的数据量已经超过万亿条，在峰值期间可以承担超过 4.72 亿次每秒的访问量，最典型的应用场景是阿里巴巴“双 11”大屏。

Flink 具有十分强大的功能，可以支持不同类型的应用程序。Flink 的主要特性包括批流一体化、精密的状态管理、事件时间支持以及“精确一次”的状态一致性保障等。Flink 不仅可以运行在包括 YARN、Mesos、Kubernetes 等在内的多种资源管理框架上，还支持在裸机集群上独立部署。当采用 YARN 作为资源调度管理器时，Flink 计算平台可以运行在开源的 Hadoop 集群之上，并使用 HDFS 作为数据存储，因此，Flink 可以和开源大数据软件 Hadoop 实现无缝对接。在启用高可用选项的情况下，Flink 不存在单点失效问题。事实证明，Flink 已经可以扩展到数千核心，其状态可以达到 TB 级别规模，且仍能保持高吞吐、低延迟的特性。世界各地有很多要求严苛的流处理应用都运行在 Flink 上。

3.2　选择 Flink 的原因

数据架构设计领域正在发生一场变革，开始由传统数据处理架构、大数据 Lambda 架构向流处理架构演变，在这种全新的架构中，基于流的数据处理流程被视为整个架构设计的核心。这种转变把 Flink 推向了分布式计算框架这个舞台的中心，使它可以在现代数据处理中扮演重要的角色。

3.2.1　传统数据处理架构

传统数据处理架构的一个显著特点就是采用一个中心化的数据库系统，用于存储事务性数据，如图 3-2 所示。例如，在一个企业内部，会存在 ERP 系统、订单系统、CRM 系统等，这些系统的数据一般都被存储在关系数据库中。这些数据反映了当前的业务状态，如系统当前的登录用户数、网站当前的活跃用户数、每个用户的账户余额等。当应用程序需要较新的数据时，都会访问这个中心数据库。

在应用的初期，这种传统数据处理架构的效率很高，在各大企业应用中成功服务了几十年。但是，随着企业业务量的不断增大，数据库的负载开始不断增加，最终变得不堪重负，而一旦数据库系统发生问题，整个业务系统就会受到严重影响。此外，采用传统数据

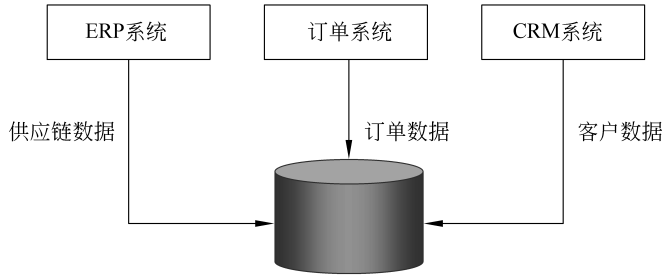


图 3-2 传统数据处理架构

处理架构的系统,一般都拥有非常复杂的异常问题处理方法,当出现异常问题时,很难保证系统还能够很好地运行。

3.2.2 大数据 Lambda 架构

随着信息技术的普及和企业信息化建设步伐的加快,企业逐步认识到建立企业范围内的统一数据存储的重要性,越来越多的企业建立了企业数据仓库。企业数据仓库有效集成了来自不同部门、不同地理位置、具有不同格式的数据,为企业管理决策者提供了企业范围内的单一数据视图,从而为综合分析和科学决策奠定了坚实的基础。

起初数据仓库主要借助于 Oracle、SQL Server、MySQL 等关系数据库进行数据的存储,但是,随着企业数据量的不断增长,关系数据库已经无法满足海量数据的存储需求。因此,越来越多的企业开始构建基于 Hadoop 的数据仓库,并借助 MapReduce、Spark 等分布式计算框架对数据仓库中的数据进行处理分析。但是,数据仓库中的数据一般都是周期性进行加载,如每天一次、每周一次或者每月一次,这样就无法满足一些对实时性要求较高的应用的需求。为此,业界提出了一套大数据 Lambda 架构方案来处理不同类型的数据,从而满足企业不同应用的需求。如图 3-3 所示,大数据 Lambda 架构主要包含两层,即批处理层和实时处理层。在批处理层中,采用 MapReduce、Spark 等技术进行批量数据处理;而在实时处理层中,则采用 Storm、Spark Streaming 等技术进行数据的实时处理。

分开处理连续的实时数据和有限批次的批量数据,可以使系统构建工作变得更加简单,这种架构在一定程度上解决了不同计算类型的问题。但是,这种做法将管理两套系统的复杂性留给了系统用户,由于存在太多的框架,就会导致平台复杂度和运维成本过高,因为在一套资源管理平台中管理不同类型的计算框架是比较困难的事情。

3.2.3 流处理架构

作为一种新的选择,流处理架构解决了企业在大规模系统中遇到的诸多问题。以流为基础的架构设计,让数据记录持续地从数据源流向应用程序,并在各个应用程序间持续流动。不需要设置一个数据库来集中存储全局状态数据,取而代之的是共享且永不停止的流数据,它是唯一正确的数据源,记录了业务数据的历史。

为了高效地实现流处理架构,一般需要设置消息传输层和流处理层(见图 3-4)。消

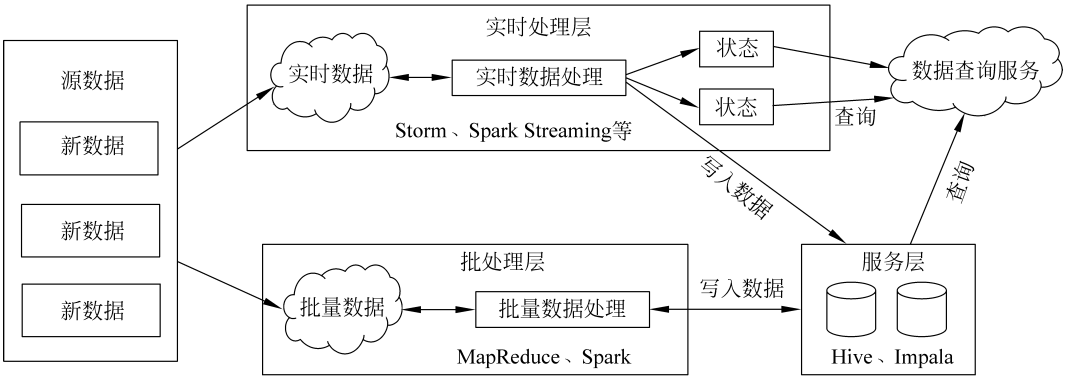


图 3-3　大数据 Lambda 架构

息传输层从各种数据源采集连续事件产生的数据，并传输给订阅了这些数据的应用程序；流处理层会持续地将数据在应用程序和系统间移动、聚合并处理事件，并在本地维持应用程序的状态。这里的“状态”就是计算过程中产生的中间计算结果，在每次计算中，新的数据进入流系统中，都是在中间状态结果的基础上进行计算，最终产生正确的计算结果。

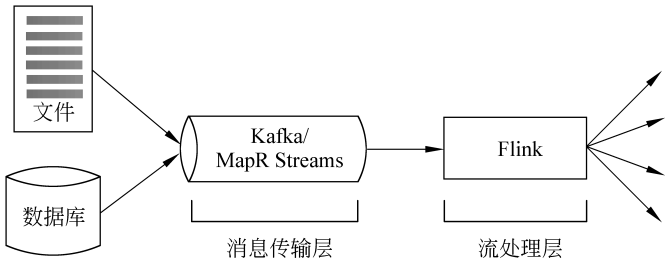


图 3-4　流处理架构

流处理架构的核心是使各种应用程序互连在一起的消息队列，消息队列连接应用程序，并作为新的共享数据源，这些消息队列取代了从前的大型集中式数据库。如图 3-5 所示，流处理器从消息队列中订阅数据并加以处理，处理后的数据可以流向另一个消息队列，这样，其他应用程序都可以共享流数据。有时处理后的数据会被存储到本地数据库中。

流处理架构正在逐步取代传统数据处理架构和大数据 Lambda 架构，成为大数据处理架构的一种新趋势。一方面，由于流处理架构中不存在一个大型集中式数据库，因此，避免了传统数据处理架构中存在的“数据库不堪重负”的问题；另一方面，在流处理架构中，批处理被看作流处理的一个子集，因此，就可以用面向流处理的框架进行批处理，这样就可以用一个流处理框架来统一处理流计算和批量计算，避免了大数据 Lambda 架构中存在的“多个框架难管理”的问题。

3.2.4　Flink 是理想的流计算框架

流处理架构需要具备低延迟、高吞吐和高性能的特性，而目前从市场上已有的产品来

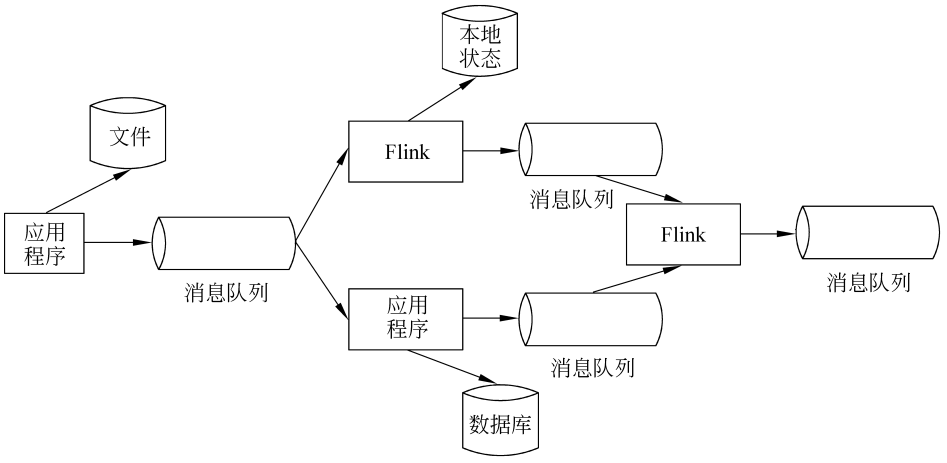


图 3-5 流处理架构中的消息队列

看(见表 3-1),只有 Flink 可以满足要求。Storm 虽然可以做到低延迟,但是无法实现高吞吐,也不能在故障发生时准确地处理计算状态。Spark 的流计算组件 Spark Streaming 通过采用微批处理方法实现了高吞吐和容错性,但是牺牲了低延迟和实时处理能力。Spark 的另一个流计算组件 Structured Streaming,包括微批处理和持续处理两种处理模型。采用微批处理模型时,最快响应时间需要 100ms,无法支持毫秒级别响应;采用持续处理模型时,可以支持毫秒级别响应,但是只能做到“至少一次”的一致性,无法做到“精确一次”的一致性。

表 3-1 不同流计算框架的对比

产 品	消息保证机制	容错机制	状态管理	延 时	吞 吐 量
Storm	至少一次	Acker 机制	无	低	低
Spark Streaming	精确一次	基于 RDD 的检查点	基于 DStream	中	高
Flink	精确一次	检查点	基于操作	低	高

Flink 实现了 Google Dataflow 流计算模型,是一种兼具高吞吐、低延迟和高性能的实时流计算框架,并且同时支持批处理和流处理。此外,Flink 支持高度容错的状态管理,防止状态在计算过程中因为系统异常而出现丢失。因此,Flink 就成为了能够满足流处理架构要求的理想的流计算框架。

3.2.5 Flink 的优势

Flink 实现了 Google Dataflow 流式计算模型,与其他的流计算框架相比,Flink 具有突出的特点,它不仅是一个高吞吐、低延迟的计算引擎,同时还具备其他的高级特性,如支持有状态的计算、状态管理、强一致性的语义,以及对消息乱序的处理等。

总体而言,Flink 具有以下优势。

1. 同时支持高吞吐、低延迟、高性能

对于分布式流计算框架，同时支持高吞吐、低延迟和高性能是非常重要的。但是，目前在开源社区中，能够同时满足这 3 方面要求的流计算框架只有 Flink。Storm 可以做到低延迟，但是无法实现高吞吐。Spark Streaming 可以实现高吞吐和容错性，但是不具备低延迟和实时处理能力。

2. 同时支持流处理和批处理

在典型的大数据的业务场景下，数据业务最通用的做法是，选用批处理的技术处理全量数据，采用流式计算处理实时增量数据。在绝大多数的业务场景下，用户的业务逻辑在批处理和流处理中往往是相同的。但是，在 Flink 被推广普及之前，用户用于批处理和流处理的两套计算引擎是不同的。因此，用户通常需要写两套代码。毫无疑问，这带来了一些额外的负担和成本。因此，我们就希望能够有一套统一的大数据引擎技术，用户只需要根据自己的业务逻辑开发一套代码。这样在各种不同的场景下，不管是全量数据还是增量数据，或者实时处理，一套方案即可全部支持，这就是 Flink 诞生的背景和初衷。Flink 不仅擅长流处理，同时也能够很好地支持批处理。对于 Flink，批量数据是流数据的一个子集，批处理被视作一种特殊的流处理，因此，可以通过一套引擎来处理流数据和批量数据。

3. 高度灵活的流式窗口

在流计算中，数据流是无限的，无法直接进行计算，因此，Flink 提出了窗口的概念，一个窗口是若干元素的集合，流计算以窗口为基本单元进行数据处理。窗口可以是时间驱动的(Time Window，如每 30s)，也可以是数据驱动的(Count Window，如每 100 个元素)。窗口可以分为翻滚窗口(Tumbling Window，无重叠)、滚动窗口(Sliding Window，有重叠)和会话窗口(Session Window)。

4. 支持有状态计算

流计算分为无状态和有状态两种情况。无状态计算观察每个独立的事件，并根据最后一个事件输出结果，Storm 就是无状态的计算框架，每条消息来了以后，彼此都是独立的，和前后都没有关系。有状态的计算则会基于多个事件输出结果。正确地实现有状态计算，比实现无状态计算难得多。Flink 就是可以支持有状态计算的新一代流处理系统。Flink 的有状态应用程序针对本地状态访问进行了优化。任务状态保留在内存中，但是如果状态大小超过可用内存，则保存在访问高效的磁盘数据结构中。因此，任务通过访问本地(通常是内存)状态来执行所有计算，从而产生非常低的处理延迟。

5. 具有良好的容错性

当分布式系统引入状态时，就会产生“一致性”问题。一致性实际上是“正确性级别”的另一种说法，也就是说，在成功处理故障并恢复之后得到的结果，与没有发生故障时得

到的结果相比,前者有多正确。Storm 只能实现“至少一次”的容错性。Spark Streaming 虽然可以支持“精确一次”的容错性,但是,无法做到毫秒级的实时处理。Flink 提供了容错机制,可以恢复数据流应用到一致状态。该机制确保在发生故障时,程序的状态最终将数据流中的每个记录只反映一次,即实现了“精确一次”的容错性。容错机制不断地创建分布式数据流的快照,对于小状态的流式程序,快照非常轻量,可以高频率创建而对性能影响很小。

6. 具有独立的内存管理

Java 本身提供了垃圾回收机制来实现内存管理,但是,在大数据面前,JVM 的内存结构和垃圾回收机制往往会成为掣肘。所以,目前包括 Flink 在内的越来越多的大数据项目开始自己管理 JVM 内存,为的就是获得像 C 语言一样的性能以及避免内存溢出的发生。Flink 通过序列化或反序列化方法,将所有的数据对象转换成二进制在内存中存储,这样做一方面降低了数据存储的空间,另一方面能够更加有效地利用内存空间,降低垃圾回收机制带来的性能下降或任务异常风险。

7. 支持迭代和增量迭代

对某些迭代而言,并不是单次迭代产生的下一次工作集中的每个元素都需要重新参与下一轮迭代,有时只需要重新计算部分数据同时选择性地更新解集,这种形式的迭代就是增量迭代。增量迭代能够使得一些算法执行得更高效,它可以让算法专注于工作集中的“热点”数据部分,这导致工作集中的绝大部分数据冷却得非常快,因此随后的迭代面对的数据规模将会大幅缩小。Flink 的设计思想主要来源于 Hadoop、MPP 数据库和流计算系统等,支持增量迭代计算,具有对迭代进行自动优化的功能。

3.3 Flink 典型应用场景

Flink 典型应用场景包括事件驱动型应用、数据分析应用和数据流水线应用。

3.3.1 事件驱动型应用

1. 事件驱动型应用简介

事件驱动型应用是一类具有状态的应用,它从一个或多个事件数据流中读取事件,并根据到来的事件做出反应,包括触发计算、状态更新或其他外部动作等。事件驱动型应用是在传统事务型应用设计基础上进化而来的。在传统事务型应用设计中,通常都具有独立的计算和数据存储层,应用会从一个远程的事务数据库中读写数据。而事件驱动型应用是建立在有状态流处理应用的基础之上的。在这种设计中,数据和计算不是相互独立的层,而是放在一起的,应用只需访问本地(内存或磁盘)即可获取数据。系统容错性是通过定期向远程持久化存储写入检查点来实现的。图 3-6 描述了传统事务型应用和事件驱动型应用架构的区别。

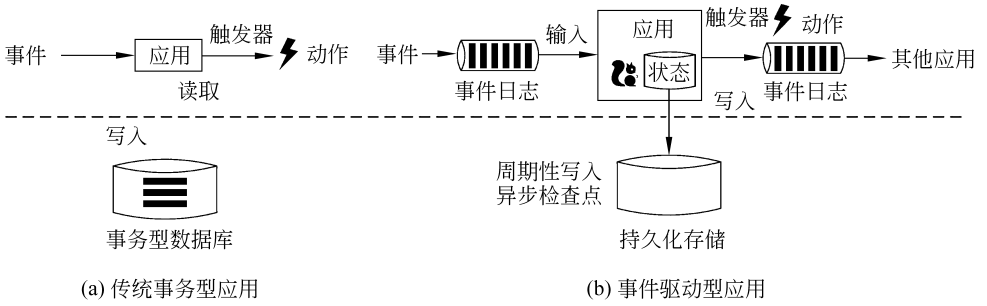


图 3-6　传统事务型应用和事件驱动型应用架构的区别

典型的事件驱动型应用包括反欺诈、异常检测、基于规则的报警、业务流程监控、Web 应用(社交网络)等。

2. 事件驱动型应用的优势

事件驱动型应用都是访问本地数据,而无须查询远程的数据库,因此无论是在吞吐量方面,还是在延迟方面,都可以获得更好的性能。向一个远程的持久化存储周期性地写入检查点,可以采用异步和增量的方式来实现,因此检查点对于常规的事件处理的影响是很小的。事件驱动型应用的优势不仅限于本地数据访问。在传统事务型应用的分层架构中,多个应用共享相同的数据库,是一个很常见的现象。因此,数据库的任何变化,如由于一个应用的更新或服务的升级而导致数据布局的变化,都需要谨慎协调。由于每个事件驱动型应用都只需要考虑自身的数据,对数据表示方式的改变或者应用的升级,都只需要很少的协调工作。

3. Flink 是如何支持事件驱动型应用的

一个流处理器如何能够很好地处理时间和状态,决定了事件驱动型应用的局限性。Flink 的许多优秀特性都是围绕这些方面进行设计的。它提供了丰富的状态操作原语,可以管理大量的数据(可以达到 TB 级别),并且可以确保“精确一次”的一致性。Flink 还支持事件时间、高度可定制的窗口逻辑和细粒度的时间控制,这些都可以帮助实现高级的商业逻辑。它还拥有一个复杂事件处理(Complex Event Processing,CEP)类库,可以用来检测数据流中的模式。

Flink 中针对事件驱动型应用的突出特性当数保存点(Save Point)。保存点是一个一致性的状态镜像,它可以作为许多相互兼容应用的一个初始化点。给定一个保存点以后,就可放心地对应用进行升级或扩容,还可以启动多个版本的应用来完成 A/B 测试。

3.3.2　数据分析应用

1. 数据分析应用简介

分析作业会从原始数据中提取信息,并得到富有洞见的观察。传统的数据分析通常先对事件进行记录,然后在这个有界的数据集上执行批量查询。为了把最新的数据融入

查询结果中,就必须把这些最新的数据添加到被分析的数据集中,然后重新运行查询。查询的结果会被写入一个存储系统中,或者形成报表。

一个高级的流处理引擎,可以支持实时的数据分析。这些流处理引擎并非读取有限的数据集,而是获取实时事件流,并连续产生和更新查询结果。这些结果或者被保存到一个外部数据库中,或者作为内部状态被维护。仪表盘应用可以从这个外部的数据库中读取最新的结果,或者直接查询应用的内部状态。

如图 3-7 所示,Apache Flink 同时支持批量及流式分析应用。

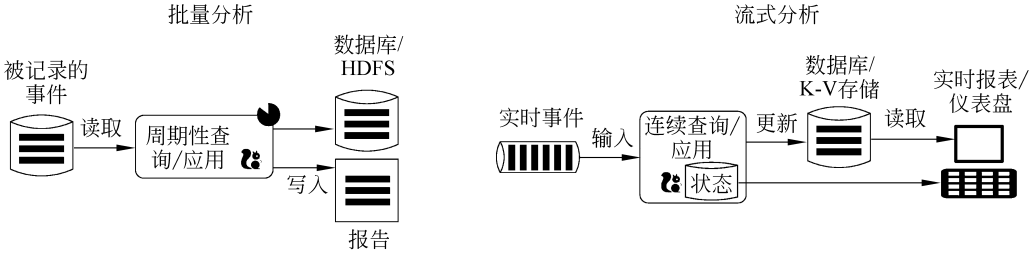


图 3-7 Flink 同时支持批量及流式分析应用

典型的数据分析应用包括电信网络质量监控、移动应用中的产品更新及实验评估分析、消费者技术中的实时数据即席分析、大规模图分析等。

2. 流式分析应用的优势

与批量分析相比,连续流式分析的优势:一方面,由于消除了周期性的导入和查询,因此从事件中获取洞察结果的延迟更低。此外,流式查询不需要处理输入数据中人为产生的边界。

另一方面,流式分析具有更加简单的应用架构。一个批量分析流水线会包含一些独立的组件来周期性地调度数据提取和查询执行。如此复杂的流水线,操作起来并非易事,因为一个组件的失败就会直接影响到流水线中的其他步骤。相反,运行在一个高级流处理器(如 Flink)之上的流式分析应用,会把从数据提取到连续结果计算的所有步骤都整合起来,因此,它就可以依赖底层引擎提供的故障恢复机制。

3. Flink 是如何支持数据分析应用的

Flink 可以同时支持批处理和流处理。Flink 提供了一个符合 ANSI 规范的 SQL 接口,它可以为批处理和流处理提供一致的语义。不管是运行在一个静态的数据集上,还是运行在一个实时的数据流上,SQL 查询都可以得到相同的结果。它还提供了丰富的用户自定义函数,使得用户可以在 SQL 查询中执行自定义代码。如果需要进一步定制处理逻辑,Flink 的 DataStream API 和 DataSet API 提供了更加底层的控制。此外,Flink 的 Gelly 库为基于批量数据集的大规模高性能图分析提供了算法和构建模块支持。

3.3.3　数据流水线应用

1. 数据流水线应用简介

Extract-transform-load(ETL)是一个在存储系统之间转换和移动数据的常见方法。通常,ETL 作业会被周期性地触发,从而把事务型数据库系统中的数据复制到一个分析型数据库或数据仓库中。

数据流水线可以实现和 ETL 类似的功能,它们可以转换、清洗数据,或者把数据从一个存储系统转移到另一个存储系统中。但是,它们是以一种连续的流模式来执行的,而不是周期性地触发。因此,当数据源中源源不断地生成数据时,数据流水线就可以把数据读取过来,并以较低的延迟转移到目的地。例如,一个数据流水线可以对一个文件系统目录进行监控,一旦发现有新的文件生成,就读取文件内容并写入事件日志中。再例如,将事件流物化到数据库或增量构建和优化查询索引。

图 3-8 描述了周期性 ETL 作业和持续数据流水线的差异。

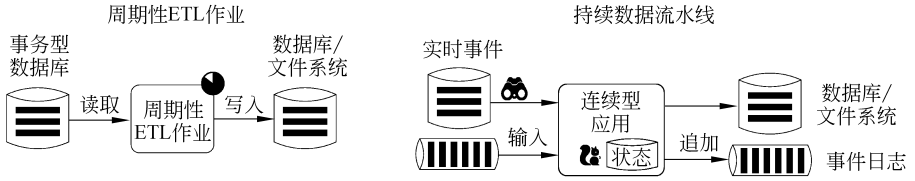


图 3-8　周期性 ETL 作业和持续数据流水线的差异

典型的数据流水线应用包括电子商务中的实时查询索引构建、电子商务中的持续 ETL 等。

2. 数据流水线应用的优势

相对于周期性 ETL 作业,持续数据流水线的优势是,减少了数据转移过程的延迟。此外,由于它能够持续消费和发送数据,因此用途更广,支持用例更多。

3. Flink 如何支持数据流水线应用

Flink 的 SQL 接口(或者 Table API)以及丰富的用户自定义函数,可以解决许多常见的数据转换问题。通过使用更具通用性的 DataStream API,还可以实现具有更加强大功能的数据流水线。Flink 提供了大量的连接器,可以连接到各种不同类型的数据存储系统,如 Kafka、Kinesis、Elasticsearch 和 JDBC 数据库系统。同时,Flink 提供了面向文件系统的连续型数据源,可用来监控目录变化,并提供了数据槽(Sink),支持以时间分区的方式写入文件。

3.4　Flink 的统一数据处理

根据数据的产生方式,可以把数据集分为两种类型:有界数据集和无界数据集(图 3-9)。

有界数据集具有时间边界,在处理过程中数据一定会在某个时间范围内起始和结束,

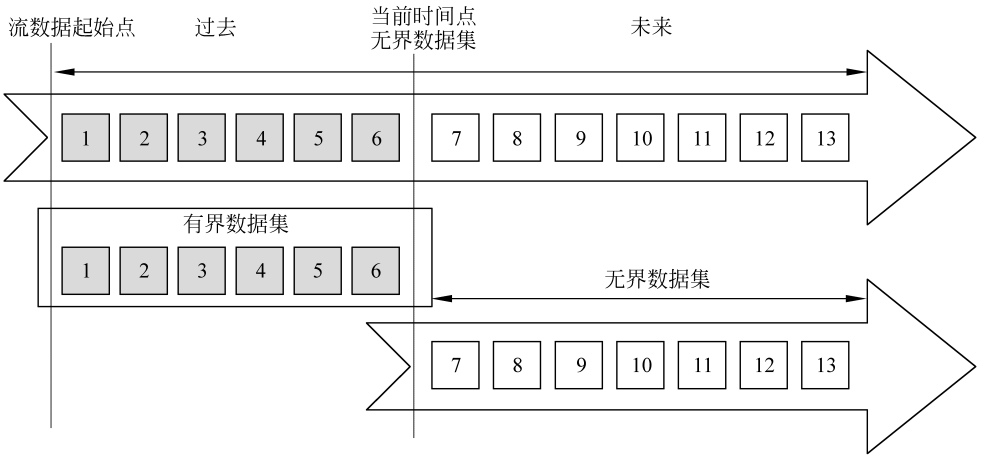


图 3-9　有界数据集和无界数据集

有可能是一小时,也有可能是一天内的交易数据。有界数据集的特点是,数据是静止不动的,不会存在数据的追加操作。对有界数据集的数据处理方式称为批处理,例如首先将数据从关系数据库或文件系统等系统中读取出来,其次在分布式系统内进行处理,最后再将处理结果写入存储系统中,整个过程称为批处理过程。目前业界比较流行的分布式处理框架 Hadoop 和 Spark 等,都是面向批处理的。

对于无界数据集,数据从开始生成就一直持续不断地产生新的数据,因此数据是没有边界的,例如服务器信令、网络传输流、传感器信号数据、实时日志信息等。和批量数据处理方式对应,对无界数据集的数据处理方式被称为流处理。流处理需要考虑处理过程中数据的顺序错乱,以及系统容错等方面的问题,因此流处理系统的设计与实现的复杂度要明显高于批处理系统。Storm、Spark Streaming、Flink 等分布式计算引擎是不同时期具有代表性的流处理系统。

为了更形象、直观地理解无界数据集与有界数据集,可以分别把二者类比成池塘和江河。对有界数据集进行计算时,就好比计算池塘中的鱼的数量,只需要把池塘中当前所有的鱼都计算一次就可以了。那么当前时刻,池塘中有多少条鱼就是最终结果。对于无界数据集的计算,类似于计算江河中的鱼,在奔流到海的过程中,每时每刻都会有鱼流过而进入大海,那么计算鱼的数量就是持续追加的。

有界数据集与无界数据集是一个相对模糊的概念。对于有界数据集,如果数据一条一条地经过处理引擎,那么也可以认为是无界的。反过来,对于无界数据集,如果每间隔一分钟、一小时、一天进行一次计算,那么也可以认为这一段时间内的数据又相对是有界的。所以,有界数据集与无界数据集的概念有时候是可以存在互换的,因此,学界和业界也就开始追寻批、流统一的框架,Spark 和 Flink 都属于能够同时支持批处理和流处理的分布式计算框架。

对于 Spark,它会使用一系列连续的微小批处理来模拟流处理,即它会在特定的时间间隔内发起一次计算,而不是每条数据都触发计算,这就相当于把无界数据集切分为多个

小量的有界数据集。对于 Flink，它把有界数据集看成无界数据集的一个子集，因此，将批处理与流处理混合到同一套引擎当中，用户使用 Flink 引擎能够同时实现批处理与流处理任务。

3.5　Flink 技术栈

Flink 的发展越来越成熟，已经拥有了自己丰富的核心组件栈。Flink 核心组件栈分为 3 层（见图 3-10）：物理部署层、Runtime 核心层和 API&Libraries 层。

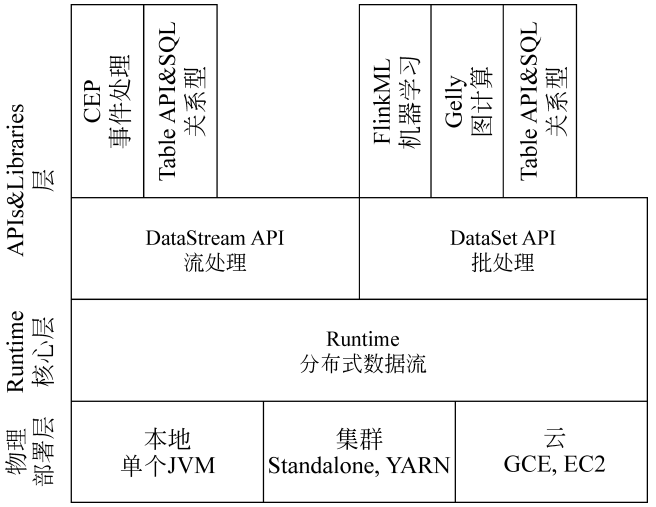


图 3-10　Flink 核心组件栈

（1）物理部署层。Flink 的底层是物理部署层。Flink 可以采用 Local 模式运行，启动本地单个 JVM，也可以采用 Standalone 集群模式运行，还可以采用 YARN 集群模式运行，或者也可以运行在 GCE（谷歌云服务）和 EC2（亚马逊云服务）上。

（2）Runtime 核心层。该层主要负责对上层不同接口提供基础服务，也是 Flink 分布式计算框架的核心实现层。该层提供了两套核心的 API：DataStream API（流处理）和 DataSet API（批处理）。

（3）API&Libraries 层。作为分布式数据库处理框架，Flink 同时提供了支撑批计算和流计算的接口，同时，在此基础上抽象出不同的应用类型的组件库，如 CEP（基于流处理的事件处理库）、Table API&SQL（既可以基于流处理，也可以基于批处理的关系数据库）、FlinkML（基于批处理的机器学习库）、Gelly（基于批处理的图计算库）等。

这里需要说明的是，Flink 虽然也构建了一个大数据生态系统，功能涵盖流处理、批处理、SQL、图计算和机器学习等，但是，它的强项仍然是流计算，Flink 的图计算组件 Gelly 和机器学习组件 FlinkML 并不是十分成熟。因此，本书不介绍 Gelly 和 FlinkML，将详细讲解 DataStream API、DataSet API、Table API&SQL 和 CEP 等组件。

3.6 Flink 工作原理

如图 3-11 所示,Flink 系统主要由两个组件组成,分别为 JobManager 和 TaskManager,Flink 架构也遵循主从(Master-Slave)架构设计原则,JobManager 为 Master 节点,TaskManager 为 Slave 节点。Flink 系统各组件的功能如下。

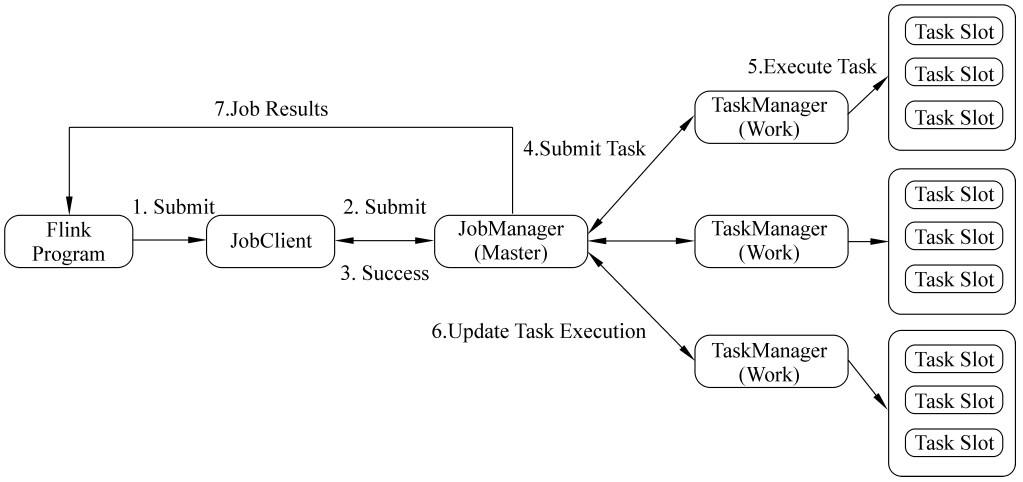


图 3-11 Flink 的体系架构及其工作原理

(1) JobClient:负责接收程序,解析和优化程序的执行计划,然后提交执行计划到 JobManager。这里执行的程序优化是将相邻的算子融合,形成"算子链",以减少任务的数量,提高 TaskManager 的资源利用率。

(2) JobManager:负责整个 Flink 集群任务的调度以及资源的管理,它从客户端中获取提交的应用,然后根据集群中 TaskManager 上 Task Slot 的使用情况,为提交的应用分配相应的 Task Slot 资源,并命令 TaskManager 启动从客户端中获取的应用。为了保证高可用,一般会有多个 JobManager 进程同时存在,它们之间也是采用主从模式,一个进程被选举为 Leader,其他进程为 Follower,在作业运行期间,只有 Leader 在工作,Follower 是闲置的,一旦 Leader 死亡,就会引发一次选举,产生新的 Leader 继续处理作业。JobManager 除了调度任务,另一个主要工作就是容错,主要依靠检查点机制进行容错。

(3) TaskManager:相当于整个集群的 Slave 节点,负责具体的任务执行和对应任务在每个节点上的资源申请与管理。客户端通过将编写好的 Flink 应用编译打包,提交到 JobManager,然后 JobManager 会根据已经注册在 JobManager 中 TaskManager 的资源情况,将任务分配给有资源的 TaskManager 节点,然后启动并运行任务。TaskManager 从 JobManager 接收需要部署的任务,然后使用 Slot 资源启动 Task,建立数据接入的网络连接,接收数据并开始数据处理。同时 TaskManager 之间的数据交互都是通过数据流

的方式进行的。

(4) Slot：Slot 是 TaskManager 资源粒度的划分，每个 TaskManager 像一个容器一样，包含一个或多个 Slot，每个 Slot 都有自己独立的内存，所有 Slot 平均分配 TaskManager 的内存。需要注意的是，Slot 仅划分内存，不涉及 CPU 的划分，即 CPU 是共享使用的。每个 Slot 可以运行多个任务，而且一个任务会以单独的线程来运行。

采用 Slot 设计主要有 3 个好处：①可以起到隔离内存的作用，防止多个不同作业的任务竞争内存；②Slot 的个数就代表了一个 Flink 程序的最高并行度，简化了性能调优的过程；③允许多个任务共享 Slot，提升了资源利用率。

(5) Task：是在算子的子任务进行链化之后形成的，一个作业中有多少个 Task 和算子的并行度和链化的策略有关。

Flink 系统的工作原理：在执行 Flink 程序时，Flink 程序需要首先提交给 JobClient，然后，JobClient 将作业提交给 JobManager。JobManager 负责协调资源分配和作业执行，它首先要做的是分配所需的资源。资源分配完成后，任务将提交给相应的 TaskManager。在接收任务时，TaskManager 启动一个线程以开始执行。执行到位时，TaskManager 会继续向 JobManager 报告状态更改，可以有各种状态，例如开始执行、正在进行或已完成。作业执行完成后，结果将发送回客户端(JobClient)。

3.7　Flink 编程模型

Flink 提供了不同级别的抽象(见图 3-12)，以开发流处理或批处理作业。

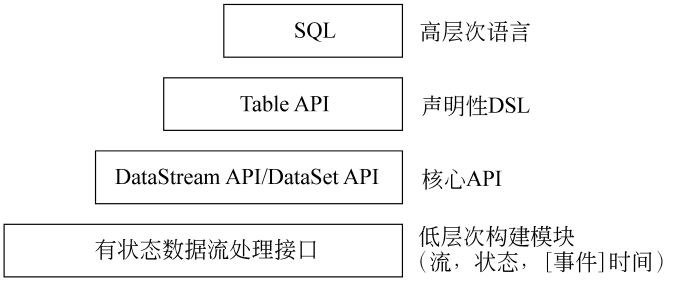


图 3-12　Flink 编程模型

(1) 在 Flink 编程模型中，最低级的抽象接口是有状态数据流处理接口。这个接口是通过过程函数(Process Function)被集成到 DataStream API 中的。该接口允许用户自由地处理来自一个或多个流中的事件，并使用一致的容错状态。另外，用户也可以通过注册事件时间并处理回调函数的方法来实现复杂的计算。

(2) 实际上，大多数应用并不需要上述的底层抽象，而是针对核心 API 进行编程，如 DataStream API(有界或无界流数据)以及 DataSet API(有界数据集)。这些 API 为数据处理提供了大量的通用模块，如用户定义的各种各样的转换(Transformation)、连接(Join)、聚合(Aggregation)、窗口(Window)等。DataStream API 集成了底层的处理函

数,使得对一些特定的操作可以提供更低层次的抽象。DataSet API 为有界数据集提供了额外的支持,例如循环与迭代。

(3) Table API 以表为中心,能够动态地修改表(在表达流数据时)。Table API 是一种扩展的关系模型:表有二维数据结构(类似于关系数据库中的表),同时 API 提供可比较的操作,例如 select、project、join、group-by、aggregate 等。Table API 程序定义的是应该执行什么样的逻辑操作,而不是直接准确地指定程序代码运行的具体步骤。尽管 Table API 可以通过各种各样的用户自定义函数(UDF)进行扩展,但是它在表达能力上仍然比不上核心 API,不过,它使用起来会显得更加简洁(代码量更少)。除此之外,Table API 程序在执行之前会经过内置优化器进行优化。用户可以在表与 DataStream/DataSet 之间无缝切换,以允许程序将 Table API 与 DataStream API/DataSet API 混合使用。

(4) Flink 提供的最高级接口是 SQL。这一层抽象在语法与表达能力上与 Table API 类似,唯一的区别是通过 SQL 实现程序。SQL 抽象与 Table API 交互密切,同时 SQL 查询可以直接在 Table API 定义的表上执行。

3.8 Flink 的应用程序结构

如图 3-13 所示,一个完整的 Flink 应用程序结构包含如下 3 部分。

图 3-13 Flink 应用程序结构

(1) 数据源(Source):Flink 在流处理和批处理上的数据源大致有 4 类,即基于本地集合的数据源、基于文件的数据源、基于网络套接字的数据源、自定义的数据源。常见的自定义数据源包括 Apache Kafka、Amazon Kinesis Streams、RabbitMQ、Twitter Streaming API、Apache NiFi 等,当然用户也可以定义自己的数据源。

(2) 数据转换(Transformation):数据转换的各种操作包括 map、flatMap、filter、keyBy、reduce、aggregation、window、windowAll、union、select 等,可以将原始数据转换成满足要求的数据。

(3) 数据输出(Sink):数据输出是指 Flink 将转换计算后的数据发送的目的地。常见的数据输出包括写入文件、打印到屏幕、写入 Socket 、自定义 Sink 等。常见的自定义 Sink 有 Apache Kafka、RabbitMQ、MySQL、Elasticsearch、Apache Cassandra、Hadoop FileSystem 等。

图 3-14 以一段简单代码为实例,演示了 Flink 的应用程序结构。

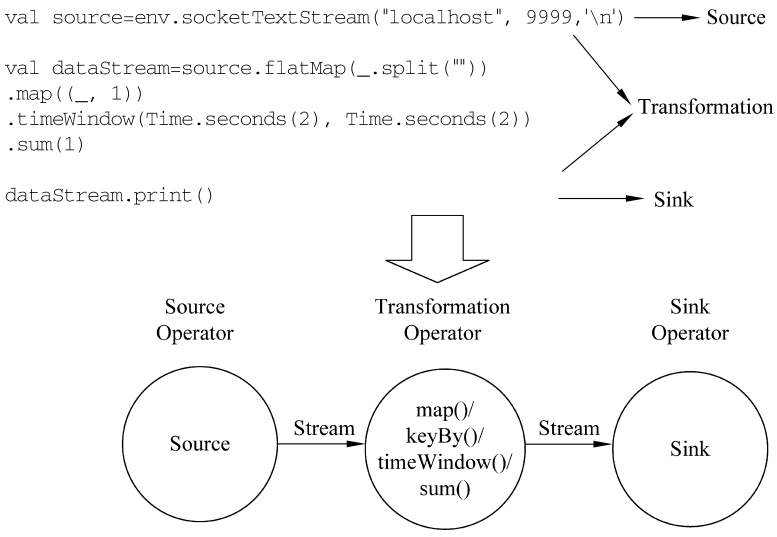


图 3-14　Fink 应用程序结构实例

3.9　Flink 的数据一致性

对于分布式流处理系统，高吞吐、低延迟往往是最主要的需求。与此同时，数据一致性在分布式流处理系统中也很重要，对于正确性要求较高的场景，“精确一次”一致性的实现往往也非常重要。如何保证分布式系统有状态计算的一致性，是 Flink 作为一个分布式流计算框架必须要解决的问题。Flink 通过异步屏障快照机制来实现“精确一次”一致性的保证，当任务中途崩溃或者取消之后，可以通过检查点或者保存点来进行恢复，实现数据流的重放，让任务达到一致性的效果，同时，这种机制不会牺牲系统的性能。

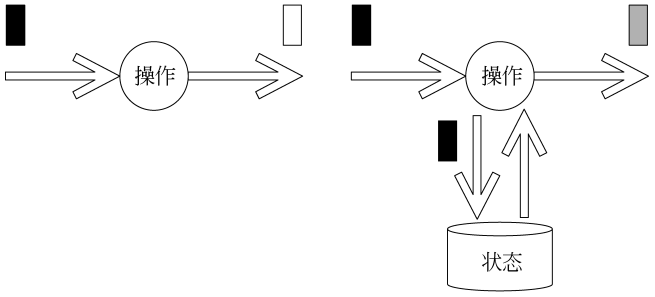
3.9.1　有状态计算

流计算分为无状态和有状态两种情况。无状态计算观察每个独立的事件，每条消息来了以后和前后其他消息都没有关系，例如一个应用程序实时接收温度传感器的数据，当温度超过 40℃时就报警，这就是无状态的数据。有状态计算则会基于多个事件输出结果，例如，计算过去 1 小时的平均温度，就属于有状态计算。

图 3-15 给出了无状态流处理与有状态流处理的区别，输入记录由黑条表示。无状态流处理每次只转换一条输入记录，并且仅根据最新的输入记录输出结果（白条）。有状态流处理则需要维护所有已处理记录的状态值，并根据每条新输入的记录更新状态，因此输出记录（灰条）反映的是综合考虑多个事件之后的结果。

3.9.2　数据一致性

当在分布式系统中引入状态时，自然也引入了一致性问题。根据正确性级别的不同，一致性可以分为如下 3 种形式。



(a) 无状态流处理　　(b) 有状态流处理

图 3-15　无状态流处理和有状态流处理的区别

(1) 最多一次：尽可能正确，但不保证一定正确。也就是说，当故障发生时，什么都不做，既不恢复丢失状态，也不重播丢失的数据。这就意味着，在系统发生故障以后，聚合结果可能会出错。

(2) 至少一次：在系统发生故障以后，聚合计算不会漏掉故障恢复之前窗口内的事件，但可能会重复计算某些事件，这通常用于实时性较高但准确性要求不高的场合。该模式意味着系统将以一种更加简单的方式对算子的状态进行快照处理，系统崩溃后恢复时，算子的状态中有一些记录可能会被重放多次。例如，失败后恢复时，统计值将大于或等于流中元素的真实值。

(3) 精确一次：在系统发生故障后，聚合结果与假定没有发生故障情况时一致。该模式意味着系统在进行恢复时，每条记录将在算子状态中只被重播一次。例如在一段数据流中，不管该系统崩溃或者重启了多少次，该统计结果将总是与流中元素的真实个数一致。这种语义加大了高吞吐和低延迟的实现难度。与“至少一次”模式相比，“精确一次”模式整体的处理速度会相对比较慢，因为在开启“精确一次”模式后，为了保证一致性，就会开启数据对齐，从而会影响系统的一些性能。

在流计算框架的发展史上，“至少一次”一致性曾经非常流行，第一代流处理框架(如 Storm 和 Samza)刚问世时只能保证“至少一次”一致性。最先保证“精确一次”一次性的系统(如 Storm Trident 和 Spark Streaming)，在性能和表现力这两方面付出了很大的代价。而 Flink 在没有牺牲性能的前提下，实现了“精确一次”一致性。

3.9.3　异步屏障快照机制

“精确一次”模式要求作业从失败恢复后的状态以及管道中的数据流要与失败时一致，通常这是通过定期对作业状态和数据流进行快照实现的。但是，传统的快照机制存在两个主要问题。

(1) 需要所有节点停止工作，即暂停整个计算过程，这个必然会影响数据处理效率和时效性。

(2) 需要保存所有节点操作中的状态以及所有在传输中的数据，这会消费大量的存储空间。

为了解决上述问题，Flink 采用了异步快照方式，它基于 Chandy-lamport 算法，制定

了应对流计算“精确一次”语义的检查点机制——异步屏障快照(Asynchronous Barrier Snapshot)机制。

异步屏障快照是一种轻量级的快照技术,能以低成本备份有向无环图(Directed Acyclic Graph,DAG)或有向有环图(Directed Cycline Graph,DCG)计算作业的状态,这使得计算作业可以频繁进行快照并且不会对性能产生明显影响。异步屏障快照机制的核心思想是,通过屏障消息来标记触发快照的时间点和对应的数据,从而将数据流和快照时间解耦,以实现异步快照操作,同时也大大降低了对管道数据的依赖(对 DAG 类作业甚至完全不依赖),减小了随之而来的快照大小。

如图 3-16 所示,检查点屏障(简称“屏障”)是一种特殊的内部消息,用于将数据流从时间上切分为多个窗口,每个窗口对应一系列连续的快照中的一个。屏障由 JobManager 定时广播给计算任务所有的 Source,并伴随数据流一起流至下游。每个屏障是属于当前快照的数据与属于下个快照的数据的分割点。例如,如图 3-16 所示,第 $n-1$ 个屏障之后、第 n 个屏障之前的所有数据都属于第 n 个检查点。下游算子如果检测到屏障的存在,就会触发快照动作,不必再关心时间无法静止的问题。

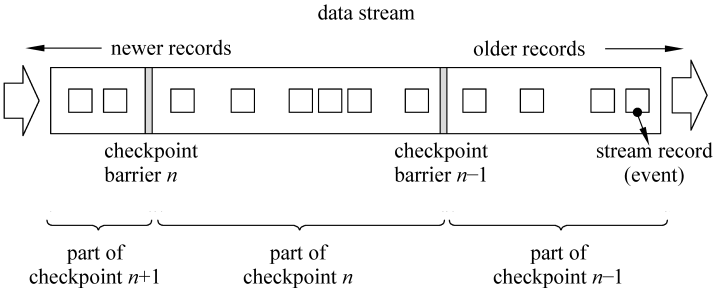


图 3-16　实时数据流屏障

异步屏障快照机制中的“异步”指的是快照数据写入的异步性,也就是说,在算子检测到屏障并触发快照之后,不会等待快照数据全部写入“状态后端”,而是一边后台写入,一边立刻继续处理数据流,并将屏障发送到下游,这样就实现了最小化延迟。

3.10　本章小结

Apache Flink 是一个分布式处理引擎,用于对无界和有界数据流进行有状态计算。Flink 以数据并行和流水线方式执行任意流数据程序,流水线运行时系统可以执行批处理和流处理程序。此外,Flink 运行时本身也支持迭代算法的执行。

近年来,数据架构设计开始由传统数据处理架构、大数据 Lambda 架构向流处理架构演变,这种转变使得 Flink 可以在大数据应用场景中“大显身手”。目前,Flink 支持的典型应用场景包括事件驱动型应用、数据分析应用和数据流水线应用。

经过多年的发展,Flink 已经形成了完备的生态系统,它的技术栈可以满足企业多种应用场景的开发需求,减轻了企业大数据应用系统的开发和维护负担。在未来,随着企业实时应用场景的不断增多,Flink 在大数据市场上的地位和作用将会更加凸显,发展前景

值得期待。

3.11 习题

1. 简述传统数据处理架构的局限性。
2. 简述大数据 Lambda 架构的优点和局限性。
3. 简述与传统数据处理架构和大数据 Lambda 架构相比,流处理架构具有哪些优点?
4. 举例说明 Flink 在企业中的应用场景。
5. 简述 Flink 核心组件栈包含哪些层次? 每个层次具体包含哪些内容?
6. Flink 的 JobManager 和 TaskManager 具体有哪些功能?
7. 简述 Flink 编程模式的层次结构。
8. 对 Spark、Flink 和 Storm 进行对比分析。
9. 简述数据一致性的 3 个级别。
10. 简述 Flink 的异步屏障快照机制。