数据流计算模型及其在大数据处理中的应用

毕倪飞，丁光耀，陈启航，徐辰，周傲英
华东师范大学数据科学与工程学院，上海 200062

摘要
如今无界、乱序的大规模数据集越来越普遍，并且消费者对这些数据集的处理需求日益复杂，如时间语义、窗口以及处理时延等。针对在无界、乱序的大规模数据集上演进的数据处理需求，探讨了大数据处理中的数据流计算模型。一方面，从执行引擎层面分析了大数据处理中的数据流计算模型所体现的数据流图；另一方面，从统一编程层面分析了大数据处理中的数据流计算模型所体现的数据流编程模型。在此基础上，进一步结合Spark批处理引擎和Flink流计算引擎等多个执行引擎，对比分析了数据流图和数据流编程模型在2类执行引擎中的具体实现。

关键词
数据流计算模型；大数据处理；执行引擎；统一编程

中图分类号：TP274 文献标识码：A doi: 10.11959/j.issn.2096-0271.2020025

Dataflow model and its applications in big data processing

BI Nifei, DING Guangyao, CHEN Qihang, XU Chen, ZHOU Aoying
School of Data Science & Engineering, East China Normal University, Shanghai 200062, China

Abstract
Unbounded, unordered and large scale datasets are increasingly common in recent years. Meanwhile, the processing requirements from data consumers are becoming more and more sophisticated, such as event time, window and latency. In order to deal with the evolved processing requirements on these unbounded, unordered and large scale datasets, the dataflow model in big data processing was introduced. On one hand, the dataflow graph of the dataflow model in big data processing was analyzed from the level of execution engine. On other hand, the dataflow programming model of the dataflow model in big data processing was analyzed from the level of unified programming. Furthermore, the different implementations of dataflow graph and dataflow programming model in multiple execution engines were analyzed, including Spark, a batch processing engine, and Flink, a stream processing engine.

Key words
dataflow model, big data processing, execution engine, unified programming

1 引言

计算机体系结构的计算模型可以分为控制流和数据流[1-5]两大类。控制流计算机也被称为冯·诺伊曼型计算机，它是主流计算机一直采用的体系结构。控制流计算模型按指令的顺序来驱动操作，数据是否参加运算取决于当时所执行的指令是否需要。数据流计算模型采用数据驱动方式，只有当一条或一组指令所需的操作数全部准备好时，才能激发相应指令的执行，执行结果又流向等待这一数据的下一条或一组指令，以驱动该条或该组指令的执行。大数据处理中也存在数据流计算模型的概念，但是大数据处理中的数据流计算模型用于完成复杂的数据处理工作，与计算机体系结构中的数据流计算模型位于不同层面，并非同一个概念。此外，Murray和McSherry等人[6-7]提出增量数据流计算模型，主要用于解决迭代算法中增量计算的问题，TensorFlow[8]的数据流模型主要用于抽象描述机器学习算法中的状态和计算，Bonna和Loubach等人[9]提出的场景感知数据流模型主要对动态应用程序进行建模和仿真，而本文大数据处理中的数据流计算模型用于低时延、正确地处理大规模、无界、乱序的数据，因此这些数据流计算模型与本文大数据处理中的数据流计算模型不是同一个概念。

现有的大数据处理系统按照执行引擎可以分为两大类。一类是基于批处理引擎的大数据处理系统，如MapReduce[10]、Spark[11-12]、Spark Streaming[13]、Structured Streaming[14]和Dryad[15]等；另一类是基于流计算引擎的大数据处理系统，如Storm[16]、Millwheel[17]、Samza[18-19]和Flink[20]等。在执行引擎层面，大数据处理中的数据流计算模型体现为数据流图。大数据处理系统通常使用数据流图来直观地表达复杂的数据处理逻辑，用户编写的数据处理流程在系统中先被转换为逻辑数据流图，该图是由一组顶点和边构成的有向无环图，该有向无环图在被交给底层执行引擎之前，根据特定的并发度又被进一步转换为物理数据流图。在统一编程层面，大数据处理中的数据流计算模型体现为数据流编程模型[21]。数据流编程模型将批处理和流计算引擎的编程方式进行抽象统一，引入了事件时间、窗口和水位线等重要概念，旨在满足数据消费者对窗口、时间语义以及处理时延等的需求。

本文结合Spark批处理引擎和Flink流计算引擎等多个执行引擎，对比分析了数据流图和数据流编程模型在两者中的具体实现。

2 数据流图

本节首先介绍大数据处理中的逻辑数据流图，其次介绍物理数据流图，最后结合Spark批处理引擎和Flink流计算引擎分析物理数据流图在两者中的具体体现。

2.1 逻辑数据流图

大数据处理系统通常使用逻辑数据流图来抽象描述整个数据处理的逻辑流程，逻辑数据流图是一个由一组顶点和边构成的有向无环图。有向无环图中的每个顶点代表了整个数据处理流程中一个特定的数据处理步骤，封装了用户定义的数据转换操作，如选择、过滤、聚合、连接等，对接收到的输入数据执行转换

操作后产生输出数据。顶点和顶点之间通过有向边连接，每条有向边代表了数据的流动和数据的依赖。与有向边起点相连的顶点表示数据的生产者，与有向边终点相连的顶点表示数据的消费者，数据由生产者流向消费者，消费者对数据的处理依赖于生产者的处理结果。如**图1**所示，该逻辑数据流图由5个表示计算逻辑的顶点和4条表示数据流动和数据依赖的有向边组成，表达了数据从读取顶点被读取后，依次流经映射、按键值分组和过滤3个顶点，并在这3个顶点中进行转换处理，最终通过保存顶点将处理结果存储下来的整个数据处理流程。

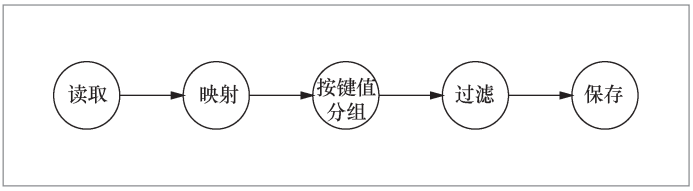


图1 逻辑数据流图

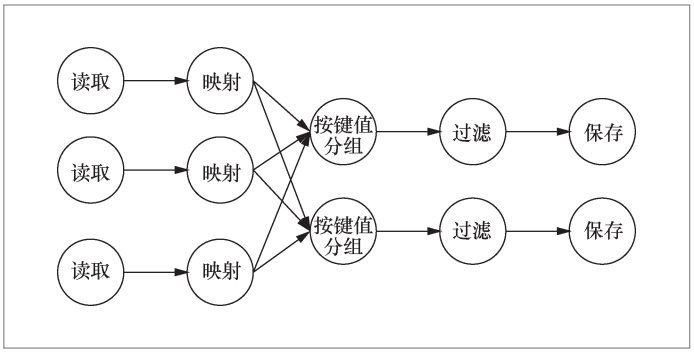


图2 物理数据流图

2.2 物理数据流图

大数据处理系统通常采用并行化的策略进行数据处理，将数据按照特定的分区策略进行分区，并为每个数据处理顶点设定并行度，让不同的数据分区流入各自相应的数据处理顶点实例，以达到并行处理的目的。但是逻辑数据流图中的顶点和边仅仅是对处理过程的逻辑抽象，即每个顶点是一个逻辑的处理步骤，不包含系统实际处理数据时并行化的概念，每条边也只描述了逻辑顶点之间的数据流动。因此，逻辑数据流图不能被直接应用到底层执行引擎，而需要先在逻辑数据流图中引入并行度，将其转换为物理数据流图后才能交给底层执行引擎。**图2**展示了**图1**中描述的逻辑数据流图根据特定的并行度转换后得到的物理数据流图，该物理数据流图中读取和映射2个数据处理顶点的并行度为3，按键值分组、过滤和保存3个数据处理顶点的并行度为2。由于批处理引擎和流计算引擎2种执行引擎的数据交换机制不同，物理数据流图在这2种执行引擎中的具体体现也有所不同。

2.2.1 批处理引擎中的物理数据流图

在批处理引擎中，一个物理数据流图通常被划分为多个阶段，阶段之间根据依赖关系按序执行，一个阶段只有等其依赖的所有阶段都执行结束后才能开始执行。每个阶段由与分区数相同个数的任务组成，一个任务负责一个分区，各个任务之间相互独立执行，不会发生数据交换。当某个任务中的一条数据被处理完成后，并不会立刻通过网络将其传输到下一个阶段的任务中，而是先将其放在缓存中，当缓存达到一定的阈值时，再将缓存中的数据溢写到本地磁盘上。只有当一个阶段中所有的任务都完成数据处理，并将处理结果写入磁盘后，才开始将这个阶段处理后的中间结果通过网络传输到下一个阶段进行后续处理。

例如，在基于批处理引擎的Spark系统中，将每个逻辑数据流图根据给定的并

行度转换为物理数据流图后，系统会根据数据交换将该物理数据流图划分为多个阶段按序执行。如**图3**所示，因为在按键值分组顶点处发生数据交换，所以整个物理数据流图在此处被切分，形成阶段0和阶段1这2个阶段。其中，阶段1中的数据处理依赖于阶段0处理后的中间结果，即2个阶段的执行存在先后顺序，阶段1只有在阶段0的处理全部完成后才能开始执行。在阶段0中，系统启动3个线程分别处理相互独立的3个分区中的数据，并将得到的中间结果存储在3个线程各自的本地磁盘上。等到阶段0中的3个线程都完成处理后，系统开始进行阶段1的处理，阶段1中启动2个线程分别负责2个分区的数据，每个线程通过网络从阶段0的中间结果处获取属于自己的数据进行后续处理。

2.2.2 流计算引擎中的物理数据流图

在流计算引擎中，物理数据流图不会被划分为多个阶段，数据流图中的所有处理任务同时启动并且长时间运行，直到整个作业完成或终止。任务之间的数据交换不需要阻塞式地将中间结果数据先写入磁盘再发送给下游任务，而是采用流水线的方式，即在处理完一条数据后立即将其发送给下游任务。这种方式有效地降低了数据处理的时延，但会导致过多的网络输入/输出（input/output，I/O）次数，从而造成系统吞吐量的下降。为了减少发送数据的网络I/O次数对吞吐量性能的影响，流计算引擎通常会设置内存缓冲区收集结果数据，当缓冲区内的数据量积累到一定大小（例如32 KB）后再一并发送给下游任务。

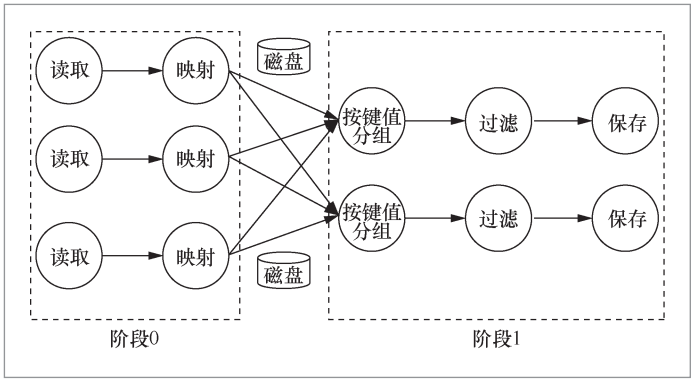


图 3 批处理引擎中的物理数据流图

在基于流计算引擎的Flink系统中，物理数据流图中的每个任务在处理一条数据后将结果放入内存缓冲区中，缓冲区不断接收任务产生的结果数据，当缓冲区数据大小达到阈值（默认32 KB）或缓冲区保存数据的时间超过设定的阈值（默认100 ms）时，系统就将缓冲区内的数据通过网络传输给下游任务。Flink通过设置内存缓冲区一次发送小批数据来避免过多的网络I/O次数，以牺牲部分时延性能为代价提升了系统的吞吐量，用户可以根据应用需求设置超时阈值，以在系统的吞吐量和时延之间进行权衡。如**图4**所示，每个物理任务都维护一个本地缓冲池，缓冲池中含有多个用于网络传输的缓冲区。最右侧已经填满数据的缓冲区将被发送到下游算子，中间还未填满数据的缓冲区直到填满数据或触发超时机制后才会被发送到下游任务，最左侧还未填充任何数据的缓冲区只有在先驱缓冲区被填满或者触发超时机制后才开始接收本地任务的输出数据。

3 数据流编程模型

本节介绍Google公司提出的数据流编程模型，首先阐明有界数据和无界数据的概念，其次介绍数据流编程模型中的时间语义和水位线2个重要概念，在此基础上依次在原语算子中介绍计算结果的

方式，在窗口操作中介绍数据按事件时间被分到哪个窗口中计算结果，在触发器中介绍被分配到窗口内的数据按处理时间何时被处理，并展示给用户，在修正策略中介绍同一窗口的多个结果之间如何相互关联。

3.1 有界数据和无界数据

当谈到有限/无限数据时，有些地方可能会将其描述为批/流数据。但是批/流数据容易让人产生误解，即批处理系统用于处理批数据，流计算系统用于处理流数据。而事实上，批处理系统也可以用于处理流数据，如Structured Streaming常被用于处理流数据，而其底层是Spark批处理系统。类似地，也可以用流计算系统Flink来处理批数据。因此，使用批/流数据概念容易造成误解，故本文统一使用有界/无界数据来表示有限/无限数据，将批和流用于描述批处理引擎和流计算引擎。

3.2 时间语义和水位线

考虑到系统处理记录的顺序和它们的原始顺序可能存在不一致性，在处理数据时，需要考虑2个时间域。

- 事件时间：事件实际发生的时间，即当该事件发生时，其所在系统的当前时间。
- 处理时间：系统执行数据处理的过程中，一个事件被数据处理系统观察到的时间。也就是，该事件被系统处理时，其所在系统的当前时间。

比如在传感器采集事件时，对应的系统时间就是事件时间，然后将事件发送到相应的数据处理系统进行处理时对应的系统时间就是处理时间。一个事件的事件时间是永远不变的，但是一个事件的处理时间会随着它在数据管道中一步步被处理而持续变化。很多时候需要根据事件时间进行数据分析，而不是处理时间。例如，收到传感器采集到的数据后，希望统计某一时间段内所监控事物的变化情况，那么依据事件时间统计更为合理（而不是处理时间）。

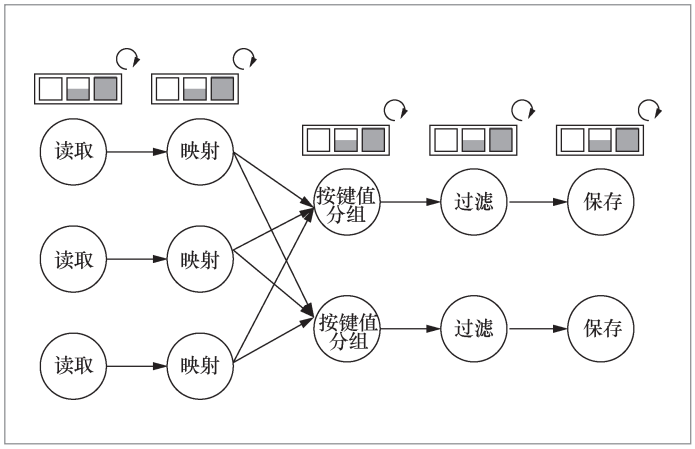


图 4 流计算引擎中的物理数据流图

在理想情况下，事件时间和处理时间的差值为零，也就意味着当事件发生时，其能得到及时处理。但在实际情况中，系统本身的一些现实影响（通信时延、调度算法、处理时长、时钟异步等）会导致这2个时间存在差值且动态波动。数据流编程模型使用水位线来衡量二者之间的差值，水位线是一个时间戳，小于这个时间戳的数据已经完全被系统处理了。如图5所示，理想情况下水位线呈现为斜率为1的直线，即事件一旦发生，系统就立即处理，事件时间与处理时间的差值为0。但现实情况下水位线通常向左倾斜，与理想情况存在偏差。

3.3 原语算子

从有界数据集的角度来看，数据流编程模型把所有的数据抽象为键值对，基于

键值对有2个核心的原语算子。

- ParDo：对数据进行并行化处理，相当于MapReduce中的map原语，将输入的键值对进行一次变换，产生若干个新的键值对。
- GroupByKey：按键值把元素重新分组，与MapReduce中的Shuffle类似，将含有相同键值的元素分到同一组。

这2个核心原语算子可以组合成聚合、去重和连接等复合算子，例如，**图6**右侧的SumByKey算子是由**图6**左侧的GroupByKey和ParDo组合而成的一个复合算子，该复合算子是一个聚合算子，统计每个字母出现次数的总和。其中，GroupByKey操作将含有相同字母的元素分配到同一组，形成新的键值对；ParDo在新的键值对上进行求和运算，得到最终的聚合结果。

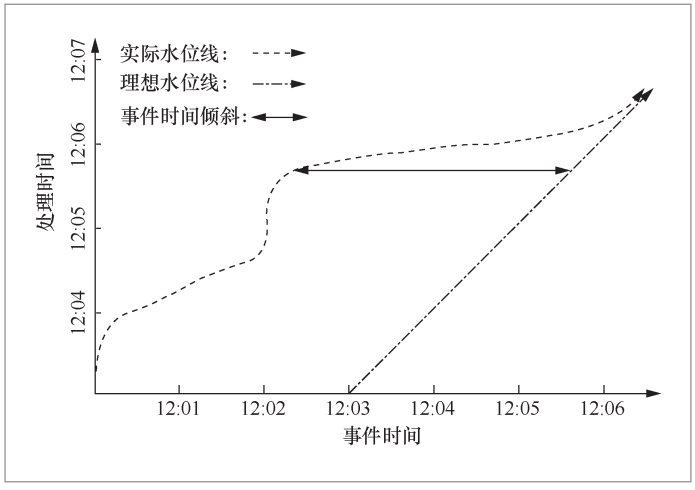


图 5 水位线

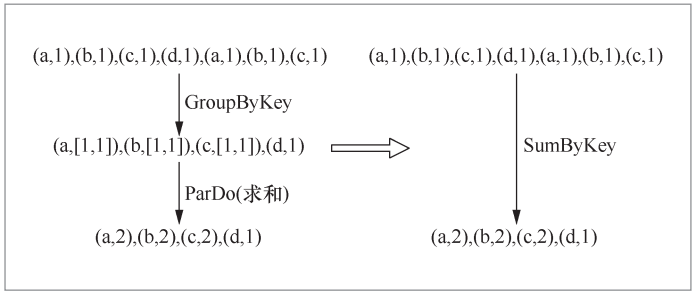


图 6 由原语算子组合成的复合算子

3.4 窗口操作

当处理无界数据时，由于ParDo原语只涉及处理单个数据，ParDo可以自然地以一次处理一条已到达数据的方式来处理无界数据。与ParDo原语不同，GroupByKey原语涉及同时处理一个给定Key上的所有数据，而由于数据是无界的，系统永远无法等到给定Key上的所有数据都到达的那一刻，所以GroupByKey无法直接用于处理无界数据。为了支持无界数据上的GroupByKey操作，需要结合窗口操作将GroupByKey重新定义为GroupByKeyAndWindow。窗口操作的核心是通过引入窗口将无界的数据集切分为有界的数据块，在每个窗口中的有界数据块上进一步实现按Key进行聚合。

窗口可以分为基于时间的窗口和基于元组的窗口，但两者本质上都是基于时间的窗口，这是由于基于元组的窗口本质上可以看作基于逻辑时间域的窗口，每个窗口中的元素带有递增的逻辑时间戳。基于时间的窗口又可以进一步分为对齐窗口和非对齐窗口，对齐窗口用于落在窗口时间范围内的所有数据，非对齐窗口用于落在窗口时间范围内的特定数据。滑动窗口和会话窗口是处理无界数据时常用的2种窗口。

- 滑动窗口：滑动窗口通过一个窗口长度和一个滑动间隔来定义，滑动间隔小于窗口长度。滑动窗口通常是对齐窗口。例如**图7**中定义了一个窗口长度为10 min的滑动窗口，每隔5 min滑动一次生成一个新的窗口。值得注意的是，此处的滑动窗口只是为了给人一种滑动的感觉，实际上3个不同的Key上都有3个窗口，而不仅仅是一个窗

口。当滑动间隔等于窗口长度时，该窗口被称为固定窗口。当滑动间隔大于窗口长度时，该窗口被称为跳跃窗口。

● 会话窗口：会话窗口是指在数据子集上有一段活动时间的窗口。会话窗口通过一个超时时间来定义，在超时时间内的所有数据都被分在同一个窗口中，形成一个会话窗口。会话窗口是非对齐窗口。例如**图7**中定义了一个超时时间为5 min的会话窗口，Key1上的会话1和会话2之间的间隔为6 min，超过了超时时间，因此被划分为2个会话。

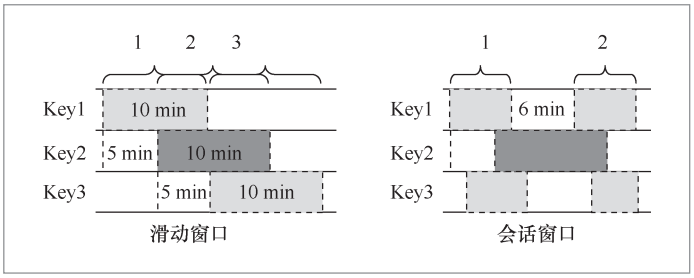


图 7 常用窗口

3.5 触发器

窗口操作决定了数据按事件时间被分到哪个窗口内一起进行聚合操作，划分好数据后进一步需要解决的是窗口内的数据按处理时间何时被处理并展示给用户。本文在第3.2节中介绍了一种水位线机制，水位线机制用于评估窗口内数据到达的完整性，每个窗口都带有起始和终止事件时间戳，一旦水位线越过了某个窗口的终止时间戳，就认为该窗口中的数据都已到达，于是处理该窗口内的数据，并将结果反馈给用户。但是水位线机制本质上只是对窗口内数据到达完整性的一种猜测，这种猜测与真实的数据到达完整性相比可能过快或过慢。如**图8**所示，如果水位线设置得过快，那么水位线之后仍有属于该窗口内的数据<a,1>继续到达，但没有被处理，造成处理结果不正确；如果水位线设置得过慢，那么当水位线越过窗口终止事件时间戳时才触发计算，可能导致整个处理结果的展示具有较高的时延。因此，仅仅使用水位线机制来触发计算和展示结果是不够的。

为了既能使用户尽快获得结果，又能保证结果的正确性，需要定义多个触发器，

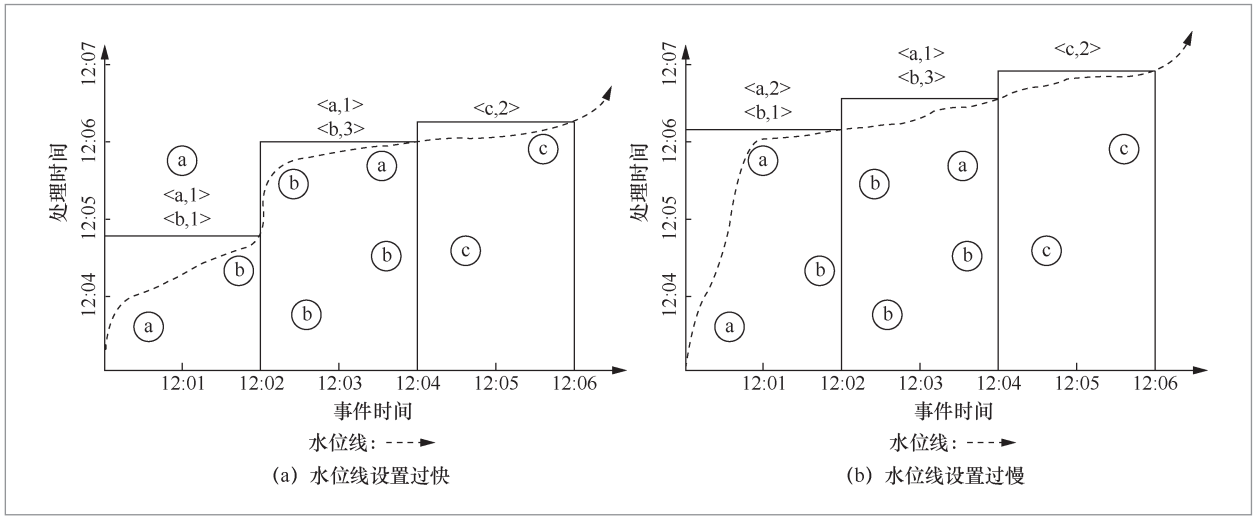


图 8 水位线设置过快或过慢

针对每个窗口多次向用户提供结果。如图9所示，在水位线触发窗口计算得到结果之前，可以定义一个基于固定处理时间的触发器向用户尽早提供结果，该触发器按处理时间每隔1 min触发一次计算并提供结果。同样地，在水位线之后到达的数据可以由一个基于元组个数的触发器来处理，该触发器每遇到一条数据就触发计算，并为用户提供结果。

3.6 修正策略

除了控制何时触发计算并展示结果之外，还需要一种方法来控制同一窗口因多次触发计算而得到的多个结果之间如何相互关联，触发器机制提供了抛弃、累积和累积并撤回3种不同的策略来修正同一个窗口的计算结果。

- 抛弃：触发器一旦触发，窗口中的内容就被抛弃，之后触发得到的结果和之前的结果不存在任何相关性。
- 累积：触发器触发后，窗口中数据的聚合结果被保留到系统状态中，之后触发的计算会累积到之前的结果上，成为针对之前结果的一个修正版本。

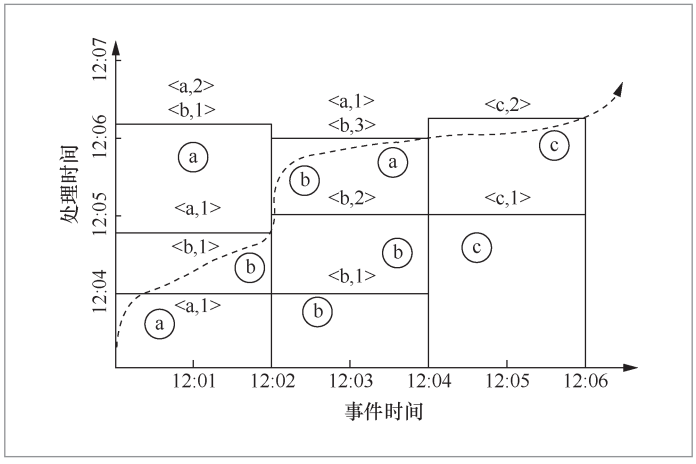


图 9 触发器

- 累积并撤回：触发器触发后，窗口中数据的聚合结果被保留到系统状态，当窗口再次触发计算时，先对上一次的结果做撤回处理，再将新的结果作为修正后的结果。

4 数据流编程模型在执行引擎中的实现

本节先介绍批处理引擎和流计算引擎各自的执行模型，再结合执行引擎实例，从数据流编程模型的时间语义和水位线机制、操作算子、窗口操作、触发器以及修正策略5个方面，分析数据流编程模型在批处理引擎和流计算引擎中的具体实现，最后对这2种执行引擎在实现数据流编程模型上的异同进行对比。

在批处理引擎的执行模型中，数据操作的粒度为一批数据，即一次读取并处理一整批数据。整个数据处理逻辑通常被划分为多个阶段，多个阶段按序被调度执行，每个阶段中的任务处理所得的中间结果需要落盘，只有等到该阶段所有数据都处理完成后，才能将中间结果发送给下一个阶段继续处理。而在流计算引擎的执行模型中，数据操作的粒度为一条数据，即一次计算一条数据。所有数据处理任务一开始就同时启动，并长时间运行直到终止，每个长时间任务随着数据的不断进入而不停地执行计算，每个任务处理完一条数据后，就将其发送给下一个任务继续处理，而不需要像批处理引擎的执行模型那样将中间结果落盘。

4.1 数据流编程模型在批处理引擎中的实现

为了支持基于事件时间语义的处理，

批处理引擎要求读入的每一批数据中的每一条元组都自带一个事件时间戳，并且在处理数据时由用户指定每条数据中哪个字段是事件时间戳，该事件时间戳用于生成水位线。按照数据流编程模型中提出的水位线机制，批处理引擎应当能够根据一个批次中的每一条数据立即更新当前的水位线，并且一旦水位线越过某个窗口的终止事件时间戳，就触发该窗口的计算，并向用户展示结果。但是这与批处理引擎的处理机制是矛盾的，因为批处理引擎只能将整个批次中的所有数据一起进行处理，而不能对一个批次中的数据进行切割，只处理一个批次中的一部分数据，所以批处理引擎难以实现数据流编程模型中的水位线机制。例如，在基于批处理引擎的Structured Streaming系统中支持事件时间语义和水位线，但无法按照水位线触发窗口计算。如**图10**所示，假设在Structured Streaming中定义一个窗口长度为1 min的滚动窗口，每隔10 s读取一批数据且一起进行处理。如果要让Structured Streaming做到根据每一条数据更新水位线并按水位线触发计算，则在该例子中，当读到(b,1,12:01:01)这条数据时，就应该将水位线按照当前收到的数据的最大事件时间戳更新为12:01:01，并触发[12:00:00,12:01:00]窗口的计算，但这违背了批处理引擎按一整批数据进行计算的机制。Structured Streaming只能做到按一整批数据进行处理，并将水位线设置为当前已收到的所有数据中的最大事件时间戳的值，下一批次中如果存在某个元组的事件时间小于上一批次生成的水位线，如该例子中的(a,1,12:00:58)的事件时间戳12:00:58小于上一批次生成的水位线12:01:02，则直接丢弃该元组，不再对其进行处理。此外，基于批处理引擎的MapReduce、Spark和Dryad等系统都没有引入时间语义和水位线。在基于批处理引擎的Spark Streaming系统中虽然引入了时间语义，但仅支持基于处理时间语义的处理，不支持基于事件时间语义的处理，因此也没有水位线机制，无法按水位线触发计算。

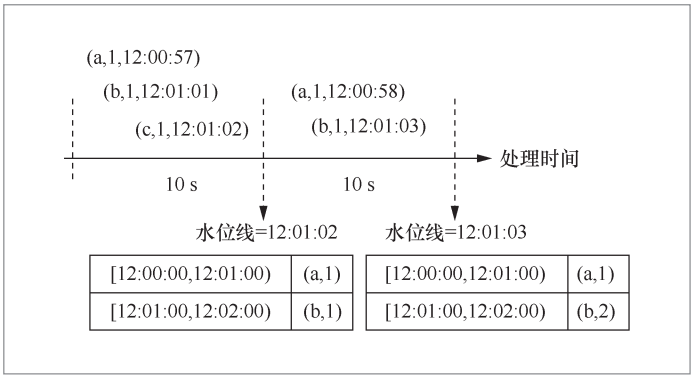


图 10 Structured Streaming 中的水位线机制

在操作算子方面，批处理引擎能够按照数据流编程模型的概念，提供类似ParDo和GroupByKeyAndWindow的算子，但前提是需要批处理引擎支持基于窗口的计算。值得注意的是，批处理引擎中的聚合操作是在每一批数据上的操作，即要等到一个批次中的数据都获取后，才对这一批次中的所有数据按键值进行分组。例如，在Structured Streaming中支持基于窗口的计算，提供了类似ParDo和GroupByKeyAndWindow的算子，最终所有算子被转换为Spark批处理引擎的RDD数据模型上的操作，一次操作一批数据。在MapReduce、Spark和Dryad等系统中，有类似ParDo的算子，一次处理一批数据，但不提供类似GroupByKeyAndWindow的算子，因为它们都不支持基于窗口的操作。在Spark Streaming中，由于Spark Streaming仅支持基于处理时间语义的窗口，因此无法提供事件时间语义上的GroupByKeyAndWindow算子，仅提供处

理时间语义上的GroupByKeyAndWindow算子。

关于窗口操作，批处理引擎按批次将数据分配到窗口。也就是说，对于到达的每一个批次，根据事件时间将该批次中的每一个元组划分到其所属的窗口中。对于触发器来说，由于批处理引擎中数据成批到达、成批处理的特性，批处理引擎难以实现按水位线和按元组个数触发计算，易实现按照固定处理时间间隔触发窗口的计算。如上所述，批处理引擎难以按水位线触发计算，按元组个数触发计算同样也要求批处理引擎切割一个批次，只计算一个批次中的一部分数据，这与批处理引擎的处理机制相背离。Structured Streaming只支持按固定时间间隔触发窗口计算，即根据固定处理时间间隔划分数据批次，并按批次触发窗口计算。如**图11**所示，假定Structured Streaming每隔10 s读取一批数据，如果要支持每到达一个元组就触发窗口计算，那么当数据元素（a,1,12:00:57）到达时就要触发计算，并将结果展示给用户，这违背了批处理引擎成批处理的机制。在该例子中，系统每隔10 s读取一次数据并触发一次计算，然后将处理结果展示给用户，再过10 s读取下一批数据并触发下一次的计算。对于每一个窗口多次触发计算而得到的多个结果，批处理引擎理论上能够提供抛弃、累积和累积并撤回3种策略来进行关联，但是目前在Structured Streaming中只实现了累积策略。在MapReduce、Spark和Dryad中，由于不支持基于窗口的计算，因此没有触发器和修正策略。而Spark Streaming支持基于处理时间语义的窗口计算，可提供基于固定时间间隔的触发器和累积修正策略。

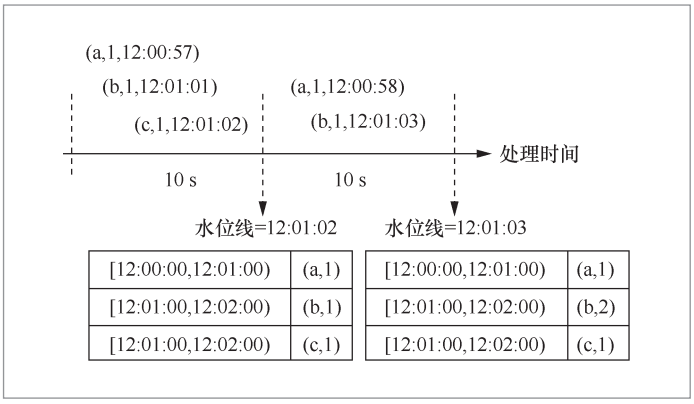


图 11 Structured Streaming 中的触发器

4.2 数据流编程模型在流计算引擎中的实现

与批处理引擎相同，为了支持事件时间，流计算引擎也要求数据源中的每条数据都自带事件时间戳，并由用户指定每条数据中的某个字段作为事件时间戳。但是流计算引擎的水位线机制与批处理引擎不同。对于流计算引擎来说，每来一条数据就立即处理一条数据，因此它能够做到来一条数据就更新一次水位线，并当水位线越过某个窗口的终止事件时间戳时，就触发该窗口的计算并将结果反馈给用户。例如，基于流计算引擎的Flink系统支持事件时间语义和水位线，并且能够按照水位线触发窗口计算。如**图12**所示，在Flink中定义一个窗口长度为1 min的滚动窗口，当数据（b,1,12:01:01）到达时，根据当前已收到的数据的最大事件时间戳，水位线被设置为12:01:01。此时，水位线超过了[12:00:00,12:01:00]窗口的终止事件时间戳，立即触发该窗口的计算，并将结果反馈给用户。同样地，基于流计算引擎的Storm、Millwheel、Samza系统也支持事件时间语义和水位线，并能够按照水位线触发窗口计算。

在操作算子方面，流计算引擎能够

按照数据流编程模型的概念，提供类似ParDo和GroupByKeyAndWindow的算子，但前提是系统支持基于窗口的计算。值得注意的是，与批处理引擎不同，流计算引擎是来一条数据就处理一条数据。因此在流计算引擎中，聚合操作是在每一条数据上的操作，即每来一条数据就将其按键值划分到所属的分组中。例如，在基于流计算引擎的Flink、Storm和MillWheel、Samza系统中，所有算子都是每来一条数据就处理一条数据。

窗口操作在流计算引擎中的实现与在批处理引擎中不同，流计算引擎按元组将数据分配到窗口。也就是说，每来一个元组，系统就按照事件时间戳将其分配到对应的窗口中。对于触发器，流计算引擎中每到达一条数据就处理一条数据，因此流计算引擎自然能够按元组个数触发窗口计算。如上所述，流计算引擎也能按水位线触发窗口计算。此外，流计算引擎还能做到按固定处理时间间隔触发窗口计算，只需在到达处理时间间隔时，对窗口中已到达的数据进行处理，并将处理结果展示给用户。例如，Flink提供了按水位线、按元组个数和按固定时间间隔3种触发策略的触发器，Flink甚至还支持用户自定义触发器的触发策略，以便灵活组合使用多种触发策略。如**图13**所示，在Flink中设置一个按每到达一个元组就触发窗口计算的触发器，那么当数据（a,1,12:00:57）到达时就触发计算，并将处理结果展示给用户；当数据（b,1,12:01:01）到达时，再次触发窗口计算，并将结果反馈给用户。对于每一个窗口多次触发计算而得到的多个结果，流计算引擎理论上能够提供抛弃、累积和累积并撤回3种策略来进行关联，但是目前在Flink中只实现了累积策略。在Storm中也提供了按水位线、按元组个数和按固定时间间隔3种触发策略的触发器，但是难以组合使用多种类型的触发器，在修正策略上Storm仅支持累积策略。Samza支持以上3种触发策略的触发器，并且可以基于过早触发和过晚触发的条件来组合使用各类触发器，在修正策略上同样只支持累积策略。

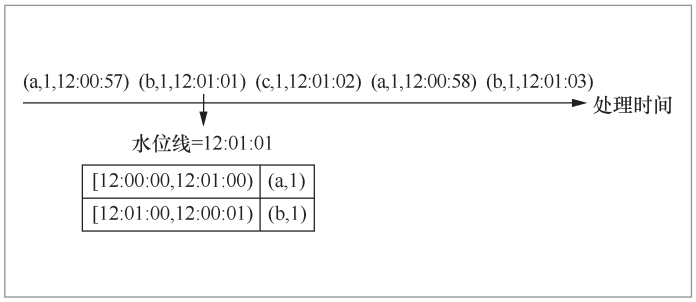


图 12 Flink 中的水位线

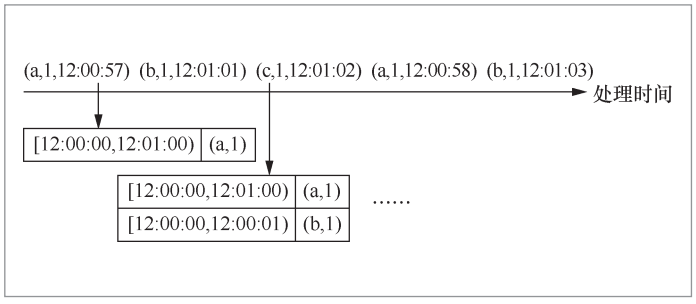


图 13 Flink 中的触发器

4.3 数据流编程模型在批/流引擎实现中的异同

数据流编程模型在批处理引擎和流计算引擎中的实现既有相同之处，也有不同之处。相同之处是两者都要求引入时间语义，并让用户指定数据中的某一列为事件时间戳列，以支持基于事件时间语义的处理。此外，修正策略在批处理引擎和流计算引擎中的实现也有相同之处，两者理论

上都能实现数据流编程模型中提出的3种修正策略。

批处理引擎和流计算引擎的数据处理机制不同，两者在实现数据流编程模型时也有不同之处。批处理引擎一次处理一批数据，因此批处理引擎中的操作算子和窗口操作的对象都是一批数据，而流计算引擎一次计算一条数据，因此流计算引擎中的操作算子和窗口操作的对象都是一条数据。此外，两者的触发机制也不同，批处理引擎难以实现按水位线或元组个数触发窗口计算，只支持按固定处理时间间隔触发计算。而流计算引擎能够实现数据流编程模型中提出的各种触发器，如基于水位线的触发器、基于元组个数的触发器以及基于固定处理时间间隔的触发器等。总体来说，流计算引擎比批处理引擎更适合用于实现数据流编程模型。

5 结束语

本文说明了计算机计算模型中的数据流计算模型与大数据处理中的数据流计算模型的不同。一方面，从执行引擎层面分析了大数据处理中的数据流计算模型体现的数据流图，并结合Spark批处理引擎和Flink流计算引擎2个典型的执行引擎，描述了数据流图在两者中的具体体现；另一方面，从统一编程层面阐述了大数据处理中的数据流计算模型体现的数据流编程模型，结合批处理引擎和流计算引擎各自的执行模型，并选取基于批处理引擎的Structured Streaming、Spark Streaming、Dryad和基于流计算引擎的Flink、Storm、Samza等多个系统，对比分析了数据流编程模型在批处理引擎和流计算引擎中的具体实现。

目前，无界、乱序的大规模数据已经越来越普遍，消费者对数据处理的需求也越来越复杂，这对大数据处理系统提出了更高的要求。本文介绍的数据流计算模型是朝这个方向迈出的重要一步，该模型将批处理引擎和流计算引擎的编程方式进行抽象统一，并引入事件时间、窗口、水位线和触发器等概念，使得大数据处理系统能够高效地应对无界、乱序的大规模数据。如今，许多大数据处理系统已经朝着该数据流计算模型发展，基于流计算引擎实现该数据流计算模型的大数据处理系统将是一个研究方向，但是基于批处理引擎的大数据处理系统在时延性和触发机制等方面还存在缺陷，需要进一步研究。

参考文献:

[1] VEEN A H. Dataflow machine architecture[J]. ACM Computing Surveys, 1986, 18(4): 365-396.

[2] SRINI V P. An architectural comparison of dataflow systems[J]. IEEE Computer, 1986, 19(3): 68-88.

[3] DENNIS J B, MISUNAS D P. A preliminary architecture for a basic data-flow processor[C]// The 2nd Annual Symposium on Computer Architecture. New York: ACM Press, 1975: 126-132.

[4] RUMBAUGH J. A data flow multiprocessor[J]. IEEE Transactions on Computers, 1977, 26(2): 138-146.

[5] DAVIS A L. A data flow evaluation system based on the concept of recursive locality[C]// The 1979 International Workshop on Managing Requirements Knowledge. Piscataway: IEEE Press, 1979: 1079-1086.

[6] MCSHERRY F, MURRAY D G, ISAACS R, et al. Differential dataflow[C]// The 6th Biennial Conference on Innovative Data

Systems Research. [S.l.:s.n.], 2013.

[7] MURRAY D G, MCSHERRY F, ISAACS R, et al. Naiad: a timely dataflow system[C]// The 24th ACM Symposium on Operating Systems Principles. New York: ACM Press, 2013: 439-455.

[8] ABADI M, BARHAM P, CHEN J, et al. TensorFlow: a system for large-scale machine learning[C]// The 12th USENIX Symposium on Operating Systems Design and Implementation. Berkeley: USENIX Association, 2016: 265-283.

[9] BONNA R, LOUBACH D S, UNGUREANU G, et al. Modeling and simulation of dynamic applications using scenario-aware dataflow[J]. ACM Transactions on Design Automation of Electronic Systems, 2019, 24(5): 1-29.

[10] DEAN J, GHEMAWAT S. MapReduce: simplified data processing on large clusters[J]. Communications of the ACM, 2008, 51(1): 107-113.

[11] ZAHARIA M, CHOWDHURY M, DAS T, et al. Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing[C]// The 9th USENIX Conference on Networked Systems Design and Implementation. Berkeley: USENIX Association, 2012: 2.

[12] ZAHARIA M, CHOWDHURY M, FRANKLIN M J, et al. Spark: cluster computing with working sets[C]// The 2nd USENIX Workshop on Hot Topics in Cloud Computing. Berkeley: USENIX Association, 2010: 10.

[13] ZAHARIA M, DAS T, LI H, et al. Discretized streams: fault-tolerant streaming computation at scale[C]// The 24th Symposium on Operating Systems Principles. New York: ACM Press, 2013: 423-438.

[14] ARMBRUST M, DAS T, TORRES J, et al. Structured streaming: a declarative API for real-time applications in Apache Spark[C]// The 2018 International Conference on Management of Data. New York: ACM Press, 2018: 601-613.

[15] ISARD M, BUDIU M, YU Y, et al. Dryad: distributed data-parallel programs from sequential building blocks[C]// The 2007 EuroSys Conference. New York: ACM Press, 2007: 59-72.

[16] TOSHNIWAL A, TANEJA S, SHUKLA A, et al. Storm@twitter[C]// The 2014 ACM SIGMOD International Conference on Management of Data. New York: ACM Press, 2014: 147-156.

[17] AKIDAU T, BALIKOV A, BEKIROĞLU K, et al. MillWheel: fault-tolerant stream processing at internet scale[J]. Proceedings of the VLDB Endowment, 2013, 6(11): 1033-1044.

[18] NOGHABI S A, PARAMASIVAM K, PAN Y, et al. Samza: stateful scalable stream processing at LinkedIn[J]. Proceedings of the VLDB Endowment, 2017, 10(12): 1634-1645.

[19] PATHIRAGE M, HYDE J, PAN Y, et al. SamzaSQL: scalable fast data management with streaming SQL[C]// The 2016 IEEE International Parallel and Distributed Processing Symposium Workshops. Piscataway: IEEE Press, 2016: 1627-1636.

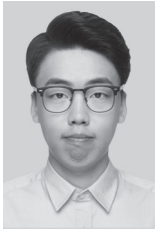
[20] CARBONE P, KATSIFODIMOS A, EWEN S, et al. Apache Flink: stream and batch processing in a single engine[J]. Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, 2015, 38(4): 28-38.

[21] AKIDAU T, BRADSHAW R, CHAMBERS C, et al. The dataflow model: a practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing[J]. Proceedings of the VLDB Endowment, 2015, 8(12): 1792-1803.

作者简介

毕倪飞(1996-),男,华东师范大学数据科学与工程学院硕士生,主要研究方向为异构分布式系统中的查询优化。

丁光耀(1996-),男,华东师范大学数据科学与工程学院博士生,主要研究方向为并行与分布式系统。



陈启航(1996-),男,华东师范大学数据科学与工程学院硕士生,主要研究方向为异构分布式计算中的查询优化。

徐辰(1988-),男,华东师范大学数据科学与工程学院副教授、硕士生导师,主要研究方向为大规模分布式数据管理。

周傲英(1965-),男,博士,华东师范大学副校长、"智能+"研究院院长、数据科学与工程学院教授。现任第七届国务院学位委员会学科评议组成员,中国计算机学会会士,上海市计算机学会副理事长,《计算机学报》《大数据》期刊副主编。曾入选"长江学者计划"特聘教授,曾获国家杰出青年基金项目资助,主要研究方向为数据库、数据管理、数据驱动的计算教育学,以及教育科技(EduTech)、物流科技(LogTech)等基于数据的应用科技。

收稿日期: 2020-01-19

通信作者: 徐辰, cxu@dase.ecnu.edu.cn

基金项目: 国家重点研发计划基金资助项目(No.2018YFB1003400);国家自然科学基金资助项目(No.61902128);上海市青年科技英才扬帆计划基金资助项目(No.19YF1414200)

Foundation Items: The National Key Research and Development Program of China(No.2018YFB1003400), The National Natural Science Foundation of China(No.61902128), Shanghai Sailing Program(No.19YF1414200)