深入理解Flink核心技术

李呈祥 · 1 年前

Flink项目是大数据处理领域最近冉冉升起的一颗新星，其不同于其他大数据项目的诸多特性吸引了越来越多的人关注Flink项目。本文将深入分析Flink一些关键的技术与特性，希望能够帮助读者对Flink有更加深入的了解，对其他大数据系统的开发者也能有所裨益。

注：本文假设读者对MapReduce，Spark及Storm等大数据处理系统有基本了解，同时熟悉流处理与批处理的基本概念。

Flink简介

Flink的核心是一个流式的数据流执行引擎，其针对数据流的分布式计算提供了数据分布，数据通信以及容错机制等功能。基于流执行引擎，Flink提供了诸多更高抽象层的API以方便用户编写分布式任务：

1. DataSet API, 对静态数据进行批处理操作，将静态数据抽象成分布式的数据集，用户可以方便的采用Flink提供的各种操作符对分布式数据集进行各种操作，支持Java，Scala和Python。

2. DataStream API，对数据流进行流处理操作，将流式的数据抽象成分布式的数据流，用户可以方便的采用Flink提供的各种操作符对分布式数据流进行各种操作，支持Java和Scala。

3. Table API，对结构化数据进行查询操作，将结构化数据抽象成关系表，并通过Flink提供的类SQL的DSL对关系表进行各种查询操作，支持Java和Scala。

此外，Flink还针对特定的应用领域提供了领域库，例如：

1. Flink ML，Flink的机器学习库，提供了机器学习Pipelines API以及很多的机器学习算法实现。

2. Gelly，Flink的图计算库，提供了图计算的相关API以及很多的图计算算法实现。

Flink的技术栈如下图所示：

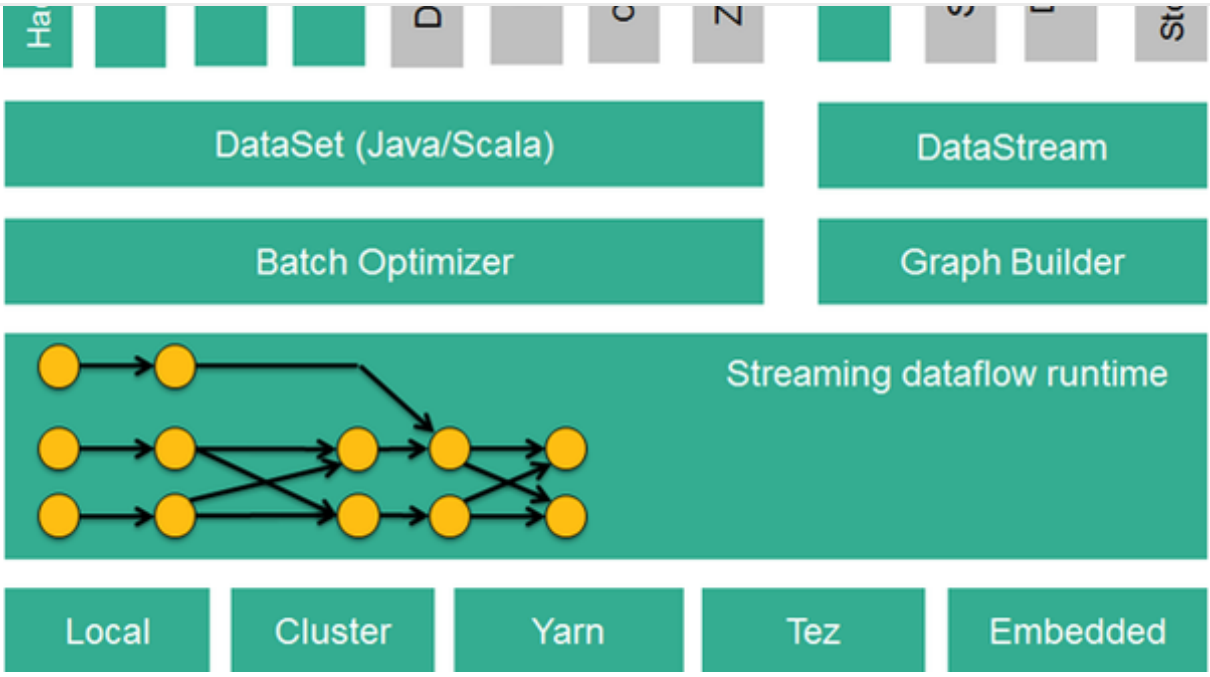


图1 Flink技术栈

此外，Flink也可以方便地和其他的Hadoop生态圈的项目集成，例如，Flink可以读取存储在HDFS或HBase中的静态数据，以Kafka作为流式的数据源，直接重用MapReduce/Storm代码，或是通过YARN申请集群资源等等。

统一的批处理与流处理系统

在大数据处理领域，批处理任务与流处理任务一般被认为是两种不同的任务，一个大数据项目一般会被设计为只能处理其中一种任务，例如Apache Storm，Apache Smaza只支持流处理任务，而Aapche MapReduce， Apache Tez，Apache Spark只支持批处理任务。Spark Streaming是Apache Spark之上支持流处理任务的子系统，看似一个特例，实则不然。Spark Streaming采用了一种micro-batch的架构，即将输入的数据流切分成细粒度的batch数据，对于每一个batch数据，以此为输入提交一个批处理Spark任务，所以Spark Streaming本质上还是基于Spark批处理系统对流式数据进行处理，和Apache Storm，Apache Smaza等完全流式的数据处理方式完全不同。Flink能够同时处理批处理任务与流处理任务，其灵活的执行引擎支持完全原生的批量的数据处理和流式的数据处理。

在执行引擎这一层，流处理系统与批处理系统最大的不同在于节点间数据传输的方式。对于一个流处理系统，其节点间数据传输的标准模型是：当一条数据被处理完成后，序列化到缓存中，然后立刻通过网络传输到下一个节点，由下一个节点继续处理。而对于一个批处理系统，其节点间数据传输的标准模型是：当一条数据被处理完成后，序列化到缓存中，并不会立刻通过网络传输到下一个节点，当缓存写满，就持久化到本地硬盘上，当所有数据都被处理完成后，才开始将处理后的数据通过网络传输到下一个节点。这两种数据传输模式是两个极端，对应的是流处理系统

缓存块超时值指定缓存块的传输时机。如果缓存块的超时值为0，则Flink的数据传输方式类似上面提到的流处理系统的标准模型，此时系统可以获得最低的处理延迟。如果缓存块的超时值为无限大，则Flink的数据传输方式类似上面提到的批处理系统的标准模型，此时系统可以获得最高的处理吞吐量。同时缓存块的超时值也可以设置为0到无限大之间的任意值。缓存块的超时阈值越小，则Flink流处理执行引擎的数据处理延迟越低，但吞吐量也会越低，缓存块的超时阈值越大时，则反之。通过调整缓存块的超时阈值，用户可根据自己的需要灵活的权衡Flink的延迟和吞吐量。

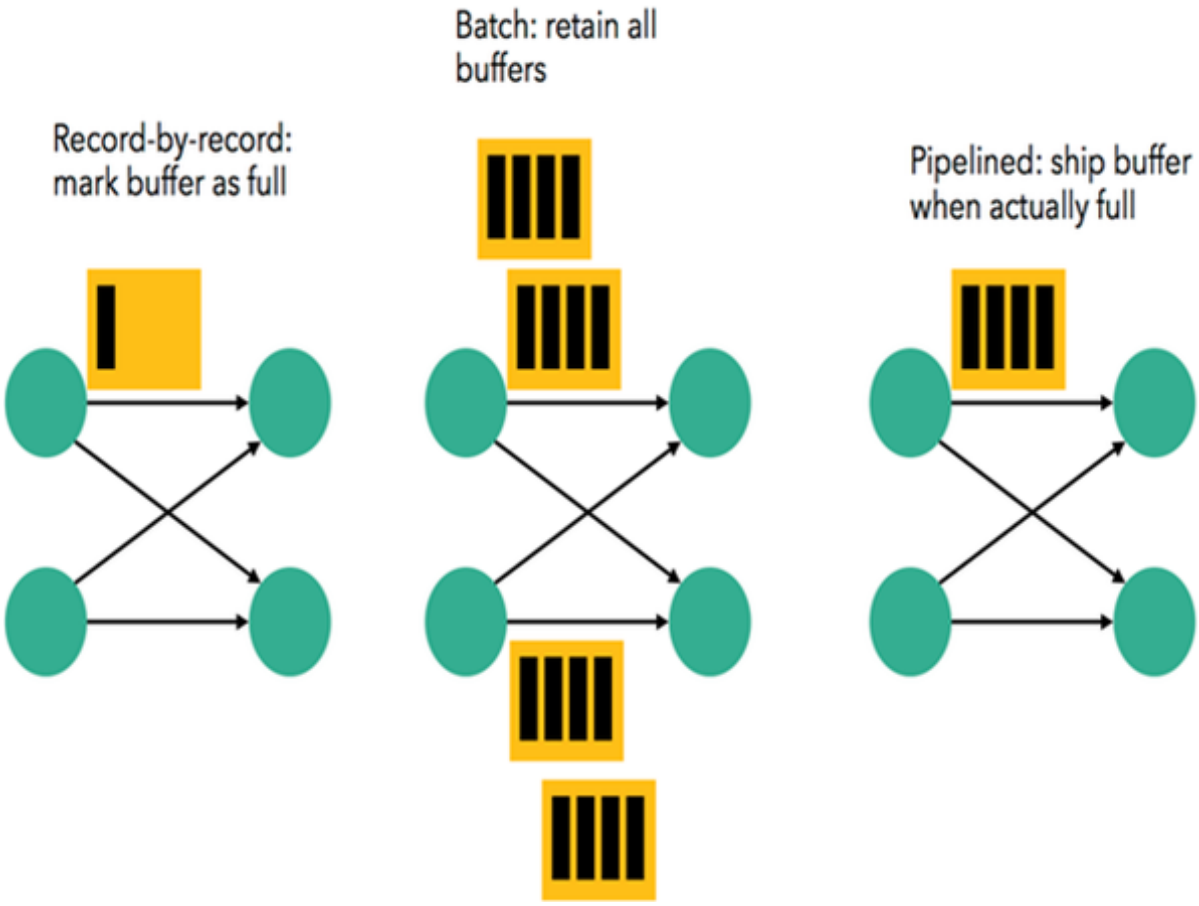


图2 Flink执行引擎数据传输模式

在统一的流式执行引擎的基础上，Flink同时支持了流处理系统与批处理系统，并且保证了其流处理系统与批处理系统的性能（延迟，吞吐量等），相对于其他原生的流处理与批处理系统，并没有因为统一的执行引擎而受到影响。用户可以在Flink上同时执行批处理任务与流处理任务，这大大减轻了用户安装，部署，监控，维护等成本。

Flink流处理的容错机制

对于一个分布式系统来说，单个进程或是节点崩溃导致整个Job失败是经常发生的事情，在异常发生的时候不会丢失用户数据，并能够自动恢复是分布式系统的需要支持的特性之一。本节主要介绍Flink流处理系统对于任务级别的容错机制。

可。但是在流处理系统中，由于数据源是无限的数据流，一个流处理任务甚至可能会执行几个月，将所有数据缓存或是持久化，留待以后重复访问基本上是不可行的。Flink基于分布式快照与可部分重发的数据源实现了容错，用户可自定义对整个Job进行快照的时间间隔，当出现任务失败时，Flink将整个Job恢复到最近一次快照的状态，并从数据源重发快照之后的数据。

Flink的分布式快照的实现借鉴了Chandy和Lamport在1985年发表的一篇关于分布式快照的论文，其实现的主要思想如下：

按照用户自定义的分布式快照间隔时间，Flink会在定时在所有数据源中插入一种特殊的快照标记消息，这些快照标记消息和其他消息一样在DAG中流动，但是不会被用户定义的业务逻辑所处理，每一个快照标记消息都将其所在的数据流分成两部分：本次快照数据和下次快照数据。

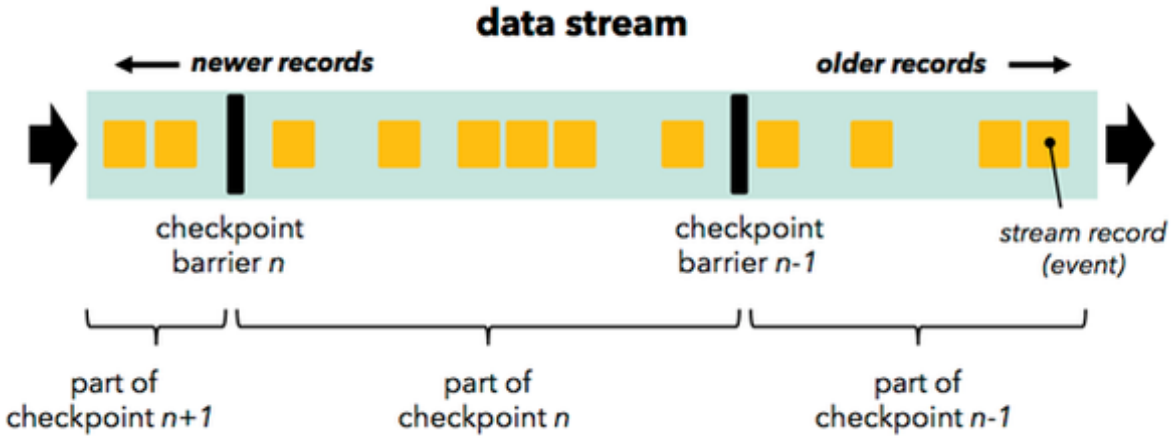


图3 Flink包含快照标记消息的消息流

快照标记消息沿着DAG流经各个操作符，当操作符处理到快照标记消息时，会对自己的状态进行快照，并存储起来。当一个操作符有多个输入的时候，Flink会将先抵达的快照标记消息及其之后的消息缓存起来，当所有的输入中对应该次快照的快照标记消息全部抵达后，操作符对自己的状态快照并存储，之后处理所有快照标记消息之后的已缓存消息。操作符对自己的状态快照并存储可以是异步与增量的操作，并不需要阻塞消息的处理。分布式快照的流程如下图所示：

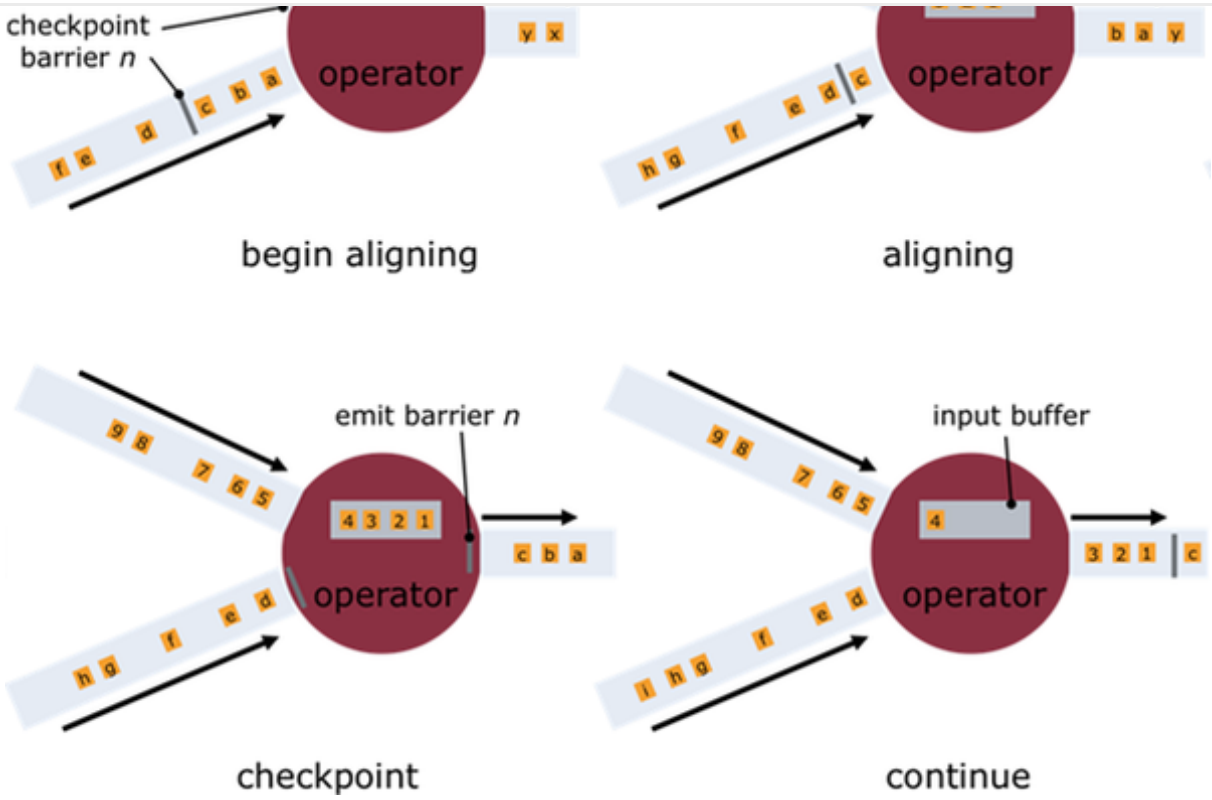


图4 Flink分布式快照流程图

当所有的Data Sink(终点操作符)都收到快照标记信息并对自己的状态快照和存储后，整个分布式快照就完成了，同时通知数据源释放该快照标记消息之前的所有消息。若之后发生节点崩溃等异常情况时，只需要恢复之前存储的分布式快照状态，并从数据源重发该快照以后的消息就可以了。

Exactly-Once是流处理系统需要支持的一个非常重要的特性，它保证每一条消息被流处理系统处理一次，且仅被处理一次，许多流处理任务的业务逻辑都依赖于Exactly-Once特性。相对于At-Least-Once或是At-Most-Once, Exactly-Once特性对流处理系统的要求更严格，实现也更困难。Flink基于分布式快照实现了Exactly-Once特性。

相对于其他流处理系统的容错方案，Flink基于分布式快照的方案在功能和性能方面都具有很多优点，包括：

1. 低延迟。由于操作符状态的存储可以是异步的，所以进行快照的过程基本上不会阻塞消息的处理，对消息的延迟不会产生负面的影响。

2. 高吞吐量。当操作符状态较少时，对吞吐量基本没有影响。当操作符状态较多时，相对于其他的容错机制，分布式快照的时间间隔是用户自定义的，所以用户可以权衡错误恢复时间和吞吐量的要求，调整分布式快照的时间间隔。

3. 与业务逻辑的隔离。Flink的分布式快照机制与用户的业务逻辑是完全隔离的，用户的业务逻辑

4. 错误恢复代价。分布式快照的时间间隔越短，错误恢复的时间越少，与吞吐量负相关。

Flink流处理的时间窗口

对于流处理系统来说，流入的消息是无限的，所以对于聚合或是连接等操作，流处理系统需要对流入的消息进行分段，然后基于每一段数据进行聚合或是连接等操作。消息的分段即称为窗口，流处理系统支持的窗口有很多类型，最常见的就是时间窗口，基于时间间隔对消息进行分段处理。本节主要介绍Flink流处理系统支持的各种时间窗口。

对于目前大部分流处理系统来说，时间窗口一般是根据Task所在节点的本地时钟来进行切分，这种方式实现起来比较容易，不会阻塞消息处理。但是可能无法满足某些应用的要求，例如：

1. 消息本身带有时间戳，用户希望按照消息本身的时间特性进行分段处理。

2. 由于不同节点的时钟可能不同，以及消息在流经各个节点时延迟不同，在某个节点属于同一个时间窗口处理的消息，流到下一个节点时可能被切分到不同的时间窗口中，从而产生不符合预期的结果。

Flink支持三种类型的时间窗口，分别适用于用户对于时间窗口不同类型的要求：

1. Operator Time。根据Task所在节点的本地时钟来进行切分的时间窗口。

2. Event Time。消息自带时间戳，根据消息的时间戳进行处理，确保时间戳在同一个时间窗口的所有消息一定会被正确处理。由于消息可能是乱序流入Task的，所以Task需要缓存当前时间窗口消息处理的状态，直到确认属于该时间窗口的所有消息都被处理后，才可以释放其状态。如果乱序的消息延迟很高的话，会影响分布式系统的吞吐量和延迟。

3. Ingress Time。有时消息本身并不带有时间戳信息，但用户依然希望按照消息而不是节点时钟划分时间窗口(例如，避免上面提到的第二个问题)。此时可以在消息源流入Flink流处理系统时，自动生成增量的时间戳赋予消息，之后处理的流程与Event Time相同。Ingress Time可以看成是Event Time的一个特例，由于其在消息源处时间戳一定是有序的，所以在流处理系统中，相对于Event Time，其乱序的消息延迟不会很高，因此对Flink分布式系统的吞吐量和延迟的影响也会更小。

Event Time时间窗口的实现

Flink借鉴了Google的MillWheel项目，通过WaterMark来支持基于Event Time时间窗口。

当操作符通过基于Event Time的时间窗口来处理数据时，它必须在确定所有属于该时间窗口的消

WaterMark标记所有小于该时间戳的消息都已流入，Flink的数据源在确认所有小于某个时间戳的消息都已输出到Flink流处理系统后，会生成一个包含该时间戳的WaterMark，插入到消息流中输出到Flink流处理系统中，Flink操作符按照时间窗口缓存所有流入的消息，当操作符处理到WaterMark时，它对所有小于该WaterMark时间戳的时间窗口的数据进行处理并发送到下一个操作符节点，然后也将WaterMark发送到下一个操作符节点。

为了保证能够处理所有属于某个时间窗口的消息，操作符必须等到大于这个时间窗口的WaterMark之后，才能开始对该时间窗口的消息进行处理，相对于基于Operator Time的时间窗口，Flink需要占用更多的内存，且会直接影响消息处理的延迟时间。对此，一个可能的优化措施是，对于聚合类的操作符，可能可以提前对部分消息进行聚合操作，当有属于该时间窗口的新消息流入时，基于之前的部分聚合结果继续计算，这样的话，只需缓存中间计算结果即可，无需缓存该时间窗口的所有消息。

对于基于Event Time时间窗口的操作符来说，流入WaterMark的时间戳与当前节点的时钟一致是最简单理想的状况了，但是在实际环境中是不可能的，由于消息的乱序以及前面节点处理效率的不同，总是会有某些消息流入时间大于其本身的时间戳，真实WaterMark时间戳与理想情况下WaterMark时间戳的差别称为Time Skew，如下图所示：

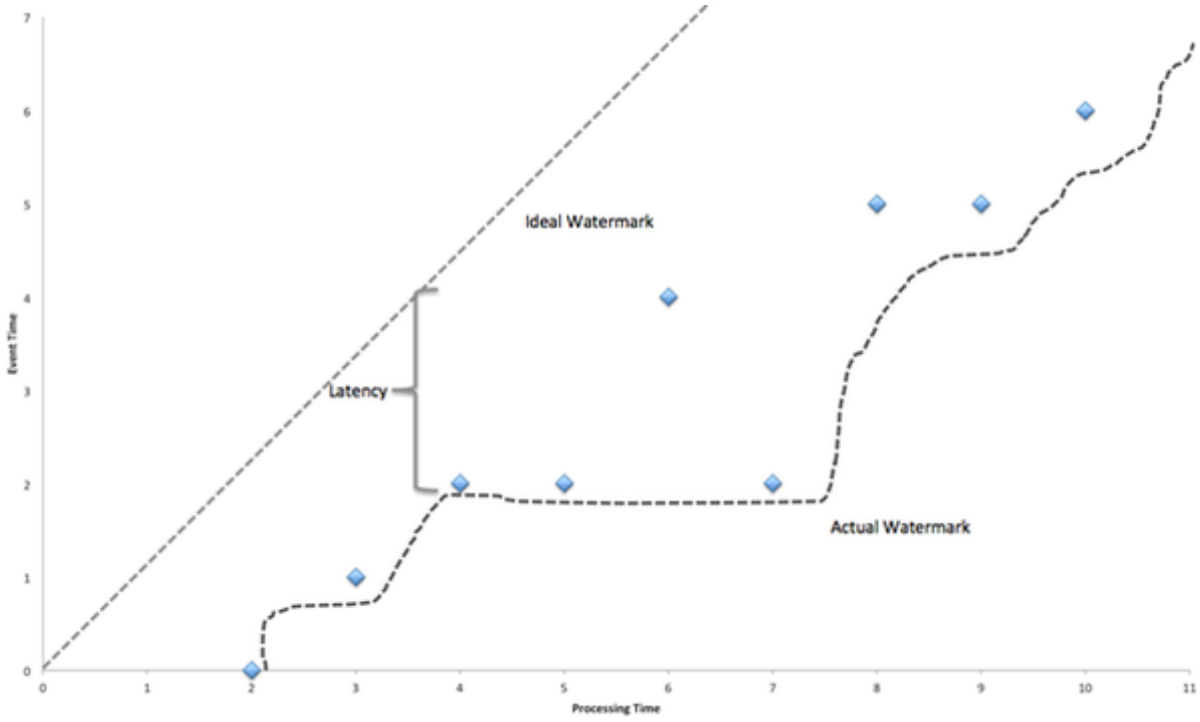


图5 WaterMark的Time Skew图

Time Skew决定了该WaterMark与上一个WaterMark之间的时间窗口所有数据需要缓存的时间，Time Skew时间越长，该时间窗口数据的延迟越长，占用内存的时间也越长，同时会对流处理系统的吞吐量产生负面影响。

在流处理系统中，由于流入的消息是无限的，所以对消息进行排序基本上被认为是不可行的。但是在Flink流处理系统中，基于WaterMark，Flink实现了基于时间戳的全局排序。

Flink基于时间戳进行排序的实现思路如下：排序操作符缓存所有流入的消息，当其接收到WaterMark时，对时间戳小于该WaterMark的消息进行排序，并发送到下一个节点，在此排序操作符中释放所有时间戳小于该WaterMark的消息，继续缓存流入的消息，等待下一个WaterMark触发下一次排序。由于WaterMark保证了其之后不会出现时间戳比它小的消息，所以可以保证排序的正确性。需要注意的是，如果排序操作符有多个节点，只能保证每个节点的流出消息是有序的，节点之间的消息不能保证有序，要实现全局有序，则只能有一个排序操作符节点。

通过支持基于Event Time的消息处理，Flink扩展了其流处理系统的应用范围，使得更多的流处理任务可以通过Flink来执行。

定制的内存管理

略，请参考上篇文章：脱离JVM？ Hadoop生态圈的挣扎与演化 - 大数据技术与实践 - 知乎专栏

总结

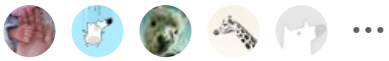
本文主要介绍了Flink项目的一些关键特性，Flink是一个拥有诸多特色的项目，包括其统一的批处理和流处理执行引擎，通用大数据计算框架与传统数据库系统的技术结合，以及流处理系统的诸多技术创新等，因为篇幅有限，Flink还有一些其他很有意思的特性没有详细介绍，比如DataSet API级别的执行计划优化器，原生的迭代操作符等，感兴趣的读者可以通过Flink的官网了解更多Flink的详细内容。希望通过本文的介绍能够让读者对Flink项目能有更多的了解，也让更多的人使用甚至参与到Flink项目中去。

引用

1. project tungsten：Project Tungsten: Bringing Spark Closer to Bare Metal
2. The "Memory Wall":Modern Microprocessors
3. flink memory management:Flink: Juggling with Bits and Bytes
4. java GC：Tuning Java Garbage Collection for Spark Applications
5. Project Valhalla:OpenJDK: Valhalla
6. java object size:dweiss/java-sizeof · GitHub

7.Big Data Performance Engineering

9. Flink Event Time: Time and Order in Streams



4 条评论

写下你的评论

凡柯

好论文

1 年前

牛尾巴

好文章。。因本人文化水平有限，希望通过你的专业知识建立数据模型。空间和时间交易思路。

10 个月前

兴中

感谢 最近工作中开始需要用到flink

king zin

昨天刚好在 survey. Flink 阿里有个小伙 jark 写的也不错 jark flink

5 天前

文章被以下专栏收录

大数据技术与实践

主要介绍大数据相关的技术分析与业界应用。

进入专栏

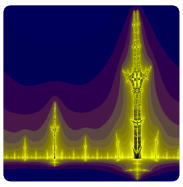
推荐阅读

大数据与数据脱敏

李呈祥 · 1 年前 发表于 大数据技术与实践

脱离JVM？ Hadoop生态圈的挣扎与演化

李呈祥 · 2 年前 发表于 大数据技术与实践



数学家救火法与数学竞赛

Yu Deng · 20 天前 · 编辑精选

不会日语也能在日本点菜 | 通用篇

Nugget Jiang · 6 天前 · 编辑精选 发表于 NJ万事屋