22-ApacheBeam的前世今生

你好，我是蔡元楠。

今天我要与你分享的主题是“ Apache Beam的前世今生”。

从这一讲开始，我们将进入一个全新的篇章。在这一讲中，我将会带领你了解Apache Beam的完整诞生历程。

让我们一起来感受一下，Google是如何从处理框架上的一无所有，一直发展到推动、制定批流统一的标准的。除此之外，我还会告诉你，在2004年发布了MapReduce论文之后，Google在大规模数据处理实战中到底经历了哪些技术难题和技术变迁。我相信通过这一讲，你将会完整地认识到为什么Google会强力推崇Apache Beam。

在2003年以前，Google内部其实还没有一个成熟的处理框架来处理大规模数据。而当时Google的搜索业务又让工程师们不得不面临着处理大规模数据的应用场景，像计算网站URL访问量、计算网页的倒排索引（ Inverted Index ）等等。

那该怎么办呢？这个答案既简单又复杂：自己写一个。

没错，当时的工程师们需要自己写一个自定义的逻辑处理架构来处理这些数据。因为需要处理的数据量非常庞大，业务逻辑不太可能只放在一台机器上面运行。很多情况下，我们都必须把业务逻辑部署在分布式环境中。所以，这个自定义的逻辑处理架构还必须包括容错系统（ Fault Tolerant System ）的设计。

久而久之，Google内部不同组之间都会开发出一套自己组内的逻辑处理架构。因为工程师们遇到的问题很多都是相似的，开发出来的这些逻辑处理架构很多时候也都是大同小异，只是有一些数据处理上的逻辑差别而已。这无疑就变成了大家一起重复造轮子的情况。

这时候，就有工程师想到，能不能改善这一种状况。MapReduce的架构思想也就由此应运而生。

MapReduce

其实MapReduce的架构思想可以从两个方面来看。

一方面，它希望能提供一套简洁的API来表达工程师数据处理的逻辑。另一方面，要在这一套API底层嵌套一套扩展性很强的容错系统，使得工程师能够将心思放在逻辑处理上，而不用过于分心去设计分布式的容错系统。

这个架构思想的结果你早就已经知道了。MapReduce这一套系统在Google获得了巨大成功。在2004年的时候，Google发布的一篇名为“MapReduce: Simplified Data Processing on Large Clusters”的论文就是这份成果的总结。

在MapReduce的计算模型里，它将数据的处理抽象成了以下这样的计算步骤

- Map：计算模型从输入源（ Input Source ）中读取数据集合，这些数据在经过了用户所写的逻辑后生成出一个临时的键值对数据集（ Key/Value Set ）。MapReduce计算模型会将拥有相同键（ Key ）的数据集集中起来然后发送到下一阶段。这一步也被称为Shuffle阶段。

- Reduce：接收从Shuffle阶段发送过来的数据集，在经过了用户所写的逻辑后生成出零个或多个结果。

很多人都说，这篇MapReduce论文是具有划时代意义的。可你知道为什么都这么说吗？

这是因为Map和Reduce这两种抽象其实可以适用于非常多的应用场景，而MapReduce论文里面所阐述的容错系统，可以让我们所写出来的数据处理逻辑在分布式环境下有着很好的可扩展性（Scalability）。

MapReduce在内部的成功使得越来越多的工程师希望使用MapReduce来解决自己项目的难题。

但是，就如我在模块一中所说的那样，使用MapReduce来解决一个工程难题往往会涉及到非常多的步骤，而每次使用MapReduce的时候我们都需要在分布式环境中启动机器来完成Map和Reduce步骤，以及启动Master机器来协调这两个步骤的中间结果（Intermediate Result），消耗不少硬件上的资源。

这样就给工程师们带来了以下一些疑问：

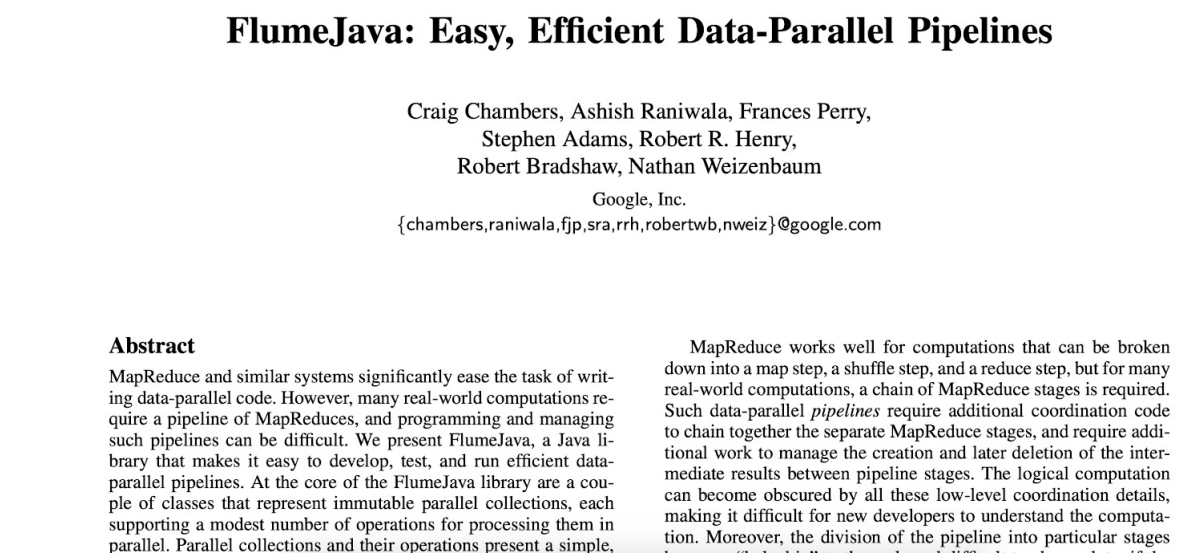
- 我们的项目数据规模是否真的需要运用MapReduce来解决呢？是否可以在一台机器上的内存中解决呢？
- 我们所写的MapReduce项目是否已经是最优的呢？因为每一个Map和Reduce步骤这些中间结果都需要写在磁盘上，会十分耗时。是否有些步骤可以省略或者合并呢？我们是否需要让工程师投入时间去手动调试这些MapReduce项目的性能呢？

问题既然已经提出来了，Google的工程师们便开始考虑是否能够解决上述这些问题。最好能够让工程师（无论是新手工程师亦或是经验老到的工程师）都能专注于数据逻辑上的处理，而不用花更多时间在测试调优上。

FlumeJava就是在这样的背景下诞生的。

FlumeJava

这里，我先将FlumeJava的成果告诉你。因为FlumeJava的思想又在Google内容获得了巨大成功，Google也希望将这个思想分享给业界。所以在2010年的时候，Google公开了FlumeJava架构思想的论文。



FlumeJava: Easy, Efficient Data-Parallel Pipelines

Craig Chambers, Ashish Raniwala, Frances Perry,
Stephen Adams, Robert R. Henry,
Robert Bradshaw, Nathan Weizenbaum
Google, Inc.
{chambers,raniwala,fjp,sra,rrh,robertwb,nweiz}@google.com

Abstract

MapReduce and similar systems significantly ease the task of writing data-parallel code. However, many real-world computations require a pipeline of MapReduces, and programming and managing such pipelines can be difficult. We present FlumeJava, a Java library that makes it easy to develop, test, and run efficient data-parallel pipelines. At the core of the FlumeJava library are a couple of classes that represent immutable parallel collections, each supporting a modest number of operations for processing them in parallel. Parallel collections and their operations present a simple,

MapReduce works well for computations that can be broken down into a map step, a shuffle step, and a reduce step, but for many real-world computations, a chain of MapReduce stages is required. Such data-parallel *pipelines* require additional coordination code to chain together the separate MapReduce stages, and require additional work to manage the creation and later deletion of the intermediate results between pipeline stages. The logical computation can become obscured by all these low-level coordination details, making it difficult for new developers to understand the computation. Moreover, the division of the pipeline into particular stages

FlumeJava的思想是将所有的数据都抽象成名为PCollection的数据结构，无论是从内存中读取的数据，还是

在分布式环境下所读取的文件。

这样的抽象对于测试代码中的逻辑是十分有好处的。要知道，想测试MapReduce的话，你可能需要读取测试数据集，然后在分布式环境下运行，来测试代码逻辑。但如果你有了PCollection这一层抽象的话，你的测试代码可以在内存中读取数据然后跑测试文件，也就是同样的逻辑既可以在分布式环境下运行也可以在单机内存中运行。

而FlumeJava在MapReduce框架中Map和Reduce思想上，抽象出4个了原始操作（Primitive Operation），分别是parallelDo、groupByKey、 combineValues和flatten，让工程师可以利用这4种原始操作来表达任意Map或者Reduce的逻辑。

同时，FlumeJava的架构运用了一种Deferred Evaluation的技术，来优化我们所写的代码。

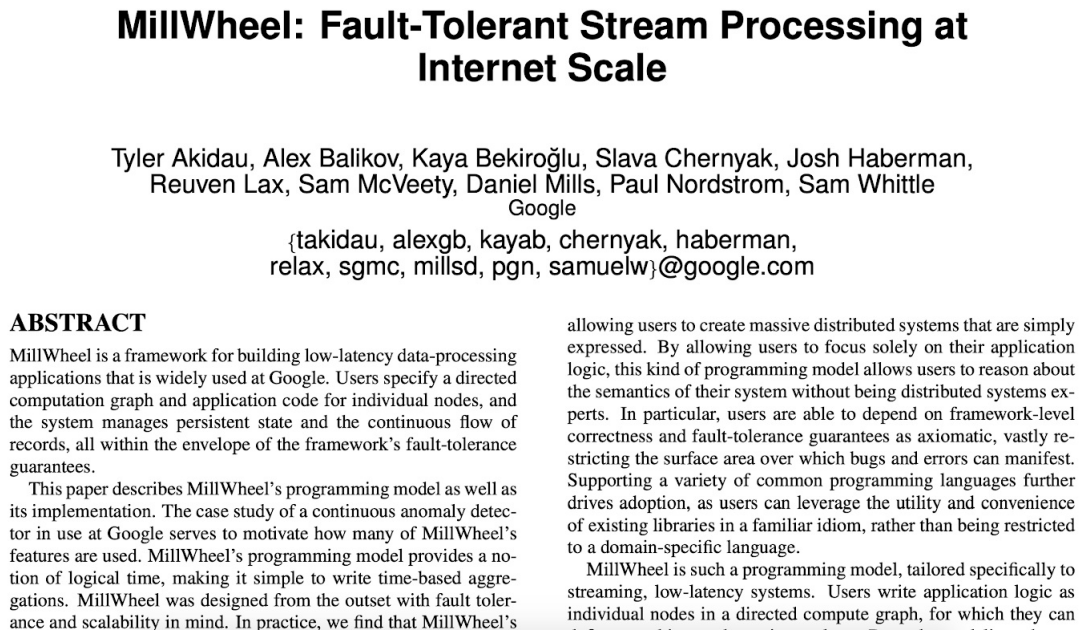
对于Deferred Evaluation，你可以理解为FlumeJava框架会首先会将我们所写的逻辑代码静态遍历一次，然后构造出一个执行计划的有向无环图。这在FlumeJava框架里被称为Execution Plan Dataflow Graph。

有了这个图之后，FlumeJava框架就会自动帮我们优化代码。例如，合并一些本来可以通过一个Map和Reduce来表达，却被新手工程师分成多个Map和Reduce的代码。

FlumeJava框架还可以通过我们的输入数据集规模，来预测输出结果的规模，从而自行决定代码是放在内存中跑还是在分布式环境中跑。

总的来说，FlumeJava是非常成功的。但是，FlumeJava也有一个弊端，那就是FlumeJava基本上只支持批处理（Batch Execution）的任务，对于无边界数据（Unbounded Data）是不支持的。所以，Google内部有着另外一个被称为Millwheel的项目来支持处理无边界数据，也就是流处理框架。

在2013年的时候，Google也公开了Millwheel思想的论文。



MillWheel: Fault-Tolerant Stream Processing at Internet Scale

Tyler Akidau, Alex Balikov, Kaya Bekiroğlu, Slava Chernyak, Josh Haberman,
Reuven Lax, Sam McVeety, Daniel Mills, Paul Nordstrom, Sam Whittle
Google
{takidau, alexgb, kayab, chernyak, haberman,
relax, sgmc, millsd, pgn, samuelw}@google.com

ABSTRACT

MillWheel is a framework for building low-latency data-processing applications that is widely used at Google. Users specify a directed computation graph and application code for individual nodes, and the system manages persistent state and the continuous flow of records, all within the envelope of the framework's fault-tolerance guarantees.

This paper describes MillWheel's programming model as well as its implementation. The case study of a continuous anomaly detector in use at Google serves to motivate how many of MillWheel's features are used. MillWheel's programming model provides a notion of logical time, making it simple to write time-based aggregations. MillWheel was designed from the outset with fault tolerance and scalability in mind. In practice, we find that MillWheel's

allowing users to create massive distributed systems that are simply expressed. By allowing users to focus solely on their application logic, this kind of programming model allows users to reason about the semantics of their system without being distributed systems experts. In particular, users are able to depend on framework-level correctness and fault-tolerance guarantees as axiomatic, vastly restricting the surface area over which bugs and errors can manifest. Supporting a variety of common programming languages further drives adoption, as users can leverage the utility and convenience of existing libraries in a familiar idiom, rather than being restricted to a domain-specific language.

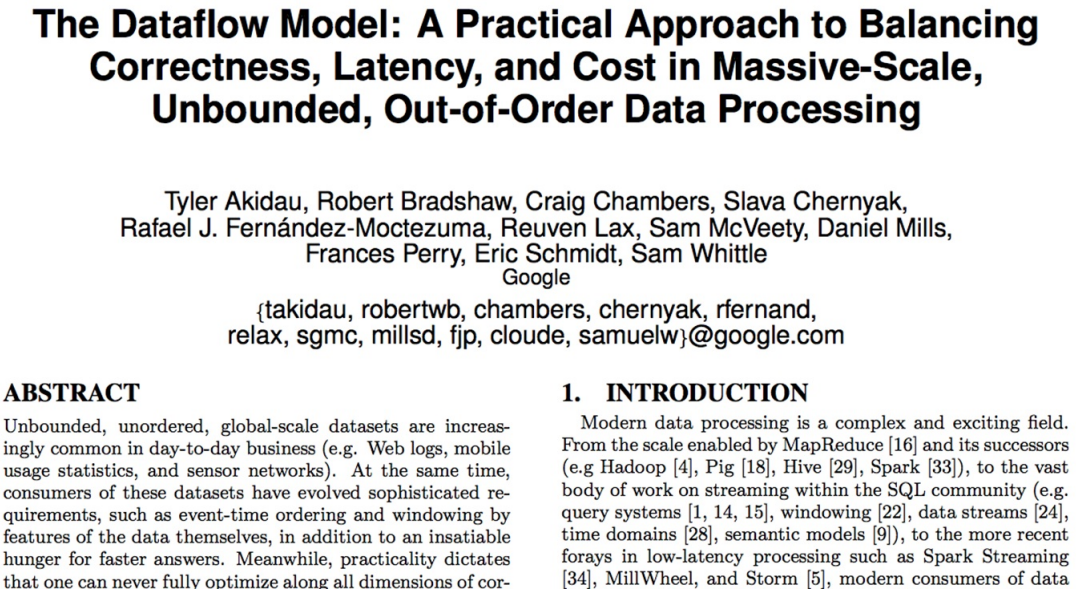
MillWheel is such a programming model, tailored specifically to streaming, low-latency systems. Users write application logic as individual nodes in a directed compute graph, for which they can

这时Google的工程师们回过头看，感叹了一下成果，并觉得自己可以再优秀一些：既然我们已经创造出好几个优秀的大规模数据处理框架了，那我们能不能集合这几个框架的优点，推出一个统一的框架呢？

这也成为了Dataflow Model诞生的契机。

Dataflow Model

在2015年时候，Google公布了Dataflow Model的论文，同时也推出了基于Dataflow Model思想的平台Cloud Dataflow，让Google以外的工程师们也能够利用这些SDK来编写大规模数据处理的逻辑。



The Dataflow Model: A Practical Approach to Balancing Correctness, Latency, and Cost in Massive-Scale, Unbounded, Out-of-Order Data Processing

Tyler Akidau, Robert Bradshaw, Craig Chambers, Slava Chernyak,
Rafael J. Fernández-Moctezuma, Reuven Lax, Sam McVeety, Daniel Mills,
Frances Perry, Eric Schmidt, Sam Whittle
Google

{takidau, robertwb, chambers, chernyak, rfernand,
relax, sgmc, millsd, fjp, cloude, samuelw}@google.com

ABSTRACT

Unbounded, unordered, global-scale datasets are increasingly common in day-to-day business (e.g. Web logs, mobile usage statistics, and sensor networks). At the same time, consumers of these datasets have evolved sophisticated requirements, such as event-time ordering and windowing by features of the data themselves, in addition to an insatiable hunger for faster answers. Meanwhile, practicality dictates that one can never fully optimize along all dimensions of cor-

1. INTRODUCTION

Modern data processing is a complex and exciting field. From the scale enabled by MapReduce [16] and its successors (e.g Hadoop [4], Pig [18], Hive [29], Spark [33]), to the vast body of work on streaming within the SQL community (e.g. query systems [1, 14, 15], windowing [22], data streams [24], time domains [28], semantic models [9]), to the more recent forays in low-latency processing such as Spark Streaming [34], MillWheel, and Storm [5], modern consumers of data

讲到这么多，你可能会有个疑问了，怎么Apache Beam还没有出场呢？别着急，Apache Beam的登场契机马上就到了。

Apache Beam

前面我说了，Google基于Dataflow Model的思想推出了Cloud Dataflow云平台，但那毕竟也需要工程师在Google的云平台上面运行程序才可以。如果有的工程师希望在别的平台上面跑该如何解决呢？

所以，为了解决这个问题，Google在2016年的时候联合了Talend、Data Artisans、Cloudera这些大数据公司，基于Dataflow Model的思想开发出了一套SDK，并贡献给了Apache Software Foundation。而它Apache Beam的名字是怎么来的呢？就如下图所示，Beam的含义就是统一了批处理和流处理的一个框架。

这就是Apache Beam的发展历史，从中你可以看到它拥有很多优点，而这也是我们需要Beam的原因。

在现实世界中，很多时候我们不可避免地需要对数据同时进行批处理和流处理。Beam提供了一套统一的API来处理这两种数据处理模式，让我们只需要将注意力专注于在数据处理的算法上，而不用再花时间去对两种数据处理模式上的差异进行维护。

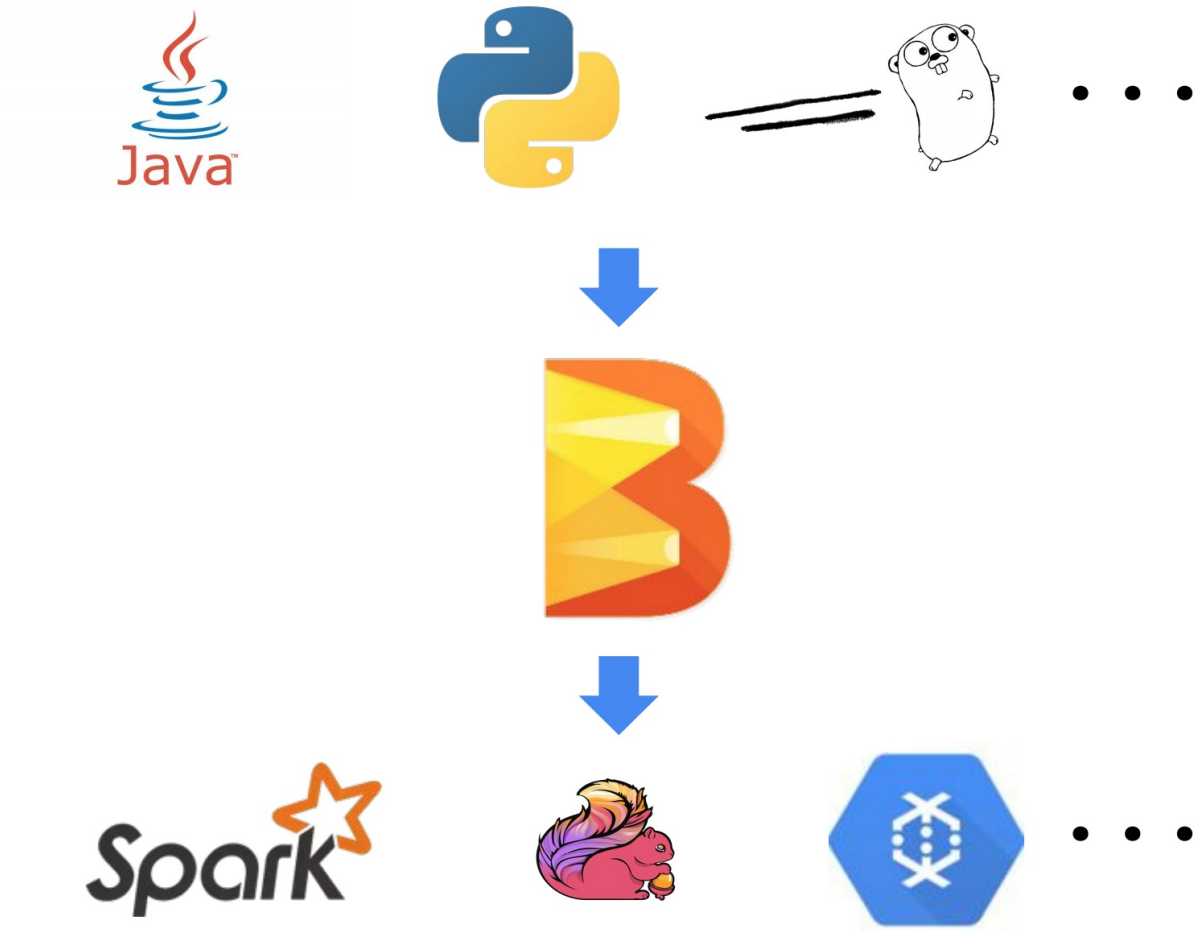
它能够将工程师写好的算法逻辑很好地与底层的运行环境分隔开。也就是说，当我们通过Beam提供的API写好数据处理逻辑后，这个逻辑可以不作任何修改，直接放到任何支持Beam API的底层系统上运行。

关于怎么理解这个优点，其实我们可以借鉴一下SQL（Structure Query Language）的运行模式。

我们在学习SQL语言的时候，基本上都是独立于底层数据库系统来学习的。而在我们写完一个分析数据的Query之后，只要底层数据库的Schema不变，这个Query是可以放在任何数据库系统上运行的，例如放在MySql上或者Oracle DB上。

同样的，我们用Beam API写好的数据处理逻辑无需改变，可以根据自身的需求，将逻辑放在Google Cloud Dataflow上跑，也可以放在Apache Flink上跑。在Beam上，这些底层运行的系统被称为Runner。现阶段Apache Beam支持的Runner有近十种，包括了我们很熟悉的Apache Spark和Apache Flink。

当然最后Apache Beam也是希望对自身的SDK能够支持任意多的语言来编写。现阶段Beam支持Java、Python和Golang。



也就是说，通过Apache Beam，最终我们可以用自己喜欢的编程语言，通过一套Beam Model统一地数据处理API，编写好符合自己应用场景的数据处理逻辑，放在自己喜欢的Runner上运行。

小结

今天，我与你一起回顾了Apache Beam的完整诞生历程。

通过这一讲，我希望你知道每一项技术都不会毫无理由地诞生，而每一项技术诞生的背后都是为了解决某些特定问题的。了解前人一步步解决问题的过程，有助于我们更有层次地理解一项技术产生的根本原因。在学习一项技术之前，先了解了它的历史源流，可以让我们做到知其然，并知其所以然。

思考题

你也能分享一些你所经历过的技术变迁或是技术诞生的故事吗？

欢迎你把答案写在留言区，与我和其他同学一起讨论。如果你觉得有所收获，也欢迎把文章分享给你的朋友。

精选留言：

- JohnT3e 2019-06-10 08:54:55
  文章中的几篇论文地址：
  0. MapReduce: https://research.google.com/archive/map reduce-osdi04.pdf
  1. Flumejava: https://research.google.com/pubs/archive/35650.pdf
  2. MillWheel: https://research.google.com/pubs/archive/41378.pdf
  3. Data flow Model: https://www.vldb.org/pvldb/vol8/p1792-Akidau.pdf

  个人认为还是应该读一读的，毕竟几十年的发展不能靠看一两篇文章就搞清楚的
  [10赞]

  作者回复2019-06-10 09:17:40
  👍