大规模数据处理的演化历程（2003 - 2018）

总结

* **MapReduce**

可扩展性和简单性。

通过在强大且可扩展的执行引擎之上提供一组简单的数据处理抽象，MapReduce让我们的数据工程师专注于他们的数据处理需求的业务逻辑，而不是去构建能够适应在一大堆普通商用服务器上的大规模分布式处理程序。

* **Hadoop**

开源生态系统。

通过构建一个关于MapReduce的开源平台，无意中创建了一个蓬勃发展的生态系统，其影响力所及的范围远远超出了其最初Hadoop的范围，每年有大量的创新性想法在Hadoop社区蓬勃发展。

* **Flume**

管道及优化。

通过将逻辑流水线操作的高级概念与智能优化器相结合，Flume可以编写简洁且可维护的Pipeline，其功能突破了MapReduce的“Map 🡪 Shuffle 🡪 Reduce”的限制，而不会牺牲性能。

* **Storm**

弱一致性，低延迟。

通过牺牲结果的正确性以减少延迟，Storm为大众带来了流计算，并开创了Lambda架构的时代，其中弱一致的流处理引擎与强大一致的批处理系统一起运行，以实现真正的业务目标低延迟，最终一致性的结果。

* **Spark**

强一致性。

通过利用强大一致的批处理引擎的重复运行来提供无界数据集的连续处理，Spark Streaming证明至少对于有序数据集的情况，可以同时具有正确性和低延迟结果。

* **MillWheel**

乱序处理。

通过将强一致性、精确一次处理与用于推测时间的工具（如水印和定时器）相结合，MillWheel做到了无序数据进行准确的流式处理。

* **Kafka**

持久化的流式存储，流和表对偶性。

通过将持久化数据日志的概念应用于流传输问题，Kafka支持了流式数据可重放功能。通过对流和表理论的概念进行推广，阐明数据处理的概念基础。

* **Cloud Dataflow**

统一批流处理引擎。

通过将MillWheel的无序流式处理与高阶抽象、自动优化的Flume相结合，Cloud Dataflow为批流数据处理提供了统一模型，并且灵活地平衡正确性、计算延迟、成本的关系。

* **Flink**

开源流处理创新者。

通过快速将无序流式数据处理的强大功能带到开源世界，并将其与分布式快照及保存点功能等自身创新相结合，Flink提高了开源流处理的业界标准并引领了当前流式处理创新趋势。

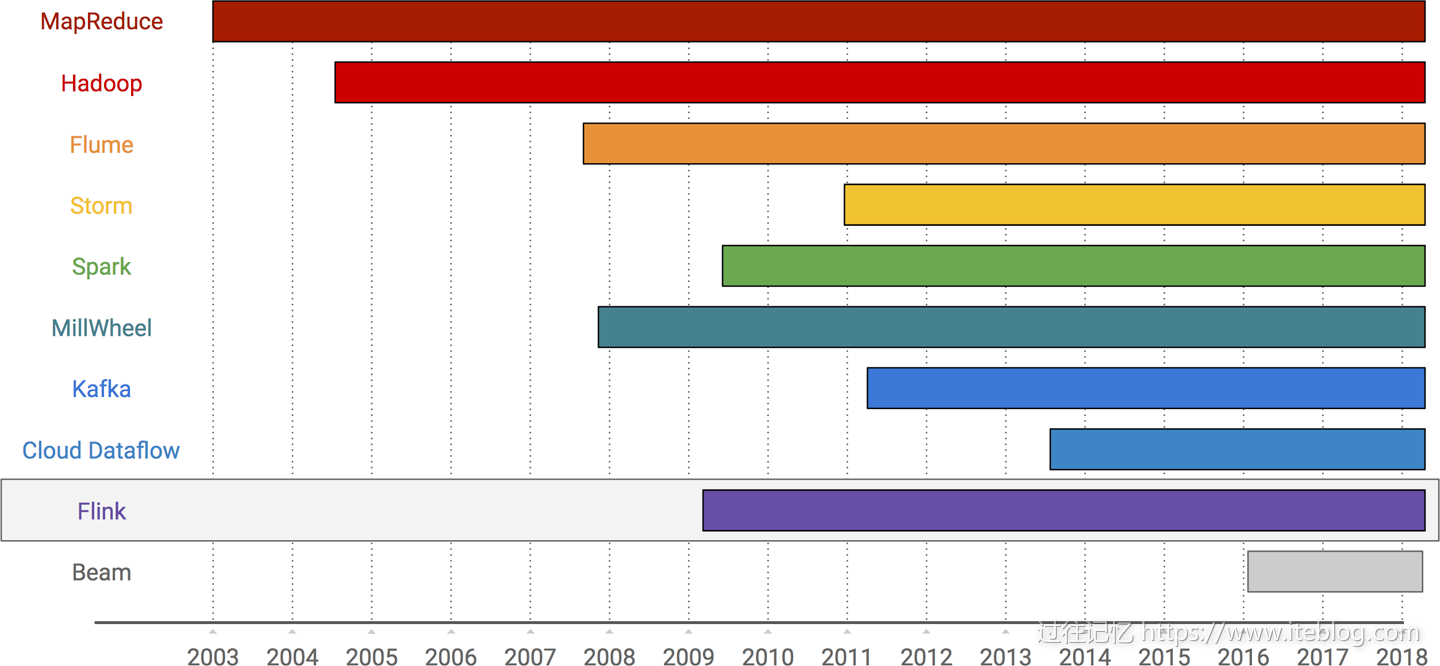
* **Beam**

可移植性。

通过提供整合行业最佳创意的强大抽象层，Beam提供了一个可移植API抽象，其定位为与SQL提供的声明性通用语言等效的程序接口，同时也鼓励在整个行业中推进创新。

可以肯定的说，这里强调的这10个项目及其成就的说明并没有超出当前大数据的历史发展。但是，它们是一系列重要且值得注意的大数据发展里程碑，它共同描绘了过去流处理演变的时间轴。自最早的MapReduce系统开始，尽管沿途有许多起伏波折，但不知不觉我们已经走出来很长一段征程。即便如此，在流式系统领域，未来我们仍然面临着一系列的问题亟待解决。正所谓：路漫漫其修远兮，吾将上下而求索。

Flink



Flink在2015年突然出现在大数据舞台，然后似乎在一夜之间从一个无人所知的系统迅速转变为人人皆知的流式处理引擎。

Flink崛起有两个主要原因：

* + 采用Dataflow/Beam编程模型，使其成为完备语义功能的开源流式处理系统。
  + 其高效的快照实现方式为其提供了正确性所需的强一致性保证。

Flink的一致性机制，其基本思想是在系统中的Worker之间沿着数据传播路径上产生周期性Barrier。这些Barrier充当了在不同Worker之间传输数据时的对齐机制。当一个Worker在其所有上游算子输入来源（即来自其所有上游一层的Worker）上接收到全部Barrier时，Worker会将当前所有key对应的状态写入一个持久化存储。这个过程意味着将这个Barrier之前的所有数据都做了持久化。通过调整Barrier的生成频率，可以间接调整Checkpoint的执行频率，从而降低时延并最终获取更高的吞吐（其原因是做Checkpoint过程中涉及到对外进行持久化数据，因此会有一定的IO导致延时）。

Flink既能够支持精确一次的语义处理保证，同时又能够提供支持事件时间的处理能力，这让Flink获取了巨大的成功。有人曾经发布过一组针对Flink性能的具体测试数据，总结起来有以下结论：

* + 构建一个用于测试的Flink数据管道，其拥有比Twitter Storm更高的准确性（归功于Flink的强一次性语义），但成本却降到了1%。
  + Flink在精确一次的处理语义参数设定下，仍然达到Storm的7.5倍吞吐量（而且，Storm还不具备精确一次的处理语义）。此外，由于网络被打满导致Flink的性能受到限制；进一步消除网络瓶颈后Flink的吞吐量几乎达到Storm的40倍。

从那时起，许多其他流式处理项目（特别是Storm和Apex）都采用了类似算法的数据处理一致性机制。

通过快照机制，Flink获得了端到端数据一致性。Flink更进了一步，利用其快照的全局特性，提供了从过去的任何一点重启整个管道的能力，这一功能称为SavePoint。Savepoints功能参考了Kafka应用于流式传输层的持久化和可重放特性，并将其扩展应用到整个底层Pipeline。流式处理仍然遗留大量开放性问题有待优化和提升，但Flink的Savepoints功能是朝着正确方向迈出的第一步，也是整个行业非常有特点的一步。

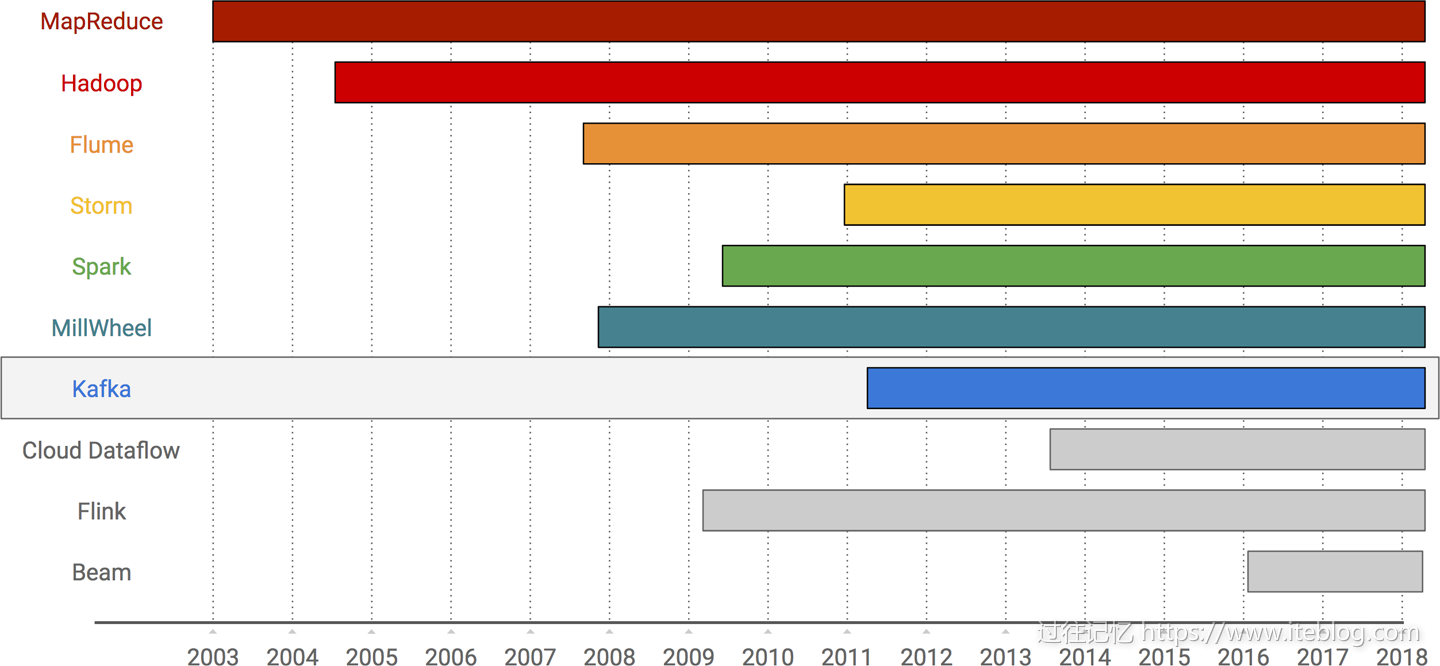
除了保存点之外，Flink社区还在不断创新，包括将第一个实用流式SQL API推向大规模分布式流处理引擎的领域。总之，Flink的迅速崛起成为流计算领军角色主要归功于三个特点：

* + 整合行业里面现有的最佳想法（例如，成为第一个开源DataFlow/Beam模型）
  + 创新性在表上做了大量优化，并将状态管理发挥更大价值，例如基于Snapshot的强一致性语义保证，Savepoints以及流式SQL。
  + 迅速且持续地推动上述需求落地。

另外，所有这些改进都是在开源社区中完成的，我们可以看到为什么Flink一直在不断提高整个行业的流计算处理标准。

Kafka

Kafka不是数据计算框架，而是数据传输和存储的工具。但是，毫无疑问，Kafka扮演了推动流处理的最有影响力的角色之一。



如果你不熟悉Kafka，我们可以简单描述为：Kafka本质上是一个持久的流式数据传输和存储工具，底层系统实现为一组带有分区结构的日志型存储。其卓越的特性有：

* 提供一个干净的持久性模型，让大家在流式处理领域里面可以享受到批处理的产品特性，例如持久化、可重放。
* 在生产者和消费者之间提供弹性隔离。
* 定义了流和表之间的关系，揭示了思考数据处理的基本方式，同时还提供了和数据库打通的思路和概念。
* 来自于上述所有方面的影响，不仅让Kafka成为整个行业中大多数流处理系统的基础，而且还促进了流处理数据库和微服务运动。

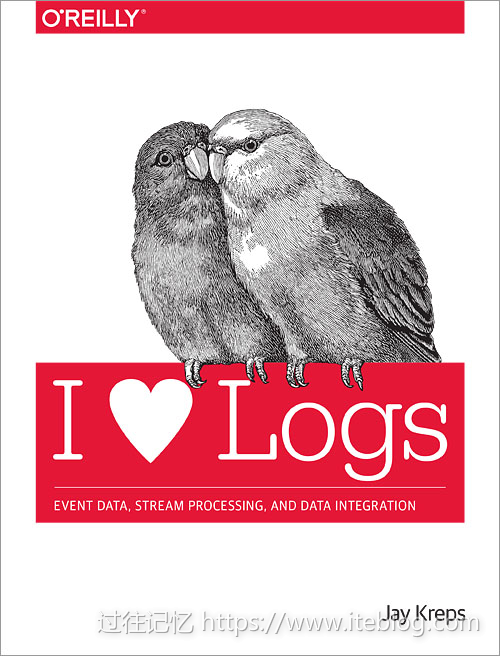
在这些特性中，有两个最为突出。

第一个是流数据的持久化和可重放性的应用。在Kafka之前，大多数流处理系统使用某种临时、短暂的消息系统，如Rabbit MQ甚至是普通的TCP套接字来发送数据。数据处理的一致性往往通过生产者数据冗余备份来实现（即，如果下游数据消费者出现故障，则上游生产者将数据进行重新发送），但是上游数据的备份通常也是临时保存一下。大多数系统设计完全忽略在开发和测试中需要重新拉取数据重新计算的需求。但Kafka的出现改变了这一切。从数据库持久日志概念得到启发并将其应用于流处理领域，Kafka让我们享受到了如同Batch数据源一样的安全性和可靠性。凭借持久化和可重放的特点，流计算在健壮性和可靠性上面又迈出关键的一步，为后续替代批处理系统打下基础。

作为一个流式系统开发人员，Kafka的持久化和可重放功能对业界产生一个更有意思的变化就是：当今大量流处理引擎依赖源头数据可重放来提供端到端精确一次的计算保障。可重放这个特点是Apex、Flink、Kafka Streams、Spark和Storm的端到端精确一次保证的基础。当以精确一次模式执行时，每个系统都假设 / 要求输入数据源能够重放之前的部分数据（从最近Checkpoint到故障发生时的数据）。当流式处理系统与不具备重放能力的输入源一起使用时（哪怕是源头数据能够保证可靠的一致性数据投递，但不能提供重放功能），这种情况下无法保证端到端的完全一次语义。这种对可重放（以及持久化等其他特点）的广泛依赖是Kafka在整个行业中产生巨大影响的间接证明。

Kafka系统中第二个值得注意的重点是流和表理论的普及。可以说流和表构成了数据处理的基础，无论是MapReduce及其演化系统、SQL数据库系统，还是其他分支的数据处理系统。并不是所有的数据处理方法都直接基于流或者表来进行抽象，但从概念或者理论上说，表和流的理论就是这些系统的运作方式。作为这些系统的用户和开发人员，理解我们所有系统构建的核心基础概念意义重大。

如果您想了解更多关于Kafka及其理论核心，JackKreps的《I love Logs》是一个很好的学习资料。



Kafka为流处理领域做出了巨大贡献，可以说比其他任何单一系统都要多。特别是，对输入和输出流的持久性和可重放的设计，帮助将流计算从近似工具的小众领域发展到在大数据领域妇孺皆知的程度起了很大作用。此外，Kafka社区推广的流和表理论对于数据处理引发了我们深入思考。