浅谈数据湖的过去，现状和未来

你说得对
大数据搬运工程师
已关注

南昌时光书城 等 24 人赞同了该回答

本文部分copy自我在公司的内部分享。

背景：

大数据自Google经典的三驾马车[+]论文发布以来，已经蓬勃发展了近20年，并在近年开始逐步走向成熟。伴随着实时计算的逐步普及，目前领域内下一个风口大概就是数据湖了。转向这个方向也有一段时间了，写本文的目的是给准备入坑的小伙伴介绍一些基本概念，另外想和同样做这个领域的同学交流一些看法。

大数据领域的发展历程：

想要了解一个概念，最关键的其实是明白这个概念解决了什么样的问题。由此才能明白其中的一些设计。在过去的大数据领域中，我们借助HDFS完成了分布式的文件存储，MapReduce解决了分布式的批式计算，Yarn解决了分布式计算的资源调度问题。Hbase解决了基本的近线的KV类大数据存储和访问。Hive及其生态完成了传统数据仓库从数据库到Hadoop生态的迁移。Flink以相对用户友好的方式收敛了实时数据计算。那么目前还有哪些问题没有被很好的解决呢?

1. HDFS不支持完整的增删改，不支持事务。虽然现有离线数仓的范式非常契合HDFS的immutable特性，但是这一特性带来了流程的冗余和成本升高，并进一步block了这种分布式文件存储的更多发展。
2. 实时计算和离线计算很难达成一致。由于实时计算所使用的存储和HDFS有特性上的区别，进而导致了架构，存储，建模等方方面面的不同，所以目前批流一体都处于非常割裂得状态，大家希望能用kappa[+]架构替换掉已有的lambda架构。
3. 大数据领域的元数据管理[+]正在面临可扩展性问题，伴随着数据复杂性提升，不论是NN还是hive，都遭遇越加严重的单点瓶颈问题。
4. 传统数仓建模范式只能针对结构数据，对于不能被建模为二维表的非结构化数据[+]，基本无能为力。
5. 对于非数仓的其他大数据领域，现有的体系仍未形成最佳实践。这里主要想说的是，机器学习。
6. 大数据的serving目前还没有比较完整的解决方案。大数据量，非预设逻辑，高并发，低延时的OLAP还不存在。

数据湖：

数据湖这个概念大概有10年的历史了，中间的探索方向几经周折。大约在4-5年前，数据湖的主流是想解决非结构化数据存储的问题，一顿折腾之后没了声息。今年来正在趋向收敛，有几个专门的Lake框架作为数据湖这个概念的实现出现，并做出了差异化的价值，获得了大家的初步青睐。在上述几个问题中，数据湖针对其中的1、3尝试给出解法，并努力想要收敛其中的2和5。对于4和6，其实是没有任何解决的迹象的。所以很多文章上来就说数据湖可以解决非结构化数据存储问题，非常误导人。

目前市面上比较火的数据湖框架有Iceberg,Hudi和DeltaLake。其中每个框架由于其诞生背景不同，优劣势各有侧重。在市场选型上，在选择使用开源框架的大型互联网公司中，iceberg和hudi基本是对半分的。DeltaLake相对比较少。其中，对CDC比较看重的一般会选hudi。对和hive和已有数仓体系兼容或者使用flink写入的，一般会选iceberg。deltaLake因为其背后商业公司运营方式的原因，一般比较少有大公司会参与投入。也因为其[illegible]

比较大的顾虑。另外就是因为立场原因，deltaLake不太可能会支持flink，与现在的实时计算生态冲突。

Hudi：由Uber开源的数据湖框架，主打流式的增删改查。

优点：

- 具备索引，支持高效的修改能力。流式写入效率高。
- 支持事务。
- 基于NN和HDFS文件来管理元数据。
- 支持主键、upsert等语义。
- 可以解决小文件问题。
- 支持增量读取。
- 社区活跃，迭代速度快

缺点：

- 表级别并发能力。尚无成熟特性支持更细粒度的并发。

Iceberg：由Netflix开源的数据湖框架，主打非中心化的二维表管理。将自己定位成表格式，是hive的超集。

优点：

- 基于HDFS文件来管理元数据。
- 可以解决小文件问题。
- 支持分区演进。
- 抽象好，计算引擎无关。对spark和flink支持都非常成熟。
- 对sql支持较完备。

缺点：

- 流式写入性能不太好
- update等语义支持还不算特别成熟

DeltaLake：由databricks开源的数据湖框架，主打事务和修改操作。

优点：

- 支持事务，支持细粒度事务。
- 可以解决小文件问题。
- 支持增量读取。
- 对spark sql支持完备。

缺点：

- 和spark强绑定，流式写入生态较差。
- databricks对于开源社区的运营方式比较难获得其他人的参与，社区活跃度不高。

数据湖已有的进展：

基于HDFS的增删改查：

由于HDFS本身不支持文件的修改（除了Append这种），所以所有的数据湖都基于这个特性加入了版本的概念。修改数据通过创建修改后的新版本来实现。其中有两种模式，COW（CopyOnWrite）和MOR（MergeOnRead），后者适合流式写性能较好，但是会有一些额外的成本，比如时效性没那么高或者读时成本高等。前者

大数据事务：

在使用大数据的时候是否遇到过以下问题呢？

- HDFS的原子操作粒度是文件，其他粒度一概不支持。
- 数仓修改原子操作粒度是文件夹。不支持细粒度的修改，改一行/一列就要重写整个分区
- 读写任务需要额外的通信机制，否则表粒度读写数据有并发问题

其本质是，HDFS这种存储，没有提供一个体系内闭环的事务能力，中间的方案都是基于不同需求的各种hack。各个数据湖框架实现事务的思路或多或少都有类似percolate的思路。将数据和元数据共同存放在HDFS上，并通过HDFS的文件原子性来完成元数据的提交。整个体系的原子性由元数据原子性来保证，不符合预期的数据会被元数据过滤，并借助某些机制最终被清除掉。

A: 为了实现原子性，三种数据湖框架其实都是借助或者部分借助了HDFS的文件原子语义，来完成元信息的原子提交，并借助元信息来完成数据的原子性可见。

C: 因为没有库的概念，没有多表级联。需要用户自己来对数据的修改进行判断，并借助同一个事务的原子性来保证。

I: 所有数据湖框架都提供了SNAPSHOT语义，实现snapshot isolation。

D: HDFS本身是一种具备持久化语义的存储。就算不是HDFS，也基本实现了HDFS的协议或者有类似保证。

唯一不同的是，不同的数据湖框架对事务冲突的解决能力不一样，DeltaLake是最好的，基于乐观锁实现了细粒度的冲突控制。hudi的锁粒度最粗，目前稳定版本是表锁，文件粒度锁是一个实验特性。iceberg和hudi差不多。

元数据可扩展和小文件问题：

小文件问题的本质其实是没法处理海量元数据的问题，所以被归类到同一个问题了。

首先是如何处理更多元数据，HDFS NN的可扩展性已经被诟病很久了，社区目前的解决方案基本都不能彻底解决问题。联邦/主从有一致性问题。C++重写迟早有一天还会有瓶颈的。Hive则是因为基于关系型数据库实现，导致了上限被单机mysql限制。业界目前有的做法是使用可扩展的关系型数据库来替换mysql，比如tidb。而现有数据湖的解法其实是把元数据作为文件存在HDFS上，通过HDFS本身的可扩展性解决元数据可扩展性。在这过程中，iceberg和deltaLake做得比较彻底，hudi目前还是半依赖HDFS NN的，正在推彻底迁移到HDFS文件上。但就我个人实践而言，HDFS存文件这个思路并不行。本质是文件这个东西不支持任何查询范式，处理起来非常麻烦不说，速度还很慢。未来是有可能限制数据湖的进一步应用的。目前我知道的该领域探索比较多的，基本都会采用某种可扩展的数据库来实现数据湖的元数据存储。开源社区不肯做的一个重要原因其实是不愿因引入其他依赖让框架变得太重，从而增加了用户接入的门槛。而公司内部则没有这个问题。

小文件问题则是另一个解决元数据问题的思路，那就是如何不制造过多元数据。两个思路，数据合并和小文件补偿。其中数据合并是hive年代就有的成熟解法，三个数据湖框架都有涉及。数据补偿这块据我所知只有hudi有，因为对文件的粒度控制比较强。

常见的数据湖场景的应用：

数据修改：

有时候已知会对数仓表做频繁/不那么频繁的大数据量的修改，为了避免严重的overwrite带来的成本。选择数据湖可以一定程度上减少用户的修改成本。目前最起码能解决的用户的使用和理解成本。借助合理的索引结构还可以减少读写放大（parquet本身的索引/hudi的索引）。

CDC数据导入：

目前现有的数仓架构，ODS层基于批式的数据传输服务。延迟较高，对在线影响较大。最典型的就是mysql 0点开始扫库进hive。使用CDC数据接入后，可以实现较低成本的，近实时的CDC数据流接入数仓，提高每天报表产出时间，减少在线存储的压力。如果选择了支持增量处理的数据湖+框架，还可以以较低的读成本完成一些本来的操作。

数据湖的发展：

与现有数仓建模+语义冲突：

之前有提到，数仓建模语义和HDFS特别契合，但是和数据湖其实是不太契合的。因为数仓的ETL pipeline都是基于表不变的基础上来设计的。比如天级全量表这个概念就没法用数据湖来表达。只能每天做checkPoint，目前是没有很好的实践方式的，哪怕是支持savePoint的hudi。另外数据湖的增量语义看似是低成本获取了新增数据，但是由于目前dws/ads这种汇总层是不像flink state支持增量语义的，并不能很好得利用增量获取。如果想实现LakeHouse的目标，让数据湖全面和数仓体系结合，而不仅仅是加速一个ODS层+的生成。基本需要推翻现有的数仓建模范式，重新来一套，让数据湖接管所有hive表+，并在计算引擎层抽象出更好的增量处理语义。

实时离线架构：

lambda+引起的链路冗余，存储冗余，实时离线数据一致性问题目前已经比较迫切了。

这种架构的出现，本质其实是实时/离线链路的存储特性不一致。flink的state是一种有主键的存储，但是通常只能存状态，不能支持特别大数据量的存储。kafka没有主键，不能获取全量数据，而且不支持exactly-once。这俩加起来和提供list语义，可以获取全部数据的hive还是很不一样。这也是为啥目前实时数仓建模的时候一般都和离线数仓不太一样，层会比较少。太多层会增加延时固然是一个重要原因。另一个就是一旦制造很多层，就得尝试加状态来弥补kafka+带来的问题，不管是资源成本还是建模中的理解成本都会迅速升高。Flink提供的批流一体语义只能在计算上弥补两者的语义差距，并不能解决存储的问题。

后来提出kappa架构本质上就是想解决这个问题，不管这个kappa架构是拿什么做实现的，都是希望把离线和实时的数据存储统一到同一种语义的存储上面来，这样来抹平数据在不同语义存储上带来的差异。目前来看比较有希望的分别是mq和数据湖。最近以来mq领域一直在一致性语义，存储计算分离上大力推进。前者和state结合可以替换不支持增删改和事务的hive，后者则可以提供全量数据的存储。

除此之外来看，数据湖同样有成为这一存储的潜力。相比hive而言，对于流式写入更加友好，能够支持类似state的增删改查语义，支持事务一致性语义。数据湖本身已经可以完成全量数据的存储，在流式读语义上比mq还有差距。

谁能先追上这个差距，就有较大概率成为下一代实时离线统一架构中的存储标准。

性能：

因为引入了新的架构设计的缘故，其实目前数据湖在各方面性能上是没有办法和原有技术方案持平的，只是胜在语义多样性上。在单纯写入上不如hive，在流式性能上不如mq。想追上hive，就要继续减少流程中的overhead，把为了事务/主键语义引入的一些额外的流程的成本追回来。想追上mq，就要加强对底层存储特性的利用，列存追上行存本来就难，中间还隔了HDFS。任重而道远。

机器学习的应用：

没有空了，回头再说吧。

编辑于 2022-06-12 11:59

内容所属专栏

DMD505

订阅专栏

大数据 数据湖 LakeHouse

理性发言，友善互动

6 条评论

默认 最新

车VS可

高质量文章啊！

2023-12-22 回复 喜欢

阿拉丁

楼主是少有讲出大数据目前现状的文章，其他都是各种吹。这样的好文章应该多些热度

2023-02-28 回复 喜欢

Amicus VERITAS

数据湖，是为了适应各种应用对数据的要求，就是把各种数据，包括图片、音视频、文本、结构化数据这些汇总到一个地方，进行增删改查等操作，从而实现数据高效管理。是这意思吗

2023-02-21 回复 喜欢

你说得对 作者

我觉得不是。文章里有讲数据湖解决不了非结构化的问题

2023-02-22 回复 喜欢

你说得对 作者 ▸ **Amicus VERITAS**

要说管理必然要有一整套用户便捷的，有体系的，反应数据联系的写入储存和查询范式。不然的话，按目录存在oss上也叫管理。就没有非结构化的问题了。

2023-02-22 回复 喜欢

展开其他 1 条回复 〉

推荐阅读

数据湖搭建指南——几个核心问题

大数据流动

什么是数据湖？及其架构

Jacen... 发表于Jacen...

从数据仓库到数据湖(湖导论

Light Gao