阿里云云原生数据湖体系全解读

首次全方位拆解全新的数据湖架构和产品组合

阿里云首次发布云原生数据湖体系，基于对象存储 OSS 、数据湖构建 Data Lake Formation 和 E-MapReduce 产品的强强组合，提供存储与计算分离架构下，涵盖湖存储、湖加速、湖管理和湖计算的企业级数据湖解决方案。

藏经阁

阿里云开发者电子书系列

加入数据湖钉群
进行更多技术交流

阿里云开发者“藏经阁”
海量免费电子书下载

目录

阿里云重磅发布云原生数据湖体系

作者：无谓、铁杰、周皓、亦龙、扬清

“数据湖”正在被越来越多人提起，尽管定义并不统一，但企业们都已纷纷下水实践，无论是 AWS 还是阿里云、华为。

我们认为：数据湖是大数据和 AI 时代融合存储和计算的全新体系。

为什么这么说？还要从它的发展说起。

数据量爆发式增长的今天，数字化转型成为 IT 行业的热点，数据需要更深度的价值挖掘，因此需要确保数据中保留的原始信息不丢失，应对未来不断变化的需求。

当前以 Oracle 为代表的数据库中间件已经逐渐无法适应这样的需求，于是业界也不断地产生新的计算引擎，以便应对大数据时代的到来。

企业开始纷纷自建开源 Hadoop 数据湖架构，原始数据统一存放在 HDFS 系统上，引擎以 Hadoop 和 Spark 开源生态为主，存储和计算一体。

缺点是需要企业自己运维和管理整套集群，成本高且集群稳定性较差。

在这种情况下，云上托管 Hadoop 数据湖架构（即 EMR 开源数据湖）应运而生。底层物理服务器和开源软件版本由云厂商提供和管理，数据仍统一存放在 HDFS 系统上，引擎以 Hadoop 和 Spark 开源生态为主。

这个架构通过云上 IaaS 层提升了机器层面的弹性和稳定性，使企业的整体运维成本有所下降，但企业仍然需要对 HDFS 系统以及服务运行状态进行管理和治理，即应用层的运维工作。

因为存储和计算耦合在一起，稳定性不是最优，两种资源无法独立扩展，使用成本也不是最优。

同时，受到开源软件本身能力的限制，传统数据湖技术无法满足企业用户在数据规模、存储成本、查询性能以及弹性计算架构升级等方面的需求，也无法达到数据湖架构的理想目标。

企业在这个时期需要更低廉的数据存储成本、更精细的数据资产管理、可共享的数据湖元数据、更实时的数据更新频率以及更强大的数据接入工具。

云原生时代到来，我们可以有效利用公有云的基础设施，数据湖平台也有了更多的技术选择。比如云上纯托管的存储系统逐步取代 HDFS，成为数据湖的存储基础设施，并且引擎丰富度也不断扩展。

除了 Hadoop 和 Spark 的生态引擎之外，各云厂商还发展出面向数据湖的引擎产品。如分析类的数据湖引擎有 AWS Athena 和华为 DLI，AI 类的有 AWS Sagemaker。

这个架构仍然保持了一个存储和多个引擎的特性，所以统一元数据服务至关重要。

基于此，阿里云正式发布了云原生数据湖体系，由对象存储 OSS、数据湖构建 Data Lake Formation、E-MapReduce 产品强强组合，提供存储与计算分离架构下，湖存储、湖加速、湖管理、湖计算的企业级数据湖解决方案。

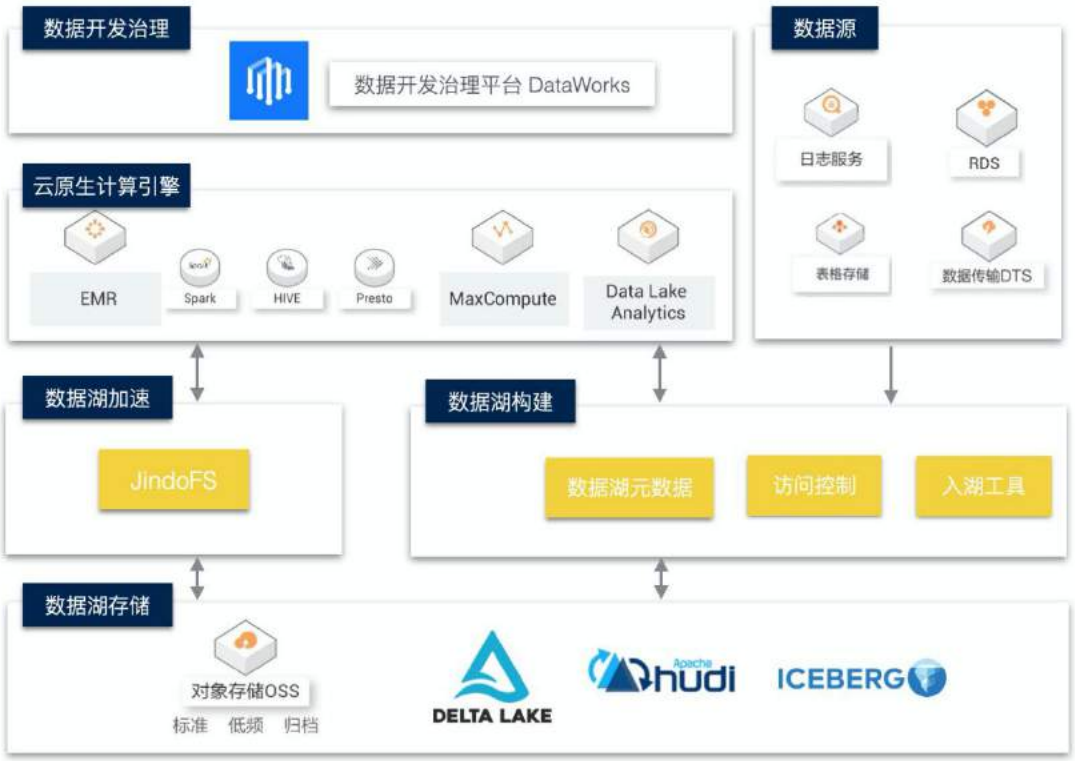
- 数据湖存储用云上的对象存储 OSS 加上 JindoFS 取代 HDFS，提升数据规模、降低存储成本、实现计算和存储分离架构；
- 数据湖构建（DLF）服务提供统一元数据和统一的权限管理，支持多套引擎接入；
- EMR 上 Spark 等计算引擎的云原生化，可以更好的利用弹性计算资源；
- 云上的数据开发治理平台 DataWorks 解决了数据湖元数据治理、数据集成、数据开发等问题。

数据是最好的佐证：阿里云云原生数据湖体系可支持 EB 级别的数据湖，存储超过 10 万 Database、1 亿 Table 以及 10 亿级别的 Partition，每天支持超过 30 亿次的元数据服务请求，支持超过 10 个开源计算引擎以及 MaxCompute 和 Hologres 等云原生数仓引擎。

同时，阿里云数据湖存储成本相对于高效云盘下降 10 倍以上，查询性能相对于传统对象存储提速 3 倍以上，并且查询引擎有着极高的弹性，能在 30 秒内启动超过 1000 个 Spark Executor。由此可见，阿里云强大的存储和计算能力共同打造了业界领先的数据湖体系。

这些背后都在告诉我们：想在大数据时代占据先机，你需要有一套系统，能够在保留数据的原始信息情况下，又能快速对接多种不同的计算平台。

在此之际，我们推出云原生数据湖技术系列专题，将告诉大家如何基于阿里云 OSS、JindoFS 和数据湖构建（DataLakeFormation，DLF）等基础服务，结合阿里云上丰富的计算引擎，打造一个全新云原生数据湖体系。



数据湖存储 OSS

阿里云对象存储 OSS 是数据湖的统一存储层，它基于 12 个 9 的可靠性设计，可存储任意规模的数据，可对接业务应用、各类计算分析平台，非常适合企业基于 OSS 构建数据湖。相对于 HDFS 来说，OSS 可以存储海量小文件，并且通过冷热分层、高密度存储、高压缩率算法等先进技术极大降低单位存储成本。同时 OSS 对 Hadoop 生态友好，且无

缝对接阿里云各计算平台。针对数据分析场景，OSS 推出 OSS Select、Shallow Copy 和多版本等功能，加速数据处理速度，增强数据一致性能力。

数据湖加速

对象存储系统在架构设计上和 HDFS 等分布式文件系统存在一定差异，同时存储和计算分离架构中 OSS 是远端的存储服务，在大数据计算层面缺少对数据本地化的支持。因此，在 OSS 对象存储服务的基础上，阿里云定制了自研的大数据存储服务 —— JindoFS，极大的提升数据湖上的引擎分析性能，在 TPC-DS、Terasort 等常见的 benchmark 测试中，采用计算存储分离架构的 JindoFS 性能已经达到或超过了本地部署的 HDFS。同时 JindoFS 完全兼容 Hadoop 文件系统接口，给客户带来更加灵活、高效的计算存储方案，目前已验证支持 Hadoop 开源生态中最主流的计算服务和引擎：Spark、Flink、Hive、MapReduce、Presto、Impala 等。当前 JindoFS 存储服务包含在阿里云 EMR 产品中，未来 JindoFS 会有更多的产品形态服务于数据湖加速场景。

数据湖构建（DLF）

传统的数据湖架构非常强调数据的统一存储，但对数据的 Schema 管理缺乏必要的手段和工具，需要上层分析和计算引擎各自维护元数据，并且对数据的访问没有统一的权限管理，无法满足企业级用户的需求。数据湖构建（DLF）服务是阿里云在 2020 年 9 月推出的针对数据湖场景的核心产品，主要为了解决构建数据湖过程中用户对数据资产的管理需求。DLF 对 OSS 中存储的数据提供统一的元数据视图和统一的权限管理，并提供实时数据入湖和清洗模板，为上层的数据分析引擎提供生产级别的元数据服务。

云原生计算引擎

当前阿里云上众多云原生计算引擎已经接入或准备接入数据湖构建服务，包括阿里云 EMR 上的开源计算引擎 Spark、Hive、Presto、Flink 以及大数据计算服务 MaxCompute、数据洞察 Databricks 引擎和数据湖分析（DLA）等。以最常用的开源引擎 Spark 为例，阿里云 Spark 可以直接对接数据湖构建的元数据服务，运行在多集群或多平台上的阿里云 Spark 任务共享同一个数据湖元数据视图。并且 EMR 为 Spark 推出了 Shuffle Service 服务，Spark 引擎因此获得云原生平台上的弹性扩缩容能力。云原生计算引擎结合数据湖架构可以获得更高的灵活度并极大的降低数据分析成本。

另外，云原生数据仓库 MaxCompute 也准备接入数据湖构建服务，未来数仓和数据湖将会发生什么样的化学反应呢？敬请期待。

数据湖治理

DataWorks 数据综合治理可为阿里云客户提供统一的数据视图，用于掌握数据资产的现状、助力数据质量的提升、提高获取数据的效率、保障数据安全的合规并提升数据查询的分析效率。可以有效支撑离线大数据仓库的构建、数据联邦的查询和分析处理、海量数据的低频交互式查询和智能报表的构建，以及数据湖方案的实现。

综上所述，利用阿里云的基础组件和整体解决方案，用户可以方便的构建一个数据湖平台，完成企业大数据架构转型。

数据湖存储 OSS

基于 OSS 的 EB 级数据湖

作者：旭一

背景

随着数据量的爆发式增长，数字化转型成为整个 IT 行业的热点，数据也开始需要更深度的价值挖掘，因此需要确保数据中保留的原始信息不丢失，从而应对未来不断变化的需求。当前以 oracle 为代表的数据库中间件已经逐渐无法适应这样的需求，于是业界也不断的产生新的计算引擎，以便应对数据时代的到来。在此背景下，数据湖的概念被越来越多的人提起，希望能有一套系统在保留数据的原始信息情况下，又能快速对接多种不同的计算平台，从而在数据时代占的先机。

概述

什么是数据湖

数据湖（Data Lake）以集中式存储各种类型数据，包括：结构化、半结构化、非结构化数据。数据湖无需事先定义 Schema，数据可以按照原始形态直接存储，覆盖多种类型的数据输入源。数据湖无缝对接多种计算分析平台，对 Hadoop 生态支持良好，存储在数据湖中的数据可以直接对其进行数据分析、处理、查询，通过对数据深入挖掘与分析，洞察数据中蕴含的价值。

	数仓	数据湖
Data(数据类型)	结构化数据(Processed)	结构化、半结构化、非结构化数据(Raw)
Schema	schema-on-write	schema-on-read
性价比	Fastest query results using higher cost storage	Query results getting faster using cost-effective storage
Users	Business Analysts(BA)	Data Scientists, BA, Dev
分析场景	Batch reporting, BI and visualizations	Ad, Predictive analytics, data discovery and profiling, Machine Learning, etc.

数据湖的关键特征与价值

- **海量数据存储：**面向海量数据存储设计，完全独立于计算框架之外，无需额外的挂载操作，数据可直接访问，具备极大的灵活性和弹性能力，足以应对数据爆炸式发展，同时支持多层冗余能力，实现数据高可靠与高可用。

- **高效数据计算：**丰富的数据存储类型和共享能力，支持存储结构化、半结构化、非结构化数据，同时可以适配多种不同的计算平台，避免数据孤岛与无效的数据拷贝。

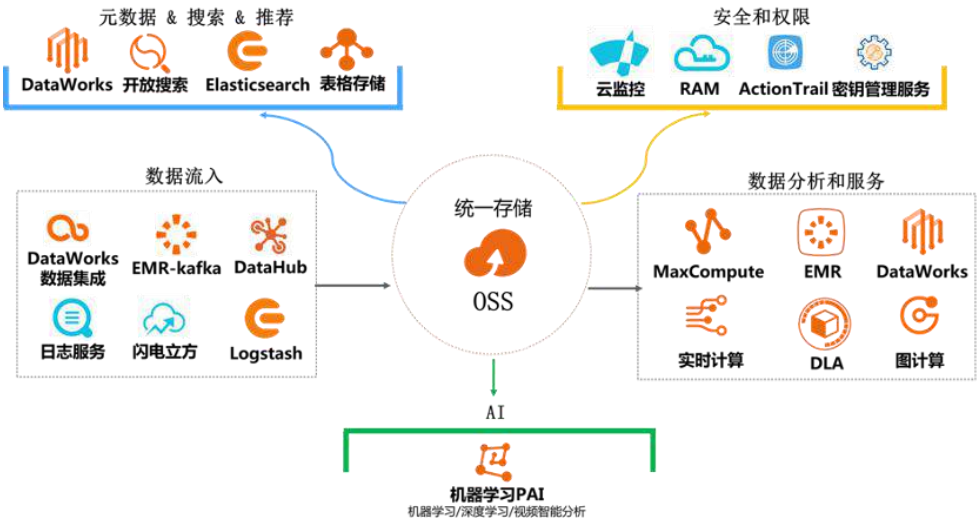
- **安全数据管理：**支持数据目录功能，智能化的管理海量的数据资产，通过精细化权限控制保障数据安全。

基于 OSS 的数据湖存储

OSS 介绍

阿里云对象存储 OSS（Object Storage Service）是阿里云提供的海量、安全、低成本、高可靠的云存储服务。其数据设计持久性不低于 99.9999999999%（12 个 9），服务可用性（或业务连续性）不低于 99.995%。OSS 具有与平台无关的 RESTful API 接口，您可以在任何应用、任何时间、任何地点存储和访问任意类型的数据。

基于 OSS 构建数据湖存储



OSS 在作为数据湖存储，充分满足数据湖的关键特性：

海量数据存储：

- OSS 采用分布式系统架构，扁平命名空间设计，支持无限制的存储规模，并且性能和容量可以随着系统扩展线性提升。

- OSS 支持弹性扩容，容量自动扩展，不限制存储空间大小，用户可以根据所需存储量无限扩展存储空间，并只按照实际使用量收取费用，无需客户自己提前配置。

- OSS 支持数据高可用

  - 在同一地域内（region）采用多可用区（AZ）冗余机制以及跨地域的复制机制，避免单点故障导致数据丢失或无法访问；
  - 支持数据周期性校验，避免静默数据损坏；
  - 支持 Object 操作强一致性，写入 Object 的数据在返回成功响应后，立即可读；
  - 支持多版本能力，防止数据误删。整体 OSS 满足 12 个 9 的数据持久性以及 995% 的服务可用性。

高效数据计算：

- OSS 提供 RESTFul API，具有互联网可访问能力，用户可以随时随地立即存储或者访问数据，无需提前进行映射和挂载操作。

- OSS 兼容开源 Hadoop 生态，且无缝对接阿里云多种不同的计算平台，使得数据无需拷贝即可被计算平台共享使用。同时针对部分计算平台优化特定操作，从而提升数据处理性能。

- OSS 支持算子卸载能力，目前提供了 Select 语句支持，可以让用户从单个文件中仅读取需要的数据，从而提升数据获取效率。

安全数据管理：

- OSS 支持数据生命周期管理，用户可以通过设置生命周期规则，将符合规则的数据自动删除或者转储到更低成本的存储中。

- OSS 支持客户端和服务端两种数据加密能力，用户可以根据自身情况灵活选择加密方案，避免数据泄露。

- OSS 通过 WORM（Write Once Read Many）特性，支持数据保留合规，允许用户以“不可删除、不可篡改”方式保存和使用数据，符合美国证券交易委员会（SEC）和金融业监管局（FINRA）的合规要求（OSS 已获得对应的合规认证）。

- OSS 支持多种数据访问安全控制策略，实现针对 bucket、object、role 的长期或者临时授权，从而满足最小权限数据共享的安全策略。

总结

综合以上内容，在未来面向海量数据的数据湖场景下，对象存储 OSS 非常适合企业构建海量、高效、安全的数据湖。

数据湖加速

基于 JindoFS+OSS 构建高效数据湖

作者：诚历

为什么要构建数据湖

大数据时代早期，Apache HDFS 是构建具有海量存储能力数据仓库的首选方案。随着云计算、大数据、AI 等技术的发展，所有云厂商都在不断完善自家的对象存储，来更好地适配 Apache Hadoop/Spark 大数据以及各种 AI 生态。由于对象存储有海量、安全、低成本、高可靠、易集成等优势，各种 IoT 设备、网站数据都把各种形式的原始文件存储在对象存储上，利用对象存储增强和拓展大数据 AI 也成为了业界共识，Apache Hadoop 社区也推出了原生的对象存储“Ozone”。从 HDFS 到对象存储，从数据仓库到数据湖，把所有的数据都放在一个统一的存储中，也可以更加高效地进行分析和处理。

对于云上的客户来说，如何构建自己的数据湖，早期的技术选型非常重要，随着数据量的不断增加，后续进行架构升级和数据迁移的成本也会增加。在云上使用 HDFS 构建大规模存储系统，已经暴露出来不少问题。HDFS 是 Hadoop 原生的存储系统，经过 10 年来的发展，HDFS 已经成为大数据生态的存储标准，但我们也看到 HDFS 虽然不断优化，但是 NameNode 单点瓶颈，JVM 瓶颈仍然影响着集群的扩展，从 1 PB 到 100+ PB，需要不断的进行调优、集群拆分来，HDFS 可以支持到 EB 级别，但是投入很高的运维成本，来解决慢启动，心跳风暴，节点扩容、节点迁移，数据平衡等问题。

云原生的大数据存储方案，基于阿里云 OSS 构建数据湖是最合适的选择。OSS 是阿里云上的对象存储服务，有着高性能、无限容量、高安全、高可用、低成本等优势，JindoFS 针对大数据生态对 OSS 进行了适配，缓存加速，甚至提供专门的文件元数据服务，满足上云客户的各种分析计算需求。**因此在阿里云上，JindoFS + OSS 成为客户采取数据湖架构迁移上云的最佳实践。**

JindoFS 介绍

Jindo 是阿里云基于 Apache Spark / Apache Hadoop 在云上定制的分布式计

算和存储引擎。Jindo 原是阿里云 开源大数据团队的内部研发代号，取自筋斗(云)的谐音，Jindo 在开源基础上做了大量优化和扩展，深度集成和连接了众多阿里云基础服务。

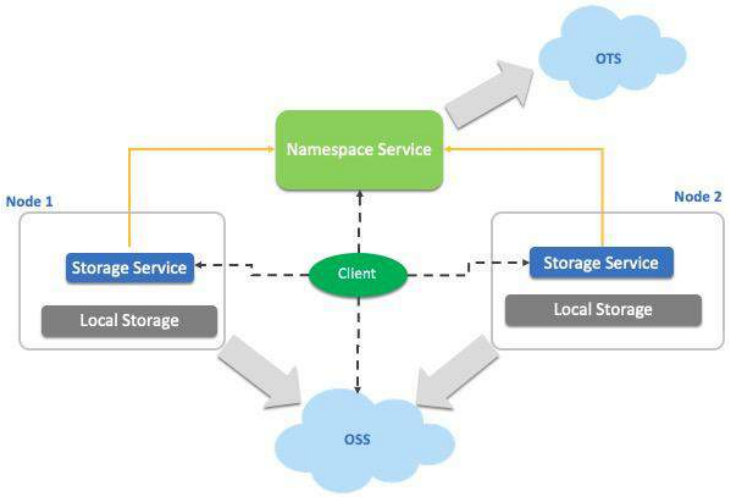
JindoFS 是阿里云针对云上存储自研的大数据缓存加速服务，JindoFS 的设计理念是云原生：弹性、高效、稳定和低成本。JindoFS 完全兼容 Hadoop 文件系统接口，给客户带来更加灵活、高效的数据湖加速方案，完全兼容阿里云 EMR 中所有的计算服务和引擎：Spark、Flink、Hive、MapReduce、Presto、Impala 等。JindoFS 有两种使用模式，块存储模式(BLOCK)和缓存模式(CACHE)。下面我们介绍下如何在 EMR 中配置和使用 JindoFS 以及不同模式对应的场景。

JindoFS 架构

JindoFS 主要包含两个服务组件：元数据服务(NamespaceService) 和存储服务(StorageService)：

- NamespaceService 主要负责元数据管理以及管理 StorageService。
- StorageService 主要负责管理节点的本地数据和 OSS 上的缓存数据。

下图是 JindoFS 架构图：元数据服务 NamespaceService 部署在独立的节点，对于生产环境推荐部署三台(Raft)来实现服务高可用；存储服务 StorageService 部署在集群的计算节点，管理节点上的闲置存储资源（本地盘/SSD/内存等），为 JindoFS 提供分布式缓存能力。



JindoFS 元数据服务

JindoFS 的元数据服务叫 JindoNamespaceService，内部基于 K-V 结构存储元数据，相对于传统的内存结构有着操作高效，易管理，易恢复等优势。

高效元数据操作

JindoFS NamespaceService 基于内存 + 磁盘管理和存储元数据，但是性能上比使用内存的 HDFS NameNode 还要好，一方面是 JindoFS 使用 C++ 开发，没有 GC 等问题，响应更快；另一方面是由于 Namespace Service 内部有更好的设计，比如文件元数据上更细粒度的锁，更高效的数据块副本管理机制。

秒级启动

有大规模 HDFS 集群维护经验的同学比较清楚，当 HDFS 元数据存储量过亿以后，NameNode 启动初始化要先加载 Fsimage ，再合并 edit log，然后等待全部 DataNode 上报 Block，这一系列流程完成要花费一个小时甚至更长的时间， 由于 NameNode 是双机高可用（Active/Standby），如果 standby 节点重启时 active 节点出现异常 ，或两台 NameNode 节点同时出现故障，HDFS 将出现停服一小时以上的损失。JindoFS 的元数据服务基于 Raft 实现高可用，支持 2N+1 的部署方式，允许同时挂掉 N 台；元数据服务（NamespaceService）在元数据内部存储上进行了设计和优化，进程启动后即可提供服务，可以做到了快速响应。由于 NamespaceService 近实时写入 OTS 的特点，元数据节点更换，甚至集群整体迁移也非常容易。

低资源消耗

HDFS NameNode 采用内存形式来存储文件元数据。在一定规模下，这种做法性能上是比较不错的，但是这样的做法也使 HDFS 元数据的规模受限于节点的内存，经过推算，1 亿文件 HDFS 文件大约需要分配 60 GB Java Heap 给 NameNode，所以一台 256 GB 的机器最多可以管理 4 亿左右的元数据，同时还需要不断调优 JVM GC。JindoFS 的元数据采用 Rocksdb 存储元数据，可以轻松支持到 10 亿规模，对于节点的内存需求也非常小，资源开销不到 NameNode 的十分之一。

JindoFS 缓存服务

JindoFS 的数据缓存服务叫 JindoStorageService，本地 StorageService 主要提供高性能缓存加速，所以运维上可以基于这样的设定大大简化。

弹性运维

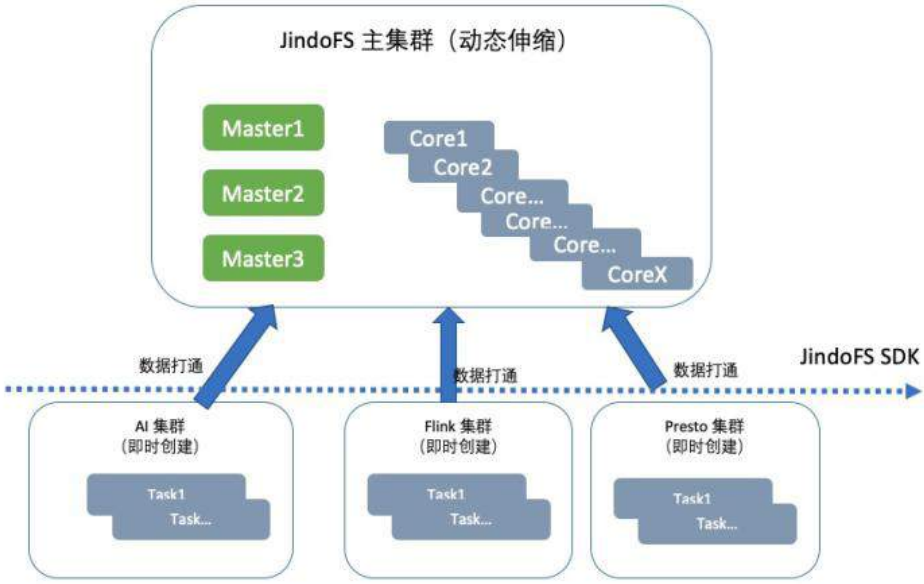
HDFS 使用 DataNode 在存储节点上来管理节点存储，全部数据块都存储在节点的磁盘上，依靠 DataNode 定期检查和心跳把存储状态上报给 NameNode，NameNode 通过汇总和计算，动态地保证文件的数据块达到设定的副本数（一般 3 副本）。对于大规模集群（节点 1000+），经常需要进行集群节点扩容，节点迁移，节点下线，节点数据平衡这样的操作，大量的数据块的副本计算增加了 NameNode 负载，同时，节点相关操作要等待 NameNode 内部的副本调度完成才能进行，通常一个存储节点的下线需要小时级别的等待才能完成。JindoFS 使用 StorageService 来管理节点上的存储，由于 JindoFS 保证了数据在 OSS 上有一副本，所以本地的副本主要用来进行缓存加速。对于节点迁移、节点下线等场景，JindoFS 无需复杂副本计算，通过快速的“标记”即可完成下线。

高性能存储

StorageService 采用 C++ 语言开发，在对接最新高性能存储硬件上也有着天然优势。StorageService 的存储后端不仅可以同时对接 SSD、本磁盘、OSS 满足 Hadoop/Spark 大数据框架各种海量、高性能的存储访问需求，可以对接内存、AEP 这样的高性能设备满足 AI/机器学习的低延时、高吞吐的存储使用需求。

JindoFS 适用场景

JindoFS 的元数据存储在 Master 节点的 NamespaceService（高可用部署）上，性能和体验上对标 HDFS；Core 节点的 StorageService 将一份数据块存储在 OSS 上，本地数据块可以随着节点资源进行快速的弹性伸缩。多集群之间也可以相互打通。



为了支持数据湖多种使用场景，一套 JindoFS 部署同时提供两种 OSS 使用方式，存储模式（Block）和缓存模式（Cache）。

缓存模式

对于已经存在于 OSS 上的数据，可以使用缓存模式访问，正如“缓存”本身的含义，通过缓存的方式，在本地集群基于 JindoFS 的存储能力构建了一个分布式缓存服务，把远端数据缓存在本地集群，使远端数据“本地化”。使用上也沿用原来的路径访问，如 oss://bucket1/file1 ，这种模式全量的文件都在 OSS 上面，可以做到集群级别的弹性使用。

存储模式

存储模式（Block）适用于高性能数据处理场景，元数据存储在 NamespaceService（支持高可用部署）上，性能和体验上对标 HDFS；StorageService 将一份数据块存储在 OSS 上，本地数据块可以随着节点资源可以进行快速的弹性伸缩。基于 JindoFS Block 模式这样的特性，可以用作构建高性能数仓的核心存储，多个计算集群可以访问 JindoFS 主集群的数据。

JindoFS 方案优势

基于 JindoFS+OSS 来构建数据湖相比于其他数据湖方案同时具有性能和成本优势。

性能上

JindoFS 针对一些常用的场景和 Benchmark 进行了对比测试，如 DFSIO、NNbench、TPCDS、Spark、Presto 等，通过测试我们可以看到性能上，Block 模式完全领先于 HDFS，Cache 模式完全领先于 Hadoop 社区的 OSS SDK 实现，由于篇幅的原因，后续我们会发布详细的测试报告。

成本上

成本是也是用户上云的重要考量，JindoFS 的成本优势主要体现在运维成本和存储成本两方面。运维成本指的是集群日常维护，节点上下线、迁移等。如前面分析，当 HDFS 集群增长到一定规模时，比如 10PB+，除了对 HDFS 进行专家级别调优外，还需要业务上的拆分规划，避免达到 HDFS 元数据上的瓶颈。同时，随着集群数据不断增长，一些节点和磁盘也会出现故障，需要进行节点下线和数据平衡，也给大集群的运维带来一定的复杂度。JindoFS 可以使用 OSS + OTS 的存储模式，OSS 上保留原始文件和数据块备份，对节点和磁盘出现的问题可以更好兼容；元数据（NamespaceService）采用 C++ 开发加上工程打磨，相比 NameNode + JVM 在容量上和性能上也更有优势。

下面我们重点来看存储成本。存储成本指的是存放数据后产生的存储费用，使用 OSS 是按量付费的，相比基于本地盘创建的 HDFS 集群有更好的成本优势，下面来计算和对比一下二者成本：

基于 HDFS + 本地盘方案构建大数据存储：

由于本地盘机型为整体价格，需要如下进行换算，预估存储成本如下：

实例规格	按月标价(元)	CPU/内存成本换算价格(元)	存储成本 换算价格（元）
ecs.d1ne.14xlarge（56C/224G/154TB）	13475	6720	6755

（参考链接：https://www.aliyun.com/price/product#/ecs/detail ）

考虑到实际使用 HDFS 会有 3 副本以及一定的预留空间，我们以 HDFS 3 副本、80% 使用率进行成本计算：

$$\frac{\text{6755 元/月}}{\text{154TB * 80\% / 3}} = \text{0.16 元/GB/每月}$$

基于 JindoFS 加速方案构建数据湖：

OSS 数据存储（标准型单价）= 0.12 元/GB/每月

（参考链接：https://www.aliyun.com/price/product#/oss/detail ）

我们可以看到使用 JindoFS 加速方案构建数据湖，要节省 25% 的存储成本。同时 OSS 是按量计费，即计算存储分离，当计算和存储比例存在差异时，比如存储资源高速增长，计算资源增加较小时，成本优势会更加明显。

对 OSS 数据进行缓存加速，需要额外使用计算节点上部分磁盘空间，带来一定成本。这部分成本，一般取决于热数据或者要缓存数据的大小，跟要存储的数据总量关系不大。增加这部分成本，可以换取计算效率的提升和计算资源的节省，整体效果可以根据实际场景进行评估。

JindoFS 生态

数据湖是开放的，需要对接各种计算引擎。目前 JindoFS 已经明确支持 Spark、Flink、Hive、MapReduce、Presto 和 Impala 组件。同时，JindoFS 为了支持更好地使用数据湖，还提供 JindoTable 对结构化数据进行优化和查询加速；提供 JindoDistCp 来支持 HDFS 离线数据往 OSS 迁移；支持 JindoFuse 方便数据湖上加速机器学习训练。

JindoFS 缓存加速数据湖上的机器学习训练

作者：抚月

背景介绍

近些年，机器学习领域快速发展，广泛应用于各行各业。对于机器学习领域的从业人员来说，充满了大量的机遇和挑战。Tensorflow、PyTorch 等深度学习框架的出现，使开发者能够轻松地构建和部署机器学习应用。随着近些年云计算技术的不断成熟，越来越多的人接受将他们的开发、生产服务搬到云上平台，因为云环境在计算成本、规模扩展上比传统平台有显著的优势。云上平台为了达到弹性、节约成本，通常采用计算存储分离的解决方案。使用对象存储构建数据湖，可以降低成本、存储海量数据。在机器学习这个场景下，尤其适合将训练数据存储在数据湖上。

将训练数据存储在数据湖上具有以下优势：

1. 不需要将数据提前同步到训练节点。传统方式，我们需要将数据提前导入到计算节点的本地磁盘。而如果将数据存储在对象存储上，我们可以直接读取数据进行训练，减少准备工作。

2. 可以存储更大的训练数据，不再受限于计算节点本地磁盘大小。对于深度学习，拥有更多的数据，往往能取得更好的训练效果。

3. 计算资源可以弹性扩缩容，节约成本。机器学习通常使用使用更多核数的 CPU 或高端 GPU，较为昂贵，对象存储的成本就相对较低。将训练数据存储在数据湖上，可以与计算资源解耦。计算资源可以按需付费，随时释放，达到节省成本的目的。

然而，这种方式同时存在着一些问题和挑战：

1. 远端拉取数据的延迟和带宽无法随着计算资源线性扩展。硬件计算能力在不断发展，利用 GPU 进行计算可以取得更快的训练速度。使用云上弹性计算 ECS、容器服务可以快速调度起大规模的计算资源。访问对象存储需要走网络，得益于网络技术的发展，我

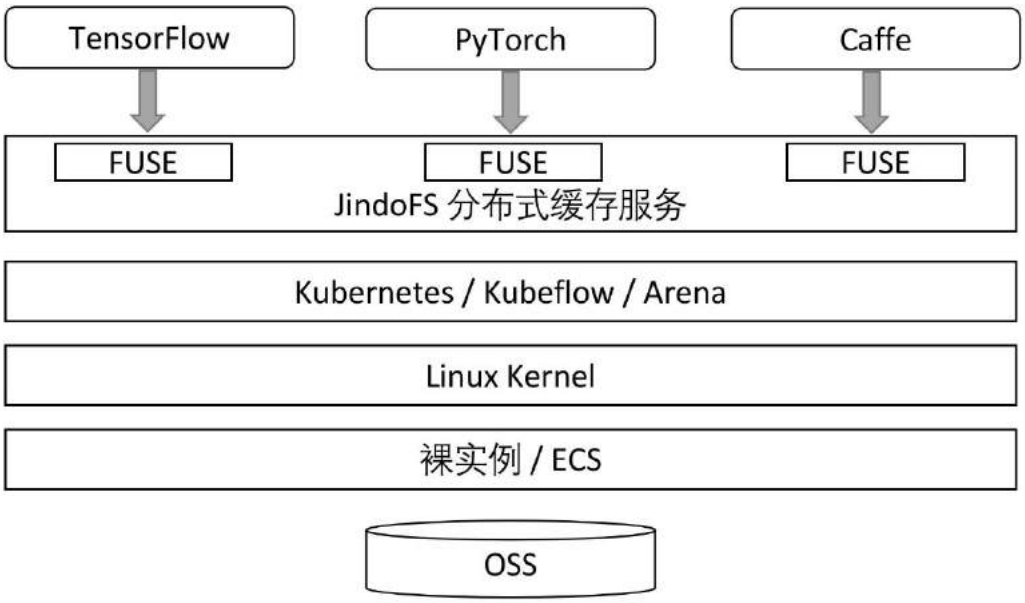
们访问对象存储有一个高速网络，即便如此，对象存储的网络延时和带宽无法随着集群规模线性扩展，可能会成为瓶颈，限制了训练速度。在计算存储分离架构下，如何高效地访问到这些数据，成为了一个巨大的挑战。

2. 需要更加便捷的通用的数据访问方式。深度学习框架如 TensorFlow 对于 GCS、HDFS 支持较为友好，而对于诸多第三方对象存储的支持上较为滞后。而 POSIX 接口是一种更自然友好的方式，使用类似于本地磁盘一样的方式访问数据，大大简化了开发者对存储系统的适配工作。

为了解决数据湖上机器学习训练常规方案存在的上述问题，JindoFS 针对这种场景提供了缓存加速优化方案。

基于 JindoFS 缓存加速的训练架构方案

JindoFS 提供了一个计算侧的分布式缓存系统，可以有效利用计算集群上的本地存储资源（磁盘或者内存）缓存 OSS 上的热数据，从而减少对 OSS 上数据的反复拉取，消耗网络带宽。



内存缓存

对于深度学习，我们可以选择计算能力更强的 GPU 机型，来获取更快的训练速度。此时需要高速的内存吞吐，才能让 GPU 充分跑满。此时我们可以使用 JindoFS 基于内存搭建分布式高速缓存。当整个集群的所有内存加起来足以支撑整个数据集时（除去任务本身所

需内存量），我们就可以利用内存缓存以及本地高速网络，来提供高的数据吞吐，加快计算速度。

磁盘缓存

对于一些机器学习场景，训练数据的规模超过了内存所能承载的大小，以及训练所需的 CPU/GPU 能力要求没有那么高，而要求数据访问有较高的吞吐。此时计算的瓶颈会受限于网络带宽压力。因此我们可以搭建使用本地 SSD 作为缓存介质的 JindoFS 分布式缓存服务，利用本地存储资源缓存热数据，来达到提高训练速度的效果。

FUSE 接口

JindoFS 包含了 FUSE 客户端，提供了简便的、熟悉的数据访问方式。通过 FUSE 程序将 JindoFS 集群实例映射到本地文件系统，就可以像访问本地磁盘文件一样，享受到 JindoFS 带来的加速效果。

实战：搭建 Kubernetes+JindoFS+Tensorflow 训练集群

创建 kubernetes 集群

我们前往阿里云-容器服务，创建一个 Kubernetes 集群。

安装 JindoFS 服务

1、前往容器服务->应用目录，进入“JindoFS”安装配置页面。

2、配置参数

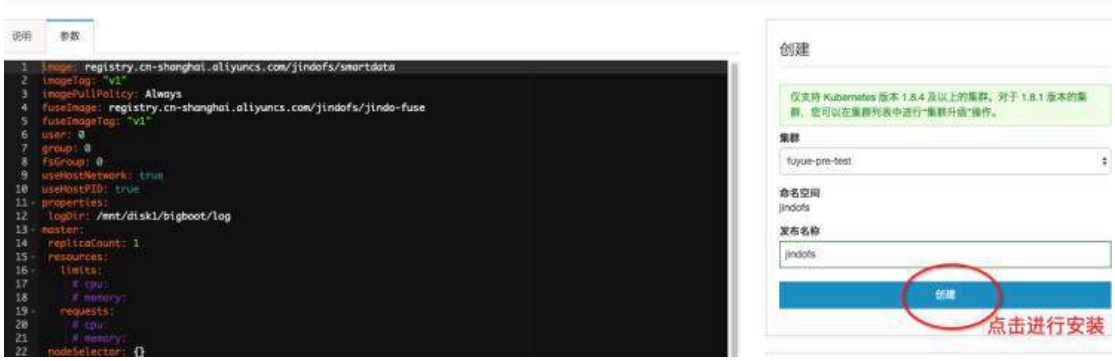
完整的配置模板可以参考容器服务-应用目录-jindofs 安装说明。

配置 OSS Bucket 和 AK，参考文档使用 JFS Scheme 的部署方式。我们需要修改以下配置项：

```
    jfs.namespaces: test
    jfs.namespaces.test.mode :  cache
    jfs.namespaces.test.oss.uri :  oss://xxx-sh-test.oss-cn-shanghai-internal.al
iyuncs.com/xxx/k8s_c1
    jfs.namespaces.test.oss.access.key :  xx
    jfs.namespaces.test.oss.access.secret :  xx
```

通过这些配置项，我们创建了一个名为 test 的命名空间，指向了 chengli-sh-test 这个 OSS bucket 的 xxx/k8s_c1 目录。后续我们通过 JindoFS 操作 test 命名空间的时候，就等同于操作该 OSS 目录。

3、安装服务



（1）验证安装成功

```
# kubectl get pods
NAME                              READY   STATUS      RESTARTS   AGE
jindofs-fuse-267vq                 1/1     Running     0          143m
jindofs-fuse-8qwdv                 1/1     Running     0          143m
jindofs-fuse-v6q7r                 1/1     Running     0          143m
jindofs-master-0                   1/1     Running     0          143m
jindofs-worker-mncqd               1/1     Running     0          143m
jindofs-worker-pk7j4               1/1     Running     0          143m
jindofs-worker-r2k99               1/1     Running     0          143m
```

在宿主机上访问/mnt/jfs/目录，即等同于访问 JindoFS 的文件

```
ls /mnt/jfs/test/
15885689452274647042-0  17820745254765068290-0  entrypoint.sh
```

安装 kubeflow（arena）

Kubeflow 是开源的基于 Kubernetes 云原生 AI 平台，用于开发、编排、部署和运行可扩展的便携式机器学习工作负载。Kubeflow 支持两种 TensorFlow 框架分布式训练，分别是参数服务器模式和 AllReduce 模式。基于阿里云容器服务团队开发的 Arena，用户可以提交这两种类型的分布式训练框架。

我们参照 github repo 上的使用文档进行安装。

启动 TF 作业

```
    arena submit mpi \
    --name job-jindofs\
     --gpus=8 \
     --workers=4 \
     --working-dir=/perseus-demo/tensorflow-demo/ \
     --data-dir /mnt/jfs/test:/data/imagenet \
     -e DATA_DIR=/data/imagenet -e num_batch=1000 \
     -e datasets_num_private_threads=8  \
     --image=registry.cn-hangzhou.aliyuncs.com/tensorflow-samples/perseus-benchmark-dawnbench-v2:centos7-cuda10.0-1.2.2-1.14-py36 \
     ./launch-example.sh 4 8
```

本文中，我们提交了一个 ResNet-50 模型作业，使用的是大小 144GB 的 ImageNet 数据集。数据以 TFRecord 格式存储，每个 TFRecord 大小约 130MB。模型作业和 ImageNet 数据集都可以在网上轻松找到。这些参数中，/mnt/jfs/是通过 JindoFS FUSE 挂载到宿主机的一个目录，test 是一个 namespace，对应一个 oss bucket。我们使用--data-dir 将这个目录映射到容器内的/data/imagenet 目录，这样作业就可以读取到 OSS 的数据了，对于读取过的数据，会自动缓存到 JindoFS 集群本地。

总结

通过 JindoFS 的缓存加速服务，只需要读取一遍数据，大部分的热数据将缓存到本地内存或磁盘，深度学习的训练速度可以得到显著提高。对于大部分训练，我们还可以使用预加载的方式先将数据加载到缓存中，来加快下一次训练的速度。

JindoTable 数据湖优化与查询加速

作者：健身

概述

近几年，数据湖架构的概念逐渐兴起，很多企业都在尝试构建数据湖。相比较大数据平台，数据湖在数据治理方面提出了更高的要求。对于数据湖场景所提出的新需求，“传统”的大数据工具在很多方面都面临着新的挑战。JindoTable 正是专为解决数据湖管理结构化数据甚至是半结构化数据的痛点而设计的，包括数据治理功能和查询加速功能。

数据优化

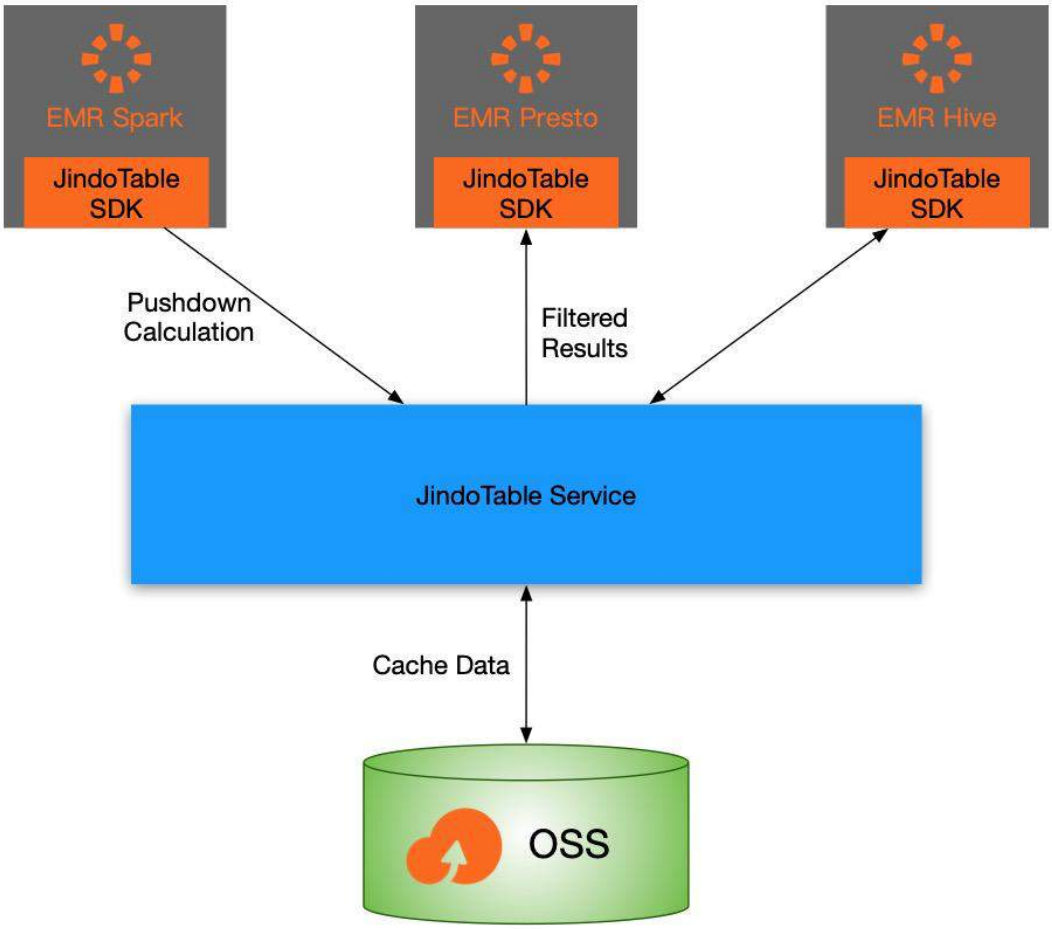
数据湖需要存储来自各种数据源的数据。对于 HDFS 集群，小文件问题让很多用户倍感烦恼。在存储计算分离的数据湖中，小文件同样会产生很多问题：过多的文件数会导致目录 list 时间显著变长，小文件也会影响很多计算引擎的并发度。此外，由于对象存储一般以对象为单位，小文件也会导致请求数量的上升，会明显影响元数据操作的性能，更会增加企业需要支付的费用。而如果数据文件过大，如果数据又使用了不可分割的压缩格式，后续计算的并发度会过低，导致无法充分发挥集群的计算能力。因此，即使是数据湖架构中，对数据文件进行治理和优化也是非常必要的。

基于数据湖所管理的元数据信息，JindoTable 为客户提供了一键式的优化功能，用户只要在资源较为空闲时触发优化指令，JindoTable 可以自动为用户优化数据，规整文件大小，进行适当的排序、预计算，生成适当的索引信息和统计信息，结合计算引擎的修改，可以为这些数据生成更加高效的执行计划，大幅减少用户查询的执行时间。数据优化对用户透明，优化前后不会出现读取的数据不一致的情况。这也是数据湖的数据治理所不可或缺的功能。

查询加速

JindoTable 还有一项重磅功能，就是查询加速功能。在数仓中，数据分析总是越快越好。尤其是 Ad-Hoc 场景，对查询延迟非常敏感。现在“湖仓一体”的概念也很火，对于数据湖这种普遍使用存储计算分离场景的架构，如何尽可能减少 IO 开销，对于缩短查询时间是非常关键的。

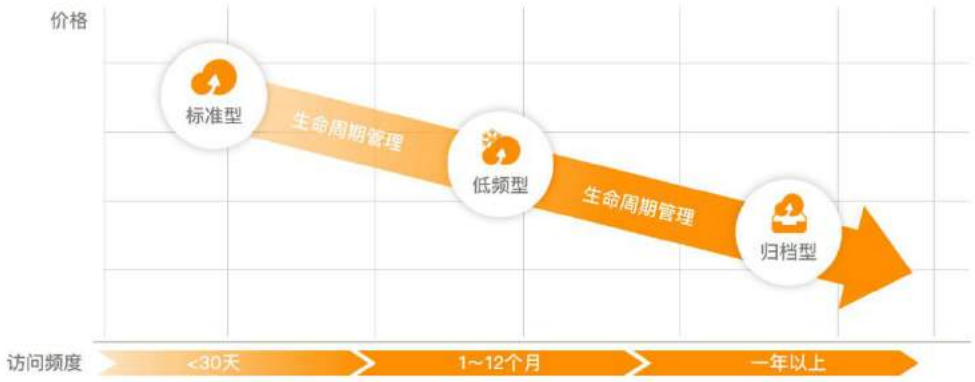
之前介绍的 JindoTable 数据优化功能，是在存储端减少额外开销，并且通过提前的计算，为运行时优化打好基础。JindoTable 的查询加速功能则是在查询执行时，通过把计算推向存储，减少计算时整体的 IO 压力，同时利用存储端空闲的计算资源提供高效的计算，缩短整体查询时间。JindoTable 的加速服务结合修改后的各种计算引擎，可以把尽可能多的算子下推到缓存端，并且利用高效的 native 计算能力过滤大量原始数据，再把数据高效地传输给计算引擎。这样，计算引擎所需处理的数据大大减少，甚至一些计算也可以直接略过，后续的计算所需的时间自然也就大为减少。



分层存储

数据湖所存储的数据量通常增长迅速。对于传统的 Hadoop 集群，如果数据量急剧增长，所需的存储资源也要相应增加，这样会导致集群规模迅速扩大，计算资源也会变得过剩。抛开集群规模增长导致的其他问题不谈，光是运营集群的成本问题就足够让人头疼。好在公有云平台提供了对象存储的服务，我们可以按存储的数据量来付费，这在节约成本的同时，用户也不用担心 HDFS 在集群资源和数据量快速增长情况下的稳定性问题。但数据量快速增长还是会等比例的增加整体开销。

阿里云的对象存储服务 OSS，为用户提供了低频存储和归档存储，对于访问不是那么频繁的数据，如果能够转为低频或归档模式来存储，可以尽量节约成本。而一部分数据如果有频繁的访问需求，放在远离计算资源的对象存储上，又会导致计算时的 IO 出现瓶颈。JindoTable 对接数据湖中各种计算引擎，以表或分区为最小单位，统计数据的访问频次。根据用户设定的规则，JindoTable 可以告诉用户哪些表或者分区的访问频次较高，让用户可以通过 JindoTable 命令，借助 JindoFS 提供的底层支持，把这些表或者分区对应的数据缓存到计算集群内，加速查询的执行。同时，对于访问频次较低的表或者分区，用户也可以使用 JindoTable 把对应的数据转为低频或者归档存储类型，或是设置生命周期。在需要对归档数据操作的时候，可以直接用 JindoTable 对归档数据进行解冻。JindoTable 还为用户提供了元数据管理，方便用户检视表或者分区当前的存储状态。JindoTable 让用户能尽可能高效地管理自己的数据，节约成本的同时，不牺牲计算性能。



小结

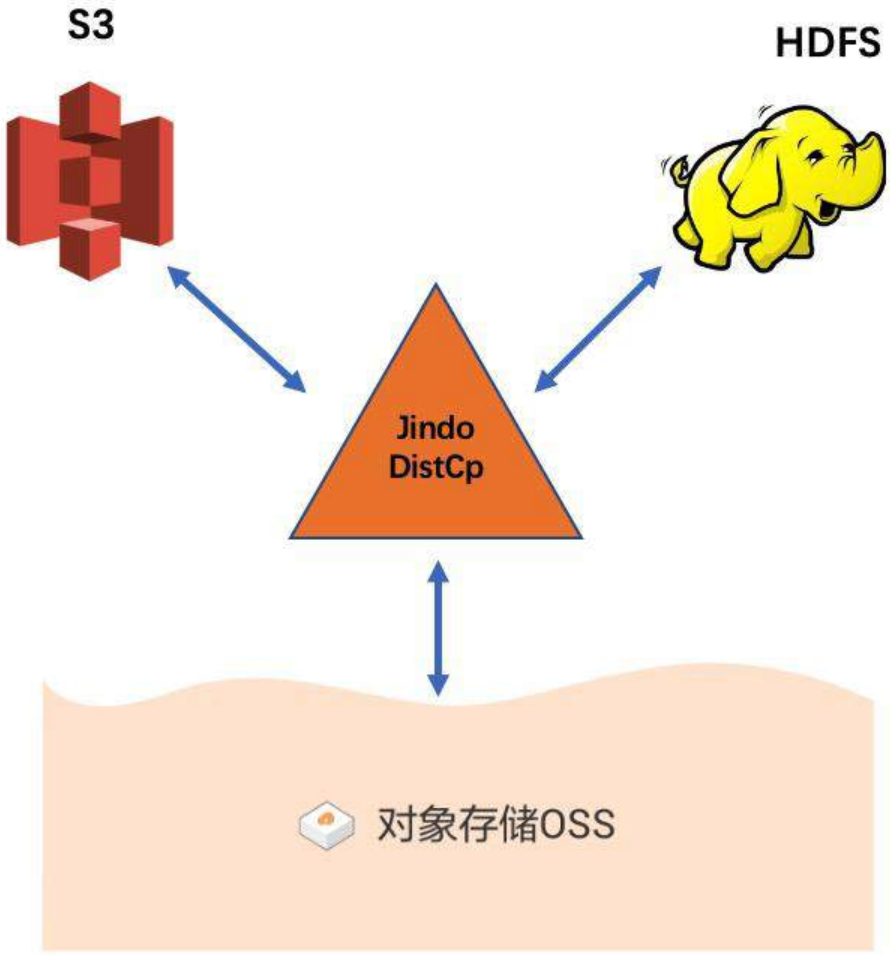
对于企业来说，数据湖为各种来源的数据提供了整合的可能性。背靠丰富的云产品体系，数据湖架构可以帮助客户进一步发掘数据价值，实现企业愿景。JindoTable 在数据湖解决方案中，为用户提供数据治理和查询加速的增值功能，进一步降低用户数据入湖的门槛，帮助用户在更低的成本下，实现更高的数据价值。

JindoDistCp 数据湖离线数据迁移最佳实践

作者：扬礼

数据湖就像是一个“大水池”，是一种把各类异构数据进行集中存储的架构。数据湖是一种存储架构，在阿里云上可以利用 OSS 对象存储，来当数据湖的地基。企业基于阿里云服务，可以快速挖出一个适合自己的"湖"，而且这个"湖"根据需求，可大可小，按"注水量"付费。在挖好这个"湖"后，重要的步骤就是如何把各种异构数据注入到湖里。在传统的大数据领域用户经常使用 HDFS 作为异构数据的底层存储来储存大量的数据，其中大部分可通过离线数据迁移来注入到以 OSS 作为底层存储的数据湖中。在进行数据迁移、数据拷贝的场景中，大家选择最常用的离线数据迁移工具是 Hadoop 自带的 DistCp 工具，但是它不能很好利用对象存储系统如 OSS 的特性，导致效率低下并且不能最终保证一致性，提供的功能选项也比较简单，不能很好的满足用户的需求。此时一个高效、功能丰富的离线数据迁移工具成为影响离线数据入湖效率的重要因素。

随着阿里云 JindoFS SDK 的全面放开使用，基于 JindoFS SDK 的数据湖离线数据迁移利器 JindoDistCp 现在也全面面向用户开放使用。JindoDistCp 是阿里云 E-MapReduce 团队开发的大规模集群内部和集群之间分布式文件拷贝的工具。 它使用 MapReduce 实现文件分发，错误处理和恢复，把文件和目录的列表作为 map/reduce 任务的输入，每个任务会完成源列表中部分文件的拷贝。目前全面支持 HDFS/S3/OSS 之间的数据拷贝场景，提供多种个性化拷贝参数和多种拷贝策略。重点优化从 HDFS 和 S3 到数据湖底座 OSS 的数据拷贝场景，通过定制化 CopyCommitter，实现 No-Rename 拷贝，并保证数据拷贝落地的一致性。功能覆盖 S3DistCp 和 HadoopDistCp 的功能，性能较 HadoopDistCp 有较大提升，目标提供高效、稳定、安全的数据湖离线数据迁移工具。本文主要介绍如何使用 JindoDistCp 来进行基本离线数据迁移，以及如何在不同场景下提高离线数据迁移性能。值得一提的是，此前 JindoDistCp 仅限于 E-MapReduce 产品内部使用，此次全方位面向整个阿里云 OSS/HDFS 用户放开，并提供官方维护和支持技术，欢迎广大用户集成和使用。



HadoopDistCp

HadoopDistCp 是 Hadoop 集成的分布式数据迁移工具，提供了基本文件拷贝、覆盖拷贝、指定 map 并行度、log 输出路径等功能。在 Hadoop2x 上对 DistCp 进行了部分优化例如拷贝策略的选择，默认使用 uniformsize（每个 map 会平衡文件大小）如果指定 dynamic，则会使用 DynamicInputFormat。这些功能优化了普通 hdfs 间数据拷贝，但是对于对象存储系统如 OSS 缺少数据写入方面的优化。

S3DistCp

S3DistCp 是 AWS 为 S3 上存储提供的 distcp 工具，S3DistCp 是 HadoopDistCp 的扩展，它进行了优化使得其可以和 S3 结合使用，并新增了一些实用功能。新增功能如增量复制文件、复制文件时指定压缩方式、根据模式进行数据聚合、按照文件清单进行拷贝等。

JindoDistCp

JindoDistCp 是一个简单易用的分布式文件拷贝工具，目前主要用在 E-Mapreduce 集群内，主要提供 HDFS 和 S3 到 OSS 的数据迁移服务，相比于 HadoopDistCp 和 S3DistCp，JindoDistCp 做了很多优化以及新增了许多个性化功能，并且深度结合 OSS 对象存储的特性，定制化 CopyCommitter，实现 No-Rename 拷贝，大大缩短离线数据入湖迁移时间消耗。

为什么使用 JindoDistCp?

1. 效率高，在测试场景中最高可到 1.59 倍的加速，大大提高数据湖离线数据迁移效率。

2. 基本功能的选项丰富，提供多种拷贝方式和场景优化策略。

3. 深度结合 OSS，迁移后的文件直接以归档和低频、压缩等方式存放，无需额外操作。

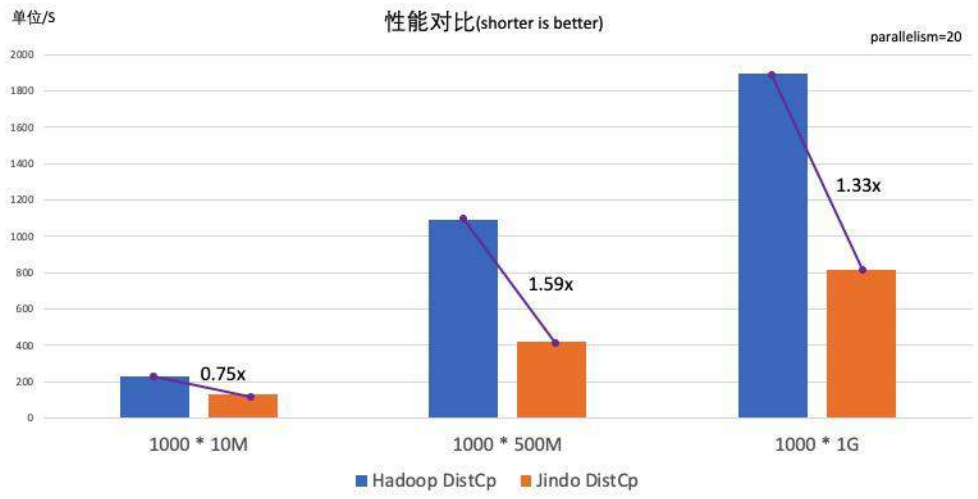
4. 实现 No-Rename 拷贝，保证数据一致性。

5. 场景全面，可完全替代 HadoopDistCp，支持多 Hadoop 版本。

使用 JindoDistCp 性能提升多少?

我们做了一个 JindoDistCp 和 HadoopDistCp 的性能对比，在这个测试中我们以 HDFS 到 OSS 离线数据迁移为主要场景，利用 Hadoop 自带的测试数据集 TestDFSIO 分别生成 1000 个 10M、1000 个 500M、1000 个 1G 大小的文件进行从 HDFS 拷贝数据到 OSS 上的测试过程。

单位/s	1000 * 10M	1000 * 500M	1000 * 1G
JindoDistcp	132	420	813
HadoopDistcp	229	1089	1900



分析测试结果，可以看出 JindoDistCp 相比 HadoopDistCp 具有较大的性能提升，在测试场景中最高可达到 1.59 倍加速效果。

使用工具包

下载 jar 包

我们去 github repo 下载最新的 jar 包 jindo-distcp-x.x.x.jar

注意： 目前 Jar 包只支持 Linux、MacOS 操作系统，因为 SDK 底层采用了 native 代码。我们会尽快推出全新版本，支持更多平台，敬请关注。

配置 OSS 访问 AK

您可以在命令中使用程序执行时指定--ossKey、--ossSecret、--ossEndPoint 参数选项来指定 AK。

示例命令如下：

```
   hadoop jar jindo-distcp-2.7.3.jar --src /data/incoming/hourly_table --dest
oss://yang-hhht/hourly_table    --ossKey    yourkey    --ossSecret    yoursecret
--ossEndPoint oss-cn-hangzhou.aliyuncs.com
```

您也可以将 oss 的 ak、secret、endpoint 预先配置在 hadoop 的 core-site.xml 文件里 ，避免每次使用时临时填写 ak。

```
<configuration>
    <property>
        <name>fs.jfs.cache.oss-accessKeyId</name>
        <value>xxx</value>
    </property>

    <property>
        <name>fs.jfs.cache.oss-accessKeySecret</name>
        <value>xxx</value>
    </property>

    <property>
        <name>fs.jfs.cache.oss-endpoint</name>
        <value>oss-cn-xxx.aliyuncs.com</value>
    </property>
</configuration>
```

另外，我们推荐配置免密功能，避免明文保存 accessKey，提高安全性。

使用手册

JindoDistCp 提供多种实用功能及其对应的参数选择，下面介绍参数含义及其示例

Option	Description	Required
--src	指定拷贝源路径，可以是 Hdfs 或者 OSS 路径 Example：--src oss://testbucket/sourceDir	Yes
--dest	指定拷贝目标路径，可以是 Hdfs 或者 OSS 路径 Example：--dest oss://testbucket/destDir	Yes
--parallelism	指定拷贝的任务并行度，可根据集群资源调节 Example：--parallelism 10	No
--policy	指定拷贝到 OSS 文件策略,可选择 archive(归档)和 ia(低频)，仅对 dest 为 OSS 路径生效 Example: --policy archive	No

--srcPattern	通过指定正则表达式来选择需要拷贝的文件，以及过滤掉不需要拷贝的文件 Example：--srcPattern .*\.log	No
--deleteOnSuccess	指定是否在拷贝完成后删除源路径下的文件 Example：--deleteOnSuccess	No
--outputCodec	指定拷贝文件按何种方式压缩，当前版本支持编解码器 gzip、gz、lzo、lzop 和 snappy 以及关键字 none 和 keep（默认值）。这些关键字含义如下： "none" - 保存为未压缩的文件。如果文件已压缩，则 jindo distcp 会将其解压缩。 "keep" - 不更改文件压缩形态，按原样复制。 Example：--outputCodec gzip	No
--srcPrefixesFile	指定需要一次拷贝的文件列表，列表里文件以 src 路径作为前缀 Example:--srcPrefixesFile file:///opt/folders.txt	No
--outputManifest	指定在 dest 目录下生成一个 Gzip 压缩的文件，记录已完成拷贝的文件信息 Example:--outputManifest manifest-2020-04-17.gz	No
--requirePreviousManifest	指定本次拷贝是否需要读取之前已拷贝文件信息，从而进行增量更新，指定该参数则表明不需要 Example：--requirePreviousManifest	No
--previousManifest	指定本次拷贝需要读取之前已拷贝文件的信息，完成增量更新 Example:--previousManifest oss://testbucket/manifest-2020-04-16.gz	No

--copyFromManifest	从已完成的 Manifest 文件中进行拷贝，通常和 --previousManifest 配合使用 Example:--previousManifest oss://testbucket/manifest-2020-04-16.gz --copyFromManifest	No
--groupBy	指定正则表达式将符合规则的文件进行聚合，和 targetSize 选项配合使用 Example：--groupBy='.*/([a-z]+).*.txt'	No
--targetSize	指定聚合的文件大小，聚合完的文件不超过这个值得大小，单位 M Example：10	No
--enableBalancePlan	执行策略：适用于数据量差异不大的场景 Example：--enableBalancePlan	No
--enableDynamicPlan	执行策略：适用于数据量差异较大的场景如大小文件混合 Example：--enableDynamicPlan	No
--enableTransaction	执行策略：保证 Job 级别的一致性，默认是 task 级别 Example：--enableTransaction	No
--diff	对比策略，查看本次拷贝是否完成全部文件拷贝，未完成会生成文件列表 Example：--diff	No
--ossKey	指定 OSS 访问的 accessKey Example：--ossKey yourkey	No
--ossSecret	指定 OSS 访问的 accessSecret Example：--ossSecret yoursecret	No

--ossEndPoint	指定 OSS 访问的 region 信息 Example:--ossEndPoint oss-cn-hangzhou.aliyuncs.com	No
--cleanUpPending	清理 OSS 残留文件，这可能会花费一定时间 Example：--cleanUpPending	No
--policy	以归档类型写入 OSS Example：--policy archive 以低频类型写入 OSS Example：--policy ia	No
--queue	指定 MR 任务的队列名称 Example：--queue yourQueue	No
--bandwidth	指定写入带宽和速率(以 M 为单位) Example：--bandwidth 5	No
--s3Key	指定访问 s3 的 key --s3Key youKey	No
--s3Secret	指定访问 s3 的 secret --s3Secret yourSecret	No
--s3EndPoint	指定访问 s3 的 region 信息 --s3EndPoint s3-us-west-1.amazonaws.com	No

更多详细使用细节，请参考 JindoDistCp 使用指南。

数据湖构建（DLF）

数据湖元数据服务的实现和挑战

作者：履霜

大数据引擎的现状

在大数据计算和存储领域，因不同业务场景、不同数据规模，诞生了很多适合处理不同需求的各类大数据引擎，比如计算引擎类有数据分析引擎 Hive、交互式分析引擎 Presto、迭代计算引擎 spark 以及流处理引擎 Flink 等，存储类有日志存储系统的 SLS、分布式文件系统 HDFS 等，这些引擎和系统很好的满足了某一领域的业务需求，但也存在非常严重的数据孤岛问题：在同一份数据上综合使用这些系统，必然面临着大量的 ETL 工作，而且更关键的是在目前各种公司业务链路上这种使用方式非常常见，同时因数据加工、转储产生的成本以及整体延时大大增加，业务决策时间也相应变长，解决这一问题的关键在于引擎元数据需要互通，只有构建满足各种引擎需求的数据湖统一元数据服务视图，才能实现数据共享，避免其中额外的 ETL 成本以及降低链路的延时。

数据湖元数据服务的设计

数据湖元数据服务的设计目标是能够在大数据引擎、存储多样性的环境下，构建不同存储系统、格式和不同计算引擎统一元数据视图，并具备统一的权限、元数据，且需要兼容和扩展开源大数据生态元数据服务，支持自动获取元数据，并达到一次管理多次使用的目的，这样既能够兼容开源生态，也具备极大的易用性。另外元数据应该支持追溯、审计，这就要求数据湖统一元数据服务具备以下能力和价值：

- 提供统一权限、元数据管理模块：统一的权限/元数据管理模块是各类引擎和存储互通的基础，不仅权限/元数据模型需要满足业务对于权限隔离的需要，也需要能够合理支持目前引擎的各种权限模型。

- 提供大规模元数据的存储和服务能力，提升元数据服务能力极限，满足超大数据规模和场景。

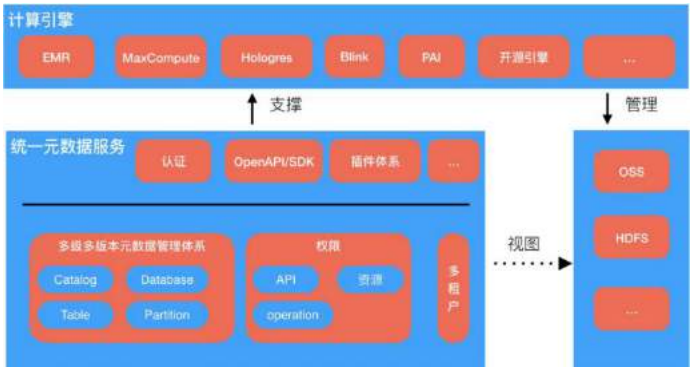
- 提供存储统一的元数据管理视图：将各类存储系统（对象、文件、日志等系统）上数据进行结构化既能够方便数据的管理，也因为有了统一元数据，才能进行下一步的分析和处理。

- 支撑丰富的计算引擎：各类引擎，通过统一元数据服务视图访问和计算其中的数据，满足不同的场景需求。比如 PAI/MaxCompute/Hive 等可以在同一份 OSS 数据上进行计算和分析。通过引擎支撑的多样化，业务场景将越来容易进行场景转换和使用。

- 元数据操作的追溯/审计。

- 元数据自动发现和收集能力：通过对文件存储的目录/文件/文件格式的自动感知，自动创建和维护元数据的一致性，方便存储数据的自动化维护和管理。

数据湖元数据服务的架构



- 元数据服务上层是引擎接入层

    - 通过提供各种协议的 SDK 和插件，能够灵活支撑各种引擎的对接，满足引擎对于元数据服务的访问需要。并且通过元数据服务提供的视图，对底层文件系统进行分析和处理。
    - 通过插件体系无缝兼容 EMR 引擎，能够使 EMR 全家桶开箱即用，用户全程无感知，即可体验统一元数据服务，避免原 Mysql 等存储的可扩展性差的问题。

- 元数据服务提供存储视图

    - 通过对不同存储格式/存储目录文件的抽象，为引擎提供统一元数据服务，同时能够避免多引擎独立使用元数据服务之间的不一致性。

- 元数据的管理和自动发现

元数据通过各种方式能够灵活的、跨引擎管理元数据，既能使用户方便的集成元数据服务、扩展元数据服务能力，也能够降低管理成本。

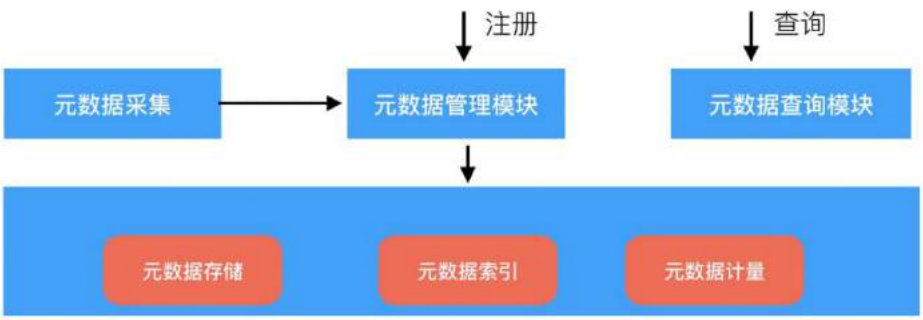
- Web Console、Sdk、各类引擎客户端和接口

  - 兼容开源生态引擎的各类数据库/表/分区上的 DDL 操作。
  - 提供多版本元数据管理/追溯的能力。
  - 通过元数据能力的开放，在 ETL 部分/开源工具部分将来也能通过各式插件进行对接，进一步完善整体生态。

- 元数据自动发现

元数据自动发现能力是元数据管理能力的另一核心部分，能够自动收集各处文件系统散落的数据，极大了拓宽了统一元数据服务的场景，节省了管理的代价和复杂性。这其中的能力包括：

  - 自动分析目录层次，动态增量创建 database/table/partition 等元数据。
  - 自动分析文件格式，对于各类格式比如常规文本格式及开源大数据格式 parquet、orc 等都进行了支持。



元数据服务的未来

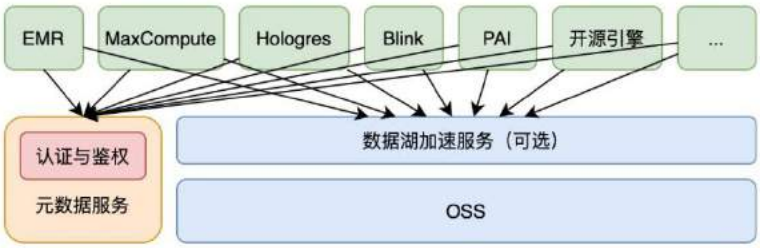
数据湖元数据服务为大数据而生，为互通生态而生，期望后续继续完善其服务能力和支撑更多的大数据引擎，通过开放的服务能力、存储能力、统一的权限及元数据管理能力，为客户节省管理/人力/存储等各项成本，实现客户自己的业务价值。

多引擎集成挖掘湖上数据价值

作者：辛庸

数据湖已经逐步走到了精细化的管理，这意味着原始的计算引擎直接读写存储的方式应当逐步演变为使用标准方式读写数据湖存储。然而“标准方式”实际上并无业界标准，与具体的计算引擎深度绑定，因此，支持计算引擎的丰富程度也就成了衡量数据湖的一个准则。

阿里云数据湖构建服务支持丰富的计算引擎对接，包括但不限于阿里云产品 E-MapReduce（EMR）、MaxCompute（开发中）、Blink（开发中）、Hologres（开发中）、PAI（开发中），以及开源系的 Hive、Spark、Presto 等等。



数据湖的对接主要体现在元数据与存储引擎两个方面。元数据为所有用户所共享，提供统一的元数据访问接口。各个引擎使用定制化的元数据访问客户端来访问元数据。元数据服务为各个用户提供租户隔离保证和认证鉴权服务。在数据存储方面，用户使用自己的 OSS 存储存储数据，各个引擎需要具备访问 OSS 的功能，这对于阿里云服务和大部分支持 HDFS 存储的引擎都不是什么问题。在 OSS 存储上层，数据湖构建服务还提供了可选的数据湖加速服务。而使用该服务也非常简单，只需要在引擎侧将对 OSS 的访问路径替换为加速服务提供的路径即可。

多引擎支持

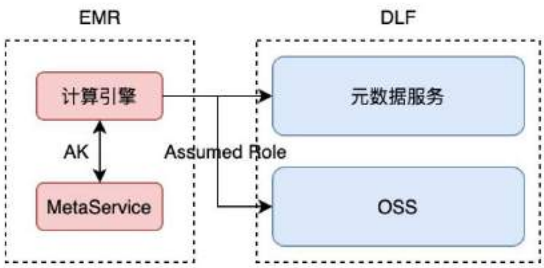
EMR

使用 EMR 访问数据湖构建服务最为简单。在用户创建 EMR 集群的时候，可以直接选择使用数据湖构建服务的元数据服务作为元数据库，EMR 服务与数据湖构建服务在认证鉴权方面深度打通，只要获得用户的授权，EMR 基于数据湖元数据的使用体验和用本地元数据库体验几乎完全一致。因为在 EMR 集群创建阶段已经自动安装了数据构建服务的相

关 SDK，同时 EMR 上的开源计算引擎 Spark、Hive 和 Presto 都完成了对数据湖构建服务的兼容支持，所以用户通过 EMR 引擎可获得数据湖分析的最佳体验。

即便用户在创建集群的时候没有选择使用数据湖元数据服务，也可以在后期通过配置将元数据转移到数据湖元数据服务。在数据存储方面，由于数据湖构建在用户 OSS 之上，因此不存在跨服务认证问题。而 EMR 天然内置了 OSS 的访问支持，并提供了 JindoFS 的增强功能，使得 EMR 用户访问数据湖数据将获得比原生 OSS 访问更佳的性能提升（JindoFS 是构建在 OSS 之上的，专门为大数据使用场景深度定制的文件系统，在元数据操作、数据缓存等方面有独到的优势。更多相关文章请参考文末链接。）。对于那些已经在使用 EMR 的用户，数据湖构建服务提供了一键迁移工具，方便用户将元数据从本地元数据库转移到数据湖元数据服务。对于数据部分，用户可以选择将数据继续存储在本地 HDFS 系统，或者迁移至 OSS。

值得一提的是 EMR 提供了免 AK 访问数据湖构建服务的功能。用户不需要提供 AK 即可访问元数据服务以及 OSS 上的数据，前提是数据湖构建服务获得了用户的授权。该功能的实现基于 EMR 的 MetaService 功能，在用户授权的前提下，MetaService 为本机请求提供 Assume Role 的 AK 信息，从而让调用者以用户身份访问目标服务。免 AK 访问功能最小化了用户机密信息丢失的风险。



阿里云服务

MaxCompute、实时计算、Hologres、PAI 等阿里云服务，通过数据湖构建服务在底层打通数据，在一定程度上达到通过使用一份数据满足多种不同场景的计算需求。例如，MaxCompute 可以直接读取数据湖构建服务的数据内容，配合 DataWorks 给用户良好的数仓使用体验；实时计算服务可以将数据实时注入数据湖，从而实现基于数据湖的流批一体计算；而 Hologres 则可以应对对性能要求较高的场景，既能够对于数据湖内温冷数据做加速分析，也能够配合 Hologres 的内置存储处理温热数据，从而构建数据从产生到归档一整个生命周期的一站式分析方案；PAI 则可以将数据湖作为数据来源，也可以将其他引擎计算的离线特征存储到数据湖做模型构建之用等等。

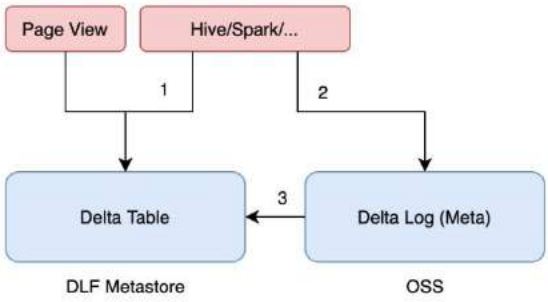
开源引擎

对于直接使用开源产品的用户，数据湖构建服务也提供了一些方法让这些服务能够快速对接数据湖，不过这可能需要用户基于开源代码打个 patch，以便在开源代码中插件化地嵌入数据湖元数据的访问。对于数据访问，由 EMR 团队贡献 Hadoop 社区的 OSSFileSystem 可以完成对用户 OSS 存储的访问，因此只要引擎支持读写 HDFS，就可以读写 OSS。对于元数据和 OSS 的访问控制，则统一使用阿里云 RAM 方式，体验上保证一致。

特殊格式支持

除了计算引擎本身，某些引擎还可以读写特定的表格式，如近两年兴起的 Delta Lake、Hudi、Iceberg 等等。数据湖构建服务不局限用户使用哪一种存储格式，但是由于这些表格式有自己的元数据，因此在引擎对接方面仍然需要做一些额外的工作。以 Delta Lake 为例，其元数据存储在 OSS 之上，而元数据服务中也会存一份元数据，这样便引出了两个问题：一是引擎应该读取哪份元数据，二是如果有必要存两份元数据的话，元数据的一致性如何保证。对于第一个问题，如果是开源组件，我们应当遵循开源默认的做法，通常

是让引擎去读取 OSS 中的元数据。对于元数据服务中的元数据也非常有必要保存，这可以服务于页面显示，并且可以省去 OSS 元数据解析的巨大性能损耗。对于第二个问题，数据湖构建服务开发了一个 hook 工具，每当 Delta Lake 有事务提交时，便会自动触发 hook，将 OSS 中的元数据同步到元数据服务中。除此之外，Delta Lake 元数据的设计最大程度的保证了一份元数据同时支持 Spark、Hive、Presto 等多个引擎，用户不必像开源产品那样为不同引擎维护不同的元数据。



如图所示，元数据服务中的元数据（Delta Table）为 OSS 中 Delta 表元数据（Delta Log）的映像，并通过 commit hook（3）保持同步；Delta Tabe 为页面展示，以及 Hive、Spark 引擎读取必要信息（如 table path、InputFormat/OutputFormat 等）所用（1）；当引擎获得必要信息后读取 table path 下 Delta 表的元数据，完成元数据解析和数据读取工作（2）；当引擎完成数据操作并提交事务后，Delta Log 通过 commit hook 同步至 Delta Table（3），完成一个循环。

未来工作

数据湖构建服务的多引擎支持在诸多方面尚在快速发展之中。未来我们将围绕以下几点继续开展工作：

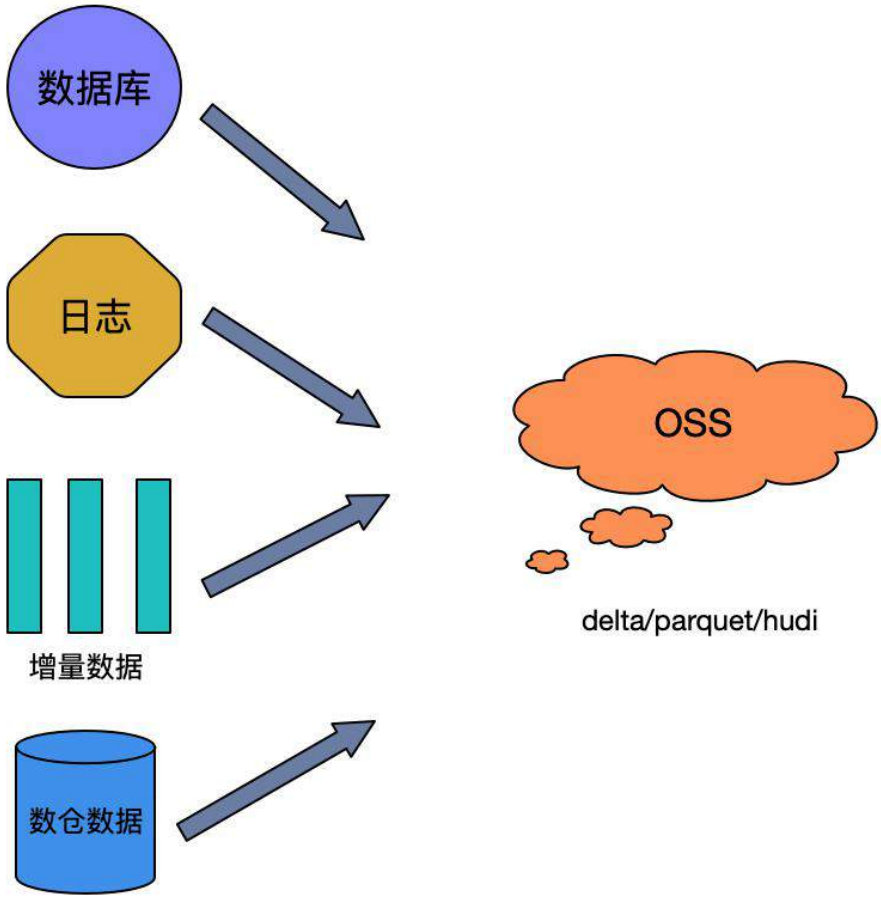
- 阿里云服务对接：从解决方案角度完善阿里云产品对接，给用户更平滑的体验；
- 开源引擎支持：更多的开源引擎支持，如 Impala、Flink 等；
- 功能丰富：如完善统计信息支持，支持事务接口等；
- 性能提升：做到超越本地元数据与本地 HDFS 存储的性能。

多数据源一站式入湖

作者：空净

背景

数据湖作为一个集中化的数据存储仓库，支持的数据类型具有多样性，包括结构化、半结构化以及非结构化的数据，数据来源上包含数据库数据、binglog 增量数据、日志数据以及已有数仓上的存量数据等.数据湖能够将这些不同来源、不同格式的数据集中存储管理在高性价比的存储如 OSS 等对象存储中，并对外提供统一的数据分析方式，有效解决了企业中面临的数据孤岛问题，同时大大降低了企业存储和使用数据的成本。



由于数据湖数据来源的多样性，如何简单高效的将这些异构数据源的数据迁移到中心化的数据湖存储中，是数据湖构建过程面临的问题.为此，我们需要提供完善的一站式入湖的能力，解决我们面临的问题，主要包括以下几点：

- 支持异构数据源统一的入湖方式

提供一个简单统一的入湖方式，用户可以通过简单的页面配置实现异构数据源的入湖操作。

- 满足数据入湖的时效性

对于日志、binglog 等类型的数据源，需要实现分钟级延迟的数据入湖能力，满足实时交互式 分析场景对时效性的要求。

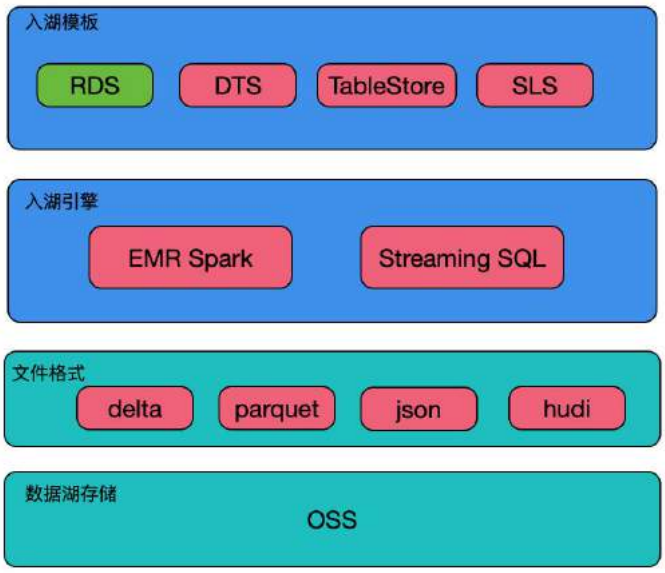
- 支持数据源的实时变更

对于数据库、TableStore Tunnel 等类型的数据源，源头数据会经常发生变更，比如数据层面的 update，delete 等操作，甚至 schema 层面的字段结构变更. 需要利用更好的数据格式来支持这类变更行为。

为此，阿里云新推出了**数据湖构建**（Data Lake Formation，DLF）服务，提供了完整的一站式入湖解决方案。

整体方案

数据湖构建的入湖技术方案如下图所示：



数据入湖整体上分为入湖模板、入湖引擎、文件格式以及数据湖存储四个部分：

入湖模板

入湖模板定义了常见的数据源入湖方式，目前主要包括 RDS 全量模板、DTS 增量模板、TableStore 模板、SLS 模板以及文件格式转换 5 种模板。

用户根据不同的数据源选择相应的入湖模板，然后填写源头相关参数信息，即可完成入湖模板的创建，并提交给入湖引擎运行。

入湖引擎

入湖引擎使用了阿里云 EMR 团队自研的 Spark Streaming SQL 以及 EMR Spark 引擎，Streaming SQL 基于 Spark Structured Streaming，提供了相对完善的 Streaming SQL 语法，极大简化了实时计算的开发成本。对于实时增量模板，上层入湖模板部分将入湖模板翻译成 Streaming SQL，然后提交 Spark 集群运行。我们在 Streaming SQL 里面扩展了 Merge Into 语法来支持 update、delete 操作。对于 RDS 等全量模板，则直接翻译成 Spark SQL 运行。

文件格式

DLF 支持的文件格式包括 Delta Lake、Parquet、json 等，更多文件格式比如 Hudi 也在接入中。Delta Lake 和 Hudi 等文件格式能很好的支持 update、delete 等操作，同时支持 schema merge 功能。可以很好的解决数据源实时变更问题。

数据湖存储

数据湖数据统一放在 OSS 对象存储中，OSS 提供了海量数据存储的能力，同时在可靠性，价格等方面更具优势。

一站式入湖方案在很好的解决了前面提的几个问题：

- 支持异构数据源统一的入湖方式

通过模板配置，实现了统一简单的数据入湖方式。

- 满足数据入湖的时效性

通过自研 Streaming SQL 实现了分钟级延迟的数据实时入湖，满足了时效性要求。

- 支持数据源的实时变更

通过引进 Delta Lake 等更优的文件格式，实现了对 update、delete 等数据实时变更要求。

实时入湖

随着大数据的不断发展，用户对数据时效性的要求越来越高，实时入湖也是我们重点关注的场景，目前我们已经支持了 DTS、TableStore 以及 SLS 的实时入湖能力。

DTS 增量数据实时入湖

DTS 是阿里云提供了高可靠的数据传输服务，支持不同类型数据库增量数据的订阅和消费。我们实现了 DTS 实时订阅数据的入湖，支持用户已有订阅通道入湖和自动创建订阅通道入湖两种方式，减少用户配置成本。

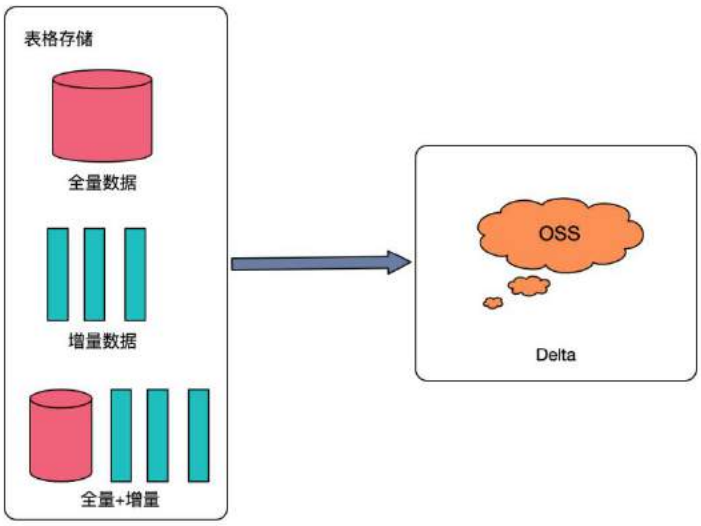
在技术上，支持增量数据对历史数据的 update、delete 变更操作，实现分钟级延迟的数据变更感知能力。技术实现上在 Streaming SQL 中扩展了 merge into 语法来对接底层文件格式 Delta Lake 的相关接口。

```
MERGE INTO delta_tbl AS target
USING (
  select recordType, pk, ...
  from {{binlog_parser_subquery}}
) AS source
ON target.pk = source.pk
WHEN MATCHED AND source.recordType='UPDATE' THEN
UPDATE SET *
WHEN MATCHED AND source.recordType='DELETE' THEN
DELETE
WHEN NOT MATCHED THEN
INSERT *
```

和传统数仓的 binlog 入仓相比，基于数据湖的方案具有更大的优势. 在传统数仓中，为了实现数据库等变更数据的入仓，通常需要维护两张表，一张增量表用于存放每天新增的数据库变更明细数据，另外一张全量表，存放历史所有的 merge 数据，全量表每天和增量表更据主键做 merge 操作. 显然，基于数据湖方案在实现的简单性和时效性上都更优。

TableStore 实时入湖

TableStore 是阿里云提供的是阿里云自研的 NoSQL 多模型数据库，提供海量结构化数据存储以及快速的查询和分析服务.它同时支持了通道功能，支持变更数据的实时消费。我们支持 TableStore 全量通道、增量通道以及全量加增量通道的实现入湖.其中全量通道包含历史全量数据，增量通道包含增量变化的数据，全量加增量通道则包含了历史全量和增量变化的数据。



SLS 日志实时入湖

SLS 是阿里云提供的针对日志类数据的一站式服务，主要存放用户日志数据。将 SLS 中的日志数据实时归档到数据湖中，进行分析处理可以充分挖掘数据中的价值。目前通过 SLS 入湖模板，填写 project、logstore 等少量信息，即可完成日志实时入湖的能力。

总结展望

一站式入湖功能极大的降低了异构数据源入湖的成本，满足了 SLS、DTS 等数据源入湖的时效性要求，同时也支持了数据源实时变更的能力。通过一站式入湖，将不同数据源的数据统一归并到以 OSS 对象存储为基础架构的集中式数据湖存储中，解决了企业面临的数据孤岛问题，为统一的数据分析打好了基础。

后续一站式入湖一方面将继续完善功能，支持更多类型的数据源，入湖模板方面开放更多能力给用户，支持自定义 ETL 的功能,提高灵活性。另一方面，将会在性能优化方面不断投入，提供更好的时效性和稳定性。

数据湖构建服务搭配 Delta Lake 玩转 CDC 实时入湖

作者：嵩林

什么是 CDC

Change Data Capture(CDC)用来跟踪捕获数据源的数据变化，并将这些变化同步到目标存储(如数据湖或数据仓库)，用于数据备份或后续分析，同步过程可以是分钟/小时/天等粒度，也可以是实时同步。CDC 方案分为侵入式(intrusive manner)和非倾入性(non-intrusive manner)两种。

侵入式

侵入式方案直接请求数据源系统(如通过 JDBC 读取数据)，会给数据源系统带来性能压力。常见的方案如下:

- 最后更新时间(Last Modified)

源表需要有修改时间列，同步作业需要指定最后修改时间参数，表明同步某个时间点之后变更的数据。该方法不能同步删除记录的变更，同一条记录多次变更只能记录最后一次。

- 自增 id 列

源表需要有一个自增 id 列，同步作业需要指定上次同步的最大 id 值，同步上次之后新增的记录行。该方法也不能同步删除记录的变更，而且老记录的变更也无法感知。

非侵入式

非侵入性一般通过日志的方式记录数据源的数据变化(如数据库的 binlog)，源库需要开启 binlog 的功能。数据源的每次操作都会被记录到 binlog 中(如 insert/update/delete 等)，能够实时跟踪数据插入/删除/数据多次更新/DDL 操作等。

示例：

```
insert into table testdb.test values("hangzhou",1);
update testdb.test set b=2 where a="hangzhou";
update testdb.test set b=3 where a="hangzhou";
delete from testdb.test where a="hangzhou";
```

recordID	recordType	dbtable	fields	beforeImage	afterImage
1	INSERT	testdb.test	["a","b"]	{}	{"a":"hangzhou","b":"1"}
2	UPDATE	testdb.test	["a","b"]	{"a":"hangzhou","b":"1"}	{"a":"hangzhou","b":"2"}
3	UPDATE	testdb.test	["a","b"]	{"a":"hangzhou","b":"2"}	{"a":"hangzhou","b":"3"}
4	DELETE	testdb.test	["a","b"]	{"a":"hangzhou","b":"3"}	{}

通过将 binlog 日志有序的回放到目标存储中，从而实现对数据源的数据导出同步功能。

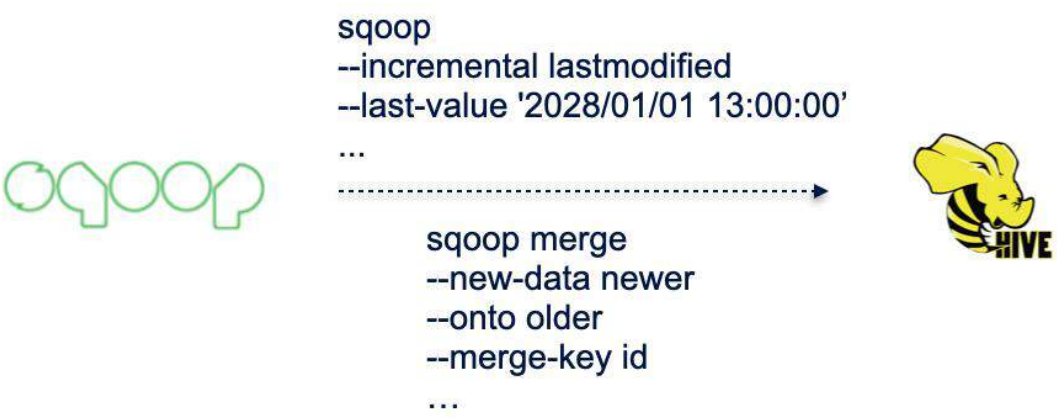
常见的 CDC 方案实现

开源常见的 CDC 方案实现主要有两种：

Sqoop 离线同步

sqoop 是一个开源的数据同步工具，它可以将数据库的数据同步到 HDFS/Hive 中，支持全量同步和增量同步，用户可以配置小时/天的调度作业来定时同步数据。

sqoop 增量同步是一种侵入式的 CDC 方案，支持 Last Modified 和 Append 模式。

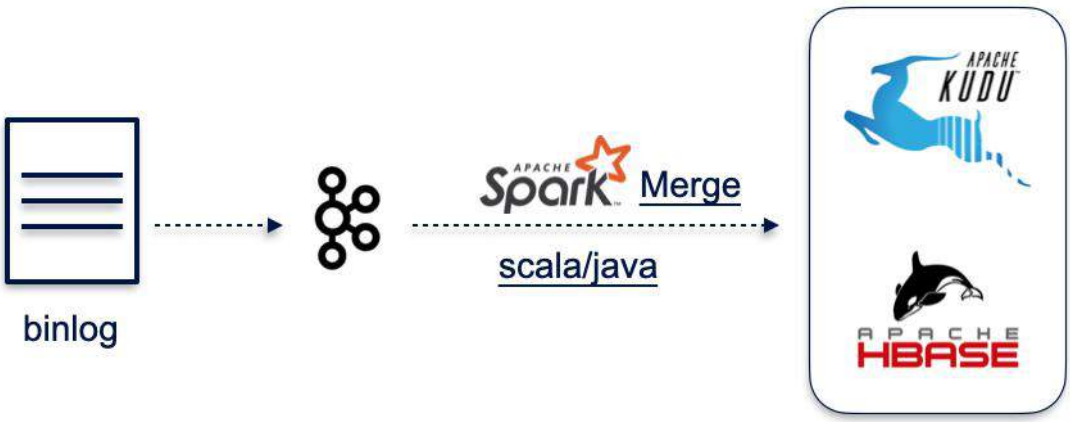


缺点：

- 直接 jdbc 请求源库拉取数据，影响源库性能
- 小时/天调度，实时性不高
- 无法同步源库的删除操作，Append 模式还不支持数据更新操作

binlog 实时同步

binlog 日志可以通过一些工具实时同步到 kafka 等消息中间件中，然后通过 Spark/Flink 等流引擎实时的回放 binlog 到目标存储(如 Kudu/HBase 等)。



缺点:

- Kudu/HBase 运维成本高
- Kudu 在数据量大的有稳定性问题，HBase 不支持高吞吐的分析
- Spark Streaming 实现回放 binlog 逻辑复杂，使用 java/scala 代码具有一定门槛

Streaming SQL+Delta Lake 实时入湖方案

前面介绍了两种常见的 CDC 方案，各自都有一些缺点。阿里云 E-MapReduce 团队提供了一种新的 CDC 解决方案，利用自研的 Streaming SQL 搭配 Delta Lake 可以轻松实现 CDC 实时入湖。这套解决方案同时通过阿里云最新发布的数据湖构建（Data Lake Formation，DLF）服务提供一站式的入湖体验。

Streaming SQL

Spark Streaming SQL 在 Spark Structured Streaming 之上提供了 SQL 能力，降低了实时业务开发的门槛，使得离线业务实时化更简单方便。

Spark Streaming SQL 支持的语法如下:

DDL	*CREATE TABLE/CREATE TABLE/CREATE SCAN/CREATE STREAM*
DML	*INSERT INTO/MERGE INTO*
DQL	*SELECT FROM/WHERE/GROUP BY/JOIN/UNION ALL*

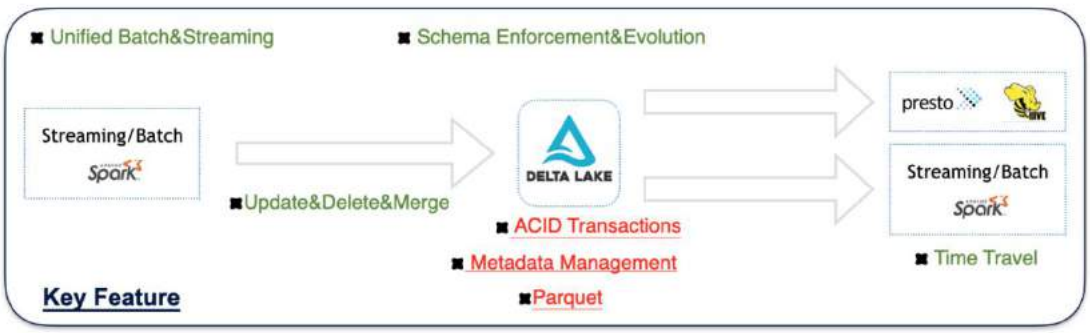
UDF	*TUMBLING/HOPPING/DELAY/SparkSQL UDF*
Data Source	*Delta Lake/Kafka/HBase/JDBC/Druid/Kudu/Alibaba Cloud(SLS/OTS/DataHub/...)*

下面以实时消费 SLS 为例：

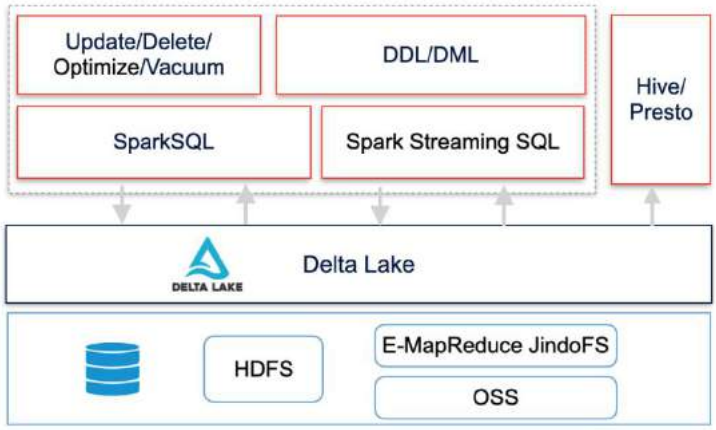
```
    # 创建 loghub 源表
    spark-sql> CREATE TABLE loghub_intput_tbl(content string)
             > USING loghub
             > OPTIONS
      > (...)

    # 创建 delta 目标表
    spark-sql> CREATE TABLE delta_output_tbl(content string)
       > USING delta
        > OPTIONS
       > (...);

    # 创建流式 SCAN
    spark-sql> CREATE SCAN loghub_table_intput_test_stream
      > ON loghub_intput_tbl
          > USING STREAM;

    # 将 loghub 源表数据插入 delta 目标表
    spark-sql> INSERT INTO delta_output_tbl SELECT content FROM loghub_table_i
ntput_test_stream;
```

Delta Lake

Delta Lake 是 Databricks 开源的一种数据湖格式，它在 parquet 格式之上，提供了 ACID 事务/元数据管理等能力，同时相比 parquet 具有更好的性能，能够支持更丰富的数据应用场景(如数据更新/schema 演化等)。

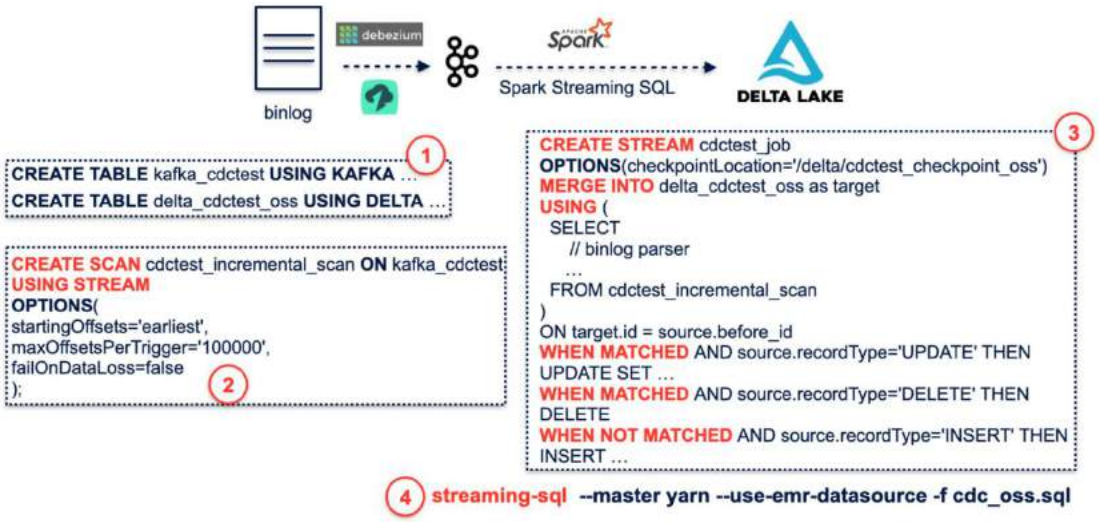


E-MapReduce 团队在开源 Delta Lake 基础上做了很多功能和性能的优化，如小文件合并 Optimize/DataSkipping/Zorder，SparkSQL/Streaming SQL/Hive/Presto 深度集成 Delta 等。



Streaming SQL+Delta Lake CDC 实时入湖

Spark Streaming SQL 提供了 Merge Into 的语法，搭配 Delta Lake 的实时写入能力，可以很方便的实现 CDC 实时入湖方案。



如上图所示，只需要 SQL 就能完成 CDC 实时入湖，细节步骤详见 E-MapReduce 文档。

阿里云最新发布的**数据湖构建**（Data Lake Formation，DLF）服务，提供了完整的一站式入湖解决方案。

云原生计算引擎

云原生计算引擎挑战与解决方案

作者：辰繁

云原生背景介绍与思考

图一是基于 ECS 底座的 EMR 架构，这是一套非常完整的开源大数据生态，也是近 10 年来每个数字化企业必不可少的开源大数据解决方案。主要分为以下几层：

- ECS 物理资源层，也就是 Iaas 层
- 数据接入层，例如实时的 Kafka，离线的 Sqoop
- 存储层，包括 HDFS 和 OSS，以及 EMR 自研的缓存加速 JindoFS
- 计算引擎层，包括熟知的 Spark，Presto、Flink 等这些计算引擎
- 数据应用层，如阿里自研的 DataWorks、PAI 以及开源的 Zeppelin，Jupyter

每一层都有比较多的开源组件与之对应，这些层级组成了最经典的大数据解决方案，也就是 EMR 的架构。我们对此有以下思考：

- 是否能够做到更好用的弹性，也就是客户可以完全按照自己业务实际的峰值和低谷进行弹性扩容和缩容，保证速度足够快，资源足够充足。
- 不考虑现有状况，看未来几年的发展方向，是否还需要支持所有的计算引擎和存储引擎。这个问题也非常实际，一方面是客户是否有能力维护这么多的引擎，另一方面是是否某些场景下用一种通用的引擎即可解决所有问题。举个例子说 Hive 和 Mapreduce，诚然现有的一些客户还在用 Hive on Mapreduce，而且规模也确实不小，但是未来 Spark 会是一个很好的替代品。

- 存储与计算分离架构，这是公认的未来大方向，存算分离提供了独立的扩展性，客户可以做到数据入湖，计算引擎按需扩容，这样的解耦方式会得到更高的性价比。

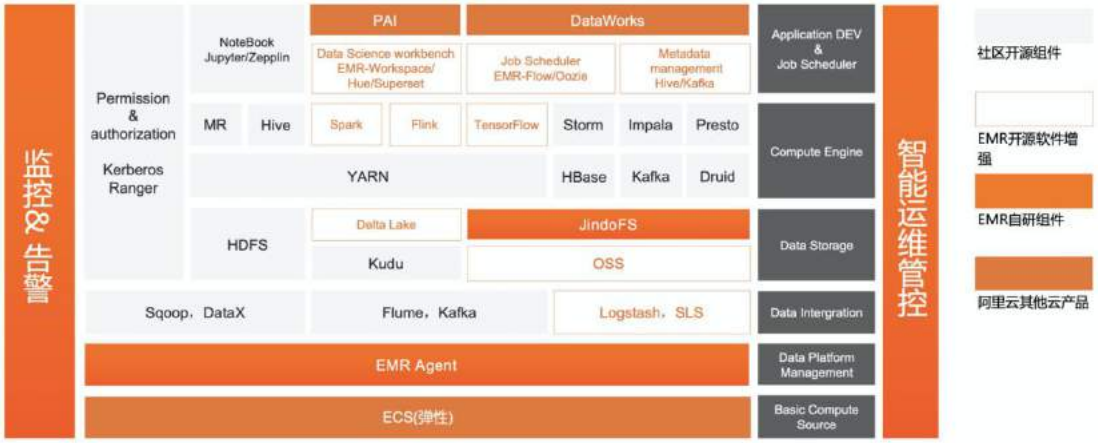


图 1　基于 ECS 的开源大数据解决方案

基于以上这些思考，我们考虑一下云原生的这个概念，云原生比较有前景的实现就是 Kubernetes，所以有时候我们一提到云原生，几乎就等价于是 Kubernetes。随着 Kubernetes 的概念越来越火，客户也对该技术充满了兴趣，很多客户已经把在线的业务搬到了 Kubernetes 之上。并且希望在这种类似操作系统上，建设一套统一的、完整的大数据基础架构。所以我们总结为以下几个特征：

- 希望能够基于 Kubernetes 来包容在线服务、大数据、AI 等基础架构，做到运维体系统一化。

- 存算分离架构，这个是大数据引擎可以在 Kubernetes 部署的前提，未来的趋势也都在向这个方向走。

- 通过 Kubernetes 的天生隔离特性，更好的实现离线与在线混部，达到降本增效目的。

- Kubernetes 生态提供了非常丰富的工具，我们可以省去很多时间搞基础运维工作，从而可以专心来做业务。

EMR 计算引擎 on ACK

图 2 是 EMR 计算引擎 on ACK 的架构。ACK 就是阿里云版本的 Kubernetes，在兼容社区版本的 API 同时，在本文中不会区分 ACK 和 Kubernetes 这两个词，可以认为代表同一个概念。

基于最开始的讨论，我们认为比较有希望的大数据批处理引擎是 Spark 和 Presto，当然我们也会随着版本迭代逐步的加入一些比较有前景的引擎。

EMR 计算引擎提供以 Kubernetes 为底座的产品形态，本质上来说是基于 CRD+Operator 的组合，这也是云原生最基本的哲学。我们针对组件进行分类，分为 service 组件和批处理组件，比如 Hive Metastore 就是 service 组件，Spark 就是批处理组件。

图中绿色部分是各种 Operator，技术层面在开源的基础上进行了多方面的改进，产品层面针对 ACK 底座进行了各方面的兼容，能够保证用户在集群中很方便的进行管控操作。右边的部分，包括 Log、监控、数据开发、ECM 管控等组件，这里主要是集成了阿里云的一些基础设施。我们再来看下边的部分：

- 引入了 JindoFS 作为 OSS 缓存加速层，做计算与存储分离的架构。
- 打通了现有 EMR 集群的 HDFS，方便客户利用已有的 EMR 集群数据。
- 引入 Shuffle Service 来做 Shuffle 数据的解耦，这也是 EMR 容器化区别于开源方案的比较大的亮点，之后会重点讲到。

这里明确一下，由于本身 Presto 是无状态的 MPP 架构，在 ACK 中部署是相对容易的事情，本文主要讨论 Spark on ACK 的解决方案。

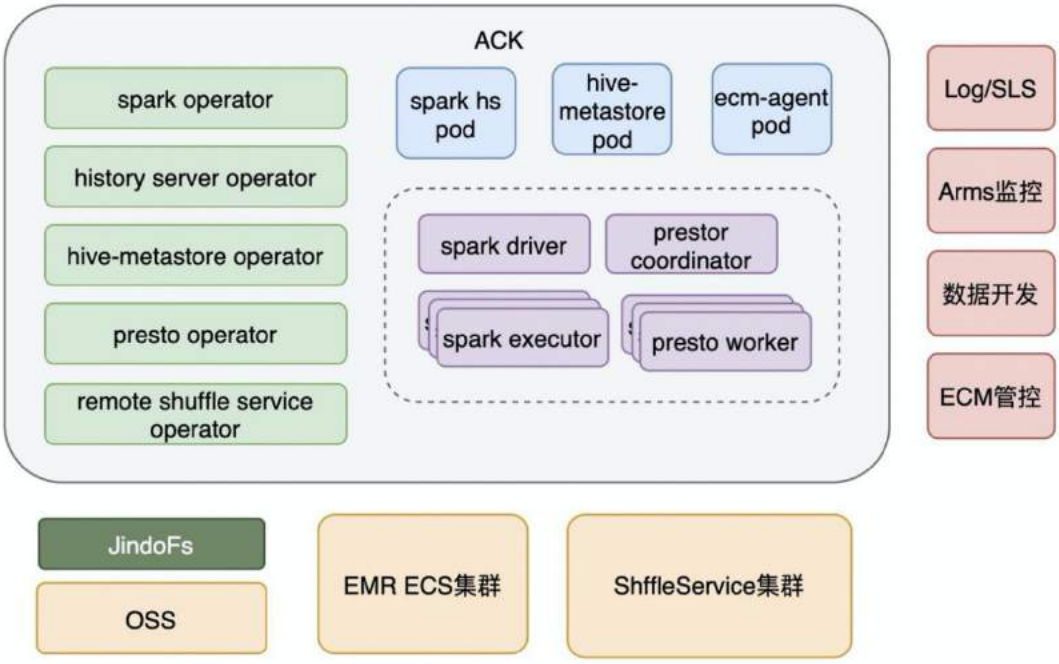


图 2 计算引擎 Kubernetes 化

Spark on Kubernetes 的挑战

整体看，Spark on Kubernetes 面临以下问题：

- 我个人认为最重要的，就是 Shuffle 的流程，按照目前的 Shuffle 方式，我们是没办法打开动态资源特性的。而且还需要挂载云盘，云盘面临着 Shuffle 数据量的问题，挂的比较大会很浪费，挂的比较小又支持不了 Shuffle Heavy 的任务。

- 调度和队列管理问题，调度性能的衡量指标是，要确保当大量作业同时启动时，不应该有性能瓶颈。作业队列这一概念对于大数据领域的同学应该非常熟悉，他提供了一种管理资源的视图，有助于我们在队列之间控制资源和共享资源。

- 读写数据湖相比较 HDFS，在大量的 Rename，List 等场景下性能会有所下降，同时 OSS 带宽也是一个不可避免的问题。

针对以上问题，我们分别来看下解决方案。

Spark on Kubernetes 的解决方案

Remote Shuffle Service 架构

Spark Shuffle 的问题，我们设计了 Shuffle 读写分离架构，称为 Remote Shuffle Service。首先探讨一下为什么 Kubernetes 不希望挂盘呢，我们看一下可能的选项：

- 如果用是 Docker 的文件系统，问题是显而易见的，因为性能慢不说，容量也是极其有限，对于 Shuffle 过程是十分不友好的。

- 如果用 Hostpath，熟悉 Spark 的同学应该知道，是不能够启动动态资源特性的，这个对于 Spark 资源是一个很大的浪费，而且如果考虑到后续迁移到 Serverless K8s，那么从架构上本身就是不支持 Hostpath 的。

- 如果是 Executor 挂云盘的 PV，同样道理，也是不支持动态资源的，而且需要提前知道每个 Executor 的 Shuffle 数据量，挂的大比较浪费空间，挂的小 Shuffle 数据又不一定能够容纳下。

所以 Remote Shuffle 架构针对这一痛点、对现有的 Shuffle 机制有比较大的优化，图 3 中间有非常多的控制流，我们不做具体的讨论，具体架构详见文章《Serverless Spark 的弹性利器 – EMR Shuffle Service》。主要来看数据流，对于 Executor 所有的 Mapper 和 Reducer，也就是图中的蓝色部分是跑在了 K8s 容器里，中间的架构是 Remote Shuffle Service，蓝色部分的 Mapper 将 Shuffle 数据远程写入 service 里边，消除了 Executor 的 Task 对于本地磁盘的依赖。Shuffle Service 会对来自不同 Mapper 的同一 partition 的数据进行 merge 操作，然后写入到分布式文件系统中。等到 Reduce 阶段，Reducer 通过读取顺序的文件，可以很好的提升性能。这套系统最主要的实现难点就是控制流的设计，还有各方面的容错，数据去重，元数据管理等等工作。

简而言之，我们总结为以下 3 点：

- Shuffle 数据通过网络写出，中间数据计算与存储分离架构

- DFS 2 副本，消除 Fetch Failed 引起的重算，Shuffle Heavy 作业更加稳定

- Reduce 阶段顺序读磁盘，避免现有版本的随机 IO，大幅提升性能

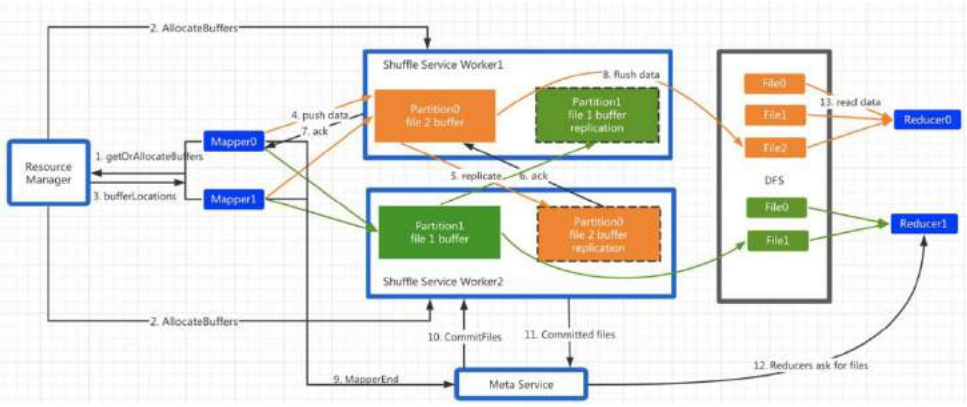


图 3 Remote Shuffle Service 架构图

Remote Shuffle Service 性能

我们在这里展示一下关于性能的成绩，图 4 是 Terasort 的 Benchmark 成绩。之所以选取 Terasrot 这种 workload 来测试，是因为他只有 3 个 stage，而且是一个大 Shuffle 的任务，大家可以非常有体感的看出关于 Shuffle 性能的变化。左边图，蓝色是 Shuffle Service 版本的运行时间，橙色的是原版 Shuffle 的运行时间。我们观察有 2T，4T，10T 的数据，可以看到随着数据量越来越大，Shuffle Service 的优势就越明显。右图观察，作业的性能提升主要体现在了 Reduce 阶段，可以看到 10T 的 Reduce Read 从 1.6 小时下降到了 1 小时。原因前边已经解释的很清楚了，熟悉 spark shuffle 机制的同学知道，原版的 sort shuffle 是 M*N 次的随机 IO，在这个例子中，M 是 12000，N 是 8000，而 Remote Shuffle 就只有 N 次顺序 IO，这个例子中是 8000 次，所以这是提升性能的根本所在。

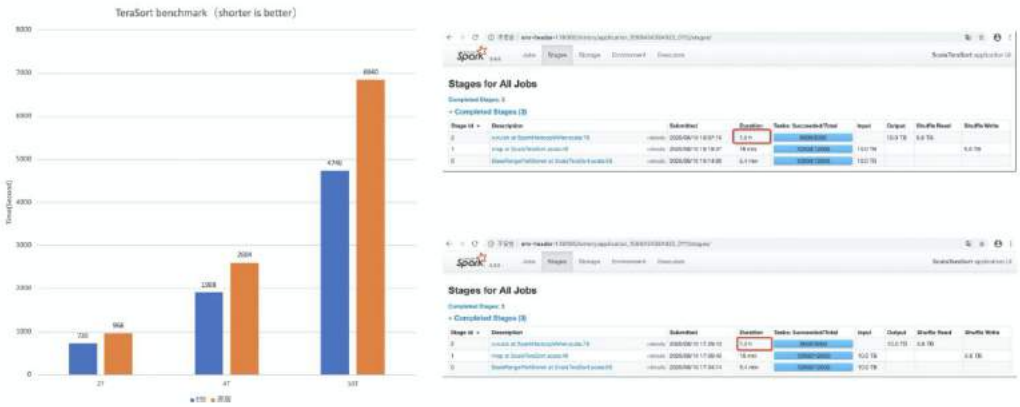


图 4 Remote Shuffle Service 性能 Benchmark

其他方面的重点优化

这里讲一下 EMR 在其他方面做得优化

- 调度性能优化，我们解决了开源的 Spark Operator 的一些不足，对于 Executor pod 的很多配置 Spark Operator 把他做到了 Webhook 里边，这个对调度来说是十分不友好的，因为相当于在 API Server 上绕了一圈，实际测试下来性能损耗很大。我们把这些工作直接做到 spark 内核里边，这样避免了这方面的调度性能瓶颈。然后从调度器层面上，我们保留了两种选择给客户，包括 ACK 提供的 SchedulerFrameworkV2 实现方式和 Yunicorn 实现方式。

- 读写 OSS 性能优化，我们这里引入了 JindoFS 作为缓存，解决带宽问题，同时 EMR 为 OSS 场景提供了 Jindo Job Committer，专门优化了 job commit 的过程，大大减少了 Rename 和 List 等耗时操作。

- 针对 Spark 本身，EMR 积累了多年的技术优势，也在 TPCDS 官方测试中，取得了很好的成绩，包括 Spark 性能、稳定性，还有 Delta lake 的优化也都有集成进来。

- 我们提供了一站式的管控，包括 Notebook 作业开发，监控日志报警等服务，还继承了 NameSpace 的 ResourceQuota 等等。

总体来说，EMR 版本的 Spark on ACK，无论在架构上、性能上、稳定性上、易用性方面都有着很大的提升。

Spark 云原生后续展望

从我的视角来观察，Spark 云原生容器化后续的方向，一方面是达到运维一体化，另一方面主要希望做到更好的性价比。我们总结为以下几点：

- 可以将 Kubernetes 计算资源分为固定集群和 Serverless 集群的混合架构，固定集群主要是一些包年包月、资源使用率很高的集群，Serverless 是做到按需弹性。

- 可以通过调度算法，灵活的把一些 SLA 不高的任务调度到 Spot 实例上，就是支持抢占式 ECI 容器，这样可以进一步降低成本。

- 由于提供了 Remote Shuffle Service 集群，充分让 Spark 在架构上解耦本地盘，只要 Executor 空闲就可以销毁。配合上 OSS 提供的存算分离，必定是后续的主流方向。

- 对于调度能力，这方面需要特别的增强，做到能够让客户感受不到性能瓶颈，短时间内调度起来大量的作业。同时对于作业队列管理方面，希望做到更强的资源控制和资源共享。

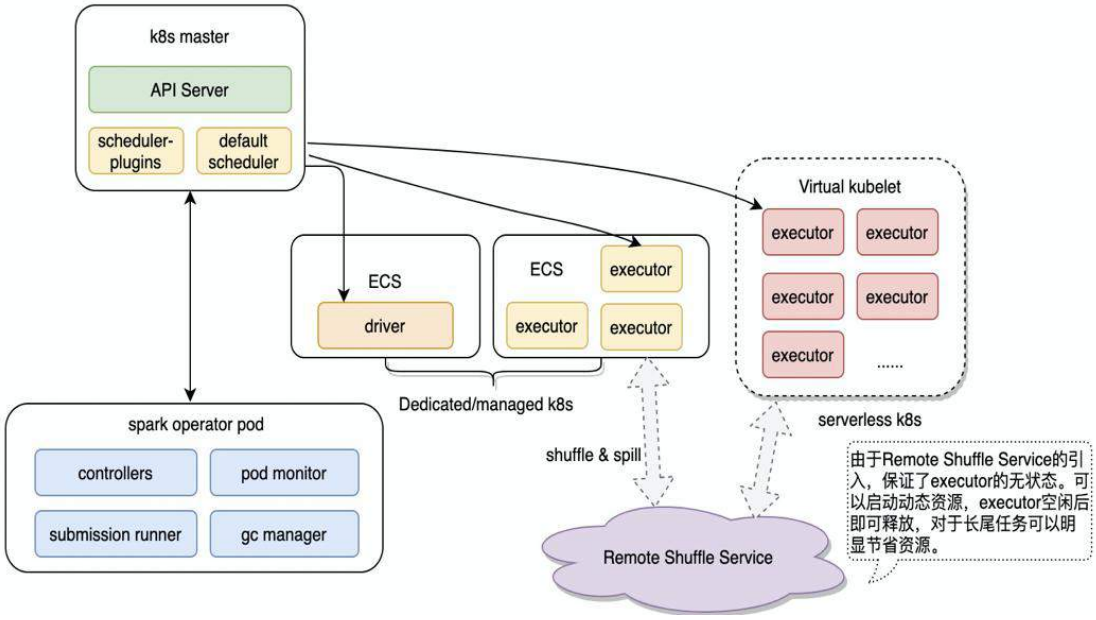


图 5 Spark on Kubernetes 混合架构

大数据云原生的落地是十分具有挑战性的，EMR 团队也会和社区和合作伙伴一起打造技术和生态，我们的愿景是：

- 存算分离，按需扩展

- 极致弹性，随用随得

- 运维闭环，高性价比

Serverless Spark 的弹性利器 - EMR Shuffle Service

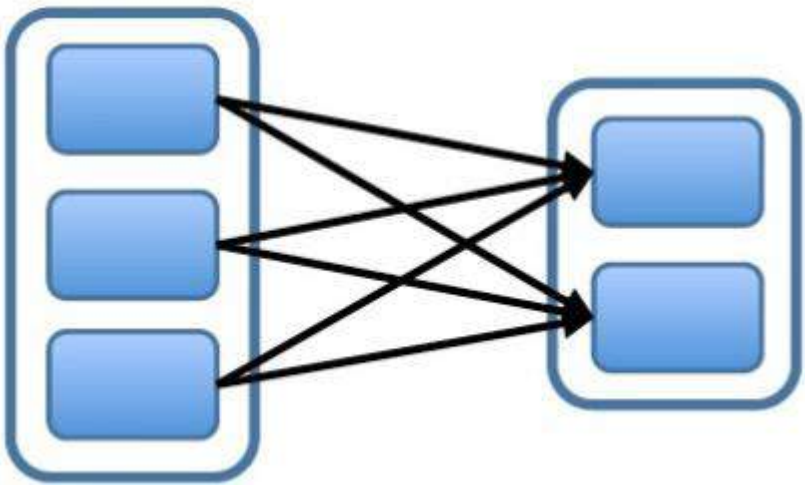
作者：一锤

背景与动机

计算存储分离下的刚需

计算存储分离是云原生的重要特征。通常来讲，计算是 CPU 密集型，存储是 IO 密集型，他们对于硬件配置的需求是不同的。在传统计算存储混合的架构中，为了兼顾计算和存储，CPU 和存储设备都不能太差，因此牺牲了灵活性，提高了成本。在计算存储分离架构中，可以独立配置计算机型和存储机型，具有极大的灵活性，从而降低成本。

存储计算分离是新型的硬件架构，但以往的系统是基于混合架构设计的，必须进行改造才能充分利用分离架构的优势，甚至不改造的话会报错，例如很多系统假设本地盘足够大，而计算节点本地盘很小；再例如有些系统在 Locality 上的优化在分离架构下不再适用。Spark Shuffle 就是一个典型例子。众所周知，Shuffle 的过程如下图所示。



每个 mapper 把全量 shuffle 数据按照 partitionId 排序后写本地文件，同时保存索引文件记录每个 partition 的 offset 和 length。reduce task 去所有的 map 节点拉取属于自己的 shuffle 数据。大数据场景 T 级别的 shuffle 数据量很常见，这就要求本地磁盘足够大，导致了跟计算存储分离架构的冲突。因此，需要重构传统的 shuffle 过程，把 shuffle 数据卸载到存储节点。

稳定性和性能

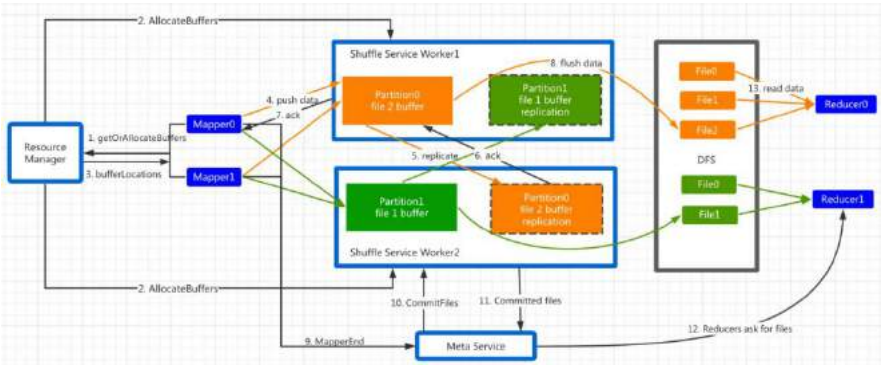
除了计算存储分离架构下的刚需，在传统的混合架构下，目前的 shuffle 实现也存在重要缺陷：大量的随机读写和小数据量的网络传输。考虑 1000 mapper * 2000 reducer 的 stage，每个 mapper 写 128M shuffle 数据，则属于每个 reduce 的数据量约为 64k。从 mapper 的磁盘角度，每次磁盘 IO 请求随机读 64K 的数据；从网络的角度，每次网络请求传输 64k 的数据：都是非常糟糕的 pattern，导致大量不稳定和性能问题。因此，即使在混合架构下，重构 shuffle 也是很必要的工作。

EMR Shuffle Service 设计

基于以上的动机，阿里云 EMR 团队设计开发了 EMR Shuffle Service 服务(以下称 ESS)，同时解决了计算存储分离和混合架构下的 shuffle 稳定性和性能问题。

整体设计

ESS 包含三个主要角色：Master，Worker，Client。其中 Master 和 Worker 构成服务端，Client 以不侵入方式集成到 Spark 里。Master 的主要职责是资源分配和状态管理；Worker 的主要职责是处理和存储 shuffle 数据；Client 的主要职责是缓存和推送 shuffle 数据。整体流程如下所示(其中 ResourceManager 和 MetaService 是 Master 的组件)：



ESS 采用 Push Style 的 shuffle 模式，每个 Mapper 持有一个按 Partition 分界的缓存区，Shuffle 数据首先写入缓存区，每当某个 Partition 的缓存满了即触发 PushData。

在 PushData 触发之前 Client 会检查本地是否有 PartitionLocation 信息，该 Location 规定了每个 Partition 的数据应该推送的 Worker 地址。若不存在，则向 Master 发起 getOrAllocateBuffers 请求。Master 收到后检查是否已分配，若未分

配则根据当前资源情况选择互为主从的两个 Worker 并向他们发起 AllocateBuffer 指令。Worker 收到后记录 Meta 并分配内存缓存。Master 收到 Worker 的 ack 之后把主副本的 Location 信息返回给 Client。

Client 开始往主副本推送数据。主副本 Worker 收到请求后，把数据缓存到本地内存，同时把该请求以 Pipeline 的方式转发给从副本。从副本收到完整数据后立即向主副本发 ack，主副本收到 ack 后立即向 Client 回复 ack。

为了不 block PushData 的请求，Worker 收到 PushData 请求后会先塞到一个 queue 里，由专有的线程池异步处理。根据该 Data 所属的 Partition 拷贝到事先分配的 buffer 里，若 buffer 满了则触发 flush。ESS 支持多种存储后端，包括 DFS 和 local。若后端是 DFS，则主从副本只有一方会 flush，依靠 DFS 的双副本保证容错；若后端是 Local，则主从双方都会 flush。

在所有的 Mapper 都结束后，Master 会触发 StageEnd 事件，向所有 Worker 发送 CommitFiles 请求，Worker 收到后把属于该 Stage 的 buffer 里的数据 flush 到存储层，close 文件，并释放 buffer。Master 收到所有 ack 后记录每个 partition 对应的文件列表。若 CommitFiles 请求失败，则 Master 标记此 Stage 为 DataLost。

在 Reduce 阶段，reduce task 首先向 Master 请求该 Partition 对应的文件列表，若返回码是 DataLost，则触发 Stage 重算或直接 abort 作业。若返回正常，则直接读取文件数据。

ESS 的设计要点，一是采用 PushStyle 的方式做 shuffle，避免了本地存储，从而适应了计算存储分离架构；二是按照 reduce 做了聚合，避免了小文件随机读写和小数据量网络请求；三是做了两副本，提高了系统稳定性。

容错

除了双副本和 DataLost 检测，ESS 在容错上做了很多事情保证正确性。

PushData 失败

当 PushData 失败次数(Worker 挂了，网络繁忙，CPU 繁忙等)超过 MaxRetry 后，Client 会给 Master 发消息请求新的 Partition Location，此后本 Client 都会使用新的 Location 地址。

若 Revive 是因为 Client 端而非 Worker 的问题导致，则会产生同一个 Partition 数据分布在不同 Worker 上的情况，Master 的 Meta 组件会正确处理这种情形。若发生 WorkerLost，则会导致大量 PushData 同时失败，此时会有大量同一 Partition 的 Revive 请求打到 Master。为了避免给同一个 Partition 分配过多的 Location，Master 保证仅有一个 Revive 请求真正得到处理，其余的请求塞到 pending queue 里，待 Revive 处理结束后返回同一个 Location。

WorkerLost

当发生 WorkerLost 时，对于该 Worker 上的副本数据，Master 向其 peer 发送 CommitFile 的请求，然后清理 peer 上的 buffer。若 Commit Files 失败，则记录该 Stage 为 DataLost; 若成功，则后续的 PushData 通过 Revive 机制重新申请 Location。

数据冗余

Speculation task 和 task 重算会导致数据重复。解决办法是每个 PushData 的数据片里 encode 了所属的 mapId，attemptId 和 batchId，并且 Master 为每个 map task 记录成功 commit 的 attemtpId。read 端通过 attemptId 过滤不同的 attempt 数据，并通过 batchId 过滤同一个 attempt 的重复数据。

ReadPartition 失败

在 DFS 模式下，ReadPartition 失败会直接导致 Stage 重算或 abort job。在 Local 模式，ReadPartition 失败会触发从 peer location 读，若主从都失败则触发 Stage 重算或 abort job。

多 backend 支持

ESS 目前支持 DFS 和 Local 两种存储后端。

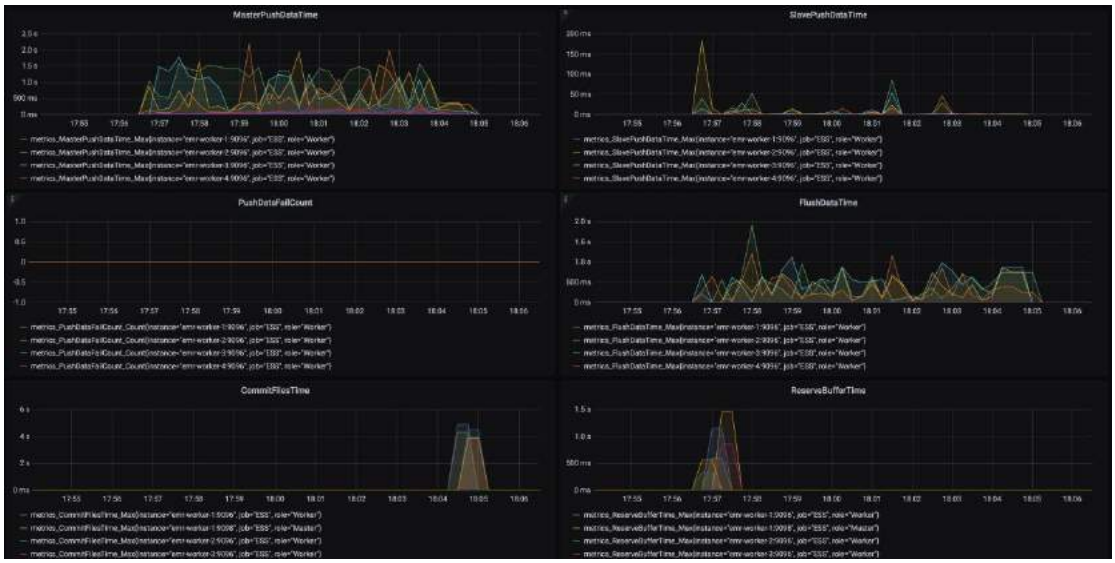
跟 Spark 集成

ESS 以不侵入 Spark 代码的方式跟 Spark 集成，用户只需把我们提供的 Shuffle Client jar 包配置到 driver 和 client 的 classpath 里，并加入以下配置即可切换到 ESS 方式：

```
spark.shuffle.manager=org.apache.spark.shuffle.ess.EssShuffleManager
```

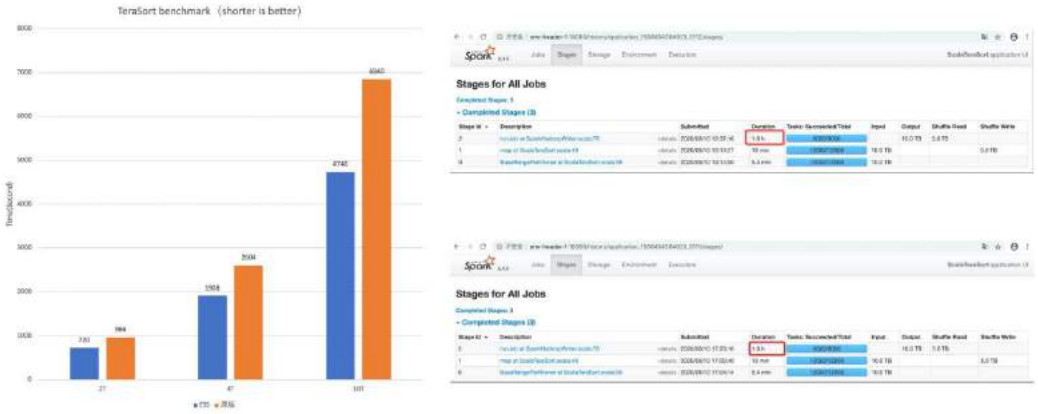
监控报警

我们对 ESS 服务端进行了较为详尽的监控报警并对接了 Prometheus 和 Grafana，如下所示：

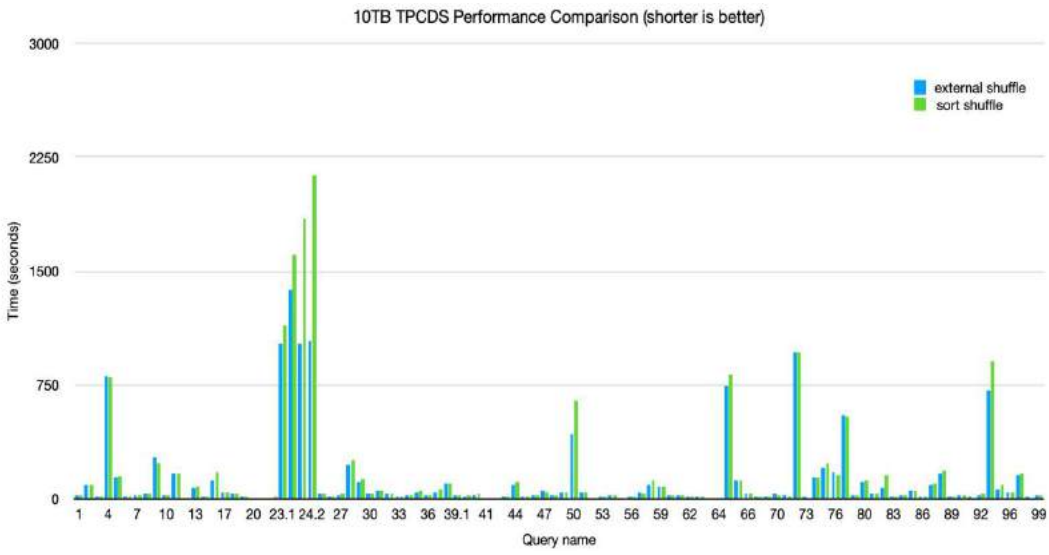


性能数字

TeraSort 的性能数字如下(2T，4T，10T 规模)：



10T 规模 TPC-DS 的性能数字如下：



后续

我们后续会持续投入，集中在产品化、极致性能、池化等方向，欢迎大家使用！

数据湖治理

数据湖开发治理平台 DataWorks

作者：涵康

数据湖的定义

wikipedia 中对于数据湖的定义是：“A data lake is a system or repository of data stored in its natural/raw format, usually object blobs or files. A data lake is usually a single store of all enterprise data including raw copies of source system data and transformed data used for tasks such as reporting, visualization, advanced analytics and machine learning.”

可见数据湖是一个通用的数据存储，通用到可以存储任意类型的数据。

数据湖要考虑的首要问题

从定义看，一块 u 盘即符合数据湖的定义。u 盘可以是数据湖，oss 可以是数据湖，hdfs、盘古也可以是数据湖。它们均严格的符合数据湖的定义。作为企业的数据湖技术选型第一个需要考虑的问题就是：采用什么样的存储介质或存储系统作为自己的数据湖解决方案。众所周知，不同的存储介质或存储系统有不同的优势和劣势。比如：有的存储系统随机读取的响应时间更好、有的系统批量读取的吞吐量更好、有的系统存储成本更低、有的系统扩展性更好、有的系统结构化数据组织得更高效...相应的，这些提到的各个指标中有些恰恰是有些存储所不擅长的，如何享有所有存储系统的优势、规避所有存储系统的劣势变成了云上数据湖服务要考虑的首要问题。

要解决这个矛盾的问题，在理论上是不可能一劳永逸的。聪明的做法是对上提供一个逻辑上的存储解决方案，然后让需要不同访问特点的数据灵活地在各种底层存储系统中迁移。通过便捷的数据迁移（、以及数据格式转化）的能力， 来充分发挥出各个存储系统的优势。结论：成熟的数据湖一定是一个逻辑上的存储系统，它的底层是多个各种类型的存储系统所组成。

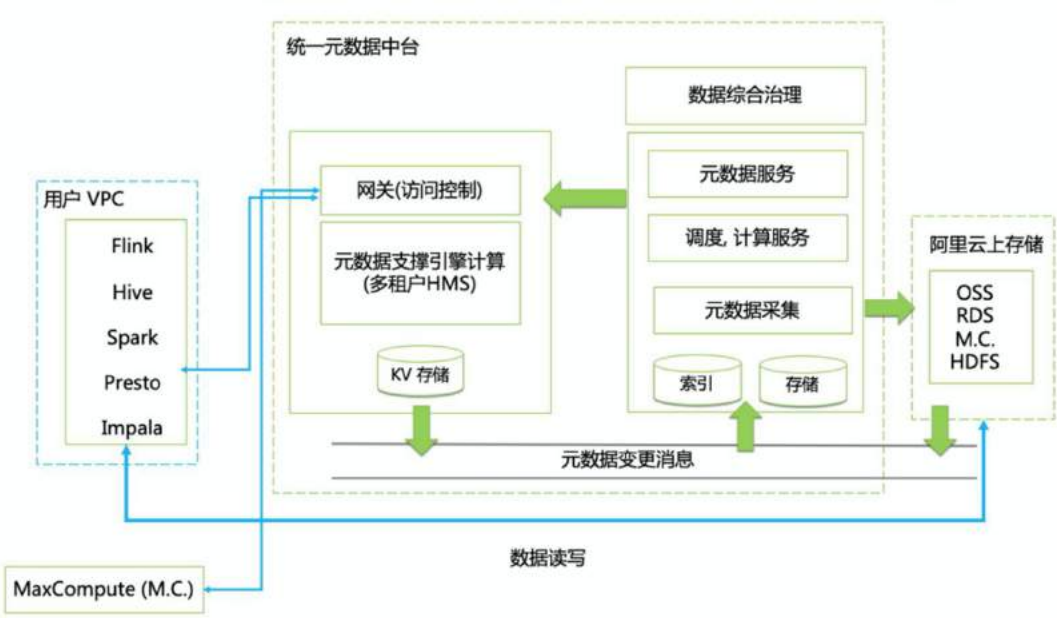
数据湖要解决的三大问题

元数据管理、数据集成、数据开发是数据湖需要解决的三大问题，阿里云的 DataWorks 作为一个通用的大数据平台，除了很好的解决了数仓场景的各类问题，也同样解决了数据湖场景中的核心痛点。

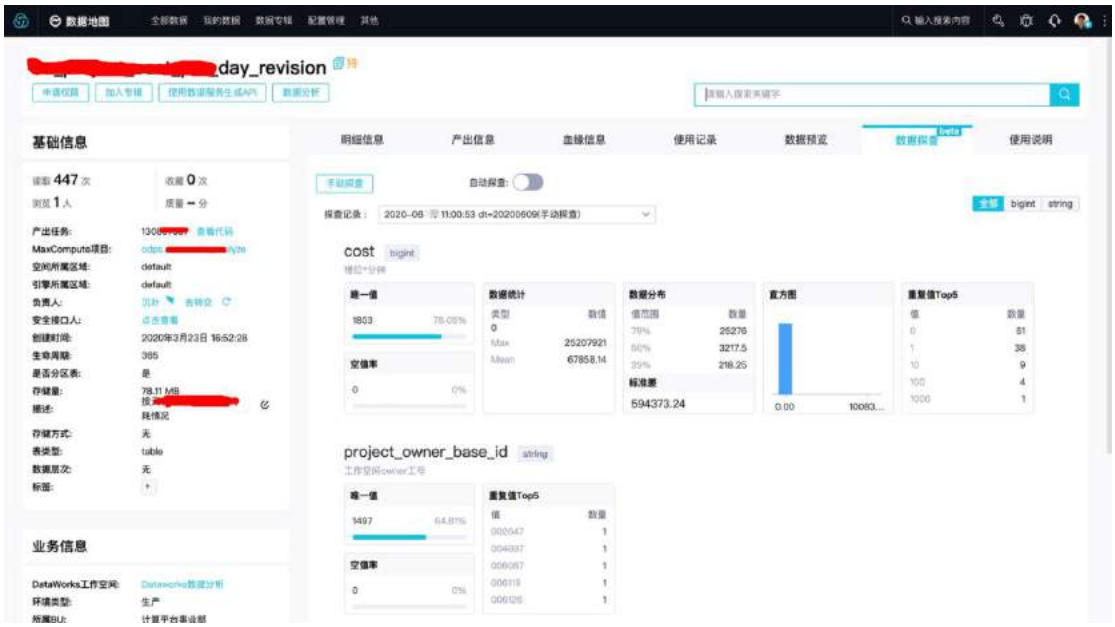
元数据管理

用户的湖上数据需要有个统一集中的管理能力，这就成了数据湖的第一个核心能力。DataWorks 的数据治理能力便是用来解决数据湖中的各类存储系统的元数据管理的。目前它管理了云上 11 中数据源的元数据。涵盖 OSS、EMR、MaxCompute、Hologres、mysql、PostgreSQL、SQL Server、Oracle、AnalyticDB for PostgreSQL、AnalyticDB for MySQL 2.0、AnalyticDB for MySQL 3.0 等云上主要数据源类型的元数据管理。功能上涵盖元数据采集、存储检索、在线元数据服务、数据预览、分类打标、数据血缘、数据探查、影响分析、资源优化等能力。

技术的宏观架构如图：



产品形态如图：

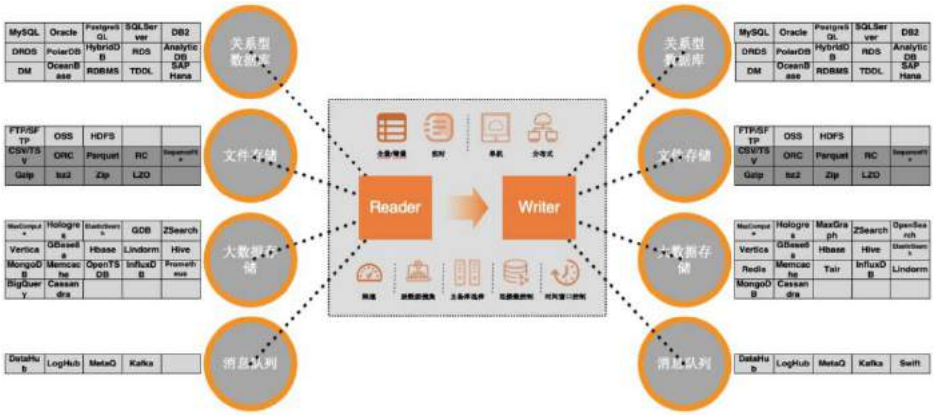


数据集成

数据湖中的数据管理起来之后，就会面临数据在各个存储系统中迁移和转化的能力。为此 DataWorks 的数据集成能力可以做到 40 种类常见数据源的导入导出及格式转化的能力，同时覆盖了离线和实时两大同步场景，以及可以解决对外对接时的复杂网络场景。

数据集成核心能力：

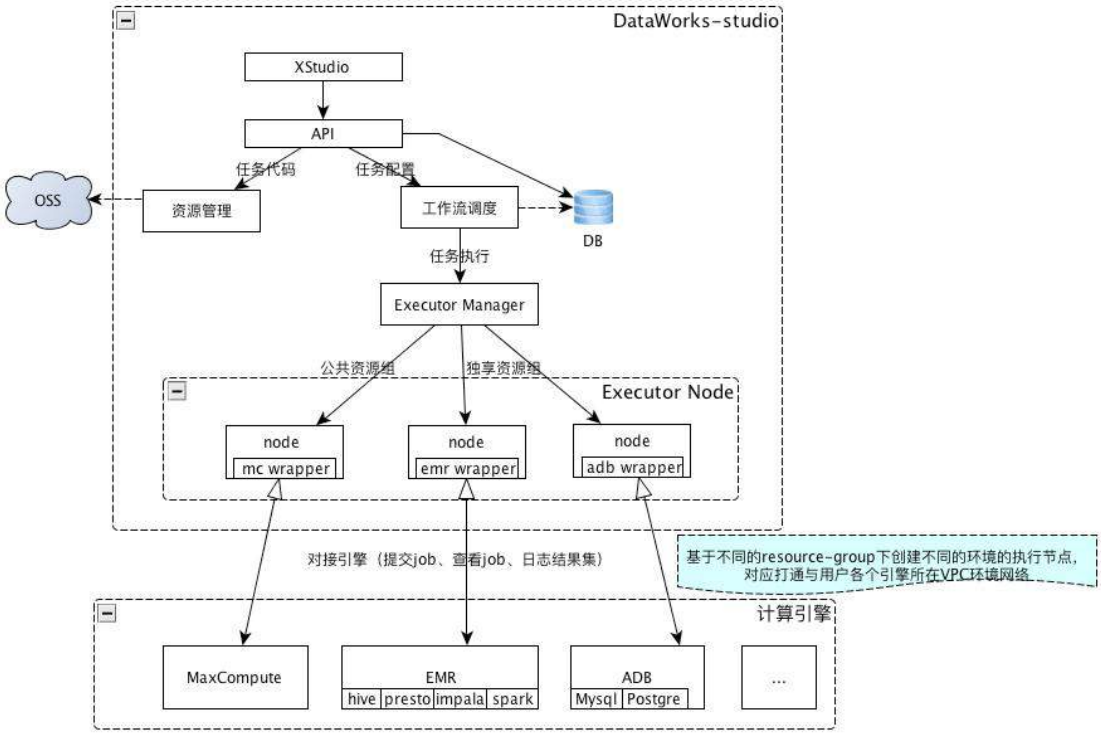
离线同步功能：



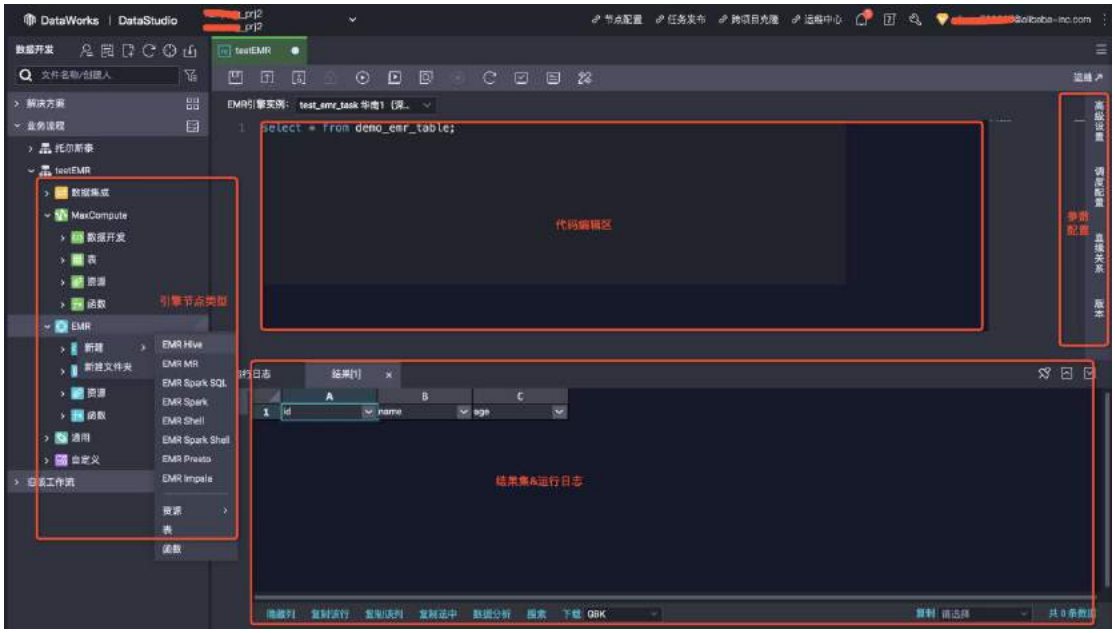
实时同步功能：

数据开发

解决了数据湖的存储管理和数据迁移问题后，接下来就是如何让数据湖中的数据更好的赋能业务。这就需要引入各类计算引擎，计算平台事业部拥有丰富的各类计算引擎，有开源体系的 spark、presto、hive、flink，还有自研的 MaxCompute、Hologres，这里的挑战在于如何方便的发挥各类引擎的长处，让湖中的数据能够被各类引擎访问和计算。为此 DataWorks 提供了便捷的数据迁移方式（方便数据在各类引擎中流转穿梭）、提供一站式的数据开发环境，从即席查询到周期的 etl 开发，DataWorks 提供了各个计算引擎的统一计算任务的开发和运维能力。



数据开发产品：



在解决了数据湖底层的存储系统差异的难题后，提供了完备的湖上元数据管理、数据治理、数据迁移转换、数据计算的全流程能力。让阿里云上的数据湖更好的给客户发挥出业务价值。

加入数据湖钉群
进行更多技术交流

阿里云开发者“藏经阁”
海量免费电子书下载