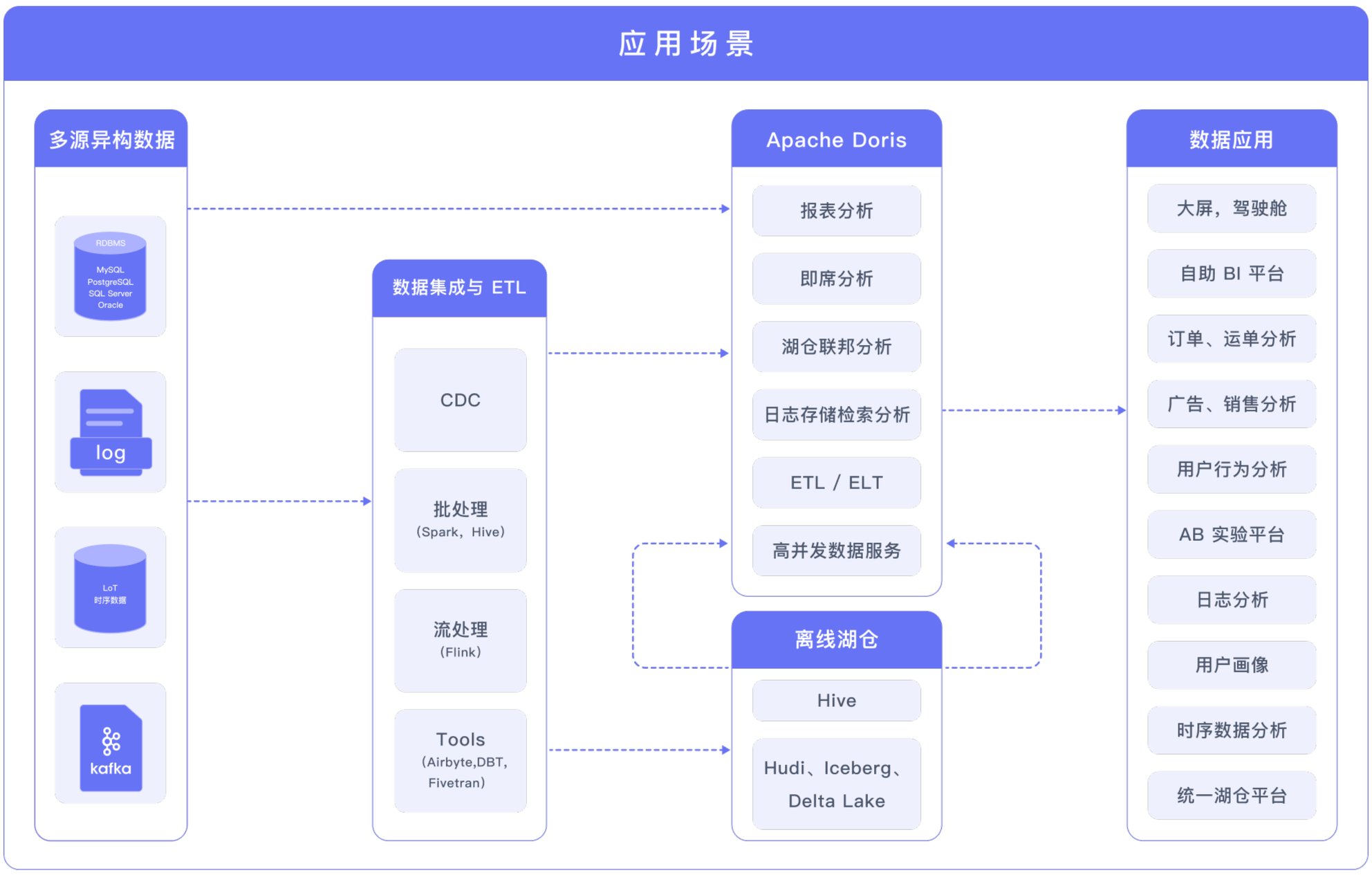
Apache Doris 精选用户案例集

第 2 期

DORIS × SELECTDB

飞轮科技出品

Apache Doris – 最受用户认可的开源实时数据仓库



核心优势

高效	极速的分析性能	高效的数据更新	丰富的数据导入	极致弹性与存算分离
简单	高可用与高可靠	多租户管理	易用易管理	
统一	半结构化数据分析	湖仓一体		

社区用户规模

4000+ 企业

Apache Doris 在中国乃至全球范围内都拥有着广泛的用户群体，目前已经在全球超过 4000 家企业的生产环境中得到应用。

30000+ 开发者

截止目前，Apache Doris 的用户社群已聚集开发者超 3 万人。

社区活跃度

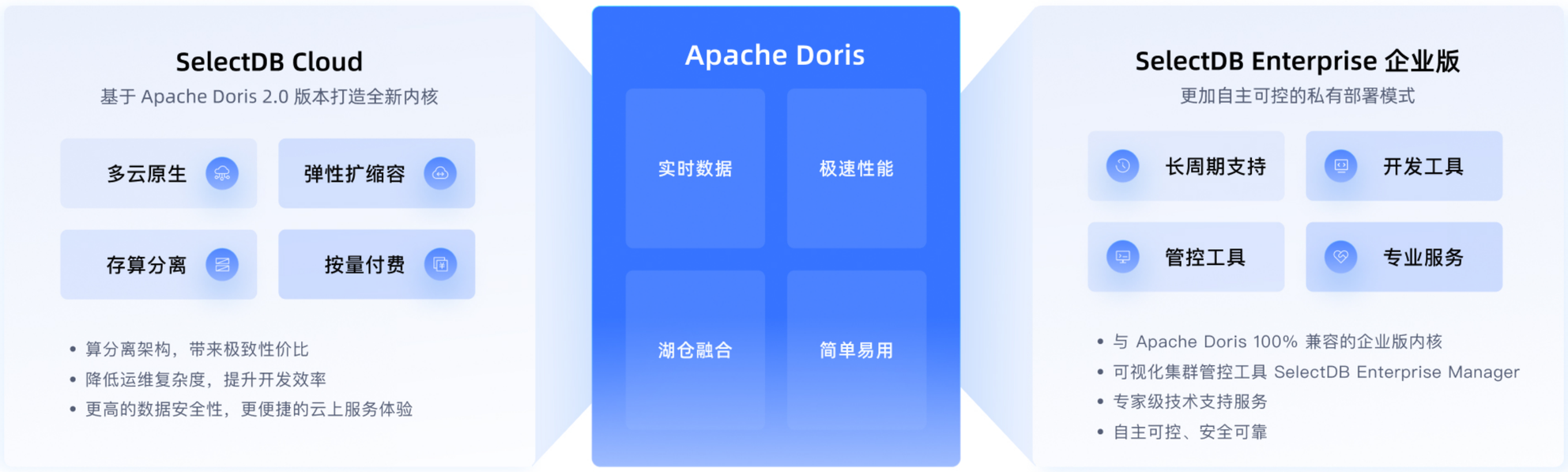
600+ 贡献者

活跃贡献者数连续数月稳居全球大数据开源项目排行榜第一。

已成为全国数据库和大数据领域最为活跃的开源项目之一。

SelectDB – 面向实时分析的现代化数据仓库

基于 Apache Doris 构建，支持大规模实时数据上的极速查询分析。



SelectDB Cloud

全托管的实时数仓服务，多云原生。公有云交付，目前已在阿里云、腾讯云、华为云、AWS 上开通。

SelectDB Enterprise

自管理私有化软件，部署在 IDC、私有云/专有云 VPC 上可运行在裸金属服务器、虚拟机、K8S。

产品特性

实时数据
数据的实时导入与实时可见

极速性能
多种查询负载上都拥有极速性能

湖仓融合
数据湖的开放性与数据仓库的高性能兼得

简单易用
提升开发体验和降低运维复杂度

主要应用场景

用户行为与画像分析

日志管理与分析

实时报表与实时决策

交互式探查

混合云和多云部署

support@selectdb.com

400-092-6099

北京市海淀区中关村街道知春路甲 48-1 号盈都大厦 A 座 9 层 9B

联系我们

目录

01

招商信诺人寿基于 Apache Doris 统一 OLAP 技术栈实践 01

02

河北幸福消费金融基于 Apache Doris 构建实时数仓，查询提速 400 倍！ 08

03

Apache Doris 在金融壹账通指标中台的应用实践 19

04

星云零售信贷基于 Apache Doris 的 OLAP 演进之路 29

05

Apache Doris 遇上大模型：腾讯音乐基于大模型 + OLAP 构建智能数据服务平台 38

06

Apache Doris 助力中国联通万亿日志数据分析提速 10 倍 55

07

查询平均提速 700%，奇安信基于 Apache Doris 升级日志安全分析系统 63

08

秒级数据写入，毫秒查询响应，天眼查基于 Apache Doris 构建统一实时数仓 72

招商信诺人寿

基于 Apache Doris 统一 OLAP 技术栈实践

作者：招商信诺大数据平台研发团队

导读：当前，大数据、人工智能、云计算等技术应用正在推动保险科技发展，加速保险行业数字化进程。在这一背景下，招商信诺不断探索如何将多元数据融合扩充，以赋能代理人掌握更加详实的用户线索，并将智能分析贯穿业务全链路，实现对用户、产品、场景策略的全面洞察与闭环迭代。本文将详细介绍招商信诺在大数据基础建设方面的探索之旅，从最初为线报表、Ad-hoc 分析提供服务的 OLAP 引擎，逐步发展至基于 Apache Doris 构建的统一实时数据仓库，**通过一套架构实现各业务领域的多元数据实时分析与融合统一管理**，最终实现保险一线业务降本增收的目标。

招商信诺人寿是由招商银行与信诺集团中外合资的寿险公司，为企业和个人提供涵盖保险保障、健康管理、财富规划等产品及服务。目前，招商信诺已累积服务客户超千万、完成理赔客户超百万，并凭借一站式便捷的健康管理服务、可灵活配置“定制化”的保险方案获得广大用户的持续选择与信赖。

面对全球数据量爆炸性增长的趋势，数据的时效性与准确性对企业精细化运营越来越重要。我们希望通过数据能够快速感知客户行为、定位客户问题、高效匹配用户所需的产品与服务，以达到精细化业务营销、拓宽可保边界等目标。

随着业务不断拓展、分析场景逐渐多元化，业务分析师的要求也变得更为复杂，不仅要求数仓能够快速开发数据报表，还需要实现流批一体、湖仓一体、多元化数据类型的统一分析与管理。在大数据基础建设中，这些融合统一的特性变得至关重要。在这样的背景下，我们持续升级与改进数仓架构，从最初仅支持 BI 报表、数据大屏的一代架构到采用多个系统和组件提供数据服务的二代架构，再到如今新一代统一实时数据仓库，**通过 Apache Doris 一套组件实现了架构的简化、技术栈的统一、数据的统一管理与分析**，不仅提升了数据处理效率，并且满足了更多样化的数据分析需求。

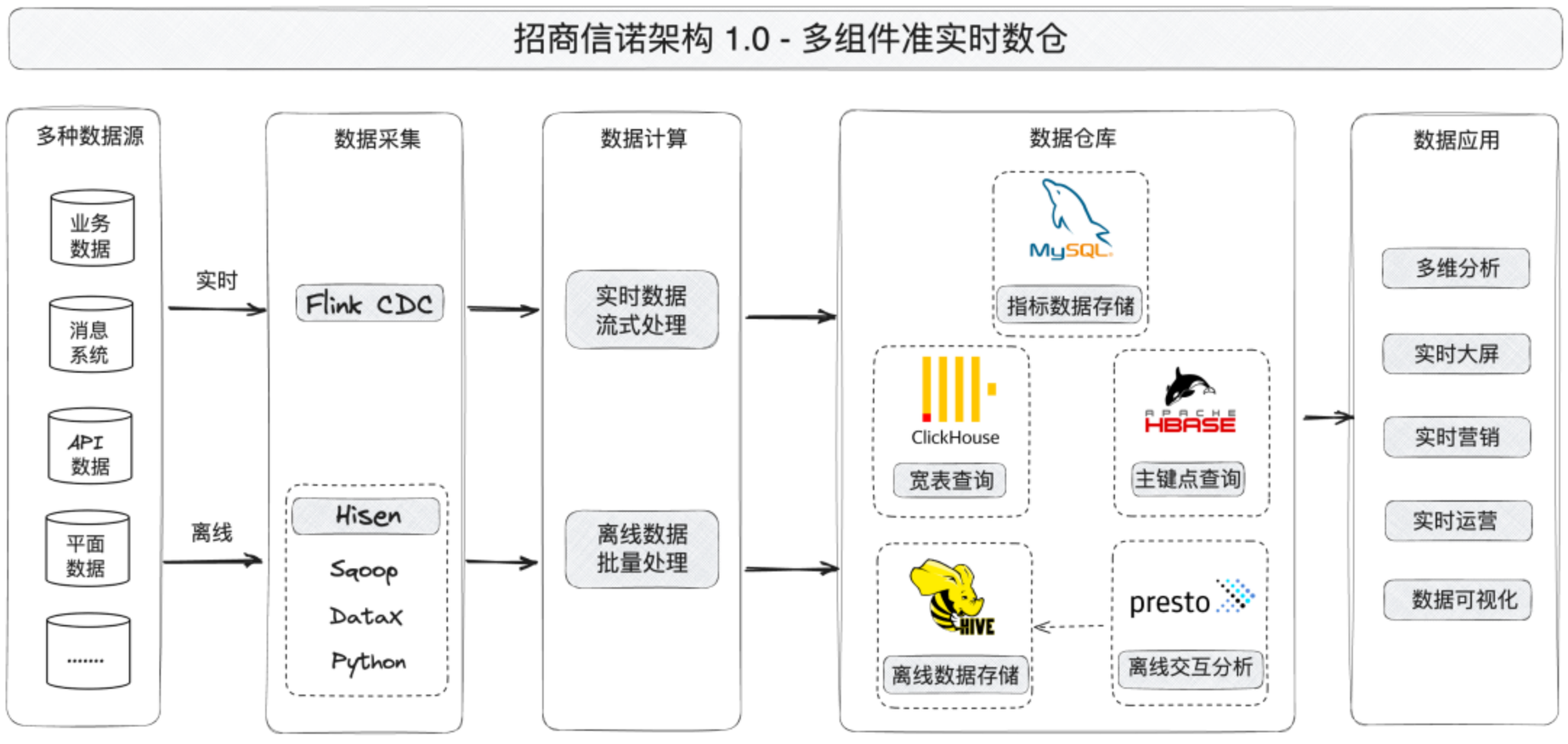
本文将详细介绍招商信诺在数仓架构迭代与升级过程中如何基于 Apache Doris 统一存储、计算和查询出口、如何满足写入时效性的要求、如何在高并发点查与多表关联等场景下实现极速查询性能，为销售线索高效写入与查询、客户留存信息高频更新、服务场景数据一致打通等方面提供助力，进一步将客户线索转化为私域商机，赋予企业在经营、服务、营销等多方面的能力。

架构1.0：多组件准实时数仓

最初的业务需求是希望通过数仓来承载面向 C 端用户的保单自助查询、面向业务分析人员的多维分析报表以及面向管理者的实时数据大屏（Dashboard）三类业务场景。数仓需要满足业务数据的统一存储和高效的查询能力，以支持业务高效分析决策，同时还需要支持数据回写，以实现闭环式业务运营。

- 保单自助查询：用户通过招商信诺 APP 根据保单 ID 自助查询承保合同，或者通过不同维度（如承保时间、保险类别、理赔金额）进行自定义筛选查询，查看保单生命周期内的信息。
- 多维报表分析：依据业务需求，业务分析人员通过开发明细数据、指标维度报表，获得关于保单在产品创新、费率、反理赔欺诈等方面的业务洞察，并据此支持经营策略调整。
- 数据大屏（Dashboard）：主要用于某银行渠道、某分公司的实时大屏，通过对指标等数据的统一汇总，将热门险种、每日销售额、保险种类缴纳总额与占比、历年保险缴纳涨幅趋势等信息展示于实时大屏中。

业务初期对数据服务的要求较为单一，主要是以提升报表数据的时效性为主，因此在数仓搭建的过程中，我们采用典型的 Lambda 架构，通过实时与离线两条链路分别进行数据采集、计算与存储，其中数仓主要采用宽表模型设计以支持对指标数据、明细数据的查询分析。



由架构图可以看到，FlinkCDC 负责实时数据采集，我们自研的 Hisen 工具（包括 Sqoop、DataX 以及 Python）负责离线数据采集。原始数据采集后，实时数据利用 Flink 进行计算、离线数据交由 Hive 进行批处理，最终导入至不同的 OLAP 组件（包括 Presto、Clickhouse、HBase 以及 MySQL）中，由 OLAP 向上层业务提供数据服务，其中各组件在架构中分别扮演不同的角色：

MySQL：按照业务需求，在数据完成计算后主要用于存储指标数据。目前，数仓表的数据量已经突破千万级， 而 MySQL 存储具有局限性，容易出现执行时间过长、系统返回错误等问题。

Clickhouse：Clickhouse 在单表数据读取的性能上表现出色，在大表 Join 性能较弱。随着业务场景的增加，实时数据量不断叠加与更新下，Clickhouse 面对新的业务需求存在一定局限：

- 为减少指标重复计算，需要引入星型模型进行多表关联与高并发点查询，而 Clickhouse 无法支持；
- 当保单内容发生变更时，需要数据实时更新写入，而 Clickhouse 缺少实时事务的支持，面对数据变更时需要重新生成宽表以覆盖旧数据，在数据更新时效性要求方面存在一定不足。

HBase：主要用于主键查询，从 MySQL 与 Hive 中读取用户基础状态数据，包括客户积分、承保时间、累积承保保额。由于 HBase 不支持二级索引，对于非主键的数据读取较为局限，无法满足关联查询场景，同时 HBase 也不支持 SQL 语句查询。

Presto：由于上述组件在数据查询方面的场景限制，我们还引入了 Presto 作为离线数据的查询引擎，用于与 Hive 中的数据进行交互式分析，为上游端提供报表服务。

在数仓 1.0 版本上线后，已在超过 10 余家分公司中上线使用，开发了大量的数据大屏以及 BI 报表。随着业务范围的不断拓展，营销、运营以及客户服务等场景对数据写入与查询性能提出了更高的要求，然而混合使用四个组件提供数据服务的 1.0 版本架构在实际业务中存在一些挑战。为了避免由于架构组件过多所产生的运维成本升高、研发人员学习成本升高等问题，也为了确保在离线与实时链路中多源数据的一致性，我们决定展开架构更新迭代之旅。

组件需求与系统选型

为满足业务需求，我们需要为架构“减负”，尽可能地缩短数据处理过程。而 1.0 架构由于组件过多，链路冗余等问题势必降低了数据存储与分析的性能与时效性。因此，我们希望寻找一个

OLAP 系统既能覆盖大部分的业务场景，也能够降低复杂技术栈带来的开发、运维和使用成本，还能最大化的提升架构性能。具体要求如下：

- 导入性能：具备实时写入、实时更新的能力，并支持高吞吐的海量数据写入。
- 查询性能：提供维度数据以及交易数据的查询服务，具备高性能的海量数据实时查询的能力。
- 灵活性多维分析、自助查询能力：不仅能够支持主键索引以提供点查与范围查询，还能够支持多维度检索分析，提供对亿级数据的表关联查询，实现灵活动态、下钻上卷的业务数据分析。
- 数据平台架构简化：需要一款综合能力强的组件以替换当前冗余架构，满足在实时与离线数据的读写、不同场景下的高查询性能、简单易用的 SQL 语句查询等能力。

基于此，我们开始系统选型，将市面上热门组件与现有架构进行多方面对比，评估是否满足业务方对组件的需求，最终在众多 OLAP 中锁定了 Apache Doris，具体原因如下：

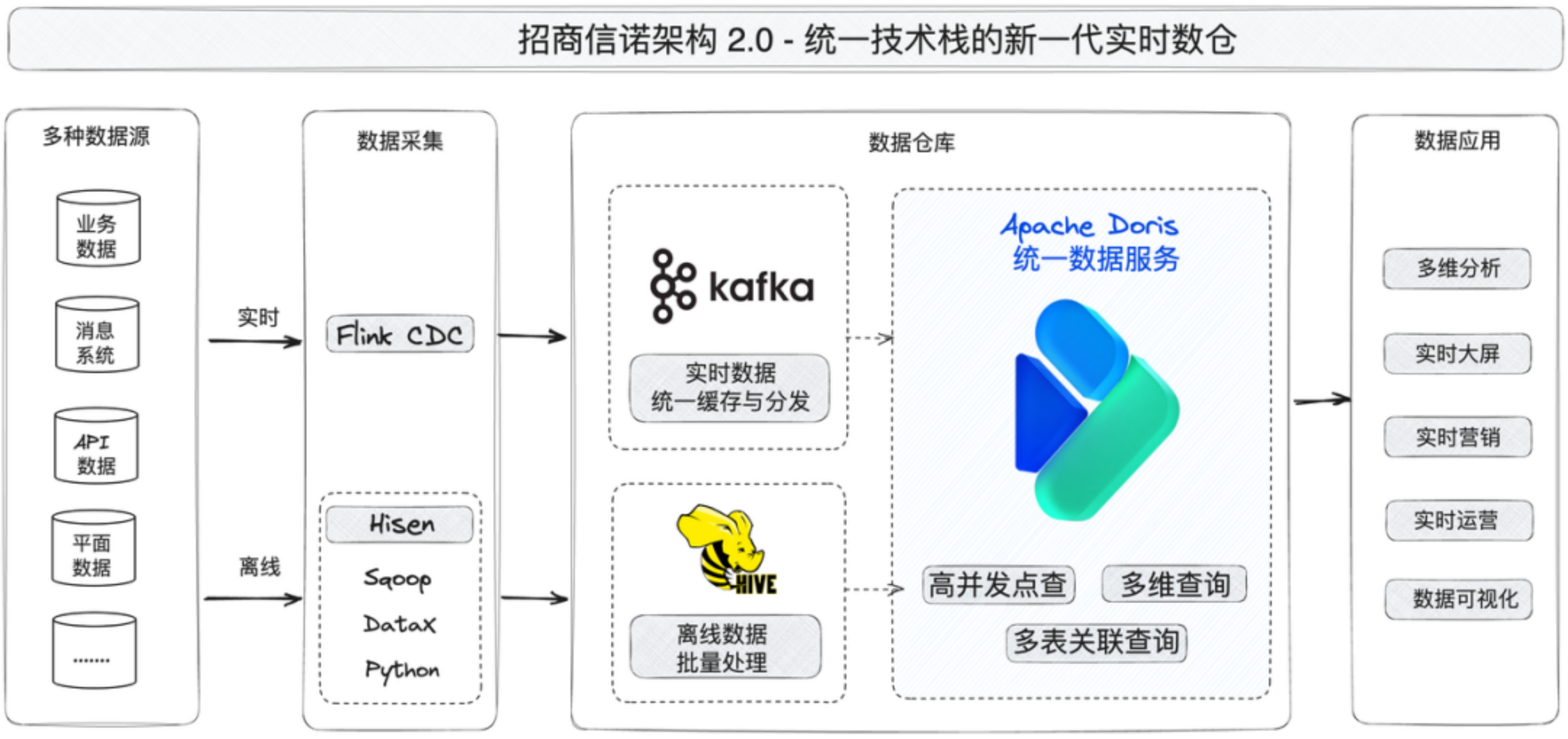
- **支持低延迟实时写入：**支持 FlinkCDC 在海量数据下的高吞吐写入，提供实时数据对外服务；支持主键表模型写时合并，实现微批高频实时写入；支持 Upsert 与 Insert Overwrite，保证高效的数据更新。
- **保证数据一致有序：**支持 Label 机制和事务性导入，保证写入过程中 Exactly Once 语义；支持主键模型 Sequence 列设置，保证数据导入过程中的有序性。
- **查询性能优异：**Doris 支持 Rollup 预聚合与物化视图完成查询加速；支持向量化处理以减少虚函数调用和 Cache Miss；支持倒排索引以加速文本类、普通数值、日期类等全文检索或范围查询。
- **支持高并发点查询：**支持分区分桶裁剪，通过 Partition 将时间分区、设置 Bucket 数量过滤非必要的数据，以减少底层数据扫描，实现查询快速定位；此外，在 Doris 2.0 版本中还新增了行式存储格式、短路径点查、预处理语句等一系列优化，进一步提升点查执行效率、降低 SQL 解析开销。
- **支持多种数据模型：**支持星型模型，满足亿级数据表关联查询需求；支持大宽表聚合，提供单表极速查询性能与多维分析能力。
- **架构简单、易运维、易扩展、高可用：**Doris FE 节点负责管理元数据与多副本、BE 节点负责数据存储与任务执行。这使得架构在部署与配置方面操作简单，易于运维；同时 Doris 能够一键加减节点、自动副本补齐与节点间的负载均衡，易于扩展；且当单节点故障时，Doris 依旧能够保持集群稳定运行，满足我们对服务高可用、数据高可靠的要求。

组件	海量数据的实时写入	实时随机查询	实时 OLAP 分析能力	支持二级索引	支持宽表模型	支持维度建模	支持表关联	离线 OLAP 分析能力	离线随机查询能力	支持高并发	支持亿级数据查询	支持 SQL	综合
MySQL	★	★★	★	★★★★	★★★	★★★	★★	★★	★★	★★	★★	★★★★	★★
HBase	★★★★	★★★★	✗	★	★★★★	★	✗	✗	★★★★	★★★★	★★★★	✗	★★
Clickhouse	★★	★★	★★	★★★★	★★★★	★★	★★	★★★★	★★★★	★★	★★★★	★★★★	★★★
Apache Doris	★★★★	★★★★	★★★★	★★★★	★★★★	★★★★	★★★★	★★★★	★★★★	★★★★	★★★★	★★★★	★★★★

注：图表评分由大数据团队内部选型小组根据需求评定，仅供实时数仓重构项目使用，不具备代表性

从对比图中我们也可以看出，不论是实时还是离线场景，Apache Doris 的综合能力最均衡也是最优秀的一个，能够支持自助查询、实时与离线 OLAP 分析能力、高并发点查与表关联等查询场景，并且写入性能、高可用、易用性等方面表现优异，是一款能够满足多个业务场景的组件。

架构2.0：基于 Apache Doris 统一技术栈



数仓架构的两代版本主要在存储、计算、查询分析方面有很大不同。1.0 版本依赖于多个组件共同构建 OLAP 分析引擎，在业务拓展阶段逐步出现架构存储冗余、数据延迟、维护成本过高等问题。架构 2.0 版本基于 Apache Doris 升级改造，替换了 Presto、MySQL、HBase、Clickhouse 四个组件并将数据迁移至 Apache Doris 中，以提供统一的对外查询服务。

新架构不仅实现了**技术栈的统一**，还降低了开发、存储与运维等各方面的成本支出，实现了业务与数据的进一步统一。基于 Apache Doris 一套系统能够同时支撑在线与离线任务处理，实现**数据存储统一**；能够满足了不同场景的数据分析服务，支持高吞吐的交互式分析与高并发的点查询，实现**业务分析统一**。

加速数据分析效率

通过 Doris 极速分析性能，**在面向 C 端用户的高并发点查询场景中，QPS 能够达到数千至数万，对于数亿或者数十亿数据的查询达到毫秒级响应**；利用 Doris 丰富的数据导入方式和高效的写入能力，实现秒级写入时延，并利用 Unique Key 写时合并来进一步加速在并行读写阶段的查询性能。此外，我们还利用了 Doris 冷热分层将海量的历史冷数据存储于廉价的存储介质中，降低了历史数据的存储成本并提升了对热数据的查询效率。

降低各类成本支出

新架构较于原有架构，核心组件的数量减少了一半，平台架构得以大幅简化，运维成本大大降低。此外，Apache Doris 使数据无需再通过不同组件完成存储与查询服务，统一了实时与离线业务负载、降低了存储成本；数据服务 API 对外提供服务时也无需再合并实时与离线数据，**使数据服务 API 接入时的开发成本缩减至 50 %；**

保证数据服务高可用

因为 Doris 的统一存储、计算和服务的数仓架构，平台整体灾备方案易于实施，不再担心多个组件造成数据丢失、重复带来的问题。**更重要的是，Doris 自带的跨集群复制 CCR 功能，能够提供集群间数据库表秒级至分钟级的同步，**当系统崩溃导致业务中断或者丢失时，我们可以从备份中快速恢复。

Doris 跨集群复制 CCR 功能两大机制满足了我们在系统服务可用性方面的抢需求，保证了数据服务高可用，具体如下

- Binlog 机制：当数据发生变更时，通过该机制我们可以自动记录数据修改的记录与操作，并且对每个操作构建了递增序列的 LogID，实现数据的可追溯性与有序性。

- 持久化机制：在系统崩溃或者发生突发事件后，通过该机制能够将数据持久化至磁盘来确保数据的可靠性和一致性。

保险一线业务收益与实践

目前，基于 Apache Doris 统一技术栈的实时数仓已经在 2022 年 Q3 上线并投入生产环境使用，用于支撑海量数据的 OLAP 高效分析能力，并在平台上支撑了更多业务相关的场景。在业务经营方面，销售线索的规模也在不断扩大，目前已达到亿级。随着 Apache Doris 的功能的进一步引入，由数仓支持的一线业务营收也在持续增长中。

- **销售线索高效追踪：**目前，我们已经在销售与业绩类追踪上线 30 + 新场景应用，业务人员能够基于销售线索准确、快速地获取客户在官网、APP、商城、公众号、小程序等渠道的保险测评、直播参与数据、企微活动参与数据、免险投保等轨迹与数据，并通过 Apache Doris 多维分析进行线索转化，最终实现精准触达客户、有效抓住客户动机、及时跟进成单。

- **客户留存信息高频更新：**在新客户转化与老客户关怀类已上线 20 + 新场景应用，业务场景的顺利进行离不开数据平台对于客户留存信息的高频更新能力，通过 Apache Doris 对老客户数据定期分析，能够有效查询客户在不同阶段的保险业务需求，发现老客户的保障缺口，拓宽老客户可保边界，进一步增加业务经营收益。

- **业务场景数据一致打通**：在客户服务方面，我们更关注为客户提供一致化的体验与快速响应的服务。目前，我们已经上线了 20 + 相关服务体验的新场景应用，避免出现信息不对称、数据不一致的情况，保证各个销售环节的数据在承保、理赔、客服咨询、会员中心等流程中能够一致统一。

未来规划

Apache Doris 的引入在实时数仓架构简化与性能提升方面起到了至关重要的作用。目前，我们已经基于 Apache Doris 替换了 Presto、Clickhouse、MySQL、HBase 多个组件以实现 OLAP 技术栈统一、各类成本降低，并提升导入与查询性能。

同时我们也计划进一步基于 Doris 在批处理层（Batch Layer）尝试应用，将离线数据批处理统一在 Doris 中进行，解决 Lambda 架构在实时和离线链路中成本叠加、无法兼容的问题，真正实现架构在计算、存储、分析的统一。同时，我们也将继续发挥 Doris 统一的优势，利用 Multi-Catalog 让数据在湖与仓之间自由流动，实现数据湖和多种异构存储之上无缝且极速的分析服务，成为一套更完整、更开放统一的大数据技术生态系统。

非常感谢 SelectDB 团队一直以来对我们的技术支持。至此，招商信诺数据仓库不再局限于简单的报表场景，通过一套架构支撑了多种不同场景的数据分析、满足了实时与离线数据的统一写入与查询，为产品营销、客户运营、C 端以及 B 端等业务提供数据价值，使保险人员更高效地获取数据、更准确地预知客户需求，为企业获得先机。

未来，我们也会持续参与到 Apache Doris 社区建设中，贡献保险行业在实时数仓的建设经验与实践应用，希望 Apache Doris 不断发展壮大，为基础软件建设添砖加瓦！

河北幸福消费金融基于Apache Doris 构建实时数仓，查询提速 400 倍！

作者：河北幸福消费金融信息科技部

导读：随着河北幸福消费金融的客户数量和放贷金额持续上升，如何依托大数据、数据分析等技术来提供更好决策支持、提高工作效率和用户体验，成为当前亟需解决的问题。基于此，公司决定搭建数据中台，从基于 TDH 的离线数仓再到基于 Apache Doris 的实时数仓，最终统一了数据出口，提升了数据质量，并实现查询速度 400 倍的提升。本文将详细为大家分享河北幸福消费金融数据中台搭建经验和应用实践，希望为其他企业带来有益的参考。

河北幸福消费金融股份有限公司由张家口银行发起设立，是 2017 年 6 月正式开业的全国第 22 家、**河北省首家消费金融公司**，主要面向个人客户发放最高额不超过 20 万元的普惠、小额、信用消费贷款。目前公司服务区域覆盖全国 32 个省级单位，相继获评为国家科学技术部认定的全国高新技术企业、河北省科技型中小企业和石家庄市科技“小巨人”企业。

随着客户数量和放贷金额持续上升，如何依托大数据、数据分析等技术，为各业务线人员提供更好的决策支持，如何提高工作效率、为客户提供更佳的使用体验，成为了当前亟需解决的问题。具体需求如下：

- 高管看板类：搭建高管驾驶舱帮助高层快速了解公司当前的整体经营状况，驾驶舱集成全业务线数据，包括实时业务指标和离线业务指标，在该场景下我们希望查询结果可以在毫秒内返回，便于管理层进行高效决策。
- 实时变量类：为给风控决策提供实时支撑，需在 500ms 内返回全产品线查询结果，并可基于申请、授信、关联人、逾期以及还款等基础信息，计算产品级、客户级和借据级高维度衍生变量。
- 决策分析类：为给各业务部门提供业务分析和决策支撑，需在秒级返回年、季、月多维度的主题报表，特别是风险部门，需要从放款开始回溯全生命周期的运营情况；而计财部门，则

需要通过以往经营数据表现，预测未来的盈利，数据量大且逻辑复杂。

- 风险建模类：提供全量、明细级的数据，用于风险建模的变量跑批，满足对客户评分评级，参与审批、授信等核心业务的要求，以支持业务指标的观测、投入产出分析、变量筛选以及决策制定等业务需求。

- 监管合规类：为确保业务符合相关法律法规、行业标准等规范，需要根据监管合规的要求进行合规子系统的定期上报，上报的数据分为指标汇总项数据和明细数据。

为了满足不同业务线对数据分析的需求，公司开始搭建数据中台并对其进行优化。最初，公司基于商业化产品 TDH 搭建了离线数仓，以满足基本数据分析需求。然而，随着数据时效性的提高和实时分析需求增加，公司迫切需要搭建一款实时数据仓库。因此引入了 Apache Doris 并在此基础上搭建了实时数仓，最终建立了一个高效、稳定的数据中台。本文将详细为大家分享河北幸福消费金融数据中台搭建经验和应用实践，希望为其他企业带来一些有益的参考。

基于 TDH 的离线数仓

因早期主要解决离线分析需求，优先基于 TDH 集群建设离线数仓。通过 Sqoop、DataX 将上游数据采集到离线数仓，经过标准化数据清洗，完成数仓日常跑批。离线数仓架构图如下所示：

随着数据的积累和业务人员对数据时效的要求越来越高，基于 TDH 离线平台的一些问题逐渐显现出来：

- 数据采集同步时效慢：离线数仓抽数工具依赖于 Sqoop、DataX 等组件，而受限于调度周期，此类工具采集的数据必然有滞后性。
- 资源冲突大：离线数仓每日跑批时间跨度大，一般从凌晨到下午5点左右，这将导致跑批和即席查询之间发生资源冲突，影响业务人员的使用体验。
- 查询分析慢：在使用离线平台进行自定义统计分析和数据探索时，查询分析响应速度慢、时效性难以保障，严重影响工作效率。
- T+1 延迟高：各业务线对实时数据处理的需求在逐渐增多，T+1 的数据已经无法适应数据快速获取和产生业务价值的诉求。
- 报表定制周期长：报表定制化开发有固定的迭代周期，难以满足业务人员对数据的灵活多样的分析探索。
- 烟囱效应：定期上报数据时，数据中台需要从多个业务系统中拉取数据。当业务系统发生变更后，会波及上述相关的报送子系统，形成烟囱效应。

技术选型

为了解决上述问题，我们迫切需要一款 MPP 引擎来构建实时数仓。对于新引擎我们有几个基本的要求：首先，需要简单易上手，以便团队快速掌握和使用；其次，需要具备强大的数据导入能力，以便快速高效地导入海量数据；同时，需要兼容离线数仓相关工具，以便与现有的数据处理工具和技术体系无缝衔接；此外，搭建和切换成本也需要低，以便快速部署使用和进行扩缩容；最后，它需要具备较好的并发能力和优异的查询性能，以便支持高并发、复杂查询等业务场景的需求。

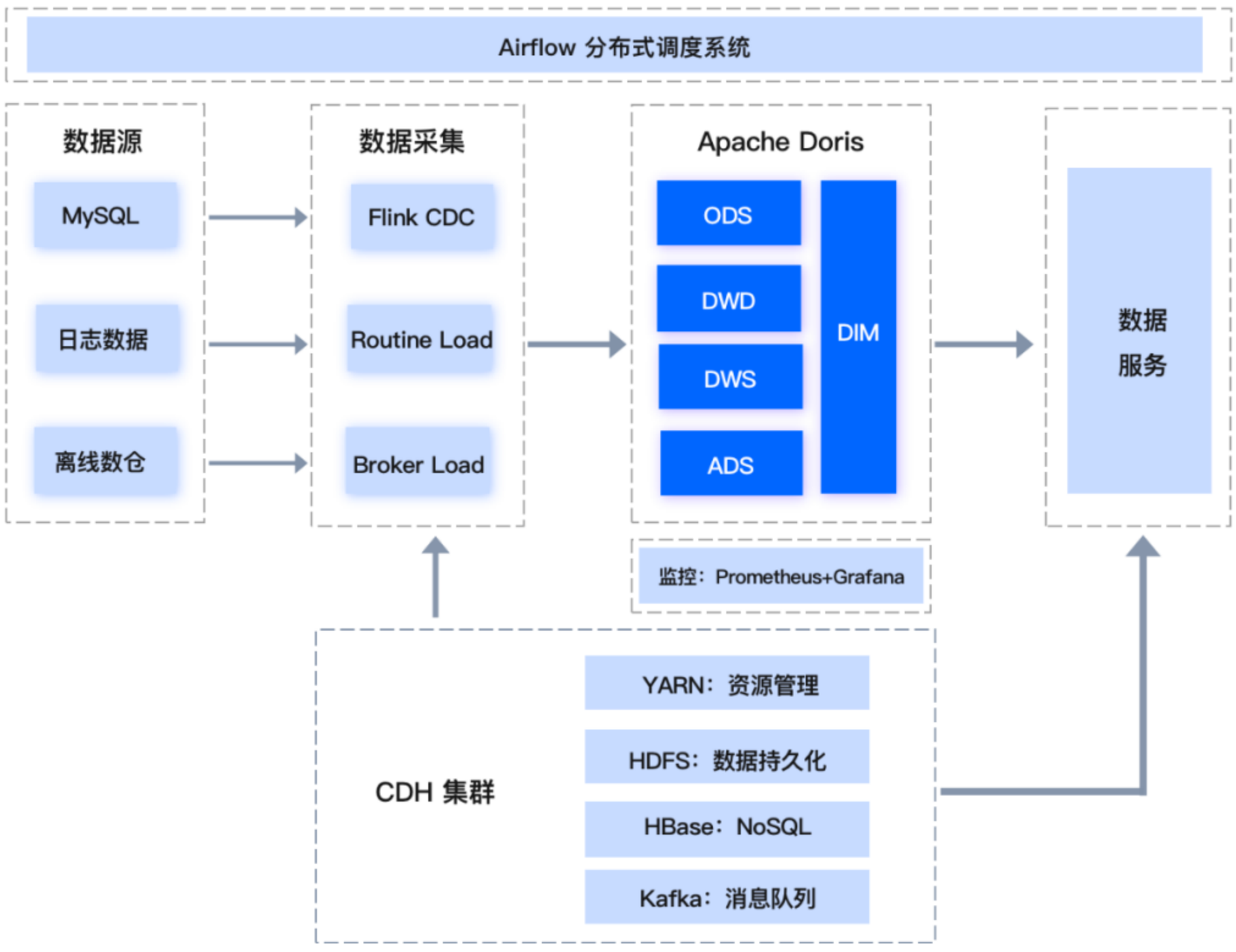
在以上选型要求驱动下，我们对目前比较流行的 ClickHouse 与 Doris 进行了系统的调研，其中 Apache Doris 更符合我们的选型的要求，具体原因如下：

- 部署成本低：Doris 采用分布式技术架构，部署只需两个进程，不依赖其他系统，在线集群扩缩容，自动副本修复，部署及使用成本较低。
- 快速上手使用：Doris 采用主流的分区分桶设计思路，索引结构与 MySQL 的思路类似，相关人员在使用 Doris 时无需学习大量的新知识。相比之下，ClickHouse 在建库建表需要分别指定类型，使用流程相对比较繁琐，上手难度也比较高。

- 工具兼容：业务人员通常使用 TDH 的客户端工具 WaterDrop 进行离线数仓查询，Doris 通过标准协议链接可完美兼容 WaterDrop，而 ClickHouse 无法兼容。
- 数据生态圈丰富：Doris 数据生态圈丰富，与 Flink、Kafka 等组件结合度较高，同时支持联邦查询，提供了丰富的数据导入和接入方式，可以满足多场景下的数据处理需求。
- 高并发能力：我们对 Doris 进行了性能压力测试，在高并发和大数据量的情况下，Doris 表现出较好的性能和稳定性，能够满足不同业务场景的需求。
- 社区活跃度高：Doris 社区非常活跃，有大量的开发者和用户参与其中，提供了丰富的技术支持和解决方案。同时 Doris 社区提供了全面的文档和资料，方便用户学习和使用 Doris。此外，SelectDB 为社区提供了一支全职专业的技术团队为社区用户提供服务与支持，任何问题均可得到快速响应。

基于 Doris 的实时数据仓库

在离线数仓的基础上，使用 Doris 结合 CDH 集群、Airflow 集群搭建了实时数仓，实时数仓的数据来源主要为离线数仓和 MySQL，使用 Flink CDC 结合 PyFlink（使用 Python 调用 Flink 的 API，简称 PyFlink）将 MySQL 中的数据实时地采集到核心计算引擎 Doris 中（后文将详细介绍），上层为 Airflow 分布式调度系统，可以将实时任务进行常规化的调度运维。我们对 Doris 引擎进行了基本数仓分层，数据经过各层处理后统一为各场景提供数据服务。



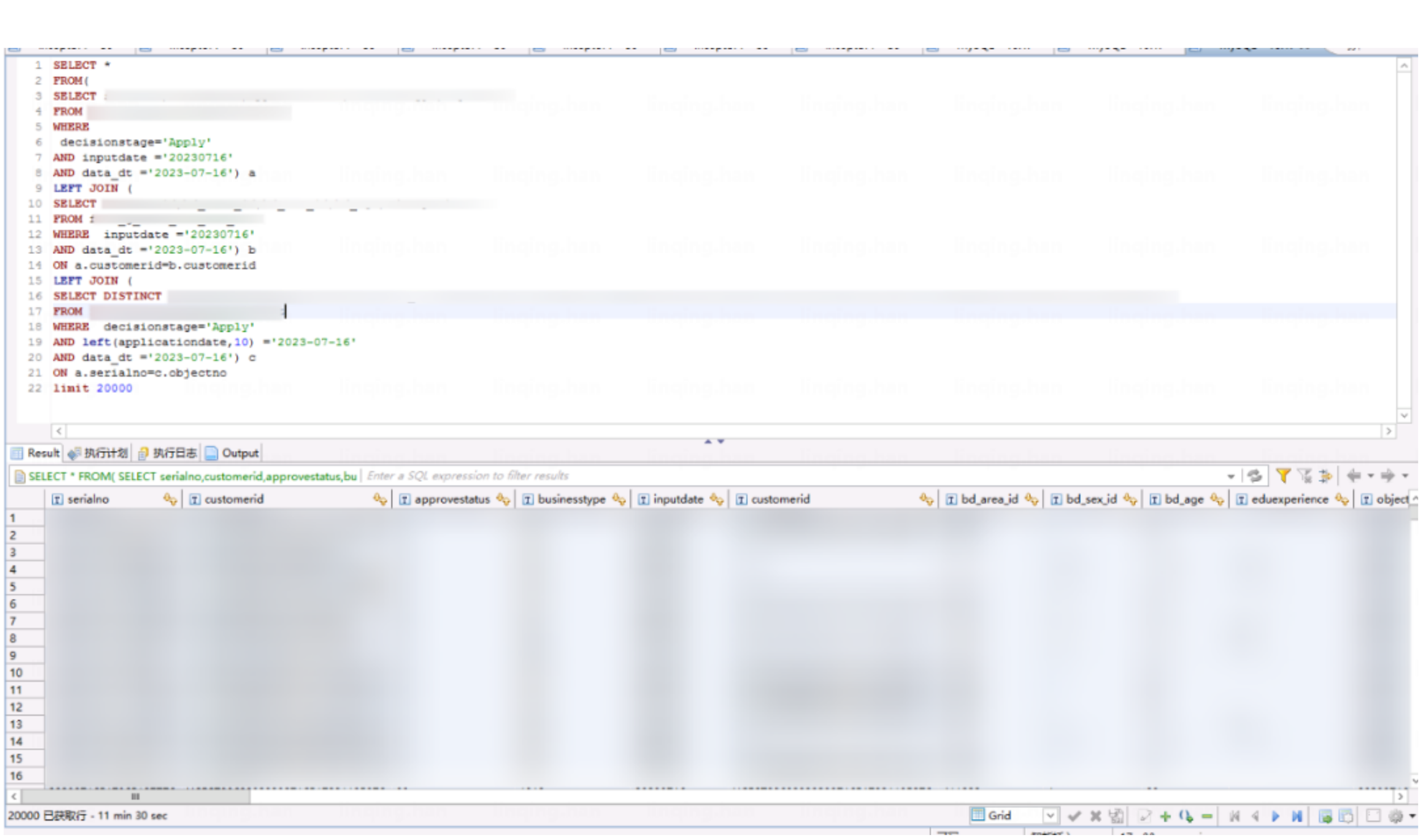
基于 Doris 提供的丰富导入方式，我们可以快速将离线数仓中的实时数据清洗整合接入到 Doris 集群中，实现数据的快速迁移。目前我们已经将基于 TDH 的查询分析和数据探索服务全部转移到 Doris 引擎上，借助 Doris 引擎快速计算的能力和优异的查询性能，可以更高效地进行数据处理和分析，业务处理速度和效率得到显著提升。

以某 SQL 为例，该 SQL 主要应用在信贷审批场景。我们对比了原有架构和新架构从十万、千万，亿级别的三个大表 Join 查询返回速度。结果显示，在过去 TDH 架构中执行查询需要 **11 分 30 秒**才可返回结果，而基于 Doris 的新架构中仅需要 **1.7 秒**即可返回结果，**速度提升 400 倍**！

数据规模：

[redacted]	298303616	亿
[redacted]	31359579	千万
[redacted]	175831	十万

原有离线数仓：需要 **11 分 30 秒**才可返回结果



基于 Doris 的新数仓：优化查询平均返回时间仅需 **1.7 秒**，有时甚至可以 **1 秒内**返回。

```sql
SELECT *
FROM(
SELECT
FROM
WHERE
 decisionstage='Apply'
AND inputdate ='20230716'
AND data_dt ='2023-07-16') a
LEFT JOIN (
SELECT
FROM
WHERE  inputdate ='20230716'
AND data_dt ='2023-07-16') b
ON a.customerid=b.customerid
LEFT JOIN (
SELECT DISTINCT
FROM
WHERE  decisionstage='Apply'
AND left(applicationdate,10) ='2023-07-16'
AND data_dt ='2023-07-16') c
ON a.serialno=c.objectno
limit 20000
```

Result | 执行计划 | 执行日志 | Output

SELECT * FROM(SELECT serialno,customerid,approvestatus,bu | Enter a SQL expression to filter results

	serialno	customerid	approvestatus	businesstype	inputdate	customerid	bd_area_id	bd_sex_id	bd_age	eduexperience	object
1											
2											
3											
4											
5											
6											
7											
8											
9											
10											
11											
12											
13											
14											
15											
16											

20000 已获取行 - 1715ms

Grid

应用实践

实时数据归集

公司的业务系统通常是按照产品进行库划分，各个产品表结构保持一致。而实时数仓核心功能就是依靠 Doris 丰富的导入能力，将分散的库对应的相同的逻辑表归集到 Doris 下的同一个逻辑表上，汇集后的数据也能在监管主题层面进行整体调整，避免烟囱效应的发生。汇集的实时数据进入数仓后，会主动触发衍生变量的自动计算，更新衍生变量的值。而衍生变量的汇总值在一个单独的表中，当进行查询时，可以实现毫秒级别的查询响应。

在进行实时数仓归集时，首先需要确定 FlinkCDC、Flink 、Flink on Yarn、Apache Doris 等核心组件的版本号，接着基于 PyFlink 进行产品化自动接入实时数仓的建设。具体操作如下：

- 在数据层面，将业务系统数据库按照水平和垂直进行切分，以提升读写性能并增加高可用性。
- 在数仓层面，我们对业务表的数据进行了维度汇集，以便进行更好的统一汇总分析。
- 在数据接入方面，我们需要高效地接入现有业务系统的存量数据，并持续稳定地接入增量数据。

此外，我们还提供了标准化的接入方案和接口，以满足不同业务场景的需求。

使用步骤：

1. 接入配置表：配置归集的业务库表的相关信息。

```
--------------------+
| CREATE TABLE "conf_source_to_sink" (
  "id" int(11) unsigned NOT NULL AUTO_INCREMENT,
                      varchar(64) DEFAULT NULL COMMENT '源库',
                       varchar(64) DEFAULT NULL COMMENT '实体物理表',
                    varchar(64) DEFAULT NULL COMMENT '目标库',
  "t  e" varchar(128) DEFAULT NULL COMMENT '目标表',
                tinyint(1) DEFAULT '0' COMMENT '库是否熔断 0 否  1 是',
  "    " tinyint(1) DEFAULT '0' COMMENT '是否初始化 0 否 1 是',
              int(11) DEFAULT '1' COMMENT '分桶个数',
             char(128) DEFAULT NULL COMMENT '正则规则',
            lfs_path" varchar(4094) DEFAULT NULL COMMENT '最近一次保存点',
            int(1) DEFAULT '1' COMMENT '是否生产任务 0 否  1 是',
             int(11) DEFAULT '1024' COMMENT '批处理大小',
              int(11) DEFAULT '2' COMMENT '默认并行度',
                    seconds" int(11) DEFAULT '120' COMMENT 'checkpoint间隔',
                 id" varchar(32) DEFAULT NULL COMMENT 'yarn实例id',
                  varchar(64) DEFAULT NULL COMMENT 'CDC个性化参数',
              timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP COMMENT '更新时间',
  PRIMARY KEY ("id"),
  UNIQUE KEY "idx_no_account" ("physical_table_name","from_database_name")
) |
```

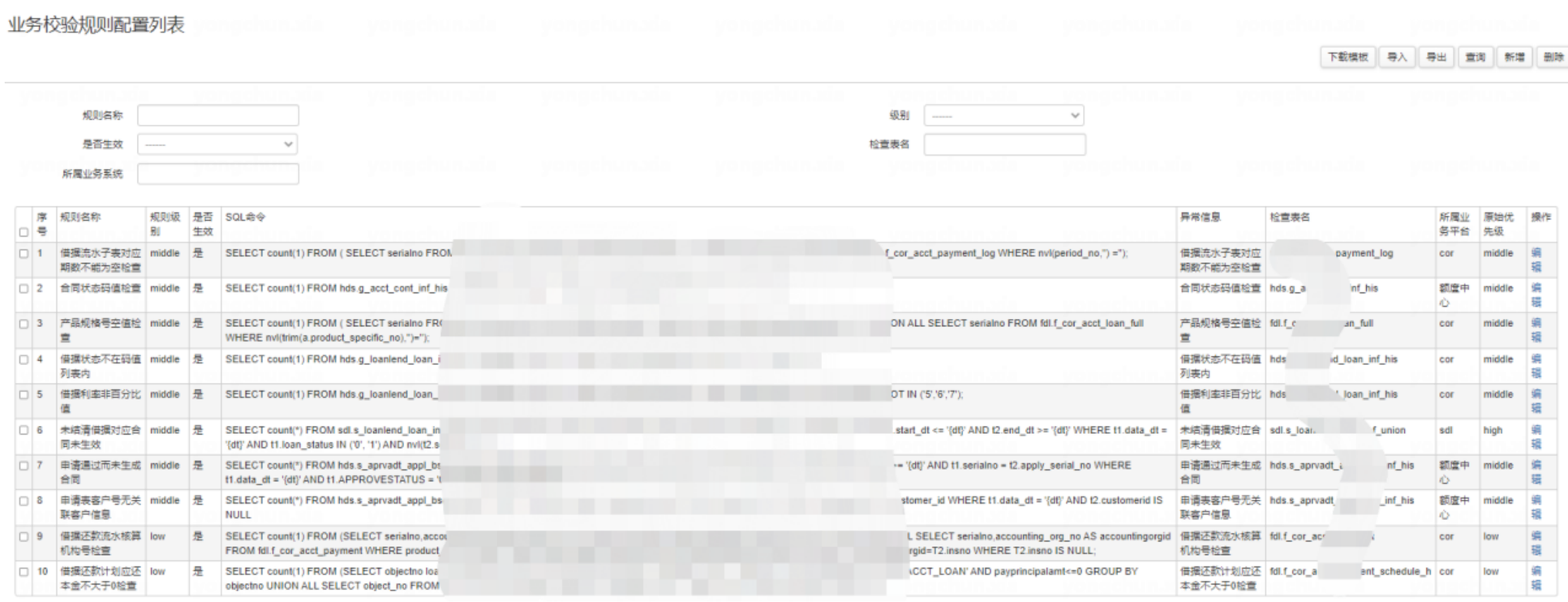
2. 调度系统部署：通过调度系统部署实时归集的任务

```
                                = BashOperator(
    task_id='
    bash_command='                                                                                                ',
    dag=dag)
```

3. 任务常规运维：我们对任务上线、启动、停止和异常恢复处理等功能进行了高度封装，并与分布式调度系统 Airflow 进行了深度集成和融合。使用人员不必关心底层细节，可以轻松地将 MySQL 表一键迁移到 Doris，实现存量和增量数据的自动化迁移。经沟通，社区目前已发布了 Doris-Flink-Connector 1.4.0 版本，该版本集成了 Flink CDC，可以实现了从 MySQL 等关系型数据库到 Apache Doris 的一键整库同步。

数据质量监控

离线数仓存在各种数据质量问题，这些问题通常在数据跑批时才会暴露出来，导致数据修复时间窗口急剧被压缩。为了解决这个问题，我们利用 Doris 建立了数据质量监控系统，同时将离线数仓的数据质量监控模型迁移到 Doris 。基于该系统可以实时监控业务指标和数据质量，并在发现问题时及时进行人工干预或报警，提高离线数仓跑批的稳定性和效率。另外当实时数仓获取归集后的数据后，可通过数据质量监控系统的校验规则第一时间对数据质量进行实时检查，保证数据归集的准确性。



目前我们已经将 30% 的数据监控指标和 35 个业务指标迁移到 Doris 实时集群上，每月可成功规避问题 3 次以上，有效提升了离线数仓跑批的数据质量。后续我们将继续将更多的数据监控指标和业务指标迁移到 Doris 集群中，以进一步提高数据处理的效率和质量。

数据联邦查询

各业务条线的核心数据存储在不同类型的数据库中，如 MySQL、Hive、ES 等。Apache Doris 1.2 版本提供的 Multi Catalog 功能可以统一数据查询出口、实现联邦查询，为数据分析提供了极大的便利。同时，借助 Doris 的持久化能力，可以通过外表的方式快速同步其他数据源数据，方便快捷。此外，通过 Apache Doris 聚合查询、向量化引擎等技术的加持，**我们真正实现了高效的数据统一门户，提高了数据分析的效率。**

优化经验

负载均衡

随着 Doris 接入的业务量不断增加，FE 的负载也在不断增长。为了实现 Doris 的高可用性，我们增加了 FE 节点数，在多个 FE 节点上部署负载均衡层。我们选择基于 Nginx TCP 反向代理的方式来构建 FE 的负载均衡，有效地实现了 FE 角色之间的负载均衡。具体配置方式如下：

```
upstream dorisfe {least_conn;#hash $remote_addr;
server 51:9030 max fails=3 fail timeout=4s;
server 52:9030 max fails=3 fail timeout=4s;
server 58:9030 max fails=3 fail timeout=4s;
}
server{
listen 9030;
proxy_connect timeout 10s;
proxy timeout 10800s;
proxy pass dorisfe;
proxy next upstream on;
proxy_next upstream timeout 3;
proxy. next upstream tries 3;
}
```

查询优化

当前审批系统的业务数据持久化在关系型数据库 MySQL 中，累计总进件量将近 2.8 亿。为了应对日益膨胀的的数据问题，业务系统的数据库采用了分表和数据归档的设计思路。但是在业务上，我们仍需要对全量数据进行业务查询，并且时效要求在 3 秒内返回结果。以下是查询需求的抽象分类：

- 以“申请编号”，“客户编号”，“身份证号”，“核心客户号”中的一个或多个作为查询条件进行查询
- 以“申请日期”或“更新日期”中的一个条件，结合“姓名”、“申请类型”、“进件渠道”、“白名单渠道”、“决策阶段”、”审批类型”、”审批结果”等形成复核条件做查询
- 以“申请日期”或“更新日期”中的任意一个为条件，对近一周的审批明细数据进行查询查询

为了满足以上查询场景的需求，我们将审批进件数据结合 Doris 引擎的分区分桶技术、布隆过滤器和位图索引进行合理的设计，最终整体实现了满足业务上 **3 秒内**的查询效率需求。

优化策略：

分区：apply_tim

分桶：ID、database_name、table_name

布隆索引：id_number, bhb_customer_id, customer_name, customer_id, serial_no

位图索引： apply_source, white_channel, approval_result, approval_status, product_type, decision_stage

基于上述查询的压测指标效果如下：

表	数据量	分桶	分区	布隆索引	位图索引
d_apply	2.8亿	√	√	√	√

优化最终策略	字段
分桶字段	id, database_name, table_name
分区字段	apply_time
布隆索引	id_number, bhh_customer_id, customer_name, customer_id, serial_no
位图索引	apply_source,white_channel,approval_result,approval_status,product_type,decision_stage

		测试次数	异常返回	查询最大值	查询最小值	查询均值	>3秒次数	场景	命令
测试对比	分区	100	0	0.6	0.07	0.22	0	姓名 + 申请时间	python doris_batch_test_v2.py d_apply_v5 customer_name apply_time
	非分区	100	0	2.15	1.5	1.72	0		python doris_batch_test_v2.py d_apply customer_name apply_time
	分区	100	0	1.47	0.96	1.23	0	姓名 + 更新时间	python doris_batch_test_v2.py d_apply_v5 customer_name operate_time
	非分区	100	0	2.34	1.58	1.78	0		python doris_batch_test_v2.py d_apply customer_name operate_time
	索引	100	0	1.47	1	1.22	0	身份证	python doris_batch_test.py d_apply_v5 id_number
	未加索引	100	0	3.07	1.96	2.31	1		python doris_batch_test.py d_apply id_number
	索引	100	0	2.38	1.03	1.3	0	申请编号	python doris_batch_test.py d_apply_v5 serial_no
	未加索引	100	0	2.14	1.44	1.82	0		python doris_batch_test.py d_apply serial_no
	索引	100	0	2.38	1.1	1.46	0	客户编号	python doris_batch_test.py d_apply_v5 customer_id
	未加索引	100	0	3.35	1.81	2.48	6		python doris_batch_test.py d_apply customer_id
	索引	100	0	1.95	0.99	1.34	0	河行客户号	python doris_batch_test.py d_apply_v5 bhh_customer_id
	未加索引	100	0	2.31	1.4	1.77	0		python doris_batch_test.py d_apply bhh_customer_id
	索引	100	0	1.29	0.1	0.39	0	白名单渠道+申请时间	python doris_batch_test_v2.py d_apply_v5 white_channel apply_time
	未加索引	100	0	1.79	0.7	0.82	0		python doris_batch_test_v2.py d_apply white_channel apply_time
	索引	100	0	2.88	1.03	1.34	0	白名单渠道+更新时间	python doris_batch_test_v2.py d_apply_v5 white_channel operate_time
	未加索引	100	0	1.34	0.69	0.8	0		python doris_batch_test_v2.py d_apply white_channel operate_time
	索引	100	0	2.01	1.19	1.66	0	决策阶段+更新时间	python doris_batch_test_v2.py d_apply_v5 decision_stage operate_time
	未加索引	100	0	1.44	0.8	1.09	0	决策阶段+更新时间	python doris_batch_test_v2.py d_apply decision_stage operate_time
	索引	100	0	0.28	0.04	0.14	0	申请时间	python doris_batch_test_v3.py d_apply apply_time
	未加索引	100	0	0.32	0.03	0.14	0	更新时间	python doris_batch_test_v3.py d_apply operate_time

	说明	场景
总结	按照时间字段进行分区后对于基于分区的条件查询，会有明显的提速	姓名 + 申请时间
	常用查询字段通过索引技术：布隆索引（高基数列）+位图索引（低基数列），会有提速	身份证 申请编号 客户编号 河行客户号
后期安排	离线数仓的数据出仓时，由于doris集群负载高，集群存在压力需持续观测 将上述测试做成常规化 24小时监测，并针对实时审批表接入的数据稳定性、时效性做出指标监	

运维管理

通过 Doris 提供的 Prometheus 和 Grafana 可以快速获取 Doris 集群的整体健康状况以及各个角色的多方面指标值。同时我们还将监控平台与公司统一告警平台进行二次融合，告警平台可以通过 API 获取 Prometheus 的基础指标值与阈值进行比较，从而触发不同级别的报警或者达到服务自动重启。此外，我们在 FE 和 BE 服务级别上实现任务的自动运维，确保在服务异常时能够自动拉起，保证核心服务的可用性。

总结收益

Doris 已经在公司内部得到了广泛应用，目前已搭建数十台集群规模，为公司带来了以下收益：

- **数据处理时效提升：**数据处理的时效从 T+1 到实时，解决了离线数据延迟的问题。
- **秒级查询响应：**借助 Doris 分区分桶、物化视图、布隆索引等功能进行查询优化，即席查询的速度从原先的 20 分钟左右降低到分钟甚至秒级响应，相较之前有 **400 倍**的速度提升。

- **统一查询出口：**依赖于 Doris 强大的导入能力和 Multi Catalog 功能成功将各业务库的数据整合汇总到 Doris 中，由 Doris 统一提供数据查询及分析服务，极大的提升了查询分析响应效率。

- **提升数据质量：**基于 Doris 建立了数据质量监控系统，目前我们已经将 30% 的数据监控指标和 35 个业务指标迁移到 Doris 实时集群上，有效提升了离线数仓跑批的数据质量。

综上所述，Doris 在公司内部的广泛应用，为我们带来了多方面的收益，助力企业提升数据分析效率、降低数据管理成本、实现统一、实时、高效的数据中台建设，为业务向好发展注入了新的动力。

未来我们将继续扩大 Doris 的使用范围，在实时、性能、时效要求更高的业务领域发力，其次我们还将尝试使用 Doris 更多的功能及新特性，一方面深化 Doris 在公司的使用，另一方面我们会将真实的使用体验反馈到社区，帮助 Doris 进一步迭代优化。

Apache Doris

在金融壹账通指标中台的应用实践

作者：金融壹账通数据挖掘工程师，兰后

导读：金融壹账通作为中国平安集团的联营公司，依托平安集团 30 多年金融行业的丰富经验及自主科研能力，向客户提供“横向一体化、纵向全覆盖”的整合产品，以“技术+业务”为独特竞争力，帮助客户提升效率、提升服务、降低成本、降低风险，实现数字化转型。在搭建数字化解决方案的过程中，面对传统报表制作过程中指标口径不统一、计算重复与交付效率低等痛点，金融壹账通决定基于 Apache Doris 搭建一体化指标数据服务平台，实现指标集中构建和管理、减少 ETL 开发工作量等业务目标。本文将详细介绍金融壹账通两代架构的演进过程，分享数据服务平台的建设经验与应用实践，向大家展示如何基于 Apache Doris 在多表关联与高并发场景下实现毫秒级查询响应。

金融壹账通是面向金融机构的商业科技服务提供商（Technology-as-a-Service Provider），为国家高新技术企业。作为中国平安集团的联营公司，金融壹账通依托平安集团 30 多年金融行业的丰富经验及自主科研能力，向客户提供“横向一体化、纵向全覆盖”的整合产品，包括数字化银行、数字化保险和提供金融科技数字基础设施的加马平台，以“技术+业务”为独特竞争力，帮助客户提升效率、提升服务、降低成本、降低风险，实现数字化转型。

业务痛点

在搭建数字化解决方案过程中，我们主要利用指标（如银行经营业绩指标：客户 AUM）来直观地反映企业经营状态，通过指标开发报表以帮助业务人员进行数据分析，进而驱动管理决策、赋能数字化经营。我们早期的报表制作方式是由不同的业务线人员根据自己的业务范围，使用不同的分析工具去定义指标，这种传统的方式在跨业务合作时会带来两大痛点：

- **指标口径、标准不统一：**各个业务线生成的报表堆积如山，由于使用不同分析工具，使对接数据源多样复杂，导致指标口径相互打架的问题。

- **指标计算重复、交付效率低：**开发流程需要业务方提出后，由 IT 人员下探数据源并加工，再制作报表、上线验收。整个过程中，IT 需要和业务多次沟通来同步信息，导致普通报表开发需要两周时间完成。

为了解决这两大问题，我们集团内部决定自研一体化指标数据服务平台，通过建立指标体系贴合客户战略目的，借助服务平台规范开发流程和使用方式，实现指标集中构建和管理。同时，使用 OLAP 查询引擎助力指标开发与应用，让业务人员能够快速找到所需数据，减少 ETL 开发工作量、缩短报表开发周期、加速指标发布与可视化看板生成的时间。

在数据服务平台建设过程中，金融壹账通经历了两代数仓架构演进。第一代架构基于 Apache Kylin 预计算的方式查询指标数据，并在架构使用后，发现其查询性能不足的问题。为了满足业务诉求，我们进一步开展 OLAP 选型调研，最终引入 Apache Doris 进行架构升级，借助 Apache Doris 的高性能分析能力为指标高效查询保驾护航。

本文将详细介绍金融壹账通两代架构的演进过程，分享如何基于 Apache Doris 搭建指标统一构建、查询、治理的一体化数据平台，并在多表关联与高并发场景下实现毫秒级查询响应。

早期架构挑战

架构 1.0： Hadoop + Presto + Apache Kylin

在业务初期，我们基于 Apache Kylin 进行 T+1 报表开发，上图是指标构建和查询的过程。在指标构建过程中，开发人员会根据选择的指标和维度进行 SQL 拼接， 通过 API 调用 Apache Kylin 的方式对各个维度进行与计算，完成模型构建和数据加载。在指标查询的过程中，采用快速查询和下压查询的组合策略，如果查询字段命中 Cube， 我们可以在 Apache Kylin 直接查询；如果没有命中，则下压至 Presto 再进行查询。

随着业务量不断增长，使用平台的业务用户越来越多，在面向客户推广与集团内部使用过程中，我们发现该架构在以下方面表现不足，无法满足我们的业务诉求：

- **灵活分析：**Apache Kylin 预计算只能满足部分场景需求，没有办法满足更灵活的分析需求。
- **查询性能：**当查询字段未命中 Cube 时，需要下压至 Presto。而 Presto 的查询性能得不到保障，特别是在查询码值的场景下，会遇到查询超时的现象，阻碍指标发布。
- **使用与运维成本：**Apache Kylin 架构在查询与开发过程中需要使用多套组件，造成了过高的维护成本。

基于第一代架构的使用经验，我们急需找到一个既可支持指标多表关联查询的场景，又可以达到降本增效的 OLAP 引擎。带着这样的诉求，我们对比了当下比较热门的 OLAP 引擎进行系统选型，从多表关联场景、使用协议、使用成本、金融应用场景与案例四大方面进行比较。

OLAP 选型对比

OLAP	优点	缺点	是否开源	协议
Clickhouse	• MPP • 单表查询性能强 • 保留明细 • 物化视图与索引	• 多表 Join 支持差 • 并发度低 • 实时更新支持一般	开源	非标准协议
Apache Doris	• MPP • 高并发和高吞吐的 Ad-hoc 查询 • 多表 Join • 支持明细和聚合 • 物化视图与索引引 • 实时+离线数据导入 • Schema 在线变更 • 支持一定程度更新	• 社区相对年轻 • 产品迭代较快	开源	兼容 MySQL协议
TiDB	• HTAP • 架构简单	• OLAP 能力不足	开源	兼容 MySQL协议
Greenplum	• HTAP • 支持多级分区 • 多表 Join	• 单机性能瓶颈	开源	兼容 MySQL 协议

我们首先排除了 TiDB ，主要因为其更倾向于满足 TP 需求，在应对大数据量分析场景时性能相对不足。其次，我们也排除了 Clickhouse 和 Greenplum。由于 Greenplum 单机性能较差，不适用于我们的查询场景；Clickhouse 虽然在单表查询性能表现不错，但是不支持 MySQL 协议，多表 Join 无法发挥性能，因此两款产品均不能满足我们对于海量数据在多表关联场景下的查询诉求。

最终，在集团内部其他子公司的使用反馈与推荐下，我们发现 Apache Doris 非常符合我们的诉求，并坚定不移地选择 Apache Doris 进行架构升级，主要原因如下：

- **开发简易方便：**Apache Doris 不仅兼容 MySQL 协议，还能够支持标准的 SQL 查询语法使开发简单方便。
- **复杂场景多表关联查询性能：**Apache Doris 支持分布式 Join、明细聚合等方式，在进行多表 Join 时能够提供多种优化机制，提升查询效率。同时 Apache Doris 还支持物化视图与索引功能来完成预计算效果，在命中物化视图时实现快速查询响应。
- **运维简单、方便扩展：**Apache Doris 的整体部署只有 FE 与 BE 两种角色，极大简化了架构链路，使架构无需再依赖其他组件，实现低成本运维。

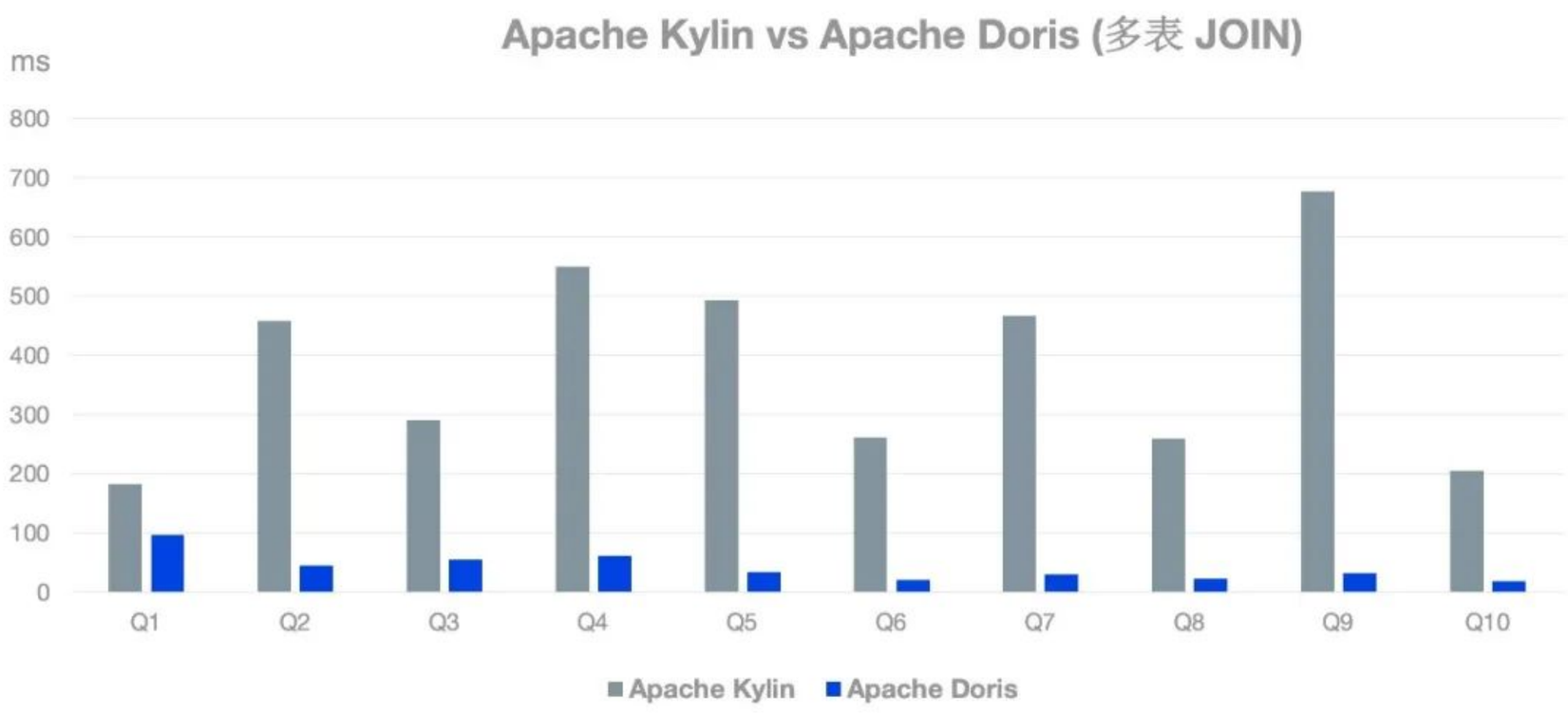
基于 Apache Doris 架构升级

架构 2.0: Apache Doris

基于 Apache Doris 的性能优势，我们从数据迁移和应用两方面进行了架构升级。在数据迁移过程中，Apache Doris 替代了第一代架构中 Apache Kylin 与 Presto，统一进行指标数据存储、处理、计算，并利用 Duplicate Key 模型对明细数据进行查询，使用 Range 进行时间分区并制定维度关联键作为 Key，有效解决了早期架构中 Presto 明细查询时性能不足、并发不够的痛点。同时，Apache Doris 在查询引擎方面采用了 MPP 模型，**具备高并发、低延迟的计算能力，使节点间和节点内都能够并行执行，支持多个大表分布式 Shuffle Join，能够满足我们对复杂场景下多表关联查询的需求。**

在应用方面，我们重写了 MySQL 兼容的查询引擎，当使用指标平台进行查询时，不再需要借助架构 1.0 中 Apache Kylin 调用接口、从页面中点击重跑指标等一系列比较繁琐的工作，**开发人员可以基于 Apache Doris 直接使用 MySQL 语法进行查询，极大简化了指标发布过程。**

Apache Kylin vs. Apache Doris



我们选取了指标平台常见的多表关联场景对 Apache Kylin 与 Apache Doris 进行性能对比，发现 Apache Doris 在查询性能与指标开发效率上表现都更为优异。如上图所示，Apache Doris 在数十万数据集关联情况下，查询响应基本达到毫秒级。同时，我们不再需要等待 Apache Kylin 完成 Segment 构建后使用指标，指标发布从原来的平均 30 分钟到现在的即发即用，显著提升指标开发效率。

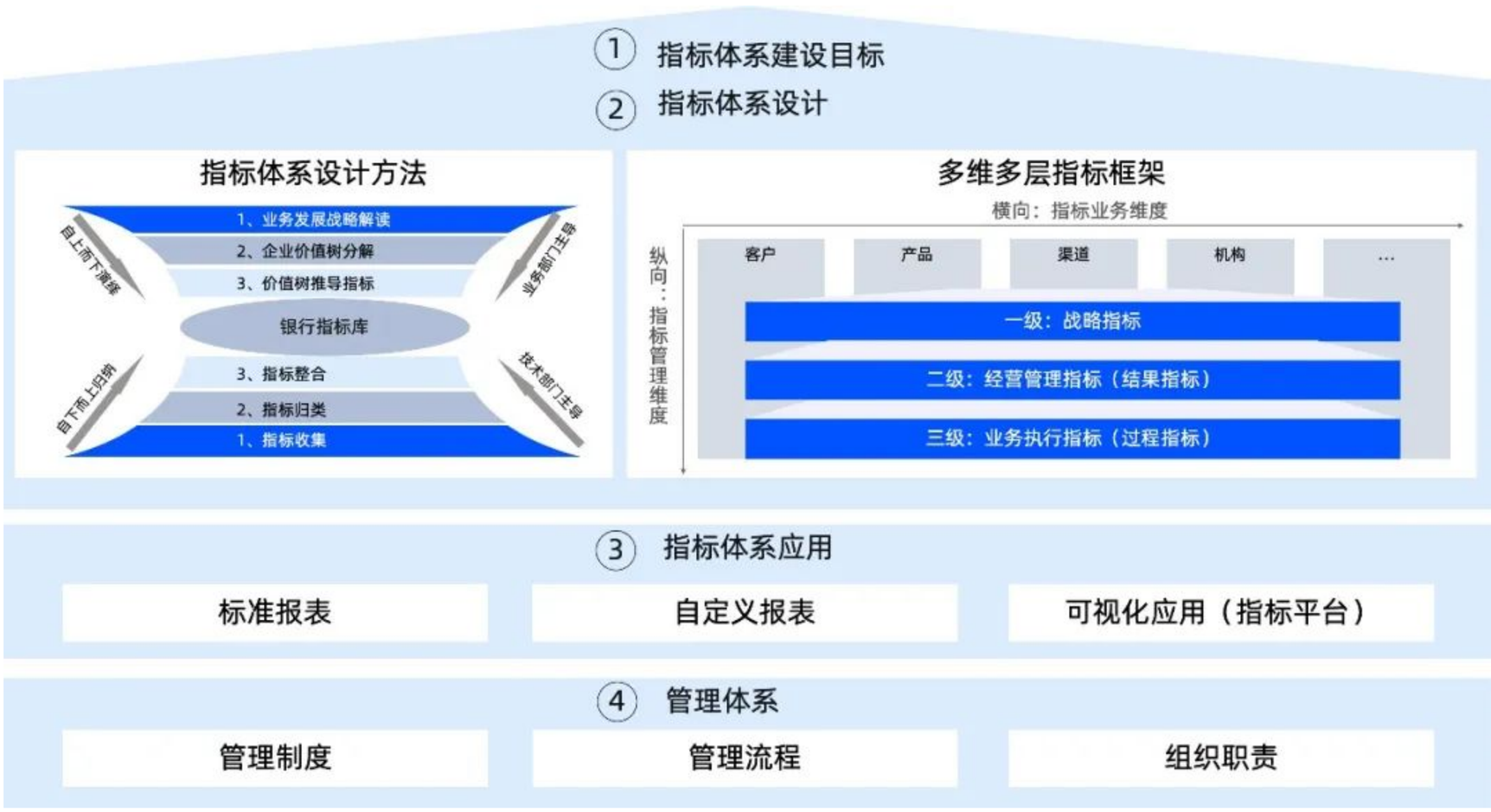
Apache Doris 的引入还大大节省了指标存储空间，符合集团降本的需求。集团内部的其他业务线（产险、寿险）也因此开始对 Apache Doris 进行铺开使用。新架构的升级不论是从硬件、人力、时间上都实现了非常高效能的提升，成为一体化数字指标平台建设的强大后盾。

一体化指标数据平台

在架构升级完成后，我们可以建设统一的指标体系，通过指标内容、BI 与 AI 技术构建平台功能，共同建设一体化指标数据平台。

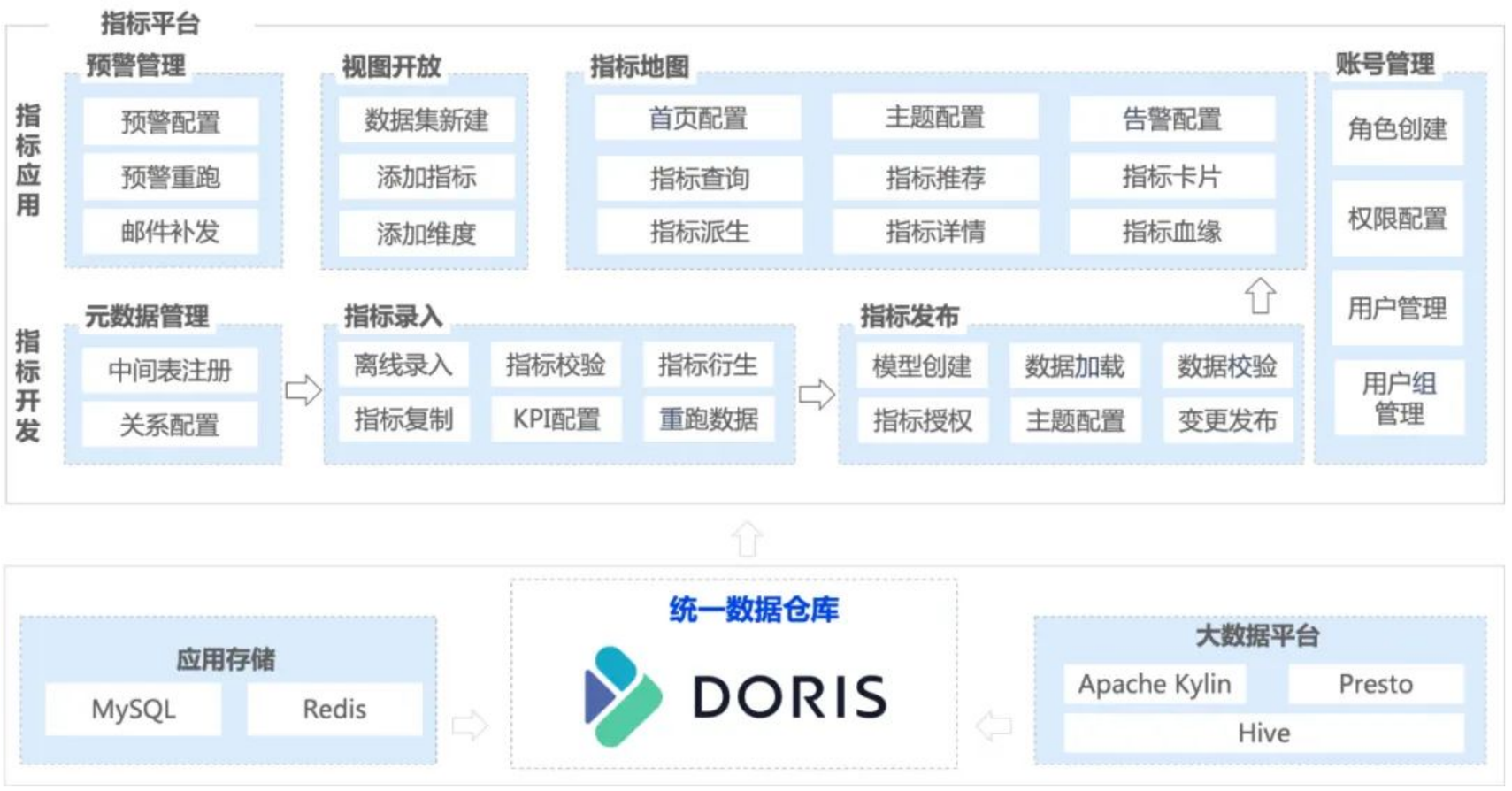
构建指标体系

金融壹账通借助归因关系分析帮助机构自上而下对指标进行建设，梳理核心 KPI 并逐层拆建指标，保障指标体系的完整性与可落地性。根据指标生成方式，将指标类型进行细分，以银行营销场景举例，针对银行资产管理中对客户资产总值的衡量指标（AUM）可以细分为以下三种类型：



- 原子指标：通过数据源接入到指标平台的最细粒度指标，一般为表字段，例如 AUM 余额。
- 衍生指标：为了进一步指标分析，平台自动衍生一系列指标，如 AUM 同比、环比净增等。
- 派生指标：为了满足复杂的指标分析场景，基于原子指标，添加过滤条件或者结合其他指标进行运算，帮助用户自助配置看板，节省取数过程。例如用户希望生成客均 AUM 余额进行分析，平台可以借助原子指标 AUM 余额与全量客户数生成该指标。

构建指标平台功能



指标平台的功能实现主要依赖于 Apache Doris 数仓架构的支持，整体指标线上流程基于开发和业务配合完成。开发人员首先统一在平台进行元数据管理和指标录入，包括对加工报表的底表进

行注册，配置中间表的数据粒度和更新频率等，接着对表进行关联、录入指标名称和指标口径信息。在输入指标基础信息之后，交由业务人员负责，选择对指标分析所需维度，对指标进行发布。

基于以上两个步骤，我们可以在平台中对指标数据进一步分析。如上图左侧所示，指标平台提供了各种柱状分析视图，业务人员能够可视化地查看指标排行榜看板，分析各银行分行 AUM 排名情况。同时，我们融入了 AI 智能算法，借助时序模型检测指标异常，通过根因分析算法辅助 KPI 检视，并分析指标异动原因。对于存量指标，平台提供了价值评分体系，能够及时下线价值低的指标，达到边使用边治理的目的。

基于 Apache Doris 指标应用实践

一体化数据平台的建设完全解决了金融壹账通在传统报表开发时指标口径不统一和指标重复计算的问题。在分析效率方面，我们希望在复杂的多表关联场景下，实现接口 600 毫秒响应时间、查询响应在 100 毫秒内的目标。因此，我们对 Apache Doris 进行了测试与调优，从数据的前期准备、集群部署、模型调优三方面分享 Apache Doris 在该场景下的应用实践。

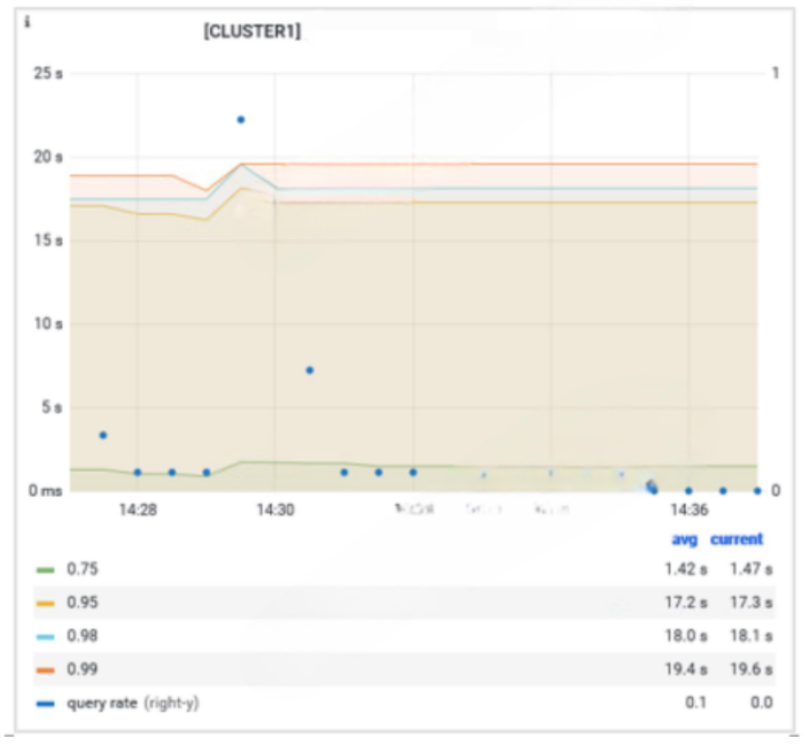
在前期数据准备过程中，考虑到我们的数据集和官网测试的 SSB 数据集很相似，我们选择了官网推荐的开发测试环境配置，选用 Apache Doris 1.1 版本进行测试。因为我们是通过 Python Mock 数据直接生成 CSV 文件，所以我们采用 Stream Load 的方式分批导数，每次导入的 CSV 文件都在 Stream Load 推荐的文件大小 1 – 10G 以内，最终数据压缩比达到 3 : 1 ，但单节点导入速度超过 40 MB /s。

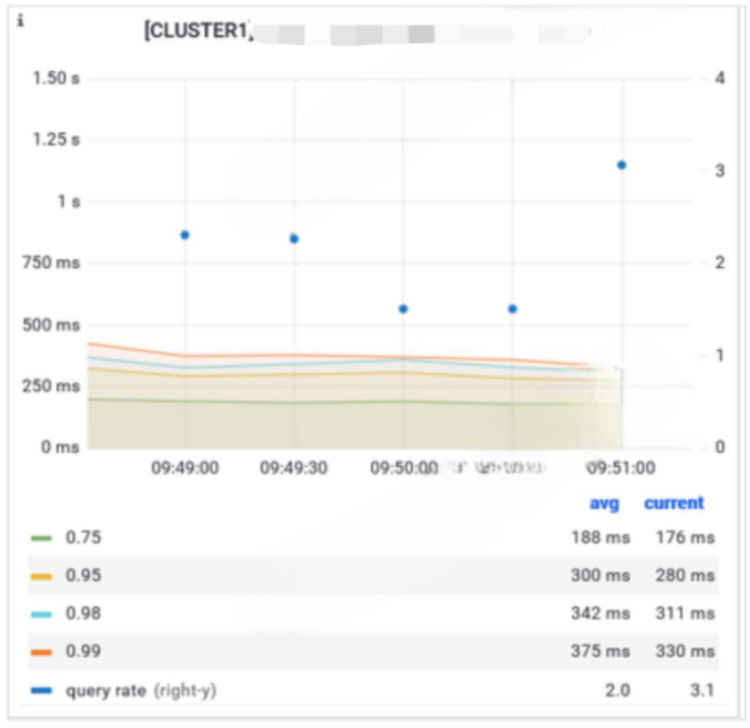
在集群部署过程中，为了对指标性能和服务器监控（CPU、IO、磁盘和内存），我们借助 Prometheus 导入 Apache Doris 监控模版对集群部署监控，由 Prometheus 接收 Apache Doris 暴露监控项，再借助 Grafana 进行可视化呈现。

在准备工作完成后即可开始进行大表关联查询，我们选择了耗时较长的 SQL 来查询指标趋势图。基于毫秒级查询目标，我们实施了两个优化解决方案。第一个方案是利用 Colocation Join 将数据在建表时提前聚合。第二个方案是借助 Audit Loader 的方式收集高频 SQL，反向优化数仓的表构建以及改写 SQL，使用偏宽表设计代替之前的星型 / 雪花模型。通过两个方案的测试与评估，我们发现第二个方案能够在查询响应、服务资源节省中达到更加显著的收益。

亿级数据多表关联查询，实现毫秒级查询响应

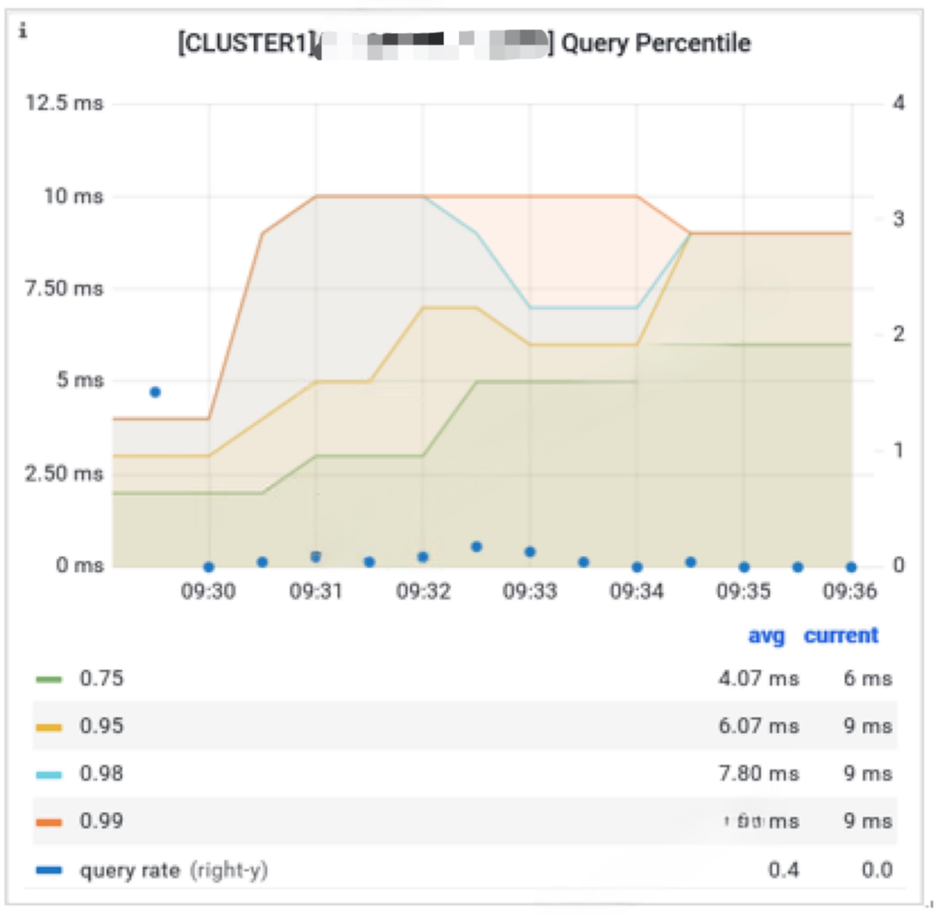
我们将 SQL 查询执行时间进行了统计，如上图所示在采取方案一 Colocation Join 的方式时，查询响应时间从之前的 5 秒提升至 1 秒。虽然查询效率有所提升，但是我们希望能够更进一步缩短响应时间，完成预期目标。在采用方案二来调整数据模型后，SQL 执行时间从原来的 5 秒达到 63 毫秒响应时间，查询响应时间得到显著提升，满足我们对查询响应毫秒级的目标。





同时，我们借助 Grafana 查看 Apache Doris 查询性能，发现宽表构建的方案能够使查询时间从原来的十多秒缩短至百毫秒内，服务器也不再出现抖动的情况。

启用 SQL 缓存，节省服务器资源



采取宽表构建方案后，为了进一步提升查询性能，我们还启用了 SQL 缓存，帮助 T+1 报表场景实现高效查询性能：

- 在启用缓存之后，基本所有查询时长都在个位数，最终达到单用户访问页面在 4 秒内加载的成果；
- 在 30 个指标同时进行时（SQL 指令超 120 条），接口都可以满足 600ms 内返回；
- 在并发场景下，最优 TPS 达到 300， CPU、内存、磁盘和 IO 满足 80% 以下；
- 经评估，我们发现在官网推荐的测试集群规模下，Apache Doris 都可以缓存上万指标，极大节省了资源。

未来规划

目前，金融壹账通基于 Apache Doris 实现了指标统一构建、查询、治理的一体化数据平台，为金融机构提供了全面的指标分析与展示，智能的指标生命周期管理等服务。在这样的平台建设下，**集团内外多场景取得了非常显著的成果，截止目前，完成上万活跃指标、上千分析维度的积累，加工形成了上万个看板，减少了 30 % ETL 开发工作量。**未来，公司将基于 Apache Doris 不断探索与优化，我们将重点推进以下几个方面的工作：

- 平台实时分析：基于 Apache Doris 构建湖仓一体，结合 Flink CDC、Apache Iceberg 共同构建统一实时分析。
- 平台物化视图：期待新版本亮点，探索多表关联下的查询优化，比如构建多表物化视图。
- 其他产品迁移：将中台其他产品迁移至 Apache Doris。目前，标签平台基于 Elasticsearch 存在一定的使用问题，未来我们也准备将该平台迁入 Apache Doris。

在此特别感谢 SelectDB 技术团队和 Apache Doris 社区在使用过程中遇到任何问题都能及时响应，为我们降低了许多试错成本。未来，我们也会更积极参与社区贡献及活动，与社区共同进步和成长！

星云零售信贷

基于 Apache Doris 的 OLAP 演进之路

作者：腾梭科技研发总监，刘建波

导读：腾梭科技是国内领先的零售金融数字化及安全服务提供商，是腾讯投资且在金融领域的战略合作伙伴，并与腾讯联合研发了“星云智慧信贷解决方案。在其信贷业务转型过程中，随着系统规模不断扩大，早期架构无法再满足业务需求，因此腾梭科技决定引入 Apache Doris 实现业务升级。本文将详细介绍其信贷业务如何基于 Apache Doris 在助贷系统日志分析场景的实践应用，实现毫秒级并发查询响应与极致存储性价比的性能表现。

腾梭科技是国内领先的零售金融数字化及安全服务提供商，是腾讯投资且在金融领域的战略合作伙伴，专注于为持牌金融机构提供零售金融业务全流程解决方案。结合核心团队多年风控与反欺诈经验，腾梭与腾讯联合研发了“星云智慧信贷解决方案“和“伽利略智慧风控系统”，全方位解决金融机构在零售金融业务中遇到的技术难题，推动金融机构信贷风险管理由策略驱动转向人工智能模型驱动，增强业务精准营销能力，助力银行实现普惠金融，延伸金融服务广度和深度，打通服务实体经济的“最后一公里”。

作为互联网金融科技公司，腾梭科技由早期的网贷转型为以助贷为核心的业务模式，联合行业内各大友商行共同提供信贷服务。在获得一定客户群体基础后，腾梭科技信贷服务开展了全量客群的挖掘，开启线上以及线下结合或纯线上贷款全流程业务模式的探索，为新零售金融场景提供微服务技术应用，包括潜在客户信用评估、拒贷款分析、贷后管理、预期催收以及风控管理等方面。

基于以上业务模式的演进，系统技术架构也随之发生了变化与升级，从集中式的单机系统转化为分布式场景。由于引入了各大合作银行的 TP 系统（如 MySQL 、Oracle、PostgreSQL 等），使系统规模不断扩大，当传统的关系型数据库无法完成多种数据源之间的数据关联时，架构逐渐出现数据孤岛与数据割裂的现象，因此 OLAP 引擎的应用成为打破数据孤岛必不可少的工具。对于助贷系统的架构升级，如何进行高效数据采集和处理、如何在庞大的客户数据中准确了解潜

在客户需求与信用情况以加快贷款审批速度、如何通过客户数据管理实现高校精准获客、赋能营销策略，成为了我们业务重点发展方向。

为了满足这些要求，腾梭科技历经三代架构演进。第一代架构基于 Kettle + MySQL 离线数仓，第二代架构引入 Trino 进行统一查询，在经过两代架构使用后发现其性能的不足，决定引入 OLAP 引擎。最终通过产品调研选择 Apache Doris 作为第三代架构核心进行数据统一存储与分析。本文将详细介绍三代架构的演进历程和应用实践，分享业务场景落地经验。

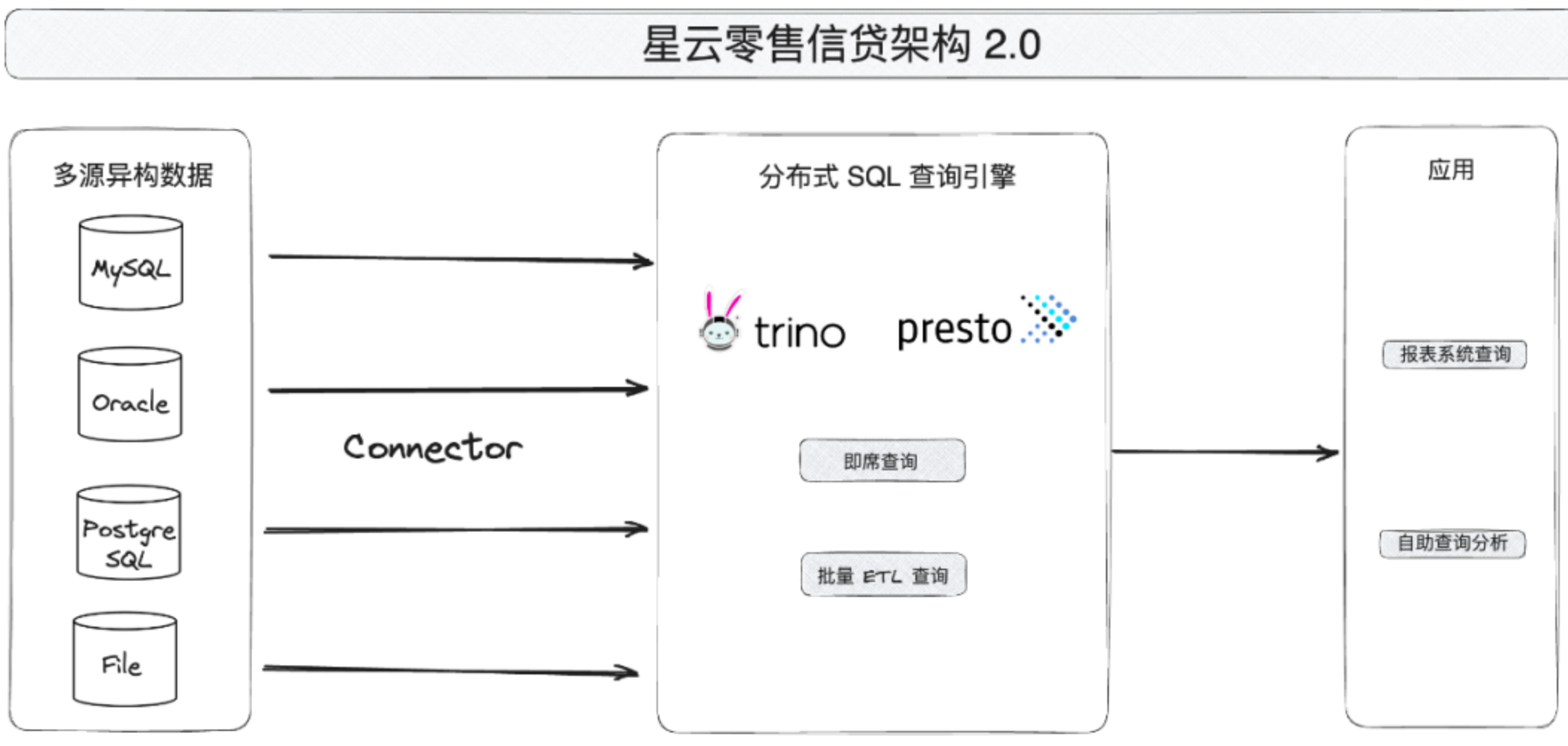
早期架构演进

架构1.0：基于 Kettle + MySQL 离线数仓

在业务初期，我们使用 Flume Sink 进行数据采集，利用 DolphineScheduler + Shell 进行数据调度，基于 Kettle 抽取离线数据进入关系型数据库中形成离线数仓，进行基础的 T+1 报表取数工作。由于 Kettle 仅支持离线取数的功能，不支持数据存储，因此数据始终保存在原始端。随着数据量的不断增大，当事实表条数达到千万级时，Kettle 的性能逐渐变得不稳定，单表查询任务的执行时间出现延迟现象，无法满足较大业务规模的使用需求。同时，Kettle 不支持多数据源之间的关联查询功能，在 TP 系统多样的情况下，查询效率无法得到保障。

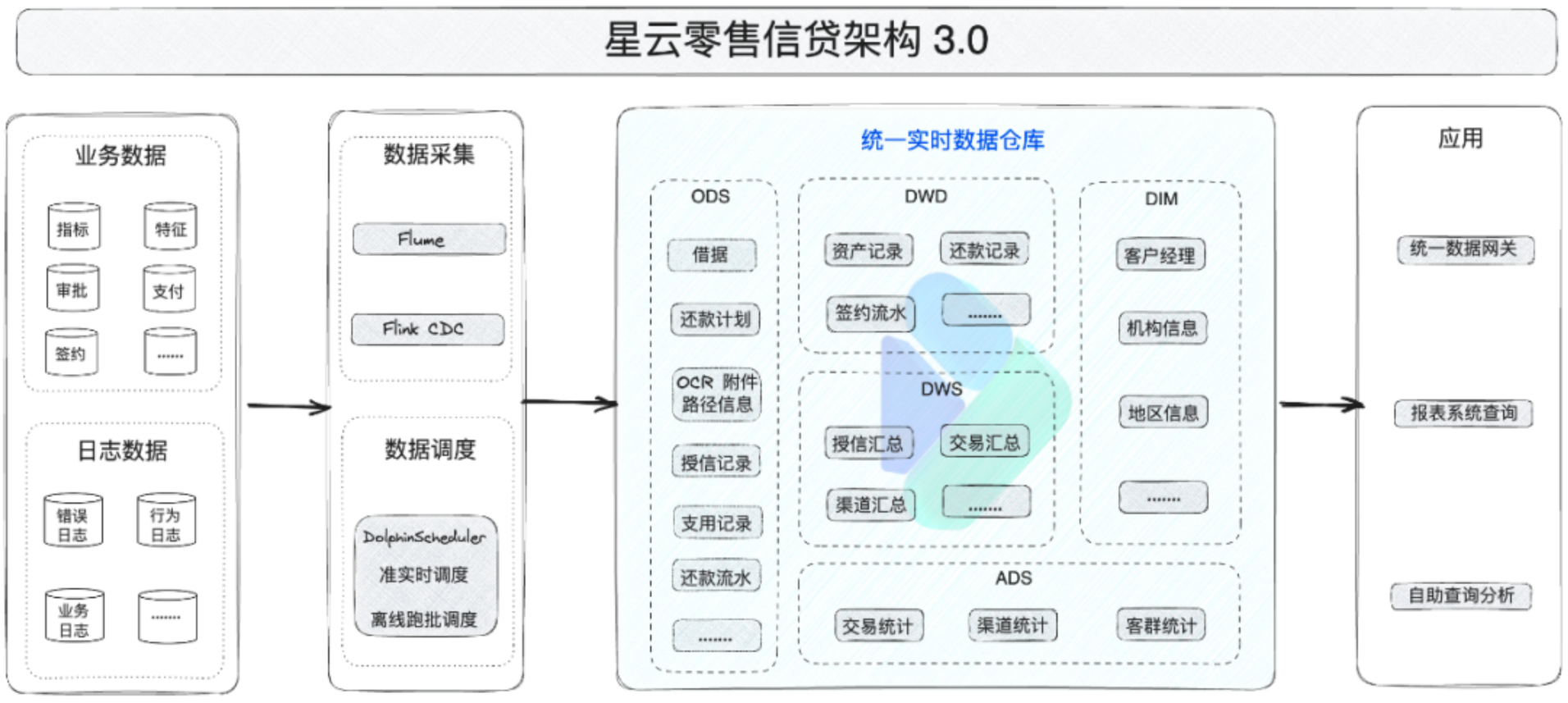
架构 2.0： 基于 Presto / Trino 统一查询

针对第一代架构存在的问题，我们在第二代架构升级中借助 Trino 作为分布式 SQL 查询引擎进行联邦查询，实现多种类型数据源的即席查询和批量 ETL 查询，打通信贷、风控之间的多源异构数据查询需求。由于 Trino 缺少存储和管理元数据的功能，在面对高并发点查场景下导致联合查询响应较慢，查询效率依旧无法得到改善。



基于 Apache Doris 的新一代架构

为了彻底解决早期架构的问题，我们重新整理了架构的核心诉求：满足企业数据规模、支持灵活关联查询、架构使用和运维成本可控。基于此，我们对当下热门的 OLAP 产品进行了调研比对，如 Apache Doris、Clickhouse 等 MPP 数据库以及除 Presto 以外的其他 SQL on Hadoop 相关引擎。我们首先放弃了 SQL on Hadoop 这一类产品，因为其技术生态太过于庞大、涉及组件过多，考虑到架构投入产出比，可能造成团队的负担，成为技术债务。其次我们放弃了 Clickhouse 选项，主要因为它不支持 MySQL 协议、学习成本高、在多表 Join 查询性能中表现较差、对组件依赖较高等问题，并且开发人员需要花费大量成本在扩容与运维工作中，不满足我们的核心诉求。最终，我们发现 Apache Doris 不论是在大规模数据量下的查询性能还是使用难度与运维成本等方面都具有一定优势，因此决定引入其进行架构重构。



如上图所示，银行各类业务数据与用户日志由 Flume 与 Flink CDC 进行数据采集、DolphinScheduler 进行数据调度写入数仓。 Apache Doris 实时数仓主要负责数据分层存储与汇总处理，为应用端提供报表开发、查询分析等功能。在 ODS 层中，我们主要利用 Apache Doris 存储客户在发起贷款申请后所产生的身份证 OCR 识别附件、相关征信数据授权（如还款流水、支用记录、公积金或税务）等第三方数据，其中身份证 OCR 附件存放于对象存储中，ODS 层中主要负责存放其在对象存储的 URL 路径信息。这些原始数据会通过 DWD 与 DWS 层进行标签分类汇总，最终在 ADS 层形成各类统计数据，供前端业务人员查询与分析。

在搭建过程，Apache Doris 的高性能、极简易用、实时统一等诸多特性使我们的实时数据流程架构变得简单，大大降低了维护和使用成本。新架构的升级为我们优化了早期架构的痛点，具体表现如下：

- **元数据管理**：Apache Doris 通过对外 API 提供元数据管理功能，彻底解决了早期架构中多源异构数据无法联合查询的痛点，实现在各 TP 系统中无缝衔接地进行数据开发。

- **查询性能提升**：Apache Doris 完全实现了向量化查询引擎，能够胜任各种查询并发、吞吐的场景并且性能表现强悍，解决了第二代架构中 Trino 在查询并发响应慢的问题。
- **运维难度低：** Apache Doris 基于 Sytemd 进程保活，具备多副本+ 副本自动均衡机制，除了需要定时备份元数据外几乎可以达到零运维，极大降低了运维成本与难度，实现降本增效的需求。
- **使用简单：**Apache Doris 兼容 MySQL 协议，能够支持使用标准 SQL，不仅极大降低了业务人员的学习成本，还轻松实现 MySQL 业务迁移至 Apache Doris，带来开发效率的提升。

新数仓架构搭建经验

在新架构搭建完成后，我们开始基于 Apache Doris 进行应用实践，通过并发查询加速与数仓底座建设两方面助力复杂场景下的业务应用。以下是我们总结出来的一些经验：

并发查询加速

风控分析是星云零售最最常见的业务，由于金融交易系统会涉及大量的交易日志与明细日志等数据，存在大量高并发低延时的点查询以及高吞吐低并发的大表关联等需求场景，需要在多场景下保持一致的高性能分析体验，因此我们最重要的实践就是并发查询加速。

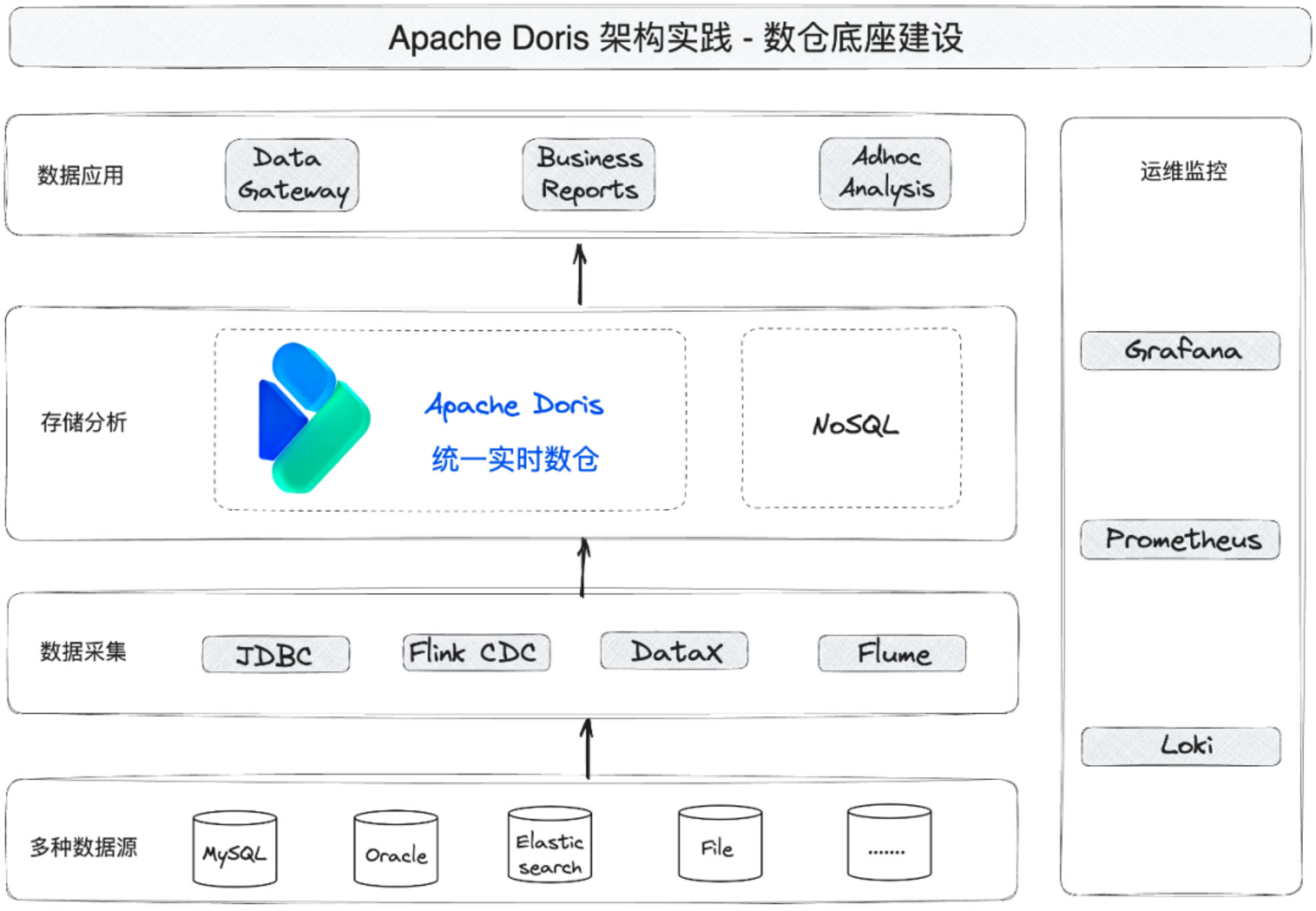
```
    `asset_item_no`                 VARCHAR(48)    COMMENT '项目编号，有一系列规则，用来标识某个项目，可以从项目编号里看到很多信息'
    `asset_type`                    VARCHAR(32)    COMMENT '资产类型',
    `asset_interest_type`           VARCHAR(32)    COMMENT '计息方式：等额本息-eq_loan_pmt,等额本金-eq_principal_pmt',
    `asset_period_type`             VARCHAR(64)    COMMENT '还款周期类型',
    `asset_period_count`            TINYINT(3)     COMMENT '还款总期数',
    `asset_real_period_count`       TINYINT(3)     COMMENT '实际放款期数',
    `asset_product_category`        VARCHAR(64)    COMMENT '资产类别名称：14天,30天,3个月,6个月',
    `asset_biz_order`               VARCHAR(48)    COMMENT '业务订单号',
    `asset_capital_order`           VARCHAR(48)    COMMENT '资金方订单号，即银行的资产号',
    `asset_loan_account`            VARCHAR(32)    COMMENT '贷款账户',
    `asset_create_at`               DATETIME       COMMENT '创建时间',
    .....
    `etl_time`                      DATETIME       COMMENT 'ETL时间'
) ENGINE=olap
UNIQUE KEY(`asset_item_no`)
DISTRIBUTED BY HASH(`asset_item_no`) BUCKETS 16
PROPERTIES (
    "replication_num" = "2",
    "colocate_with" = "asset_group"
);
```

在引入新架构之前，我们使用 MySQL 预聚合的方式进行数据分库，这会造成 IO 与 CPU 消耗非常大的问题，导致 MySQL 系统崩溃。在引入 Apache Doris 之后，我们采用 Unique Key 模型对明细数据进行存储，引入 Aggregate Key 模型进行数据预聚合，为后续的物化视图与实时报表做准备。在社区的帮助下，我们还使用了逻辑分区和物理分桶进行了 Key 列的优化，利用 Colocation Join 的方式创建业务关联表模型，保证分区和分桶、分区键以及 Key 值统一一致。如上图所示，各业务人员在进行大表关联查询时，不需要再进行跨节点 Shuffle Join，可以直接通过本地节点查询，避免了数据在网络传输中带来的性能开销，有效提升了点查时高吞吐场景下的查询效率。

```
CREATE TABLE IF NOT EXISTS dwd.dwd_cust_info (
 pt_date             DATE          COMMENT '分区日期'
,cust_id             VARCHAR(128)  COMMENT '客户ID'
,cust_individual_id  BIGINT        COMMENT '前端个人ID,为空,则未实名'
,cust_name           VARCHAR(128)  COMMENT '姓名'
,cust_sex            VARCHAR(128)  COMMENT '性别'
,cust_tel            VARCHAR(128)  COMMENT '手机号码'
,cust_cert_id        VARCHAR(128)  COMMENT '身份证号'
,cust_cert_id_area   VARCHAR(128)  COMMENT '身份证号对应省份'
,cust_channel_1      VARCHAR(128)  COMMENT '一级渠道名称'
,cust_channel_2      VARCHAR(128)  COMMENT '二级渠道名称'
,cust_status         VARCHAR(128)  COMMENT '客户状态 froze 冻结,no_quali 暂无资格'
,create_at           DATETIME      COMMENT '创建时间'
,update_at           DATETIME      COMMENT '更新时间'
,cust_project_code   VARCHAR(32)   COMMENT '项目编号'
,etl_time            DATETIME      COMMENT 'ETL时间'
) ENGINE = OLAP
UNIQUE KEY(pt_date, cust_id)
COMMENT '客户信息表'
PARTITION BY RANGE(pt_date) ()
DISTRIBUTED BY HASH(cust_id) BUCKETS 16
PROPERTIES (
 "replication_num" = "2"
,"bloom_filter_columns"="cust_id,cust_individual_id"
);
```

除金融交易系统外，风控分析还需要进行特征指标计算与贷中行为分析等业务。Apache Doris 的 MPP 架构完全支持了业务所需的高吞吐和多表查询能力，并且在列表多维度查询时，可以根据不同的业务场景，借助其 Bloom Filter 物化索引机制进行 Key 列的优化设计。这种方式不仅改善了客户的查询体验，还能够大幅提升查询效率，达到毫秒级查询响应。

数仓底座建设



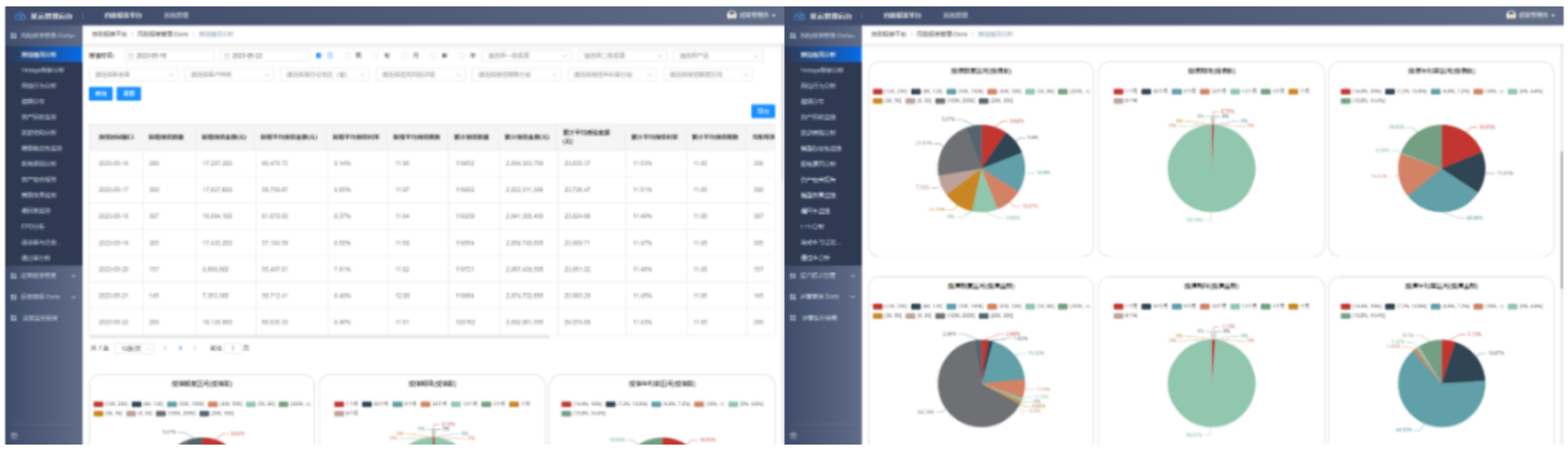
在与 B 端合作开展助贷业务过程中会产生大量的离线报表业务，因此，我们首先基于 Apache Doris 作为数仓底座，利用调度工具 DolphinScheduler、日志采集工具 Flume 以及数据同步工具 DataX 等进行数据采集。同时，通过增量或者全量的方式将数据从业务端或者异构数据源中采集落库至 Apache Doris 数据仓库中，形成数据集市。

在该集市中，业务人员可以方便地提取所需数据进行报表开发，并展示于实时交易大屏，以支持风控数据分析和业务决策。为了确保数仓稳定性和性能，我们利用了 Grafana 和 Prometheus 对集群状态进行监控，主要用于关注 Apache Doris 的内存使用情况、 ETL 过程中 Compaction 的稳定性以及查询响应时间。通过这些监控工具，可以帮助我们及时发现数据集市的运行效果与异常情况。

Apache Doris 助力信货业务场景落地

基于 Apache Doris 的功能实践，我们建设了星云零售管理后台、自助报表等一体化业务分析平台。接下来，我们主要介绍在业务场景落地过程中，风控大数据报表平台、统一日志存储分析与用户行为分析的业务实践。

交互式分析查询，实现风控大数据平台智能化



如上图所示，星云管理后台会对风控数据进行分析，涉及授信情况分析、用信分析、放款结构分析、拒绝申贷原因分析等报表业务，我们希望通过风控报表平台实现风控策略化、智能化，提升线上的风控能力、提高审批效率并完善信贷业务流程。以授信情况分析为例，具体的操作流程如下：

- **数据调度：**指标数据首先通过 DolphinScheduler 和 Shell 任务编排实现风控离线数仓各分层数据的调度与流通、统一管理。
- **数据同步：**借助 Apache Doris 的 JDBC Catalog 以 Insert Into 的方式，将多个外部源表中的数据增量导入数仓贴源层，实现统一建模、统一数据口径。

- **数据处理：**在 Apache Doris 的 DW 层中进行数据关联分析、聚合、日区分落盘等操作，最终结合维表数据共同创建物化视图或者落地大宽表。基于 Apache Doris 的分层存储与数据处理，我们的报表开发时间从天级别提升至小时级别，大幅提高报表开发的效率。

- **数据分析：**基于以上三个步骤，业务人员可以在平台中进行自定义交互式分析查询，如查询某一段时间内授信额度区间的占比，并以饼状图形式呈现。

极致性价比，达成统一日志存储分析

```
CREATE TABLE `error_log` (
  `event_id` varchar(32) NULL,
  `log_date` date NULL,
  `app_name`  varchar(10) NULL,
  `create_time` datetime NULL,
  `log_level` varchar(10) NULL,
  `class` varchar(200) NULL,
  `method` varchar(100) NULL,
  `msg` varchar(4000) NULL,
  `exception` text NULL
) ENGINE = OLAP
UNIQUE KEY(`event_id`, `log_date`) COMMENT 'OLAP'
PARTITION BY RANGE(`log_date`) ()
DISTRIBUTED BY HASH(`event_id`, `log_date`) BUCKETS AUTO
PROPERTIES (
  "replication_allocation" = "tag.location.default: 1",
  "dynamic_partition.enable" = "true",
  "dynamic_partition.time_unit" = "DAY",
  "dynamic_partition.time_zone" = "Asia/Shanghai",
  "dynamic_partition.start" = "-7",
  "dynamic_partition.end" = "3",
  "dynamic_partition.prefix" = "p",
  "dynamic_partition.replication_allocation" = "tag.location.default: 1",
  "dynamic_partition.buckets" = "10",
  "dynamic_partition.create_history_partition" = "false",
  "dynamic_partition.history_partition_num" = "-1",
  "dynamic_partition.hot_partition_num" = "0",
  "dynamic_partition.reserved_history_periods" = "NULL",
  "dynamic_partition.storage_policy" = "",
  "in_memory" = "false",
  "storage_format" = "V2",
  "disable_auto_compaction" = "false" );
```

Run successfully

```
select log_date,count(1) from error_log group by log_date order by log_date;
```

Execution Time: 2850 ms

log_date	count(1)
2022-12-28	115591396
2022-12-29	76440446
2022-12-30	73843088
2023-01-03	85245888
2023-01-04	46749024

Run successfully

```
select * from error_log  where exception like '%153%' order by create_time limit 10;
```

Execution Time: 1227 ms

星云零售在业务运营过程中会存在大量的日志存储分析场景，如使用 API 访问异常日志。在引入 Apache Doris 之前，我们使用 Grafana + Loki 进行多节点本地支持存储，这种方式不仅无法保证存储统一性，并且增加运维成本。

在引入 Apache Doris 后，我们基于 Stream Load 自定义开发 Flume Sink 与 Tail Dir 日志采集组件，能够支持动态配置，使节点灵活且易于扩展。我们还采用了 Apache Doris 的动态分区表模型，实现动态添加分区或者删除分区，减少了运维过程中的使用负担。更重要的是，Apache Doris 提供了极致的列存储压缩比，使存储成本大幅度下降，并且 **2.0 版本的倒排索引功能支持文本类型的全文检索，也能对普通数值日期的等值、范围查询进行加速，能够从海量数据中秒级检索出满足条件的日志，更加契合我们后续对日志数据分析的需求。**总而言之，基于 Apache Doris 的实时日志存储功能为我们提供了全面的实时预警监控、实时监控大屏、故障分析等能力，真正意义上实现统一实时的日志存储分析。

JSON 统一存储 + 丰富解析函数，助力用户行为日志分析

```
CREATE TABLE `api_log` (
  `request_time` datetime NOT NULL,
  `request_id` varchar(32) NOT NULL,
  `caller` varchar(16) NOT NULL,
  `client_ip` varchar(46) NOT NULL,
  `header` jsonb NOT NULL,
  `request_method` varchar(16) NOT NULL,
  `url` varchar(256) NOT NULL,
  `request_params` jsonb NOT NULL,
  `response_code` varchar(8) NOT NULL,
  `response_data` jsonb NULL,
  `request_microtime` double NOT NULL,
  `spend_microtime` double NOT NULL,
  `log_date` date NOT NULL
) ENGINE = OLAP
DUPLICATE KEY(`request_time`, `request_id`, `caller`) COMMENT 'OLAP'
PARTITION BY RANGE(`log_date`) ()
DISTRIBUTED BY HASH(`request_time`, `request_id`, `caller`) BUCKETS AUTO
PROPERTIES (
  "replication_allocation" = "tag.location.default: 1",
  "bloom_filter_columns" = "request_time, request_id, url",
  "dynamic_partition.enable" = "true",
  "dynamic_partition.time_unit" = "DAY",
  "dynamic_partition.time_zone" = "Asia/Shanghai",
  "dynamic_partition.start" = "-7",
  "dynamic_partition.end" = "3",
  "dynamic_partition.prefix" = "p",
  "dynamic_partition.replication_allocation" = "tag.location.default: 1",
  "dynamic_partition.buckets" = "10",
  "dynamic_partition.create_history_partition" = "false",
  "dynamic_partition.history_partition_num" = "-1",
  "dynamic_partition.hot_partition_num" = "0",
  "dynamic_partition.reserved_history_periods" = "NULL",
  "dynamic_partition.storage_policy" = "",
  "in_memory" = "false",
  "storage_format" = "V2",
  "disable_auto_compaction" = "false" );
```

```
select log_date,count(1) from api_log group by log_date order by log_date;
```

Execution Time: 855 ms

log_date	count(1)
2022-12-30	35098180
2022-12-31	6363808
2023-01-01	6428208
2023-01-03	69945387
2023-01-04	10402381
2023-01-05	10304243
2023-01-06	7160616

Run successfully

```
select  count(1) from api_log where log_date ='2022-12-30' and get_json_int(response_data,'$.code')=101018 group by log_date order by log_date;
```

Execution Time: 37637 ms

count(1)
1110981

在营收信贷业务过程中，我们会对潜在客户进行广告投放，通过自动获取用户行为日志数据，分析信贷需求来加强营销活动、提升获客效果，达到精准投放的目的。我们借助 Stream Load 自定义的日志采集工具收集用户在小程序或者 App 中的访问日志，**利用 JSON 统一存储功能与丰富的解析函数对行为日志进行实时查询分析、跑批离线宽表加工等操作。**

在这一过程中，**Apache Doris 的引入使用户行为日志降低 70% 的存储成本，**同时提供了丰富且开箱即用的用户行为分析函数，避免业务人员重复进行复杂 SQL 函数编写、验证、推导再应用，极大提高了数据开发效率，为后续广告精准投放提供了强有力的数据支持。

总结收益

当前，腾梭科技星云零售信贷业务基于 Apache Doris 搭建了高度统一实时的数据仓库，实现星云管理后台中的风控报表管理、运营报表管理、用户行为日志分析等信贷业务应用。 Apache Doris 的引入为我们带来以下收益与成果：

- **灵活数据分析**：不论是业务端还是数据开发端，都可以基于 Doris 支持自定义导数、动态配置，实现灵活及易扩展的多维数据分析。
- **查询快速响应**：从业务层面来看，现阶段的风控信贷点查、偏离计算等复杂场景都可以基于 Apache Doris 进行多表关联，并且实现毫秒级查询响应，大幅提升查询效率。
- **交付效率提升**：助贷业务的核心业务为客户管理，在引入 Apache Doris 后，其数据分层存储与开箱即用的分析函数，在用户行为、信用评估、风险控制等多方面提供了有效报表分析，以挖掘更多潜在用户，大幅提升交付效率，实现精准获客的目标。
- **综合成本降低**：与之前数据源端存储不同，Apache Doris 极致的存储压缩比，降低了 70 % 的存储成本。同时，Apache Doris 支持集群节点进程保活、自动均衡极致，几乎达到零运维，为公司运维成本控制提供了核心收益。

未来规划

未来，我们希望基于 Apache Doris 冷热分层技术实现统一的数据归档功能，将冷数据、历史数据定时进行归档，进一步优化数仓存储空间。同时，利用 Apache Doris 湖仓一体功能实现智能数据网关，使 Schema 列类型等元数据能够映射至 Apache Doris 的数据结构中，形成统一元数据映射结构，提供一致性的查询体验。

最后，感谢 Apache Doris 社区和 SelectDB 技术团队在数仓搭建过程中的积极响应与技术支持，未来我们也会持续参与社区活动，将相关成果贡献回馈社区，希望 Apache Doris 飞速发展，越来越好！

当 Apache Doris 遇上大模型：探秘腾讯音乐如何基于大模型 + OLAP 构建智能数据服务平台

作者：腾讯音乐大数据架构师，张俊、罗雷

导读：当前，大语言模型的应用正在全球范围内引发新一轮的技术革命与商业浪潮。腾讯音乐作为中国领先在线音乐娱乐平台，利用庞大用户群与多元场景的优势，持续探索大模型赛道的多元应用。本文将详细介绍腾讯音乐如何基于 Apache Doris 构建查询高效、实时统一分析的 OLAP 引擎，使 OLAP 作为底层基建加强模型连接转化效率、结果输出准确率，最终将大模型 + OLAP 引擎结合为用户提供个性化、实时化、灵活化的智能数据服务平台。

腾讯音乐娱乐集团（以下简称“腾讯音乐”）是中国在线音乐娱乐服务开拓者，有着广泛的用户基础，总月活用户数超过 8 亿，通过“一站式”的音乐娱乐平台，用户可以在多场景间无缝切换并享受多元的音乐服务。我们希望通过技术和数据赋能，为用户带来更好的体验，为音乐人和合作伙伴在音乐制作、发行、销售等方面提供支持。

基于公司丰富的音乐内容资产，需要将歌曲库、艺人资讯、专辑信息、厂牌信息等大量数据进行统一存储形成音乐内容数据仓库，并通过产品工具为业务人员提供数据分析服务。在内容数仓搭建的过程中，我们的工作始终围绕**降本增效**为主要目的进行优化与迭代，希望在**数据服务方面**不断提升产品工具的开发与分析效率，同时在**数仓架构方面**能够有效减少架构成本与资源开销。

在传统数据服务中，我们为业务分析师提供了多种数据服务，包括 SQL 查询、固定看板、定制化的分析工具以及人工跑数。然而，在实际应用过程中仍然存在一定痛点：

- **SQL 查询平台：**业务分析师根据需求进行 SQL 语句编写，对平台数据进行查询分析，每位业务人员都需要掌握 SQL，导致学习成本高、上手难度大。
- **固定看板（Dashboard）：**技术人员基于常规业务开发制作数据看板，虽然能够简化业务分析师查询的过程，但是看板制作成本高且灵活度低，当面对复杂的用户问题时，看板无法及时调整以满足需求变更。

- **定制分析工具：**基于特定的业务需求，技术人员需要定制化开发产品分析工具，整体开发成本过高，且单一的开发工具不具备通用性，随着工具数量增加，操作介面变得散乱，从而降低业务效率。

- **人工跑数：**当以上三个场景都无法满足业务需求时，业务分析师需要向技术人员提需求进行人工跑数，沟通成本过高、整体解决效率低下。

随着行业发展趋势，LLMs 大语言模型（LLMs – Large Language Models，以下统一简称为大模型）出现有效地解决了这些问题。当平台融入大模型后，平台用户输入的问题会进入大模型进行语义解析，自动转化为 SQL 语句触发 OLAP 引擎开启数据分析与查询。通过**平台智能问答交互**的方式，业务分析师不再需要依靠人工编写 SQL 提供查询分析结果，技术人员也不需要再制作过于固定或者过于定制化的产品工具。大模型 + OLAP 引擎结合的**全新数据服务模式**，不仅为平台用户提供了个性化、灵活表达、秒级回复的服务体验，还大幅降低了企业内部技术与业务学习成本，加速数据分析效率，实现多端入口统一、界面统一的平台构建。

本文将详细介绍腾讯音乐如何基于 Apache Doris 构建查询高效、实时写入且统一的 OLAP 分析引擎，使 OLAP 作为底层基建加强大模型与之连接转化的效率、结果输出的准确率，最终提供更智能化的问答交互服务，也希望通过这篇文章为有相关业务需求的公司提供不同视角和思路。

大模型 + OLAP：开启数据服务平台新模式

在大模型 + OLAP 架构方案中，目前经典方案如下图所示，大模型充当中间层将用户输入的自然语言转化为 SQL 执行语句，OLAP 作为底层存储和数据处理的引擎，负责接受和执行从大模型发送过来的 SQL 语句，对数据进行预聚合、多维分析等操作，满足大规模数据集的查询分析需求。

然而，这种架构在实际落地过程中也面临一定挑战，例如语义理解的准确性、查询效率的优化、私域知识的理解等方面，具体如下：

- 复杂数据口径不统一：大模型对于技术方面的词汇，如字段、行列、表等无法理解，相反对于业务方面的词汇，如公司收入情况、日活跃用户数量等能够提供有效翻译与转换。因此挑战之一是需要思考如何引导用户进入指标范围内提问，挑战之二是当用户存在对多种指标、多类指标查询时，需要考虑如何保持指标维度口径的统一、如何有效生成对应的指标计算公式。

- 模型处理效率较低：现阶段大模型虽然支持交互能力，但推理速度较慢，需要花费十秒级以上响应，用户每增加一个问题输入，就需要花费更多等待时间，使服务质量降低。同时大模型整体按照 Token 收费，使用量增加时也会导致平台成本升高。

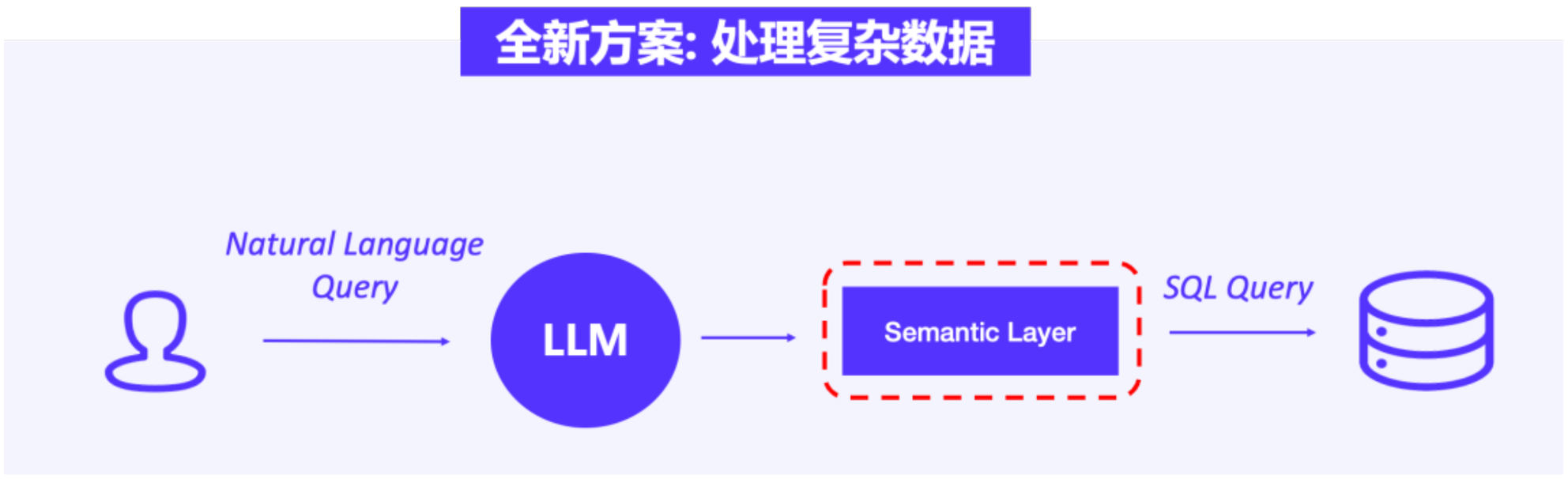
- 私域知识无法识别：虽然大模型已经开展许多公开数据集的语言转换训练，但面对企业内部的大量专业术语仍无法很好地理解转化。以音乐内容数据库为例，大模型时常缺少对于某些冷门歌曲的认知，在问答过程中无法正确给出交互反馈，因此我们需要增强大模型对于私域知识的理解。

- 定制场景无法满足：大模型主要依据自身数据集进行回答，会出现“知识幻觉”（输出缺乏依据的内容）问题，我们需要允许第三方插件的接入使大模型得以联网，让用户借助内部插件完成更定制化、更多样的任务。因此如何接入、匹配并触发组件功能是我们的重点优化目标。

面对经典方案中的落地难点，我们的总体解决思路是将以上四大挑战逐一拆解，通过组件叠加分

阶段完善大模型 + OLAP 架构构建，最终实现全新的交互问答服务模式，接下来我们将介绍各阶段挑战对应的解决方案。

增加语义层：处理复杂数据问题

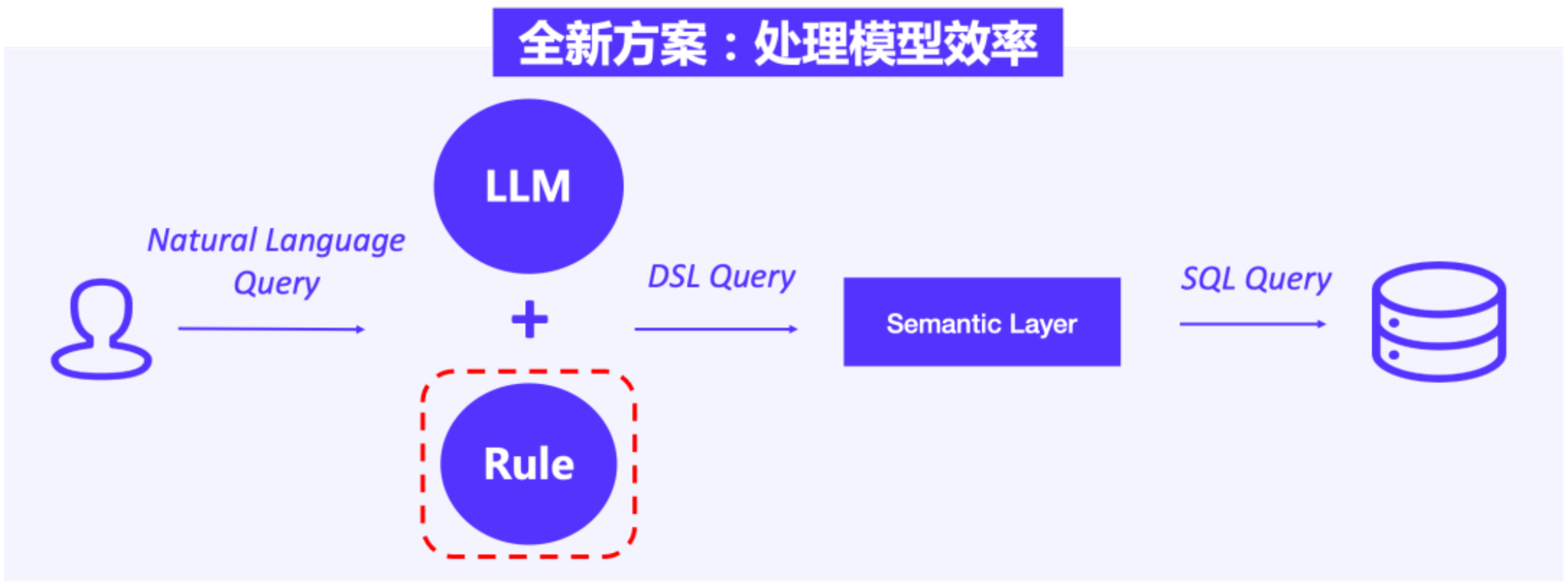


为了解决复杂数据处理问题，我们在大模型与 OLAP 中间增加 Semantic Layer（以下简称语义层）。

一方面语义层作为连接技术与业务之间的转换桥梁，能够将数据字段翻译为业务用户的术语，使业务知识作为额外的抽象层。通过语义层，业务分析师不需要在定义指标后存储于 OLAP 数仓中，能够直接在语义层中指定过滤条件，将所需指标筛选后生成 SQL 语句并在 OLAP 中进行字段查询。这意味着，业务分析师能够把多源数据按照需求定义成语义信息并形成语义标准，有效解决了多种指标、多类维度计算口径不统一的挑战。

另一方面语义层能够针对业务计算逻辑，进行语义加工、描述、关联和运算。语义层在过滤数据后，能够屏蔽由表关联所产生的复杂指标计算公式，将多表 Join 场景进行拆解、转化，形成较为简单的单表查询，以提升语义转化的准确性。

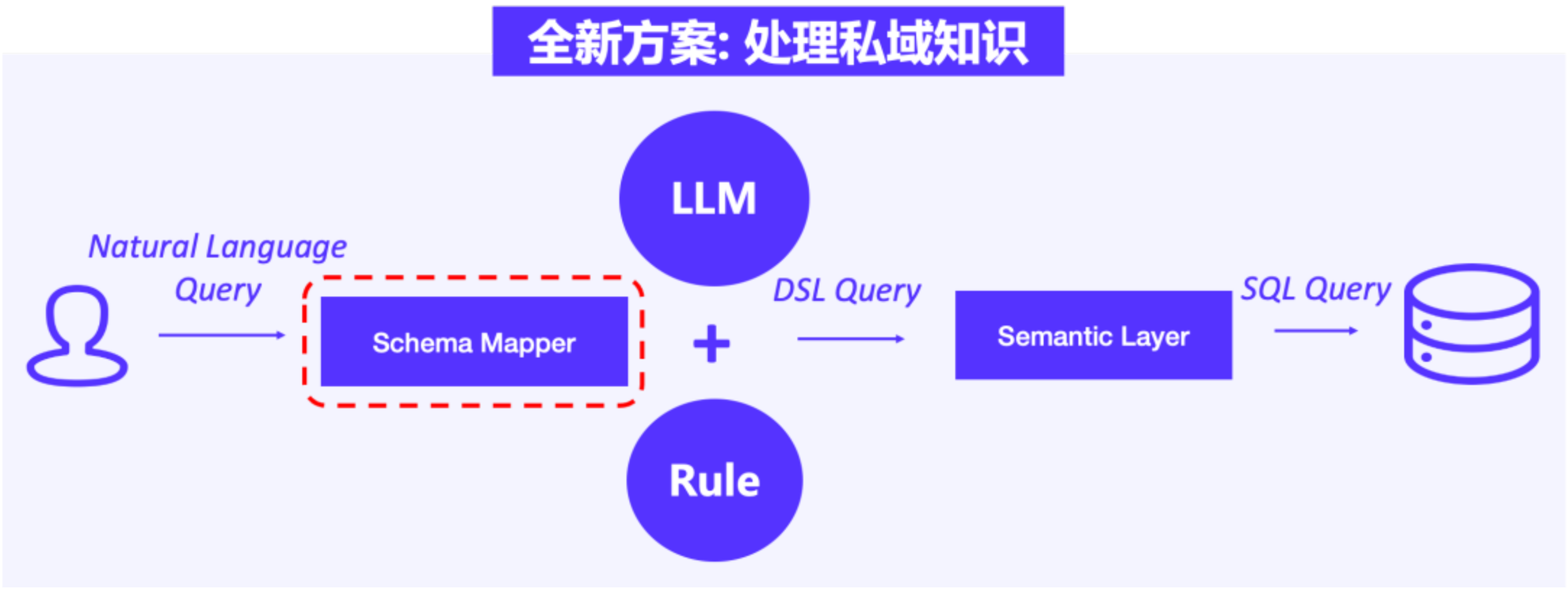
设定人工经验：处理模型效率问题



针对模型效率问题，我们的解决思路是对指标计算、明细查询、人群圈选等查询场景进行复杂度判定，将简单查询场景直接跳过大模型解析的步骤，进入底层 OLAP 进行处理分析，使大模型更加专注处理复杂查询场景。

为此，如上图所示我们在模型中添加人工经验判断。当业务分析师输入“查询各大音乐平台收入”问题时，模型依据判定规则发现该场景只需要提供某个指标或几个维度即可完成，这时不需要将问题进入大模型解析，直接使用 OLAP 进行查询分析，能够有效缩短响应时间，提升结果反馈效率。此外，跳过大模型解析的步骤也能节省 API 调用经费，解决平台使用成本升高问题。

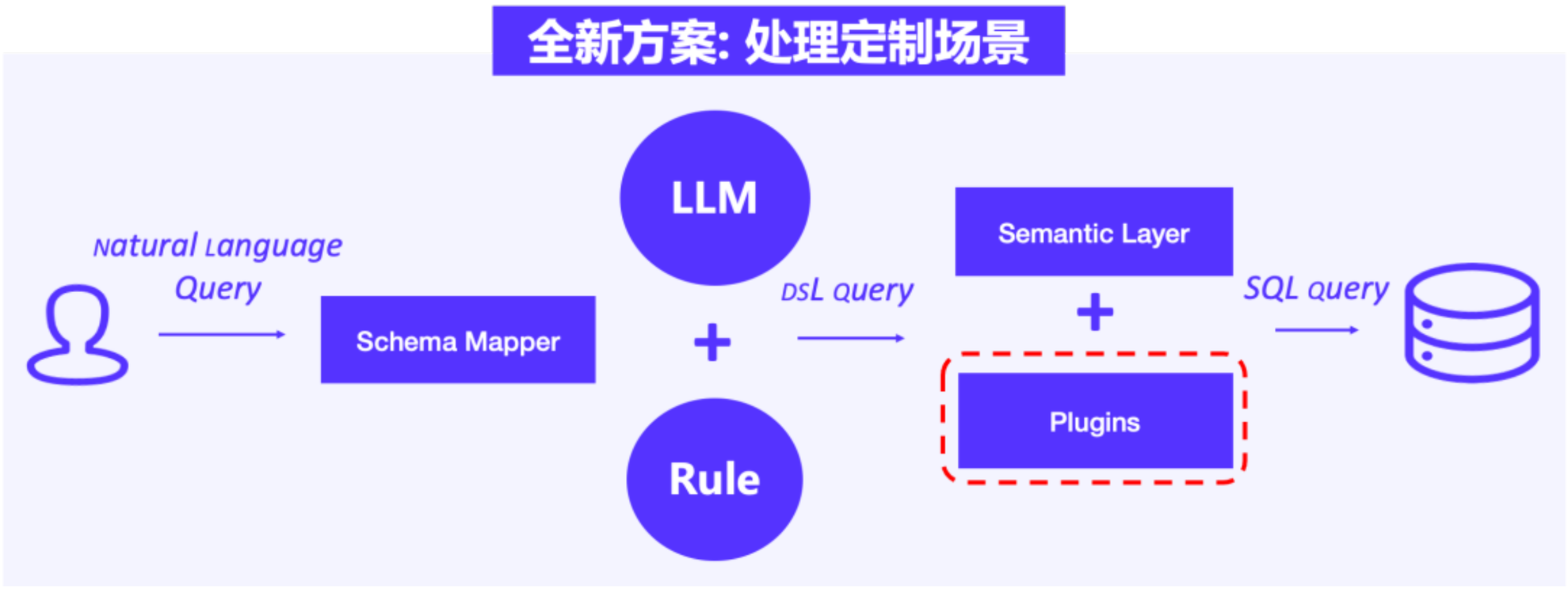
增加内容映射：处理私域知识问题



针对私域知识的问题，我们在大模型上游增加 Schema Mapper 、在外部建立业务知识库，将平台用户的问题与知识库进行连接，通过 Schema Mapper 判定是否存在部份文字能够与知识库内容匹配。如果匹配成功，大模型将进一步解析转化、OLAP 分析处理。Schema Mapper 与业务知识库的引入，有效解决了大模型对私域知识理解不足的问题，提升语言处理的效果。

目前，我们正在不断对 Schema Mapper 匹配准确性进行测试与优化，将知识库中的内容进行分类处理、字段评级等操作，同时将输入文本进行不同范围的内容映射（如全文本映射与模糊映射），通过映射结果来加强模型语义解析的能力。

插件接入：处理定制场景问题



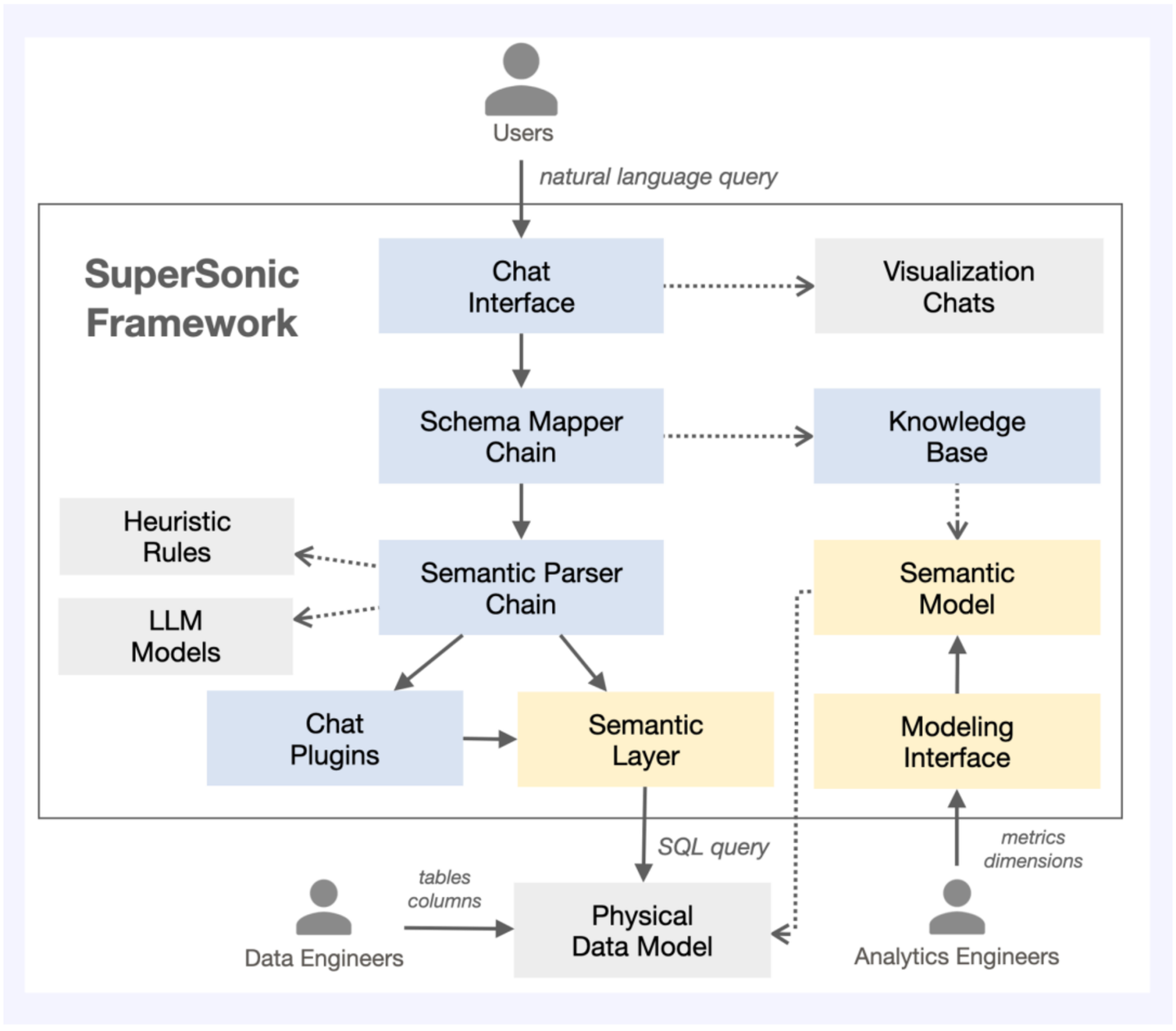
定制化场景主要指代业务范围之外的查询需求，需要将音乐内容数据与法律、政治、金融、监管等方面信息结合提供问答服务。通过增加插件，使平台用户能够访问实时更新且无法包含在训练数据或业务知识库中的信息，以实现定制化交互。

由于插件类型不同，模型接入方式也会有所不同，常见的接入方式主要分为两种：

- **Embedding 本地文本接入：**该方式首先对本地文档进行向量化处理，通过语义向量搜索，找到本地文档中相关或者相似的词语进行匹配，之后将文档内容注入大模型解析窗口中生成答案。这种方式非常适合业务分析师希望将音乐内容数据库与最新政策等一类较为私有的文件结合完成查询需求。
- **ChatGPT 第三方插件接入：**每款插件具备对应的 Prompt 与调用函数。业务人员在安装某款插件之后，在与模型对话中可以通过 Prompt 词触发函数开启调用。目前第三方插件类型丰富，涉及行业广泛，能够有效增加多元场景的处理与响应能力。

超音数平台框架构思

根据上述大模型 + OLAP 的四大解决方案进行了方案整合，以此进行框架设计并将其命名为超音数平台。大模型主要作用于自然语言与 SQL 分析语句的连接与转化，OLAP 引擎则作为数据存储与查询分析的核心基建。



超音数平台对于业务流程如图所示，模型运转具体过程如下：

- 用户输入问题通过 Schema Mapper 检索，判定字段是否匹配与业务知识库。
- 如若匹配则跳过大模型解析步骤，直接利用知识库中的指标计算公式触发 OLAP 进行查询分析；如若不匹配则进入大模型，开启下一步判定。
- 大模型首先通过人工经验判定问题复杂度，简单查询将指定 OLAP 引擎直接分析，复杂查询则开启语义解析形成 DSL 语句。
- DSL 语句通过语义层进一步过滤、拆解关联查询场景，生成简易单表 SQL 语句以触发 OLAP 数据处理与查询加速。
- 针对需要与外部信息结合的查询场景，大模型会判断是否调用第三方插件来辅助完成查询。

以“某首歌曲能否在综艺节目播出”为例，在经过检索匹配、语义解析后，大模型选择利用 OLAP 数据查询与第三方版权行业插件结合的方式进行回答，最终呈现结果由数仓中的歌曲信息与插件判定结果构成。

如今，业务分析师只需要在超音数平台中定义指标含义、维度类型即可直接开展自然语言的问答交互服务。同时还可以在平台中内置插件、丰富指标市场来拓展语义解析能力，完全覆盖了业务在常规与定制化场景下的查询需求。平台基于大模型 + OLAP 的模式加速业务分析效率，减少技术开发成本，向智能化、个性化、实时化的全新业务服务模式更近一步。

在这里希望可以与大家分享该开源项目，让更多人体验和学习大模型构建，也欢迎感兴趣的读者们共同参与大模型开发与建设。

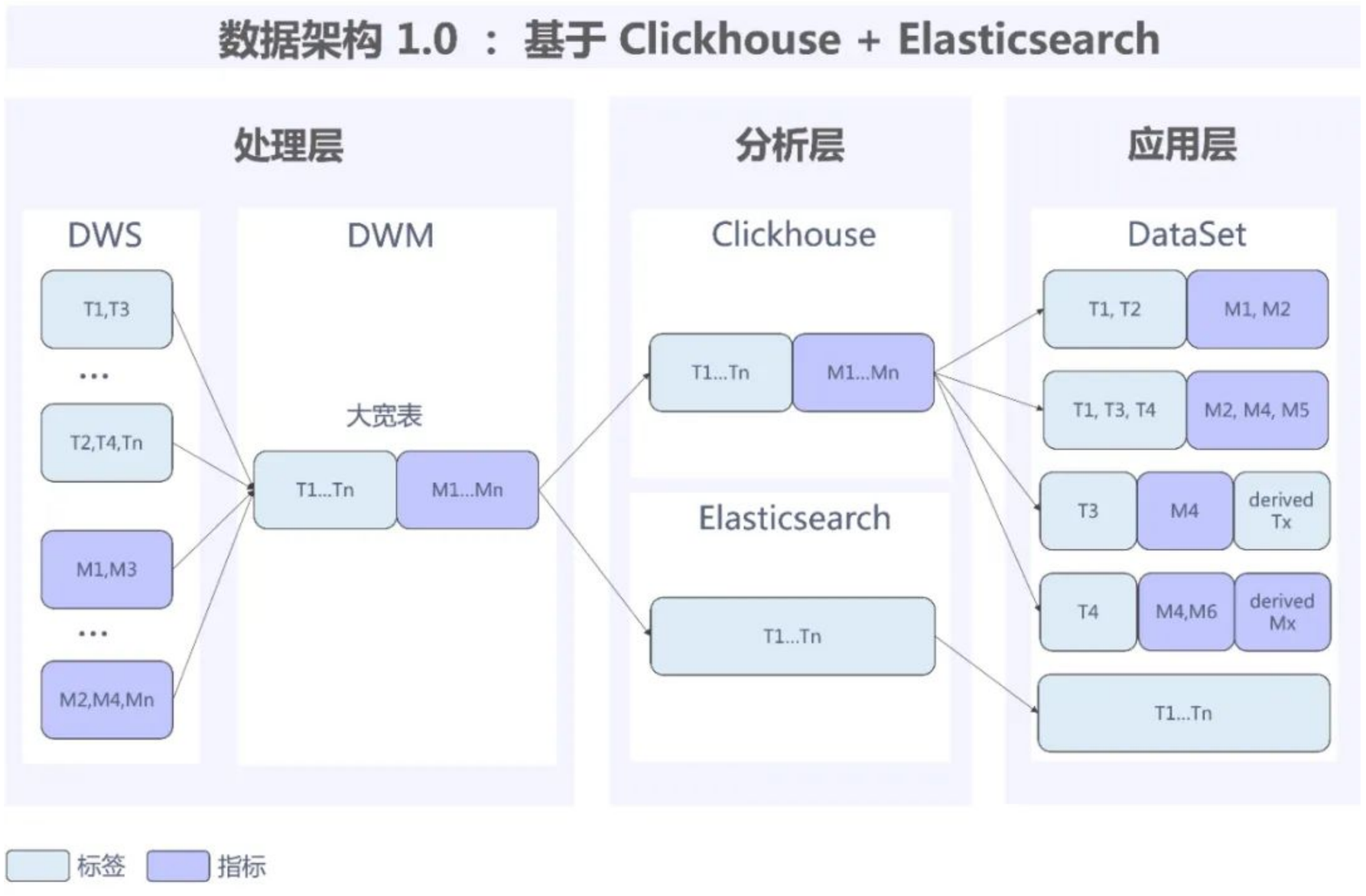
> 超音数开源框架：https://github.com/tencentmusic/supersonic

超音数平台框架演进

在平台构建的过程中，OLAP 引擎作为整体架构的基建对 SQL 语句处理、数据存储分析、上游应用层的查询响应等有着至关重要的作用，我们希望通过架构升级以加强大模型到 OLAP 引擎的转化效率与结果输出准确性。

接下来我们将对比介绍 OLAP 早期架构与新一代架构在数据写入与查询两方面的差异，分享在架构演进过程中大模型 + OLAP 模型优化历程，最终助力超音数平台的构建，开启新一代的数据服务模式。

数据架构 1.0



我们初期的业务架构如上图所示，分为处理层、分析层、应用层三部份，用户文本在进入大模型之后解析为 SQL 语句使 OLAP 开始执行任务，具体的工作原理如下：

- 处理层：在 ODS– DWD– DWS 三层中将数据整合为不同主题的标签和指标体系之后，通过对 DWS 调度与采集所需字段，在 DWM 层将维度与指标数据加工成大宽表。
- 分析层：通过大宽表进入分析层，将数据导入 Clickhouse 与 Elasticsearch，其中 Clickhosue 主要负责维度与指标两类数据的查询加速，作为分析引擎为后续提供报表开发服务；Elasticsearch 主要负责维度数据处理，作为搜索/圈选引擎。
- 应用层：业务人员基于场景选取所需要的标签与指标，在应用层中创建数据集作为逻辑视图，同时可以二次定义衍生的标签与指标。

在实际业务使用中，早期架构的数据处理方式存在大宽表带来的数据延迟与存储浪费、多套组件导致架构冗余带来指标维度重复定义、学习与运维成本高等问题，具体如下：

- **数据延迟：**处理层不支持部分列表更新，DWS 层数据写入产生延迟后会造成大宽表的延迟，进而导致数据时效性下降。
- **运维成本高：**在处理层大宽表中维度数据量平均占一张大宽表的 50%，且在大部份情况下变化缓慢，这意味着每一张宽表的开发会将维度数据叠加，造成存储资源浪费、维护成本增加

在分析层中存在多引擎使用问题，查询 SQL 语句需要同时适配 Clickhouse 与 Elasticsearch 两个组件，增加人力成本，且两套组件也会加大运维难度，运维成本进一步升高。

- **架构冗余：**在应用层进行指标与维度定义时，导致相同数据会进行多次定义使各种指标、维度定义口径不一致，造成权限不可控，例如上图所示的 T1 （标签）与 M1 （维度）在应用层中，被不同数据集多次定义。

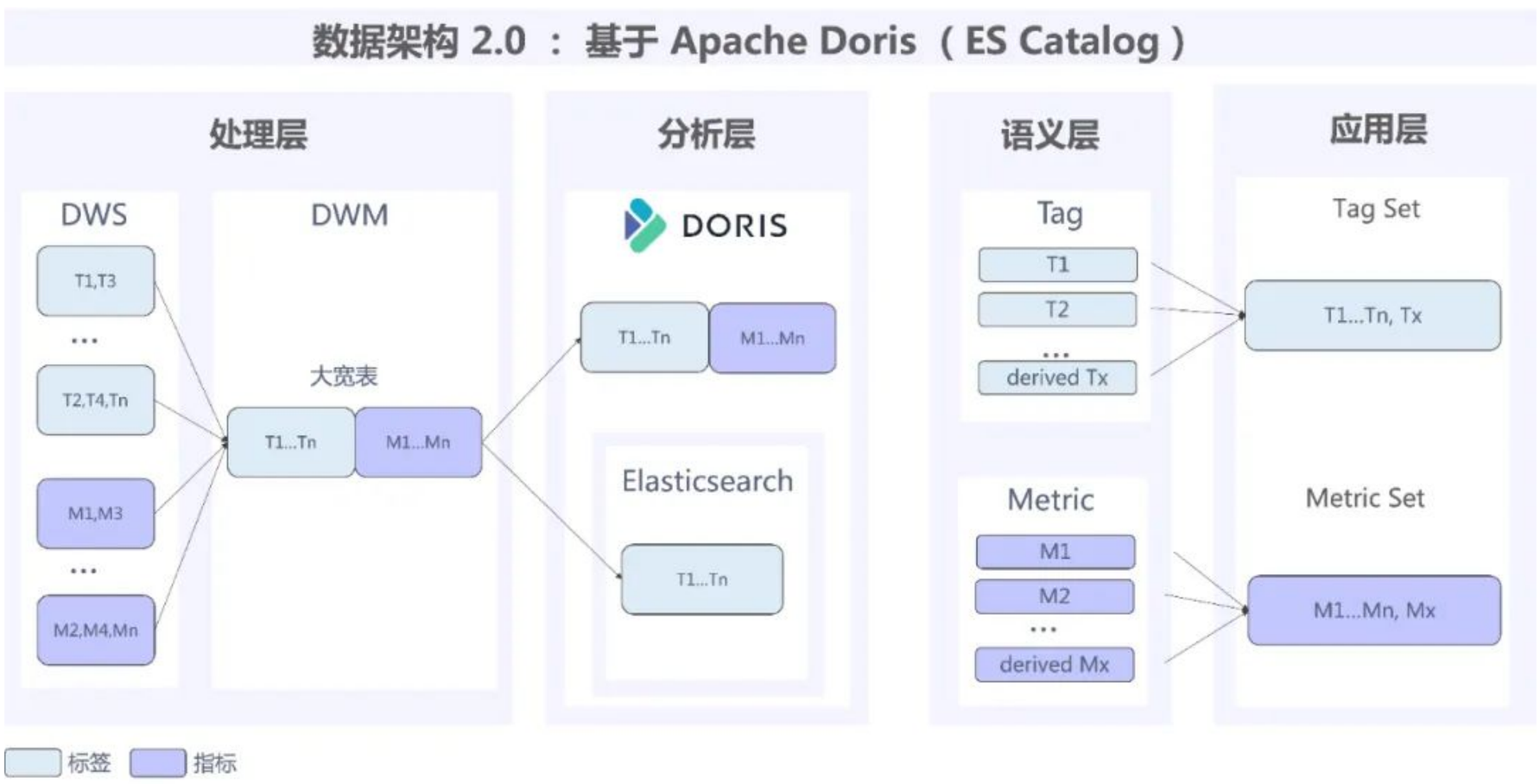
数据架构 2.0

基于以上问题，我们开始对架构进行改造升级，并在众多 OLAP 引擎中选择了 Apache Doris 来替换原有组件，主要因为 Apache Doris 具备以下核心优势：

- **实时导入：** Apache Doris 能够支持海量业务数据的高吞吐实时写入，时效性可以做到秒级完成导入。

- **引擎统一：**支持 Multi-Catalog 功能，能够通过 Elasticsearch Catalog 外表查询，实现查询出口统一，查询层架构实现链路极简，维护成本也大幅降低。

- **查询分析性能：**Apache Doris 是 MPP 架构，支持大表分布式 Join，其倒排索引、物化视图、行列混存等功能使查询分析性能更加高效极速。

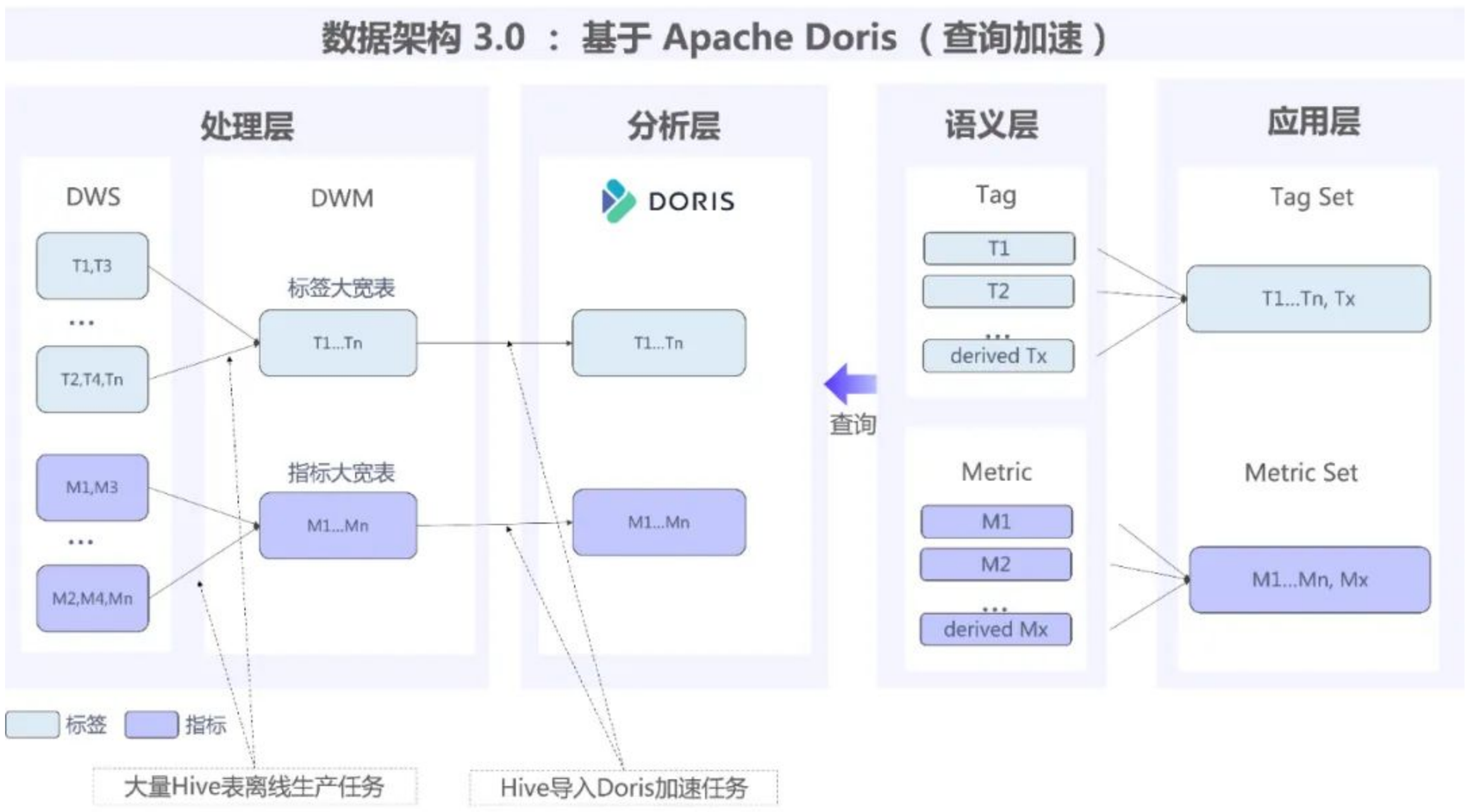


在数据架构 2.0 版本中，数据架构保留处理层部份，主要升级分析层架构，并进行语义层叠加：

- 分析层：引入 Apache Doris 替换 Clickhouse 组件，利用 Doris 的 Elasticsearch Catalog 功能对 Elasticsearch 外表进行查询，实现查询出口统一；
- 语义层：应用层不再需要创建数据集视图，直接通过语义层获取指标与标签内容执行查询任务，有效解决标签与指标口径问题。

数据架构 3.0

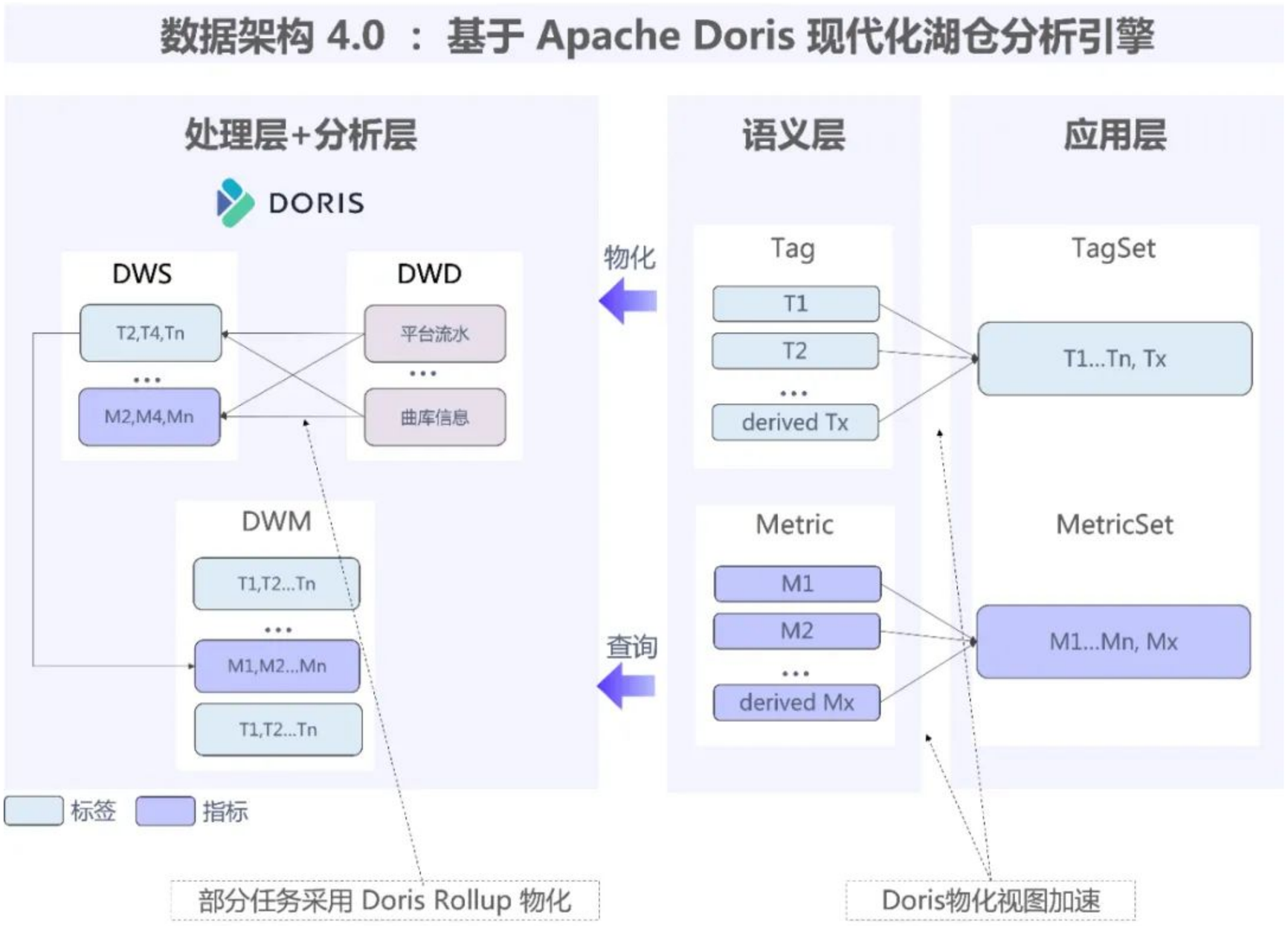
由于宽表开发过程中，维度数据一般变化较小、字符存储空间较大，且分析查询一般只需要查询最新的维度数据。在这种情况下，如果不断叠加维度数据制作宽表，会造成存储空间浪费的问题，同时查询响应速度也受到影响。



为了进一步提升架构性能，数据架构 3.0 主要将处理层中大宽表进行拆分，同时将分析层统一使用 Apache Doris 作为查询分析引擎：

- 处理层：按照业务分类在 DWM 中将大宽表拆分成缓慢维度表与指标表，使两类表在本地 Hive 中进行关联，通过 Hive 导入 Apache Doris 分析层中加速任务；
- 分析层：将关联数据表直接导入 Apache Doris 中，结合语义层暴露指标与维度以实现语义统一，用户只需要通过过滤条件就能够直接查询数据，得到所需要的结果。

数据架构 4.0



我们延续了 3.0 架构中分析层统一的优势，对处理层、分析层、语义层架构进一步优化，使查询性能显著提升：

- 分析层 + 处理层：数仓 DWD 层数据采用 Rollup 功能使事实表与维度表实时关联并创建多个视图进入 DWS 中。通过这种方式，分析层与处理层中的各类指标数据无需再重复定义，能够基于 Apache Doris 全部写入新建的 Rollup 视图中并利用 `GROUP BY` 将维度传入视图进行查询加速，直接对外暴露所需数据。

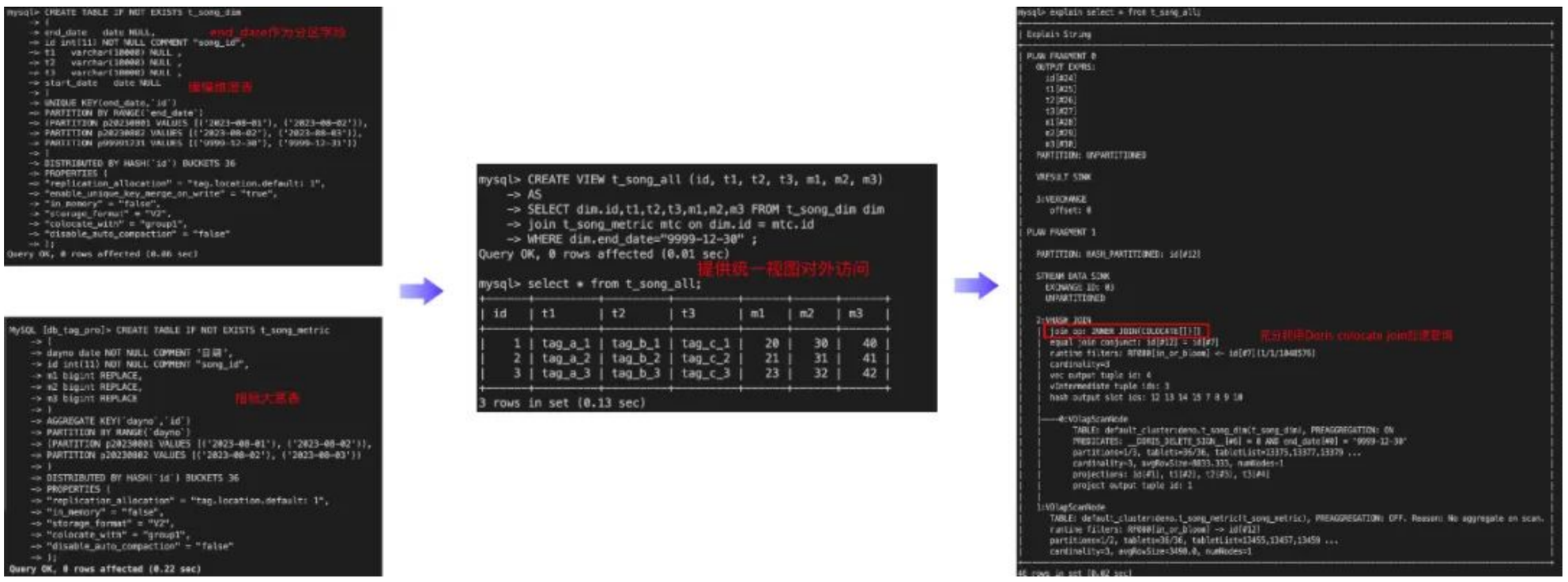
- 语义层：利用 Apache Doris 物化视图对指标与维度自定义口径，通过语义物化层进行查询加速，并将指标与维度通过 `SUM` 加工开发衍生标签与维度数据。

- 应用层：利用 Apache Doris 2.0 版本的倒排索引功能，对现有的索引结构进行丰富，满足了对知识库进行模糊查询、等值查询和范围查询等场景中的能力，进一步加速指标、维度查询响应速度。

数仓架构基于 Apache Doris 迭代升级，最终实现导入实时、引擎统一、查询高效的现代化湖仓 OLAP 引擎，简化架构链路的同时，有效解决大宽表中指标重复定义所带来的问题。在架构演进的过程，我们也积累许多关于 Apache Doris 性能优化经验，希望分享给读者们带来一些参考。

Apach Doris 性能优化实践

Colocate Join 宽表优化



在上文架构改造中我们提及，由于宽表开发会不断叠加字符数据，消耗存储空间，降低查询性能，因此我们充分利用了 Colocate Join 功能对宽表拆分、本地关联查询加速进行优化，具体过程如下：

- 指标大宽表：采用 Apache Doris 的 Aggregate Key 模型，使用增量的方式将数据覆盖写入；
- 缓慢维度表：主要通过 `start_date` 和 `end_date` 的设置进行表建设，同时利用 `end_date` 进行分区，当我们需要查询最新的维度数据时只需要将 `end_date` 设置为 `'9999-12-31'` 即可。此外我们引用 Doris 2.0 版本中的写时合并，利用 Unique Key 模型进行维度数据聚合，使查询性能在该场景中得到很大的提升。
- 对外访问视图：在指标与维度表建设完成之后，利用 `CREAT VIEW` 提供统一对外访问视图，同时添加 `end_date` 条件，使视图保持最新数据的展示。通过这样的方式不仅能够大幅度降低查询的复杂性，还能够充分利用 Doris 特性实现查询加速。

Rollup 解决指标膨胀问题

宽表拆分为指标表与维度表后，我们发现每一次视图产生都需要定义多个指标，出现指标膨胀的情况。以“歌曲播放量结算”为例，当仅定义单一指标时，我们需要将各个平台 + 各类内容进行排列组合，使语义层定义很多指标数据，造成指标数量过多。此外这些指标都需要通过离线生产任务进行加工，并通过 Hive 导入至 Apache Doris 中，造成链路较长、加工维护比较困难。

> 平台指标：覆盖四大音乐平台，包括酷我、QQ 音乐、酷狗、K 歌
> 内容指标：包含歌曲、歌手、专辑以及厂牌等数据

date_no	song_id	singer_id	plat	play_count
20230801	1	1	qy	100
20230801	2	1	qy	120
20230801	3	1	qy	130
20230802	1	1	qy	140
20230802	2	1	qy	150

```
mysql> CREATE TABLE IF NOT EXISTS t_song
    -> (
    ->     dayno DATE NOT NULL,
    ->     song_id int NOT null,
    ->     singer_id VARCHAR(16) null,
    ->     plat VARCHAR(32) null,
    ->     play_count bigint sum null,
    -> )
    -> AGGREGATE KEY(`dayno`, `song_id`, `singer_id` ,`plat`)
    -> DISTRIBUTED BY HASH(`song_id`) BUCKETS 1
    -> PROPERTIES (
    -> "replication_allocation" = "tag.location.default: 1"
    -> );
Query OK, 0 rows affected (0.01 sec)

mysql> ALTER TABLE t_song ADD ROLLUP r_singer (dayno, singer_id, plat,play_count);
Query OK, 0 rows affected (0.01 sec)
```

```
mysql>
mysql> explain select dayno, singer_id,sum(play_count) from t_song group by dayno,singer_id,plat;
+------------------------------------------------------------------------------------------------+
| Explain String                                                                                 |
+------------------------------------------------------------------------------------------------+
| PLAN FRAGMENT 0                                                                                |
|   OUTPUT EXPRS:                                                                                |
|     dayno[#17]                                                                                 |
|     singer_id[#18]                                                                             |
|     sum(play_count)[#19]                                                                       |
|   PARTITION: UNPARTITIONED                                                                     |
|                                                                                                |
|   VRESULT SINK                                                                                 |
|                                                                                                |
|   4:VEXCHANGE                                                                                  |
|      offset: 0                                                                                 |
|                                                                                                |
| PLAN FRAGMENT 1                                                                                |
|                                                                                                |
|   PARTITION: HASH_PARTITIONED: mv_dayno[#9], mv_singer_id[#10], mv_plat[#11]                   |
|                                                                                                |
|   STREAM DATA SINK                                                                             |
|     EXCHANGE ID: 04                                                                            |
|     UNPARTITIONED                                                                              |
|                                                                                                |
|   3:VAGGREGATE (merge finalize)                                                                |
|   |  output: sum(partial_sum(mva_SUM__play_count)[#12])[#16]                                   |
|   |  group by: mv_dayno[#9], mv_singer_id[#10], mv_plat[#11]                                   |
|   |  cardinality=0                                                                             |
|   |  projections: mv_dayno[#13], mv_singer_id[#14], sum(play_count)[#16]                       |
|   |  project output tuple id: 4                                                                |
|   |                                                                                            |
|   2:VEXCHANGE                                                                                  |
|      offset: 0                                                                                 |
|                                                                                                |
| PLAN FRAGMENT 2                                                                                |
|                                                                                                |
|   PARTITION: HASH_PARTITIONED: song_id[#5]                                                     |
|                                                                                                |
|   STREAM DATA SINK                                                                             |
|     EXCHANGE ID: 02                                                                            |
|     HASH_PARTITIONED: mv_dayno[#9], mv_singer_id[#10], mv_plat[#11]                            |
|                                                                                                |
|   1:VAGGREGATE (update serialize)                                                              |
|   |  STREAMING                                                                                 |
|   |  output: partial_sum(mva_SUM__play_count[#3])[#12]                                         |
|   |  group by: mv_dayno[#0], mv_singer_id[#1], mv_plat[#2]                                     |
|   |  cardinality=0                                                                             |
|   |                                                                                            |
|   0:VOlapScanNode                                                                              |
|      TABLE: default_cluster:demo.t_song(r_singer), PREAGGREGATION: ON                           |
|      partitions=0/1, tablets=0/0, tabletList=                                                  |
|      cardinality=1, avgRowSize=0.0, numNodes=1                                                 |
+------------------------------------------------------------------------------------------------+
48 rows in set (0.02 sec)
```

为了有效解决指标膨胀问题，我们引入了 Doris Rollup 功能。如图所示，在 Doris Base 表数据基础之上，可以根据指定维度来创建任意多个 Rollup 视图并自动进行 GROUP BY ，实现各个平台与各类内容指标定义不重复、查询性能提升的目标。

物化视图实现查询加速

```
mysql> create materialized view v_content_db_2 as select dayno,id, sum(m1+m2+m3) as m4 from t_content group by dayno,id;
Query OK, 0 rows affected (0.01 sec)
```
衍生指标

```
mysql> create table t_content_db (
    ->     dayno DATE NOT NULL,
    ->     id int NOT null,
    ->     t1 VARCHAR(16) null,
    ->     t2 VARCHAR(32) null,
    ->     m1 int  null,
    ->     m2 int  null,
    ->     m3 bigint  null,
    -> )
    -> duplicate key (`dayno`, `id`, `t1` ,`t2`)
    -> distributed BY hash(id) buckets 3
    -> properties("replication_num" = "1");
Query OK, 0 rows affected (0.01 sec)

mysql> create materialized view v_content_db as select dayno,t1,t2,m1 from t_content_db
    -> where (t1='a' or t2='b') and (t1='c' or t2='d');
Query OK, 0 rows affected (0.01 sec)
```
衍生维度

```
mysql> explain select dayno,t1,t2,m1 from t_content_db where (t1='a' or t2='b') and  (t1='c' or t2='d');
+------------------------------------------------------------------------------------------------+
| Explain String                                                                                 |
+------------------------------------------------------------------------------------------------+
| PLAN FRAGMENT 0                                                                                |
|   OUTPUT EXPRS:                                                                                |
|     mv_dayno[#0]                                                                               |
|     mv_t1[#1]                                                                                  |
|     mv_t2[#2]                                                                                  |
|     mv_m1[#3]                                                                                  |
|   PARTITION: UNPARTITIONED                                                                     |
|                                                                                                |
|   VRESULT SINK                                                                                 |
|                                                                                                |
|   1:VEXCHANGE                                                                                  |
|      offset: 0                                                                                 |
|                                                                                                |
| PLAN FRAGMENT 1                                                                                |
|                                                                                                |
|   PARTITION: HASH_PARTITIONED: id[#5]                                                          |
|                                                                                                |
|   STREAM DATA SINK                                                                             |
|     EXCHANGE ID: 01                                                                            |
|     UNPARTITIONED                                                                              |
|                                                                                                |
|   0:VOlapScanNode                                                                              |
|      TABLE: default_cluster:demo.t_content_db(v_content_db), PREAGGREGATION: ON                |
|      PREDICATES: mv_t1[#1] = 'a' OR mv_t2[#2] = 'b' AND mv_t1[#1] = 'c' OR mv_t2[#2] = 'd'     |
|      partitions=0/1, tablets=0/0, tabletList=                                                  |
|      cardinality=1, avgRowSize=0.0, numNodes=1                                                 |
+------------------------------------------------------------------------------------------------+
```

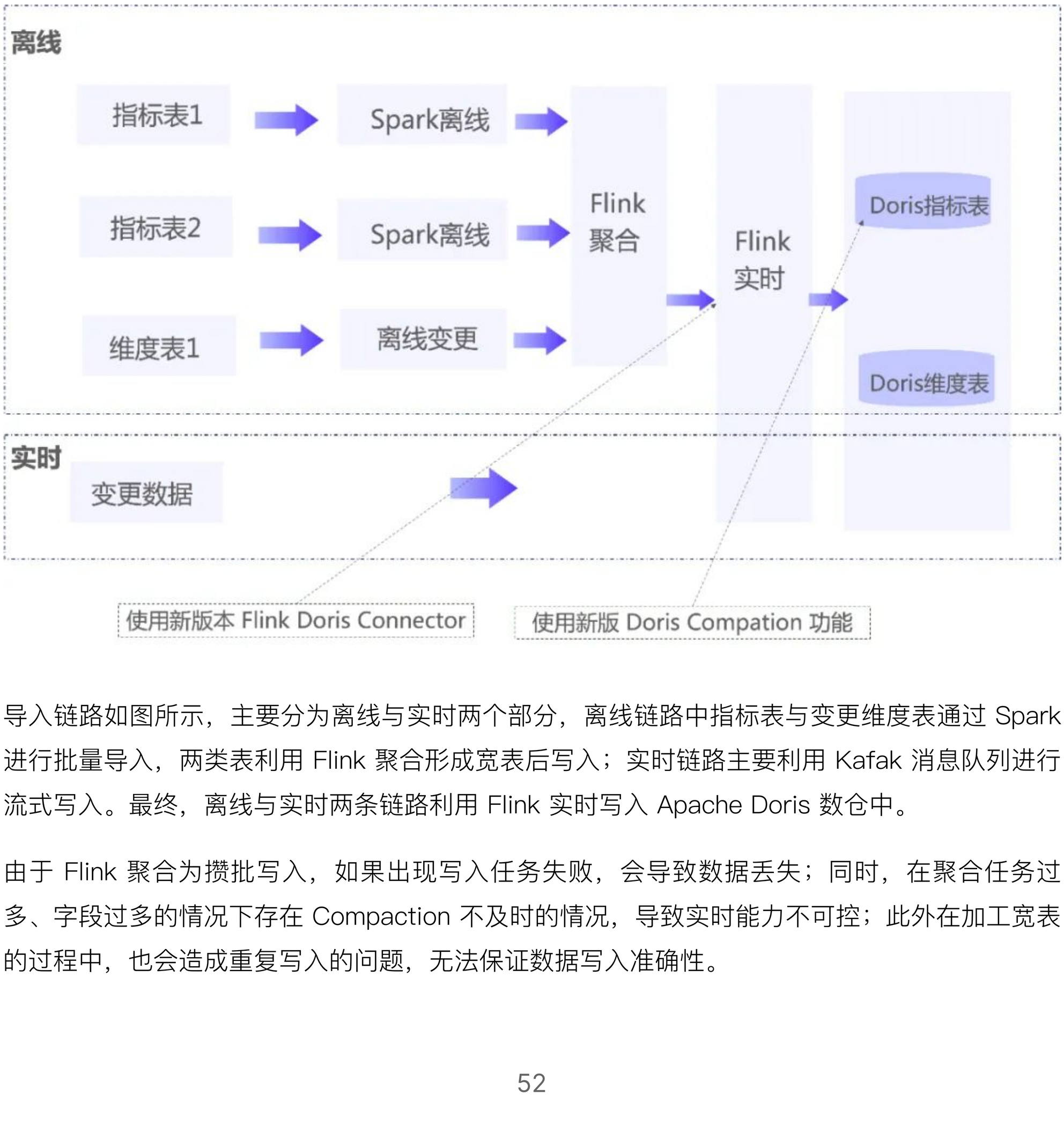
```
mysql> explain select dayno,id, sum(m1+m2+m3) as m4 from t_content group by dayno,id;
+------------------------------------------------------------------------------------------------+
| Explain String                                                                                 |
+------------------------------------------------------------------------------------------------+
| PLAN FRAGMENT 0                                                                                |
|   OUTPUT EXPRS:                                                                                |
|     dayno[#13]                                                                                 |
|     id[#14]                                                                                    |
|     m4[#15]                                                                                    |
|   PARTITION: UNPARTITIONED                                                                     |
|                                                                                                |
|   VRESULT SINK                                                                                 |
|                                                                                                |
|   2:VEXCHANGE                                                                                  |
|      offset: 0                                                                                 |
|                                                                                                |
| PLAN FRAGMENT 1                                                                                |
|                                                                                                |
|   PARTITION: HASH_PARTITIONED: mv_id[#1]                                                       |
|                                                                                                |
|   STREAM DATA SINK                                                                             |
|     EXCHANGE ID: 02                                                                            |
|     UNPARTITIONED                                                                              |
|                                                                                                |
|   1:VAGGREGATE (update finalize)                                                               |
|   |  output: sum(mva_SUM__((m1 + m2) + m3)[#2])[#12]                                           |
|   |  group by: mv_dayno[#0], mv_id[#1]                                                         |
|   |  cardinality=0                                                                             |
|   |  projections: mv_dayno[#10], mv_id[#11], m4[#12]                                           |
|   |  project output tuple id: 3                                                                |
|   |                                                                                            |
|   0:VOlapScanNode                                                                              |
|      TABLE: default_cluster:demo.t_content(v_content_db_2), PREAGGREGATION: ON                 |
|      partitions=0/1, tablets=0/0, tabletList=                                                  |
|      cardinality=1, avgRowSize=0.0, numNodes=1                                                 |
+------------------------------------------------------------------------------------------------+
```

除了减少指标数量外，我们还希望能够衍生指标并且做到查询加速。在 Apache Doris 2.0 版本中我们采用了物化视图功能进行衍生指标的开发。目前，我们主要在单一维度表中单独地去查询自定义标签与维度，在定义复杂口径后自动的通过语义层物化任务。

如上图所示我们将指标 M1 、M2、M3 与维度 T1、T2、T3 分别进行定义，并通过 SUM 加工衍生标签，在加工完成之后创建物化视图加速查询。此外，在 Doris 后续 2.1 版本中还会支持多表创建物化视图，我们也非常期待使用该功能。

Apach Doris 导入性能调优实践

目前，腾讯音乐具有 90+ 数据来源表、 3000 + 维度和指标、导入数据量达到千亿级别，我们希望数仓能够支持大规模数据快速导入，且导入过程中保证数据写入的准确性。



导入链路如图所示，主要分为离线与实时两个部分，离线链路中指标表与变更维度表通过 Spark 进行批量导入，两类表利用 Flink 聚合形成宽表后写入；实时链路主要利用 Kafak 消息队列进行流式写入。最终，离线与实时两条链路利用 Flink 实时写入 Apache Doris 数仓中。

由于 Flink 聚合为攒批写入，如果出现写入任务失败，会导致数据丢失；同时，在聚合任务过多、字段过多的情况下存在 Compaction 不及时的情况，导致实时能力不可控；此外在加工宽表的过程中，也会造成重复写入的问题，无法保证数据写入准确性。

在 Apache Doris 2.0 版本发布后，我们引入了其全新功能 Flink Doris Connector 与 Doris Compaction，有效解决了 Flink 聚合引起的问题。

Flink Doris Connector 实现快写入

Flink Doris Connector 主要是依赖 Checkpoint 机制进行流式写入，同时该功能默认开启两阶段提交，保证写入过程中 Exactly Once 语义。值得注意的是，我们在引入最新版的 Flink Doris Connector 功能后，实现了从关系型数据库到 Apache Doris 的一键整库同步，承载了我们实际业务中千亿级别的实时并行写入，满足数据快写入与不丢不重的需求。

Doris Compaction 保证写入稳定性

为了解决 Flink 聚合引起的偶发性 Compaction 不及时问题，我们引入最新版的 Vertical Compaction 与 Segment Compaction 功能。

- **Vertical Compaction 功能优势：**在单次合并过程中，我们不需要再将所有的列读出，只需要加载部份列数据即可，这能极大减少合并过程中的内存占用问题，提高压缩的执行速度，实现在大宽表场景下的部份数据合并。
- **Segment Compaction 功能优势：**在单批次大数据量的导入场景下可以有效减少 Flink 写入过程中产生的 Segment 数量，且能够使合并和导入两个过程并行，避免增加导入时间。

如上图所示在引入 Doris Compation 功能后，在写入量增加 50 % 的情况下，**Compaction Score 从平均 650 分降低至 80 分，技术人员不再需要担心夜间出现告警的情况，保证了整体链路的稳定性。**

总结收益与展望

在引入 Apache Doris 后，数据架构围绕降本增效的目标，不仅在写查方面的性能得到大幅度提升，并且有效减少架构成本与资源开销，具体的收益如下：

- **极速查询分析：**通过 Apache Doris 的 Rollup、物化视图、倒排索引功能，由原来的分钟级查询时间达到现如今秒级毫秒级；
- **导入性能提升：**导入优化完成后，原本 3000+ 维度、指标数据的导入时间需要超过一天，现如今能够在 8 小时内完成导入，导入时间缩短至原来的 1/3，实现快速导入需求；更重要的是，Apache Doris 在保证数据快写入的同时，使数据能够不丢不重、准确写入；
- **链路极简与统一：**Apache Doris 将查询与分析出口引擎统一，去除 Elasticsearch 集群使架构链路极简；
- **存储成本降低：**通过大宽表拆分的方式，使存储成本降低 30%，开发成本降低 40% 。

在未来，我们将进一步拓展使用 Apache Doris 湖仓一体功能，对 Hive、MySQL、数据湖等多源异构数据库进行网关统一，实现真正意义上的实时统一分析引擎。同时，尝试 CCR 跨集群数据同步功能，通过用户多集群的数据库表自动同步以提升在线服务数据的可用性。未来，我们也将在测试环节中验证读写负载分离以及多机房备份的性能效果。

目前，Apache Doris 社区已经公布了后续版本中将推出的存算分离全新架构，能够利用低成本的共享存储系统简化上层计算节点的复杂度，使架构带来巨大的成本经济优势。我们也希望能够进一步探索，基于 Apache Doris 本地高速缓存 + 共享存储系统的混合模式，在保障性能的同时降低系统存储开销。

最后，非常感谢 SelectDB 技术团队的积极响应与专业解答，希望通过这篇文章分享大语言模型在互联网业务中的应用，也欢迎更多人参与 Apache Doris 社区与超音数平台的开源框架构建。最后，我们也会持续参与社区活动，将相关成果贡献回馈社区，希望 Apache Doris 飞速发展，越来越好！

Apache Doris

助力中国联通万亿日志数据分析提速 10 倍

作者：联通西部创新研究院 大数据开发工程师，刘宇麒、贾涛

导读：在数据安全管理体系的背后，离不开对安全日志数据的存储与分析。以终端设备为例，中国联通每天会产生百亿级别的日志数据，对于保障网络安全、提高系统稳定性和可靠性具有至关重要的作用。目前，Apache Doris 在联通体系的落地已支持了 30 多条业务线和数百个实时作业，不仅帮助联通实现了万亿级安全日志的高效分析和低成本，也为其他运营商提供了成功的参考案例和学习经验，对推动运营商的数字化转型进程具有重要意义。

联通西部创新研究院是中国联通在西部地区布局的重要载体，也是中国联通数字化创新能力体系的重要组成部分，承载了集团公司科技创新体系和数字化创新体系的需求。依托联通数科的优质资源及能力底座，在云计算、大数据、物联网、人工智能、网络安全等业务领域具备深厚的技术能力和丰富的项目经验。

近些年来，网络高危漏洞数量的增长、DDoS 攻击比例的提升、恶意 Bot 流量的持续上升使得 Web 安全威胁态势愈发严峻，而数字化转型进程的推进在丰富业务创新的同时、也提升了网络空间复杂度、进一步加剧了网络安全风险。这样的背景之下，联通以攻防实战对抗为目标、进行国家级网络空间的安全治理工作，围绕“云–管–端–数”构建 了多级综合防控体系，聚焦于实时监测、攻击溯源、通报预警、应急处置、情报共享等工作，构建数据全生命周期安全管理体系，为客户提供从顶层设计到运营维护一站式服务。

在数据安全管理体系的背后，离不开对安全日志数据的存储与分析。以终端设备为例，每天会产生海量的设备日志，这些日志数据记录着各种网络时间和系统操作的细节信息，对于保障网络安全、提高系统稳定性和可靠性具有至关重要的作用。为了更好的管理和分析安全日志数据，联通西部创新研究院应集团要求构建一个集中化日志数据分析平台，满足对事件和日志数据自动化采集、存储、管理、分析和可视化的诉求。这要求集中化数据分析平台具备以下能力：

- 建模分析：基于网络日志数据和告警数据进行规则或智能挖掘，发现潜在的安全事件，例如钓鱼邮件、非法访问等，并进行定向威胁感知。
- 态势大屏：通过多种维度不同监控指标的组合，例如安全事件 TOP5 等，密切监控当前网络安全态势状况，通过态势大屏呈现攻击威胁的主要分布。
- 追踪溯源：通过对安全事件的快速研判，还原整个攻击链条进行精准的溯源取证，从而保障网络和数据安全。

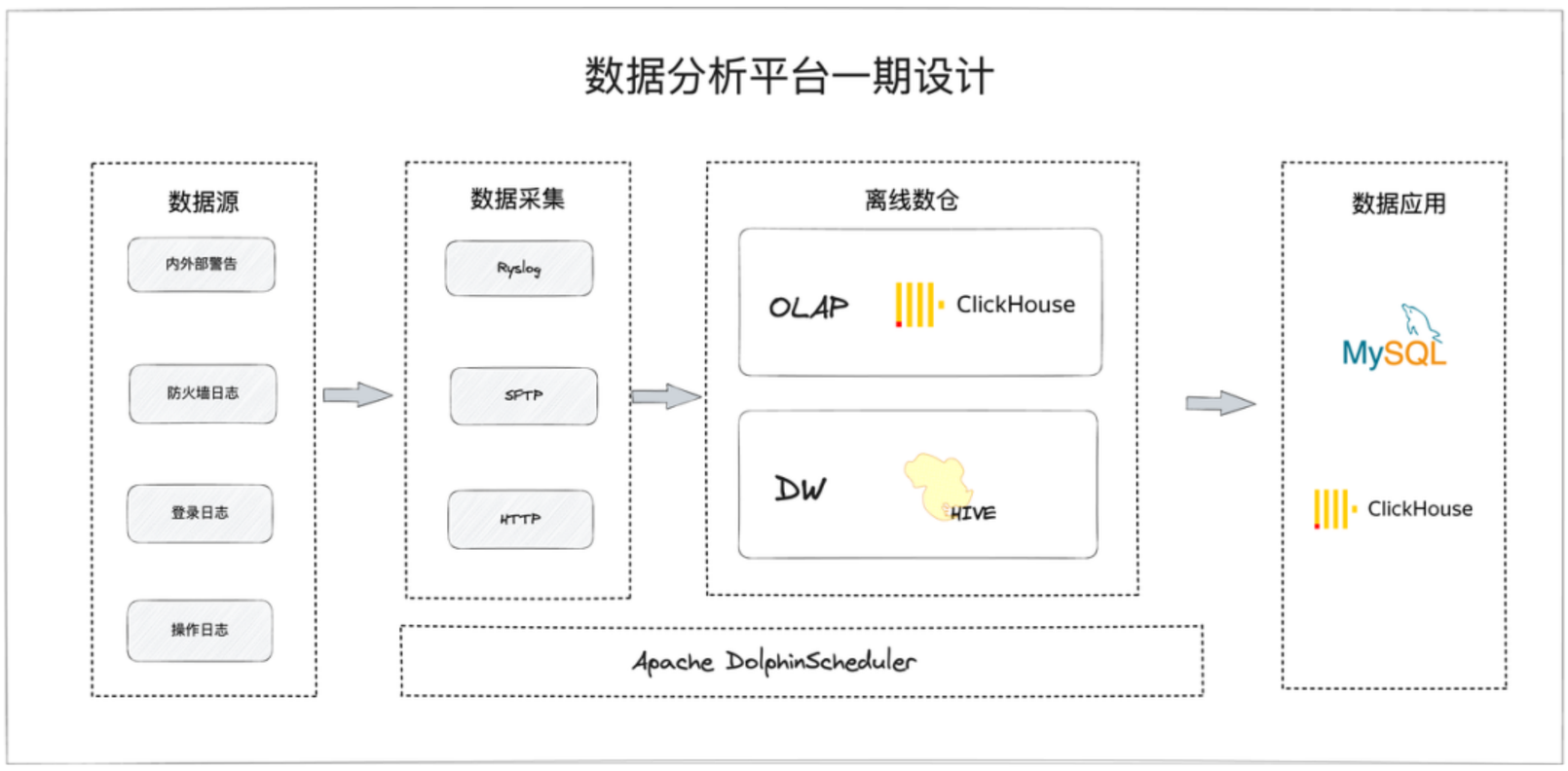
为搭建具备上述能力的集中化日志数据分析平台，在正式搭建之前，结合日志数据的特性及业务要求，我们需要综合考虑考虑如何满足以下要求，以确保平台能高效的支持联通日志分析场景的实际应用：

- 数据接入方面：日志数据具有种类繁多、格式多样化、规模庞大等特点，要求数据平台支持多种日志格式数据的导入，并支持高性能的数据写入。
- 实时性要求方面：为及时监控和了解系统运营情况和存在的问题，高实时性对于数据平台非常关键。这要求平台要实时进行数据同步，保障数据的一致性，并支持数据实时查询，以便获取最新的系统和业务状态。
- 可扩展要求方面：数据平台需要具备计算与存储的拓展能力，以便满足集团及分公司不断增长的数据处理分析需求。

在中国联通安全日志数据分析平台的迭代过程中，经历了从基于 Hive 的离线数据仓库到以 Apache Doris 为核心的实时数据仓库。从**具体业务收益来讲**，Apahce Doris 的引入支持了联通 30+ 条业务线和数百个实时作业，为联通带来了存储资源节约 50%、百亿级别数据查询秒级响应、数据导入效率提升 60% 的显著成果，成功实现了降本增效的业务目标；**从集团整体价值来讲**，通过该平台，联通可以更好地监控运营状态、保障网络安全，为运营商安全管理体系提供了重要的底层支持。总而言之， Apache Doris 在联通体系的落地，不仅帮助联通实现了万亿级安全日志的高效分析和低成本，也为其他运营商提供了成功的参考案例和学习经验，对运营商数字化转型进程的推进有着重要作用。

基于 Hive 的离线数据仓库

在项目一期建设中，我们以 Apache Hive 为核心建立了离线数仓，并在其此础上进行了数据仓库分层。当原始数据经过数据采集进入离线数仓后，由 Spark 逐层进行处理，并配合 Apache DolphinScheduler 以分钟级调度执行计算作业，最终将数据输出至 OLAP 和应用数据库。



从业务的角度来看，**该架构数据流的痛点问题在于数据实时性不足，主要受限于 Hive 的离线批处理模式，端到端的延迟最短竟然需要 10 分钟。**

其次，我们在该架构中选择了 ClickHouse 作为 OLAP 引擎，但在实际使用场景中发现 ClickHouse 存在以下不足：

- ClickHouse 并发支持能力不足，无法满足业务需求，例如实时大屏指标的计算与加载缓慢，经常会在业务高峰期出现查询超时。
- 业务中有大量安全事件表需要进行多表 Join，这些表数据量较大，而 Clickhouse 在分布式 Join 实现性能较低，时常会出现 OOM 问题，为避免该情况发生，常常需要依赖宽表才能缓解，而这既影响了业务的稳定性，也增加了许多额外的维护成本。
- 由于 ClickHouse 对于数据更新操作支持较弱、更新性能较差，这也限制了它在某些场景下的应用。
- ClickHouse 使用和运维成本较高，也给我们带来了更高的人工投入成本。

系统选型及落地

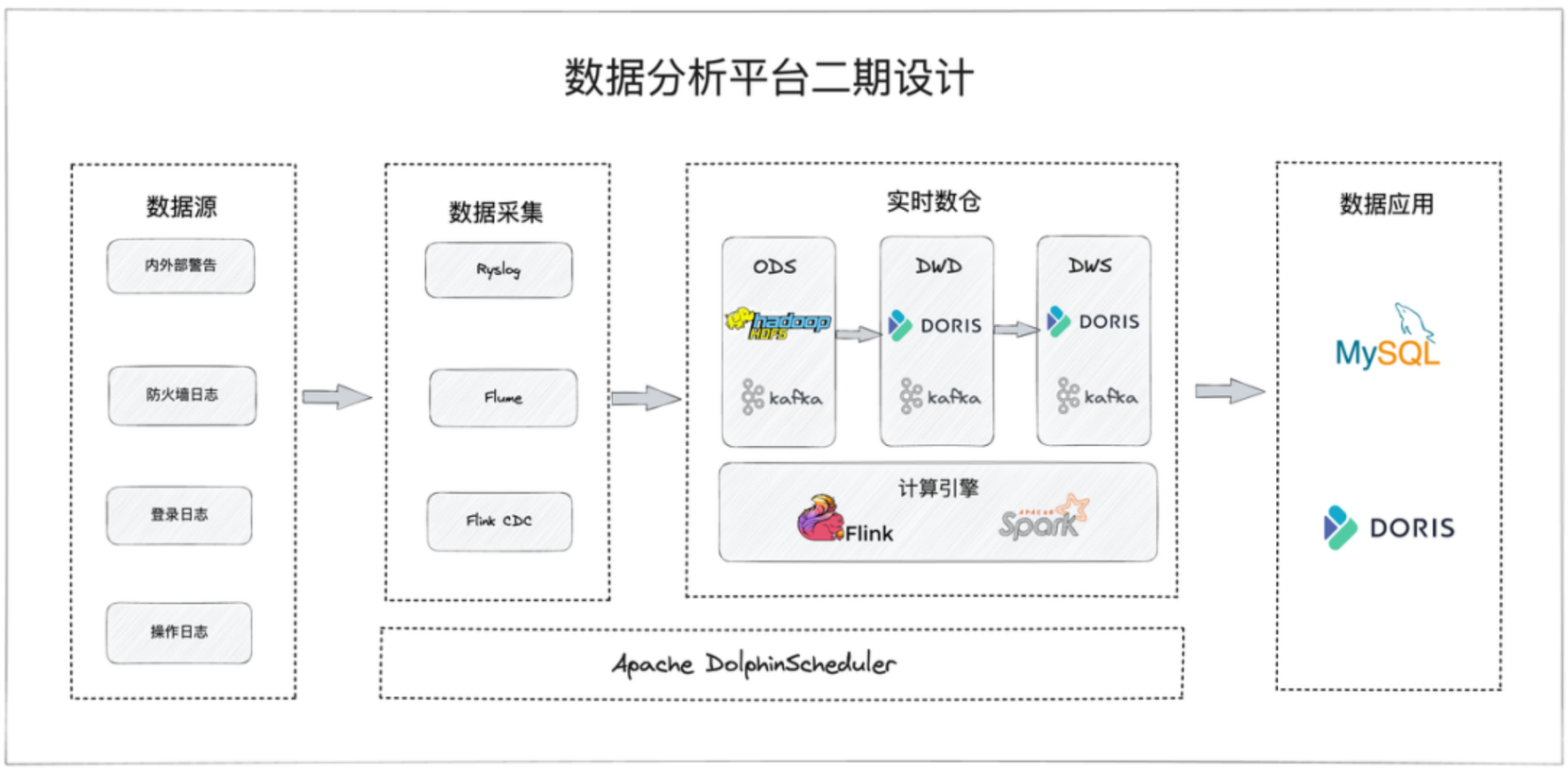
随着一期架构问题的逐步暴露，我们迫切需要对数据分析平台进行更新迭代。对于二期建设来说，提升数据的实时性被确立为首要目标，为了实现这一目标，我们计划增加实时数据处理链路，以更好地实现数据的实时收集、处理和查询要求，为系统稳定和网络安全提供更有力的支持和保障。其次，为解决一期平台存在的并发能力不足、多表 Join 性能低等核心问题，提升 OLAP 引擎性能成为二期建设的的另一关键目标，因此亟需对一期平台中 OLAP 引擎 ClickHouse 进行替换，以满足业务侧日益严格的数据分析和处理需求。

在此背景下，我们考虑是否可以只选择一个新的实时数据仓库同时满足以上两个目标，一方面即能帮助我们构建实时数据分析处理链路，另一方面又可以作为性能更强悍、更易用的 OLAP 分析引擎，这样不仅可以简化数据处理流程、提高实时效率，而且可以降低平台运维管理的成本。

为了找到符合条件的数据库，我们进行了多方调研和对比研究，最终选择以 Apache Doris 为核心来构建统一的实时数据仓库体系。为了直观展示 Apache Doris 的性能和功能特点，我们使用 Apache Doris 与 ClickHouse 进行了对比，其中最直观的感受是 Apache Doris 在系统并发、Join 性能以及多个功能的易用性都更为领先。

功能	Apache Doris	ClickHouse
响应时间	快	快
并发度	高	低
Join 性能	高	低
吞吐量	高	高
明细查询	支持	支持
幂等性	支持	不支持
可拓展性	高	低
SQL 支持	是	有限
运维难度	低	高

基于 Doris 的实时数据仓库



在项目二期的建设中，我们使用 Apache Doris 替换了 Hive 成功搭建实时数据仓库，实现数据的实时采集、处理和分析，同时使用 Apache Doris 替换 ClickHouse 作为 OLAP 引擎。架构工作机制如下所示：

- ODS 贴源层：主要用于存放未经处理的原始数据，通过 Flume 等实时采集工具，将各个厂商未经处理的原始日志以及告警数据统一汇集到 Kafka 中，同时完全相同的数据也会被存入 HDFS 中一份，作为原始数据核查依据或进行数据回放。

- DWD 明细层：该层为事实表，数据通过 Flink 计算引擎实时对生产数据及字段进行清洗、标准化、回填、脱敏之后写入 Kafka 。Kafka 中的数据还会对接到 Doris 中，以支持明细日志数据详情回溯查询、准实时模型分析、实时大屏及报表业务。由于大部分日志数据对于数据重复不是很敏感，因此 DWD 层采用 Doris 的 Duplicate Key 模型。

- DWS 汇总层：以明细层 Kafka 数据为基础，通过动态规则引擎进行细粒度的聚合分析，为后续的业务查询和 OLAP 分析做准备，同时大部分建模分析的结果也集中在 DWS 层。

- ADS 应用层：该层主要使用 Doris 的 Aggregate Key 模型和 Unique Key 模型对以上三层的数据进行自动聚合或者自动更新，以满足前端人员的具体分析需求。

新架构的应用实践

日增百亿数据，稳定快速导入

数据分析平台平均每天有 150 亿的业务日志数据新增，面对如此大规模的数据量，我们需要考虑如何将数据快速实时稳定入库。经调研，Doris Flink Connector 组件（主要依赖 Doris Stream Load ）可以实现海量数据快速导入。并且其使用非常简单，只需要导入相关依赖包进行简单的配置即可进行。在应用 Doris Flink Connector 后，**数据写入性能可达到每秒 20-30 万条**，极大地提升了数据导入的速度和效率，同时也不会对正常的数据分析造成干扰。

在采用 Flink 进行高频实时写入 Doris 时，如果未合理调整参数配置，可能导致数据版本堆积。为避免该问题，我们进行了以下调整优化：

- Flink 优化：为减轻 Doris 的写入压力，可通过提高 Flink 的 Checkpoint 时间来减少版本数量。具体来说，我们可以将 Checkpoint 时间从之前的 15 秒提高为 60 秒，以减少批次写入频率，降低 Doris 单位时间处理事务数量。这样可以在不影响业务的情况下，缓解写入压力，避免产生大量的数据版本。

- 数据预处理：为了减轻 Doris 的写入压力，部分数据我们会先在 Flink 中通过主键 ID 进行预聚合，将来自多个表中相同的 ID 进行处理并构建大宽表，降低多流数据的写入资源消耗。

- Doris 优化：调整 Doris BE 参数，增加 CPU 资源参与 Compaction 操作；根据业务设置合理的表分区、分桶和副本数量，避免过多分分片，以降低 Compaction 的开销。同时增大 `max_tablet_version_num` ，避免版本堆积。

通过以上优化措施，每日新增的百亿数据可以平稳导入 Doris 中，整个导入过程中 BE 表现稳定，Compaction Score 始终保持低位，大批量数据的写入对于前端查询的性能也没有造成任何影响。同时在 Doris 的 Unique Key 模型的加持下，我们可以利用 Flink 对输入数据进行关联、聚合等处理，再以微批、精准一次性写入 Doris 中，实现了数据秒级更新。

存储资源合理配置，成本节约 50%

日志数据具有非常大的数据量和数据增长速度，如果不对存储资源进行合理分配和控制，存储成本将会成为一个巨大的负担。日志数据中也会存在重要性的区分，有一定比例的数据价值密度比较低，如果毫无差别的将这些数据都存储下来，不仅会造成存储浪费，也会增加数据分析的难度。为了有效解决这些问题，我们采用了一系列策略来降低数据存储成本：

- **ZSTD 高效压缩算法：**利用 Doris 的新特性——ZSTD 高效压缩算法进行压缩存储。在建表时指定压缩方法为 ZSTD，特别是对数据量超过 T 级别的数据表，这种压缩方法可以有效地减少数据占用的存储空间，数据压缩比最高可达 1:10。即使采用 3 副本来保证数据的高可靠，数据存储占用的空间仍有非常大幅度的降低。

- **冷热数据精细化管理：**在 Doris 中只存储近一年的数据，将更早的数据备份到成本更低的存储介质中。同时使用热数据转冷的功能，在 SSD 中仅存储最近 7 天的数据，将 7 天之前的数据转存到 HDD 中，以进一步降低存储成本。这样可以根据数据的使用频率，合理分配存储资源，达到性能和成本的平衡。目前 Apache Doris 2.0 版本已经实现了对冷热数据分层功能的支持，这一功能可以将冷数据下沉到存储成本更加低廉的对象存储中，冷数据在对象存储上的保存方式也从多副本变为单副本，存储成本进一步降至原先的三分之一，同时也减少了因存储附加的计算资源成本和网络开销成本，目前我们正在积极测试中，未来有机会也会与大家分享实践经验。

- **分区级副本设置：**将 3 个月以内的数据设置为高频使用数据，将其分区设置为 3 副本；将 3–6 个月的数据分区设置为 2 副本；将 6 个月之前的数据分区设置为 1 副本。这样可以根据数据的使用情况，合理分配副本数量，实现存储成本降低的同时也充分利用多副本来提升热数据的查询性能。

借助于 Doris 极高效率的压缩算法、冷热数据分层管理、分区级副本设置等功能，**可对存储资源合理分配，最终实现存储成本节约 50%，成功达到性能和成本的平衡。**

数据规模分级查询，查询速度提升 10+ 倍

日志中包含了许多对分析及时性要求非常高的数据，例如异常事件、故障信息等，因此为了保障日志数据的查询效率，我们以数据量的级别为基准采用了不同的查询策略：

- 对于 **100G 以下**的数据，可以采用分区表的形式进行查询。在业务初期业务表按照天进行分区，每天执行任务需要手动管理分区为我们带来了非常大的维护成本。后来我们利用 Doris 的动态分区功能，针对数据量较大的表可以使用小时作为分区字段，为了避免分区内数据倾斜，以雪花 ID 作为分桶字段，保证数据的均衡。此外为了避免数据积压，我们还开启了动态分区的起始偏移，保留近 20 天的数据来支撑业务分析。这样可以有效地降低数据积压的风险，同时也能够满足业务的分析需求。

- 对于 **100G 到 1T** 的数据，我们采用物化视图进行查询，物化视图是一种预先计算并存储结果集的方式，可以减少查询所需的计算时间和资源消耗，从而提高查询效率。Doris 系统提供了完整的物化视图 DDL 语法，可用于创建、查看和删除等操作，这些语法与 PostgreSQL 和 Oracle 语法一致，使用简单、不需重新学习。

- 对于**上百 T** 的数据，我们通过 Aggregate 聚合模型表进行查询，使用 Aggregate 模型在数据写入前进行预聚合，通过以上方式，我**们成功将 20 亿条数据的查询时间进一步缩短至 1–2s**，有效提高了数据查询的效率。

在一期数据分析平台中，大部分业务场景都是通过 T+1 的方式进行计算。而在基于 Doris 的二期数据分析平台中，我们实现了对大部分业务准实时（分钟以及小时级）和实时计算场景的支持。同时结合以上优化措施，极大降低了各种维度指标的统计时间，以往需要分钟级别的明细查询，现在可以在**毫秒级别**迅速响应，极大地改善了用户体验；另外，在 Doris 中，我们能够快速对百亿级别的大表进行不同维度的数据分析，只需要几秒即可获得查询结果，大大提高了联通各业务部门数据分析的能力。

收益总结

自引入 Apache Doris 以来，我们已经部署了多个集群、数十台机器，支持了中国联通 30 多条业务线和数百个实时作业，日增日志数据百亿级别，单个集群的数据规模达到数 PB 。Apache Doris 的成功应用为联通带来了多方面收益，主要包括如下方面：

在数据导入方面，对于联通而言，每天都面临着庞大的日志增量，并且这些数据的实时性和准确性对于业务发展和决策至关重要，而 Doris Flink Connector 帮助我们实现了数据快速且稳定导入，可轻松应对日增百亿数据的导入要求，为后续的数据处理和分析提供了更高效的解决方案。

在存储资源分配方面，由于数据量庞大、存储周期长等原因，日志数据的存储成本一直是运营商面临的难题，通过采用 Doris 高效的压缩算法、冷热数据精细管理、分区级副本设置等功能，帮助我们降低了数据存储成本，数据存储利用效率和价值得到显著提升。

在查询性能方面，快速获取日志数据查询结果可以帮助运营商及时掌控网络及系统情况，及时发现并解决问题，也有利于及时了解用户需求和行为，优化营销策略和服务方案。Doris 在查询性能方面提供了强大的支持， 能够处理百亿级别大表按小时/天级别的明细查询，并支持不同维度聚合查询分析。业务线整体响应时间可在秒级或毫秒级别完成，甚至可以在 1–2s 内完成对 20 亿条数据的查询，查询速度较之前提升了 10+ 倍。

未来规划

在最新发布的 Apache Doris 2.0 版本中，Apache Doris 提供了大量新的功能，比如倒排索引功能和冷热数据分层等，对于日志分析场景来说都是具有重要意义的更新。目前我们是以数据存储周期为基准进行副本分配，并按照数据热度分别存储在 SSD 和 HDD 中，后续我们将使用冷热数据分层新功能，将数据从 SSD 或者 HDD 下沉到对象存储中，从而降低数据存储成本，进一步达到服务器磁盘资源节省的目的。此外，我们正在对倒排索引功能进行测试，并计划先在小范围业务场景推广使用，倒排索引对于字符串类型的全文检索和普通数值、日期等类型的等值、范围检索具有更高效的支持，希望通过倒排索可以帮助我们进一步提高日志数据查询的效率和准确度。

除此之外，基于联通的使用场景，我们对自动分桶功能提出一些建议。目前自动分桶计算逻辑是根据最近的分区数据量来动态决定当前分区的分桶数目，这种方式适用于分区数据量呈线性关系的业务表。然而，由于我们的业务表在白天的数据量较多，夜晚数据量较少，因此使用自动分桶会导致白天部分分区具有较少的分桶，而夜晚分区则具有较多的分桶。因此，未来我们期望社区可以增加一种新的分桶规则，以前一天的数据分区存储情况为参照，来对当天的分区进行自动分桶，这样可以更加准确的根据业务表特点进行自动分桶。当然我们也将对该功能的优化进行探索，及时与社区交流，将最新的优化代码贡献到社区，共同推动社区的发展进步。

最后，非常感谢 SelectDB 的同学，在我们在使用 Apache Doris 的过程中遇到任何问题，他们都能给予了快速的响应与技术支持，未来我们会持续的将在实践过程中取得的相关成果贡献到社区。希望 Apache Doris 与 SelectDB 越来越好！

查询平均提速 700%

奇安信基于 Apache Doris 升级日志安全分析系统

作者：奇安信服务端技术专家，舒鹏

导读：数智时代的到来使网络安全成为了不可忽视的重要领域。奇安信作为一家领先的网络安全解决方案领军者，致力于为企业提供先进全面的网络安全保护，其日志分析系统在网络安全中发挥着关键作用，通过对运行日志数据的深入分析，能够对漏洞和异常行为生成关键见解，帮助企业建立有效的防御策略。本文将深入探讨奇安信在网络安全与日志分析解决方案的关键优势，了解基于 Apache Doris 构建的全新一体化日志存储分析平台如何实时监测和分析日志事件，加强对可疑活动的追踪与应对，提升系统安全性与快速响应能力。

奇安信是中国企业级网络安全市场的领军者，专注于为政府和企业用户提供新一代网络安全产品和服务。目前核心产品天擎终端安全系统在国内已有 4000 万政企用户部署、全国部署服务器超过 100 万台、服务超 40 万大型机构。作为网络安全国家队，奇安信立志为国家构建安全的网络空间，在终端安全、云安全、威胁情报、态势感知等领域的技术研发持续领先。

随着现代企业数字化转型的不断深化，大数据、物联网、5G 等创新技术的广泛应用加速了企业的数字化转型步伐，这使得原先的网络边界被打破，多源多样的终端设备成为了新的安全边界。

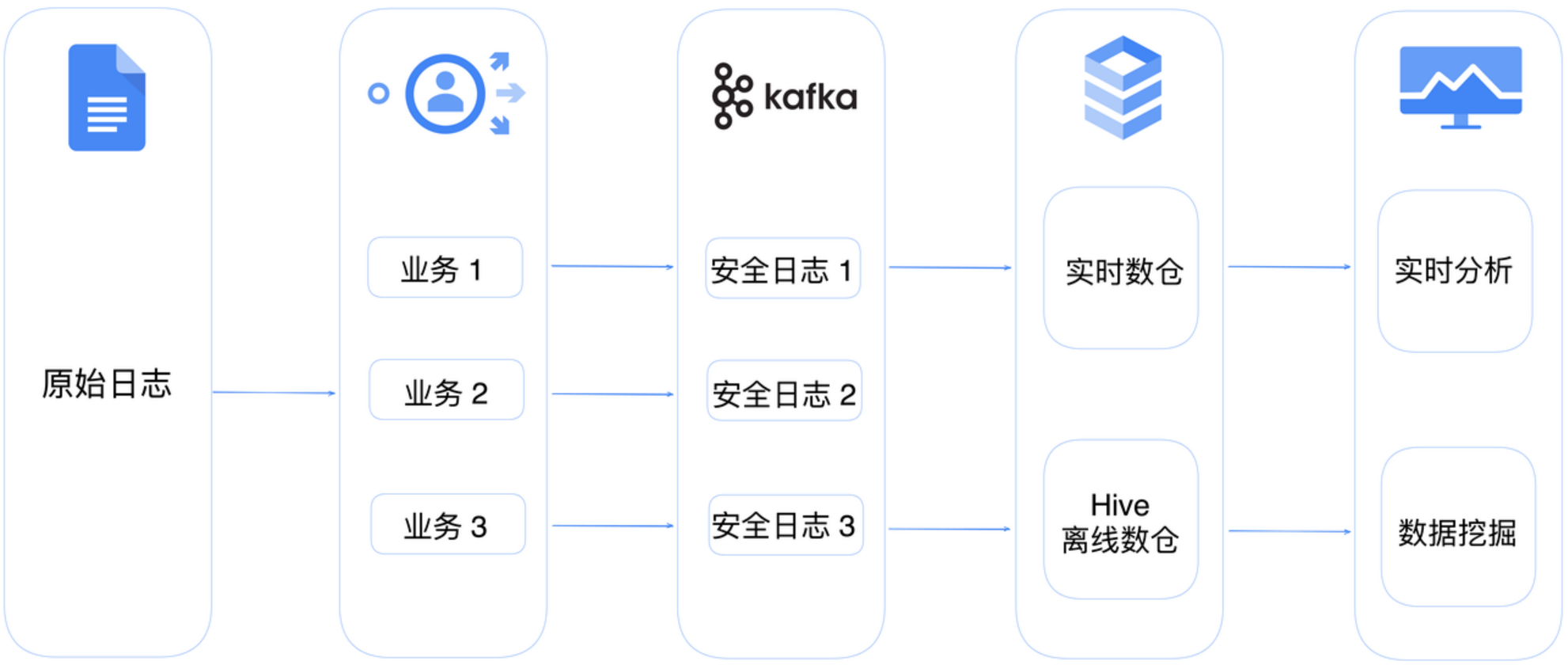
网络安全系统的防御性能与日志分析密不可分，当网络设备、操作系统以及应用程序在运行时，会产生大量的运行日志，其中蕴涵了丰富的数据价值。最大化地利用运行日志数据能够有效检测内部系统的安全风险、还原攻击路径、回溯攻击入口等，可以进一步提升系统安全性、保障企业网络安全，因此日志分析系统在其中发挥着不可或缺的作用。

本文将介绍奇安信在网络安全场景中，基于 Apache Doris 进行架构升级迭代并建设全新一体化日志存储分析平台的实践经验。

早期架构痛点与需求

安全日志平台的架构如下图所示，原始的设备、系统日志首先经过业务处理环节，包括归一化和扩充维度等操作。这些处理步骤旨在将来自不同设备和系统日志转化为半结构化 JSON 格式的安全日志，并将其写入 Kafka 消息队列中。

最新的日志会被写入实时数仓，安全分析师可以通过分析平台对实时数仓中的最新数据进行交互式查询，从而进行攻击研判和追踪溯源等安全分析工作。另外，离线数仓用于保存历史数据，以支持长周期数据挖掘的离线分析。



在以上日志数据平台中，日志数据的写入速度与查询对上层业务人员进行实时安全事件监控和分析至关重要，这也是当前我们所面对的最主要痛点。

一方面，每天所生产的安全日志数据达到千亿级，写入压力很大。最初我们选择使用某 Apache Doris 的 Fork 版本来存储日志数据，但在实际应用中，随着每天新增日志量的不断增长，入库速度逐渐降低、集群写入压力过大、高峰期数据积压严重，对集群稳定性造成很大影响，并且数据压力较高时、查询效率也达不到有效果的保证。随后我们对集群进行多次扩容，从 3 节点逐步扩容到 13 节点，尽管机器成本已经大幅超过预期、但写入效率并没有发生本质的改善。

另一方面，业务人员在进行安全日志分析时，经常需要对文本字段（如 URL，payload 等）进行关键字匹配。在原系统中只能通过 SQL LIKE 进行全量扫描和暴力匹配，整体查询性能不佳，千亿级数量的数据表查询耗时接近分钟级甚至达到数百秒，即便按照时间区间过滤大量数据后、查询耗时仍在数秒到数十秒。一旦遇到并发查询性能还会恶化，很难满足日常安全分析需求。

除写入和查询效率以外，运维监控也是我们的痛点之一，该厂商提供的可视化运维系统需要商业 License 授权，对于开源社区用户不友好，集群维护处于原始手动状态。

架构选型与升级的思考

为了解决过去版本的痛点、满足更高效实时的日志分析诉求，我们亟需对早期系统升级改造。同时面向安全日志分析场景，我们也对新日志分析平台的架构提出了更高的要求：

- **写入性能：**系统一方面需要支持海量病毒查杀事件等数据实时写入与存储，以满足分析时效性的要求，另一方面需要基于日志数据 Schema Free 特点支持丰富数据类型的写入与变更。
- **查询性能：**由于日志查询分析会涉及对文本类型、JSON 数据进行全文检索、日期或普通数值的范围查询，系统需要对字符串提供模糊查询的能力，还需要支持能够灵活创建且类型丰富的索引，以加速筛选过滤海量数据，提升查询效率。
- **存储成本：**设备每天产生大量的日志数据，为了挖掘这些有价值的日志信息，业务人员还需要从数据中进行筛选和分析，并对异常日志回溯追踪，这使得日志存储的规模很大、存储周期相对较长，因此高性价比的存储成本也是系统构建的目标之一。
- **运维成本：**系统自身运维简易程度以及是否具备合适的管控工具都能帮助我们进一步提效。

在持续关注业界 OLAP 数据库的过程中，我们发现 Apache Doris 最近一年的发展非常迅猛，最新的 2.0 版本也把日志存储和检索分析作为新的发力点，推出了倒排索引、NGram BloomFilter 索引等特性，对关键词检索、LIKE 文本匹配的性能有大幅提升，与我们文本检索慢的痛点需求非常契合，因此开启了新架构的升级之旅。

架构升级之旅

上文中提到，在整体架构选型过程中我们主要关注的地方包括写入性能、查询性能、数据存储成本以及运维成本等方面。在架构升级过程中，我们选择了 Apache Doris 当时最新发布的 2.0 版本，具体升级收益如下。

写入性能提升超 200%

为了评估 Apache Doris 写入的极限性能，我们初期使用与线上系统相同配置的 3 台服务器，从 Kafka 接入线上真实写入流量，测试期间当 CPU 写入效率跑满至 100% 时写入吞吐达到了 108 万条/s、1.15 GB/s，写入数据的可见性延迟保持在秒级。

而线上运行的原系统集群规模达 13 台，在同样的数据写入情况下，CPU 利用率 30% 左右、写入吞吐仅 30 万条/s，并且存在高峰期 CPU Load 高、系统响应慢的问题。

根据测试结果，我们预估架构替换为 Apache Doris 后保持同样 30% 的 CPU 占用，只需要 3 台服务器即可满足写入需求，机器资源成本至少节约 70%。值得注意的是，在测试中对 Apache Doris 表中一半字段开启了倒排索引，如果不开启倒排索引的话，写入性能在之前基础上还能够再提升 50% 左右。

存储成本降低近 40%

在看到写入性能的大幅提升后，Apache Doris 存储空间占用也给我们带来了惊喜。在开启倒排索引的前提下，存储空间比原系统不具备倒排索引还要略低，压缩比从 1 : 4.3 提高至 1 : 5.7。

通过对比 Apache Doris 在磁盘上存储的文件大小，同一份数据的索引文件(.idx)与数据文件(.dat) 大小相差无几。换言而之，增加索引后 Doris 数据膨胀率大约在 1 倍左右，与许多数据库和检索引擎 3–5 倍的膨胀率相比，Doris 的数据存储空间占用相对较低。经过研究发现，Apache Doris 采用了列式存储和 ZSTD 压缩算法来优化存储空间占用。Doris 将原始数据和倒排索引都以列的形式存储，使同一列的数据被存储在相邻位置，从而实现了更高的压缩率。

ZSTD 是一个优秀的新型压缩算法，使用了智能优化算法，相较于常见的 GZIP 算法， ZSTD 具有更高的压缩率更和更快的压缩解压速度，尤其在处理日志场景时表现非常出色。

查询性能平均提升 690%

对于业务最关注的查询性能，我们从线上查询日志进行去重后分析出 79 条 SQL，在同一天总数据（1000 亿条）、同样规模的集群（10 BE 节点）上对比测试 Apache Doris 与原系统的查询耗时。

我们发现，与原系统相比，所有的查询语句均有明显提升，整体查询性能提升近 7 倍，有 26 条 SQL 查询语句性能提升 10 倍以上，其中 8 条 SQL 查询提升 10–20 倍、14 条 SQL 查询提升 20–50 倍、还有 4 条 SQL 查询提升 50 倍以上。最大差异的一条 SQL 查询语句为 Q43，在原系统中执行时间接近一分钟，在 Apache Doris 中仅需不到 1 秒，其性能差异高达到 88 倍。

Query	原系统耗时 (s)	Apache Doris 耗时 (s)	性能提升倍数
1	13.83	5.29	2.6
2	6.05	0.65	9.4
3	7.57	1.68	4.5
4	1.56	0.74	2.1
5	7.08	0.28	25.2
6	0.8	0.12	6.9
7	5.15	0.3	17.1
8	17.16	4.84	3.5
9	11.89	3.13	3.8
10	16.14	5.03	3.2
11	16.22	5.46	3
12	3.64	1.39	2.6
13	9.4	1.49	6.3
14	6.4	1.05	6.1
15	7.12	0.89	8
16	2.42	0.55	4.4
17	3.02	0.89	3.4
18	400.43	44.69	9
19	12.22	1.51	8.1
20	3.82	1.23	3.1
21	391.74	38.52	10.2
22	26.89	0.53	50.3
23	12.31	0.2	62
24	24.93	1.12	22.2
25	59.21	12.54	4.7
26	3.98	1.12	3.5
27	3.46	0.34	10.1
28	3.05	0.1	31.6
29	3.88	1.89	2.1
30	5.78	0.14	42
31	3.65	0.1	35.6
32	14.54	6.12	2.4
33	5.15	0.66	7.8
34	5.26	1.03	5.1
35	16.32	1.91	8.6
36	19.8	2.55	7.8
37	23.18	15.8	1.5
38	3.36	0.32	10.6
39	4.77	0.12	38.5
40	2.9	0.31	9.4
41	8.84	2.02	4.4
42	4.87	0.23	21
43	58.66	0.67	88.2
44	34.61	0.78	44.4
45	7.63	1.63	4.7
46	5.45	1.54	3.5
47	5.79	1.53	3.8
48	2.33	0.53	4.4
49	61.77	18.38	3.4
50	25.18	0.74	33.9
51	40.06	15.64	2.6
52	27.55	0.54	51.3
53	7.69	0.61	12.6
54	5.55	0.54	10.3
55	5.12	0.33	15.4
56	33.62	12.48	2.7
57	9.5	0.6	15.7
58	5.64	0.78	7.2
59	5.91	0.97	6.1
60	7.54	3.92	1.9
61	8.7	1.08	8.1
62	6.52	1.02	6.4
63	7.03	1.12	6.3
64	6.4	3.29	1.9
65	5.27	3.26	1.6
66	6.13	0.69	8.8
67	4.74	1.05	4.5
68	4.62	0.8	5.8
69	4.9	1.03	4.7
70	8.08	0.93	8.6
71	4.45	0.15	30.6
72	4.65	0.72	6.4
73	3.88	0.76	5.1
74	3.16	0.76	4.2
75	24.96	0.56	44.4
76	23.15	0.69	33.6
77	40.29	1.94	20.8
78	30.78	0.84	36.4
79	10.27	1.33	7.7
Total	**1,757.32**	**253.12**	**6.9**

针对性能提升幅度高的查询，我们进行了对比分析并发现了其中几个共同点：

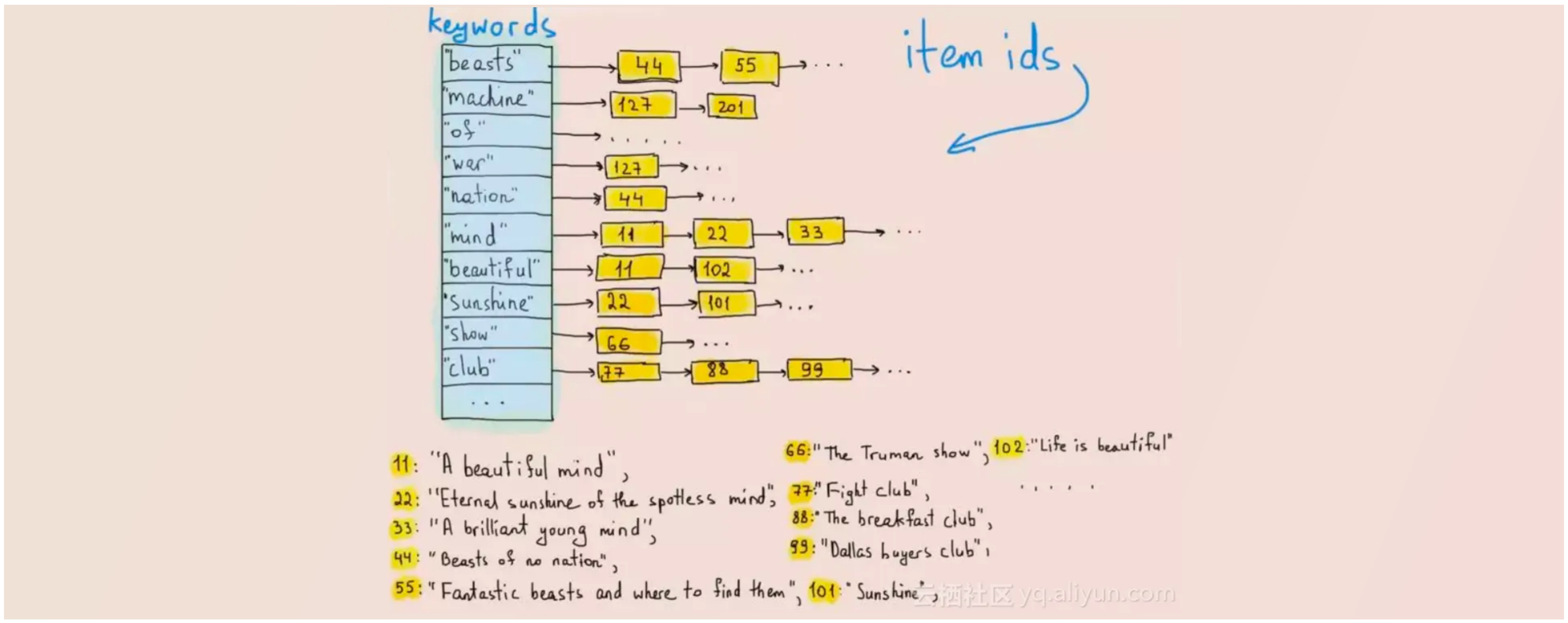
1. 倒排索引对关键词查找的加速：Q23、Q24、Q30、Q31、Q42、Q43、Q50 等

```
-- 例如q43 提升88.2倍

SELECT count() from table2
WHERE ( event_time >= 1693065600000 and event_time < 1693152000000)
  AND (rule_hit_big MATCH 'xxxx');
```

倒排索引的原理是在数据写入时将每一行文本切分成单词，并在索引中记录每个单词在哪些行出现过，例如下图中的 "machine" 在第 127 和 201 行出现过，当需要检索包含某个词的数据时，只需要在索引中查这个词在哪些行出现，将对应行号的数据读取出来即可。

这种基于倒排索引进行关键词检索的技术，相较于基本的暴力扫描后进行文本匹配具有显著的优势，一方面极大地减少了需要读取的数据量；另一方面，在查询过程中无需进行文本匹配操作，因此查询效率往往提升一个数量级甚至更高。



2. NGram BloomFilter 索引对 LIKE 的加速：Q75、Q76、Q77、Q78 等

```
-- 例如q75 提升44.4倍

SELECT * FROM table1
WHERE  ent_id = 'xxxxx'
   AND event_date = '2023-08-27'
   AND file_level = 70
   AND rule_group_id LIKE 'adid:%'
ORDER BY event_time LIMIT 100;
```

对于要查找的非一个完整关键词的场景，LIKE 仍然是有用的查询方式，Apache Doris 的 NGram BloomFilter 索引能对常规的 LIKE 进行加速。

NGram BloomFilter 索引与普通 BloomFilter 索引不同，它不是将整个文本放入 BloomFilter ，而是将文本分成连续的子串，每个子串长度为 n ，并将他们放入 NGram BloomFilter 中。对于 `cola LIKE '%pattern%'` 的查询，将 `'pattern'` 按照同样的方式分成长度为 n 的子串，判断每个子串在 BloomFilter 中是否存在，如果有一个子串不存在，则说明 BloomFilter 对应的数据块中没有跟 `'pattern'` 匹配的数据块，因此通过跳过数据块扫描的步骤，达到加速查询的效果。

3. 满足条件的最新 TopN 条日志明细查询优化：Q19-Q29 等

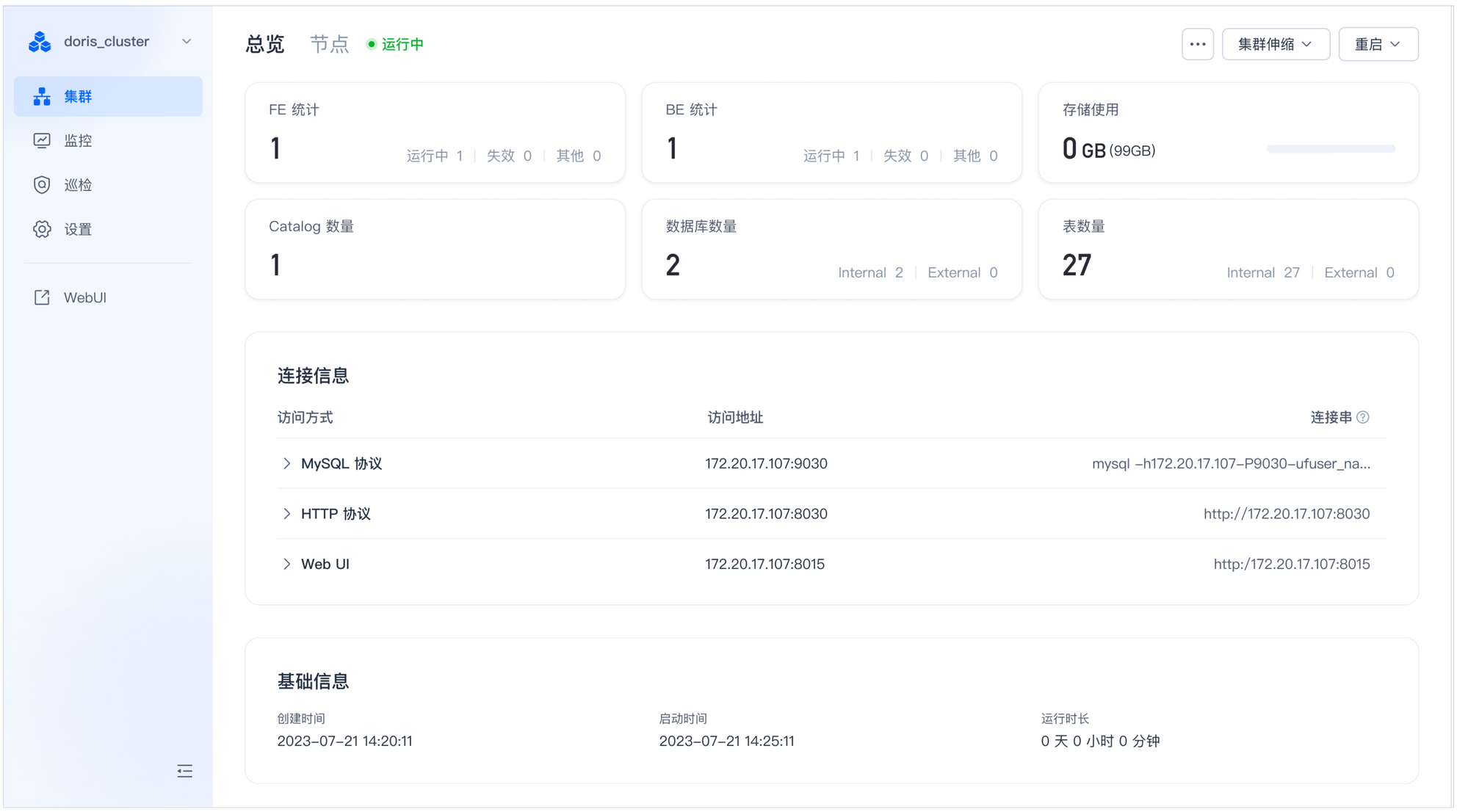
```
-- 例如q22，提升50.3倍

SELECT * FROM table1
where event_date = '2023-08-27' and file_level = 70
  and ent_id = 'nnnnnnn' and file_name = 'xxx.exe'
order by event_time limit 100;
```

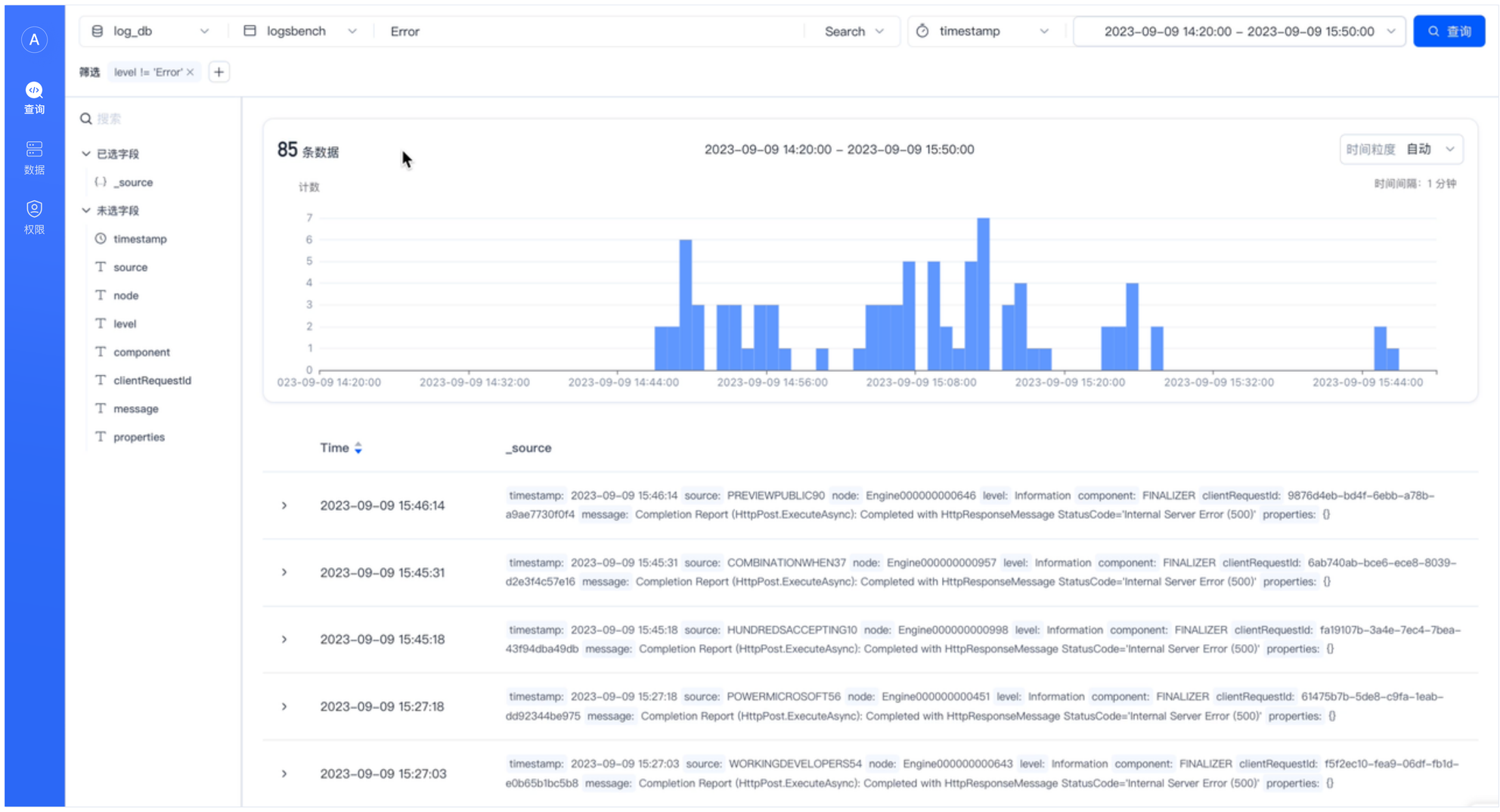
这种 `SELECT * FROM t WHERE xxx ORDER BY xx LIMIT n` 的查询，在查找满足某种条件的最新 n 条日志时使用频率非常高，Apache Doris 针对这种 SQL 查询模式进行了专门的优化，根据查询的中间状态确定排序字段的动态范围，并利用自动动态谓词下推的方式，避免读全部数据进行排序取 TopN，从而减少需要读取的数据量（有时甚至可以减少一个数量级），进而提升了查询效率。

可视化运维管控和可视化查询 WebUI，最大化减少运维和探索分析成本

为了提高日常集群维护效率，我们使用了飞轮科技免费开放的可视化集群管理工具 Cluster Manager for Apache Doris （以下简称 Doris Manager ）。Doris Manager 提供的功能可以满足日常运维中集群监控、巡检、修改配置、扩缩容、升级等操作，降低登陆机器手动操作的麻烦和误操作风险。



除了管控 Apache Doris 集群之后，Doris Manager 还集成了类似 Kibana 的可视化日志探索分析 WebUI，对于习惯 ELK 日志分析的用户非常友好，支持关键词检索、趋势图展示、趋势图拖拽日期范围、明细日志平铺和折叠展示、字段值过滤等交互方便的探索式分析，跟日志场景探索下钻的分析需求很契合。



总结与规划

在跟随 Apache Doris 2.0-alpha，2.0-beta，2.0 正式版本发布的节奏，我们根据业务场景进行了详细的评测，也为社区反馈了不少优化建议，得到社区的积极响应和解决。系统经历试运行一个月之后，我们将 2.0.1 版本正式用于生产环境，替换了原系统集群，完成架构升级改造，实现了写入性能、查询性能、存储成本、运维成本等多方面收益：

- **写入性能提升 3 倍以上：**目前，奇安信日志分析平台每日平均有数千亿的新增安全日志数据，通过 Doris 的 Routine Load 将数据实时稳定写入库，保障数据低延迟高吞吐写入。
- **查询性能平均提升 7 倍：**查询响应时间大幅减少，与之前的查询效率相比达到平均 7 倍提升，其中业务特别关注的全文检索速度达到 20 倍以上的提升，助力日志分析与网络安全运营效率。
- **高效便捷的可视化管理：**Cluster Manager for Apache Doris 工具提供了可视化集群监控告警平台，满足日常集群监控等一系列操作，同时 WebUI 多种功能为分析人员提供了操作简单、使用便捷的交互式分析。总而言之，Doris 的易用性、灵活性大幅降低了开发、运维、分析人员的学习与使用成本。

后续我们还将在日志分析场景下探索更多 Apache Doris 的能力。我们将扩大 JSON 数据类型的相关应用，加强系统对于半结构化数据深度分析的能力。同时，我们也非常期待 Apache Doris 2.1 版本中新增的 Variant 可变数据类型，支持存储任意结构的 JSON 数据，支持字段个数与类型的变化，让业务人员灵活定义特殊字符，以更好地实现半结构数据 Schema Free 分析需求。

非常感谢 SelectDB 团队一直以来对我们的技术支持，助力奇安信走向“体系化防御、数字化运营”的网络日志安全管理，帮助客户准确识别、保护和监管网络设备与各类系统，确保业务人员在任何时候都能够安全、可信、稳定地访问数据与业务。

最后，我们也将持续参与到 Apache Doris 社区建设中，将相关成果贡献回馈社区，希望 Apache Doris 飞速发展，越来越好！

秒级数据写入，毫秒查询响应

天眼查基于 Apache Doris 构建统一实时数仓

作者：天眼查实时计算负责人，王涛

导读：随着天眼查近年来对产品的持续深耕和迭代，用户数量也在不断攀升，业务的突破更加依赖于数据赋能，精细化的用户/客户运营也成为提升体验、促进消费的重要动力。在这样的背景下正式引入 Apache Doris 对数仓架构进行升级改造，实现了数据门户的统一，大大缩短了数据处理链路，数据导入速率提升 75 %，500 万及以下人群圈选可以实现毫秒级响应，收获了公司内部数据部门、业务方的一致好评。

天眼查是中国领先的商业查询平台，以公开数据为切入点、以关系为核心的产品，帮助传统企业或个人降低成本，为防范化解金融风险方面提供了产品化的解决方案。目前已收录全国3亿多家社会实体信息，300多种维度信息及时更新，致力于构建商业安全，从而实现“公平看清世界”。

业务背景

天眼查的数据仓库主要服务于三个业务场景，每个场景都有其特点和需求，具体如下：

- 亿级用户人群圈选：人群圈选场景中目前有 100+ 人群包，我们需要根据 SQL 条件圈选人群包，来支持人群包的交并差、人群包实时圈选和人群包更新通知下游等需求。例如：圈选出下单未支付超过 5 分钟的用户，我们通过用户标签可以直观掌握用户支付状态，为运营 & 营销团队提供更精细化的人群管理服务，从而提高转化率。
- 多元活动支撑的精准营销：该场景目前支持了 1000 多个指标，可支持即席查询，根据活动效果及时调整运营策略。例如在“开工季”活动中，需要为数据分析 & 运营团队提供数据支持，从而生成可视化的活动驾驶舱。
- 高并发的 C 端分析数据：该场景承载了 3 亿+实体（多种维度）的数据体量，同时要求实时更新，以供用户进行数据分析。

原有架构及痛点

为满足各业务场景提出的需求，我们开始搭建第一代数据仓库，即原有数仓：

在原有数仓架构中， Hive 作为数据计算层，MySQL、ES、PG 作为数据存储层，我们简单介绍一下架构的运行原理：

- 数据源层和数据接入层：MySQL 通过 Canal 将 BinLog 接入 Kafka、埋点日志通过 Flume 接入 Kafka，最后由 DataX 把 Kafka 中的数据接入数据计算层 Hive 中；
- 数据计算层：该层使用 Hive 中的传统的数仓模型，并利用海豚调度使数据通过 ODS -> DWD -> DWS 分层，最后通过 DataX 将 T+1 把数据导入到数据存储层的 MySQL 和 ES 中。
- 数据存储层：MySQL 主要为 DataBank、Tableau、C 端提供分析数据，ES 用于存储用户画像数据，PG 用于人群包的存储（PG 安装的插件具有 Bitmap 交并差功能），ES、PG 两者均服务于 DMP人群圈选系统。

问题与挑战：

依托于原有架构的投入使用，初步解决了业务方的需求，但随着天眼查近年来对产品的持续深耕和迭代，用户数量也在不断攀升，业务的突破更加依赖于数据赋能。精细化的用户/客户运营也成为提升体验、促进消费的重要动力。在这样的背景下，原有架构的缺点逐渐暴露：

- 开发流程冗长：体现在数据处理链路上，比如当面对一个简单的开发需求，需要先拉取数据，再经过 Hive 计算，然后通过 T+1 更新导入数据等，数据处理链路较长且复杂，非常影响开发效率。

- 不支持即席查询：体现在报表服务和人群圈选场景中，所用的指标无法根据条件直接查询，必须提前进行定义和开发。
- T+1 更新延迟高：T+1 数据时效性已经无法提供精确的线索，主要体现在报表和人群圈选场景上。
- 运维难度高：原有架构具有多条数据处理链路、多组件耦合的特点，运维和管理难度很高。

理想架构

基于以上问题，我们决定对架构进行升级改进，在正式升级之前，我们希望未来的架构可以做到以下几点：

- 原架构涉及 MySQL 、PG、ES 等多个组件，并为不同应用提供服务；我们希望未来的架构可以兼容 MySQL 协议，实现低成本替换、无缝衔接以上组件。
- 支持即席查询且性能优异，即席查询能够给业务方提供更灵活的表达方式，业务方可以从多个角度、多个维度对数据进行查询和分析，更好地发现数据的规律和趋势，帮助业务方更精准备地做出决策。
- 支持实时聚合，以减轻开发负担并保证计算结果的准确性。
- 统一数据出口，原架构中数据出口不唯一，我们希望未来的架构能更统一数据出口，缩短链路维护成本，提升数据的可复用性。
- 支持高并发， C 端的实时分析数据需要较高的并发能力，我们希望未来的架构可以高并发性能优异。

技术选型

考虑到和需求的匹配度，我们重点对 OLAP 引擎进行了调研，并快速定位到 ClickHouse 和 Apache Doris 这两款产品，在深入调研中发现 Doris 在以下几个方面优势明显，更符合我们的诉求：

- 标准 SQL：ClickHouse 对标准 SQL 支持有限，使用中需要对多表 Join 语法进行改写；而 Doris 兼容 MySQL 协议，支持标准 SQL ，可以直接运行，同时 Doris 的 Join 性能远优于 ClickHouse。

- 降本增效：Doris 部署简单，只有 FE 和 BE 两个组件，不依赖其他系统；生态内导数功能较为完备，可根据数据源/数据格式选择导入方式；还可以直接使用命令行操作弹性伸缩，无需额外投入人力；运维简单，问题排查难度低。相比之下，ClickHouse 需要投入较多的开发人力来实现类似的功能，使用难度高；同时 ClickHouse 运维难度很高，需要研发一个运维系统来支持处理大部分的日常运维工作。

- 并发能力：ClickHouse 的并发能力较弱是一个潜在风险，而 Doris 并发能力更占优势，并且刚刚发布的 2.0 版本支持了更高并发的点查。

- 导入事务：ClickHouse 的数据导入没有事务支持，无法实现 Exactly Once 语义，如导数失败需要删除重导，流程比较复杂；而 Doris 导入数据支持事务，可以保证一批次内的数据原子生效，不会出现部分数据写入的情况，降低了判断的成本。

- 丰富的使用场景：ClickHouse 支持场景单一，Doris 支持场景更加丰富，用户基于 Doris 可以构建用户行为分析、AB 实验平台、日志检索分析、用户画像分析、订单分析等应用。

- 丰富的数据模型：Doris 提供了Unique、Duplicate、Aggregate 三种数据模型，可以针对不同场景灵活应用不同的数据模型。

- 社区响应速度快：Doris 社区的响应速度是其独有特色，SelectDB 为社区组建了一直完备的社区支持团队，社区的快速响应让我们少走了很多歪路，帮助我们解决了许多问题。

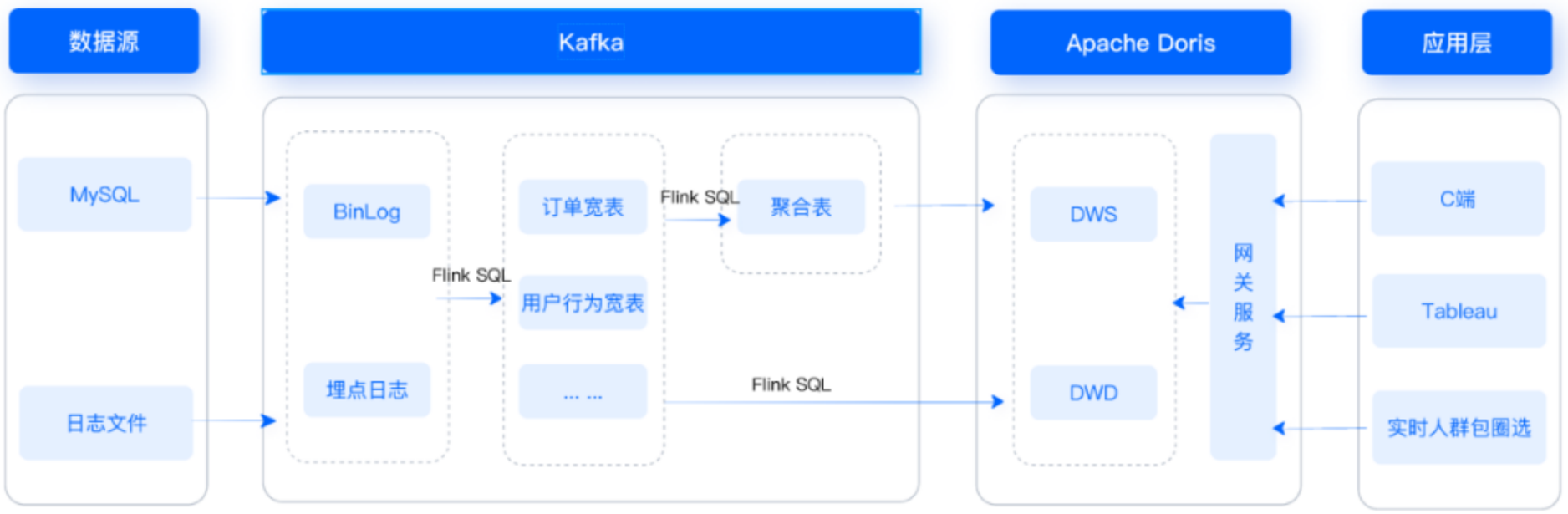
新数仓架构

经过对 Doris 进行综合评估，我们最终决定采用 Doris 对原有架构进行升级优化，并在架构层级进行了压缩。新的架构图如下所示：

在新架构中，数据源层和数据接入层与原有架构保持一致，**主要变化是将 Doris 作为新架构的数据服务层，统一了原有架构中的数据计算层和存储层，这样实现了数据门户的统一，大大缩短了数据处理链路，解决了开发流程冗长的问题。**同时，基于 Doris 的高性能，实现了即席查询能力，提高了数据查询效率。另外，Flink 与 Doris 的结合实现了实时数据快速写入，解决了 T+1 数据更新延迟较高的问题。除此之外，借助于 Doris 精简的架构，大幅降低了架构维护的难度。

数据流图

缩短数据处理链路直接或间接地带来了许多收益。接下来，我们将具体介绍引入 Doris 后的数据流图。



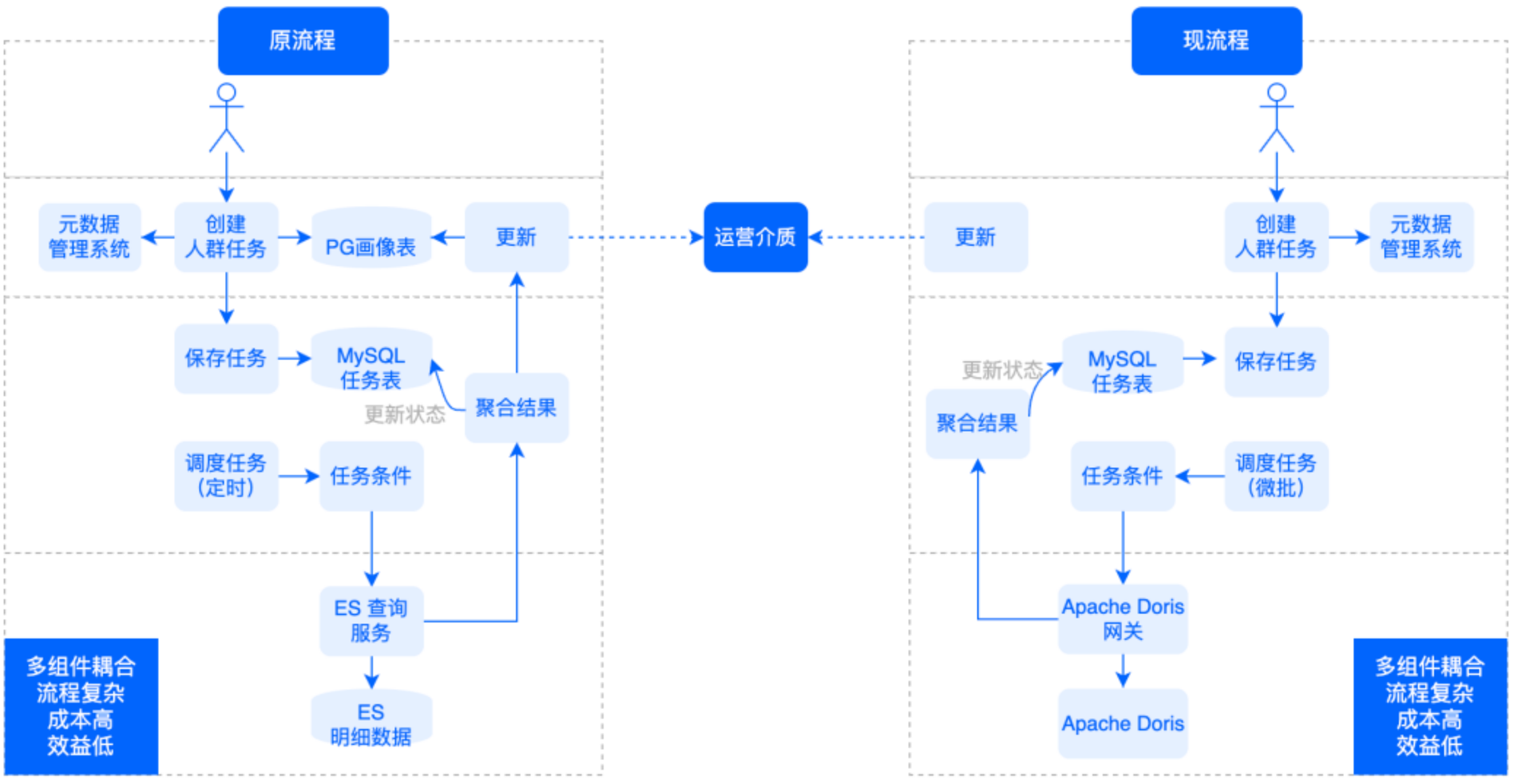
总体而言，数据源由 MySQL 和日志文件组成，数据在 Kafka 中进行分层操作（ODS、DWD、DWS），Apache Doris 作为数据终点统一进行存储和计算。应用层包含 C 端、Tableau 和 DMP 系统，通过网关服务从 Doris 中获取相应的数据。

具体来看，MySQL 通过 Canal 把 Binlog 接入 Kafka，日志文件通过 Flume 接入 Kafka 作为 ODS 层。然后经过 Flink SQL 进行清洗、关联维表，形成 DWD 层的宽表，并生成聚合表。为了节省空间，我们将 ODS 层存储在 Kafka 中，DWD 层和 DWS 层主要与 Doris 进行交互。DWD 层的数据一般通过 Flink SQL 写入 Doris。针对不同的场景，我们应用了不同的数据模型进行数据导入。MySQL 数据使用 Unique 模型，日志数据使用 Duplicate 模型，DWS 层采用 Aggregate 模型，可进行实时聚合，从而减少开发成本。

应用场景优化

在应用新的架构之后，我们必须对业务场景的数据处理流程进行优化以匹配新架构，从而达到最佳应用效果。接下来我们以人群圈选、C端分析数据及精准营销线索为主要场景，分享相关场景流程优化的实践与经验。

人群圈选



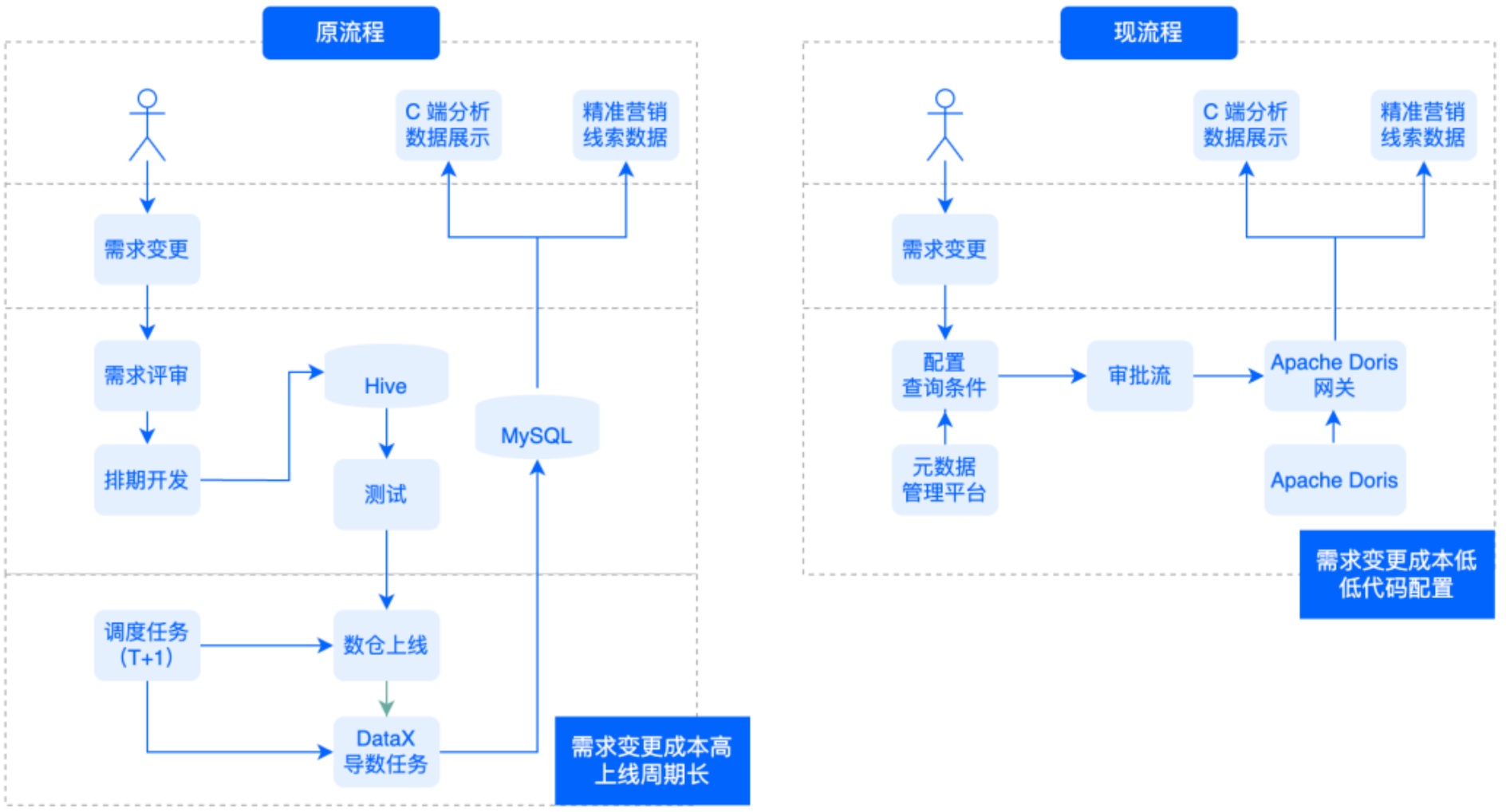
原流程（左）中，业务人员在画像平台页面上利用表的元数据创建人群圈选任务，任务创建后进行人群 ID 分配，写入到 PG 画像表和 MySQL 任务表中。接着根据任务条件定时在 ES 中查询结果，获取结果后更新任务表的状态，并把 Bitmap 人群包写入 PG。利用 PG 插件提供的 Bitmap 交并差能力操作人群包，最后下游运营介质从 PG 取相应人群包。

然而，该流程处理方式非常复杂，ES 和 PG 中的表无法复用，造成成本高、效益低。同时，原流程中的数据为 T+1 更新，标签必须提前进行定义及计算，这非常影响查询效率。

现流程（右）中，业务人员在画像平台创建人群圈选任务，后台分配人群 ID，并将其写入 MySQL 任务表中。首次圈选时，根据任务条件在 Doris 中进行即席查询，获取结果后对任务表状态进行更新，并将人群包写入 Doris。后续根据时间进行微批轮询，利用 Doris Bitmap 函数提供的交并差功能与上一次的人群包做差集，如果有人群包更新会主动通知下游。

引入 Doris 后，原有流程的问题得到了解决，新流程以 Doris 为核心构建了人群圈选服务，支持人群包实时更新，新标签无需提前定义，可通过条件配置自助生成，减少了开发时间。新流程表达方式更加灵活，为人群包 AB 实验提供了便捷的条件。流程中采用 Doris 统一了明细数据和人群包的存储介质，实现业务聚焦，无需处理多组件数据之间的读写问题，达到了降本增效的终极目标。

C 端分析数据及精准营销线索场景

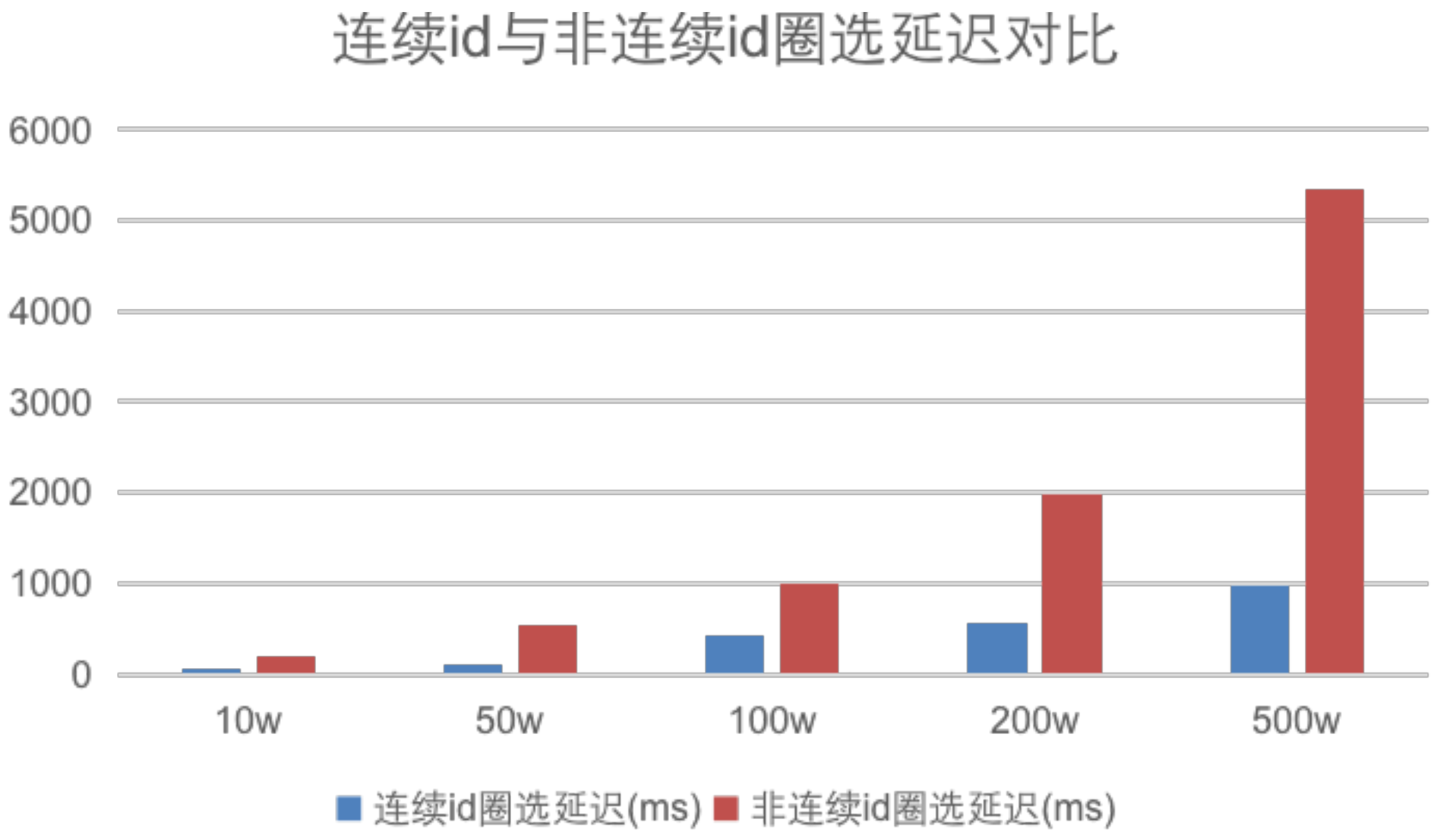


原流程：在原流程中，如果业务提出新需求，需要先发起需求变更，再经过评审、排期开发，然后开始对 Hive 中的数据模型进行开发并进行测试，测试完成后进行数仓上线，配置 T+1 调度任务写入 MySQL，最后 C端和精准营销系统对 MySQL 数据进行读取。原流程链路复杂，主要体现在流程长、成本高、上线周期长。

现流程：当前明细数据已经在 Doris 上线，当业务方发起需求变更时，只需要拉取元数据管理平台元数据信息，配置查询条件，审批完成后即可上线，上线 SQL 可直接在 Doris 中进行即席查询。相比原流程，现在的流程大幅缩短了需求变更流程，只需进行低代码配置，成功降低了开发成本，缩短了上线周期。

优化经验

为了规避风险，许多公司的人群包 `user_id` 是随机生成的，这些 `user_id` 相差很大且是非连续的。然而，使用非连续的 `user_id` 进行人群圈选时，会导致 Bitmap 生成速度较慢。因此，我们生成了映射表，并生成了连续稠密的 `user_id` 。当使用连续 `user_id` 圈选人群时，**速度较之前提升了 70%**。



用户 ID 映射表样例数据：从图可知原始用户 ID 由多位数字组合，并且 ID 很稀疏（用户 ID 间相差很大），而连续用户 ID 则 从1开始，且 ID 很稠密。

tyc_user_id	tyc_user_id_continuous	create_time	update_time
16xxxxxxxxxxxxx324	1	2023-04-21 14:03:21	2023-04-21 18:31:04
16xxxxxxxxxxxxx731	2	2023-04-22 14:03:21	2023-04-22 14:03:21
...			

案例展示：

1、用户ID 映射表：

用户 ID 映射表将用户 ID 作为唯一键模型，而连续用户 ID 则通过用户 ID 来生成，一般从 1 开始，严格保持单调递增。需要注意的是，因为该表使用频繁，因此将 `in_memory` 设置为 `true` 直接将其缓存在内存中：

```
CREATE TABLE `dim_user_comparison` (
  `tyc_user_id` bigint(20) NOT NULL DEFAULT "0" COMMENT '用户ID',
  `tyc_user_id_continuous` bigint(20) NOT NULL DEFAULT "0" COMMENT '连续用户ID',
  `create_time` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP COMMENT '写入doris时间'
  `update_time` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP COMMENT '更新doris时间'
) ENGINE=OLAP
UNIQUE KEY(`tyc_user_id`) COMMENT '用户id映射表'
DISTRIBUTED BY HASH(`tyc_user_id`) BUCKETS 9
PROPERTIES (
"replication_allocation" = "tag.location.default: 3",
"in_memory" = "true",
"storage_format" = "V2",
"enable_unique_key_merge_on_write" = "true",
"disable_auto_compaction" = "false"
);;
```

2、人群包表

人群包表是以用户标签作聚合键的模型，假设以 `user_id` 大于 0、小于 2000000 作为圈选条件，使用原始 user_id 进行圈选耗费的时间远远大于连续稠密 `user_id` 圈选所耗时间。

```
CREATE TABLE `user_tag_bitmap` (
  `tag` bigint(20) NULL COMMENT '用户标签',
  `tyc_user_id` bitmap BITMAP_UNION NULL COMMENT '人群包'
) ENGINE=OLAP
AGGREGATE KEY(`tag`) COMMENT '人群包表'
DISTRIBUTED BY HASH(`tag`) BUCKETS 3
PROPERTIES (
"replication_allocation" = "tag.location.default: 3",
"in_memory" = "false",
"storage_format" = "V2",
"disable_auto_compaction" = "false"
);;
```

如下图所示，左侧使用 `tyc_user_id` 圈选生成人群包响应时间：1843ms，右侧使用使 `tyc_user_id_continuous` 圈选生成人群包响应时间：543ms，消耗时间大幅缩短。

```
insert into
  user_tag_bitmap
select
  101,
  to_bitmap(tyc_user_id)
from
  dim_user_comparison
where
  tyc_user_id > 0
  and tyc_user_id <= 2000000;
```

```
insert into
  user_tag_bitmap
select
  101,
  to_bitmap(tyc_user_id)
from
  dim_user_comparison
where
  tyc_user_id > 0
  and tyc_user_id <= 2000000;
```

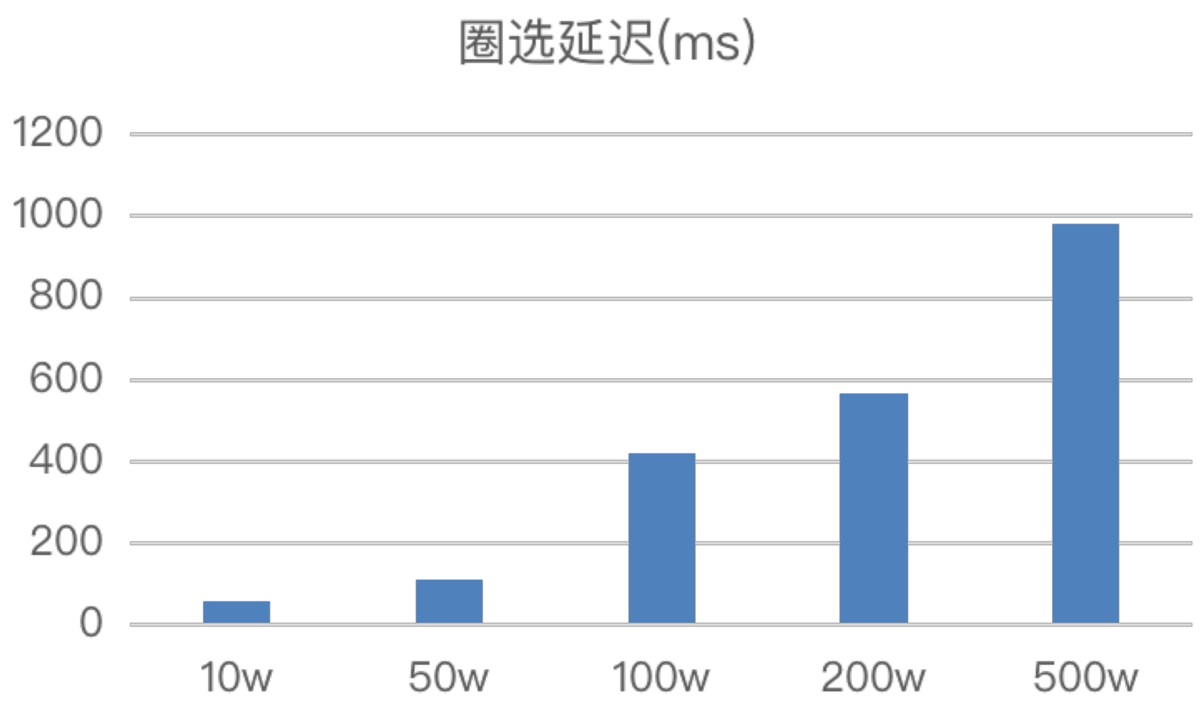
规模与收益

引入 Doris 后，我们已经搭建了 2 个集群，承载的数据规模正随着迁移的推进而持续增大。目前，**我们已经处理的数据总量已经达到了数十 TB，单日新增数据量已经达到了 数十亿条**，而数

据体量还在持续增长中。此外，我们在 Doris 上运行的指标和人群包数量已经超过了 500，分别涵盖了商查、搜索、运营、用户和营收五大类指标。

Doris 的引入满足了业务上的新需求，解决了原有架构的痛点问题，具体表现为以下几点：

- **降本增效：**Doris 统一了数据的门户，实现了存储和计算的统一，提高了数据/表的复用率，降低了资源消耗。同时，新架构优化了数据到 MySQL、ES 的流程，开发效率得到有效提升。
- **导入速率提升：**原有数据流程中，数据处理流程过长，数据的导入速度随着业务体量的增长和数据量的不断上升而急剧下降。引入 Doris 后，我们依赖 Broker Load 优秀的写入能力，使得导入速率提升了 75%以上。
- **响应速度**：Doris 的使用提高了各业务场景中的查询响应速度。例如，在人群圈选场景中，对于 500 万及以下的人群包进行圈选时，能够做到毫秒级响应。



未来规划

正如前文所讲，Apache Doris 的引入解决了许多架构及业务上的难题，初见成效，同时也收获了公司内部数据部门、业务方的一致好评，未来我们将继续探索，基于 Doris 展开更深度的应用，不久的将来，我们将重点推进以下几个方面工作：

- 离线指标实时化：将更多的指标从离线转为实时，提供更及时的数据服务。
- 搭建数据血缘系统：将代码中的血缘关系重新定义为可视，全面构建数据血缘关系，为问题排查、链路报警等提供有效支持。
- 探索批流一体路线：从使用者的角度思考设计，实现语义开发层的统一，使数据开发更便捷、更低门槛、更高效率。

在此特别感谢 SelectDB 团队，作为一家基于 Apache Doris 的商业化公司，为社区投入了大量的研发和用户支持力量，在使用过程中遇到任何问题都能及时响应，为我们降低了许多试错成本。未来，我们也会更积极参与社区贡献及活动中来，与社区共同进步和成长，欢迎大家选择和使用 Doris，相信 Doris 一定不会让你失望。

SelectDB 产品免费试用

扫码关注 SelectDB 微信公众号

获取更多技术趋势、社区动态、用户案例