Apache Doris 用户案例集

2022-2023 精选



DORIS × SELECTDB

SelectDB 出品

卷首语

下一个十年，
我们需要一款什么样的分析型数据库？

作者：Apache Doris PMC Chair 陈明雨，（本文源于 Doris Summit 2022 演讲实录）

十年对于数据库意味着什么？

身处在日新月异的时代，我们见惯了技术的兴起与繁荣、变迁与衰落，甚至是朝荣夕灭。信息技术以前所未有的速度更迭，给周遭事物带来了颠覆性地变化。数据库亦是如此，无数数据库悄然湮没在技术更迭的浪潮里，直到在浩渺如海的代码片段中都找不到些许印记。而有的则历久而弥新，经受了时间的考验，彰显出强大的生命力，并以更加繁茂的姿态扎根生长。

十年对于数据库而言，可能是一段从诞生到消逝的完整软件生命周期，也可能是迈过里程碑之后的全新旅程。

所以从 MySQL 1.0 版本诞生，到具备颠覆性意义的 MySQL 5.7 版本正式发布，时间跨度刚好是十年，而十年之后的故事，大家已经都知道了。

所以从 Benoit、Thierry、Marcin 联合创建 Snowflake，到在纽交所成功上市、成为软件行业有史以来最大规模的 IPO，再到全面开启云数据仓库时代，时间跨度也差不多十年。

而对于 Apache Doris，十年意味着什么？

留个悬念，在回答这个问题之前，我们不妨来回顾下社区发展历程。

尽管最早的历史可以追溯到 2008 年的百度凤巢广告系统，但彼时非 SQL 的单机查询引擎加 KV 存储系统在产品形态上与 OLAP 还有着较大的差异。

正式确立 OLAP 数据库这一形态是在 2013 年。通过自研全列式存储引擎 OLAP Engine 并基于 Apache Impala 改造了全新的 MPP 查询引擎，自此，Doris 真正成为了具备大数据量下高效支持数据分析能力的 OLAP 数据库，并在百度内部大规模应用，成为了百度内部统一的 OLAP 分析平台。

往往一个内部项目的发展会有两种演进模式，一种是随着需求的增加系统架构日益臃肿，当面对较为灵活的需求，常因改动成本过大而被彻底重构。另一种则是长期服务某一固定场景、需求逐渐收敛乃至停滞，最终被快速革新的外部技术彻底取代。而开源则是内部项目的一场新生，在更广阔的应用场景、更多样的开发者群体以及更高效的研发模式加持下开启新的篇章。

于是在数个版本的迭代与优化后，2017 年 Doris 的前身在 GitHub 上开源，2018 年进入 Apache 基金会孵化，并正式更名为 Apache Doris。

时至 2022 年，正是 Apache Doris 在 OLAP 领域深耕的十年之际。

我们该如何回顾过去的 2022 年？

2022 年，外部世界正处在前所未有的变化之中，无数魔幻时刻在现实中发生。需要庆幸的是，技术和开源的力量帮助我们穿越了许多不确定性。而这一年势必成为 Apache Doris 发展历程中有着浓墨重彩的一年，我们从几个角度来回顾一下 Apache Doris 过去一年的发展：

社区重要指标

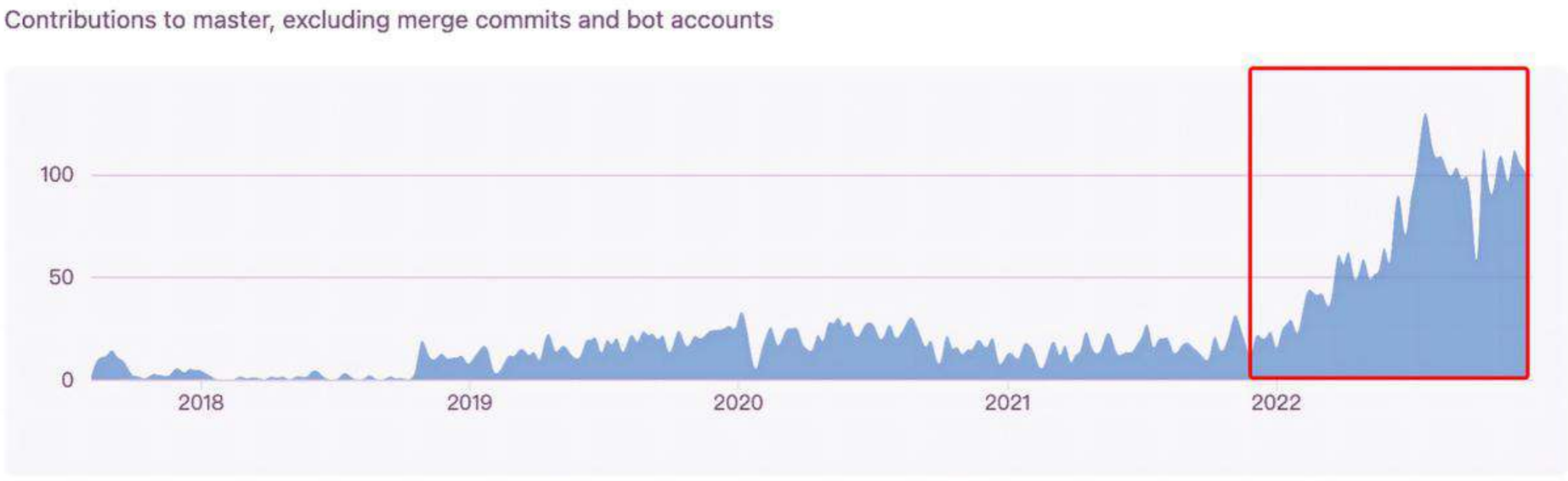
过去一年中：

- 社区累计贡献者的数量从 200 余位增长至近 420 位，同比增长**超过 100%**，目前仍在持续上升中。
- 每月活跃贡献者的数量从 50 位增长至 100 位，同样呈现**翻倍**增长的趋势。
- GitHub Star 数量从 3.6k 增长至 6.8k，多次登上 GitHub Trengding 日/周/月度**榜单前列**。
- 全部 Commits 数量从 3.7k 增长至 7.6k，过去一年新提交代码量超越了以往多年累加总和。

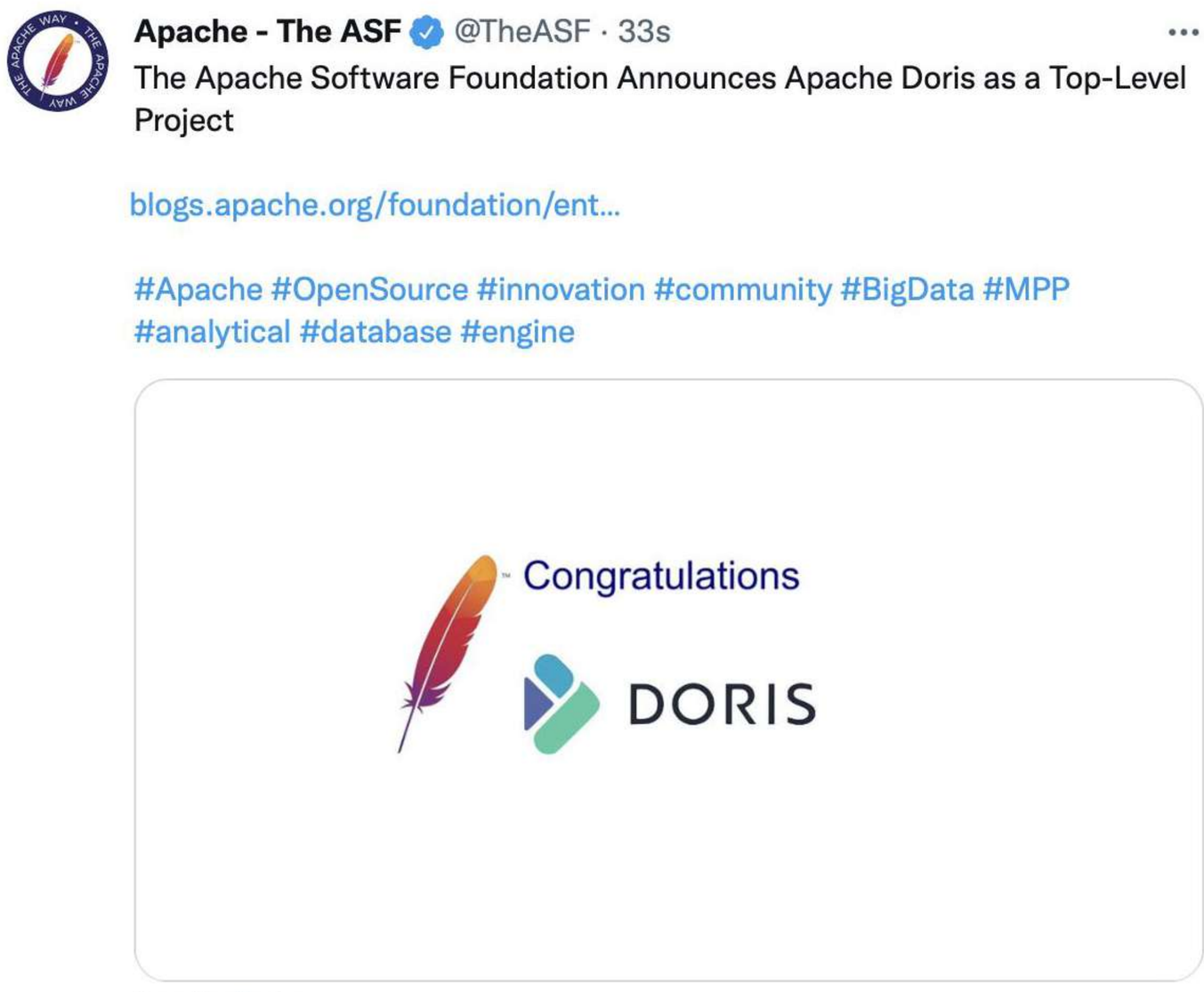
	Contributor	Monthly Active Contributor	Stars	Commits
2021	200+	50+	3600+	3700+
2022	▲100% 400+	▲100% 100+	▲83% 6600+	▲100% 7600+

从这些数据中，我们可以感受到 2022 年是 Apache Doris 全面爆发的一年，各个维度数据指标几乎都有了 100% 的增长。这一年的努力也使 **Apache Doris 成为了全球大数据和数据库领域最为活跃的开源社区之一**，下方 GitHub Contribution 增长趋势图更是证明了这一点。而这一切，正是由社区所有的用户和开发者共同创造的。

Community
Community Development 2022
DORIS SUMMIT



另外值得纪念的是，在 2022 年 6 月， Apache Doris 迎来了开源以来最重要的里程碑之一，正式从 Apache 孵化器毕业、成为了 Apache 顶级项目。



Apache Doris

开源用户规模

得益于社区成立的专职工程师团队，为 Apache Doris 社区用户提供义务的技术支持，2022 年我们在用户连接与沟通方面变得更加顺畅，可以更直面用户、去倾听用户真实的声音。



在过去的一年里，Apache Doris 已经在互联网、金融、电信、教育、汽车、制造、物流、能源、政务等数十个行业应用落地，尤其是在以海量数据著称的互联网行业。在中国市值或估值排行前 50 的互联网公司中，有 80% 企业在长期使用 Apache Doris 来解决自身业务中的数据分析问题，其中包含了百度、美团、小米、腾讯、京东、字节跳动、网易、新浪、360、 米哈游、知乎等头部知名企业。

在全球范围内，Apache Doris 已经得到了超过 1000 家企业用户的认可，并且这一数字仍在快速增长中。这 1000 多家企业用户中，绝大多数与社区有着直接联系，并通过各种方式参与到社区建设中来。他们中的许多企业用户也参与到本次 Doris Summit 的议题分享中，将自身基于真实业务场景的实践经验分享给大家。

版本更新迭代

如果说过去版本将使用和运维的简易性作为第一追求的话，那么 2022 年发布版本则是在性能、稳定性、易用性等多方面特性的全面进化。

- 4 月份社区发布了自开源以来的首个 1 位版本—— Apache Doris 1.0，在 1.0 版本中，意义非凡的向量化执行初次与大家见面，标志着 Apache Doris 开始迈入极速数据分析时代。
- 6 月份发布的 1.1 版本，我们对向量化引擎进行了进一步完善和优化，并将其作为正式功能默认开启。与此同时，社区建立了 LTS 版本发布机制，以每月发布一个 3 位版本的速度，对 1.1 版本进行快速地 Bug 修复和功能优化，力求满足更多社区用户在稳定性方面的高要求。
- 在综合考虑版本迭代节奏和用户需求后，我们决定将众多新特性在 1.2 版本中发布。同时期社区的稳定性和质量保障工作也取得了显著的成效，测试 Case 得到了极大程度地丰富，并在 Master 分支上构建了流水线。通过一系列质量手段，Apache Doris 的代码质量和稳定性得到进一步提升，这也使得版本发布有着更加严格的准出标准。
- 12 月初 1.2 版本正式面世。这一版本的发布不仅使查询性能有了近十倍的提升，同时我们还推出了过去半年时间里研发的诸多重磅功能，包括 Unique Key 模型 Merge-on-Write 的数据更新模式、支持无缝对接多种数据湖的 Multi-Catalog 多源数据目录、Java UDF 、Array 数组类型和 JSONB 类型等，让 Apache Doris 在更多数据分析场景具备了更强的适应性和可能性。
- 我们针对系统稳定性进行了大量的工作，一方面，利用 SQL Smith 等自动化测试工具以及各个知名开源项目的测试用例，构建了数以百万计的测试用例集；另一方面，通过社区准入流水线和完善的回归测试框架，保证了代码合入的质量。因此 1.2 版本不论从功能、性能还是

稳定性方面，都是一次厚积薄发后的全面进化，也是对所有开发者在 2022 年辛苦付出的最好回报。

核心特性演进

核心特性方面，社区的研发力量主要围绕四个方面开展工作，分别是**性能、实时性、半结构化数据支持与 Lakehouse**。

- **查询性能提升。**从 1.0 版本面世到 1.2 版本发布，Apache Doris 在性能方面取得了极为显著的成绩。在单表场景上，Apache Doris 荣登 Clickhouse 公司推出的 Clickbench 数据库性能榜单，并取得了**前三名**的优秀成绩。在多表关联场景上，得益于向量化执行引擎及各种查询优化技术，相对 2021 年底发布的 0.15 版本 ，Apache Doris 在 SSB 和 TPC-H 等标准测试数据集下均**取得了数倍乃至数十倍**的性能提升。这一系列性能方面的优化，已经成功让 Apache Doris 跻身全球数据库性能最优阵列中！



- **实时场景优化。**在 1.2 版本中，我们在原有 Unique Key 数据模型上实现了 Merge-On-Write 的数据更新方式，查询性能在高频更新时有 **5-10 倍的提升**，实现了在可更新数据上的低延迟实时分析体验。另外还实现了轻量 Schema Change 功能，对于数据的加减列不再需要转换历史数据，可通过 Flink CDC 等工具快速便捷地同步上游事务数据库中的 DML 或 DDL 操作，使数据同步工作能够更加流畅统一。

- **半结构化数据支持。**目前 Apache Doris 支持了 Array 和 JSONB 类型，其中 Array 类型不仅能更方便地存储复杂的数据结构，还可以通过 Array 函数满足用户行为分析等场景的业务需求。而 JSONB 是一种二进制 JSON 存储方式，它不但比纯文本 Text JSON 的访问性能快 4 倍，同时也有更低的内存消耗。通过 JSONB 可以方便地导入各种 JSON 格式的日志数据结构，并能取得优异的查询效率。这也是 Apache Doris 在日志分析领域所做的探索之一。

Semi-Structure

- **Array**
  User Behavior Analysis
  Functional Programming, Aggregation with Array

```
mysql> select *,array_difference(k2) from array_type_table;
+------+---------------------------------+---------------------------------+
| k1   | k2                              | array_difference(`k2`)          |
+------+---------------------------------+---------------------------------+
|    0 | []                              | []                              |
|    1 | [NULL]                          | [NULL]                          |
|    2 | [1, 2, 3]                       | [0, 1, 1]                       |
|    3 | [1, NULL, 3]                    | [0, NULL, NULL]                 |
|    4 | [0, 1, 2, 3, NULL, 4, 6]        | [0, 1, 1, 1, NULL, NULL, 2]     |
|    5 | [1, 2, 3, 4, 5, 4, 3, 2, 1]     | [0, 1, 1, 1, 1, -1, -1, -1, -1] |
|    6 | [6, 7, 8]                       | [0, 1, 1]                       |
+------+---------------------------------+---------------------------------+
```

- **JSONB**
  Efficient Binary JSON Format
  4 Times Faster than Text JSON
  30% Lower in Memory Consumption

```
mysql> SELECT id, j, jsonb_extract_isnull(j, '$') x FROM test_jsonb ORDER BY id;
+------+-----------------------------------------------------------------------+------+
| id   | j                                                                     |    x |
+------+-----------------------------------------------------------------------+------+
|    1 |                                                                    {} |    0 |
|    2 |                                                {"k1":"v31","k2":300} |    0 |
|    3 |                                                                    [] |    0 |
|    4 |                                                             [123,456] |    0 |
|    5 |                                                          ["abc","def"] |    0 |
|    6 |                                  [null,true,false,100,6.18,"abc"] |    0 |
|    7 |                                [{"k1":"v41","k2":400},1,"a",3.14] |    0 |
|    8 | {"k1":"v31","k2":300,"a1":[{"k1":"v41","k2":400},1,"a",3.14]} |    0 |
+------+-----------------------------------------------------------------------+------+
```

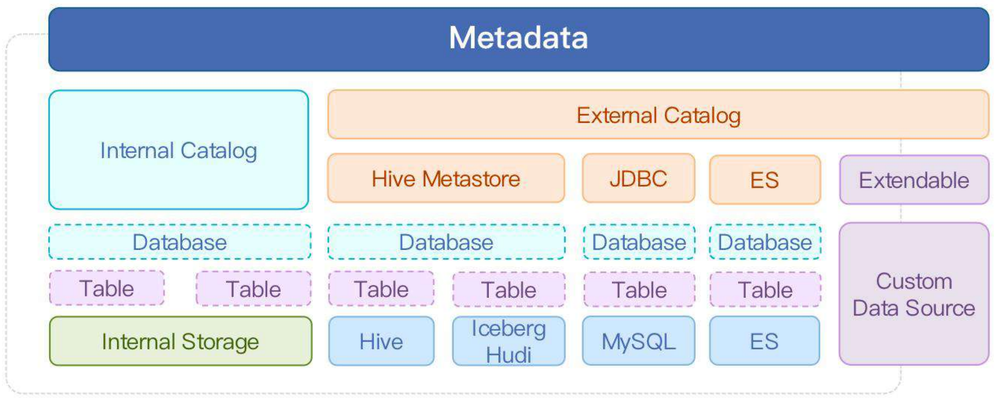
- **Lakehouse。**在最新发布的 1. 2 版本中，我们引入了全新的 Catalog 概念，正式将 Apache Doris 迈入湖仓一体时代。通过简单的命令便可以方便地连接到各自外部数据源并自动同步元数据，实现统一的分析体验。通过 Native Format Reader、延迟物化、异步 IO、数据预取等多项针对外部数据源的性能优化，并充分利用自身的高性能执行引擎和查询优化器，在对外表访问性能上，Apache Doris 可以达到 Trino/Presto 的 **3– 5 倍**、Hive 的 **10–100 倍。**

Lakehouse

- **Easy Access**
  Auto Schema Mapping & Sync

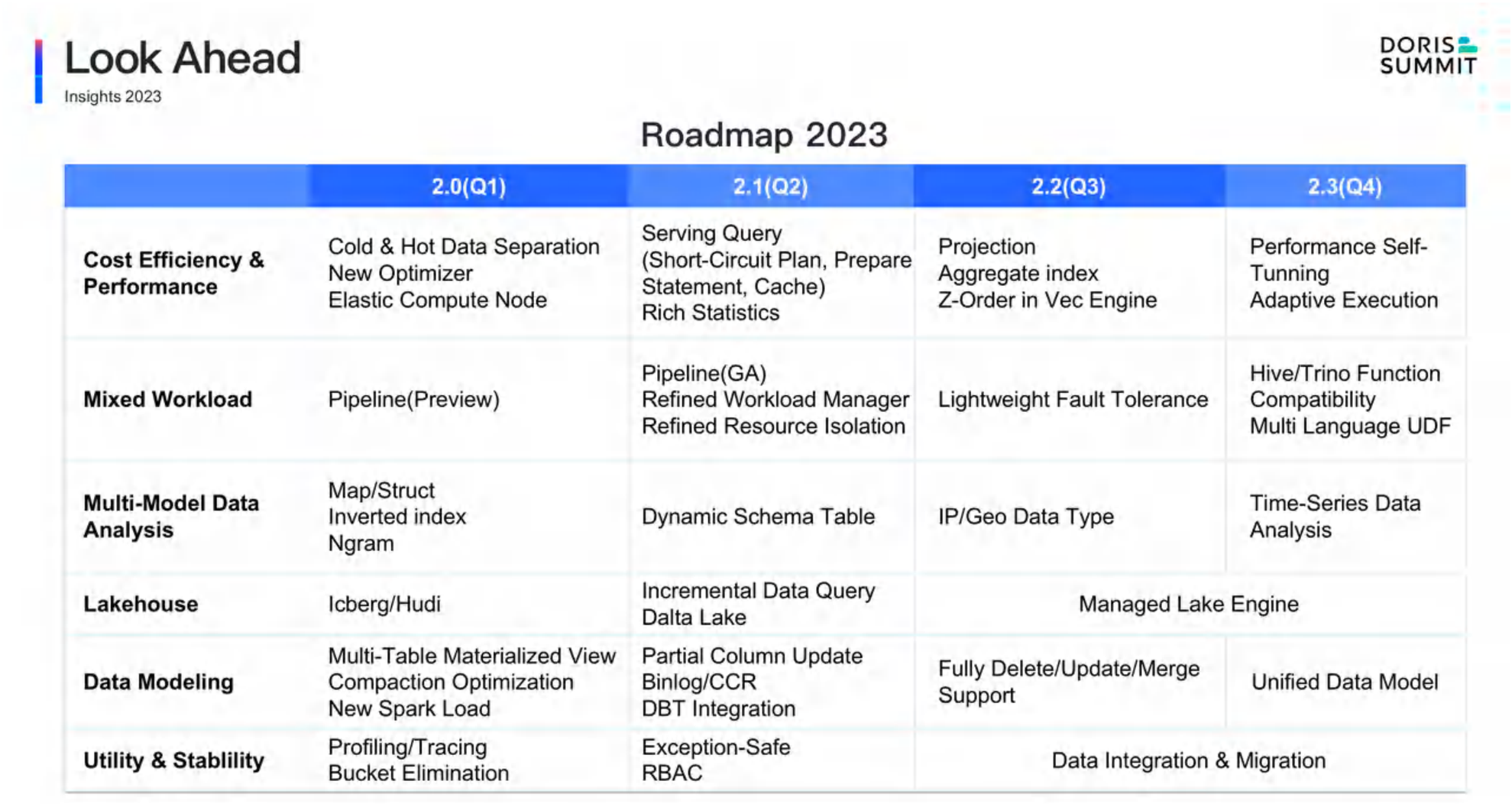
```
CREATE CATALOG hive PROPERTIES (
  "type" = "hms",
  "hive.metastore.uris"
  =
  "thrift://172.0.0.1:8000",
);
```

- **High Performance**
  Vectorization Engine & CBO Optimizer
  Native Parquet/ORC Reader
  3-5 Times Faster than Trino
  10-100 Times Faster than Hive



2023 RoadMap

承前而启后，2023 年，Apache Doris 社区在以上几方面特性持续完善的同时，也将开启更多有意义的工作。全年的 RoadMap 以及明年 Q1 的具体计划，可以参考以下的全景图：



	2.0(Q1)	2.1(Q2)	2.2(Q3)	2.3(Q4)
Cost Efficiency & Performance	Cold & Hot Data Separation New Optimizer Elastic Compute Node	Serving Query (Short-Circuit Plan, Prepare Statement, Cache) Rich Statistics	Projection Aggregate index Z-Order in Vec Engine	Performance Self-Tunning Adaptive Execution
Mixed Workload	Pipeline(Preview)	Pipeline(GA) Refined Workload Manager Refined Resource Isolation	Lightweight Fault Tolerance	Hive/Trino Function Compatibility Multi Language UDF
Multi-Model Data Analysis	Map/Struct Inverted index Ngram	Dynamic Schema Table	IP/Geo Data Type	Time-Series Data Analysis
Lakehouse	Icberg/Hudi	Incremental Data Query Dalta Lake	Managed Lake Engine	
Data Modeling	Multi-Table Materialized View Compaction Optimization New Spark Load	Partial Column Update Binlog/CCR DBT Integration	Fully Delete/Update/Merge Support	Unified Data Model
Utility & Stablility	Profiling/Tracing Bucket Elimination	Exception-Safe RBAC	Data Integration & Migration	

稳定的版本发布和迭代速度对于开源软件至关重要。在 2023 年，我们将以每季度一个 2 位版本的节奏，开始 Apache Doris 2.x 版本的迭代。同时，针对每个 2 位版本，我们也将以每月一个 3 位版本的速度进行功能维护和优化。

从功能角度来看，后续研发工作将会围绕以下几个主要方向展开：

高性能

高性能是 Apache Doris 不断追求的目标，过去一年在 Clickbench、TPC-H 等公开测试数据集上的优异表现，已经证明了其在执行层以及算子优化方面做到了业界领先。未来我们也会不断优化各个场景下的性能表现，回馈用户极速的数据分析体验，具体包括：

- **更复杂 SQL 性能提升：**2022 年已启动全新查询优化器的设计与开发，而这一成果在 2023 年一季度将与大家见面。全新查询优化器提供了丰富的规则模型，实现更智能的代价选择，可更高效支撑复杂查询，能完整执行 TPC-DS 全部 99 个 SQL。全新查询优化器还具备全查询场景的自适应优化，便于用户面对不同分析负载和业务场景获得一致性使用体验。

- **更高的点查询并发：**高并发一直是 Apache Doris 所擅长的场景，而 2023 年我们将会进一步加强这一能力，通过 Short-Circuit Plan、Prepare Statement、Query Cache 等一系列技术，实现单机数万 QPS 的超高并发支持，并具备随集群规模的拓展进而线性提升并发的能力。

- **更灵活的多表物化视图：**在过去版本中，通过强一致的单表物化视图，Apache Doris 加速了固定维度数据的分析效率。而全新的多表物化视图将会解耦 Base 表与 MV 表的生命周期，通过异步刷新和灵活的增量计算方式，满足多表关联以及更复杂 SQL 的预计算加速需求，这一特性将在接下来的 2023 年第一季度与大家见面！

高性价比

成本和效率对企业而言是赢得市场竞争的关键，对数据库而言亦是如此。过去 Apache Doris 凭借在易用性方面的诸多设计帮助用户大幅节约了计算与存储资源成本，后续我们也会引入一系列云原生能力，在不影响业务效率的同时进一步降低成本，具体包括：

- **更低的存储成本：**我们将探索与云上对象存储系统和文件系统的结合，帮助用户进一步降低存储成本，包括更完善的冷热数据分离能力，将冷数据智能转移至更廉价的对象存储或文件系统中。结合单一远程副本、冷数据 Cache 以及冷热智能转换等技术，保证业务查询效率不受影响的同时实现存储成本大幅降低，这一功能将于 2023 年第一季度发布。

- **更弹性的计算资源：**剥离存储与计算状态，引入仅用于计算的 Elastic Compute Node 。由于不存储数据，弹性计算节点具备更加快速的弹性伸缩能力，便于用户在业务高峰期进行快速扩容，进一步提升在海量数据计算场景（如数据湖分析）的分析效率，这一功能已经处于最终调试阶段，即将与大家见面。后续我们还将通过对集群内存和 CPU 运行指标的监控和自动策略配置，实现自动的节点扩缩容（Auto-scaling）。

混合负载

随着用户规模的极速扩张，越来越多的用户将 Apache Doris 用于构建企业内部的统一分析平台。这一方面需要 Apache Doris 去承担更大规模的数据处理和分析，另一方面也需要 Apache Doris 同时去应对更多分析负载的挑战，从过去的实时报表和 Ad-hoc 等典型 OLAP 场景，扩展到 ELT/ETL 、日志检索与分析等更多场景的统一。为了能更好适配这些场景，许多工作已经进入紧锣密鼓的研发中，并将于 2023 年陆续与大家见面，具体包括：

- **更灵活的 Pipeline 执行引擎：**与传统的火山模型相比，Pipeline 模型无需手动设置并发度，可以实现不同管道之间的并行计算，充分利用多核的计算能力，实现更灵活的执行调度，提升在混合负载场景下的综合性能表现。

- **Workload Manager：**在性能提升的同时，也亟需完善的资源隔离和划分的能力。我们将会基于 Pipeline 执行引擎实现更细粒度和更灵活的负载管理、资源队列以及共享隔离等功能，兼顾多种混合负载场景下的查询性能与稳定性。

- **轻量级容错：**轻量级容错能力也是我们后续持续完善的地方，既能利用 MPP 的高效率又能对错误进行容忍，以更好适应用户在 ETL/ELT 场景的挑战。

- **函数兼容与多语言 UDF：**与此同时，后续也将支持 Hive/Trino/Spark 函数的兼容性以及多语言的 UDF，来帮助用户更灵活地进行数据加工，也可以更方便地从其他数据库系统迁移到 Apache Doris。

多模数据分析

在过去 Apache Doris 更多是是擅长于结构化数据分析，随着对半结构化、非结构化数据分析需求的增加，从 1.2 版本起我们增加了 Array 和 JSONB 类型以实现数据的 Native 支持，后续版本仍将持续加强这一能力，为日志分析场景提供性价比更高、性能更强的解决方案，具体包括：

- **更丰富的复杂数据类型：**除 Array/JSONB 类型以外，2023 年第一季度我们将增加对 Map/Struct 类型的支持，包括高效写入、存储、分析函数以及类型之间的相互嵌套，以更好满足多模态数据分析的支持。后续将支持更加丰富的数据类型，包括 IP、GEO 地理信息等数据类型，并会探索在时序数据场景的高效数据分析。

- **更高效的文本分析算法：**对于文本数据，我们将引入更多的文本分析算法，包括自适应 Like、高性能子串匹配、高性能正则匹配，Like 语句的谓词下推、Ngram Bloomfilter 等，同时基于倒排索引实现全文检索能力，在日志分析场景提供比 ES 更高性能和性价比的分析能力。这些功能都已经处于就绪阶段，将在 2023 年初与大家见面。

- **动态 Schema 表：**传统数据库在设计之初 Schema 是静态的，Schema 变更时需要执行 DDL ，而这一操作往往具有阻塞性。在越来越多的现代数据分析场景中，表结构会随时间推移而变化，因此我们引入了 Dynamic Table，可以根据数据写入自动适应 Schema ，不再需要执行 DDL，由过去的人工干预数据结构进化为数据自驱动，极大提升了灵活数据分析的便捷性。这一功能将在 2022 年第一季度正式发布。

Lakehouse

随着数据湖技术的发展，分析性能成为发挥数据湖效用、挖掘数据价值最大的掣肘。基于一款简单易用和高性能的查询分析引擎在数据湖之上构建分析服务，成为新的技术趋势。在过去一年，通过在数据湖上的诸多性能优化、结合自身的高性能执行引擎和查询优化器以及，Apache Doris 实现了数据湖上极速易用的分析体验，性能较 Presto/Trino 有 3-5 倍的提升。在 2023 年，我们将会继续完善这一能力，具体包括：

- **更简易的数据对接：**在 1.2 版本中我们发布了 Multi-Catalog，支持了多种异构数据源的元数据自动映射与同步，实现了数据湖的无缝对接，后续将对 Delta Lake 的支持以及 Iceberg、Hudi 等更多数据格式的支持。

- **更完整的数据湖能力支持：**提供数据湖上数据的增量更新与查询，还会支持将分析结果写回数据湖、外表写入内表，实现数据分析流程的全闭环。同时还将支持多版本 Snapshot 读取和删除，并进一步在 Apache Doris 为数据湖数据提供物化视图。

实时性与存储引擎优化

数据价值会随着时间推移而降低，因此实时性对于高时效性要求的用户而言至关重要。在 1.1 版本中我们在 Compaction 和 Flink 实时写入方面进行了诸多优化，同时 1.2 版本的 Merge-on-Write 数据更新模式进一步使 Apache Doris 在实时更新与极速查询得以统一。2023 年我们将会持续强化对存储引擎的优化，具体包括：

- **更稳定的数据写入：**通过一系列 Compaction 操作和批量数据写入方面的优化，节省资源开销，降低写放大问题，并结合全新的内存管理框架提升写入过程的内存稳定性，进而提升系统稳定性。

- **更完善的数据更新支持：**过去部分列更新是通过 Agg 模型上的 Replace_if_not_null 来实现的，后续我们将会增加 Unique Key 模型上的部分列更新支持，并完整实现 Delete、Update、 Merge 等数据更新的操作。

- **更统一的数据模型：**当前 Apache Doris 的三种数据模型在各个场景均有丰富的应用，后续我们将尝试统一现有几种数据模型，使用户在使用体验上更加统一。

易用性和稳定性

除了功能方面的丰富与完善，更简单、更易用、更稳定同样也是 Apache Doris 一直追求的目标，2023 年我们将在以下几方面出发，让用户具有更简易和放心的使用体验：

- **简化建表：** 目前 Apache Doris 在建表时分区已经支持了时间函数，后续我们将进一步消除 Bucket 设置，帮助用户最大程度简化建表建模。
- **安全性：** 目前已经实现基于 RBAC 模型的权限管理机制，使用户权限更安全可靠；并对 ID-federation、行列级别权限，数据脱敏等进行了优化，后续将进一步完善。
- **可观测性：** Profile 是定位查询性能问题的重要手段，后续我们将加强对 Profile 的监控并提供可视化 Profile 工具，帮助用户更快定位问题。
- **更好的 BI 兼容性和更完善的数据集成迁移方案：** 当前各 BI 工具可以通过 MySQL 协议连接到 Apache Doris，后续我们将对主流 BI 软件进一步适配，保证更佳的查询体验。随着 DBT、Airbyte 等新兴数据集成和迁移工具的兴起，越来越多用户使用此类系统将数据同步至 Apache Doris ，后续我们也会提供对此些系统的官方支持。

开启下一个十年！

或许有读者或听众还记得我在开头提的问题，对于 Apache Doris，十年意味着什么?

有两层含义，上一个十年和下一个十年。

上一个十年，是 Apache Doris 起源的十年。从诞生到开源、从默默无闻到被越来越多人熟知和使用，开源赋予了 Apache Doris 更加旺盛的生命力和创造力。

而下一个十年，则是一场新的旅程。

正如我在本次 Doris Summit 分享的主题，New Journey of Apache Doris。如果说过去 Apache Doris 更多是服务于在线报表场景和 Ad-hoc 分析的 OLAP 引擎的话，那么在所有社区和开发者的努力下，当前 Apache Doris 已经具备了更为广阔的定位，即极速、易用、实时、统一的多模分析型数据库。

这其中的统一，既包含了架构的统一、也包含了业务和数据的统一。用户可以通过 Apache Doris 构建多种不同场景的数据分析服务、同时支撑在线与离线的业务负载、高吞吐的交互式分析与高并发的点查询；通过一套架构实现湖和仓的统一、在数据湖和多种异构存储之上提供无缝

且极速的分析服务；也可通过对日志/文本等半结构化乃至非结构化的多模数据进行统一管理和分析、来满足更多样化数据分析的需求。

这是我们希望 Apache Doris 能够带给用户的价值，**不再让用户在多套系统之间权衡，仅通过一个系统解决绝大部分问题，降低复杂技术栈带来的开发、运维和使用成本，最大化提升生产力。**

“我们已经出发了太久，以至于忘记了为什么出发。”

希望通过这一定位的转变迎接下一个十年的挑战，或许技术趋势会有变化，架构将会革新，但我们解决用户数据分析问题的初衷不会改变。

希望继续带着上一个十年出发的初心，开启下一个十年的旅程。

目录

01

Apache Doris 在 360商业化的统一 OLAP 应用实践 01

02

万亿数据秒级响应，Apache Doris 在 360数科实时数仓中的应用 12

03

千万数据计算时间缩短 40 倍，橙联基于 Apache Doris 的数仓架构建设实践 28

04

叮咚买菜基于 Apache Doris 统一 OLAP 引擎的应用实践 41

05

复杂查询响应速度提升 10+ 倍，度言软件基于 Apache Doris 实时数仓建设实践 55

06

赋能直播行业精细化运营，斗鱼基于 Apache Doris 的应用实践 64

07

从 Clickhouse 到 Apache Doris，慧策电商 SaaS 高并发数据服务的改造实践 73

08

杭银消金基于 Apache Doris 的统一数据查询网关改造 83

09

并发提升 10 倍，**领健**从 ClickHouse 和 Kudu 到 Apache Doris 数仓升级实践 95

10

助力传统行业数字化革新，Apache Doris 在**美联物业**的数据仓库应用实践 107

11

查询提速 20 倍，Apache Doris 在 **Moka** BI SaaS 服务场景下的应用实践 120

12

从 ClickHouse 到 Apache Doris，**腾讯音乐**内容库数据平台架构演进实践 129

13

Apache Doris 在**小米**亿级用户行为分析平台的实践 145

14

查询性能显著提升，Apache Doris 向量化版本在**小米** A/B 实验场景的调优实践 156

15

小米 A/B 实验场景基于 Apache Doris 的查询提速优化实践 171

16

人群圈选效率提升 30 倍，**云积互动**基于 Apache Doris 构建统一数仓的实践 189

17

一文揭秘**字节跳动**基于 Apache Doris 的实时数仓探索 197

18

打破数据孤岛，Apache Doris 助力**纵腾集团**快速构建流批一体数仓架构 210

Apache Doris

在 360商业化 的统一 OLAP 应用实践

作者：360商业化数据团队 ， 窦和雨、王新新

导读：360商业化为助力业务团队更好推进商业化增长，实时数仓共经历了三种模式的演进，分别是 Storm + Druid + MySQL 模式、Flink + Druid + TIDB 的模式 以及 Flink + Doris 的模式，基于 Apache Doris 的新一代架构的成功落地使得 360商业化团队完成了实时数仓在 OLAP 引擎上的统一，成功实现广泛实时场景下的秒级查询响应。本文将为大家进行详细介绍演进过程以及新一代实时数仓在广告业务场景中的具体落地实践。

360公司致力于成为互联网和安全服务提供商，是互联网免费安全的倡导者，先后推出 360安全卫士、360手机卫士、360安全浏览器等安全产品以及 360导航、360搜索等用户产品。

360商业化依托 360产品庞大的用户覆盖能力和超强的用户粘性，通过专业数据处理和算法实现广告精准投放，助力数十万中小企业和 KA 企业实现价值增长。360商业化数据团队主要是对整个广告投放链路中所产生的数据进行计算处理，为产品运营团队提供策略调整的分析数据，为算法团队提供模型训练的优化数据，为广告主提供广告投放的效果数据。

业务场景

在正式介绍 Apache Doris 在 360 商业化的应用之前，我们先对广告业务中的典型使用场景进行简要介绍：

- **实时大盘：**实时大盘场景是我们对外呈现数据的关键载体，需要从多个维度监控商业化大盘的指标情况，包括流量指标、消费指标、转化指标和变现指标，因此其对数据的准确性要求非常高（保证数据不丢不重），同时对数据的时效性和稳定性要求也很高，要求实现秒级延迟、分钟级数据恢复。

- **广告账户的实时消费数据场景：**通过监控账户粒度下的多维度指标数据，及时发现账户的消费变化，便于产品团队根据实时消费情况推动运营团队对账户预算进行调整。在该场景下数据一旦出现问题，就可能导致账户预算的错误调整，从而影响广告的投放，这对公司和广告主将造成不可估量的损失，因此在该场景中，同样对数据准确性有很高的要求。目前在该场景下遇到的困难是如何在数据量比较大、查询交叉的粒度比较细的前提下，实现秒级别查询响应。

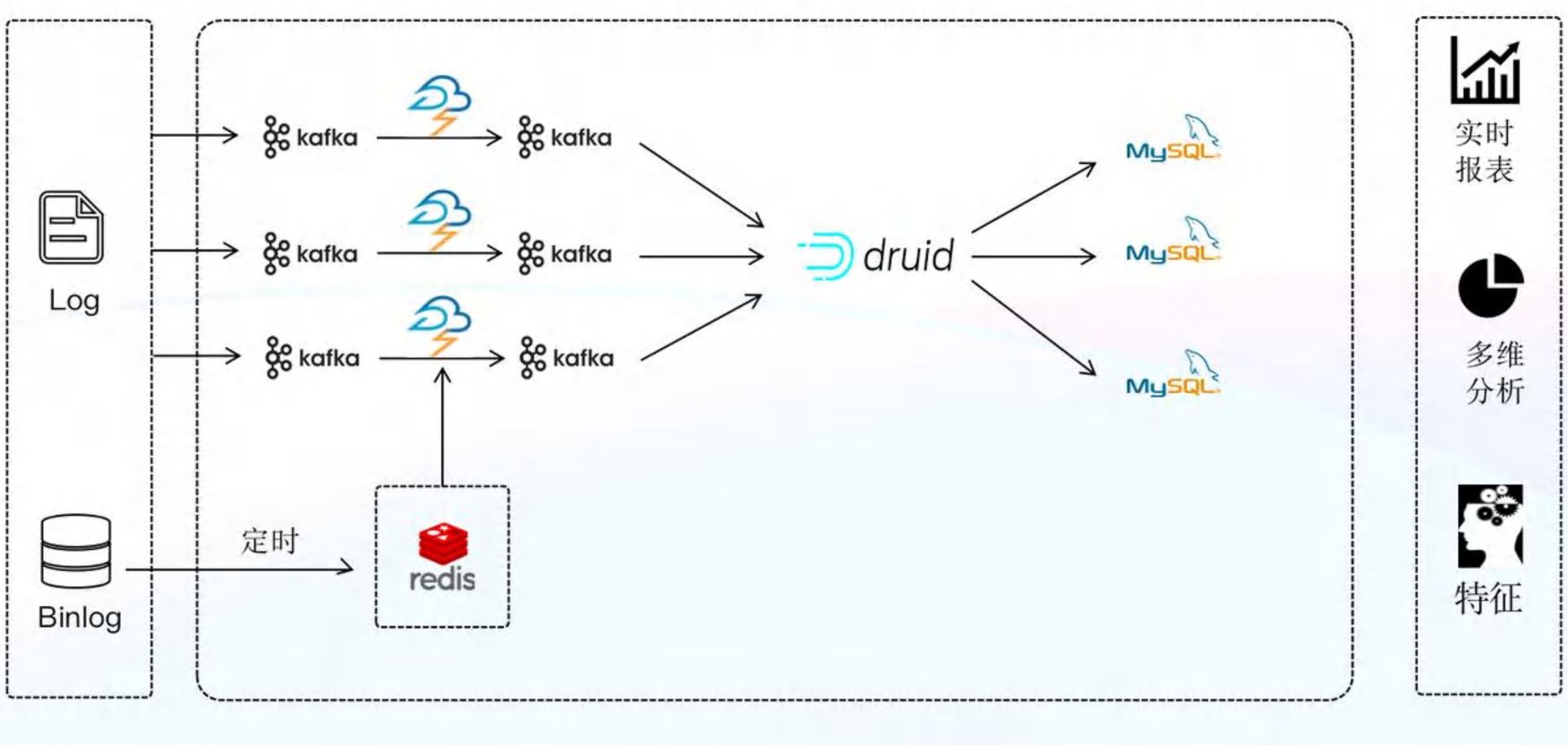
- **AB 实验平台：**在广告业务中，算法和策略同学会针对不同的场景进行实验，在该场景下，具有报表维度不固定、多种维度灵活组合、数据分析比较复杂、数据量较大等特点，这就需要可以在百万级 QPS 下保证数据写入存储引擎的性能，因此我们需要针对业务场景进行特定的模型设计和处理上的优化，提高实时数据处理的性能以及数据查询分析的效率，只有这样才能满足算法和策略同学对实验报表的查询分析需求。

实时数仓演进

为提升各场景下数据服务的效率，助力相关业务团队更好推进商业化增长，截至目前实时数仓共经历了三种模式的演进，分别是 Storm + Druid + MySQL 模式、Flink + Druid + TIDB 的模式以及 Flink + Doris 的模式，**本文将为大家进行详细介绍实时数仓演进过程以及新一代实时数仓在广告业务场景中的具体落地。**

第一代架构

该阶段的实时数仓是基于 Storm + Druid + MySQL 来构建的，Storm 为实时处理引擎，数据经 Storm 处理后，将数据写入 Druid ，利用 Druid 的预聚合能力对写入数据进行聚合。

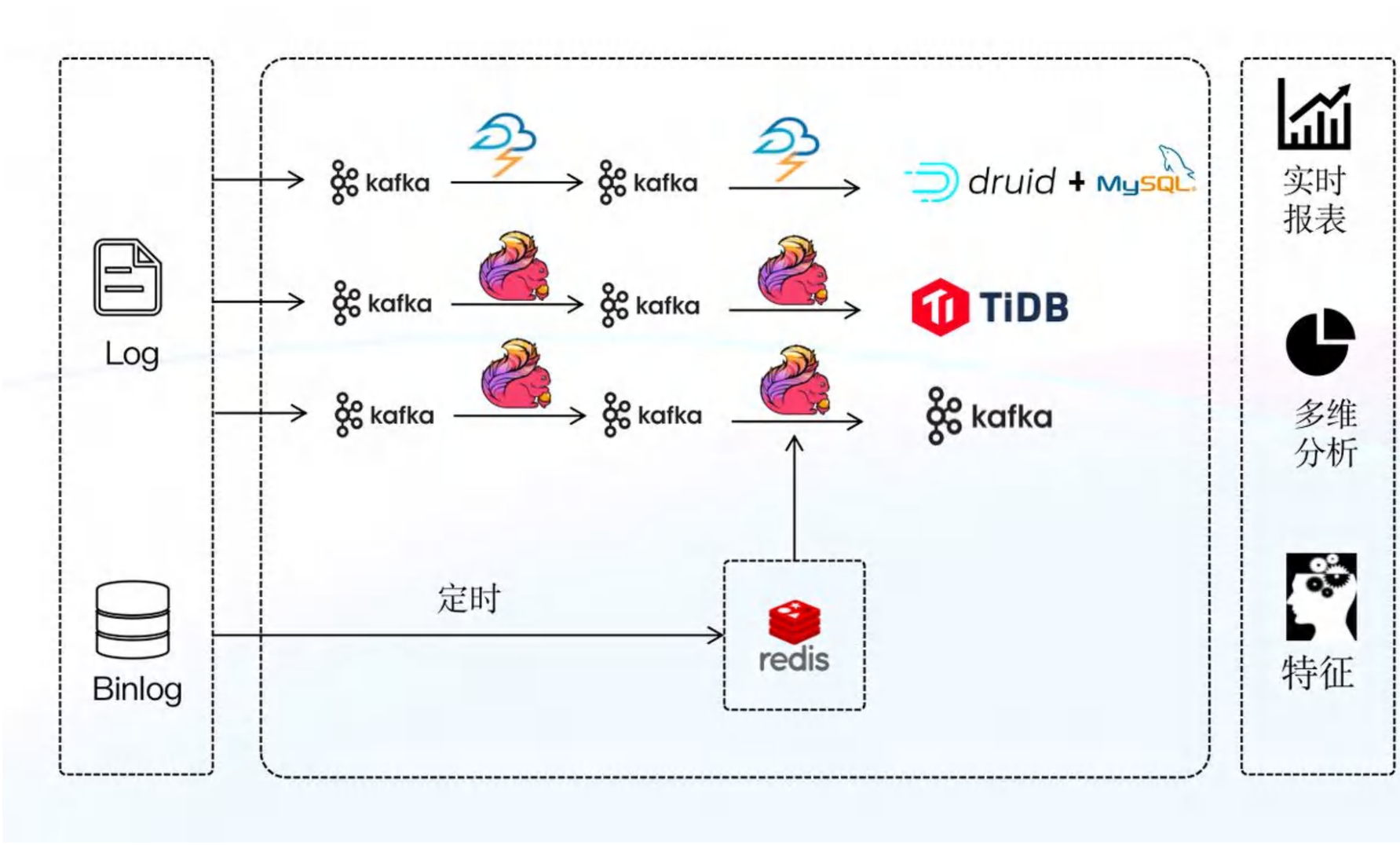


架构痛点：

最初我们试图依靠该架构解决业务上所有的实时问题，经由 Druid 统一对外提供数据查询服务，但是在实际的落地过程中我们发现 Druid 是无法满足某些分页查询和 Join 场景的，为解决该问题，我们只能利用 MySQL 定时任务的方式将数据定时从 Druid 写入 MySQL 中（类似于将 MySQL 作为 Druid 的物化视图），再通过 Druid + MySQL 的模式对外提供服务。通过这种方式暂时可以满足某些场景需求，但随着业务规模的逐步扩大，当面对更大规模数据下的查询分析需求时，该架构已难以为继，架构的缺陷也越发明显：

- 面对数据量的持续增长，数据仓库压力空前剧增，已无法满足实时数据的时效性要求。
- MySQL 的分库分表维护难度高、投入成本大，且 MySQL 表之间的数据一致性无法保障。

第二代架构



基于第一套架构存在的问题，我们进行了首次升级，这次升级的主要变化是将 Storm 替换成新的实时数据处理引擎 Flink ，Flink 相较于 Storm 不仅在许多语义和功能上进行了扩展，还对数据的一致性做了保证，这些特性使得报表的时效性大幅提升；其次我们使用 TiDB 替换了 MySQL ，利用 TIDB 分布式的特性，一定程度上解决了 MySQL 分库分表难以维护的问题（TiDB 在一定程度上比 MySQL 能够承载更大数据量，可以拆分更少表）。在升级完成后，我们按照不同业务场景的需求，将 Flink 处理完的数据分别写入 Druid 和 TiDB ，由 Druid 和 TIDB 对外提供数据查询服务。

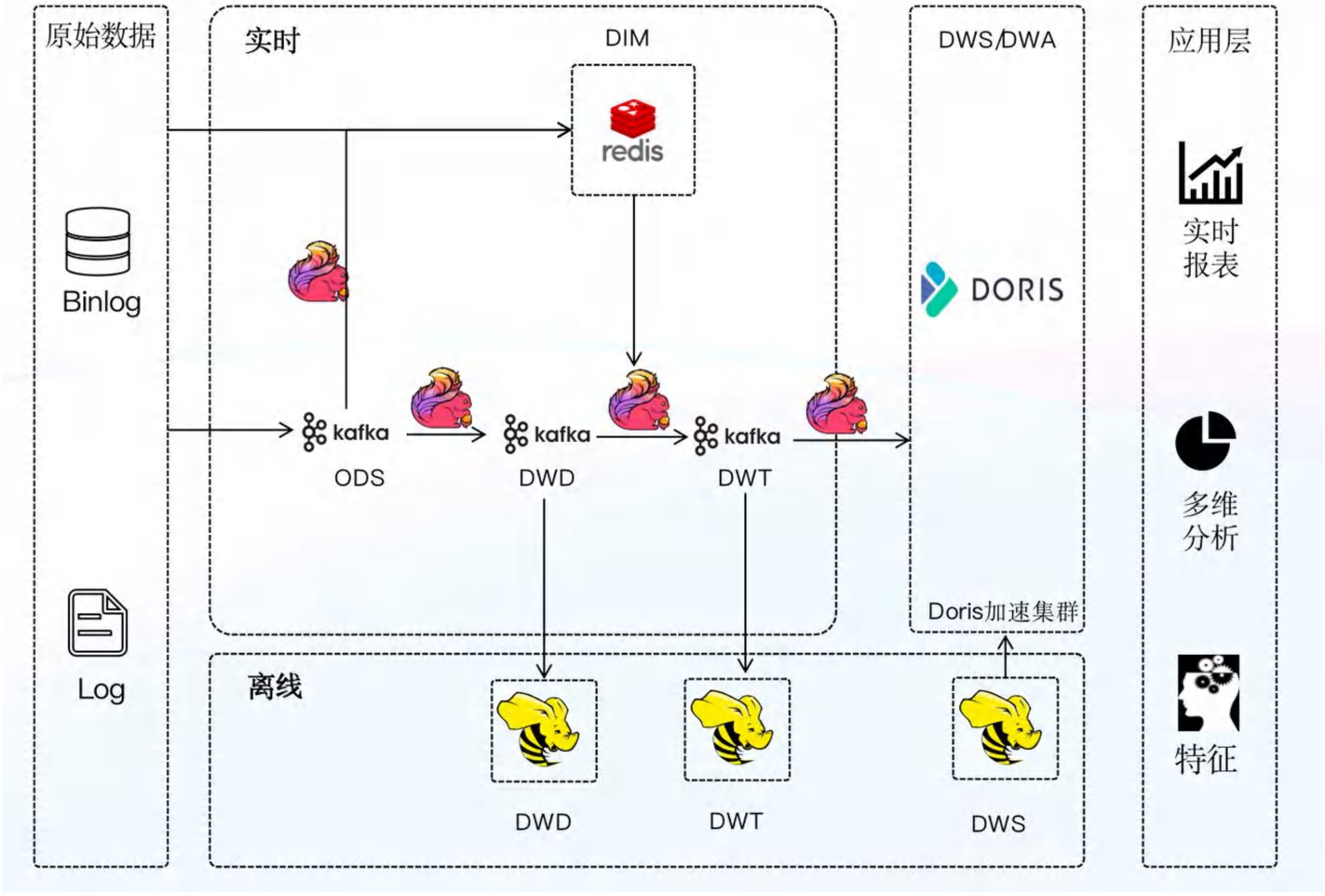
架构痛点：

虽然该阶段的实时数仓架构有效提升了数据的时效性、降低了 MySQL 分库分表维护的难度，但在一段时间的使用之后又暴露出了新的问题，也迫使我们进行了第二次升级：

- Flink + TIDB 无法实现端到端的一致性，原因是当其面对大规模的数据时，开启事务将对 TiDB 写入性能造成很大的影响，该场景下 TiDB 的事务形同虚设，心有余而力不足。
- Druid 不支持标准 SQL ，使用有一定的门槛，相关团队使用数据时十分不便，这也直接导致了工作效率的下降。
- 维护成本较高，需要维护两套引擎和两套查询逻辑，极大增加了维护和开发成本的投入。

新一代实时数仓架构

第二次升级我们引入 Apache Doris 结合 Flink 构建了新一代实时数仓架构，借鉴离线数仓分层理念对实时数仓进行分层构建，**并统一 Apache Doris 作为数仓 OLAP 引擎，由 Doris 统一对外提供服务。**



我们的数据主要源自于维表物料数据和业务打点日志。维表物料数据会定时全量同步到 Redis 或者 Aerospike （类似于 Redis 的 KV 存储）中，通过 Binlog 变更进行增量同步。业务数据由各个团队将日志收集到 Kafka，内部称为 ODS 原始数据（ODS 原始数据不做任何处理），我们对 ODS 层的数据进行归一化处理，包括字段命名、字段类型等，并对一些无效字段进行删减，并根据业务场景拆分生成 DWD 层数据，DWD 层的数据通过业务逻辑加工以及关联 Redis 中维表数据或者多流 Join，最后生成面向具体业务的大宽表（即 DWT 层数据），我们将 DWT 层数据经过聚合、经由 Stream Load 写入 Doris 中，由 Doris 对外提供数据查询服务。在离线数仓部分，同样也有一些场景需要每日将加工完的 DWS 数据经由 Broker Load 写入到 Doris 集群中，并利用 Doris 进行查询加速，以提升我们对外提供服务的效率。

选择 Doris 的原因

基于 Apache Doris 高性能、极简易用、实时统一等诸多特性，助力 360商业化成功构建了新一代实时数仓架构，本次升级不仅提升了实时数据的复用性、实现了 OLAP 引擎的统一，而且满足了各大业务场景严苛的数据查询分析需求，使得整体实时数据流程架构变得简单，大大降低了其维护和使用的成本。**我们选择 Doris 作为统一 OLAP 引擎的重要原因大致可归结为以下几点：**

- **物化视图：**Doris 的物化视图与广告业务场景的特点契合度非常高，比如广告业务中大部分报表的查询维度相对比较固定，利用物化视图的特性可以提升查询的效率，同时 Doris 可以保证物化视图和底层数据的一致性，该特性可帮助我们降低维护成本的投入。

- **数据一致性：**Doris 提供了 Stream Load Label 机制，我们可通过事务的方式与 Flink 二阶段提交进行结合，以保证幂等写入数据，另外我们通过自研 Flink Sink Doris 组件，实现了数据的端到端的一致性，保证了数据的准确性。

- **SQL 协议兼容：**Doris 兼容 MySQL 协议，支持标准 SQL，这无论是对于开发同学，还是数据分析、产品同学，都可以实现无成本衔接，相关同学直接使用 SQL 就可以进行查询，使用门槛很低，为公司节省了大量培训和使用成本，同时也提升了工作效率。

- **优秀的查询性能：**Apache Doris 已全面实现向量化查询引擎，使 Doris 的 OLAP 性能表现更加强悍，在多种查询场景下都有非常明显的性能提升，可极大优化了报表的询速度。同时依托列式存储引擎、现代的 MPP 架构、预聚合物化视图、数据索引的实现，在低延迟和高吞吐查询上，都达到了极速性能

- **运维难度低：**Doris 对于集群和和数据副本管理上做了很多自动化工作，这些投入使得集群运维起来非常的简单，近乎于实现零门槛运维。

具体应用

Apache Doris 目前广泛应用于 360商业化内部的多个业务场景。比如在实时大盘场景中，我们利用 Doris 的 Aggregate 模型对请求、曝光、点击、转化等多个实时流进行事实表的 Join ；依靠 Doris 事务特性保证数据的一致性；通过多个物化视图，提前根据报表维度聚合数据、提升查询速度，由于物化视图和 Base 表的一致关系由 Doris 来维护保证，这也极大的降低了使用复杂度。比如在账户实时消费场景中，我们主要借助 Doris 优秀的查询优化器，通过 Join 来计算同环比.....

接下来仅以 AB 实验平台这一典型业务场景为例，详尽的为大家介绍 Doris 在该场景下的落地实践，在上述所举场景中的应用将不再赘述。

AB 实验在广告场景中的应用非常广泛，是衡量设计、算法、模型、策略对产品指标提升的重要工具，也是精细化运营的重要手段，我们可以通过 AB实验平台对迭代方案进行测试，并结合数据进行分析和验证，从而优化产品方案、提升广告效果。

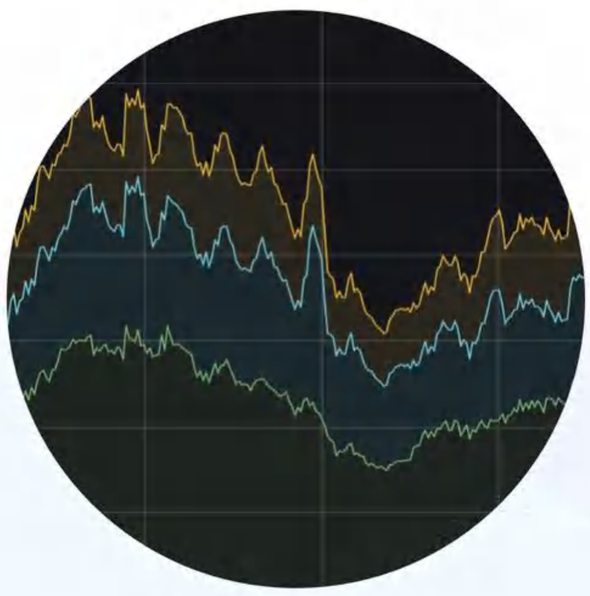
在文章开头也有简单介绍，AB 实验场景所承载的业务相对比较复杂，这里再详细说明一下：

- 各维度之间组合灵活度很高，例如需要对从 DSP 到流量类型再到广告位置等**十几个维度**进行分析，完成从请求、竞价、曝光、点击、转化等几十个指标的完整流量漏斗。
- 数据量巨大，日均流量可以达到**百亿级别**，峰值可达**百万 OPS**（Operations Per Second），一条流量可能包含几十个实验标签 ID。

基于以上特点，我们在 AB实验场景中一方面需要保证数据算的快、数据延迟低、用户查询数据快，另一方面也要保证数据的准确性，保障数据不丢不重。



业务复杂



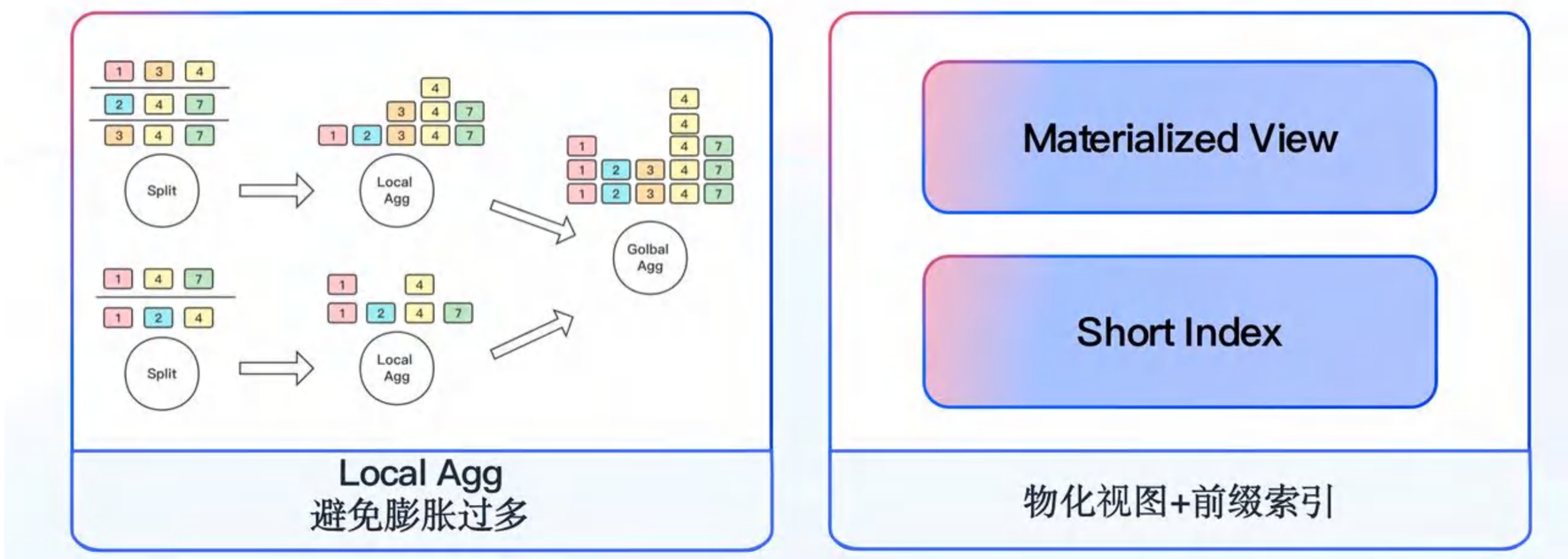
数据量大

数据落地

当面对一条流量可能包含几十个实验标签 ID 的情况时，从分析角度出发，只需要选中一个实验标签和一个对照实验标签进行分析；而如果通过 `like` 的方式在几十个实验标签中去匹配选中的实验标签，实现效率就会非常低。

最初我们期望从数据入口处将实验标签打散，将一条包含 20 个实验标签的流量拆分为 20 条只包含一个实验标签的流量，再导入 Doris 的聚合模型中进行数据分析。而在这个过程中我们遇到一个明显的问题，当数据被打散之后会膨胀数十倍，百亿级数据将膨胀为千亿级数据，即便 Doris 聚合模型会对数据再次压缩，但这个过程会对集群造成极大的压力。因此我们放弃该实现方式，开始尝试将压力分摊一部分到计算引擎，这里需要注意的是，如果将数据直接在 Flink 中打散，当 Job 全局 Hash 的窗口来 Merge 数据时，膨胀数十倍的数据也会带来几十倍的网络和 CPU 消耗。

接着我们开始第三次尝试，这次尝试我们考虑在 Flink 端将数据拆分后立刻进行 Local Merge，在同一个算子的内存中开一个窗口，先将拆分的数据进行一层聚合，再通过 Job 全局 Hash 窗口进行第二层聚合，因为 Chain 在一起的两个算子在同一个线程内，因此可以大幅降低膨胀后数据在不同算子之间传输的网络消耗。**该方式通过两层窗口的聚合，再结合 Doris 的聚合模型，有效降低了数据的膨胀程度，**其次我们也同步推动实业务方定期清理已下线的实验，减少计算资源的浪费。



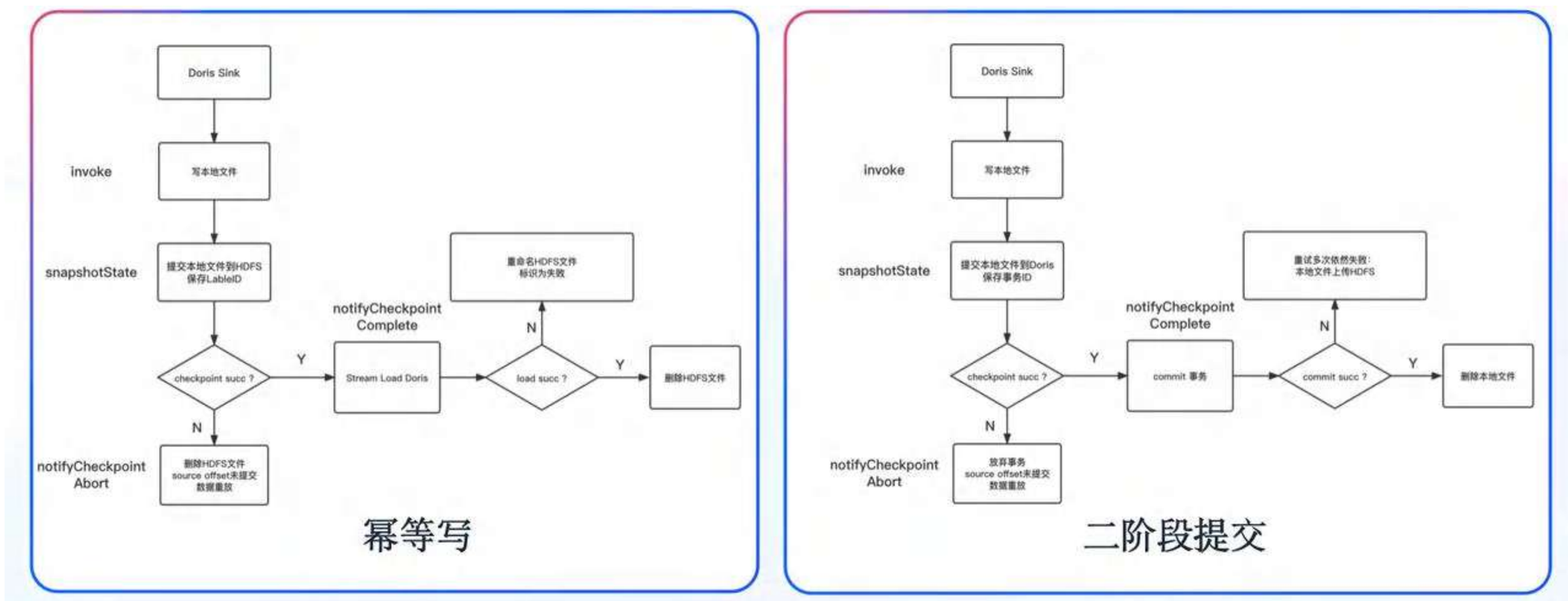
考虑到 AB 实验分析场景的特点，我们将实验 ID 作为 Doris 的第一个排序字段，利用前缀索引可以很快定位到目标查询的数据。另外根据常用的维度组合建立物化视图，进一步缩小查询的数据量，**Doris 物化视图基本能够覆盖 80% 的查询场景**，我们会定期分析查询 SQL 来调整物化视图。**最终我们通过模型的设计、前缀索引的应用，结合物化视图能力，使大部分实验查询结果能够实现秒级返回。**

数据一致性保障

数据的准确性是 AB 实验平台的基础，当算法团队呕心沥血优化的模型使广告效果提升了几个百分点，却因数据丢失看不出实验效果，这样的结果确实无法令人接受，同时这也是我们内部不允许出现的问题。那么我们该如何避免数据丢失、保障数据的一致性呢?

自研 Flink Sink Doris 组件

我们内部已有一套 Flink Stream API 脚手架，因此借助 Doris 的幂等写特性和 Flink 的二阶段提交特性，自研了 Sink To Doris 组件，保证了数据端到端的一致性，并在此基础上新增了异常情况的数据保障机制。



在 Doris 0.14 版本中（初期使用的版本），我们一般通过“同一个 Label ID 只会被写入一次”的机制来保证数据的一致性；在 Doris 1.0 版本之后，通过 “Doris 的事务结合 Flink 二阶段提交”的机制来保证数据的一致性。这里详细分享使用 Doris 1.0 版本之后，通过 “Doris 的事务结合 Flink 二阶段提交”机制保证数据的一致性的原理与实现。

> 在 Flink 中做到数据端到端的一致性有两种方式，一种为通过至少一次结合幂等写，一种为通过恰好一次的二阶段事务。

如右图所示，我们首先在数据写入阶段先将数据写入本地文件，一阶段过程中将数据预提交到 Doris，并保存事务 ID 的状态，如果 Checkpoint 失败，则手动放弃 Doris 事务；如果 Checkpoint 成功，则在二阶段进行事务提交。对于二阶段提交重试多次仍然失败的数据，将提供数据以及事务 ID 保存到 HDFS 的选项，通过 Broker Load 进行手动恢复。为了避免单次提交数据量过大，而导致 Stream Load 时长超过 Flink Checkpoint 时间的情况，我们提供了将单次 Checkpoint 拆分为多个事务的选项。**最终成功通过二阶段提交的机制实现对数据一致性保障。**

应用展示

下图为 Sink To Doris 的具体应用，整体工具屏蔽了 API 调用以及拓扑流的组装，只需要通过简单的配置即可完成 Stream Load 到 Doris 的数据写入 。

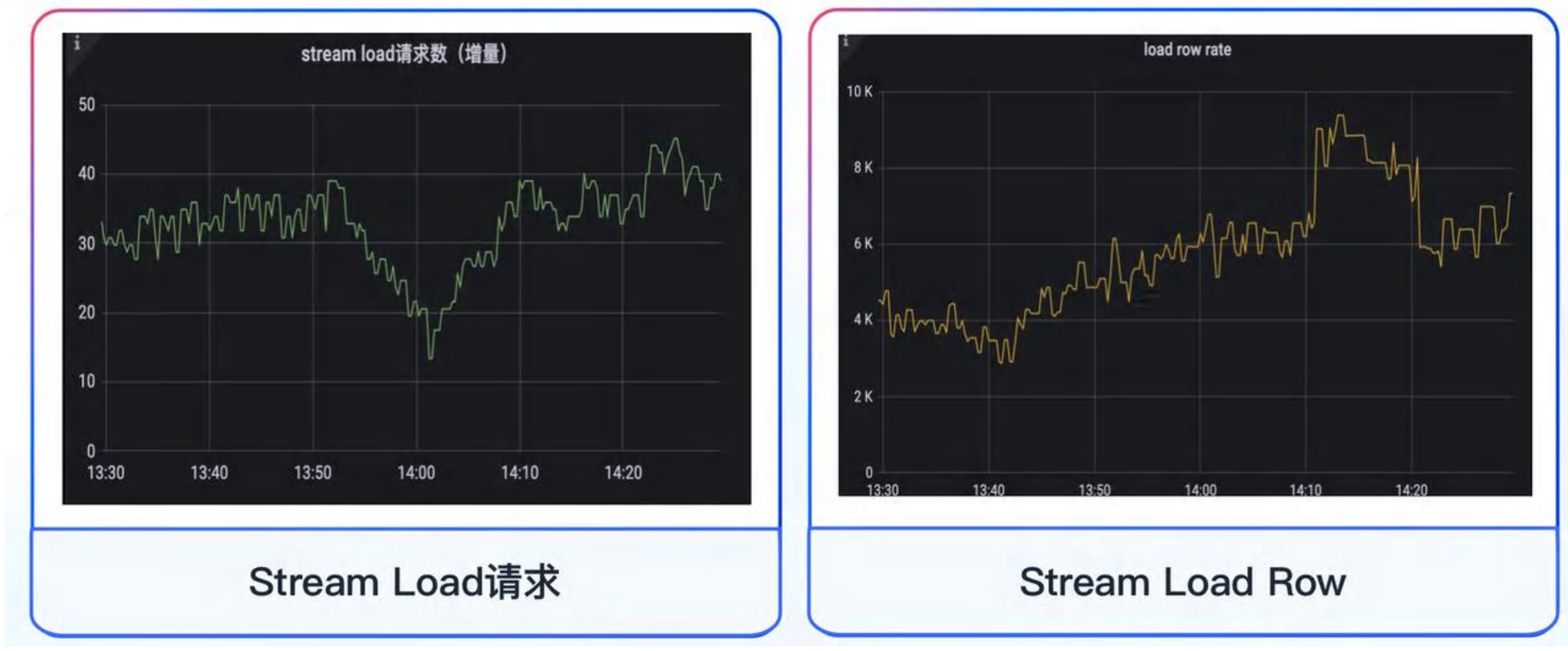
```yaml
sinks:
  -
    name: sink-doris-dwt-max-dsp-ad #必选项
    type: doris #必选项
    dorisSink:
      frontendHostPort:  #fe地址
      database:
      table:
      user:
      password: ''
      saveFailFile: false #是否将commit失败文件保存至hdfs
      reTryNum: 3 # pre-commit、 commit、abort 重试次数
      properties:
        max_filter_ratio: 0.001 #忽略某些脏数据
      fieldMap:   #key为 doris字段名    value为 avro字段名
        dsp_id: dsp_id
        deal_type: deal_type
        flow_type: flow_type
        publisher_type: publisher_type
        version_type: version_type
```

```java
public void commit() {
    if (commitTxnIdList.size() == 0) {
        removePendingFile();
        return;
    }
    long nowMs = System.currentTimeMillis();
    boolean status = false;
    try {
        for (Long txnId : commitTxnIdList) {
            dorisStreamLoader.commit(txnId);
        }
        status = true;
    } catch (Exception e) {
        if (dorisSinkConf.getSaveFailFile()) {
            //commit出错 将localFileInPending文件同步到hdfs
            pendingFileMap.forEach((path, fileName) -> {
                try {
                    hdfsFs.copyFromLocalFile(new Path(path), new Path(hdfsPath + "/" + fileN
                } catch (IOException ex) {
                    LOGGER.error("Doris Sink copy file to hdfs fail :{}", e.getMessage());
                }
            });

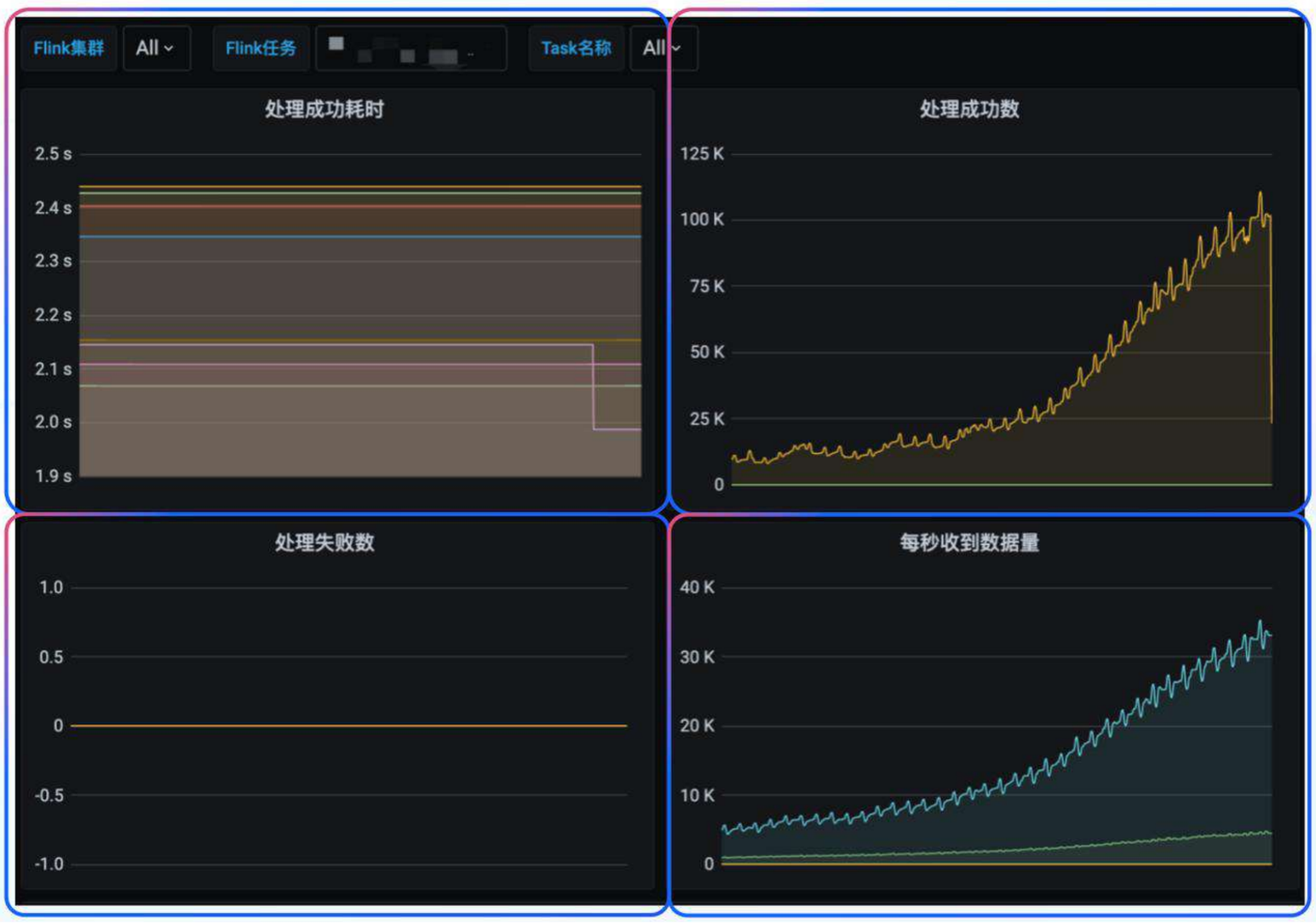
        }
```

集群监控

在集群监控层面，我们采用了社区提供的监控模板，从集群指标监控、主机指标监控、数据处理监控三个方面出发来搭建 Doris 监控体系。其中集群指标监控和主机指标监控主要根据社区监控说明文档进行监控，以便我们查看集群整体运行的情况。除社区提供的模板之外，我们还新增了有关 Stream Load 的监控指标，比如对当前 Stream Load 数量以及写入数据量的监控，如下图所示：



除此之外，我们对数据写入 Doris 的时长以及写入的速度也比较关注，根据自身业务的需求，我们对任务写入数据速度、处理数据耗时等数据处理相关指标进行监控，帮助我们及时发现数据写入和读取的异常情况，借助公司内部的报警平台进行监控告警，报警方式支持电话、短信、推推、邮件等。



总结与规划

目前 Apache Doris 主要应用于广告业务场景，**已有数十台集群机器，覆盖近 70% 的实时数据分析场景，实现了全量离线实验平台以及部分离线 DWS 层数据查询加速。当前日均新增数据规模可以达到百亿级别，在大部分实时场景中，其查询延迟在 1s 内。**同时，Apache Doris 的成功落地使得我们完成了实时数仓在 OLAP 引擎上的统一。Doris 优异的分析性能及简单易用的特点，也使得数仓架构更加简洁。

未来我们将对 Doris 集群进行扩展，根据业务优先级进行资源隔离，完善资源管理机制，并计划将 Doris 应用到 360商业化内部更广泛的业务场景中，充分发挥 Doris 在 OLAP 场景的优势。最后我们将更加深入的参与到 Doris 社区中来，积极回馈社区，与 Doris 并肩同行，共同进步！

万亿数据秒级响应

Apache Doris 在 360数科 实时数仓中的应用

作者：360数科中间件团队

导读：随着业务的不断发展，360数科对数据的安全性、准确性、实时性提出了更严格的要求，亟需对实时数仓架构做出优化和重构。基于此，2022 年 3 月正式对 Apache Doris 调研并投入使用，当前已稳定支持多条业务线，支持万亿级别数据量实现秒级实时分析。360数科将通过本文与大家分享在数仓优化重构过程中应用、优化经验，希望对大家有所帮助。

作为以人工智能驱动的金融科技平台，360数科携手金融合作伙伴，为尚未享受到普惠金融服务的优质用户提供个性化的互联网消费金融产品，致力于成为连接用户与金融合作伙伴的科技平台。360数科旗下产品主要有 360借条、360小微贷、360分期等，截止目前，已累计帮助 141 家金融机构为 4300 万用户提供授信服务、为 2630 万用户提供借款服务、单季促成交易金额 1106.75 亿元。同时作为国内领先的信贷科技服务品牌，360数科在三季度累计注册用户数首次突破 2 亿。

业务需求

随着金融科技业务的不断发展，对数据的安全性、准确性、实时性提出了更严格的要求，早期 Clickhouse 集群用于分析、标签业务场景，但是存在稳定性较低、运维复杂和表关联查询较慢等问题，除此之外，我们业务中有部分报表数据分散存储在各类 DB 中，这也导致维护管理复杂度较高，亟需做出优化和重构。

系统选型

基于以上需求及痛点，我们对实时数仓的选型目标提出了明确的需求，我们希望新的 MPP 数据库**具有以下几个特点：**

- 数据写入性能高，查询秒级
- 兼容标准的 SQL 协议
- 兼容标准的 SQL 协议
- 丰富的数据模型
- 运维复杂度低
- 社区活跃
- 对商业友好，无法律风险

2022 年 3 月开始，我们对符合以上特点的数据库 Apache Doris 展开了为期两个月的调研测试，**以下是 Apache Doris 1.1.2 在各个方面的满足情况：**

类别		Apache Doris
功能特性	向量化	全面向量化
	UDF	早期为 C++ UDF，在1.2 最新版本中已实现了 Java UDF
	资源隔离	有，满足导入、查询的资源隔离
	JSON 数据类型	1.2 版本已支持
	Flinx、Spark、DataX、SeaTunnel等	支持
运维	可视化监控管理控制台（Manager）	简单的功能页面，不满足
	稳定性	高
	性能对比	高 单表：Doris 与 Clickhouse 持平 多表：Doris > Clickhouse
成熟度	社区活跃度	Committer 分布较广且社区活跃
	技术支持	已与官方社区同学进行过 2 次腾讯会议技术交流，可以及时提供以下支持： 1、调研答疑 2、最佳实践
	开源	已于 2022 年 6 月份从 Apache 基金会孵化毕业，风险较小
	是否收费	免费
法律和安全	法律风险	Apache 2.0 License，对商业友好
	安全性	Apache Software Foundation 在消除其软件项目中的安全问题方面采取了严格的立场

基于上述情况，我们决定采用 Apache Doris，除了可以满足上文提到的几个特点，我们还考虑以下几个方面：

- Clickhouse 由于 Join 查询限制、函数局限性、数据模型局限性（只插入，不更新）、以及可维护性较差等原因，更适合日志存储以及保留当前存量业务，不满足我们当前的业务需求。

- 目前Apache Doris 社区活跃、技术交流更多，SelectDB 针对社区有专职的技术支持团队，在使用过程中遇到问题均能快速得到响应解决。

- Apache Doris 风险更小，对商业友好，无法律风险。大数据领域 Apache 基金会项目构成了事实标准，在 360数科内部已有广泛应用，且 Apache 开源协议对商业友好、无法律风险，不会有协议上的顾虑。

平台架构

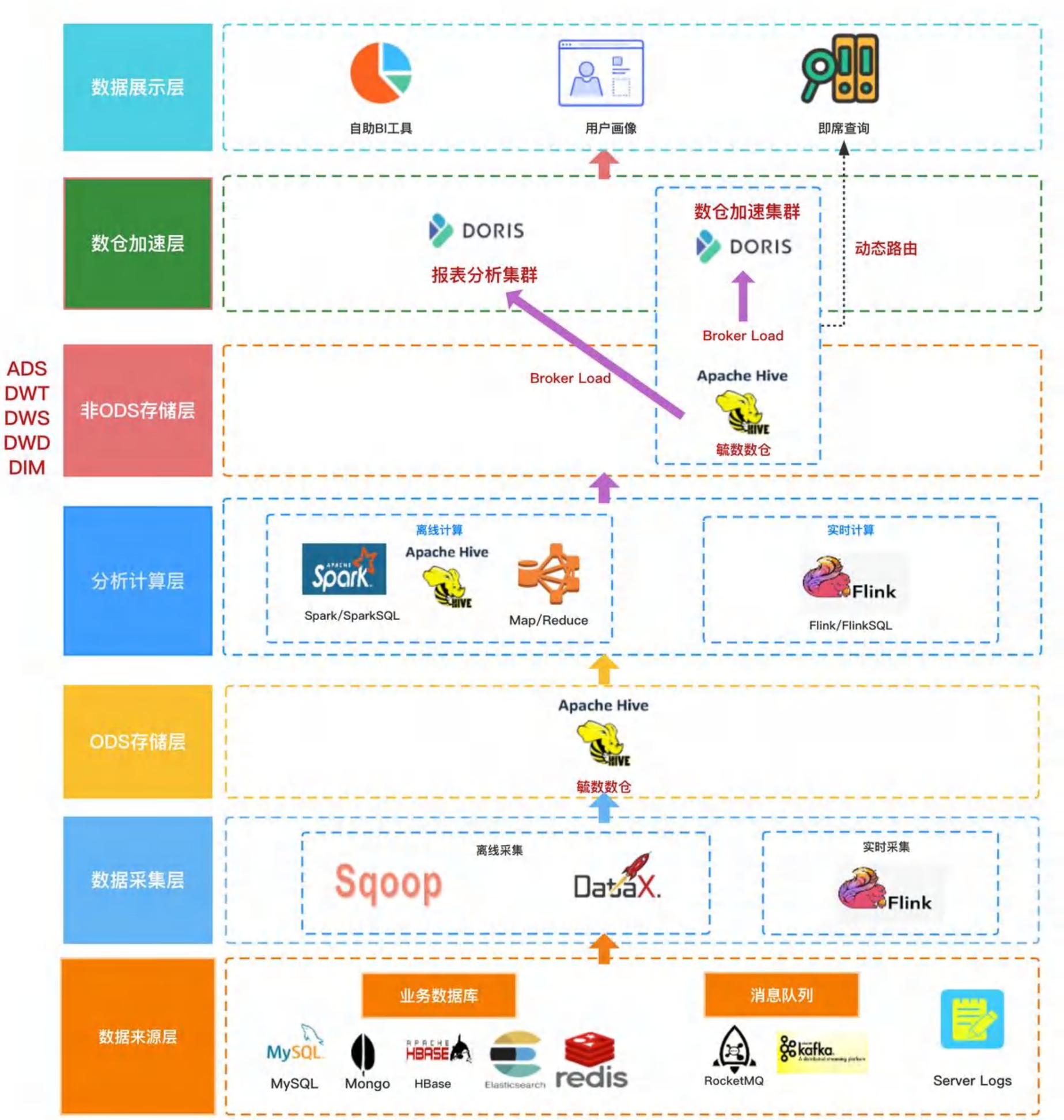
360数科大数据平台（毓数）提供一站式大数据管理、开发、分析服务，覆盖大数据资产管理、数据开发及任务调度、自助分析及可视化、统一指标管理等多个数据生命周期流程。**在整个 OLAP 中，目前 Apache Doris 主要运用离线数仓分析加速、自助 BI 报表等业务场景**（架构图见下一页）。

在引入 Doris 后，考虑已有数据分析业务以及数据规模，Doris 集群将先同步部分业务上优先级更高的数据。通过上述架构图可以看到，依托 Doris 强大的查询性能，我们将把 Doris 架设在 Hive 数仓的上层，为特定场景进行查询加速，这样的架构建设起来成本很低，只需要完成数据从 Hive 数仓到 Doris 集群的导入适配，因为 Doris 集群并没有产生任何新表，可以直接复用已经建设好的数据血缘关系。

数据导入方案，我们在调研了 Stream Load 和 Broker Load ，从导入性能、开发成本上进行了评估，在导入性能上，Broker Load 要比 Stream Load 略胜一筹，而在开发成本上两种方式并没有明显的差异。而且对于大表的同步，Broker Load 的导入方式可以做到单表一次导入一个事务，而 Stream Load 在单表数据量超 10G 时则需要拆分后进行数据导入。因此数据导入选择使用 Broker Load 来进行。

数仓即席查询方案，我们自行开发的查询引擎支持多查询引擎动态切换的机制，通过识别查询数据的元信息对本次查询做自动的查询引擎（Doris/Presto/Spark/Hive）路由和故障切换。

Doris 支持原生 MySQL 协议，对标准 SQL 支持良好，使得 Doris 可以和一些 BI 工具（帆软、观远等）无缝结合，因此单独搭建了一个 Doris 报表分析集群作为 BI 工具数据源。



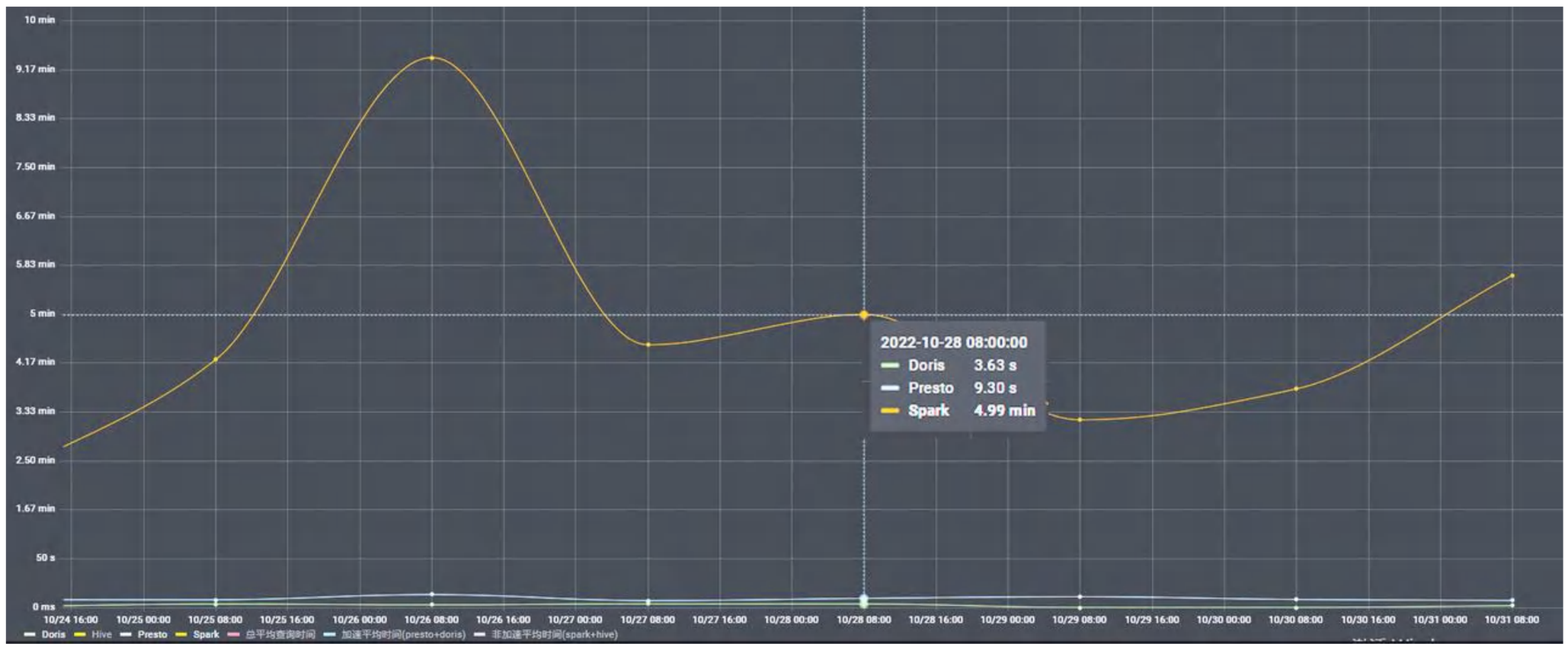
应用实践

Doris 对 Hive 数仓的加速方案

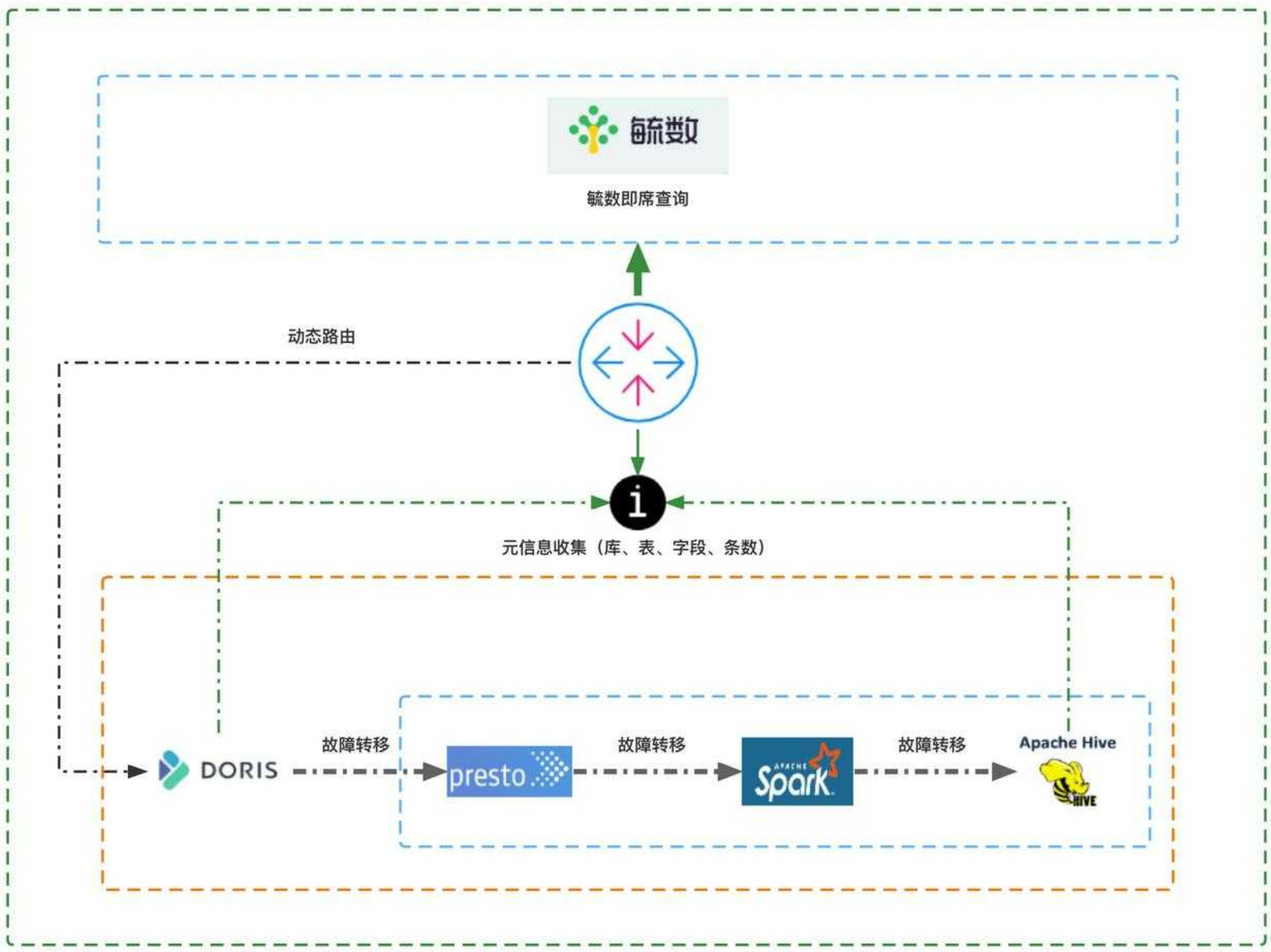
在即席查询场景中，传统的查询引擎（Hive/Spark/Presto）越来越满足不了数据开发者、数据分析师对查询响应性能提出的高要求，动辄几十秒甚者分钟级的查询耗时极大的限制了相关场景的开发效率。

为提高查询性能，我们通过架设的 Doris 数仓加速层来缩短查询耗时，目前我们在不开启 Doris 缓存、不开启用物化视图等优化策略的情况下，**命中 Doris 即席查询平均耗时即可从几分钟缩短至 5 秒内。**

未来我们将通过分析相关查询的特征，通过开启缓存、创建相关物化视图等策略来进一步优化 Doris 的查询性能。



实现 Doris 加速的核心是支持查询引擎动态切换，查询引擎动态切换的工作机制如下：



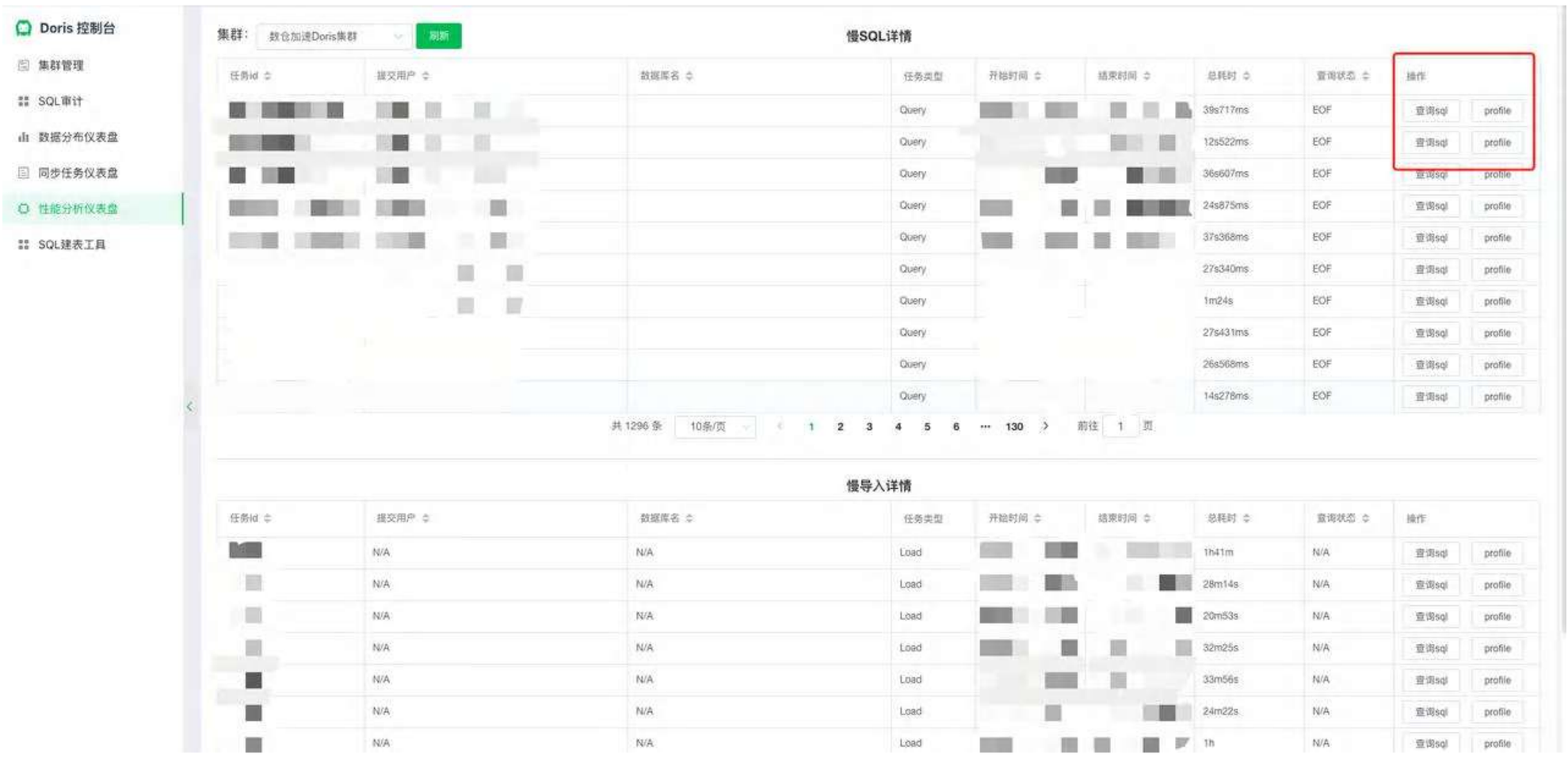
查询引擎会及时收集 Hive 和 Doris 的元信息，包括库、表、表字段、表行数等信息，在用户提交即席查询请求时，首先会解析出用户查询的表，并按照如下顺序判断：

- 查询的表是否已在 Doris 同步
- Doris 表和 Hive 表结构是否相同
- Doris 表和 Hive 表表行数是否一致

如果以上要求均被满足，则会将该查询路由到 Doris，否则会依次按照 Presto、Spark、Hive 的顺序进行路由查询，当查询出现异常时，也会按照该顺序依次进行故障转移。

慢查询慢导入分析

对于慢查询和慢导入，Doris 提供了完善的 Profile 机制，在了解相关技术细节后，我们在线上集群开启了 Profile 收集，通过调度任务定时收集慢查询、慢导入的 Profile 信息并落库。



Doris 提供的 Profile 信息非常详细，例如 `OLAP_SCAN_NODE` 提供了原始的扫描行数，各个索引的过滤行数，每个 Instance 的 `EXCHANGE_NODE` 提供了接收的数据总行数和接收的数据量大小。这些信息为查询调优提供了详细的依据，我们在使用过程中针对快速定位查询性能的瓶颈进行了优化，取得了良好的效果。

建表规范

在我们的使用场景中，有下列类型的表：

- pda表：每日全量更新，即每日分区存储全量快照数据
- pdi表：每日增量更新，即每日分区存储增量数据
- a表：全量不分区表
- s表：静态非每日更新数据

由于当前 Doris 集群中所有的表都是基于 Hive 数仓中各层级的表同步而来，因此目前仅使用了 Duplcate 模型和 Unique 模型，对于 pda、pdi 和 a 表，为了降低 Doris 表的分区数，减轻 FE 元数据管理压力，我们在建 Doris 表时均启用了根据日期划分的动态分区特性，较久远的历史数据我们按年、月的维度分区归档，近期的数据按日、小时分区，未来我们计划通过程序自动识别完成历史分区的归档合并。

对于 pda 表使用场景，pda 表需要每日同步全量数据，我们采用了 Duplicate 模型，不考虑使用 Unique 模型数据去重的原因是 Doris 的导入模型本身就提供了基于任务 Label 的数据一致性保证，同步时一次调度周期的 pda 表的一个分区的导入任务能产生唯一且不变的 Label，因此我们可以保证即使错误执行了多次，该分区的数据仍然不会重复。另外，因为 Duplicate 模型相比于 Unique 模型，在导入和查询阶段均不会做预聚合去重，所以可以一定程度上加速导入和查询的性能。

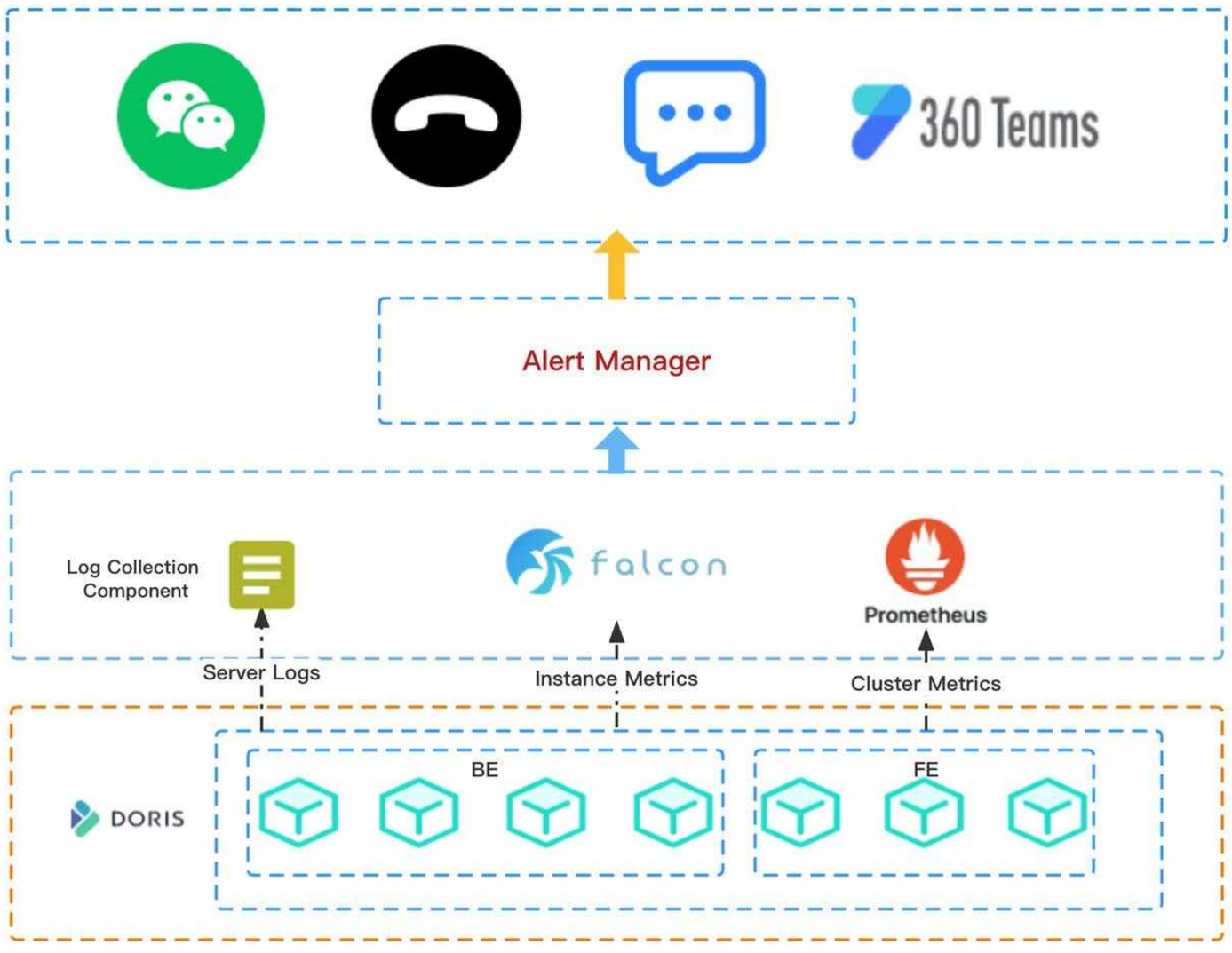
对于 pdi 表使用场景，因在实际使用中 pdi 表存在少数对历史数据的部分更新场景（绝大部分是数据更新场景，基本没有数据删除场景），考虑到 Doris 数据表的分区可用性，我们采用了 Unique 模型，这样在更新历史分区的数据时不必做重建分区操作。

对于 a 表使用场景，因业务上可以接受短时间数据不可用情况，我们启用了动态分区，在做数据导入时，每次导入都会先删除历史分区，然后将全量数据导入今天的分区内，这样做的考虑是杜绝重建表操作，且实施成本相对比较低，因此我们没有采取动态更新视图绑定当日分区的方案。

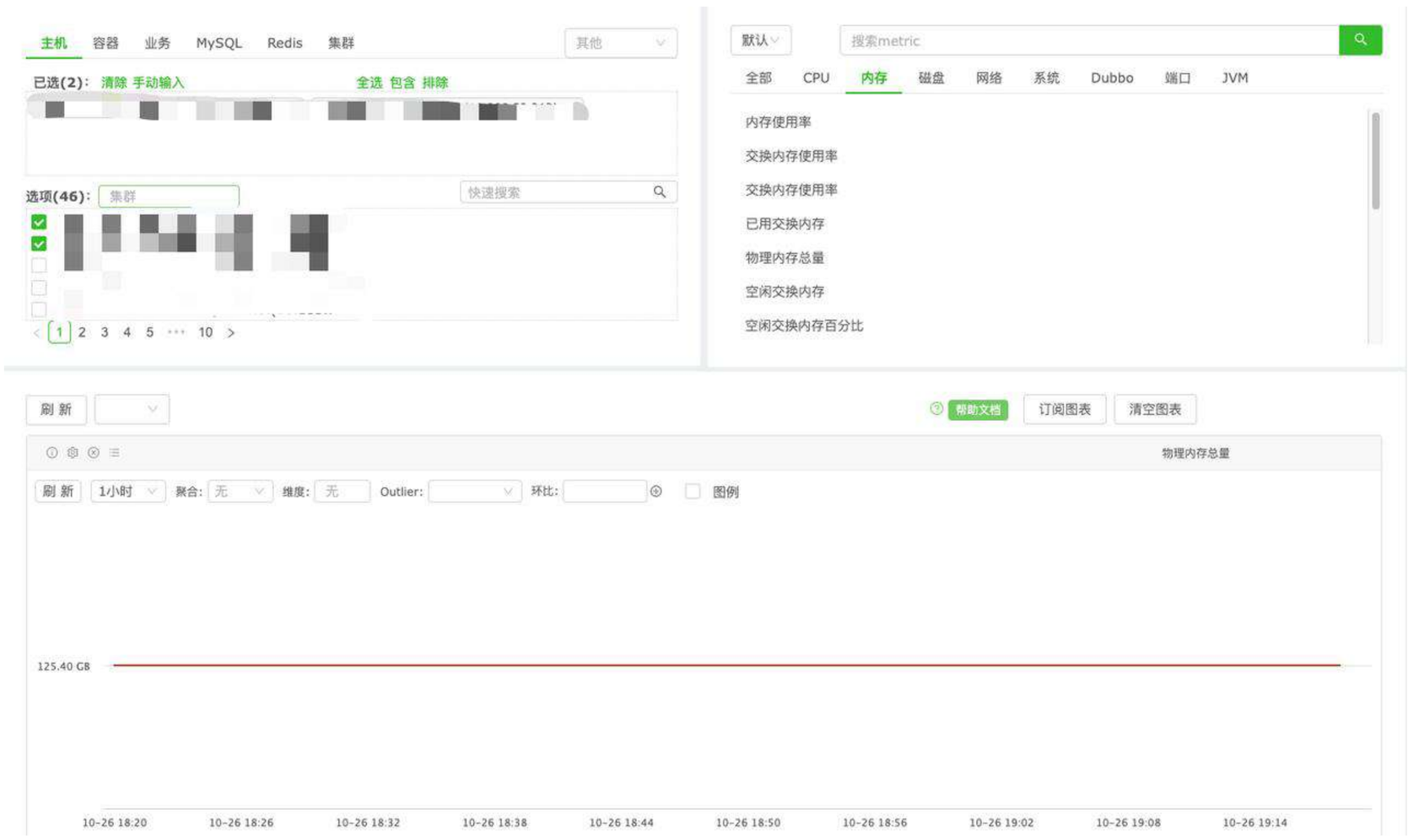
在 Doris 之前的版本中，尚未实现 Hive 元数据变更同步和管理功能，为了提高效率开发了 Doris 建表工具，我们通过选择和配置数仓集群、Hive 表名、数据模型、Bucket 数量等参数，自动关联 Hive 表，解析表字段并生成对应的建表语句。**经过与社区沟通得知，最近即将发布的 1.2 新版本中已经实现 Multi Catalog，支持 Hive 元数据的对接和 Schema 的自动同步，可以极大程度上减少这一部分的工作。**

监控体系

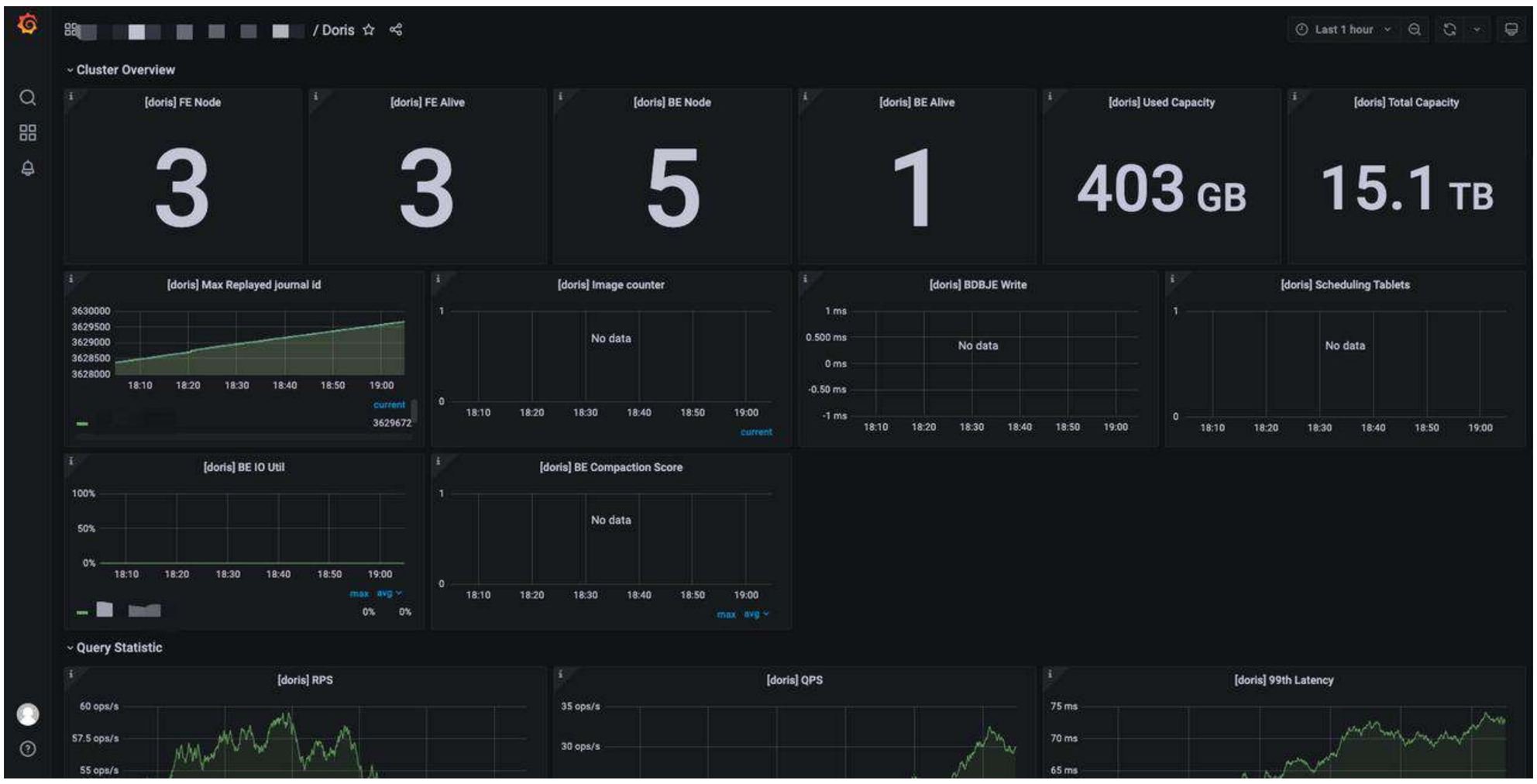
当前 Doris 集群监控体系分为主机指标监控告警、日志告警和集群指标监控告警，总体监控体系如下。



主机指标监控是基于 Open-Falcon 开发的监控告警平台，主要采集 Doris 集群节点的 CPU、IO、内存、磁盘等相关指标并进行监控告警。



集群指标监控参考了 Doris 官方文档提供的基于 Prometheus 和 Grafana 和集群指标监控方案。



日志告警仍然是基于我们的监控告警平台，主要用于监控 Doris 服务日志中容易识别但其他监控方式成本较高的监控、告警场景，是其他两种监控的补充。通过日志监控告警，我们能够准确识别数据导入任务的失败原因并能进行及时的推送通知。

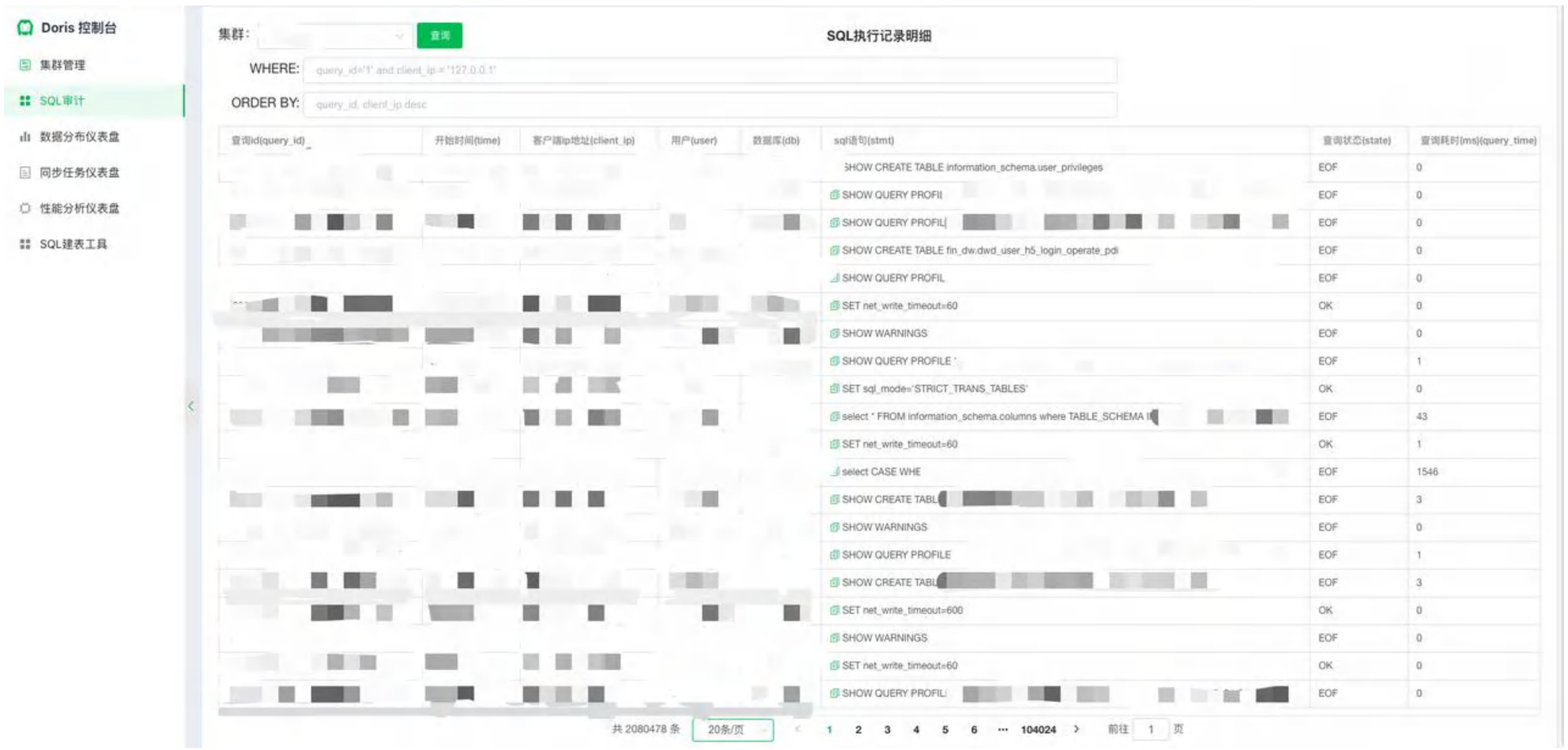
问题排查和审计日志

为了及时排查一些极端的集群问题，上述针对 Doris 的监控体系建设仍然是不够的。为了在集群 BE 出现异常宕机时快速定位堆栈，需要在所有的 BE 节点开启 Core Dump。除此之外，审计日志在集群的日常运维中也发挥了重要作用。

对于 Doris 集群的审计日志收集一般可以通过 2 种方式：

- 第一种方式是通过日志收集组件、收集各个 FE 节点上的 `fe.audit.log`
- 第二种方式是通过安装 Doris 提供的 Auditloader 插件（下载 Doris 源码，该插件在 `doris/fe_plugins/auditloader` ，具体使用文档可参考官方文档）。

考虑到第二种方式操作更简单，因此采用此方式进行日志采集。不过在使用 Auditloader 插件的过程中，陆续发现和修复了一些插件问题，并向社区提交了 PR，与此同时，我们定制开发了内部控制台，便于查看集群的同步任务情况，数据分布情况以及进行审计日志的检索。



审计日志为集群 BE 崩溃时具体 SQL 定位、客户端访问统计、查询 SQL 耗时统计、访问 SQL 特征分析等提供了详细的信息。例如，数据开发曾经反馈查询 Doris SQL 失败，检索日志出现了大量连接数超限的异常，我们通过审计日志，迅速定位到了问题原因是由于上游导入工作流 Bug 在短时间内创建较多的数据库连接。另外，对于曾经使用的低版本 Doris 出现数次 BE 异常宕机问题，我们通过 gdb 调试工具定位到崩溃时 SQL 的 `query_id` 后，配合审计日志也能快速的定位到导致崩溃的具体 SQL。

优化实践

数据导入实践和调优

初期数据源主要来自 Hive 数仓，因此大部分数据导入以 Broker Load 方式为主。大数据平台自助导入任务工作流适配了 Doris Broker Load 导入方式，数据开发零代码——通过简单的勾选配置即可完成自助的 Doris 数据导入工作流创建。

而在 Broker Load 的使用过程中，我们也陆续遇到了一些问题，这里拿出几个典型的问题和一些调优经验来分享。

在 Broker Load 导入时遇到的问题：

1）因表分桶数设置过少造成 Broker Load 导入失败，具体表现为导入任务失败且异常信息为：

```
1   tablet writer write failed, tablet_id=xxx, txn_id=xxx, err=-238
```

我们推测造成 -238 错误的原因可能是分桶设置太少，接着我们通过 BE 节点的挂载数据来查看单个 Tablet 下的文件大小，我们发现单个 Tablet 的文件占用空间远大于官方推荐的 10GB 上限范围，这也证明了我们的推测正确，因此我们通过适当提高 Doris 表的分桶数，使得这个问题有了较大的缓解。

顺便说一下，如果出现 -235（旧版本是 -215）异常，一般是由于 Compaction 过慢导致 Tablet 版本堆积超过限制，这个时候通过 Grafana 看到 BE Compaction Score 在导入前后有明显的波动，而且绝对值很高。

2）因 Hive 表字段变更导致 Broker Load 导入失败：

Hive 表在使用过程中会有一些 DDL 的执行，从而导致表字段新增，我们数仓的 Hive 表均使用 ORC 格式存储，那么就会导致 Hive 表中部分历史分区的 ORC 文件中字段信息缺失（缺失新增字段），而新分区的 ORC 文件中字段是正常的，这个时候如果对历史数据重新导入，就会有下面的异常信息：

```
1   detailMessage: ParseError : Invalid column selected xxx
```

在阅读了 Broker Load 相关代码后确认了问题原因：在一次 Broker Load 导入过程中，导入任务的字段解析器会读取一个 ORC 文件头解析字段信息，但解析器只会解析一次，如果一次导入过程中同时有新、历史分区的 ORC 文件，那么就可能导致任务失败。

修复的方法也很简单，只需针对每个 ORC 文件重新解析一次文件头的字段信息即可。在了解问题原因及分析解决思路后，我们也和社区的同学一起修复了这个问题并提交了相关 PR。

3）遇到空 ORC 文件时 Broker Load 导入失败：

这个问题的错误表现和问题 2 比较类似，具体原因是 Broker Load 导入过程没有对 ORC 文件做判空，遇到空 ORC 文件仍会尝试解析 ORC 文件字段信息导致报错，我们把这个问题反馈给社区后，社区的同学很快修复了该问题。

4）Broker Load 导入任务出现 `Broker list path exception. path=hdfs:xxx`

创建 Broker Load 任务，使用 Kerberos 认证访问 HDFS 的 Hive 文件导入数据，Hive 文件路径中分区和下一级目录使用通配符 *，访问所有分区所有文件，任务提交后隔 40 多秒出现如下的错误：

```
type:ETL_RUN_FAIL; msg:errCode = 2, detailMessage = Broker list path exception. path=hdfs:xxx
```

在阅读了 Broker Load 的访问 HDFS 相关代码后确认了问题原因，Broker Load 调用 HDFS 的 LS、DU 方法时会获取文件目录信息，由于路径下的文件过多导致耗时会超过 45 秒，而 Thrift 设置的 Socket 请求超时默认小于 40 秒，所以出现了上述的 RPC 异常，问题反馈社区后，对 FE 增加了配置参数 `broker_timeout_ms` ，设置为 90 秒后解决问题。

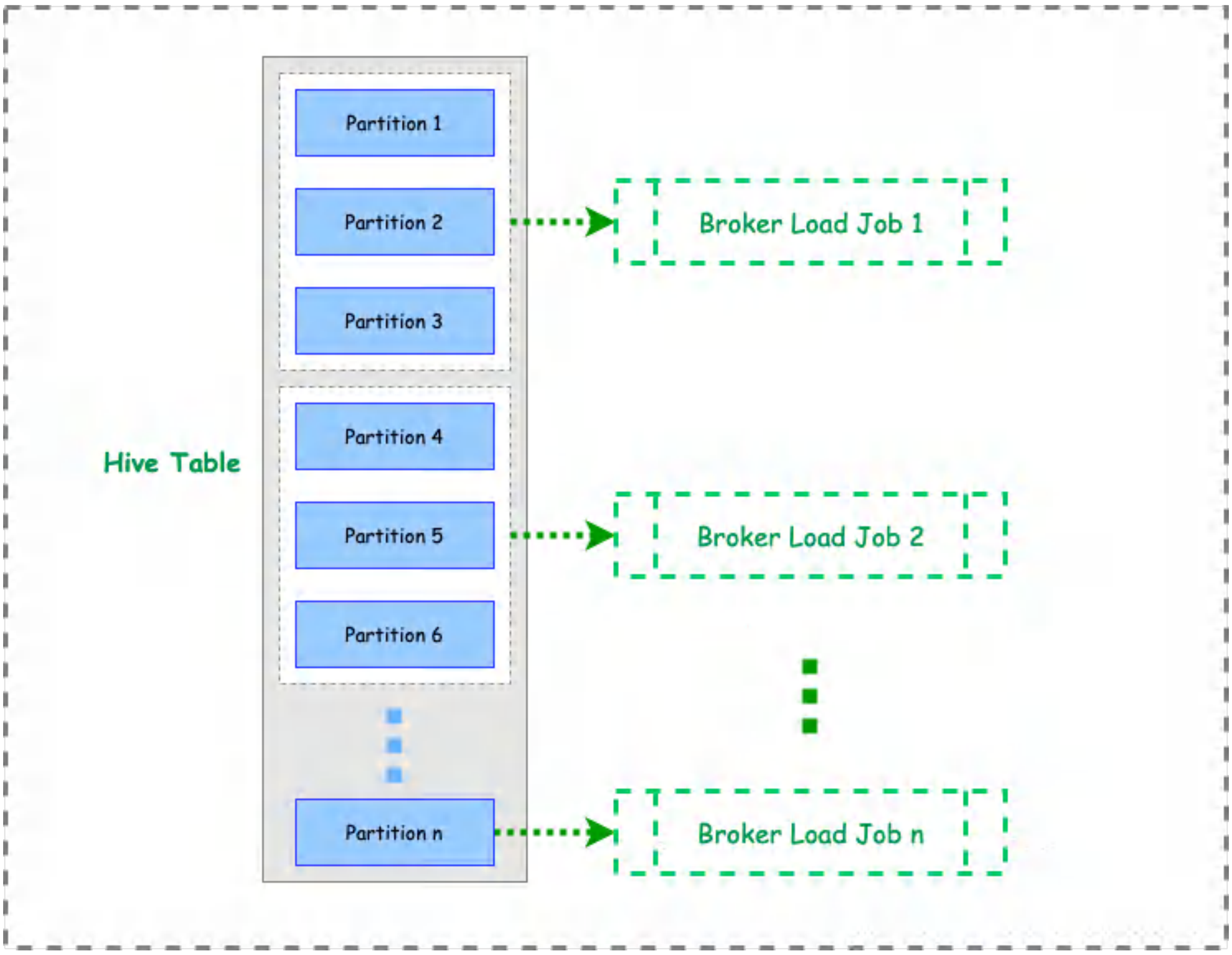
关于 Broker Load 的导入性能调优策略

我们针对 Broker Load 导入调优的主要方向在确保 Doris 集群不承压的情况下尽可能提高导入并发度，下面根据 2 个典型的案例来说明：

1）部分 pdi/pda 表因为数据规模太大导致全量导入耗时过长（**导入数据源是 Hive 分区表**）

部分 pdi/pda 表数据规模在 T 级别，在进行全量导入时，如果只提交一个 Broker Load Job ，将因为导入任务的并发不够，导致导入耗时达到 5-6 小时。针对此问题，我们可以对导入任务进行 Job 拆分，在大数据平台也适配这种场景，支持任务的自动拆分和重试机制（具体的拆分方式图见下一页）

不过要注意的是，拆分后可能会对集群有较高的写入压力，要及时监控导入任务和集群的状态，特别针对 -235 的情况可能需要进行 Compaction 调优。

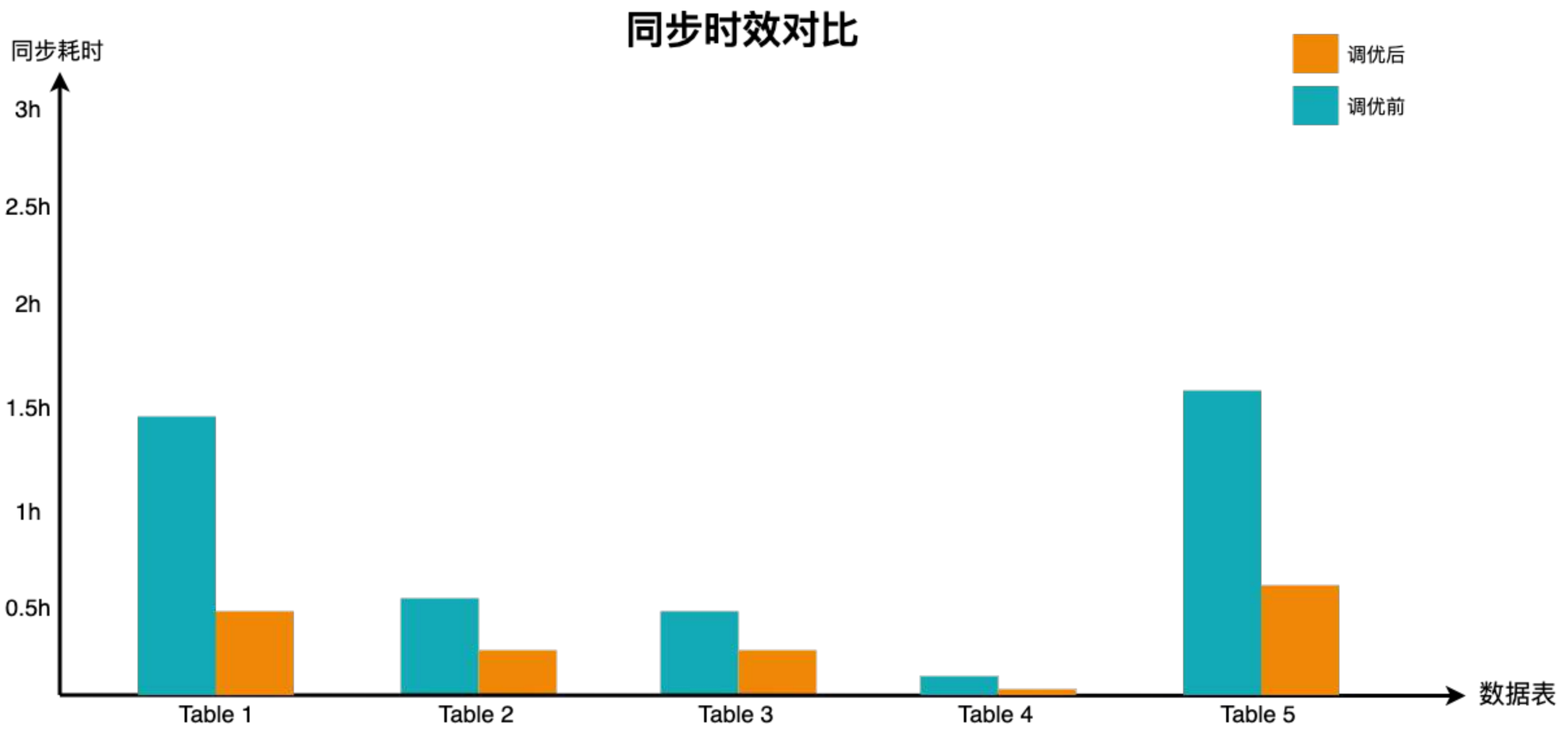


2）部分 ads 表因为数据规模太大导致全量导入耗时过长（**导入数据源是Hive非分区表**）

数据开发对部分报表的同步时效提出了很高的要求，我们在针对性的优化表同步时效时，发现一些表导入耗时较长，但通过集群监控体系发现相关表同步期间，BE、FE 节点的 CPU、内存、磁盘 IO 、网卡 IO 并没有达到瓶颈，集群的 Compaction Score 在此期间也一直稳定在低位，且整个同步过程同步任务均未出现 -235、-238 等相关的错误，我们推测瓶颈可能还是在导入任务的并发程度上。

因为有些表在 Hive 数仓是非分区的表，所以第 1 种通过划分分区范围拆分多个导入 Job 的方式就行不通了，理论上仍可以通过划分不同的 HDFS 文件来拆分 Job，但是这种方式在毓数大数据平台还需要进一步去适配，所以我们还是优先考虑通过调整集群配置的方式彻底解决此问题：

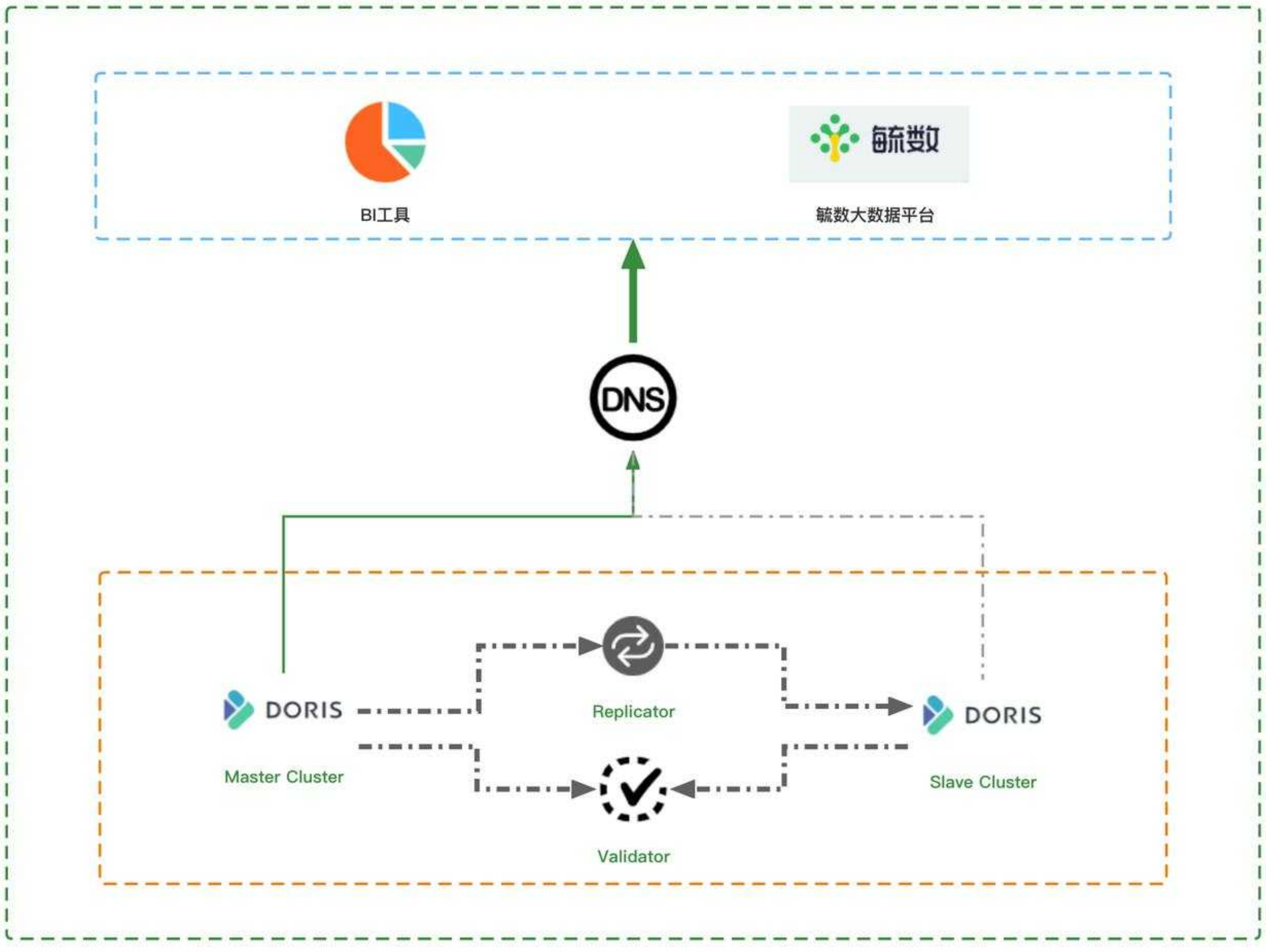
首先可以通过适当调高 FE 的 `max_broker_concurrency` 去提高 Scan HDFS 文件阶段的并发度（最高调高至 BE 节点数），而对于 Table Sink 阶段，可通过调高 FE 的 `default_load_parallelism`（设置 `fe.conf`，可调整到 BE 节点数）e和 `send_batch_parallelism` 参数（SQL Session 执行 `set global send_batch_parallelism=5` 或在提交 Broker Load 中的 PROPERTIES 中指定，最高调整到 5，如果超过此值，需要同步调整 `be.conf` 的 `max_send_batch_parallelism_per_job` 参数），提高该阶段并发度。通过提高 Broker Load Job 各阶段导入的并发度，相关报表的同步时效显著提升，这里我们选取 5 张典型表为例，**优化前后的同步时效表现如下：**



双机容灾建设

为了保障 Doris 集群的可用性，我们需要为 Doris 集群提供双机房容灾能力。Doris 目前虽然可以通过不同的 Tag 将 BE 分组部署在多个机房，但是无法解决机房出现问题时的 FE 可用性问题。经过方案调研分析，我们决定通过自行开发 Replicator 主从同步插件去实施双机房容灾建设（具体架构图见下一页）

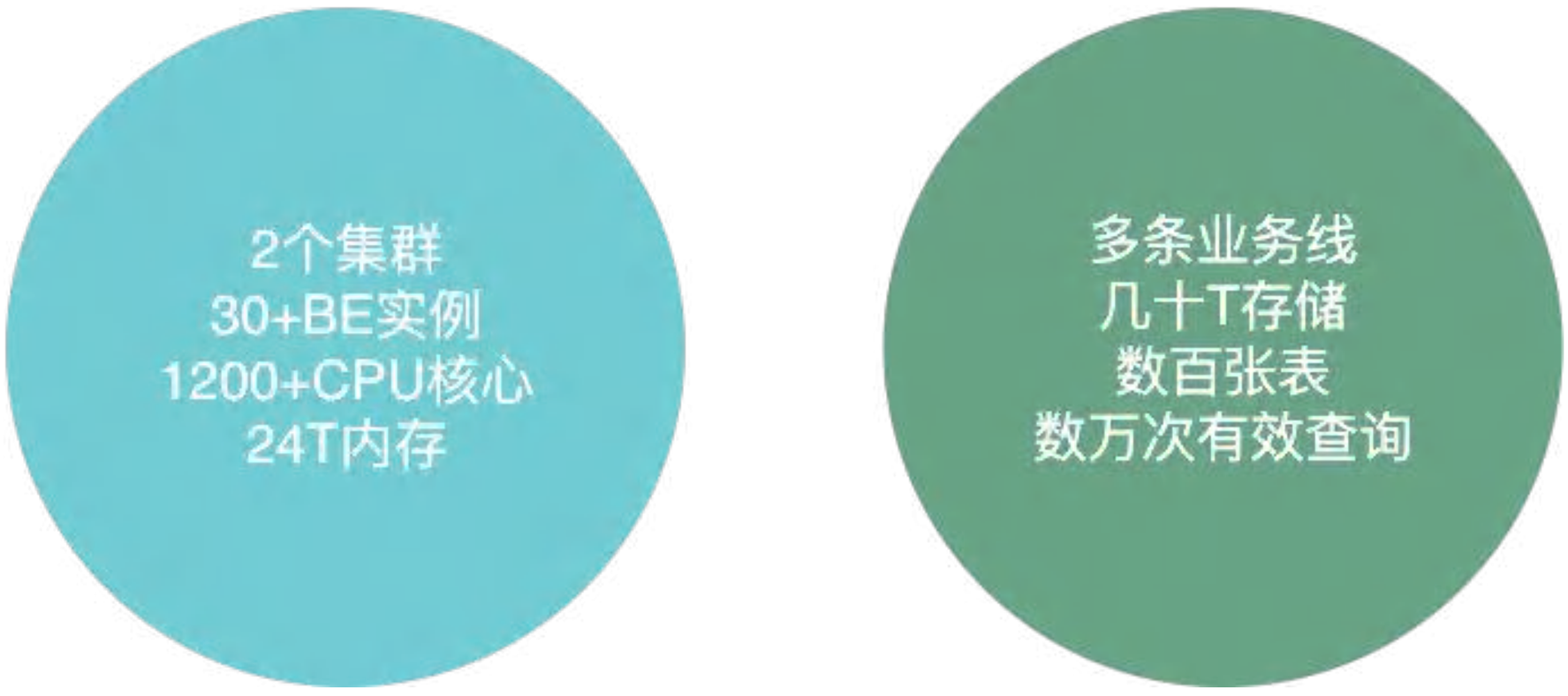
通过在主集群安装 Replicator 插件，Replicator 插件会拦截并解析主集群执行的全量 SQL，然后经过过滤操作，筛选涉及库、表结构变更和数据增、删、改相关的 SQL，并将相关 SQL（部分 SQL 需要改写）发送到备集群进行重放。除此之外，我们在 Doris 控制台开发了 Validator 数据校验程序，定期校验主备集群间的数据结构差异和数据差异并上报，在主集群因各种问题导致不可用时，直接通过切换 DNS 解析地址到备集群 LVS 地址完成主备集群的切换。



总结规划

获得收益

从 2022 年 3 月份开始进行对实时数仓沟通进行调研，7 月份正式上线生产，集群数据规模快速增长。目前，生产环境共有 2 个集群，数百张表，几十 TB 数据，每日有数百个同步工作流在运行，几十亿规模的数据新增/更新。在此规模下，Doris 对业务支持良好，稳定运行。



- **Doris 集群架构清晰简单，**不依赖其他组件，数据模型简单，数据导入方式多样化且适配成本很低，使得我们可以快速完成前期的调研测试并在短时间内上线实施。
- Doris 集群作为目前公司 BI 工具的重要数据源，承载了相当一部分的报表分析业务，极大加速了报表分析的时效性。Doris 上线 3+ 月的时间，已经承载了小部分即席查询场景，**大大缩短了相关查询的耗时。**
- Doris 具有完善的监控机制和审计机制，**极大的降低了我们的运维工作**
- **Doris 社区十分活跃，**在我们使用 Doris 过程中遇到的一些疑难问题，官方也可以及时进行响应、处理。

未来规划

在近期的规划中，**我们希望 Doris 能支撑更多的业务场景、发挥更大价值。**例如基于 Doris 建立实时数仓、基于 Doris 重构用户行为画像、Doris HIVE 外表特性等。同时我们计划通过分析用户的查询 SQL 特征，结合 Doris 的查询缓存和物化视图特性，进一步提升查询效率。通过开发集群探查工具，实时探测集群数据表的数据分布情况，比如 Tablet 有没有过大，Tablet 数据分布是否均匀等，综合探查集群的运行情况并自动给出优化建议。

目前我们使用了 Doris 有大半年时间，在这半年期间一直保持和社区同学进行交流（提交 Issues 和 PRs）。**非常感谢 SelectDB 团队一直以来对我们的技术支持，祝 Apache Doris 未来越来越好，为基础软件建设添砖加瓦。**

千万数据计算时间缩短 40 倍

橙联基于 Apache Doris 的数仓架构建设实践

作者：橙联大数据研发经理，付帅

导读：为了适应快速的增长需求，橙联于 2022 年正式引入 Apache Doris，以 Apache Doris 为核心构建了新的数仓架构，构建过程中对服务稳定性、查询稳定性、数据同步等多方面进行了优化，同时建立了以 Apache Doris 为核心的数据中台，在这一过程中积累了诸多使用及优化经验，在此分享给大家。

业务背景

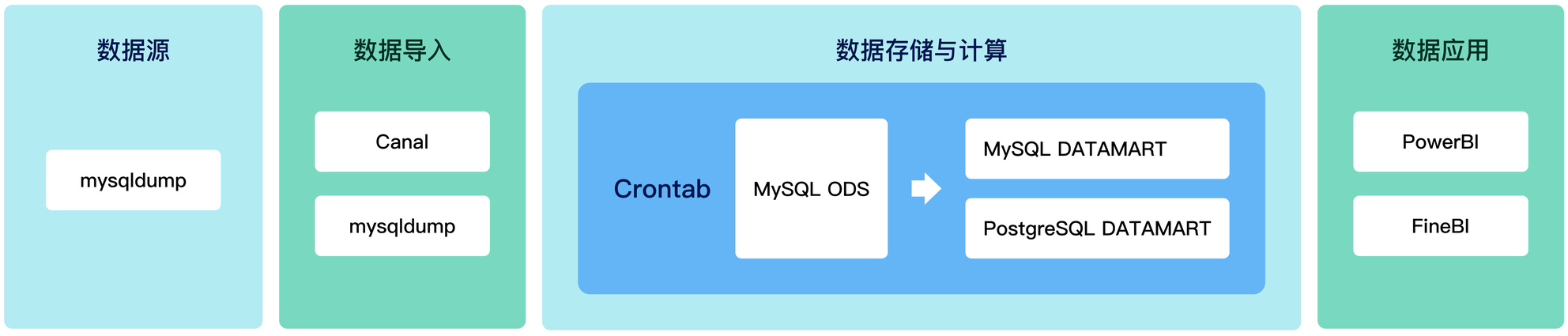
橙联股份是一家服务全球跨境电商的科技公司，致力于通过市场分析、系统研发及资源整合，为客户提供物流、金融、大数据等多方面的服务产品，为全球跨境电商提供高品质、全方位的服务解决方案。

随着公司业务的发展和数据的不断增长，早期基于 MySQL 的传统数仓架构已经无法应对公司数据的快速增长。业务的需求和运营的决策对于数据时效性的要求越来越高，对数仓准实时能力的需求越发强烈。为了适应快速的增长需求，橙联于 2022 年正式引入 Apache Doris，以 Apache Doris 为核心构建了新的数仓架构，构建过程中对服务稳定性、查询稳定性、数据同步等多方面进行了优化，同时建立了以 Apache Doris 为核心的数据中台，在这一过程中积累了诸多使用及优化经验，在此分享给大家。

数据架构演进

早期数仓架构

公司在成立初期业务量不大，数据团队规模比较小，对数据的需求仅局限于少量 T+1 定制化报表需求。因此，早期数仓架构十分简单，如下图所示，直接使用 MySQL 构建数仓集市层，使用数仓集市层的数据进行报表开发，基于商业化的报表平台向需求方提供 T+1 的数据报表服务。



存在的问题

1. 随着公司业务规模扩大、数据量激增以及对数据时效性要求不断提高，使用 MySQL 进行数据分析越来越不能满足业务方的要求。
2. 没有对数仓进行分层划域，烟囱式的开发模式数据复用性很差，开发成本高，业务方提出的需求不能快速得到响应。
3. 对数据的质量和元数据的管理缺乏管控。

新数仓架构

为了解决旧架构日益凸显的问题，适应快速增长的数据和业务需求，今年正式引入 Apache Doris 构建新的数仓架构。

选择 Apache Doris 的原因



易用性 – 在当前应用场景下，引入新技术，将面临大量报表迁移问题，因此必须要考虑的产品易用性问题，而 **Apache Doris 在学习成本、报表迁移成本、服务运维成本上有着非常优秀的表现**，具体包括：

1. 采用 MySQL 协议和语法，支持标准 SQL，可以通过各类客户端工具来访问 Doris，能够与 BI 工具无缝对接
2. 支持多表 Join，针对不同场景的 Join 提供了多种优化方案
3. 生态扩展完善，离线数据的高效批量导入、流式数据的低延迟实时导入都有很好的支持
4. 相较于业界其他热门 OLAP 数据库，Apache Doris 的分布式架构非常简洁，只有 FE、BE 两个进程，运行不依赖任何第三方系统
5. 支持弹性伸缩，对于部署、运维非常友好。

性能 – 当前报表存在大量降耦聚合操作，对多表关联的查询性能和实时查询的时效性有着十分高的要求，而 **Apache Doris 基于 MPP 架构实现，并自带了高效的列式存储引擎**，可以支持：

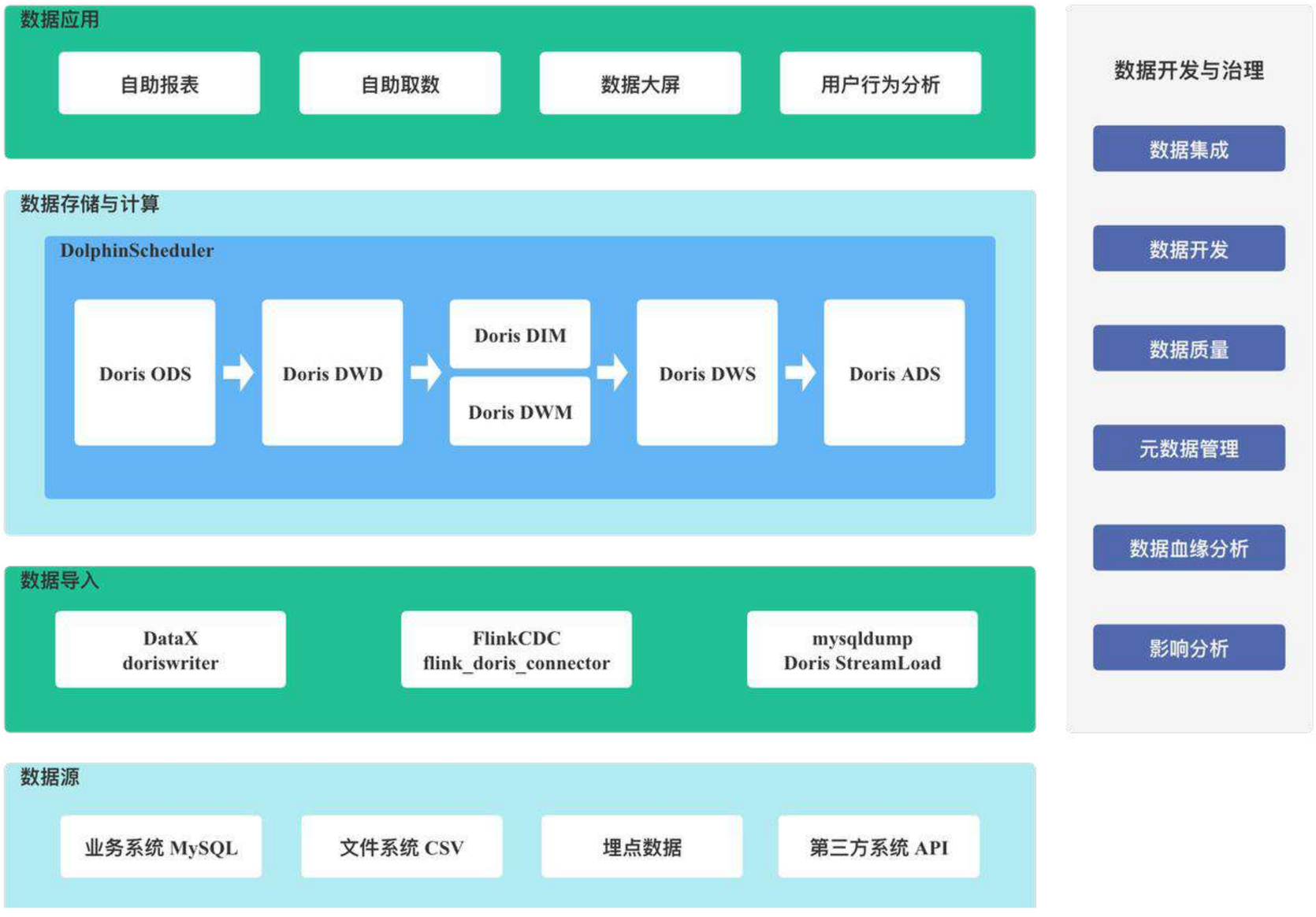
1. 数据的预聚合以及预聚合结果的自动更新
2. 数据的实时更新
3. 高并发查询

基于以上的原因，最终选择了以 Apache Doris 为核心构建新的数仓。

架构介绍

Apache Doris 的数仓架构十分简洁，不依赖 Hadoop 生态组件，构建及运维成本较低。

如架构图（见下一页）所示，我们的数据源共有 4 种，业务数据 MySQL、文件系统 CSV、埋点数据和第三方系统 API；针对不同的需求，使用了不同的数据导入方式，文件数据导入使用 Doris Stream Load，离线数据使用 DataX Doriswriter 进行数据初始化，实时增量数据使用 Flink CDC + Flink Doris Connector 的方式进行数据同步；数据存储和计算层使用了 Doris，在分层设计上采用 ODS（Operation Data Store 数据准备区，也称为贴源层）、明细层 DWD、中间层 DWM、服务层 DWS、应用层 ADS 的分层思想，ODS 之后的分层数据通过 DolphinScheduler 调度 Doris SQL 进行增量和全量的数据更新。最终上层数据应用使用自研的一站式数据服务平台，可以和 Apache Doris 无缝对接，提供自助报表、自助取数、数据大屏、用户行为分析的数据应用服务。



基于 Apache Doris 的数仓架构方案可同时支持离线和准实时应用场景，准实时的 Apache Doris 数仓可以覆盖 80% 以上的业务场景。这套架构大大降低了研发成本，提高了开发效率

当然在架构构建过程中也遇到一些问题和挑战，我们针对问题进行了相应的优化。

Apache Doris 构建数仓优化方案

在数仓的使用过程中，主要遇到三方面问题。首先是服务稳定性问题，其次是查询速度逐渐变慢的问题，最后是 Doris 数据同步和 Doris SQL 调度问题。具体体现在以下：

服务稳定性

优化前

在 Apache Doris 使用初期，FE 和 BE 的部署方式如下：

- FE 负责元数据的管理、用户请求接入、查询的解析规划，资源占用较低，因此将 FE 和其他大数据组件混合部署 FE*3。
- BE 负责数据存储、计算、查询计划的执行，资源占用较大，因此 BE 进行独立部署且分配了较多的资源 BE（16C 128G 4T*1）*7。

基于以上方式部署，使用初期运行的稳定性还不错，然而在使用一段时间之后，这种部署方式暴露的问题就越来越明显。

存在的问题

- 首先，FE 混合部署存在资源竞争。其他组件与 FE 服务竞争资源，导致 FE 资源不足，服务运行稳定性不好。具体问题表现在：每当机器资源使用率打满，就会导致 FE 节点无法连接，长时间获取不到心跳而被 FE 集群判定为离线。

- 其次，BE 单磁盘存在 Compaction 效率低的问题。初期，我们在部署 BE 时，每个节点只分配了 1 块 4T 的磁盘，虽然磁盘的空间并不小，但是磁盘的数量比较少，Compaction 线程数只有 2，Compaction 效率很低，这是导致 BE Compaction Score 不健康的原因之一。

- Compaction 配置参数：

`compaction_task_num_per_disk` 每个磁盘上的任务数，默认为 2

`max_compaction_threads Compaction` 线程的总数，默认为10

`total_permits_for_compaction_score Compaction` 任务配额，默认 10000

> Compaction 工作机制：Apache Doris 的数据写入模型使用了与 LSM-Tree 类似的数据结构。数据以追加（Append）的方式写入磁盘，在读逻辑中，需要通过 Merge-on-Read 合并处理写入的数据。Merge-on-Read 会影响读取的效率，为了降低数据读取时需要合并的数据量，使用 LSM-Tree 的系统会引入后台数据合并逻辑，以一定策略定期的对数据进行合并。

优化后

为了解决以上的问题，对部署方式进行了优化以提升服务的稳定性：

- FE 进行独立部署，避免了 FE 混合部署资源竞争问题

- BE 进行磁盘拆分，多磁盘部署，从原来一块 4T 磁盘变更为 5 块 1T 磁盘，使 BE Compaction 线程数提升 5 倍，Compaction 效率、磁盘 I/O 带宽也得到了提升。

- 增加客户端连接代理层 ProxySQL，对 FE 连接进行负载均衡，解决 FE 连接单点问题。

- 增加 Supervisor 对 FE、BE 服务进程状态监控，FE、BE 进程意外宕机可以快速恢复。

经过以上对部署的优化，Apache Doris 服务的稳定性有了很大的提升，基本可以满足目前对稳定性的需求。

查询稳定性

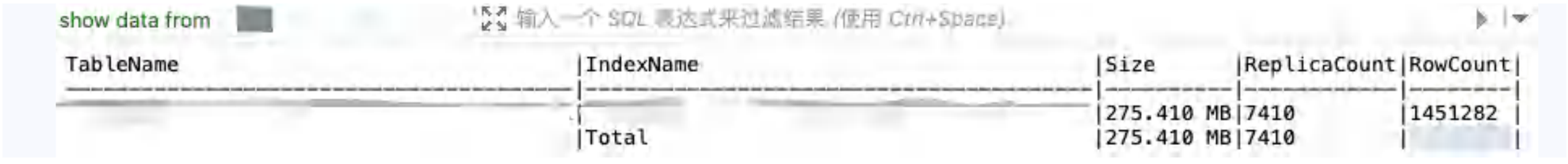
初期刚部署时，无论进行数据导入还是数据查询，执行起来都比较顺畅。但随着承载的表和数据导入作业数量不断增多，查询稳定性问题逐渐暴露出来。

优化前

随着使用时间和数据量的增加，集群开始频繁出现不可用的问题，主要体现在以下几个方面：

- DDL 操作很难执行，查询速度变得比较缓慢。
- FE 服务频繁出现 OOM 宕机，有时候甚至出现无法连接的情况。

下图是生产环境某张表的体积的大小和 Tablet 数量的情况。这张表的体积只有 275M，但是 Tablet 的数量却达到了 7410，这非常不合理。进一步排查确认整个集群 Tablet 数量非常庞大，集群只有 5T 的数据量，然而 Tablet 数量达到 150 万。



最初我们对 Apache Doris 表数据量大小、分区粒度、Bucket 数量、Tablet 数量的关系及 Tablet 数量对集群的影响没有清晰的概念。开发人员在 Apache Doris 使用中更多的是追求查询速度，将大部分的动态分区表的分区粒度设置的比较小，分区 Bucket 数量设置却比较大。
经过与 Apache Doris 社区小伙伴的沟通交流，了解到 Tablet 数量过大可能会导致元数据管理和运维压力增大，出现查询速度缓慢、FE 不稳定的问题。

优化方案

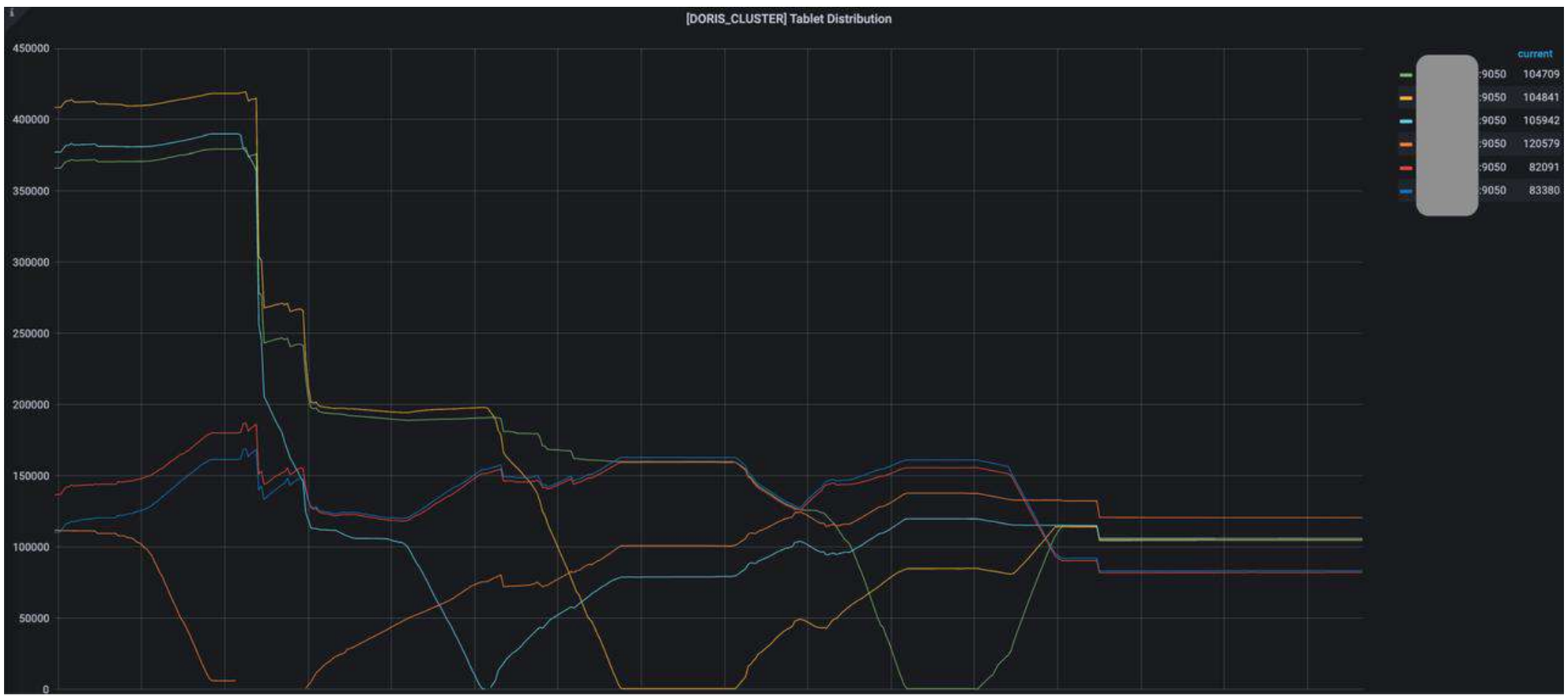
首先明确 Tablet 数量的计算方式，Tablet 数量 = 分区数量 * Bucket 数量 * 副本数。结合当前实际使用情况，确认可以在分区粒度和 Bucket 数量上进行优化。我们将分区的粒度从按天、按周分区更改为按月分区，Bucket 数量按照数据体积大小进行合理的配置。如下图所示，是建议数据体积大小对应的 Bucket 数量设定。

数据大小	Bucket 数量
0MB ~ 10MB	1
10MB ~ 50MB	2
50MB ~ 2GB	4
2GB ~ 5GB	8
5GB ~ 25GB	16
25GB ~ 50GB	32
> 50GB	64

本次的优化目标是将 Tablet 数量从 150 万降低到 15 万，同时我们也对未来的增长速度进行了规划，在三副本情况下，期望 Tablet 数量增长速度是 30000/TB。

优化后

实际上，在仅对 ODS 表进行了分区粒度和 Bucket 数量调整后，集群 Tablet 数量从 150万下降到了 50 万，效果显著。

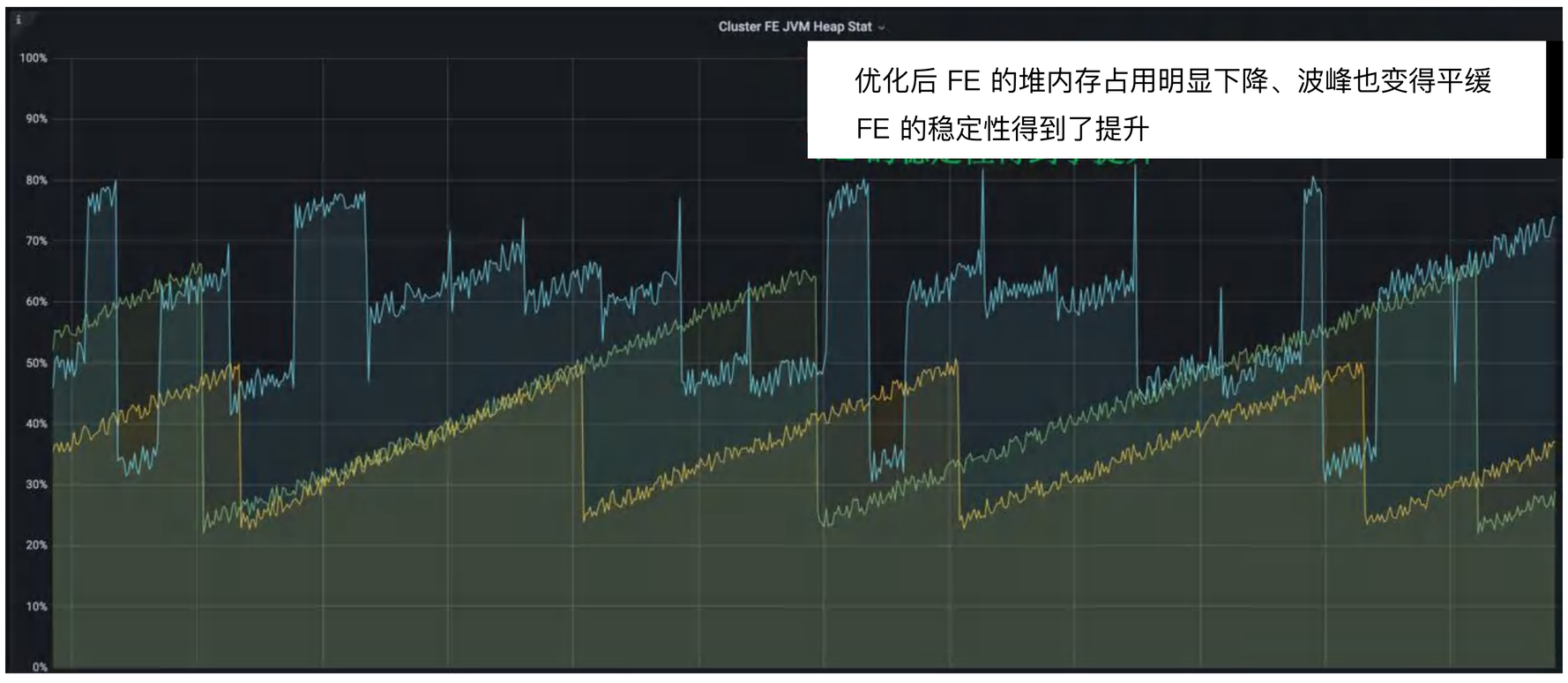


优化前的 FE

下图是 FE JVM Heap Stat 的监控情况，每当 FE 执行 Checkpoint 时，元数据就会在内存中复制一份。体现在 FE JVM Heap Stat 上就是形成一个个的波峰。优化之前 FE 对内存占用几乎持续在 90% 以上，而且每一个波峰都非常的尖锐。



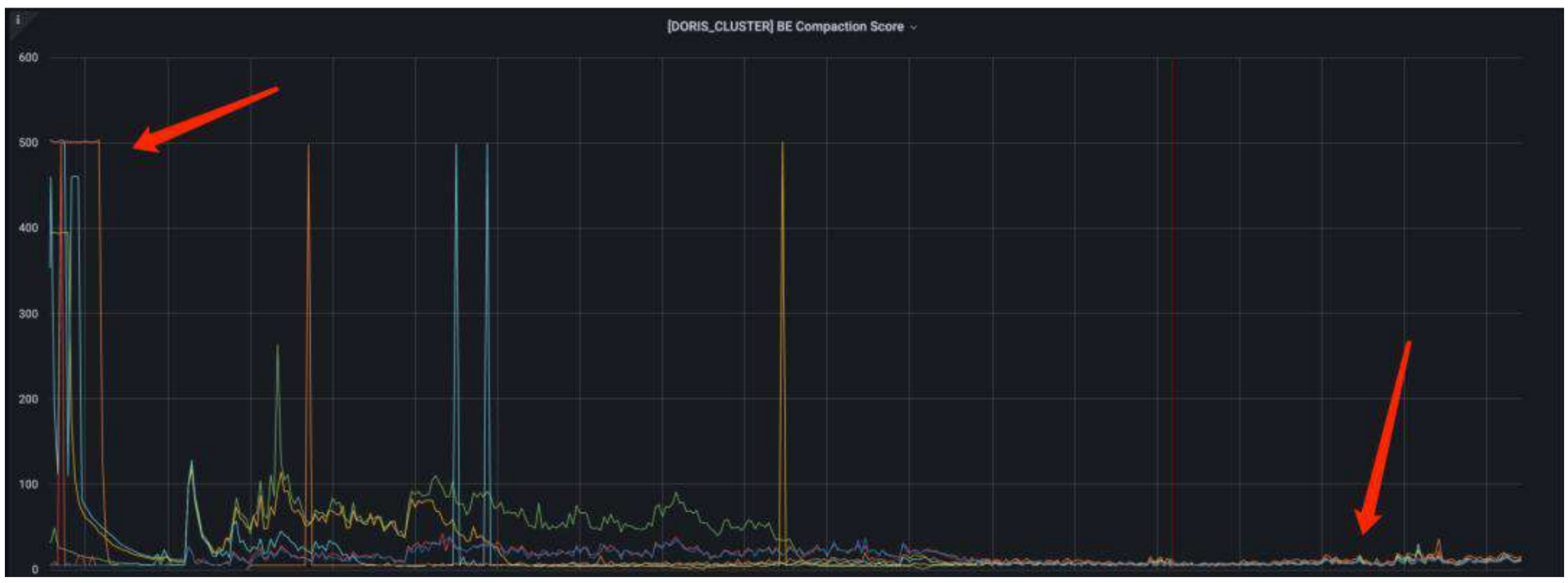
优化后的 FE



优化后，FE 堆内存占用明显下降，波峰也变得很平缓。FE 的稳定性得到了比较明显的提升。

优化前、后对 BE 的影响

BE Compaction Score 监控反映版本的堆积情况，版本堆积的数值在 100 内属于正常范围，超过 100 说明集群可能存在潜在风险。上文也讲到，查询时需要先进行文件合并，再进行数据查询，如果 Tablet 版本过多，版本合并会影响到查询的速度和稳定性。



经过磁盘的部署优化和 Tablet 优化后，BE Compaction Score 可以稳定在50 以内，查询的稳定性和性能都得到了非常大的提升。

数据同步优化

优化前

MySQL 数据同步使用 Flink CDC -> Kafka -> Flink Doris Connector -> Doris 的方式全量 + 增量进入 Apache Doris。

在这个方案中，虽然 Flink CDC 支持全量历史数据的初始化，但由于历史遗留问题，部分表数据量较大，单表有几亿数据，而且这种表大多是没有设置任何分区和索引，在执行简单的 COUNT 查询时都需要花费十几分钟的时间。

其次，Flink CDC 虽然可以进行增量数据同步，但对于这类表的全量数据初始化几乎是不能实现的，因为 Flink CDC 做全量同步要先读取全量数据，然后对数据分块，再做数据同步，这种情况下，读取是非常非常缓慢的。

优化后

针对这种情况，在数据同步上，我们做了以下优化：

- 全量同步使用 MySQL Dump -> CSV -> Doris Stream Load -> Doris
- 增量同步使用 Flink CDC -> Kafka -> Flink Doris Connector -> Doris

数据调度优化

我们在使用 DolphinScheduler 进行 Doris SQL 的任务调度时，同一 node 下配置多条 SQL 时会出现 node 执行状态异常的情况，导致工作流 DAG 的 node 依赖失效，前一个节点未执行完，后一个节点就开始执行，结果会有缺数据甚至没有数据的情况。这个问题是因为 DolphinScheduler 2.x 在同一个 node 下不支持按顺序执行 MySQL 的多段 SQL，而 Doris 在 DolphinScheduler 中使用 MySQL 数据源创建连接。

此问题在 DolphinScheduler 3.0.0 版本被修复，配置中可以设置多段 SQL 的分隔符，解决了 DAG 依赖关系失效的问题。

Apache Doris 元数据管理和数据血缘实现方案

在没有元数据管理和数据血缘之前，我们经常会遇到一些问题，比如想找一个指标，却不知道指标在哪张表，只能找相关开发人员来确认，当然也存在开发人员忘记指标的位置和逻辑的情况。因此只能通过层层筛选确认，此过程十分耗费时间。

之前我们将表的分层划域、指标口径、负责人等信息放在 Excel 表中，这种维护方式很难保证其完整性，维护起来也比较困难。当需要对数仓进行优化时，无法确认哪些表是可以复用的、哪些表是可以合并的。当需要对表结构进行变更时，或者需要对指标的逻辑进行修改时，也无法确定变更是否会对下游的报表产生影响。

在以上问题背景下，我们经常遭到用户的投诉，接下来介绍如何通过元数据管理和数据血缘分析方案来解决这些问题。

实现方案

物理元数据	业务元数据	数据血缘
Schema 信息	分层	表血缘
数据量	主题域	字段血缘
存储空间占用大小	负责人	数仓影响分析
Tablet 数量	指标信息	报表影响分析
分区信息	表的使用规则	跨层引用分析
生命周期		引用热度分析
DDL 变更		
读写记录		
产出信息		
产出脚本		

元数据管理和数据血缘是围绕 Apache Doris 展开，同时对 DolphinScheduler 的元数据进行了整合。上图中，右侧表格是物理元数据业务，元数据指标和血缘分析能够提供的数据服务。

我们将元数据分为物理元数据和业务元数据两大类：

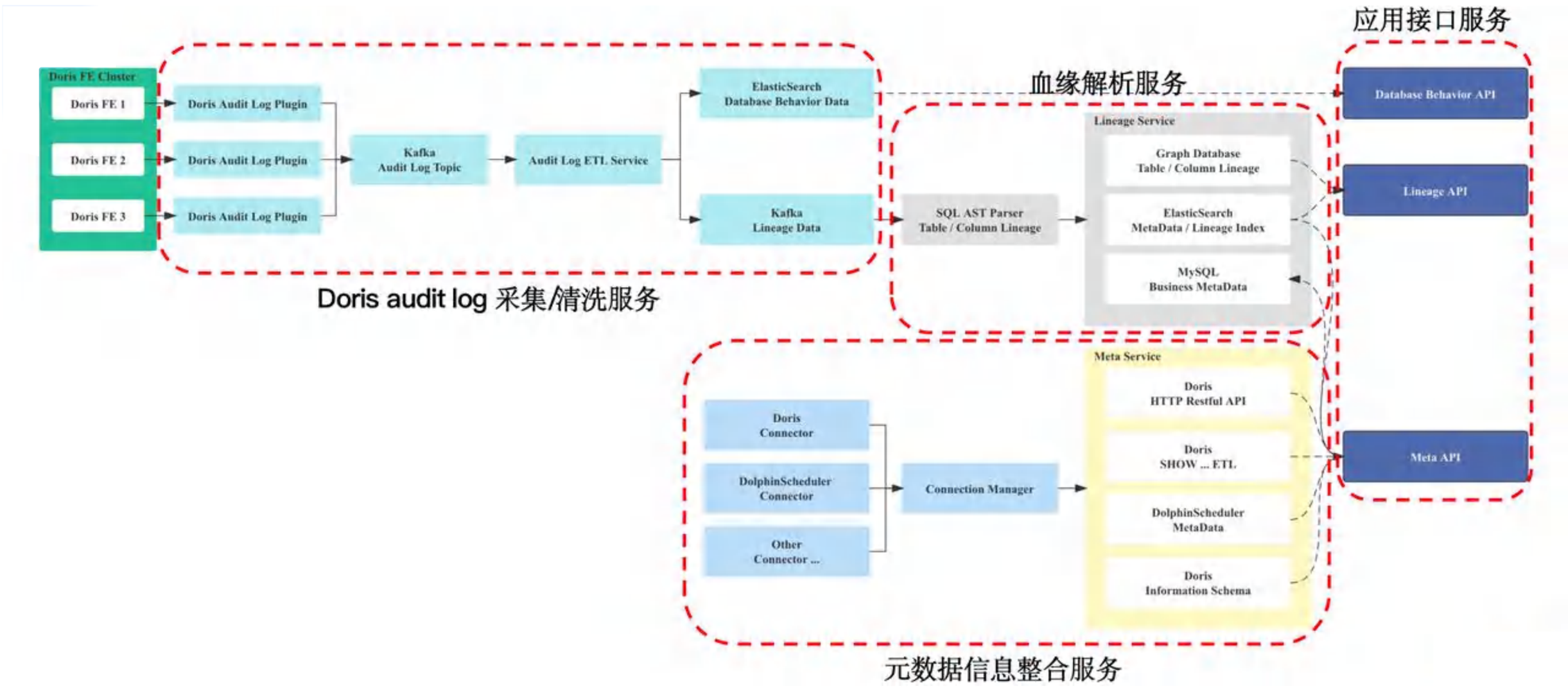
- 物理元数据维护表的属性信息和调度信息
- 业务元数据维护数据在应用过程中约定的口径和规范信息

数据血缘实现了表级血缘和字段级血缘：

- 表级血缘支持粗粒度表关系和跨层引用分析
- 字段级血缘支持细粒度的影响分析

接下来介绍元数据管理和数据血缘的架构和工作原理

架构介绍



元数据管理和数据血缘实现方案技术栈

- 数据采集：使用 Apache Doris 提供的审计日志插件 Doris Audit Plugin 进行数据采集
- 数据存储：对审计日志插件做了定制化开发，使用 Kafka 存储 Doris 审计日志数据
- 血缘解析：使用 Druid 进行 Doris SQL 解析
- 血缘关系存储：使用 Nebula Graph 存储血缘关系数据
- 业务元数据：因为业务元数据经常发生 CRUD，因此使用 MySQL存储业务元数据信息
- 搜索数据：使用 ElasticSearch 存储血缘关系查询索引以及表和字段的搜索索引数据

接下来介绍一下个架构四个组成部分：审计日志的采集和清洗服务、血缘解析服务、元数据信息整合服务、应用接口服务。

Apache Doris 审计日志的采集/清洗服务

考虑到如果将数据清洗逻辑放在审计日志插件中，当数据清洗逻辑发生变更，可能会出现数据遗漏，这样会对血缘分析和元数据管理产生影响，所以我们将审计日志插件数据采集和数据清洗进行了解耦，对 Apache Doris 的审计日志插件进行了改造，改造后审计日志插件可以实现审计日志数据的格式化以及将数据发送到 Kafka 的功能。

数据清洗服务，首先在清洗逻辑中增加数据重排逻辑，针对多个审计日志插件发送的数据进行重

新排序，解决数据乱序的问题。其次把非标准 SQL 转化成标准 SQL，虽然 Apache Doris 支持 MySQL 协议以及标准 SQL 语法，但有一些建表语句、SQL 查询语法与标准 SQL 存在一定差异，因此将非标准 SQL 转化为 MySQL 的标准语句，最后将数据发送到 ES 和 Kafka 中。

血缘解析服务

血缘解析服务使用 Druid 进行 Doris SQL 的解析，通过 Druid 抽象语法树逐层递归获取表和字段的血缘关系，最后将血缘关系数据封装发送到图数据库、血缘查询索引发送到 ES 。进行血缘解析的同时会将物理元数据和业务元数据发送到对应存储位置。

元数据信息整合服务

元数据信息整合服务借鉴了 Metacat 的架构实现方案。

- Connector Manager 负责创建 Apache Doris 和 DolphinScheduler 的元数据链接，同时也支持后续其他类型数据源接入的扩展。
- Meta Service 负责元数据信息获取的具体实现。Apache Doris 元数据信息主要从 Information Schema 库、Restful API、以及 SHOW SQL 的查询结果三种途径来获取。DolphinScheduler 工作流元数据信息和调度记录信息从 DolphinScheduler 元数据库获取。

应用接口服务

我们提供了 3 种类型的应用接口服务，分别是血缘应用接口服务、元数据应用接口服务和数据行为分析应用接口服务。

- 血缘应用接口服务提供表、字段、血缘关系、影响分析的查询服务。
- 元数据应用接口服务提供元数据的查询和字段搜索的服务。
- 数据行为分析应用接口服务提供表结构变更记录、数据读写记录、产出信息的查询服务。

以上就是元数据管理和数据血缘分析架构的整体方案的全部内容介绍。

总结及收益

今年我们完成了以 Apache Doris 为核心的准实时数仓建设，Apache Doris 经过半年的使用和优化，现在已经趋于稳定，能够满足我们生产的要求。

1. 新的准实时数仓，对数据计算效率、数据时效的提升是巨大的。以 On Time Delivery 业务场景报表计算为例，计算 1000w 单轨迹节点时效变化，使用 Apache Doris 之前需要计算 2 个多小时，并且计算消耗的资源非常大，只能在空闲时段进行错峰计算；使用 Apache Doris 之后，只需要 3 分钟就可以完成计算，之前每周更新一次的全链路物流时效报表，现在可以做到**每 10 分钟更新最新的数据**，达到了**准实时的数据时效**。

2. 得益于 Apache Doris 的标准化 SQL，上手难度小，学习成本低，报表的迁移工作全员都可以参与。原来报表使用 PowerBI 进行开发，需要对 PowerBI 有非常深入的了解，学习成本很高，开发周期也很长，而且 PowerBI 不使用标准 SQL，代码可读性差；现在基于 Doris SQL 加上自研的拖拉拽形式的报表平台，报表的开发成本直线下降，大部分需求的开发周期从周下降到了天。

未来规划

后续我们也将继续推进基于 Apache Doris 的数据中台建设，对元数据管理的完善、数据血缘的解析率持续进行优化，考虑到数据血缘是大家都渴望的应用，在未来血缘解析成熟后，我们会考虑将其贡献给社区。

与此同时，我们正在着手进行用户行为分析平台的构建，也在考虑使用 Apache Doris 作为核心的存储和计算引擎。目前 Apache Doris 在部分分析场景支持的函数还不够丰富，例如在有序窗口漏斗分析场景，虽然 Apache Doris 已经支持了 window_funnel 函数，但是每层漏斗转化的计算需要用到的 Array 相关计算函数还没有得到很好的支持。不过好在即将发布的 Apache Doris 1.2 版本将包含了 Array 类型以及相关函数，相信未来在越来越多的分析场景中 Apache Doris 都将得到落地。

叮咚买菜基于 Apache Doris 统一 OLAP 引擎的应用实践

作者：叮咚买菜资深数据工程师，韩青

导读：随着叮咚买菜业务的发展，不同的业务场景对数据分析提出了不同的需求，他们希望引入一款实时 OLAP 数据库，构建一个灵活的多维实时查询和分析的平台，统一数据的接入和查询方案，解决各业务线对数据高效实时查询和精细化运营的需求。经过调研选型，最终引入 Apache Doris 作为最终的 OLAP 分析引擎，Doris 作为核心的 OLAP 引擎支持复杂地分析操作、提供多维的数据视图，在叮咚买菜数十个业务场景中广泛应用。

叮咚买菜创立于 2017 年 5 月，是一家专注美好食物的创业公司。叮咚买菜专注吃的事业，为满足更多人“想吃什么”而努力，通过美好食材的供应、美好滋味的开发以及美食品牌的孵化，不断为人们提供美好生活的解决方案，致力让更多人吃得新鲜、吃得省心、吃得丰富、吃得健康……以更美好的舌尖体验，为现代家庭创造美味与幸福感。

业务需求

随着叮咚买菜业务的发展，不同的业务场景对数据分析提出了不同的需求，这些需求最终被数据看板、实时 Ad-Hoc、行为分析、B/C 端业务平台和标签平台等系统应用所承载，为实现这些系统应用，叮咚大数据希望引入一款实时 OLAP 数据库，旨在提供一个灵活的多维实时查询和分析的平台，统一数据的接入和查询方案，解决各业务线对数据高效实时查询和精细化运营的需求。基于上述诉求，我们希望所引入的数据库具备以下能力：

- 可以实时高效地分析和使用数据；
- 可以支持明细数据、汇总数据两种不同的数据查询方式；
- 可以对入库后的数据即席选择维度和条件查询，实时/近实时返回结果

选型与对比

我们主要对比了 Apache Doris 和 ClickHouse 两款市面上最常见的开源 OLAP 引擎，在选型过程中我们主要考虑以下几个方面：

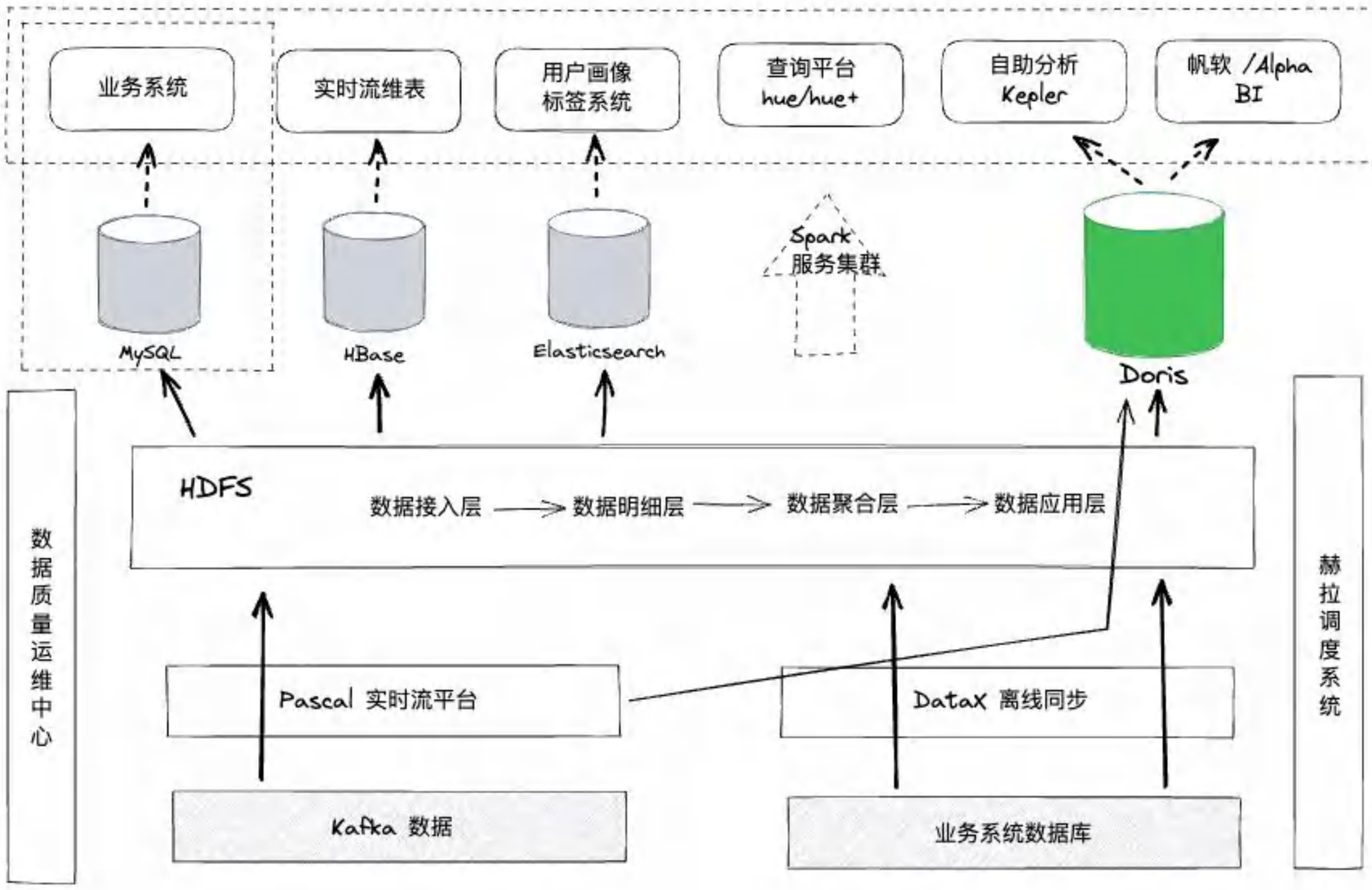
- 支持标准 SQL，无需投入额外的时间适应和学习新的 SQL 方言、直接用标准 SQL 即可直接查询，最大化降低使用门槛；
- 支持 Join 操作，方便事实表与维度表进行关联查询，在应对维度更新时具有更高的灵活性、无需对处理后的宽表进行重刷；
- 支持高并发查询，系统面临多条业务线的同时使用，因此需要有比较强的并行查询能力，以更好满足业务需求；
- 支持离线和实时导入，可与已有技术栈轻松对接，支持多个数据源或大数据组件的离线和实时导入，以更好适配不同使用场景；
- 支持大数据量的明细数据查询，以满足不同业务用户灵活多变的分析需求；

经过详尽的技术调研，Apache Doris 各项能力都比较优异，在我们的大多数业务场景中都需要明细数据级别的查询、高并发的点查和大数据量的 Join，而这几个方面 Apache Doris 相较于 ClickHouse 均更胜一筹，因此我们决定使用 Apache Doris 来搭建新的架构体系。

架构体系

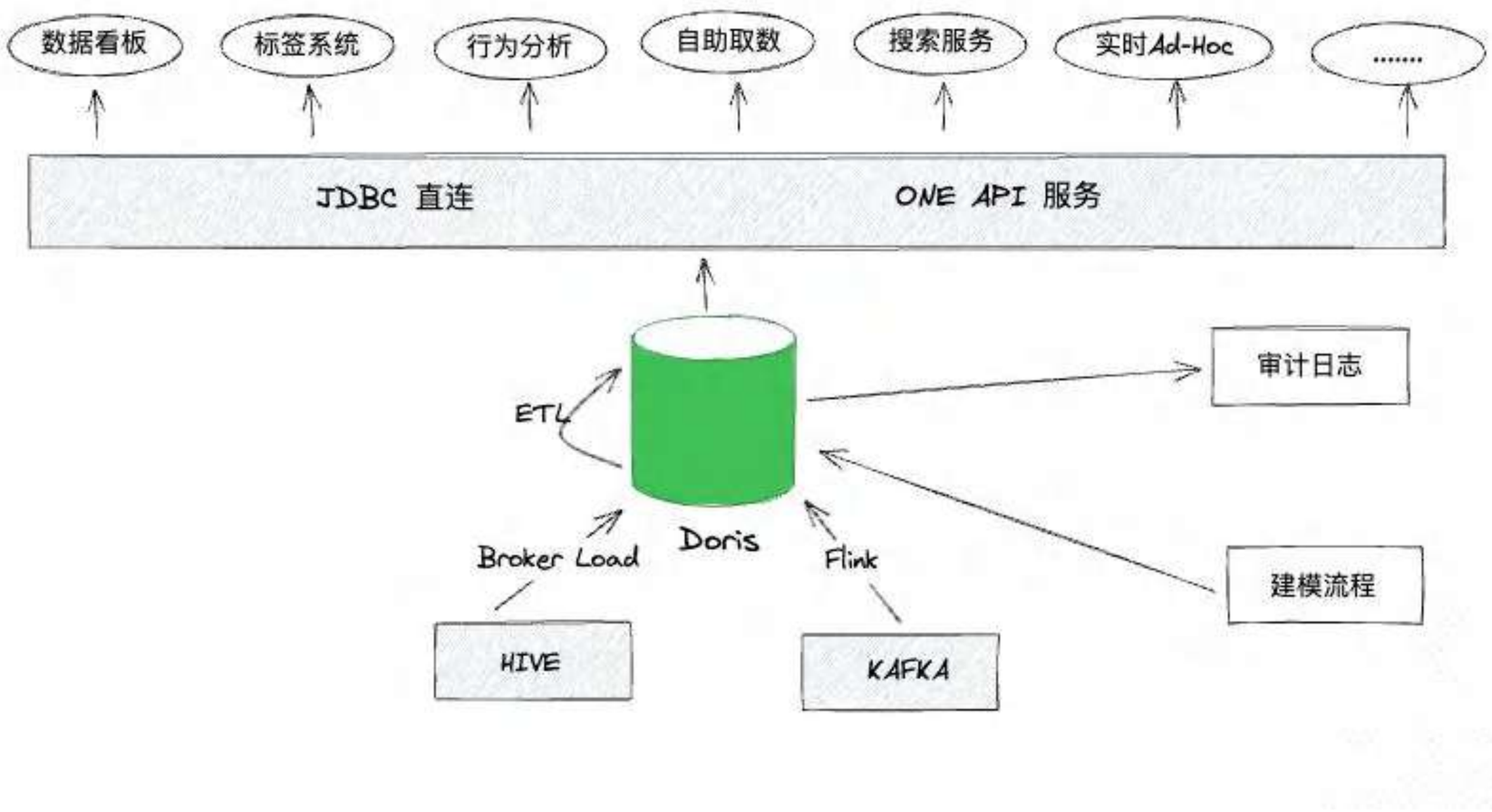
在整体架构中，各个组件承载着特定的功能，Elasticsearch 主要负责存储标签系统的数据，HBase 是实时数仓的维表层，MySQL 用于存储业务系统的数据存储，Kafka 主要存储实时数据，Spark 主要提供 Ad-Hoc 查询的计算集群服务，而 Apache Doris 作为核心的 OLAP 引擎支持复杂地分析操作、提供多维的数据视图。

- **离线部分：**数据从业务库通过 DataX 导入到数据仓库 ODS 层，经过层层处理输出到 Doris 中，上层 BI 系统链接到 Doris 进行报表展示。
- **实时部分：**数据通过 Flink 消费 Kafka 中的数据，进行相应的逻辑处理后，输出到 Doris 或者 HDFS 中，供应用层使用。



注：实线箭头表示真实数据流，虚线箭头表示访问数据流

在数据应用的 OLAP 层中，Doris 应用方案如下图所示：



- **模型创建规范化**：采用流程审批的方式进行数据建模，根据具体的业务场景来搭建 Duplicate，Unique Key 和 Aggregate 模型，并按照用户提供的数据量设置合适的 Bucket 数目，做好模型归属关系。

- **数据入口的统一：**数据的流入主要有实时和离线两种，实时数据用 Flink 任务从 Kafka 消费数据，逻辑处理流入 Doris ； 离线数据通过 Broker Load 方式从 Hive 中将数据灌入 Doris 中。

- **服务对外出口的统一：**对外服务主要通过两种方式暴露接口，一种是使用 JDBC 直连，上层系统配置 Doris 集群的 FE 的连接信息直连 Doris；另一种是业务通过内部的 One API 服务配置业务接口使用 Doris。

- **业务 SQL 的优化治理：** 通过采集 Doris FE 的审计日志，以此来对 SQL 的性能进行分析优化，以及对 Doris 服务进行治理。

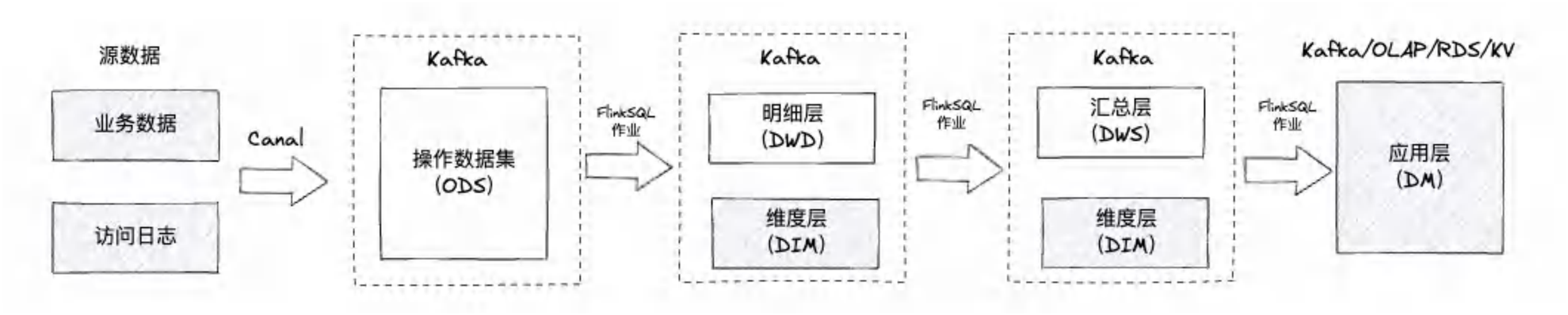
应用实践

叮咚买菜目前已经将 OLAP 引擎统一为 Apache Doris 广泛应用于业务场景中，我们将 Doris 集群拆分成了四个集群，分别支持核心报表、行为分析与算法应用、B/C 端业务和实时数据，根据不同业务场景的业务量及数据规模，集群的资源配置也不尽相同。目前总的机器规模达数十台，以行为分析场景为例，**单个集群近 20 个节点、存储数据量超过百 TB，每日新增数据量达数十亿条。**

接下来分享 Apache Doris 在叮咚买菜常见业务场景中的应用实践及使用经验。

实时数据分析

从下方数仓模型图可知，数据通过 Flink 作业进行逻辑处理，在不同层级 Kafka 中进行流转加工，经过数据汇总层之后，应用层需要一个组件来存储结果数据，该组件一般是从 MySQL 数据库、KV 存储和 OLAP 引擎三者中选择其一。



考虑到我们的结果数据大多以计算指标数据居多，缺乏维度数据，因此应用层的组件需要具备高效、低延迟的数据 Join 能力。基于以上因素，我们最终选择 Apache Doris 作为实时数仓和实时业务的数据应用层，**Doris 可以有效降低数据处理延时，提高查询效率。**

比如在销量计划项目中，该项目需要每日实时写入大量预测数据，并且这些数据需要较低时延提供给分析师进行及时对比分析、修改该预测值，并提供到供应链端。因修改预测值会影响到系统调拨，所以选用的存储必须是要具有高吞吐低延迟特性，Doris 完全符合该要求。从销量计划的整个数据生产及处理链路来看，使用 Doris 后，**最慢 2 秒内**可以看到最新的数据。

当前公司已经有**数十个实时业务**需求接入到 Doris 中，随着业务的不断发展，这个数字也在慢慢增加。

B 端业务查询取数

在实际的使用中，通常会有 B 端业务或系统需要从数据仓库中取数的需求，比如自研 Pylon 系统（主要用来基于用户偏好的数据查询）会根据 UID 查询用户画像信息。在这种场景下，通常需要进行复杂的模型关联，同时要求在秒级或者毫秒级返回查询结果。

- 使用前：我们最初使用 Spark 的 JDBC 方式来直接查询数据仓库 Hive 表数据，由于存放用户标签数据的 Hive 表的数据量有几千万体量，通过 Spark JDBC 方式要耗费几分钟才能查出结果，同时还存在 Yarn 调度耗时较高的问题，有时会因为队列资源紧张产生延迟，导致一个普通的查询甚至需要十几分钟才能跑出结果，用户的体验度非常不好。
- 使用后：经过我们对数据链路的改造，将 Hive 的用户标签数据离线灌入 Doris 中，再用同样 SQL 查询，Doris 性能在绝大多数场景下比 Spark 要好很多，可以在**秒级别**得到返回结果。

标签系统

最初我们的标签数据存放在 ES 中，但是随着业务的扩展、下游业务越来越多，标签数据规模急速膨胀，策略规则不断增加变化，标签系统遭遇严重的性能瓶颈。

- 聚合和 Join 查询的性能低
- 人群圈选花费时间近 20 分钟
- ES 导入慢、查询性能低

为解决以上问题，我们目前正在尝试使用 Doris 来替换 ES，希望通过 Doris 解决上述问题，选择 Doris 主要考虑以下三点：

1、分布式 Join 大大提升查询效率

原有商品 ID 和仓库 ID 通过嵌套类型存储在 ES 中，替换为 Doris 存储之后，需要将复杂的嵌套类型拆解为两张表来做表级关联，同时可以试用 Doris 的多种分布式的 Join 提高查询得性能。Doris 的分布式 Join 主要有 Shuffle Join、Broadcast Join 和 Colocate Join。

其中 Colocate Join 查询性能是最好的，旨在为某些 Join 查询提供本地性优化，来减少数据在节点间的传输耗时、加速查询，另外我们在该场景下基本均为千万级的表。综合来看，Colocate Join 比较符合场景与需求，最终决定使用 Colocate Join 方式提升 Join 性能。

如何使用：标签数据的使用主要涉及到两张大表的 Join，建表时需要设置相同的 Distributed Key、相同的 Bucket 数、相同的副本数，还要将两个表通过 `colocate_with` 属性划分到一个组 Colocation Group(CG)。

```
 CREATE TABLE `profile_table` (
`pdate` date NULL COMMENT "null",
`product_mongo_id` varchar(4000) NULL COMMENT "商品ID",
`station_id` varchar(4000) NULL COMMENT "仓id",
......
) ENGINE=OLAP
UNIQUE KEY(`pdate`,
`product_mongo_id`, `station_id`)
COMMENT "OLAP"
PARTITION BY RANGE(`pdate`)()
DISTRIBUTED BY
HASH(`product_mongo_id`) BUCKETS 7
PROPERTIES ("colocate_with" = "profile_table","in_memory" = "false","storage_format" = "V2")
```

```
CREATE TABLE
`station_info_table` ( `product_mongo_id` varchar(4000) NULL COMMENT "商品id", `station_id` varchar(4000)NULL
COMMENT "站点id",
`snapshot` date NULL COMMENT "日期",
`product_id` bigint(20) NULL COMMENT "商品id", ......)
ENGINE=OLAPUNIQUE KEY(`product_mongo_id`, `station_id`, `snapshot`)
COMMENT "OLAP"
PARTITION BY RANGE(`snapshot`)()
DISTRIBUTED BY
HASH(`product_mongo_id`) BUCKETS 7
PROPERTIES ("colocate_with" = "profile_table","in_memory" = "false","storage_format" = "V2")
```

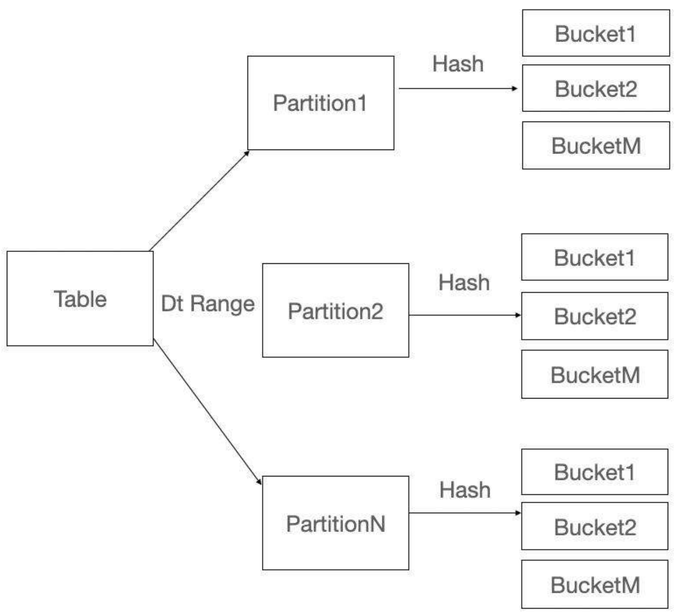
比如我们有这样一条查询语句：

```
select count(psp.product_mongo_id) from profile_table psp
left join station_info_table psi on psp.product_mongo_id=psi.product_mongo_id and psp.station_id=psi.station_id
where psp.pdate='2023-03-16' and psp.four_category='特色醋' and psp.brand_name='宝鼎天鱼'and
psp.weight_unit='ml' and psp.pmg_name='粮油调味组';
```

经过使用Colocate Join 方式优化后，可以达到毫秒级的查询效果。接下来我们介绍一下 Colocate Join 的查询性能高的原因有哪些呢?

A. 数据导入时保证数据本地性

Doris 的分区方式如下所示，先根据分区字段 Range 分区，再根据指定的 Distributed Key Hash 分桶。



所以我们在数据导入时保证本地性的核心思想就是两次映射，对于 Colocate Tables，我们保证相同 Distributed Key 的数据映射到相同的 Bucket Seq，再保证相同 Bucket Seq 的 Buckets 映射到相同的 BE。可以同查看执行计划查看是否使用了 Colocate Join ：

```
| Explain String                                                
```

对于 HashJoinFragment，由于 Join 的多张表有了数据本地性保证，所以可以去掉 Exchange Node，避免网络传输，将 ScanNode 直接设置为 Hash Join Node 的 Child。

B. 查询调度时保证数据本地性

- 查询调度的目标：一个 Colocate Join 中所有 ScanNode 中所有 Bucket Seq 相同的 Buckets 被调度到同一个 BE。
- 查询调度的策略：第一个 ScanNode 的 Buckets 随机选择 BE，其余的 ScanNode 和第一个 ScanNode 保持一致。

C. 数据 Balance 后保证数据本地性

新增一个 Daemon 线程专门处理 Colocate Table 的 Balance，并让正常的 Balance 线程不处理 Colocate Table 的 Balance。正常 Balance 的粒度是 Bucket，但是对于 Colocate Table，必须保证同一个 Colocate Group 下所有 Bucket 的数据本地性，所以 Balance 的单位是 Colocate Group。

2、高效简易的 `array_contains` 函数

在做人群圈选时，有以下类似的 Json 结构 `[{"K1":"V1","K2":200}` , `{"k1":"v2","k2":300}]` , 当配置 `k1=v1,k2=200` ，只要该 Value 里的 Json 项有一项满足全部条件就会被圈出来，我们可以借助 Doris 1.2 版本中的 `array_contains` 数组函数处理，将 Json 转化为 Array 数组处理。

```
MySQL [(none)]> SELECT array_contains(["'K1':'V1'","'K2':200","'k1':'v2'","'k2':300"], "'k2':300");
+-----------------------------------------------------------------------------------------------+
| array_contains(ARRAY("'K1':'V1'", 'K2':200', 'k1':'v2', 'k2':300'), 'k2':300') |
+-----------------------------------------------------------------------------------------------+
|                                                                                             1 |
+-----------------------------------------------------------------------------------------------+
1 row in set (0.01 sec)
```

3、Broker Load 加速数据导入效率

Doris Broker Load 是一种高效、稳定的数据导入方式，它可以将数据分成多个分片，然后将每个分片分配给不同的 Broker 节点进行处理，我们使用 Broker Load 将标签数据从 Hive 导入 Doris 中，有效提高了数据导入的效率和稳定性。

BI 看板

我们商业智能分析使用的 BI 分析平台主要是帆软和自研的阿尔法 BI，底层使用 Doris 来存储数据，目前用来做报表的 Doris 表数量已达到了 **3000 多张**，四个 Doris 集群的日 **UV 1000+**，**PV 达到十几万**，由于 Doris 可以达到毫秒级响应速度、支持高并发查询，因此单集群的 QPS 可以达到达到 **120 次/秒**，符合我们的要求。

OLAP 多维分析

随着业务的增长，我们在运营的过程中我们常常有一些疑问：最近三个月哪个品类的下单量最高？变化趋势如何？各个时段人均下单是多少？某个区域，发生购买行为的年龄段分布是怎样的？.....而想要获得结果，必须根据用户行为日志进行事件分析。

目前我们的用户行为数据日均增量为 20亿+，高峰期 100亿+，为了更好的进行事件分析，我们需要保留半年的数据，也就是几千亿的数据量。我们使用 Doris 来存储如此庞大的数据量，在应对复杂的分析场景时可以达到分钟级的响应。在多维分析的过程中， 往往也伴随着大数据量的复杂查询，接下来分享如何使用 Doris 应对：

1、 Bitmap 去重

业务使用过程中需要分析用户参与情况以及活跃程度，考查进行初始行为后的用户中，有多少人会进行后续行为，这时候一般都是使用留存分析模型实现此类需求。该模型使用中有去重操作，计算周期有某天/某周/某月/最近三个月等，由于每天的埋点数据量都能达到几十亿，高峰期 100 亿，在这个情况下，使用 `count(distinct)` 性能太差、甚至查询超时（或超过设置的时间），**而如果使用 Bitmap 来可以成倍的缩短查询时间**。

```
select
event_id,
date,
count(distinct uid) as count
from event
where
dt>='2022-06-01' and dt<'2022-06-06' and event_id in (......) group by event_id, str_to_date(dt,'%Y-%m-%d');
```

使用 Bitmap 优化 SQL 后

```
select
event_id,
date,
bitmap_count(uid) as count
from event
where
dt>='2022-06-01' and dt<'2022-06-06' and event_id in (......) group by event_id, str_to_date(dt,'%Y-%m-%d');
```

使用中需要注意 Bitmap 函数在 Apache Doris 中仍然需要先把数据汇聚到一个 FE 节点才能执行计算，并不能充分发挥分布式计算的优势，在数据量大到一定的情况下， Bitmap 函数并不能获得比 `count(distinct)` 更好的性能，上述实例之所以能达到预期结果是由于做了分组计算。

如果处理大数据量的全量去重，在建表时将 Bitmap 列的值按照 Range 划分，不同 Range 的值存储在不同的分桶中，保证了不同分桶的 Bitmap 值是正交的。当查询时，先分别对不同分桶中的正交 Bitmap 进行聚合计算，然后顶层节点直接将聚合计算后的值合并汇总并输出，从而解决顶层单节点计算瓶颈问题。

2、前缀索引和 Bloom Filter 索引

Doris 主要支持两类索引：内建的智能索引（包括前缀索引）和创建的二级索引（包括 Bloom Filter 索引和 Bitmap 倒排索引）。实际使用时我们会用到前缀索引和 Bloom Filter 索引来提高查询效率。

前缀索引

Aggregate、 Unique 和 Duplicate 三种数据模型中，底层的数据存储是按照各自建表语句中 AGGREGATE KEY、UNIQUE KEY 和 DUPLICATE KEY 指定的列进行排序存储的。前缀索引即在排序的基础上实现的一种根据给定前缀列、快速查询数据的索引方式，实现方式是将一行数据的前 36 个字节作为这行数据的前缀索引，当遇到 VARCHAR 类型时，前缀索引会直接截断。

比如我们要查询按照日期和 `event_id` 分组的去重人数，建表语句如下：

```
CREATE TABLE ubs_event_log_small_event (
event_id int(11) NULL COMMENT "事件id",
dt datetime NOT NULL COMMENT "事件时间",
uid char(128) NULL COMMENT "用户id",
dict_id int(11) NULL COMMENT "用户id字典值",
os varchar(24) NULL COMMENT "操作系统",
......
dict_id_bitmap bitmap BITMAP_UNION NULL COMMENT "bitmap用户id"
) ENGINE=OLAP
AGGREGATE KEY(event_id, dt, uid, dict_id, os, ......)
COMMENT "用户行为事件表"
PARTITION BY RANGE(dt)
()
DISTRIBUTED BY HASH(dt, event_id, uid) BUCKETS 64
```

SQL 查询的 Where 条件一般遵循建表的 AGGREGATE 模型的 KEY 的顺序，这样可以命中 Doris 内置的前缀索引。

```
SELECT
CONCAT(
TO_DATE(dt),
' 00:00:00'
) AS tm,
event_id,
BITMAP_UNION_COUNT(dict_id_bitmap) AS UNIQ_1908_1
FROM
kepler.ubs_event_log_small_event
WHERE event_id = 1908 AND
dt >= '2023-03-26'
AND dt < '2023-04-05'
AND
os IN (1, 2)
GROUP BY
1,
2;
```

Bloom Filter 索引

针对大数据量的事件表进行查询时我们会设置 `bloom_filter_columns` ，加快查询效率：

```
alter table datasets set("bloom_filter_columns" = "area_name, is_booking, user_source, source_first_order......");
```

查询语句中 `where` 条件有以上设置的字段就会命中该索引

```
SELECT * FROM datasets WHERE area_name="****" AND is_booking=0
```

3、物化视图

为了获得更粗粒度的聚合数据，Doris 允许在建表语句创建出来的 Base 表的基础上，创建若干 Rollup 表。

例如上表 `ubs_event_log_small_event` ，我们可以对 `dtevent_iddict_id_bitmap` 建立 Rollup 物化视图，这样 Rollup 只包含三列： `dt` `event_id` `dict_id_bitmap`

dt	event_id	dict_id_bitmap
2023/1/27 0:00	534	NULL
2023/1/27 0:00	534	NULL
2023/1/27 0:00	533	NULL

这时再进行上述查询就会命中这个 Rollup，从而只扫描极少的数据量就可以完成此次聚合查询。

优化经验

Broker Load 导数任务流程化

为了 Doris 使用更加便捷，我司在内部自研的叮咚大数据平台上对整个过程进行流程化；从建模到配置 Broker Load 导数任务再到导数任务调度均进行了调整，具体优化如下所述：

1.建模过程：需要用户发起建模流程申请，填写需求内容、具体建模语句、预估数据量大小、数据保留时长、所需相关权限账号等信息，足够完整的信息可以在审批时获得建模过程中的元数据信息以及选择更合适的数据模型



2.Broker Load 导数任务配置：为了提高用户使用效率、降低使用门槛，我们通过 Mapping 映射和自动化配置方式，自动生成导数任务。



3.导数任务调度：配置完 Broker Load 导数任务，就可以由用户根据需求配置小时或者天级别的调度，这样整个 Doris 数据导入流程，就可以由用户配置自动完成。

任务ID：		调度类型：	定时任务	管理员：	
任务名称：		定时表达式：	0 40 10 * * 7	责任人：	
任务描述：		调度状态：		关注人：	[]
任务类型：	Shell	重试次数：	0	前置检查ID：	-
任务优先级：	low	重试间隔：	1	后置检查id：	-
并行执行：	否	机器组：	默认组	后置检查阻断：	否

配置项信息

```
1
```

脚本

```
1
2  ##--*****************  ***************************************************************
3  ##-- ** 所属数据
4  ##-- ** 功能描述
5  ##-- ** 创建者
6  ##-- ** 创建日期：
7  ##-- ** 修改日志：
8  ##-- ** 修改日期   修改人   修改内容
9  ##--*******************************************************************************
10
11 cnt=$(spark-sql -e "select count(1) from ddmc_prediction.algo_loss_lynn where biz_date='${zdt.addDay(-1).format('yyyy-MM-dd
12 if [ $cnt -eq 0 ];
13     then
14       echo '表中无数据'
15       exit 0
16 fi
17
18 /bin/bash /dw/ddmc.etl.resource/bi                    \
19 -x drs4 \
20 -f ${zdt.addDay(-1).format('yyyyMMdd')} \
21 -d bp \
22 -t algo_loss_lynn \
23 -g parquet \
24 -u ddmc_prediction \
25 -s algo_loss_lynn \
26 -c "city_name,product_id,province,area,product_name,sale_period,purchase_management_group_name,purchase_first_category_name
27 -p biz_date \
28 -v ${zdt.addDay(-1).format('yyyy-MM-dd')} \
```

总结与展望

Apache Doris 作为叮咚买菜整体架构体系中的核心 OLAP 分析引擎，不论是作为数仓数据看板类的应用层、还是作为实时数据汇总结果接入层、或是作为 B/C 端服务数据提供方，均能够很好的满足业务需求。除此之外，Doris 使得我们无需在存储选型上耗费过多时间，大大缩短了开发的周期；同时，Doris 支持 MySQL 协议和标准 SQL，大大降低内部人员的使用成本和门槛。未来，我们希望使用 Doris 支撑内部更多的业务场景，更大范围了提升工作效率。我们也会紧跟社区步伐，积极使用推出的新版本特性，以更好解决场景需求，提升业务效果。

最后，非常感谢 SelectDB 团队对我们在 Doris 使用上的技术支持，祝愿 SelectDB 和 Apache Doris 发展越来越好！

复杂查询响应速度提升 10+ 倍

度言软件基于 Apache Doris 实时数仓建设实践

作者：杭州度言软件大数据团队

导读：近两年随着业务不断发展带而来数据量的高速增长，传统的数仓技术架构已初显瓶颈，难以满足客户日益丰富多样化的数据及分析需求。为了给客户提供更优质的服务体验，度言软件亟需对现有的技术架构做出优化和重构。度言软件于 22 年 3 月份引入 Apache Doris，目前已将客户报表及公司内部运营决策数据全部迁移至 Apache Doris。本文将介绍度言软件数仓架构演进历程以及 Doris 在数仓建设中的应用实践。

杭州度言软件有限公司（度言软件）成立于 2014 年，是信贷不良资产处置技术服务供应商，以“智能科技赋能不良资产处置，推动贷后行业合规高效发展”为使命，运用云通讯、大数据、人工智能等智能科技为信贷不良资产处置业务赋能，提供贷后管理通信能力支撑，实现了催收作业的智能化管理，客户群体为银行、消费金融公司、AMC 等金融机构和为这些机构提供人力资源外包服务的相关公司，目前已拥有 2000 多家企业客户。

度言软件以围绕信贷不良资产案件高效流转管理为目的核心，从机构管理、团队管理、坐席管理、外呼作业、调解法诉等环节入手，帮助客户构建数智化的业务管理体系，以数字化系统提升管理效能、以智能化工具赋能处置作业，打通金融机构、外包服务公司、司法系统等多方的业务系统，并按照监管要求对外呼行为、录音文件、沟通记录等案件相关数据进行记录、汇集、稽核、统计和分析，**具有海量账号同时登陆、数据请求高并发、多来源数据汇总接入的特点要求。**

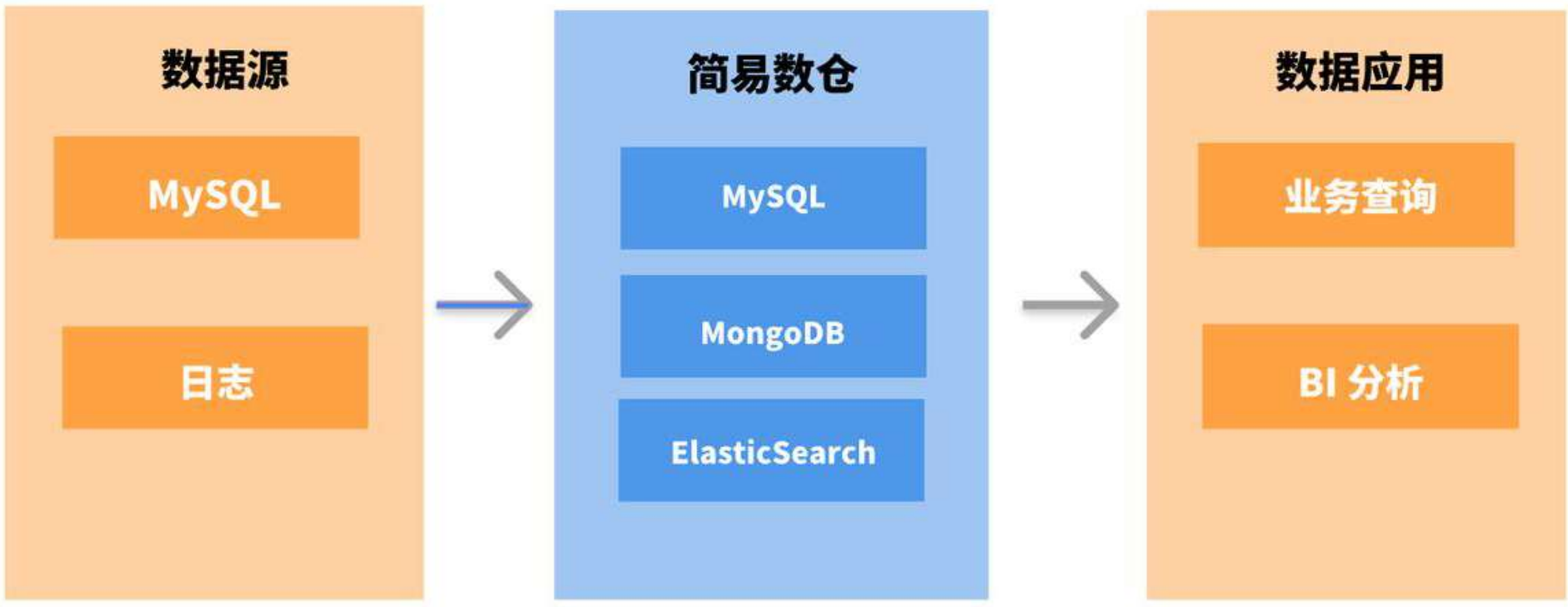
业务需求

2021 年之前，度言软件旗下产品的数据需求主要由传统数据库 MySQL，MongoDB，ElasticSearch 为主的技术架构来实现，近两年随着业务不断发展带而来数据量的高速增长，传统的数仓技术架构已初显瓶颈，难以满足客户日益丰富多样化的数据及分析需求。为了给客户提供更优质的服务体验，度言软件亟需对现有的技术架构做出优化和重构。

早期架构及痛点

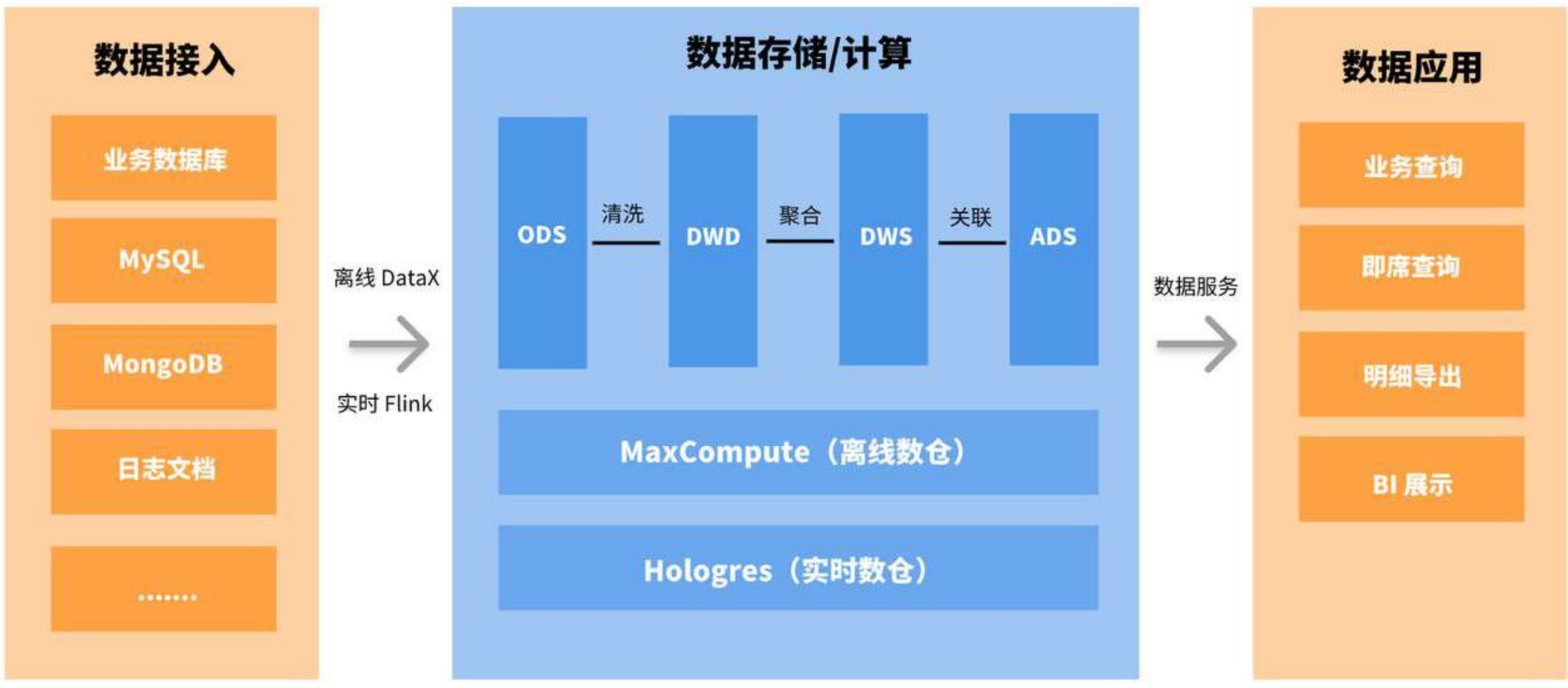
数仓架构 1.0

初创期间，由于公司业务量相对较少，主要还是以 OLTP 业务和少量的业务报表服务为主，并且对于数据分析方面的需求也很少，仅通过传统的数据库基本就能满足早期的业务数据需求。数仓架构 1.0 如下图所示：



数仓架构 2.0

随着公司业务的不断扩展，数据体量也出现极速增长的态势，业务侧对于数据分析方面的需求也逐渐多了起来，为此我们成立了专门的大数据团队，为搭建新的数仓及业务数据分析需求进行服务。如下图所示为数仓架构 2.0 ：



数仓架构 2.0 版本是基于 MaxCompute + Hologres/MySQL 来搭建的。数据来源主要有 MySQL 和 MongoDB 的业务数据以及埋点日志数据；数据首先采集到数据总线 DataHub 中，后经过 DataHub 直接导入到 MaxCompute，MaxCompute 作为一个离线数仓，我们将其进行了传统的数仓分层；数据的加工处理和分析计算主要在离线数仓中进行，并将计算好的结果导出到 MySQL 中，来对接 QuickBI 展示报表。此外，我们还尝试了将 Hologres 用作实时数仓，作为 MongoDB 的替换方案，用于业务上的查询系统。

早期架构存在的问题

- **响应速度较慢。**MySQL 对于大表的聚合计算并不友好，响应速度较慢。产品侧要求数据查询响应时间在 5 秒以内，虽然我们也基于 MySQL 进行了许多优化，但优化效果十分有限，仍无法达到 5s 的响应要求；我们甚至尝试了直接用 MaxCompute 对接 QuickBI，希望基于 MaxCompute 的查询加速和 QuickBI 的缓存来帮助我们解决问题，然而结果并不理想。

- **维系维度表成本高、难度大。**离线数仓在数据同步的时效性上并不占优势(每 5 分钟进行一次批量同步)，因此不太适合数据频繁更新和删除的场景，同时也给维度表的维护带来了额外的工作量。在数据更新和删除场景中，我们需要定期通过过滤和去重来保持数据的一致性，而事实上，大多时候需要报表数据实时关联维度表，这让我们直接放弃了在离线数仓中维系维度表的方案。

- **不支持高并发点查询场景。**原实时数仓虽然可以满足我们对数据分析的部分性能要求，但其对高并发的点查场景并不太友好，不管是采用列式存储还是行级存储建表，优化之后的响应时长在 500 毫秒左右，综合来看性价比不算太高。

思考与选型

新数仓要求

为了解决上述问题，我们希望**理想数仓能具有如下特性：**

- 实现一站式实时数仓，能同时满足多种不同业务数据需求，大大简化大数据架构体系；
- 可同时支持 OLAP，Ad-Hoc 和高 QPS 点查场景；
- 数据接入友好，写入即可见，对数据增删改和聚合等都有较好的支持；
- 架构简单，运维部署和维护简单，有较好的监控体系。

技术选型

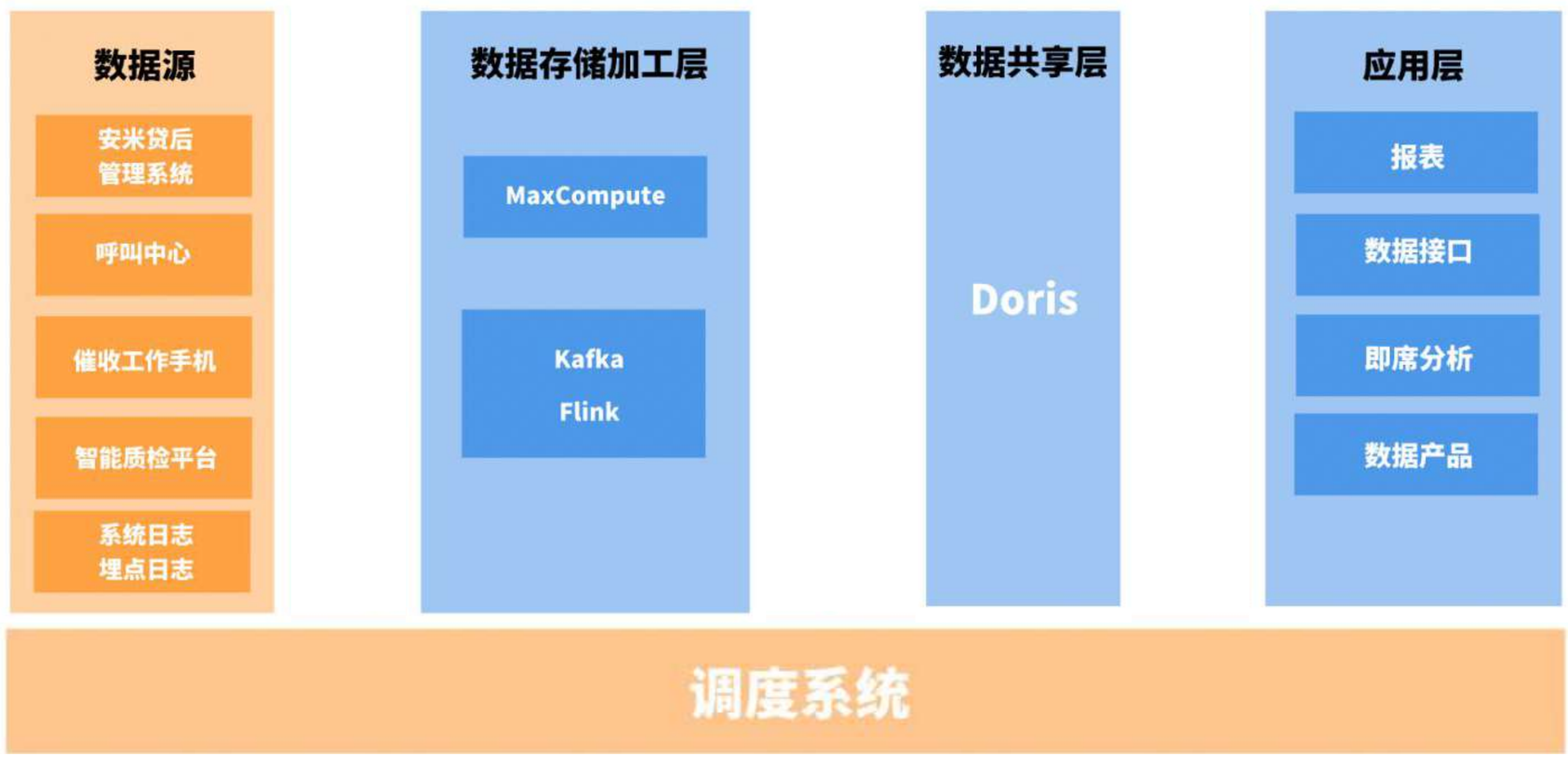
在 2022 年 3 月份，我们对市场上主流的的几款即席查询数据库进行了调研，调研中我们发现每个产品只能满足某 1 个或几个特定的使用场景，没有一个产品可以完全满足所有的选型要求，而同时采用多个产品，将大大增加开发运维成本，同时也会使架构变得更加庞大复杂，这并不符合我们对理想数仓的要求。

正在这时，我们从开源社区、技术媒体等渠道了解到了 Apache Doris ，经初步调研，我们发现 Doris 基本可以满足我们对理想数仓的所有要求。接着**我们对 Doris 展开了深入调研，并使我们最终决定使用 Doris**：Doris 架构非常简单，只有 FE 和 BE 两类进程，这两类进程都可以进行横向扩展，单集群可以支持到数百台机器、数十 PB 的存储容量，并且这两类进程通过一致性协议来保证服务的高可用和数据的高可靠。这种高度集成的架构设计极大的降低了分布式系统的运维成本，同时也不需要依赖于 Hadoop，避免了我们需要投入成本来额外部署 Hadoop 集群。

新数仓架构

基于 Doris 新数仓架构设计

最初使用 Doris 的初衷是替换部分 MySQL 数据量较大的报表，基于 MySQL 的查询约需要几十秒的响应时间，**在替换为 Doris 后，查询性能有了显著提升，几秒内即可返回结果，最长的 SQL 执行时间大概在 8 秒左右，速度相较于之前提升了8 倍。**Doris 的初步应用就给了我们一个意外的惊喜，因此我们决定使用 Doris 完全替换掉早期数仓中的 MySQL，在这使用过程中，我们又发现 Doris 远比我们想象的要强大，目前已将客户报表及公司内部运营决策数据全部迁移至 Apache Doris，并计划用 Apache Doris 来替换 MongoDB 和 ElasticSearch，用于业务上高 QPS 的点查场景。**以下是基于 Doris 的新数仓架构设计及使用情况：**



数据建模：

我们在业务上使用最多的是 Unique 模型和 Aggregate 模型，这两种模型基本能够满足业务上的需求。

- Unique 模型主要用于维度表和业务表(原始表)的接入，确保数据导入过程中的一致性。
- Aggregate 主要用于报表数据的导入，Aggregate 分为 Key (维度列) 和 Value（指标列），Value 列支持四种聚合方式： `sum` ， `replace` ， `max` ， `min` 。当前主要以 `replace` 聚合方式为主，方便统计数据重复导入和修正结果，后续也会尝试更多的方式来充分发挥 Doris 的性能。

数据接入：

- 使用 Flink-Doris-Connector 进行实时导入：主要用于业务数据的导入，基于MySQL 的 Binlog 日志写入到 Kafka 后，通过 Flink 解析加工后准实时写入 Doris。
- 使用 DataX 进行离线导入：主要用于对接离线数仓已计算后的报表数据。

数据开发：

目前 Doris 主要以提供终端查询为主，复杂的 SQL 开发任务运行比较少，为尽可能减少 Doris 在数据加工方面的资源消耗，当前仅有报表集群在凌晨执行少量的 SQL 任务，主要以 Doris SQL 通过 `insert into` 的方式来导入。

数据管理：

Doris 支持将当前数据以文件的形式通过 Broker 备份到远端存储系统中，后可以通过恢复命令从远端存储系统中将数据恢复到任意 Doris 集群。通过该功能，Doris 可支持定期对数据进行快照备份，也可以通过该功能在不同集群间进行数据迁移。我们也会定期将集群数据备份到阿里云 OSS 上，作为备用数据恢复方案。另外，在这期间我们对 Doris 集群进行了一次拆分，将报表集群和业务上的高并发查询集群分开，采用了 Doris 的数据备份和迁移功能。

监控和报警：

我们使用官网推荐的 Prometheus 和 Grafana 进行监控项的采集和展示，**Doris 本身提供了丰富的监控指标和标准监控模版，可以非常便捷地对 Doris 集群使用情况进行监控和报警。**

另外，为了进一步对慢 SQL 进行优化，我们还部署了审计日志插件，对于特定用户和特定的慢 SQL 进行优化和资源限制。如下是我们日常使用中的一些指标：

慢 SQL 查询：

```
2  --查询时长倒序
3  select  query_id,time,user,db,state,query_time,scan_bytes,scan_rows,return_rows,is_query,stmt from doris_audit_tbl__ where
4  user ='default_cluster:bi_palm' and
5  is_query=1
6  -- and state!='EOF'
7  and time>='2022-11-30 00:59:00'
8  -- and time<'2022-07-29 11:::00'
9  and (stmt like 'select%' or stmt like 'SELECT%')
0  and query_time>500
```

信息　摘要　结果 1　状态

time	user	db	state	query_time	scan_bytes	scan_rows	return_rows	is_query	stmt
2022-11-30 17:18:22	default_cluster:bi_palm	default_cluster:palm	EOF	1630	0	0	0	1	SELECT A54_T_1_.`dept_name` AS T_A0_2_, COUNT(I
2022-11-30 17:18:22	default_cluster:bi_palm	default_cluster:palm	EOF	1548	0	0	0	1	SELECT DATE_FORMAT(A32_T_1_.`call_time_trans`, '%
2022-11-30 17:39:29	default_cluster:bi_palm	default_cluster:palm	EOF	1495	0	0	0	1	SELECT A54_T_1_.`dept_name` AS T_A0_2_, COUNT(I
2022-11-30 17:20:27	default_cluster:bi_palm	default_cluster:palm	EOF	1494	0	0	1	1	SELECT COUNT(ALL_T_1_.`id`) AS T_A0_2_, SUM(ALL
2022-11-30 07:32:08	default_cluster:bi_palm	default_cluster:palm	EOF	1338	0	0	29	1	SELECT DATE_FORMAT(A54_T_1_.`send_time`, '%Y%m
2022-11-30 07:32:08	default_cluster:bi_palm	default_cluster:palm	EOF	1326	0	0	10	1	SELECT DATE_FORMAT(A32_T_1_.`call_time_trans`, '%
2022-11-30 17:18:22	default_cluster:bi_palm	default_cluster:palm	EOF	1290	0	0	0	1	SELECT DATE_FORMAT(A54_T_1_.`send_time`, '%Y%m
2022-11-30 17:18:22	default_cluster:bi_palm	default_cluster:palm	EOF	1254	0	0	0	1	SELECT A32_T_1_.`dept_name` AS T_A0_2_, SUM(A3
2022-11-30 17:39:29	default_cluster:bi_palm	default_cluster:palm	EOF	1248	0	0	0	1	SELECT DATE_FORMAT(A32_T_1_.`call_time_trans`, '%
2022-11-30 07:32:08	default_cluster:bi_palm	default_cluster:palm	EOF	1127	0	0	1	1	SELECT A32_T_1_.`dept_name` AS T_A0_2_, SUM(A3
2022-11-30 10:59:33	default_cluster:bi_palm	default_cluster:palm	EOF	916	0	0	1	1	SELECT A54_T_1_.`dept_name` AS T_A0_2_, COUNT(I
2022-11-30 07:32:08	default_cluster:bi_palm	default_cluster:palm	EOF	916	0	0	9	1	SELECT DATE_FORMAT(A82_T_1_.`send_time`, '%Y%m
2022-11-30 17:39:29	default_cluster:bi_palm	default_cluster:palm	EOF	871	0	0	0	1	SELECT A32_T_1_.`dept_name` AS T_A0_2_, SUM(A3

对于一些长文本 SQL，无法完全展示，可以进一步查看 `fe.audit.log` 。

主要查询统计指标：



优化实践

提高并发

我们考虑在资源给定的情况下，如何最大程度地提高并发。刚开始引入 Doris 集群的时候，我们尝试对复杂 SQL 进行压测（SQL 层面优化已完成），但始终无法达到预期的压测效果。后来我们尝试调低 `parallel_fragment_exec_instance_num` 的值，顺利完成了压测目标。

（需要说明的是：`parallel_fragment_exec_instance_num` 指的是 Scan Node 在每个 BE 节点上执行实例的个数，相当于在整个查询计划执行过程中的并发度，调高该参数可以提升查询效率，但同时也会增加更多机器资源的消耗。因此在资源有限且查询耗时满足业务需求的情况下，通过调低参数来节省单个 SQL 的资源消耗，有助于提高并发表现。另外，我们通过 Doris 社区了解到，在即将发布的新版本中将实现参数自动设置，无需进行手动调整。）

如下图，我们可以看到，在参数设置为 16 的时候，异常率高达 82.84%，并且期间还出现了 BE 宕机重启，将参数调低至 8 后，异常率也同步降低到了 27.66%。**最后我们将参数设置为最小值 1 后，异常率为 0，查询响应也能达到预期目标。**

说明：测试环境已重新取数，配置较低，数据仅用来说 `parallel_fragment_exec_instance_num` 变动所带来的效果。

当参数调整为 1： `parallel_fragment_exec_instance_num` = 1

Label	# 样本	平均值	最小值	最大值	标准偏差	异常 %	吞吐量	接收 KB/sec	发送 KB/sec	平均字节数
JDBC Requ...	244	7916	2476	12281	958.55	0.00%	3.7/sec	4.94	0.00	1355.0
总体	244	7916	2476	12281	958.55	0.00%	3.7/sec	4.94	0.00	1355.0

当参数调整为 8： `parallel_fragment_exec_instance_num` = 8

Label	# 样本	平均值	最小值	最大值	标准偏差	异常 %	吞吐量	接收 KB/sec	发送 KB/sec	平均字节数
JDBC Requ...	235	8339	774	13940	3614.52	27.66%	3.5/sec	4.12	0.00	1190.0
总体	235	8339	774	13940	3614.52	27.66%	3.5/sec	4.12	0.00	1190.0

```
W1222 11:00:54.724195 17865 fragment_mgr.cpp:247] Got error while opening fragment 1f02c1124208849d1-a2207619e64a369d: Cancelled: Cancelled
W1222 11:00:54.724609 15238 vdata_stream_sender.h:248] failed to send brpc batch, error=Host is down, error_text=[E112]Not connected to 127.0.0.1:8060 yet, server_id=138 [R1][E112]Not co
nected to 127.0.0.1:8060 yet, server_id=138 [R2][E112]Not connected to 127.0.0.1:8060 yet, server_id=138 [R3][E112]Not connected to 127.0.0.1:8060 yet, server_id=138 [R4][E112]Not connect
ed to 127.0.0.1:8060 yet, server_id=138 [R5][E112]Not connected to 127.0.0.1:8060 yet, server_id=138 [R6][E112]Not connected to 127.0.0.1:8060 yet, server_id=138 [R7][E112]Not connected t
o 127.0.0.1:8060 yet, server_id=138 [R8][E112]Not connected to 127.0.0.1:8060 yet, server_id=138 [R9][E112]Not connected to 127.0.0.1:8060 yet, server_id=138 [R10][E112]Not connected to 1
27.0.0.1:8060 yet, server_id=138, client: 10.73.2.34, latency = 36376
W1222 11:00:54.724653 15238 vdata_stream_sender.h:248] failed to send brpc batch, error=Host is down, error_text=[E112]Not connected to 127.0.0.1:8060 yet, server_id=138 [R1][E112]Not co
nected to 127.0.0.1:8060 yet, server_id=138 [R2][E112]Not connected to 127.0.0.1:8060 yet, server_id=138 [R3][E112]Not connected to 127.0.0.1:8060 yet, server_id=138 [R4][E112]Not connect
ed to 127.0.0.1:8060 yet, server_id=138 [R5][E112]Not connected to 127.0.0.1:8060 yet, server_id=138 [R6][E112]Not connected to 127.0.0.1:8060 yet, server_id=138 [R7][E112]Not connected t
o 127.0.0.1:8060 yet, server_id=138 [R8][E112]Not connected to 127.0.0.1:8060 yet, server_id=138 [R9][E112]Not connected to 127.0.0.1:8060 yet, server_id=138 [R10][E112]Not connected to 1
27.0.0.1:8060 yet, server_id=138, client: 10.73.2.34, latency = 69784
W1222 11:00:54.732759 17809 fragment_mgr.cpp:247] Got error while opening fragment 124adfe1ee464466-a383719e3135b6e6: Cancelled: Cancelled
W1222 11:00:54.738317 16673 fragment_mgr.cpp:247] Got error while opening fragment 1d9c0f217fdc4aa0-9f674e28908a3835: Cancelled: Cancelled
W1222 11:00:54.740142 15974 fragment_mgr.cpp:247] Got error while opening fragment 1d9c0f217fdc4aa0-9f674e28908a383d: Cancelled: Cancelled
W1222 11:00:54.741847 16988 fragment_mgr.cpp:247] Got error while opening fragment 124adfe1ee464466-a383719e3135b6f0: Cancelled: Cancelled
W1222 11:00:54.741981 15134 fragment_mgr.cpp:247] Got error while opening fragment 18bfcf567a524858-9e5e1df82b668859: Cancelled: Cancelled
W1222 11:00:54.742120 15583 fragment_mgr.cpp:247] Got error while opening fragment 3c963601677c4f5a-a2f9790786482c76: Cancelled: Cancelled
W1222 11:00:54.742213 13760 fragment_mgr.cpp:247] Got error while opening fragment c3432daa0e9e4870-a58ff34a08afbe97: Cancelled: Cancelled
```

当参数调整为 16： `parallel_fragment_exec_instance_num` = 16

Label	# 样本	平均值	最小值	最大值	标准偏差	异常 %	吞吐量	接收 KB/sec	发送 KB/sec	平均字节数
JDBC Requ...	373	4837	1120	13915	3082.33	82.84%	6.1/sec	5.21	0.00	874.4
总体	373	4837	1120	13915	3082.33	82.84%	6.1/sec	5.21	0.00	874.4

BE 内存问题

最初我们使用的是 Doris 0.15 的版本。由于刚开始处于测试阶段，Doris 集群配置比较低或参数配置的不合理，当某些 SQL 扫描数据量较大时，就可能因为内存问题导致 BE 宕机。

在向社区咨询后，了解到 Supervisor 是用 Python 开发的一套通用的进程管理程序，能将一个普通的命令行进程变为后台 Daemon，并监控进程状态，异常退出时能自动重启，因此我们参照官网给出的例子直接用 Supervisor 对 Doris 的 FE 和 BE 进程进行管理。

但是在运行了一段时间后，新的问题又出现了(已升级到 1.1.0 版本)。Doris 的 BE 节点内存一直在缓慢上升状态，并且达到了设置的最大内存参数 80% 后仍在继续上涨。后经分析发现 BE 存在内存泄漏问题，因此设置的参数并未生效。为此，我们将 Supervisor 切换为 Systemd 来守护 FE、BE 进程，限制整个节点的内存使用上限。不过在 **Doris 1.1.3 推出之后，此问题已得到彻底的修复。**

资源占用

在迁移完新集群后，我们发现通过 Flink-Doris-Connector 数据导入占用非常高的集群资源，虽然设置了按批次写入(每 3s 写入一次)，以限制数据的导入频率，但集群资源的占用仍未得到较大改善。因此我们在集群资源和数据实时可见性方面做了权衡，介于我们对数据实时性的要求相对不是太高，所以我们将每 3s 写入一次改为每 10s 或 20s 写入一次，调整写入时间后，集群的 CPU 资源占用下降明显，得到改善。

应用现状

目前度言软件有 2 个 Doris 集群，1 个集群用作线上业务的查询系统，主要是应对高 QPS 的点查场景，我们将原先基于业务库 MySQL 和 MongoDB 的查询迁移至 Doris，一方面减少了业务库的读写压力，同时也提高了用户查询服务的使用体验。**在 Doris 中，最复杂的查询在 1 秒以内即可响应，响应速度提升了十几倍**；另外 1 个集群主要用作即席查询(AD-Hoc Query)和报表分析，具体包括公司内部使用的所有报表和一些临时查询、客户报表、数据实时大屏。

总而言之，使用 Doris 替换了部分业务使用场景后，**用户在产品上的使用体验有了进一步得到提升，页面打开更加流畅**，原本因为查询慢而不能实现的功能，当前已经实现并上线使用。同时**在资源成本上也减轻了 MySQL 和 MongoDB 数据库的压力，不需要频繁进行升配和磁盘升级**。

总结规划

效果总结

- Doris 完美覆盖了原本需要多个技术栈才能实现的业务场景，**极大地简化了大数据架构体系。**

- **Doris 对 Join 支持较好，**因此我们选择了将维度表放到 Doris 中进行维护，相较于之前在离线数仓中进行维护，明显地提高了开发的效率，并降低了数据出错的可能性，满足了业务上对维度表近实时更新的需求。

- **Doris 支持 MySQL 协议和标准 SQL，**开发人员学习成本低，上手简单，可以快速将原先的业务迁移至 Doris 上来。

- **Doris 社区活跃，版本迭代速度快。**在遇到任何问题时，都可以向社区求助，**SelectDB 为 Apache Doris 组建了一支全职专业的技术支持团队，**24H 内即可得到社区的响应回复。

未来规划

到目前为止，基于 Doris 的实时数仓搭建已基本完成，但我们对 Doris 的进一步尝试才刚刚开始，比如 BE 的多磁盘部署，物化视图的使用，Doris-On-ES，Doris 多租户和资源划分等。

此外，我们也希望 Doris 未来能对以下功能进行进一步优化：

- Flink-Doris-Connector 能支持单 Sink 同时写入多张表，不需要再通过分流后多个 Sink 写入。

- 物化视图能够支持多表 Join，不再局限于单表。

- 进一步优化数据的底层 Compaction，在保证数据可见性的同时能够尽可能降低资源消耗。

- Doris-Manager 提供对慢 SQL 的优化建议以及表信息收集，对于不合理的建表进行一定的提示。

最后，从 3 月份使用 Doris 以来，我们一直都和 Doris 社区保持着密切的联系，后续我们也将继续围绕 Doris 作为实时数仓应用到更多的业务使用场景中，对于使用中遇到的问题和优化的方案，我们也会持续和社区保持热切联系，为社区进步贡献我们的一份力量。

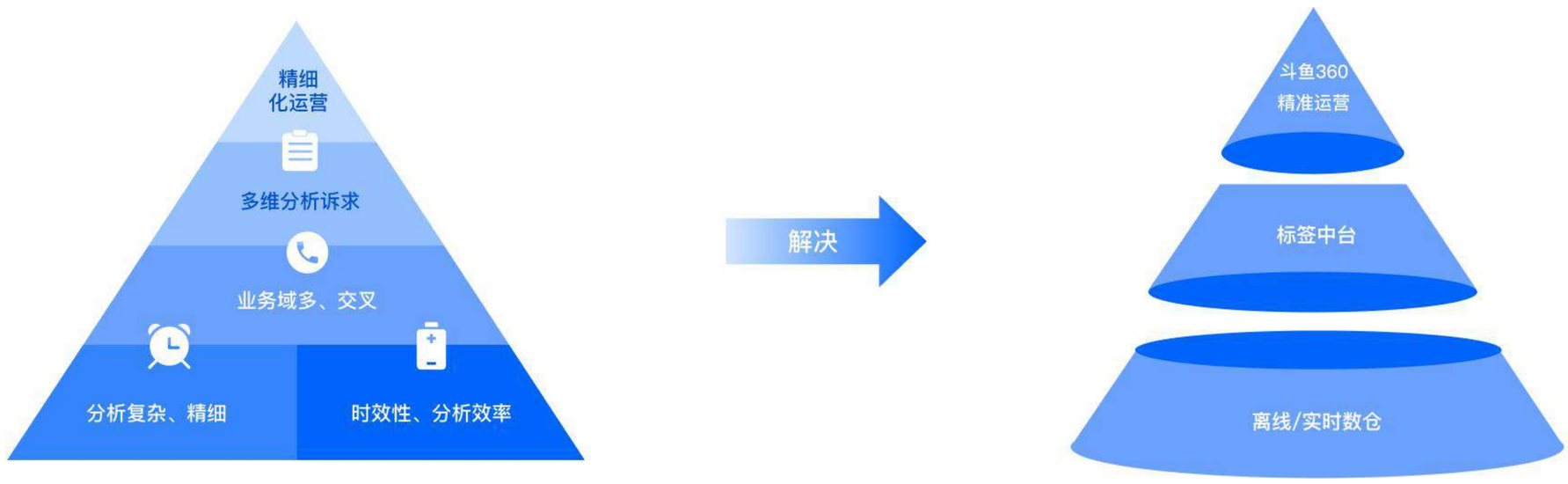
赋能直播行业精细化运营

斗鱼基于 Apache Doris 的应用实践

作者：斗鱼资深大数据工程师、OLAP 平台负责人，韩同阳

导读：斗鱼是一家弹幕式直播分享网站，为用户提供视频直播和赛事直播服务。随着斗鱼直播、视频等业务的高速发展，用户增长和营收两大主营业务线对精细化运营的需求越发地迫切，各个细分业务场景对用户的差异化分析诉求也越发的强烈。为更好满足业务需求，斗鱼在 2022 年引入了 Apache Doris 构建了一套比较相对完整的实时数仓架构，并在该基础上成功构建了标签平台以及多维分析平台，在此期间积累了一些建设及实践经验通过本文分享给大家。

斗鱼是一家弹幕式直播分享网站，为用户提供视频直播和赛事直播服务。斗鱼以游戏直播为主，也涵盖了娱乐、综艺、体育、户外等多种直播内容。随着斗鱼直播、视频等业务的高速发展，用户增长和营收两大主营业务线对精细化运营的需求越发地迫切，各个细分业务场景对用户的差异化分析诉求也越发的强烈，例如增长业务线需要在各个活动（赛事、专题、拉新、招募等）中针对不同人群进行差异化投放，营收业务线需要根据差异化投放的效果及时调整投放策略。



根据业务场景的诉求和精细化运营的要求，我们从金字塔自下而上来看，需求大致可以分为以下几点：

- 分析需求更加复杂、精细化，不再满足简单的聚合分析；数据时效性要求更高，不满足于

T+1 的分析效率，期望实现近实时、实时的分析效率。

- 业务场景多，细分业务场景既存在独立性、又存在交叉性，例如：针对某款游戏进行专题活动投放（主播、用户），进行人群圈选、 AB 实验等，需要标签/用户画像平台支持。
- 多维数据分析的诉求强烈，需要精细化运营的数据产品支持。

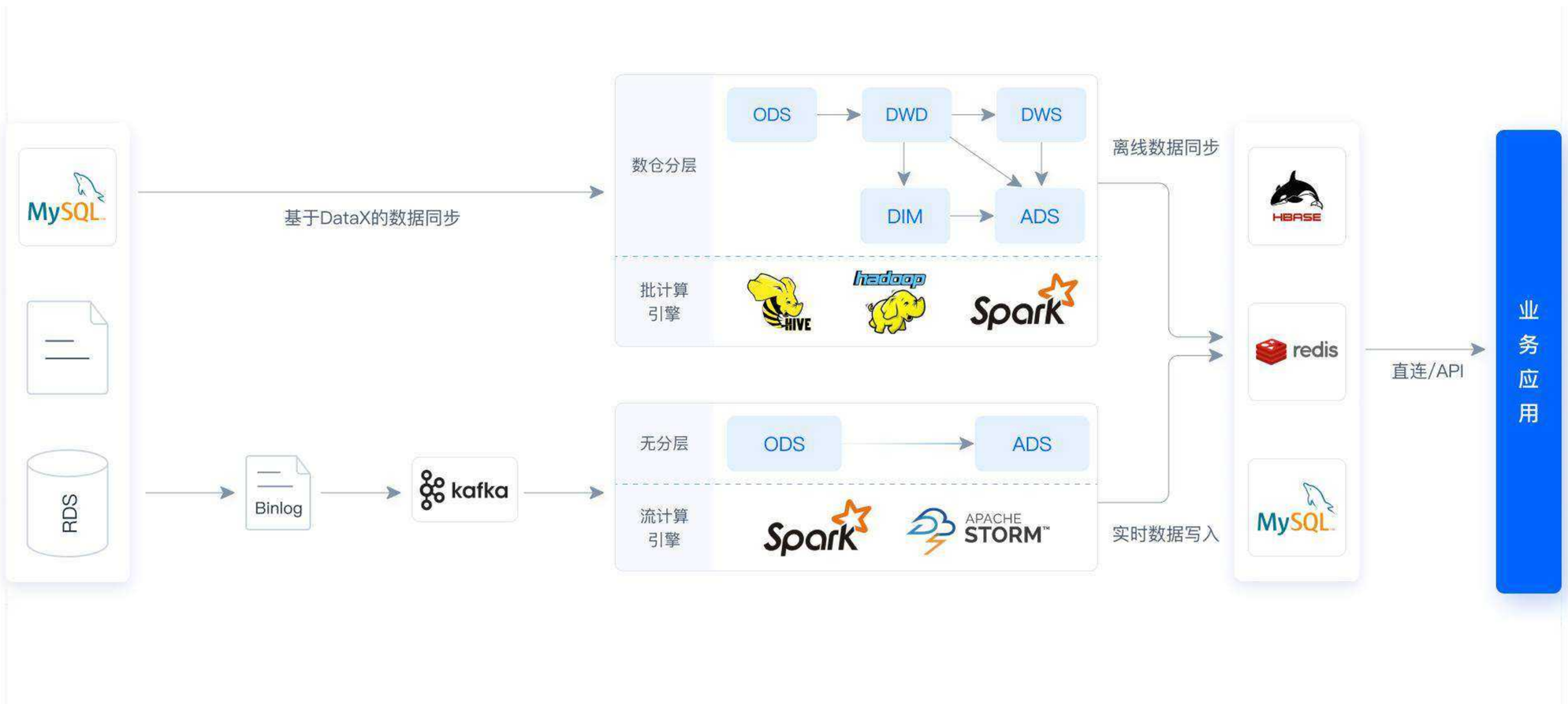
为更好解决上述需求，我们的初步目标是：

- 构建离线/实时数仓，斗鱼的离线数仓体系已成熟，希望此基础上构建一套实时数仓体系；
- 基于离线/实时数仓构建通用的标签中台（用户画像平台），为业务场景提供人群圈选、 AB 实验等服务；
- 在标签平台的基础上构建适用于特定业务场景的多维分析和精细化运营的数据产品。

在目标驱动下，斗鱼在原有架构的基础上进行升级改造、**引入 Apache Doris 构建了实时数仓体系，并在该基础上成功构建了标签平台以及多维分析平台，**在此期间积累了一些建设及实践经验通过本文分享给大家。

原有实时数仓架构

斗鱼从 2018 年开始探索实时数仓的建设，并尝试在某些垂直业务领域应用，但受制于人力的配置及流计算组件发展的成熟度，直到 2020 年第一版实时数据架构才构建完成，架构图如下：



原有实时数仓架构是一个典型的 Lambda 架构，上方链路为离线数仓架构，下方链路为实时数据仓架构。鉴于当时离线数仓体系已经非常成熟，使用 Lambda 架构足够支撑实时分析需求，但随着业务的高速发展和数据需求的不断提升，**原有架构凸显出几个问题：**

- 在实际的流式作业开发中，缺乏对实时数据源的管理，在极端情况下接近于烟囱式接入实时数据流，无法关注数据是否有重复接入，也无法辨别数据是否可以复用。
- 离线、实时数仓完全割裂，实时数仓没有进行数仓分层，无法像离线数仓按层复用，只能面向业务定制化开发。
- 数据仓库数据服务于业务平台需要多次中转，且涉及到多个技术组件，ToB 应用亟需引入 OLAP 引擎缓解压力。
- 计算引擎和存储引擎涉及技术栈多，学习成本和运维难度也很大，无法进行合理有效管理。

新实时数仓架构

技术选型

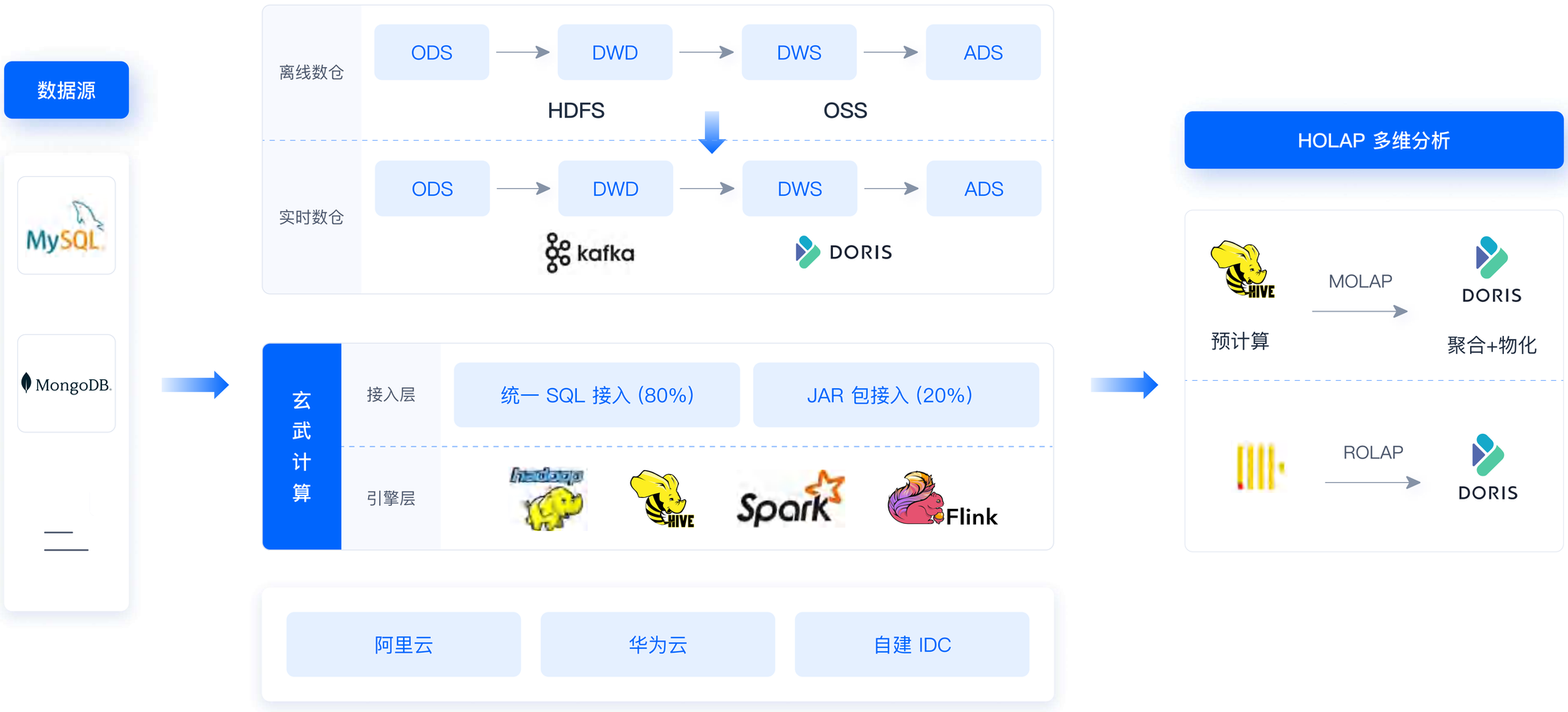
带着以上的问题，我们希望引入一款成熟的、在业内有大规模落地经验的 OLAP 引擎来帮助我们解决原有架构的痛点。我们希望该 OLAP 引擎不仅要具备传统 OLAP 的优势（即 Data Analytics ），还能更好地支持数据服务（Data Serving）场景，比如标签数据需要明细级的查询、实时业务数据需要支持点更新、高并发以及大数据量的复杂 Join 。除此之外，我们希望该 OLAP 引擎可以便捷、低成本的的集成到 Lambda 架构下的离线/实时数仓架构中。**立足于此，我们在技术选型时对比了市面上的几款 OLAP 引擎，如下图所示：**

功能/产品	Apache Doris	ClickHouse	Apache Kylin	某云厂商产品H
标准SQL	支持	支持	支持	支持
离线导入	支持	支持	支持	支持
实时导入	支持	支持	支持	支持
聚合查询	支持	支持	支持	支持
明细查询	支持	支持	不支持	支持
Ad-Hoc查询	支持	支持	差	支持
数据更新	支持（批次事务更新）	支持（不具备事务）	不支持	支持
并发	好	相对较差	好	好
Join性能	好	差	不支持	好
兼容协议	MySQL（易上手）	JDBC		PostgreSQL（上手/学习成本高）
生态发展	独立生态、架构简单、中英文档较完善	独立生态、架构简单、中文文档缺乏	依赖Hadoop生态	闭源生态、文档不够完善（云产品）
运维难度	低	中	高	未知

根据对选型的要求，**我们发现 Apache Doris 可以很好地满足当前业务场景及诉求，同时也兼顾了低成本的要求，因此决定引入 Doris 进行升级尝试。**

架构设计

我们在 2022 年引入了 Apache Doris ，并基于 Apache Doris 构建了一套比较相对完整的实时数仓架构，如下图所示。



总的来说，引入 Doris 后为整体架构带来几大变化：

- 统一了计算平台（玄武计算），底层引擎支持 Flink、Spark 等组件，接入层支持统一 SQL 和 JAR 包接入。
- 引入 Doris 后，我们将实时数仓分为 ODS、DWD、DWS、ADS 层，部分中间层实时数据直接使用 Doris 进行存储；
- 构建了基于 Doris 的 HOLAP 多维分析平台，直接服务于业务；简化了原来需要通过 Hive 进行预计算的加工链路，逐步替换使用难度和运维难度相对较高的 ClickHouse；
- 下游应用的数据存储从之前的 MySQL 和 HBase 更换为 Doris，可以在数据集市和大宽表的数据服务场景下直接查询 Doris。
- 支持混合 IDC（自建和云厂商）。

Overwrite 语义实现

Apache Doris 支持原子替换表和分区，我们在计算平台（玄武平台）整合 Doris Spark Connector 时进行了定制，且在 Connector 配置参数上进行扩展、增加了“Overwrite”模式。

当 Spark 作业提交后会调用 Doris 的接口，获取表的 Schema 信息和分区信息。

- 如果为非分区表：先创建目标表对应的临时表，将数据导入到临时表中，导入后进行原子替换，如导入失败则清理临时表；
- 如果是动态分区表：先创建目标分区对应的临时分区，将数据导入临时分区，导入后进行原子替换，如导入失败则清理临时分区；
- 如果是非动态分区：需要扩展 Doris Spark Connector 参数配置分区表达式，配完成后先创建正式目标分区、再创建其临时分区，将数据导入到临时分区中，导入后进行原子替换，如导入失败则清理临时分区。

架构收益

通过架构升级及二次开发，我们获得了 3 个明显的收益：

- 构建了规范、完善、计算统一的实时数仓平台
- 构建了统一混合 OLAP 平台，既支持 MOLAP，又支持 ROLAP，大部分多维分析需求均由该平台实现。
- 面对大批量数据导入的场景，任务吞入率和成功率提升了 50%。

Doris 在标签中台的应用

标签中台（也称用户画像平台）是斗鱼进行精准运营的重要平台之一，承担了各业务线人群圈选、规则匹配、A/B 实验、活动投放等需求。接下来看下 Doris 在标签中台是如何应用的。

原标签中台架构

下图为斗鱼原来的标签中台架构（见下一页），离线标签在数仓中加工完成后合入宽表，将最终数据写入 HBase 中，实时标签使用 Flink 加工，加工完直接写入到 HBase 中。

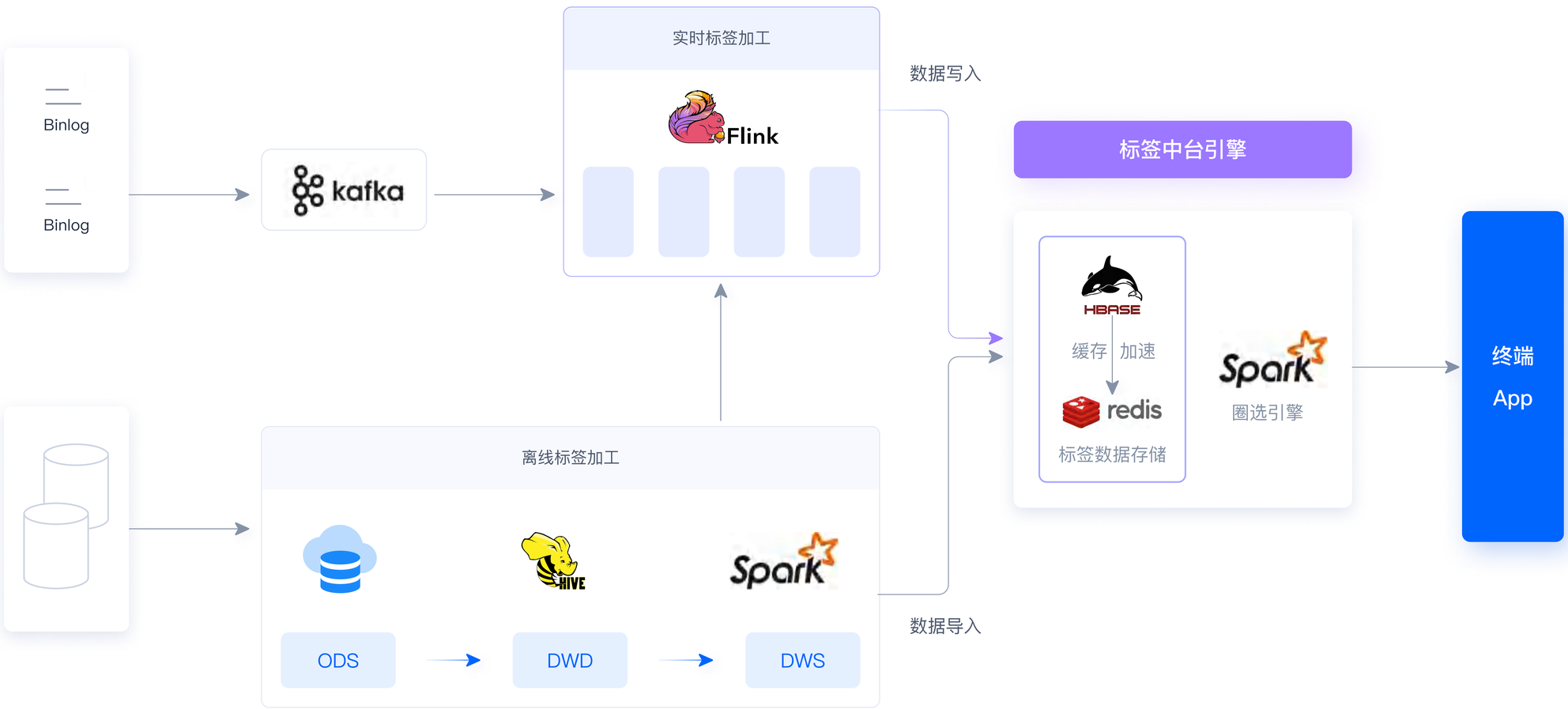
终端 APP 在使用标签中台时，主要解决两种业务需求：

- 人群圈选，即通过标签和规则找到符合条件的人。

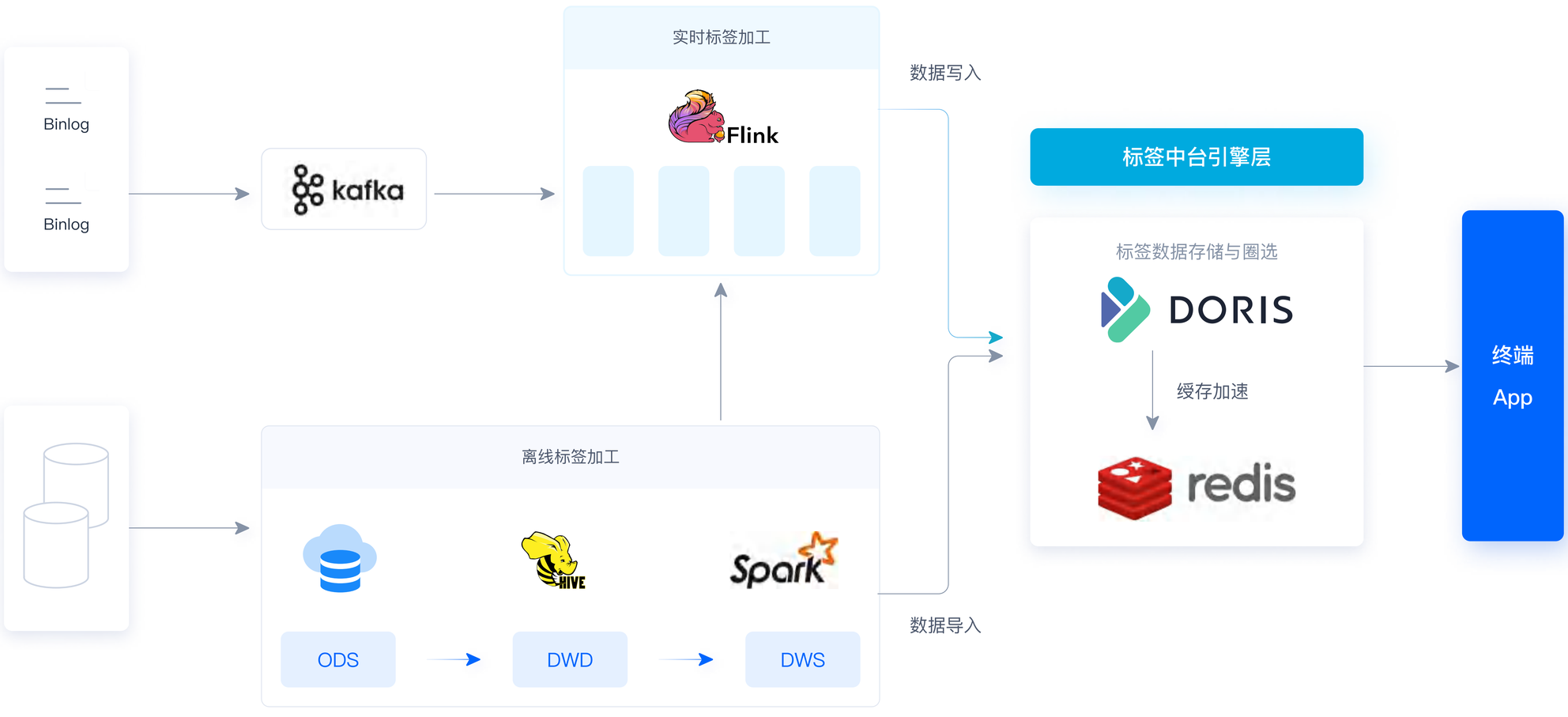
- 规则匹配，即当有一个用户，找出该用户在指定的业务场景下符合哪些已配置的规则，也可以理解是“人群圈选“的逆方向。

在应对这两种场需求中，原标签中台架构出现了两个问题：

- 实时标签链路：Flink 计算长周期实时指标时稳定性较差且耗费资源较高，任务挂掉之后由于数据周期较长，导致 Checkpoint 恢复很慢；
- 人群圈选：Spark 人群圈选效率较低，特别是在实时标签的时效性上。



新标签中台架构



引入 Apache Doris 之后，我们对标签中台架构的进行了改进，主要改进集中在实时链路和标签数据存储这两个部分：

- 实时标签链路：仍然是通过实时数据源到 Kafka 中，通过 Flink 进行实时加工；不同的是，我们将一部分加工逻辑迁移到 Doris 中进行计算，长周期实时指标的计算从单一的 Flink 计算转移到了 Flink + Doris 中进行；

- 标签数据存储：从 HBase 改成了 Doris，利用 Doris 聚合模型的部分更新特性，将离线标签和实时标签加工完之后直接写入到 Doris 中。

1. 离线/实时标签混合圈人

- 简化存储：原存储在 HBase 中的大宽表，改为在 Doris 中分区存储，其中离线标签 T+1 更新，实时标签 T 更新、T+1 采用离线数据覆盖矫正。

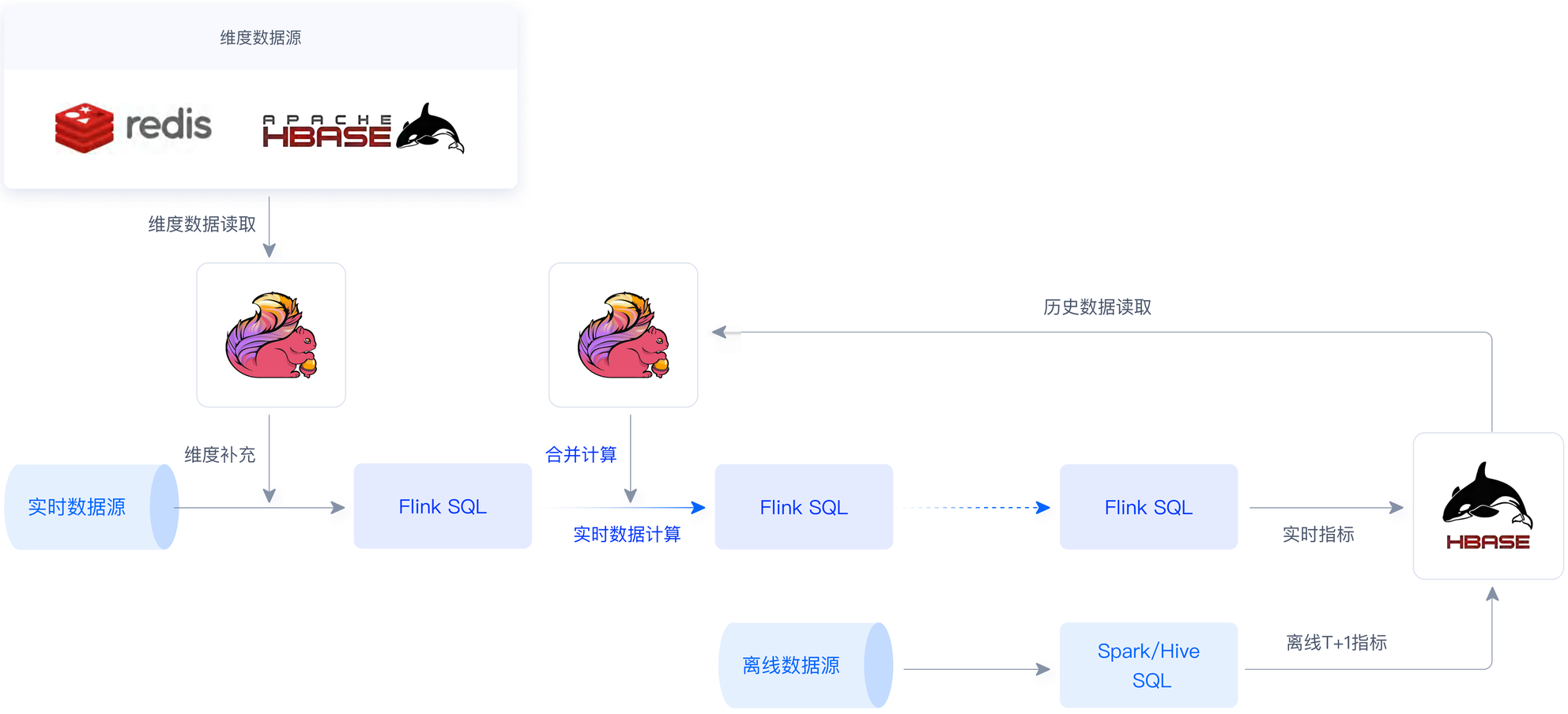
- 查询简化：**面对人群圈选场景，**无需利用 Spark 引擎，可直接在标签中台查询服务层，将圈选规则配置解析成 SQL 在 Doris 中执行、并获得最终的人群，大大提高了人群圈选的效率。**面对规则匹配场景，**使用 Redis 缓存 Doris 中的热点数据，以降低响应时间。



2. 长周期实时标签计算原链路

长周期实时标签：计算实时标签时所需的数据周期较长，部分标签还需要采用历史数据（离线）合并实时数据流一起进行计算的场景。

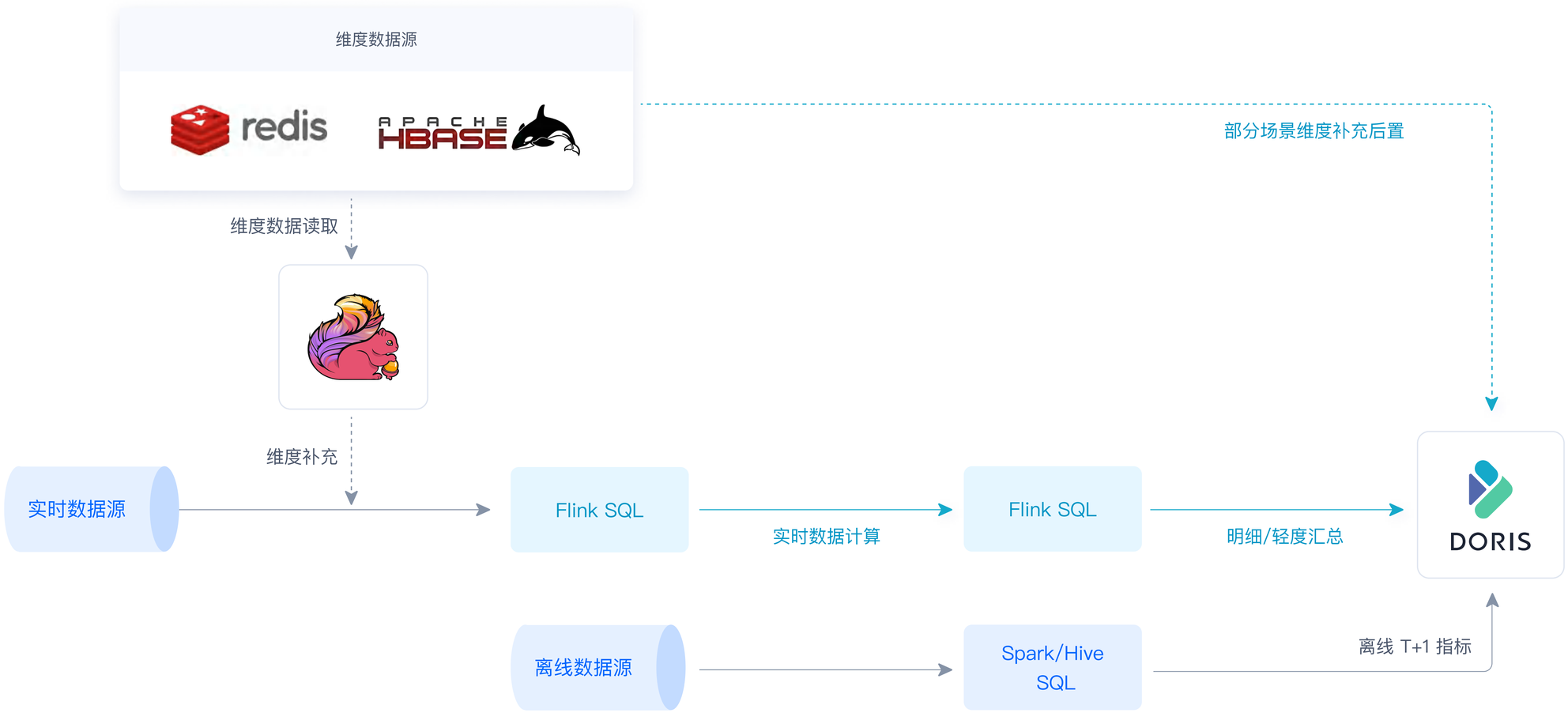
使用前：



从原来的计算链路中可知，计算长周期的实时标签时会涉及到维度补充、历史数据 Merge，在经过几步加工最终将数据写入到 HBase 中。

在实际使用中发现，在这个过程中 Merge 离线聚合的数据会使链路变得很复杂，往往一个实时标签需要多个任务参与才能完成计算；另外聚合逻辑复杂的实时标签一般需要多次聚合计算，任意一个中间聚合资源分配不够或者不合理，都有可能出现反压或资源浪费的问题，从而使整个任务调试起来特别困难，同时链路过长运维管理也很麻烦，稳定性也比较低。

使用后：



我们在长周期指标计算实时链路中加入了 Apache Doris，在 Flink 中只做维度补充和轻度加工汇总，只关注短的实时数据流，对于需要 Merge 的离线数据，Merge 的计算逻辑转移到 Doris 中进行计算，另外 Doris 中的轻度汇总/明细数据有助于问题排查，同时任务稳定性也能提升。

使用收益

目前标签中台底层有近 4 亿+ 条用户标签，每个用户标签 300+，已有 1W+ 用户规则人群，每天定时更新的人群数量达到 5K+。标签中台引入 Apache Doris 之后，**单个人群平均圈选时间实现了分钟级到秒级的跨越**，实时标签任务稳定性有所提高，实时标签任务的产出时间相较于之前约有 **40% 的提升**，资源使用成本大大降低。

Doris 在多维数据分析平台的应用

除以上所述应用及收益之外，Apache Doris 也助力内部多维数据分析平台——斗鱼 360 取得了较大的发展，受益于 Apache Doris 的 Rollup、物化视图以及向量化执行引擎，使原来需要预计算的场景可以直接导入明细数据到 Doris 中，简化了业务数据开发流程，提升了分析效率；Doris 兼容 MySQL 协议，并具有独立简单的分布式架构，使得业务开发人员入门使用也更容易，缩短了业务开发周期，有效降低了开发成本；同时我们原来基于 ClickHouse 的查询目前也全部切换到了 Doris 中进行。

目前我们用于多维分析场景的 Doris 集群共有两个，节点规模约 120 个，存储数据量达 90~100 TB，每天增量写入到 Doris 的数据约 900GB，其中查询 QPS 在 120 左右，Apache Doris 应对起来毫不费力，轻松自如。

未来展望

未来随着 Apache Doris 在斗鱼更广泛业务场景的落地，我们将在可视化运维、问题快速定位排查等方面进行更多实践和深耕。我们关注到， Apache Doris 1.2.0 版本已经对 Multi Catalog 功能进行了支持，我们也计划对其进行探索、解锁更多使用场景，同时也期待即将发布 Apache Doris 2.x 版本的行列混存功能，更好的支持 Data Serving 场景。

从 Clickhouse 到 Apache Doris

慧策电商 SaaS 高并发数据服务的改造实践

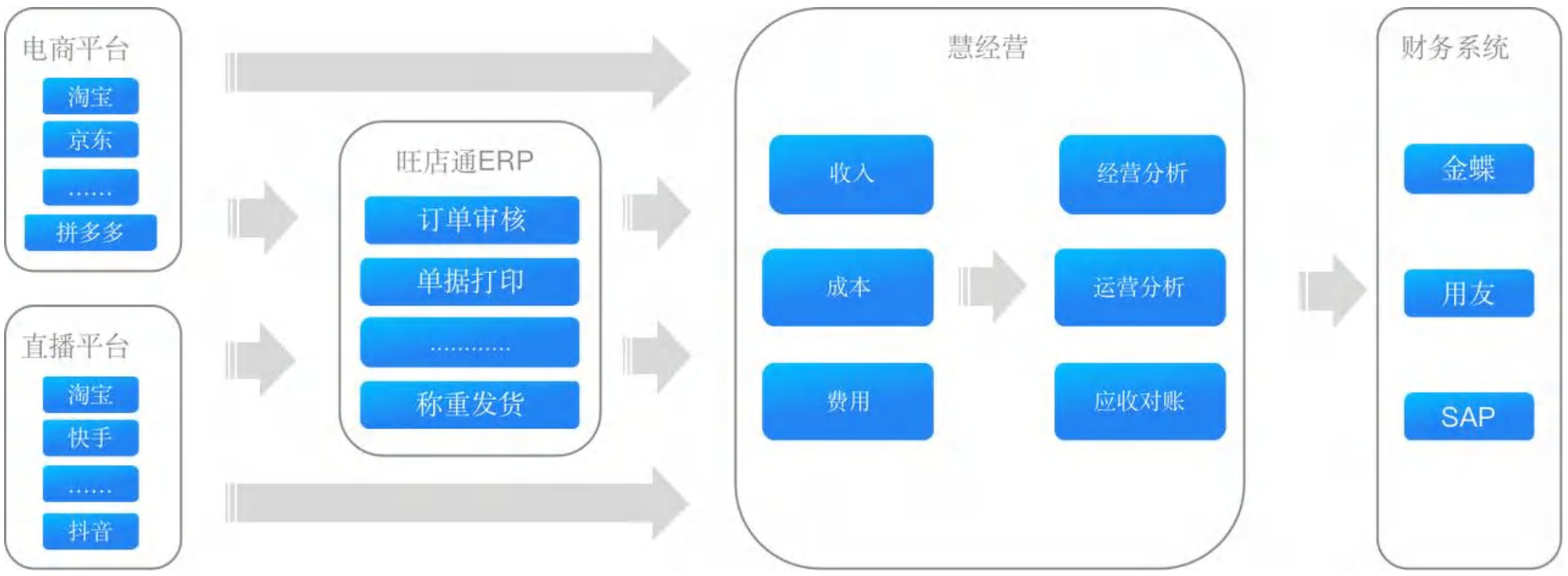
作者：慧策 Java 高级研发工程师，马成

导读：慧经营是一款数据分析产品，主要提供经营分析、运营分析、应收对账三大功能，面向企业的经营、运营、财务三个角色。可以根据客户需求提供经营分析、报表生成、账单汇总等多项数据分析服务，以帮助客户更好地经营店铺。本文将分享慧经营从 ClickHouse 到 Apache Doris 的架构演进之路，并将其在资源隔离、高并发等场景上的探索进行了经验总结，通过本文分享给大家。

慧策（原旺店通）是一家技术驱动型智能零售服务商，基于云计算 PaaS、SaaS 模式，以一体化智能零售解决方案，帮助零售企业数字化智能化升级，实现企业规模化发展。凭借技术、产品、服务优势，慧策现已成为行业影响力品牌并被零售企业广泛认可。2021 年 7 月，慧策完成最新一轮 D 轮 3.12 亿美元的融资，累计完成四轮逾 4.52 亿美元的融资，与诸多国际一流企业、行业头部企业并列而行，被全球顶尖投资公司看好。

业务需求

慧经营是一款数据分析产品，主要提供经营分析、运营分析、应收对账三大功能，面向企业的经营、运营、财务三个角色。如图所示，慧经营上游为常见的电商平台、直播平台以及慧策主要的产品旺店通，这些平台在交易过程中会产生大量的数据，包括收入、成本以及多种费用相关的数据。这些数据都比较分散，如果客户手动进行数据汇总和分析，过程繁琐、难度较高且展示不够直观。而慧经营可以在客户授权下获取、汇总这些数据，根据客户需求提供经营分析、报表生成、账单汇总等多项数据分析服务，以帮助客户更好地经营店铺。



为了能承载上述数据相关的服务，我们在系统架构设计之初需要考虑以下几点：

- **低成本**：慧经营的付费模式为按需付费，这要求架构必须具有轻量化、易维护的特性。基于这点考虑，首先排除基于 Hadoop 的生态架构，虽然 Hadoop 足够成熟，但是组件依赖度较高、繁琐复杂，无论是运维成本还是使用成本都非常高。

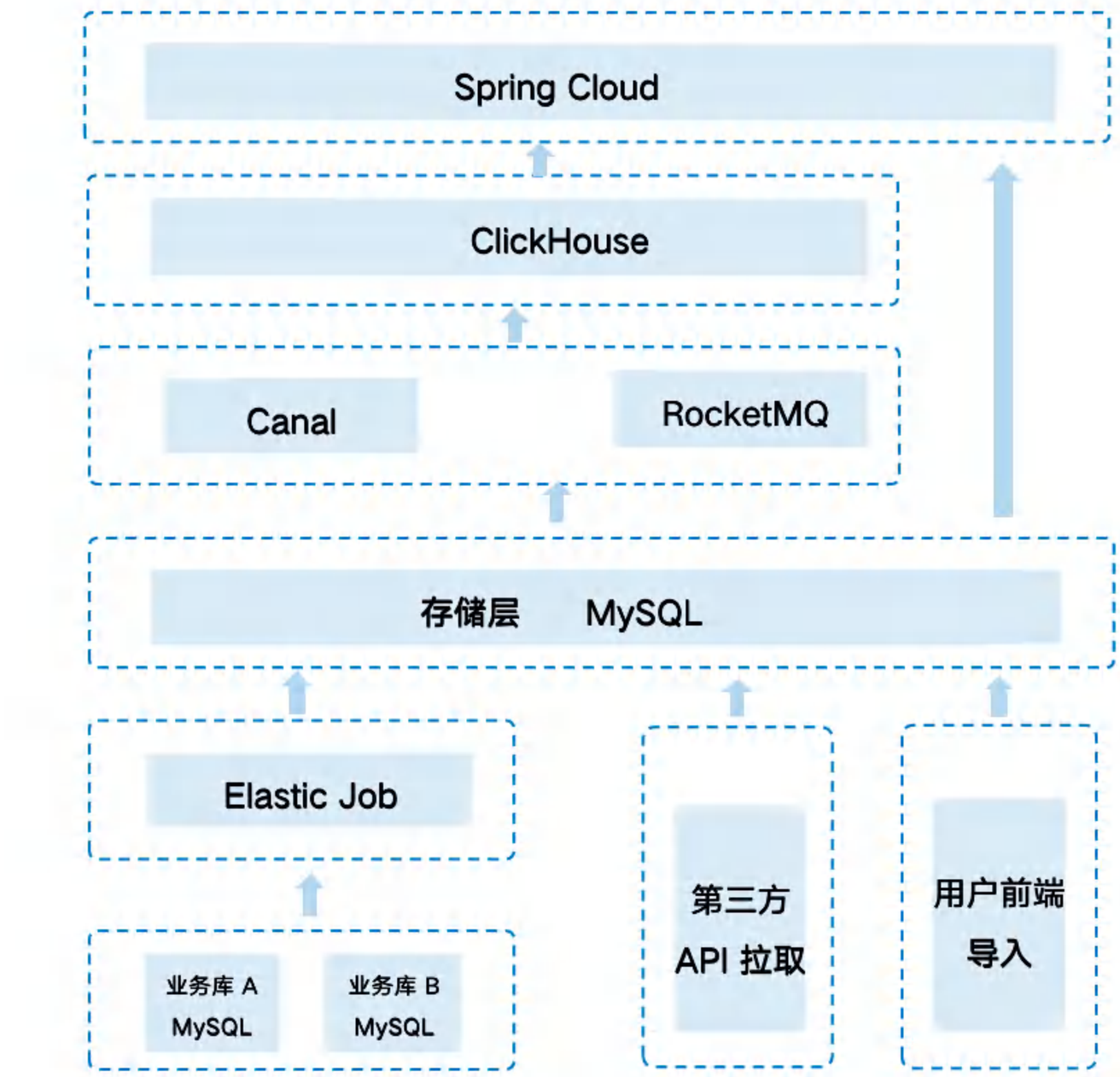
- **高性能**：电商数据为慧经营的主要数据来源，数据维度丰富， ETL 复杂度高；另外，在很多使用场景中明细查询与聚合查询是并存的，既需要够快速定位数据范围，又需要高效的计算能力，因此需要一个强大的 OLAP 引擎来支撑。

- **合理资源分配**：客户规模差异较大，有月单量 1 万的小客户，也有月单量高达千万级别的大客户，资源合理分配在这样的场景下异常重要，因此要求 OLAP 引擎可以在不影响用户体验的前提下，可以根据客户的需求进行资源分配，避免大小查询资源抢占带来的性能问题。

- **高并发**：能够承受上游平台多来源、高并发的大量数据的复杂 ETL，尤其在 618、双 11 这样的业务高峰期，需要具备高效查询能力的同时，能够承担大量的写入负载。

架构演进

架构 1.0



基于以上考虑，我们在架构 1.0 中引入 ClickHouse 作为 OLAP 引擎。从上图可看出该架构是一个完全基于后端 Java 以及 Spring 的技术架构，通过 Binlog 同步的方式通过 Canal 和 Kafka 将数据从 MySQL 同步至 ClickHouse，来承接前端的各种经营分析报表服务。

架构 1.0 在上线不久后就遭遇了慢查询问题。在早期客户和数据量较少时，查询性能尚可满足客户需求，但是随着数据量的不断增大，受限于 MySQL 本身的特性， ETL 效率逐渐低到无法令人接受，尤其在面对大客户场景时，即使命中索引、扫描数据范围仅在百万级别，MySQL 应对该场景也已十分吃力。其次，面对激增的数据，ClickHouse 的多表 Join 也遭遇性能问题，由于其几乎不支持分布式 Join，尽管提供了 Join 语义、但在使用上对多表 Join 的支撑较弱、复杂的关联查询常常会引起 OOM。

在 ClickHouse 单机性能不能支撑业务时，我们考虑扩展搭建集群以提高性能，而 ClickHouse 分布式场景需要依赖 Zookeeper，同时需要单独维护一套分布式表，这将使运维管理的难度和成本不断升高。

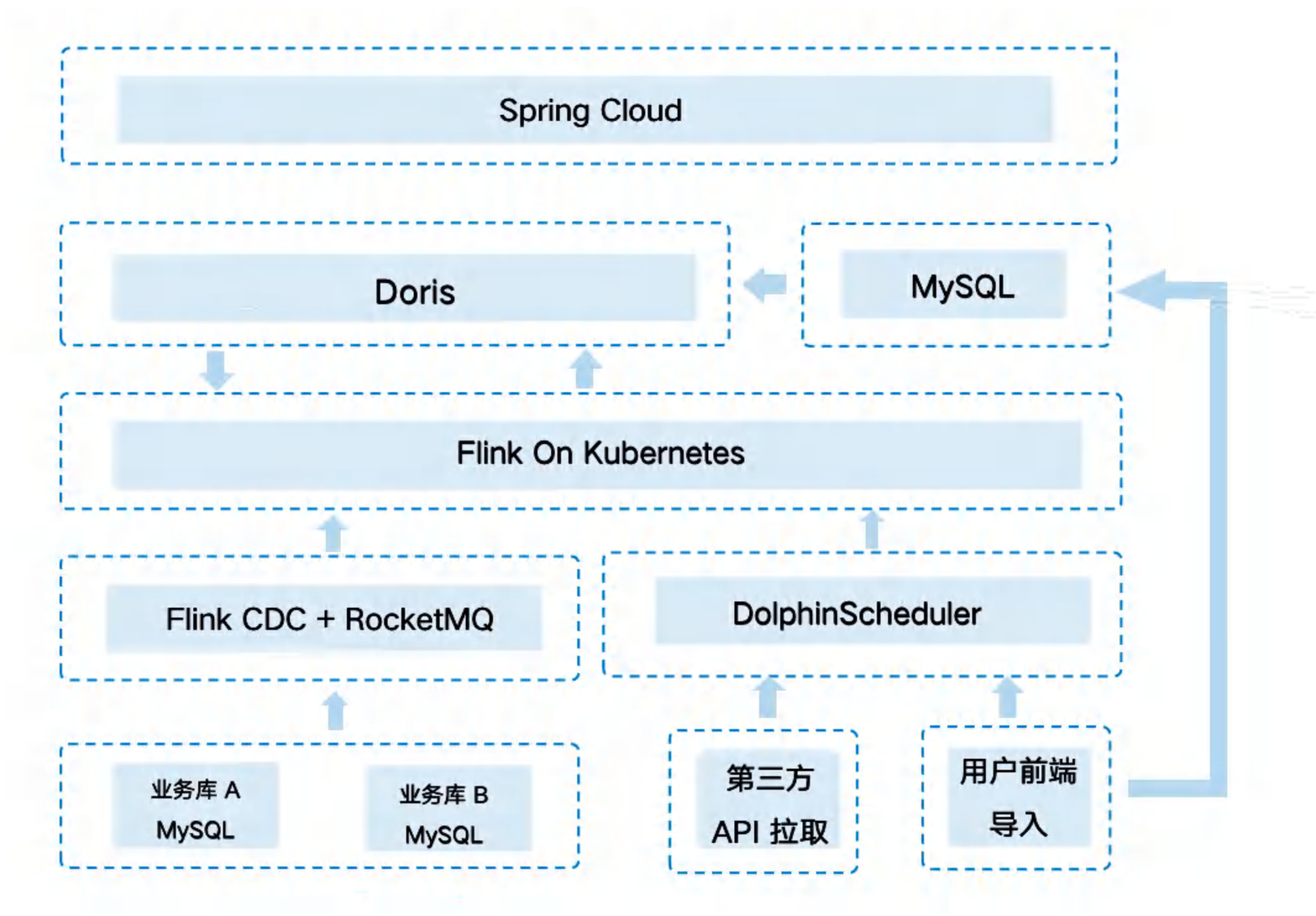
除此之外，ClickHouse 还有几个较大的问题，背离了我们最初对架构的选型要求：

- 不支持高并发，即使一个查询也会用服务器一半的 CPU。对于三副本的集群，通常会将 QPS 控制在 100 以下。
- ClickHouse 的扩容缩容复杂且繁琐，目前做不到自动在线操作，需要自研工具支持。
- ClickHouse 的 ReplacingMergeTree 模型必须添加 final 关键字才能保证严格的唯一性，而设置为 final 后性能会急剧下降。

架构 2.0

为解决以上问题，**我们在架构 2.0 中引入 Apache Doris 作为 OLAP 引擎**，并进行了一次整体的架构升级。选择 Apache Doris 的主要原因有如下几条：

- **使用成本低：**只有 FE 和 BE 两类进程，部署简单，支持弹性扩缩容，不依赖其他组件，运维成本非常低；同时兼容 MySQL协议 和标准 SQL，开发人员学习难度小，项目整体迁移使用成本也比较低。
- **Join 性能优异：**项目存在很多历史慢 SQL ，均为多张大表 Join，Apache Doris 良好的 Join 性能给我们提供了一段优化改造的缓冲期。
- **社区活跃度高：**社区技术氛围浓厚，且 SelectDB 针对社区有专职的技术支持团队，在使用过程中遇到问题均能快速得到响应解决。



如上图所示，我们将原本基于 MySQL 的基础数据存储迁移到 Doris 中，同时将大部分的 ETL 迁移至在 Doris 内部进行，MySQL 只用于存储配置信息。同时引入了 Flink，DolphinScheduler 等组件，成功构建了一套以 Doris 为核心、架构简洁、运维简单的数据生产体系。

全新的架构也给我们带来的显著的收益：

- **ETL 效率极大提升：**慧经营的数据维度丰富、 ETL 复杂度高，得益于 Apache Doris 强大的运算能力及丰富的数据模型，将 ETL 过程放在 Doris 内部进行后效率得到显著提升；
- **Join 耗时大幅降低：**查询性能同样得到极大提升，在 SQL 未经过改造的情况下，多表 Join 查询耗时相较于过去有了大幅降低。
- **并发表现出色：**在业务查询高峰期可达数千 QPS，而 Apache Doris 面对高并发查询时始终表现平稳；
- **存储空间节省：**Apache Doris 的列式存储引擎实现了最高 1:10 的数据压缩率，使得存储成本得到有效降低。
- **运维便捷：**Apache Doris 架构简单、运维成本低，扩容升级也非常方便，我们前后对 Doris 进行了 3 次升级扩容，基本都在很短的时间内完成。

在使用 Apache Doris 的过程中，我们对 Doris 如何更好地应用也进行了探索，在此期间总结出许多实践经验，通过本文分享给大家。

实践应用

分区分桶优化

在执行 SQL 时，SQL 的资源占用和分区分桶的大小有着密切的关系，合理的分区分桶将有效提升资源的利用率，同时避免因资源抢占带来的性能下降。

因此我们在建立事实表分区时，会根据客户的数据规模提供相匹配的分区方案，比如数据规模较小按年分区、规模大按月分区。在这种分区方式下，可以有效避免小查询占用过多资源问题，而大客户的使用体验也由于细粒度的分区方式得到了提升。

```
CREATE TABLE DWD_ORDER_... (
...)
...
PARTITION BY RANGE(tenant,business_date)
(
PARTITION p1_2022_01_01 VALUES [("1", '2022-01-01'), ("1", '2023-01-01')),
PARTITION p2_2022_01_01 VALUES [("2", '2022-01-01'), ("2", '2022-07-01')),
PARTITION p3_2022_01_01 VALUES [("3", '2022-01-01'), ("3", '2022-04-01')),
PARTITION p4_2022_01_01 VALUES [("4", '2022-01-01'), ("4", '2022-02-01')),
...
)
```

而分桶的设置上我们采用了最新版本中增加的**自动分桶推算功能**，使用方式非常便捷，不用再去测算和预估每张表的数据量以及对应 Tablet 的增长关系，系统自动帮助我们推算出合理分桶数，提升性能的同时也使得系统资源得到更好利用。

参考文章：一文教你玩转 Apache Doris 分区分桶新功能 | 新版本揭秘 （也可通过 SelectDB 公众号查看此文章）

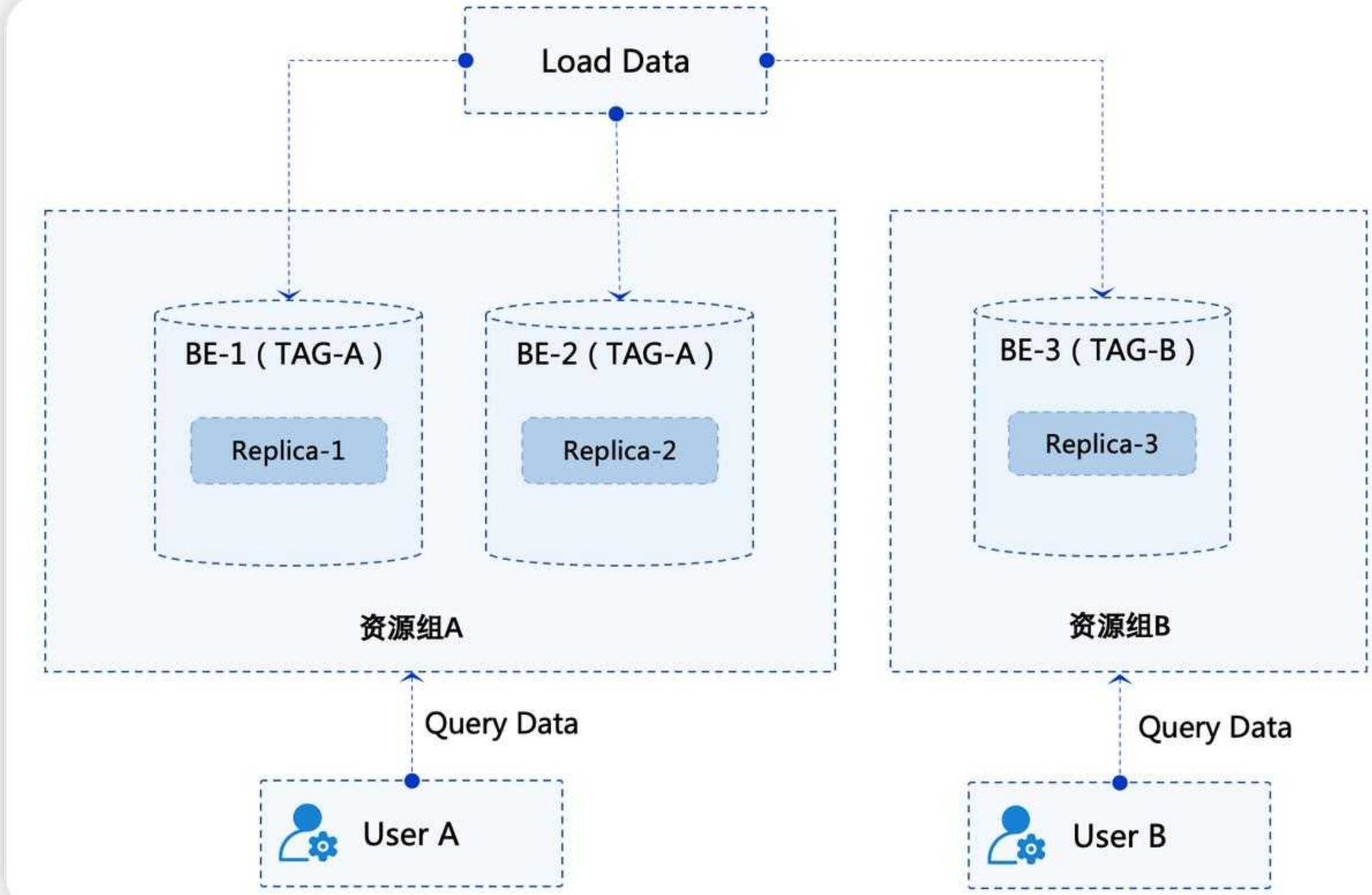
多租户和资源隔离方案

在 SaaS 业务场景中，多租户和资源划分是架构设计过程中必不可少的部分。多租户（Multi-Tenancy）指的是单个集群可以为多个不同租户提供服务，并且仍可确保各租户间数据隔离性。通过多租户可以保证系统共性的部分被共享，个性的部分被单独隔离。

对于我们来说，客户规模差异较大，有月单量只有 1 万的小客户，也有月单量高达千万级别的大客户，资源的合理分配与隔离在这样的场景下异常重要。如果为每个租户创建一个数据库集群，则集群规模不可控且造成资源浪费。如果多个租户共享一套数据库集群，用户的需求可以相互补偿，总体来时可以有很大的资源节约，并且计算规模越大成本越低，而在应对客户查询时，可以根据客户的数据规模和查询需求将资源池划分给各客户。

这里我们采用了 Apache Doris 的节点资源组来区分大小客户，避免了不同租户以及查询间的资源抢占问题。

节点资源划分指将一个 Doris 集群内的 BE 节点设置标签（Tag），标签相同的 BE 节点组成一个资源组（Resource Group），资源组可以看作是数据存储和计算的一个管理单元。数据入库的时候按照资源组配置将数据的副本写入到不同的资源组中，查询的时候按照资源组的划分使用对应资源组上的计算资源对数据进行计算。



例如我们将一个集群中的 3 个 BE 节点划分为 2 个资源组，分别为资源组 A 和资源组 B，BE-1 和 BE-2 属于资源组 A，BE-3 属于资源组 B。资源组 A 有 2 副本的数据，资源组 B 有一个副本的数据。那么当客户 A 在写入数据和查询数据的时候会按照资源组配置使用资源组 A 上的存储和计算资源，当客户 B 写入数据和查询数据的时候会按照资源组配置使用资源组 B 上 的存储和计算资源。这样便能能很好的实现同一个集群，为不同租户提供不同的负载能力。

具体操作如下：

1. 设置标签：

为 BE 节点设置标签。假设当前 Doris 集群有 3 个 BE 节点。分别为 `host[1-3]` 。在初始情况下，所有节点都属于一个默认资源组（Default）。

我们可以使用以下命令将这6个节点划分成3个资源组： `group_a` 、 `group_b` ：

```
1    alter system modify backend "host1:9050" set ("tag.location" = "group_a");
2    alter system modify backend "host2:9050" set ("tag.location" = "group_a");
3    alter system modify backend "host3:9050" set ("tag.location" = "group_b");
```

这里我们将 `host[1-2]` 组成资源组 `group_a` ， `host[3]` 组成资源组 `group_b` 。

2. 设置资源组数据分配策略

副本分布策略定义：资源组划分好后，我们可以将客户数据的不同副本分布在不同资源组内。假设一张用户表 UserTable。我们希望资源组 A 内存放 2 个副本，资源组 B 存放 1 分副本。

```
create table UserTable (k1 int, k2 int)
distributed by hash(k1) buckets 1
properties(
    "replication_allocation"="tag.location.group_a:2, tag.location.group_b:1"
)
```

资源组绑定：通过设置客户的资源组使用权限，来限制某一客户的数据导入和查询只能使用指定资源组中的节点来执行。

```
set property for 'user_a' 'resource_tags.location' = 'group_a';
set property for 'user_b' 'resource_tags.location' = 'group_b';
```

这里将 `user_a` 和 `group_a` 资源绑定，`user_b` 和 `group_b` 资源绑定。绑定完成后，`user_a` 在发起对 UserTable 表的查询时，只会访问 `group_a` 资源组内节点上的数据副本，并且查询仅会使用 `group_a` 资源组内的节点计算资源。

通过 Apache Doris 提供多租户和资源隔离方案，能够将集群资源更合理的分配给各客户，可以让多租户在同一 Doris 集群内进行数据操作时，减少相互之间的干扰，同时达到资源成本的有效降低。

高并发场景的支持

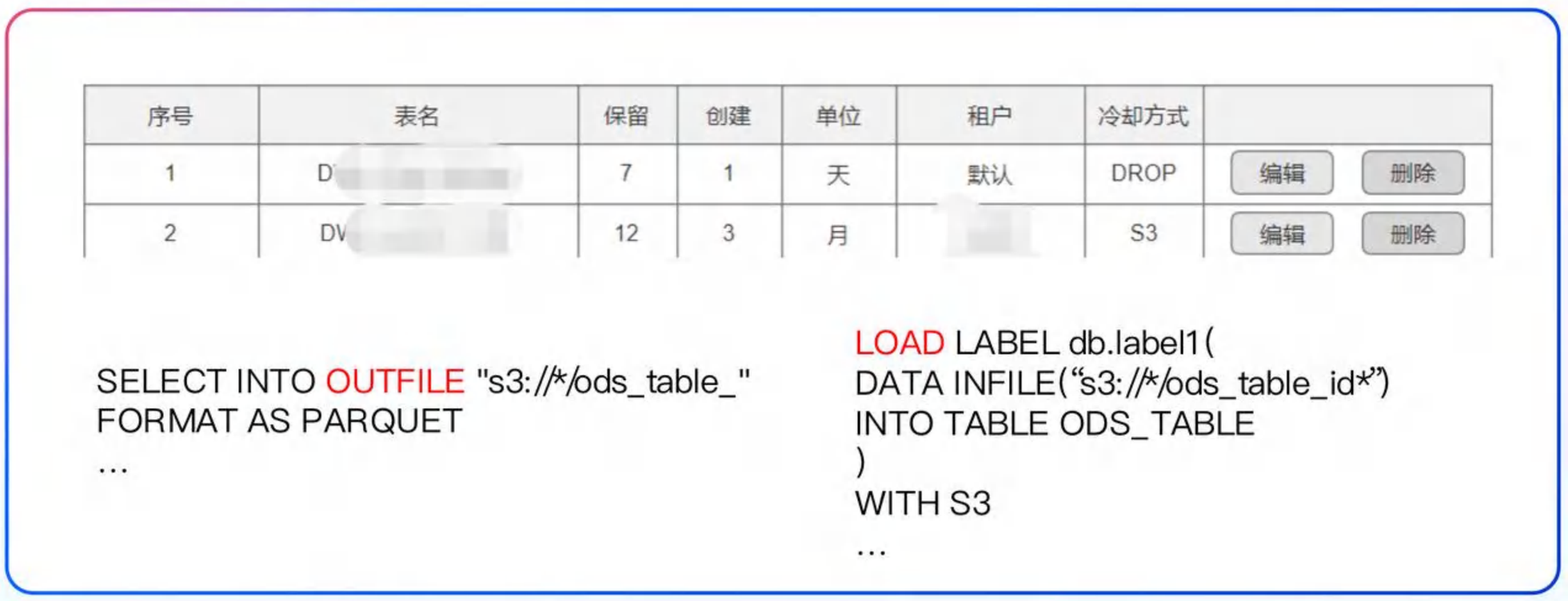
高并发也通常是 SaaS 服务场景面临的挑战，一方面是慧经营已经为众多客户提供了分析服务，需要去同时承载大规模客户的并发查询和访问，另一方面，在每日早晚业务高峰期也会面临更多客户的同时在线，通常 QPS 最高可达数千，而这也是 Apache Doris 表现得非常出色的地方。

在上游 Flink 高频写入的同时，Apache Doris 表现极为平稳，随着查询并发的提升，查询耗时相较于平时几乎没有太大的波动

应对高并发场景的关键在于利用好分区分桶裁剪、索引、缓存等系统机制来减少底层数据扫描。在 Doris 中会将数据表的前几列作为排序键来构建前缀索引，同时还有 ZoneMap 以及 Bloom Filter 等索引。在执行查询时，通过分区分桶裁剪以及索引可以快速过滤到不在查询范围内的数据，减少 CPU 和 IO 的压力。

在与社区的沟通中还了解到，**即将发布的新版本还将进一步提升并发支持，引入行存储格式、短路径优化以及预处理语句等来实现上万 QPS 的超高并发，可以满足更高并发要求的 Data Serving 场景**，这也是我们十分期待的功能。

数据生命周期管理



我们自研了一套生命周期管理工具，基于资源的分配，可以提供客户级定制化冷却策略。根据各个业务场景的客户需求及付费情况，提供不同长度的数据保留策略，同时**得益于 Apache Doris 提供的 OutFile 功能**，实现删除或者导出 S3 两种冷却方式，当我们再次需要这些数据时，可通过 Load 的方式从 S3 导回数据。

除了通过 OutFile 将数据文件导出至 S3 以外，**社区在后续版本中还将提供分区级别的冷热数据分离策略**。当前我们数据分区都是基于日期范围来设置，因此可以利用时间分区进行冷热数据的划分。通过配置分区级别的 Freeze Time，转冷后的历史数据可以自动下沉至成本更低的对象存储上。在面对历史冷数据的查询时，Doris 会自动拉取对象存储上的数据并在本地 Cache 以加速查询，而无需执行导入操作。**通过冷热数据分离以及冷数据 Cache，既能保证最大程度的资源节省，也能保证冷数据的查询性能不受影响。**

总结规划

引入 Apache Doris 之后，新架构的整体查询响应速度都有了较大的提升，存储成本显著降低，后续我们将继续探索 Doris 新特性，进一步实现降本增效。

- 目前正在强化 Apache Doris 的元数据管理建设，我们希望通过该服务能更好的协助开发人员使用 Doris，包括数据血缘追踪等；
- 尝试使用更多 Apache Doris 的新功能，在 1.2.0 版本中新增的 Java UDF 功能可以极大降低数据出入库的频率，更便捷地在 Doris 内部进行数据 ETL，这一功能我们正在尝试使用中；
- 探索更好的资源分配方案，包括 SQL 代理以及尝试 Doris 后续提供的 Spill 功能，以更好进行资源分配应用。

最后，再次感谢 Apache Doris 社区和 SelectDB 技术团队在我们使用过程中提供的许多指导和帮助，提出问题都会及时的响应并尽快协助解决，未来我们也希望深度参与到社区建设中，为社区的发展出一份力。

杭银消金

基于 Apache Doris 的统一数据查询网关改造

作者：杭银消金大数据团队，周其进、唐海定、姚锦权

导读：随着业务量快速增长，数据规模的不断扩大，杭银消金早期的大数据平台在应对实时性更强、复杂度更高的的业务需求时存在瓶颈。为了更好的应对未来的数据规模增长，杭银消金于 2022 年 10 月正式引入 Apache Doris 1.2 对现有的风控数据集市进行了升级改造，利用 Multi Catalog 功能统一了 ES、Hive、GP 等数据源出口，实现了联邦查询，为未来统一数据查询网关奠定了基础；同时，基于 Apache Doris 高性能、简单易用、部署成本低等诸多优势，也使得各大业务场景的查询分析响应实现了从分钟级到秒级的跨越。

杭银消费金融股份有限公司，成立于 2015 年 12 月，是杭州银行牵头组建的浙江省首家持牌消费金融公司，经过这几年的发展，在 2022 年底资产规模突破 400 亿，服务客户数超千万。公司秉承“数字普惠金融”初心，坚持服务传统金融覆盖不充分的、具有消费信贷需求的客户群体，以“数据、场景、风控、技术”为核心，依托大数据、人工智能、云计算等互联网科技，为全国消费者提供专业、高效、便捷、可信赖的金融服务。

业务需求

杭银消金业务模式是线上业务结合线下业务的双引擎驱动模式。为更好的服务用户，运用数据驱动实现精细化管理，基于当前业务模式衍生出了四大类的业务数据需求：

- 预警类：实现业务流量监控，主要是对信贷流程的用户数量与金额进行实时监控，出现问题自动告警。
- 分析类：支持查询统计与临时取数，对信贷各环节进行分析，对审批、授信、支用等环节的用户数量与额度情况查询分析。
- 看板类：打造业务实时驾驶舱与 T+1 业务看板，提供内部管理层与运营部门使用，更好辅助

管理进行决策。

- 建模类：支持多维模型变量的建模，通过算法模型回溯用户的金融表现，提升审批、授信、支用等环节的模型能力。

数据架构 1.0

为满足以上需求，我们采用 Greenplum + CDH 融合的架构体系创建了大数据平台 1.0 ，如下图所示，大数据平台的数据源均来自于业务系统，我们可以从数据源的 3 个流向出发，了解大数据平台的组成及分工：

- 业务系统的核心系统数据通过 CloudCanal 实时同步进入 Greenplum 数仓进行数据实时分析，为 BI 报表，数据大屏等应用提供服务，部分数据进入风控集市 Hive 中，提供查询分析和建模服务。

- 业务系统的实时数据推送到 Kafka 消息队列，经 Flink 实时消费写入 ES，通过风控变量提供数据服务，而 ES 中的部分数据也可以流入 Hive 中，进行相关分析处理。

- 业务系统的风控数据会落在 MongoDB，经过离线同步进入风控集市 Hive， Hive 数仓支撑了查询平台和建模平台，提供风控分析和建模服务。

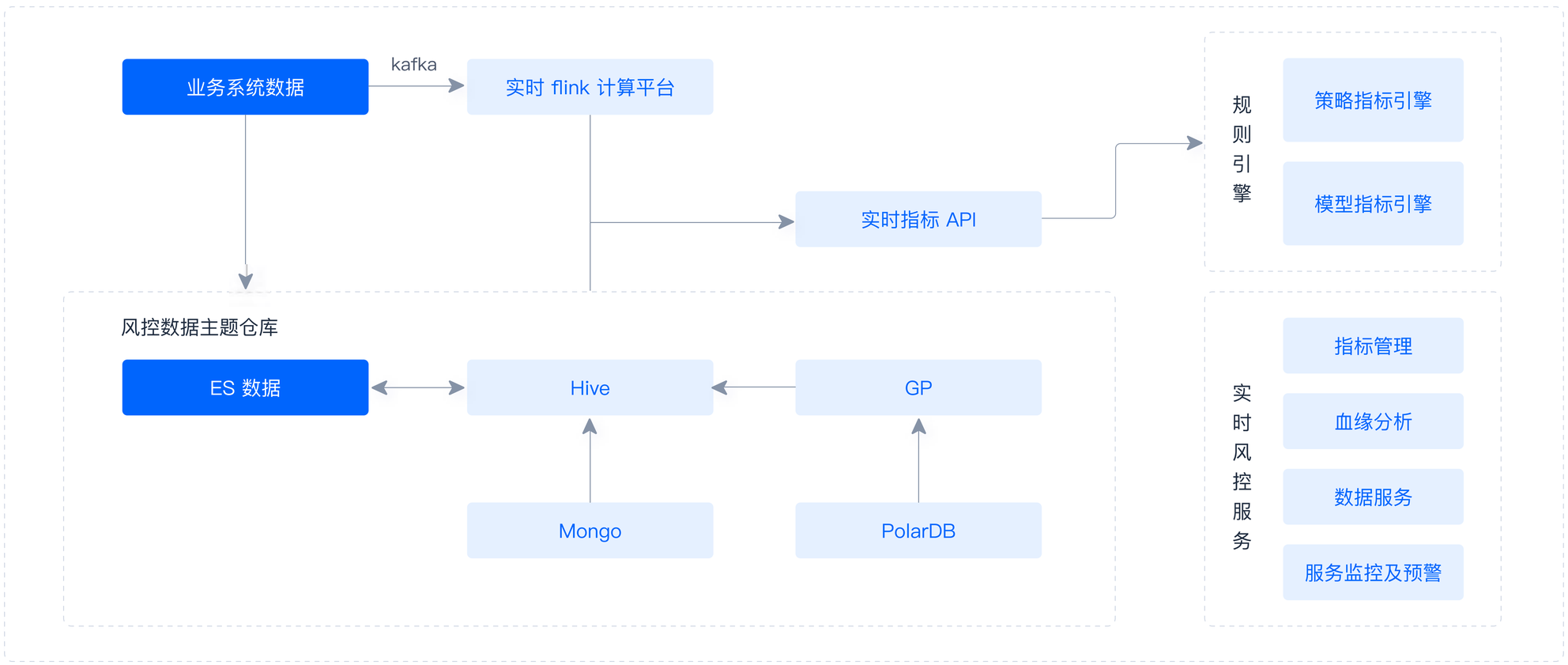


我们将 ES 和 Hive 共同组成了风控数据集市，从上述介绍也可知，四大类的业务需求基本都是

由风控数据集市来满足的，因此我们后续的改造升级主要基于风控数据集市来进行。在这之前，我们先了解一下风控数据集市 1.0 是如何来运转的。

风控数据集市 1.0

风控数据集市原有架构是基于 CDH 搭建的，由实时写入和离线统计分析两部分组成，整个架构包含ES、Hive、Greenplum 等核心组件，风控数据集市的数据源主要有三种：通过 Greenplum 数仓同步的业务系统数据、通过 MongoDB 同步的风控决策数据，以及通过 ES 写入的实时风控变量数据。



实时流数据：采用了 Kafka + Flink + ES 的实时流处理方式，利用 Flink 对 Kafka 的实时数据进行清洗，实时写入 ES，并对部分结果进行汇总计算，通过接口提供给风控决策使用。

离线风控数据：采用基于 CDH 的方案实现，通过 Sqoop 离线同步核心数仓 GP 上的数据，结合实时数据与落在 MongoDB 上的三方数据，经数据清洗后统一汇总到 Hive 数仓进行日常的跑批与查询分析。

需求满足情况：

在大数据平台 1.0 的的支持下，我们的业务需求得到了初步的实现：

- 预警类：基于 ES + Hive 的外表查询，实现了实时业务流量监控；
- 分析类：基于 Hive 实现数据查询分析和临时取数；
- 看板类：基于 Tableau + Hive 搭建了业务管理驾驶舱以及 T+1 业务看板；

- 建模类：基于 Spark + Hive 实现了多维模型变量的建模分析；

受限于 Hive 的执行效率，以上需求均在分钟级别返回结果，仅可以满足我们最基本的诉求，而面对秒级甚至毫秒级的分析场景，Hive 则稍显吃力。

存在的问题：

- 单表宽度过大，影响查询性能。风控数据集市的下游业务主要以规则引擎与实时风控服务为主，因规则引擎的特殊性，公司在数据变量衍生方面资源投入较多，某些维度上的衍生变量会达到几千甚至上万的规模，这将导致 Hive 中存储的数据表字段非常多，部分经常使用的大宽表字段数量甚至超过上千，过宽的大宽表非常影响实际使用中查询性能。

- 数据规模庞大，维护成本高。目前 Hive 上的风控数据集市已经有存量数据在百 T 以上，面对如此庞大的数据规模，使用外表的方式进行维护成本非常高，数据的接入成为一大难题。

- 接口服务不稳定。由风控数据集市离线跑批产生的变量指标还兼顾为其他业务应用提供数据服务的职责，目前 Hive 离线跑批后的结果会定时推送到 ES 集群（每天更新的数据集比较庞大，接口调用具有时效性），推送时会因为 IO 过高触发 ES 集群的 GC 抖动，导致接口服务不稳定。

除此之外，风控分析师与建模人员一般通过 Hive & Spark 方式进行数据分析建模，这导致随着业务规模的进一步增大， T+1 跑批与日常分析的效率越来越低，风控数据集市改造升级的需求越发强烈。

技术选型

基于业务对架构提出的更高要求，我们期望引入一款强劲的 OLAP 引擎来改善架构，因此我们于 2022 年 9 月份对 ClickHouse 和 Apache Doris 进行了调研，调研中发现 Apache Doris 具有高性能、简单易用、实现成本低等诸多优势，而且 Apache Doris 1.2 版本非常符合我们的诉求，原因如下：

- 宽表查询性能优异：从官方公布的测试结果来看， 1.2 Preview 版本在 SSB-Flat 宽表场景上相对 1.1.3 版本整体性能提升了近 4 倍、相对于 0.15.0 版本性能提升了近 10 倍，在 TPC-H 多表关联场景上较 1.1.3 版本上有近 3 倍的提升、较 0.15.0 版本性能提升了 11 倍以上，多个场景性能得到飞跃性提升。

- 便捷的数据接入框架以及联邦数据分析能力：Apache Doris 1.2 版本推出的 Multi Catalog 功能可以构建完善可扩展的数据源连接框架，便于快速接入多类数据源，提供基于各种异构数据源的联邦查询和写入能力。目前 Multi-Catalog 已经支持了 Hive、Iceberg、Hudi 等数据湖以及 MySQL、Elasticsearch、Greenplum 等数据库，全面覆盖了我们现有的组件栈，基于此能力有希望通过 Apache Doris 来打造统一数据查询网关。

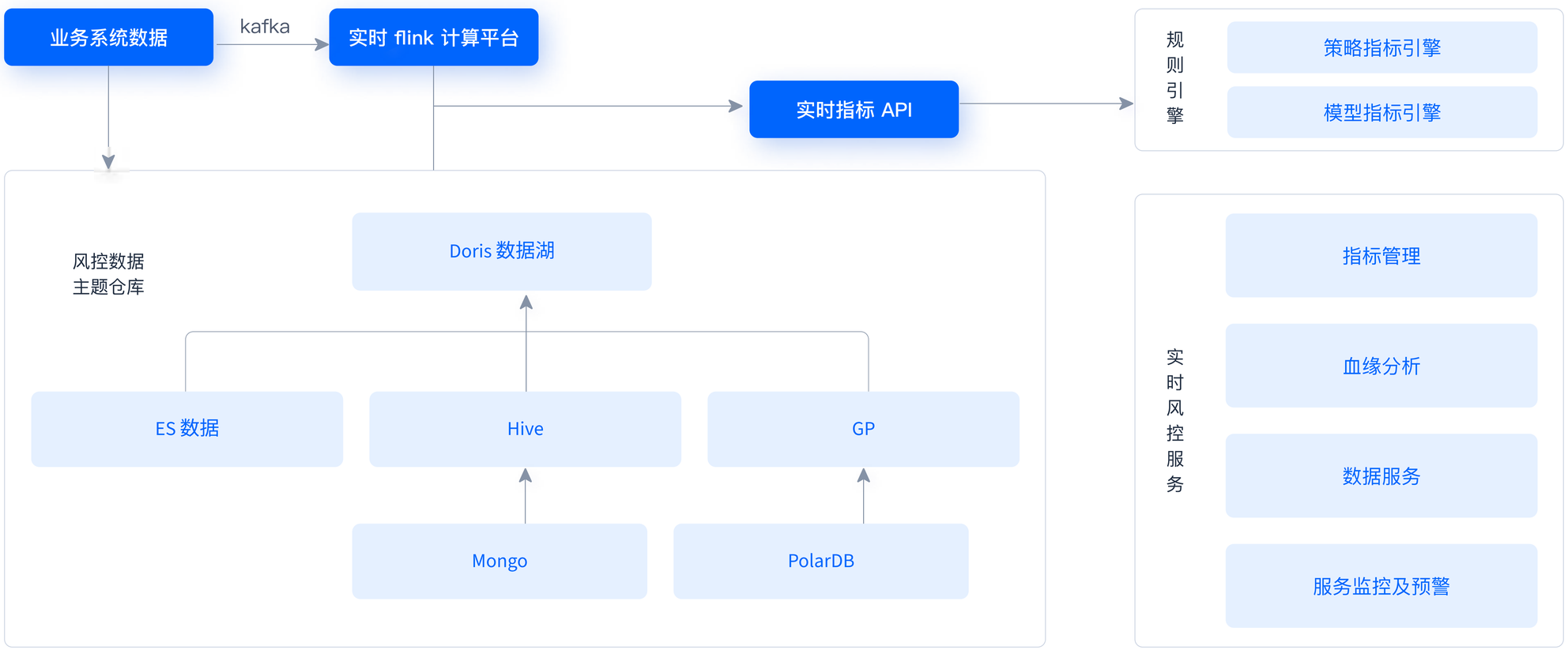
- 生态丰富：支持 Spark Doris Connector、Flink Doris Connector，方便离线与实时数据的处理，缩短了数据处理链路耗费的时间。

- 社区活跃：Apache Doris 社区非常活跃，响应迅速，并且 SelectDB 为社区提供了一支专职的工程师团队，为用户提供技术支持服务。

数据架构 2.0

风控数据集市 2.0

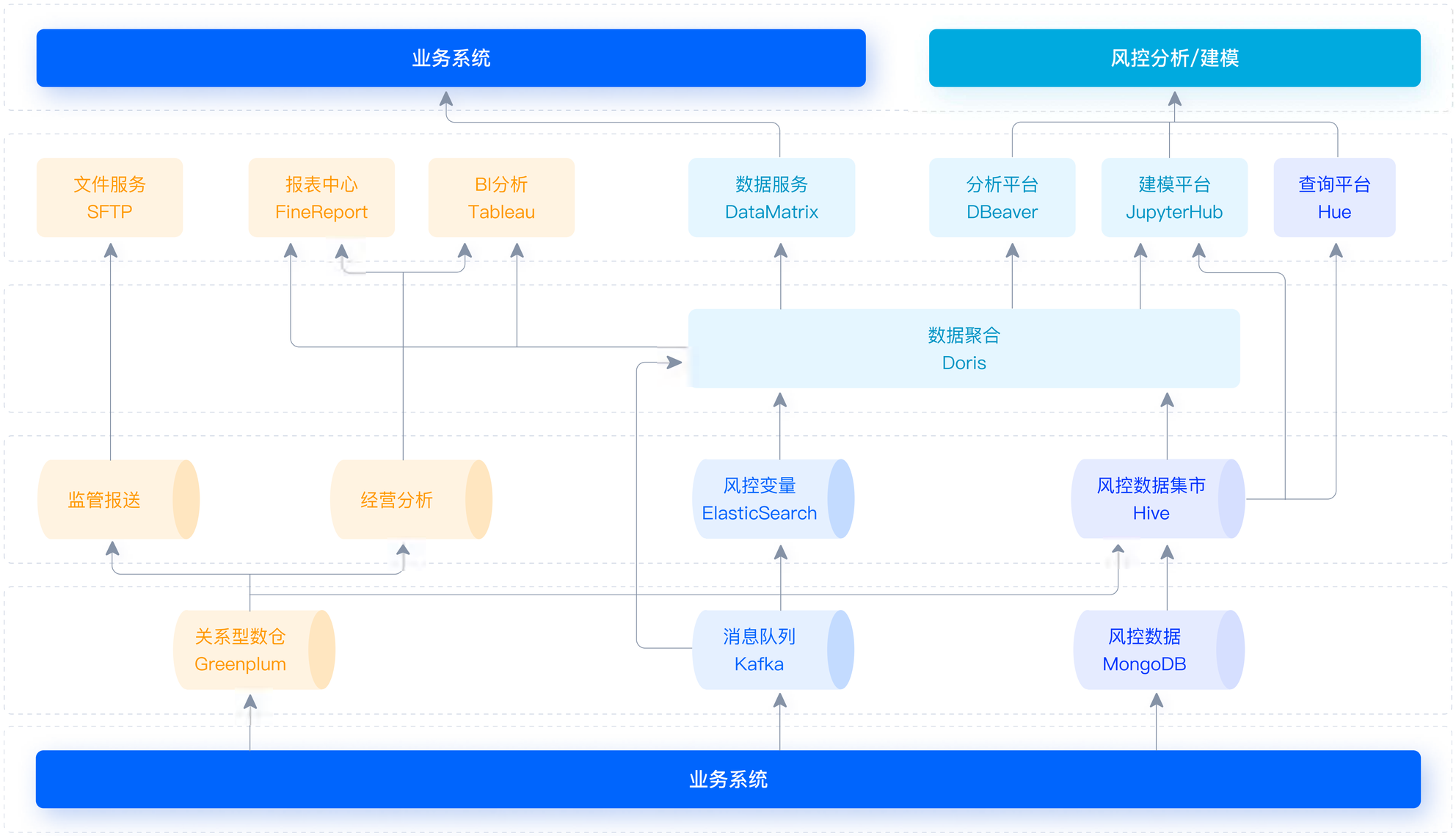
基于对 Apache Doris 的初步的了解与验证，22 年 10 月在社区的支持下我们正式引入 Apache Doris 1.2.0 Preview 版本作为风控数据集市的核心组件，**Apache Doris 的 Multi Catalog 功能助力大数据平台统一了 ES、Hive、Greenplum 等数据源出口，通过 Hive Catalog 和 ES Catalog 实现了对 Hive & ES 等多数据源的联邦查询，**并且支持 Spark-Doris-Connector，可以实现数据 Hive 与 Doris 的双向流动，与现有建模分析体系完美集成，在短期内实现了性能的快速提升。



大数据平台 2.0

风控数据集市调整优化之后，大数据平台架构也相应的发生了变化，如下图所示，仅通过 Doris 一个组件即可为数据服务、分析平台、建模平台提供数据服务。

在最初进行联调适配的时候，Doris 社区和 SelectDB 支持团队针对我们提出的问题和疑惑一直保持高效的反馈效率，给于积极的帮助和支持，快速帮助我们解决在生产上遇到的问题。



需求满足情况：

在大数据平台 2.0 的加持下，业务需求实现的方式也发生了变更，主要变化如下所示

- 预警类：基于 ES Catalog + Doris 实现了对实时数据的查询分析。在架构 1.0 中，实时数据落在 ES 集群上，通过 Hive 外表进行查询分析，查询结果以分钟级别返回；而在 Doris 1.2 集成之后， 使用 ES Catalog 访问 ES，可以实现对 ES 数据秒级统计分析。

- 分析类：基于 Hive Catalog + Doris 实现了对现有风控数据集市的快速查询。目前 Hive 数据集市存量表在两万张左右，如果通过直接创建 Hive 外部表的方式，表结构映射关系的维护难度与数据同步成本使这一方式几乎不可能实现。而 Doris 1.2 的 Multi Catalog 功能则完美解决了这个问题，只需要创建一个 Hive Catalog，就能对现有风控数据集市进行查询分析，既能提升查询性能，还减少了日常查询分析对跑批任务的资源影响。

- 看板类：基于 Tableau + Doris 聚合展示业务实时驾驶舱和 T+1 业务看板，最初使用 Hive

时报表查询需要几分钟才能返回结果，而 Apache Doris 则是秒级甚至是毫秒级的响应速度。

- 建模类：基于 Spark+Doris 进行聚合建模。利用 Doris 1.2 的 Spark-Doris-Connector 功能，实现了 Hive 与 Doris 数据双向同步，满足了 Spark 建模平台的功能复用。同时增加了 Doris 数据源，基础数据查询分析的效率得到了明显提升，建模分析能力的也得到了增强。

在 Apache Doris 引入之后，以上四个业务场景的查询耗时基本都实现了从分钟级到秒级响应的跨越，性能提升十分巨大。

生产环境集群监控

为了快速验证新版本的效果，我们在生产环境上搭建了两个集群，目前生产集群的配置是 4 个 FE + 8 个 BE，单个节点是配置为 64 核 + 256G + 4T，备用集群为 4 个 FE + 4 个 BE 的配置，单个节点配置保持一致。

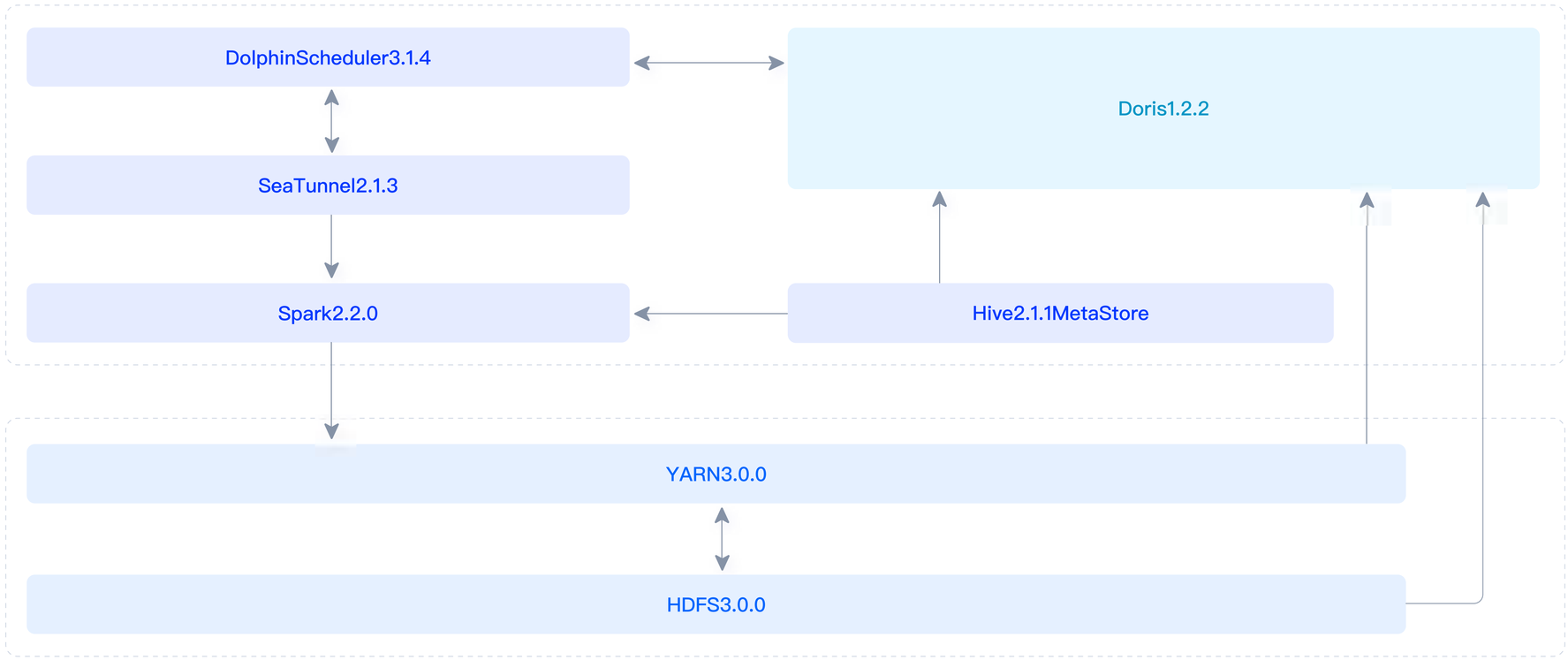
集群监控如下图所示：



可以看出，Apache Doris 1.2 的查询效率非常高，原计划至少要上 10 个节点，而在实际使用下来，我们发现当前主要使用的场景均是以 Catalog 的方式查询，因此集群规模可以相对较小就可以快速上线，也不会破坏当前的系统架构，兼容性非常好。

数据集成方案

前段时间，Apache Doris 1.2.2 版本已经发布，为了更好的支撑应用服务，我们使用 Apache Doris 1.2.2 与 DolphinScheduler 3.1.4 调度器、SeaTunnel 2.1.3 数据同步平台等开源软件实现了集成，以便于数据定时从 Hive 抽取到 Doris 中。整体的数据集成方案如下：



在当前的硬件配置下，数据同步采用的是 DolphinScheduler 的 Shell 脚本模式，定时调起 SeaTunnel 的脚本，数据同步任务的配置文件如下：

```
 env{
  spark.app.name = "hive2doris-template"
  spark.executor.instances = 10
  spark.executor.cores = 5
  spark.executor.memory = "20g"
}
spark {
  spark.sql.catalogImplementation = "hive"
}
source {
  hive {
    pre_sql = "select * from ods.demo_tbl where dt='2023-03-09'"
    result_table_name = "ods_demo_tbl"
  }
}

transform {
}

sink {
  doris {
      fenodes = "192.168.0.10:8030,192.168.0.11:8030,192.168.0.12:8030,192.168.0.13:8030"
      user = root
      password = "XXX"
      database = ods
      table = ods_demo_tbl
      batch_size = 500000
      max_retries = 1
      interval = 10000
      doris.column_separator = "\t"
    }
}
```

该方案成功实施后，资源占用、计算内存占用有明显的降低，查询性能、导入性能有大幅提升：

1. 存储成本降低

使用前：Hive 原始表包含 500 个字段，单个分区数据量为 1.5 亿/天，在 HDFS 上占用约 **810G** 的空间。

使用后：我们通过 SeaTunnel 调起 Spark on YARN 的方式进行数据同步，可以在 40 分钟左右

完成数据同步，同步后数据占用 **270G** 空间，**存储资源仅占之前的 1/3。**

2. 计算内存占用降低，性能提升显著

使用前：上述表在 Hive 上进行 Group By 时，占用 YARN 资源 720 核 1.44T 内存，需要 **162 秒**才可返回结果；

使用后

- 通过 Doris 调用 Hive Catalog 进行聚合查询，在设置 `set exec_mem_limit=16G` 情况下用时 **58.531 秒**，查询耗时较之前减少了近 2/3；
- 在同等条件下，在 Doris 中执行相同的的操作可以在 **0.828 秒**就能返回查询结果，性能增幅巨大。

具体效果如下：

使用后

（1）Hive 查询语句，用时 162 秒。

```
select count(*),product_no   FROM ods.demo_tbl where dt='2023-03-09'
group by product_no;
```

（2）Doris 上 Hive Catalog 查询语句，用时 58.531 秒。

```
set exec_mem_limit=16G;
select count(*),product_no   FROM hive.ods.demo_tbl where dt='2023-03-09'
group by product_no;
```

（3）Doris 上本地表查询语句，**仅用时 0.828 秒**。

```
select count(*),product_no   FROM ods.demo_tbl where dt='2023-03-09'
group by product_no;
```

3. 导入性能提升

使用前：Hive 原始表包含 40 个字段，单个分区数据量 11 亿/天，在 HDFS 上占用约 806G 的空间

使用后: 通过 SeaTunnel 调起 Spark on YARN 方式进行数据同步，可以在 11 分钟左右完成数据同步，**即 1 分钟同步约一亿条数据，**同步后占用 378G 空间。

可以看出，在数据导入性能的提升的同时，资源也有了较大的节省，主要得益于对以下几个参数进行了调整：

`push_write_mbytes_per_sec` BE 磁盘写入限速，300M

`push_worker_count_high_priority` 同时执行的 Push 任务个数，15

`push_worker_count_normal_priority` 同时执行的 Push 任务个数，15



新架构收益

（1）统一数据源出口，查询效率显著提升

风控数据集市采用的是异构存储的方式来存储数据，Apache Doris 的 Multi Catalog 功能成功统一了 ES、Hive、GP 等数据源出口，实现了联邦查询。同时，Doris 本身具有存储能力，可支持其他数据源中的数据通过外表插入内容的方式快速进行数据同步，真正实现了数据门户。此外，Apache Doris 可支持聚合查询，在向量化引擎的加持下，查询效率得到显著提升。

（2）Hive 任务拆分，提升集群资源利用率

我们将原有的 Hive 跑批任务跟日常的查询统计进行了隔离，以提升集群资源的利用效率。目前 YARN 集群上的任务数量是几千的规模，跑批任务占比约 60%，临时查询分析占比 40%，由于资源限制导致日常跑批任务经常会因为资源等待而延误，临时分析也因资源未及时分配而导致任务无法完成。当部署了 Doris 1.2 之后，对资源进行了划分，完全摆脱 YARN 集群的资源限制，跑批与日常的查询统计均有了明显的改善，基本可以在秒级得到分析结果，同时也减轻了数据分析师的工作压力，提升了用户对平台的满意度。

（3）提升数据接口的稳定性，数据写入性能大幅提升

之前数据接口是基于 ES 集群的，当进行大批量离线数据推送时会导致 ES 集群的 GC 抖动，影响了接口稳定性，经过调整之后，我们将接口服务的数据集存储在 Doris 上，Doris 节点并未出现抖动，实现数据快速写入，成功提升了接口的稳定性，同时 Doris 查询在数据写入时影响较小，数据写入性能较之前也有了非常大的提升，千万级别的数据可在十分钟内推送成功。

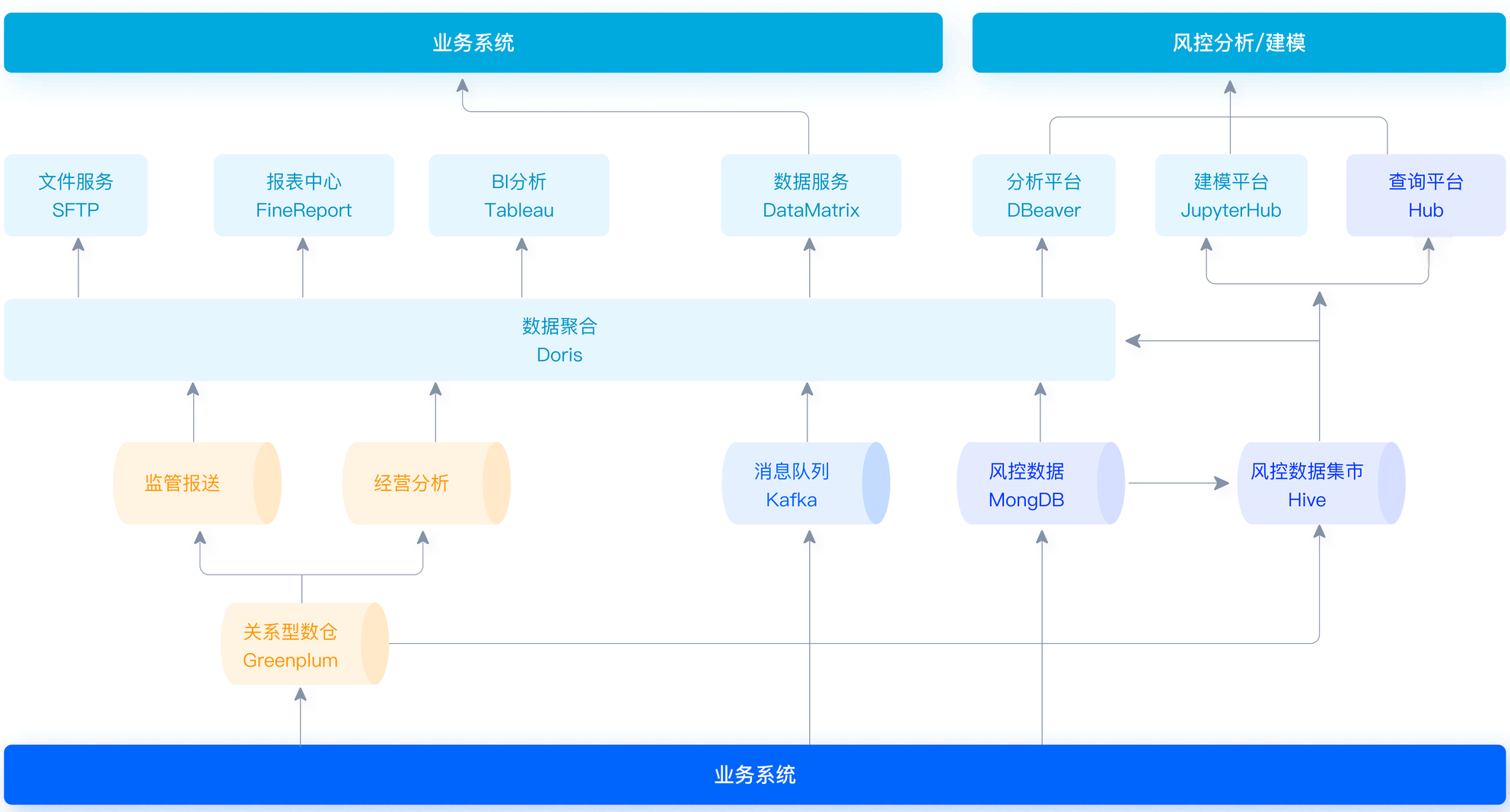
（4）Doris 生态丰富，迁移方便成本较低

Spark-Doris-Connector 在过渡期为我们减轻了不少的压力，当数据在 Hive 与 Doris 共存时，部分 Doris 分析结果通过 Spark 回写到 Hive 非常方便，当 Spark 调用 Doris 时只需要进行简单改造就能完成原有脚本的复用，迁移方便、成本较低。

（5）支持横向热部署，集群扩容、运维简单。

Apache Doris 支持横向热部署，集群扩容方便，节点重启可以在在秒级实现，可实现无缝对接，减少了该过程对业务的影响；在架构 1.0 中，当 Hive 集群与 GP 集群需要扩容更新时，配置修改后一般需要较长时间集群才可恢复，用户感知比较明显。而 Doris 很好的解决了这个问题，实现用户无感知扩容，也降低了集群运维的投入。

未来规划



当前在架构 2.0 中的 Doris 集群在大数据平台中的角色更倾向于查询优化，大部分数据还集中维护在 Hive 集群上，未来我们计划在升级架构 3.0 的时候，完成以下改造：

- 实时全量数据接入：利用 Flink 将所有的实时数据直接接入 Doris，不再经过 ES 存储；
- 数据集数据完整性：利用 Doris 构建实时数据集市的原始层，利用 FlinkCDC 等同步工具将业务库 MySQL 与决策过程中产生的 MongoDB 数据实时同步到 Doris，最大限度将现有数据都接入 Doris 的统一平台，保证数据集数据完整性。
- 离线跑批任务迁移：将现有 Hive & Spark 中大部分跑批任务迁移至 Doris，提升跑批效率；
- 统一查询分析出口：将所有的查询分析统一集中到 Doris，完全统一数据出口，实现统一数据查询网关，使数据的管理更加规范化；
- 强化集群稳定扩容：引入可视化运维管理工具对集群进行维护和管理，使 Doris 集群能够更加稳定支撑业务扩展。

总结与致谢

Apache Doris 1.2 是社区在版本迭代中的重大升级，借助 Multi Catalog 等优异功能能让 Doris 在 Hadoop 相关的大数据体系中快速落地，实现联邦查询；同时可以将日常跑批与统计分析进行解耦，有效提升大数据平台的的查询性能。

作为第一批 Apache Doris 1.2 的用户，我们深感荣幸，同时也十分感谢 Doris 团队的全力配合和付出，可以让 Apache Doris 快速落地、上线生产，并为后续的迭代优化提供了可能。

Apache Doris 1.2 值得大力推荐，希望大家都能从中受益，祝愿 Apache Doris 生态越来越繁荣，越来越好！

并发提升 10 倍，领健从 ClickHouse 和 Kudu 到 Apache Doris 数仓升级实践

作者：领健资深大数据开发工程师，杨鹭

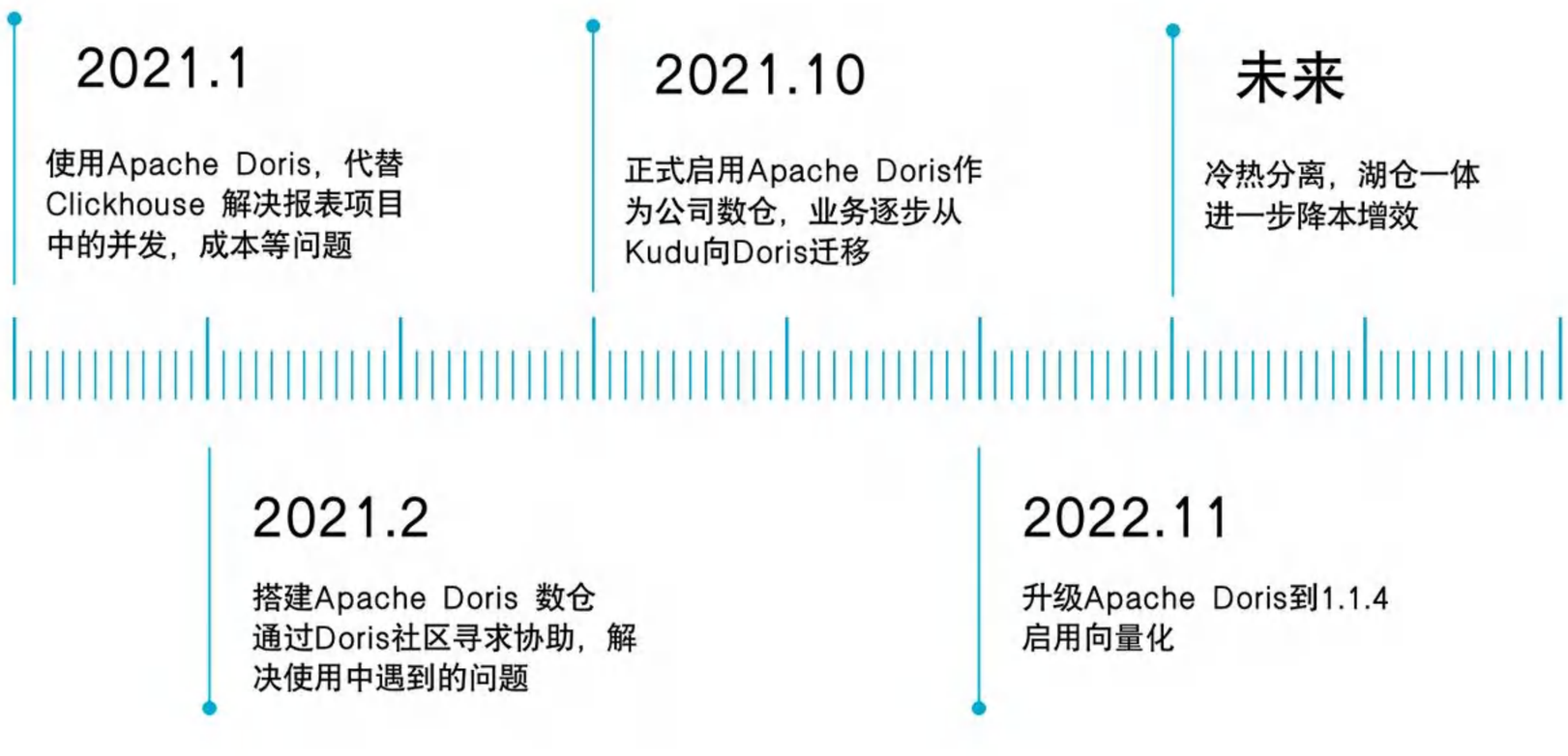
导读：领健最初使用 Apache Doris 替代了 ClickHouse，应用在为客户提供的增值报表服务中，解决了ClickHouse 并发宕机问题；在项目服务中发掘出 Doris 查询性能优异、简单易用、部署成本低等诸多优势，在 2021 年 10 月，领健决定将 Doris 引入到公司的数仓替代 Kudu，在 Doris 社区及 SelectDB 专业技术团队的支持下，业务逐步从 Kudu 迁移到 Doris，并在最近升级到 1.1.4 向量化版本。我们将通过本文为大家详细介绍领健基于 Doris 的演进实践及数仓构建的经验。

领健是健康科技行业 SaaS 软件的引领者，专注于消费医疗口腔和医美行业，为口腔诊所、医美机构、生美机构提供经营管理一体化系统，提供了覆盖单店管理、连锁管理、健康档案/电子病历、客户关系管理、智能营销、B2B 交易平台、进销存、保险支付、影像集成、BI 商业智能等覆盖机构业务全流程的一体化 SaaS 软件。同时通过开放平台连接产业上下游，与优质的第三方平台合作，为机构提供完整配套的一站式服务。截至当前，领健已经在全国设立了 20 余个分支机构，超过 30000 多家中高端以及连锁机构正在使用其服务。



Doris 在领健的演进历程

在进入正文之前，简单了解一下 Doris 在领健的发展历程。最初 Doris 替代 ClickHouse 被应用到数据服务项目中，该项目是领健为旗下客户提供的增值报表服务；后在项目服务中发掘出 Doris 查询性能优异、简单易用、部署成本低等诸多优势，在 2021 年 10 月，我们决定扩大 Doris 应用范围，将 Doris 引入到公司的数仓中，在 Doris 社区及 SelectDB 专业技术团队的支持下，业务逐步从 Kudu 迁移到 Doris，并在最近升级到 1.1.4 向量化版本。我们将通过本文为大家详细介绍领健基于 Doris 的演进实践及数仓构建的经验。



数据服务架构演进

项目需求

领健致力于为医疗行业客户提供精细化门店运营平台，为客户提供了数据报表工具，该工具可实现自助式拖拽设计图表、支持多种自带函数自建、数据实时更新等功能，可以支持门店订单查询、客户管理、收入分析等，以推动门店数字化转型，辅助门店科学决策。为更好实现以上功能，数据报表工具需满足以下特点：

- **支持复杂查询：**客户进行自助拖拽设计图表时，将生成一段复杂的 SQL 查询语句直查数据库，且语句复杂度未知，这将对数据库带来不小的压力，从而影响查询性能。

- **高并发低延时：**至少可以支撑 **100** 个并发，并在 **1 秒**内得到查询结果；

- **数据实时同步：**报表数据源自于 SaaS 系统，当客户对系统中的历史数据进行修改后，报表数据也要进行同步更改，保持一致，这就要求报表数据要与系统实现实时同步。

- **低成本易部署：**SaaS 业务存在私有云客户，为降低私有化部署的人员及成本投入，这要求架构部署及运维要足够简单。

ClickHouse 遭遇并发宕机

最初项目选用 ClickHouse 来提供数据查询服务，**但在运行过程中 ClickHouse 遭遇了严重的并发问题，即 10 个并发就会导致 ClickHouse 宕机**，这使其无法正常为客户提供服务，这是迫使我们寻找可以替代 ClickHouse 产品的关键因素。

除此之外还有 2 个较为棘手的问题：

- 云上 ClickHouse 服务成本非常高，且 ClickHouse 组件依赖性较高，数据同步时 ClickHouse 和 Zookeeper 的频繁交互，会对稳定性产生较大的压力。

- 如何进行无缝迁移，不影响客户正常使用。

技术选型

针对存在的问题及需求，我们决定进行技术选型，分别对 Doris（0.14）、ClickHouse、Kudu 这 3 个产品展开的调研测试。

对比测试	Doris	ClickHouse	Kudu
高并发	好	差	差
查询性能	好	单表良好	一般
数据更新	支持	支持	支持
使用成本	低	高	高
标准 SQL	支持	不完全支持	不完全支持
分布式 Join	支持	几乎不支持	支持
社区活跃度	非常活跃	俄方维护，沟通成本高	沉寂

如上表所示，我们对这 3 个产品进行了横向比较，**可以看出 Doris 在多方面表现优异：**

- **高并发：**Doris 并发性好，可支持上百甚至上千并发，轻松解决 10 并发导致 ClickHouse 宕机问题。

- **查询性能：**Doris 可实现毫秒级查询响应，在单表查询中，虽 Doris 与 ClickHouse 查询性能基本持平，但在多表查询中，Doris 远胜于 ClickHouse ，Doris 可以实现在较高的并发下，QPS 不下降。

- **数据更新：**Doris 的数据模型可以满足我们对数据更新的需求，以保障系统数据和业务数据的一致性，下文将详细介绍。

- **使用成本：**Doris 架构简单，整体部署简单快速，具有完备的导入功能，很好的弹性伸缩能力；同时， Doris 内部可以自动做副本平衡，运维成本极低。而 Clickouse 及 Kudu 对组件依赖较高，在使用上需要做许多准备工作，这就要求具备一支专业的运维支持团队来处理大量的日常运维工作。

- **标准 SQL：**Doris 兼容 MySQL 协议，使用标准 SQL，开发人员上手简单，不需要付出额外的学习成本。

- **分布式 Join ：** Doris 支持分布式 Join，而 ClickHouse 由于 Join 查询限制、函数局限性、以及可维护性较差等原因，不满足我们当前的业务需求。

- **社区活跃：**Apache Doris 是国内自研数据库，开源社区相当活跃，同时 SelectDB 为 Doris 社区提供了专业且全职团队做技术支持，遇到问题可以直接与社区联系沟通，并能得到快速解决，这对于国外的项目，很大地降低与社区沟通的语言与时间成本。

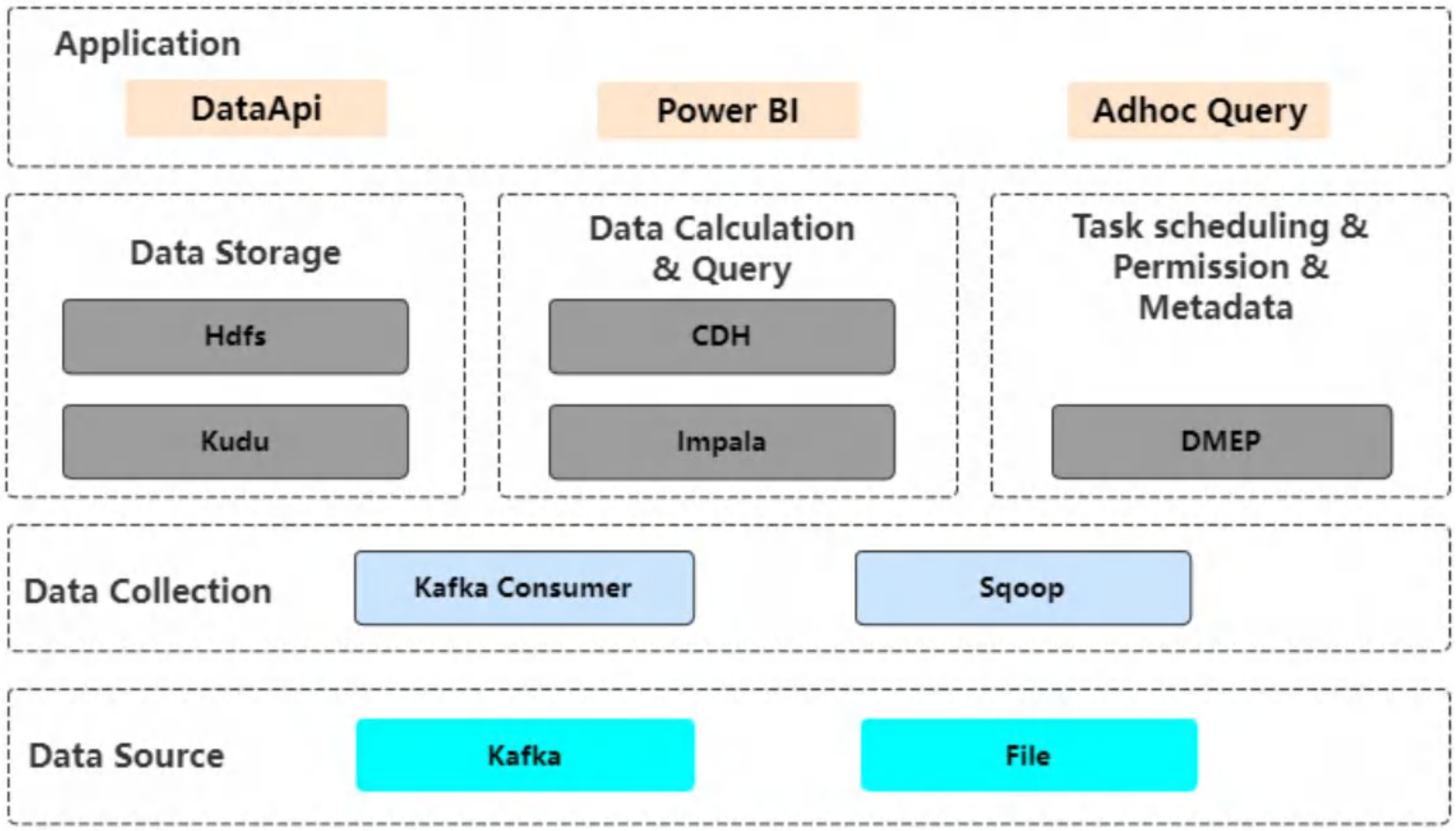
从以上调研中可以发现，Doris 各方面能力优秀，十分符合我们对选型产品的需求，因此**我们使用 Doris 替代了 ClickHouse ，解决了 ClickHouse 并发性能差、宕机等问题，很好的支撑了数据报表查询服务。**

数仓架构演进

在数据报表的使用过程中，我们逐渐发掘出 Doris 诸多优势，因此决定扩大 Doris 应用范围，将 Doris 引入到公司的数仓中来。**接下来将为大家介绍公司数仓从 Kudu 到 Doris 的演进历程，以及在搭建过程中的优化实践分享。**

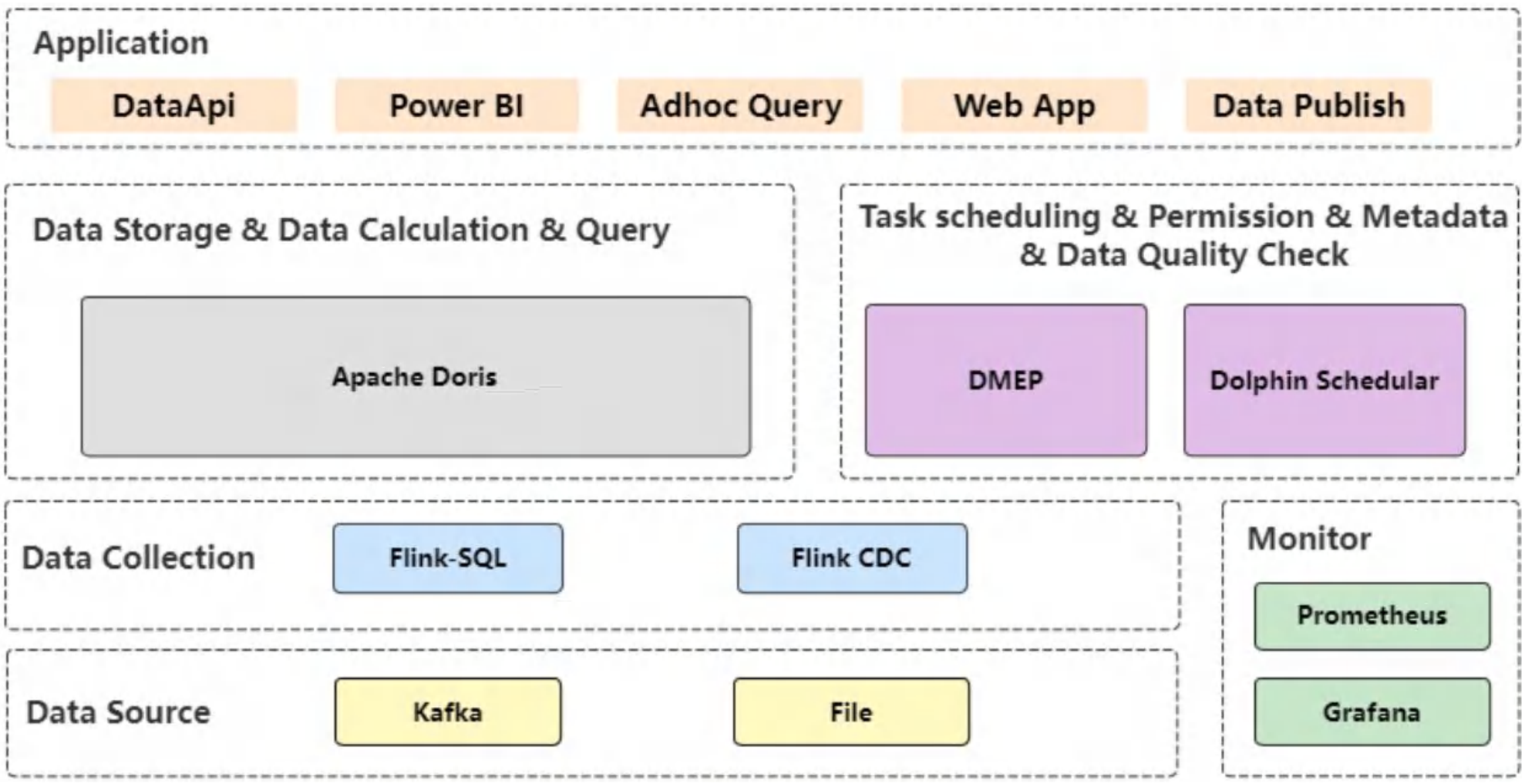
早期数仓架构

早期的公司数仓架构使用 Kudu、Impala 来作为运算存储引擎，整体架构如下图所示。



从上图可知，数据通过 Kafka Consumer 进入 ODS 层；通过 Kudu 层满足数据更新需要；运用 Impala 来执行数据运算和查询；通过自研平台 DMEP 进行任务调度。在 ETL 代码中会使用大量的 Upsert 对数据进行 Merge 操作，那么引入 Doris 的首要问题就是要如何实现 Merge 操作，支持业务数据更新，下文中将进行介绍。

新数仓架构



如上图所示，在新架构设计中使用 **Apache Doris 负责数仓存储及数据运算**；实时数据、 ODS 数据的同步从 Kafka Consumter 改为 Flink ； 流计算平台使用团队自研的 Duckula；任务调度则引入最新 的 DolphinSchedular，Dolphin schedule 几乎涵盖了自研 DMEP 的大部分功能，同时可以很方便拓展 ETL 的方式，可调度很多不同的任务。

优化实践

数据模型选择

上文中提到，当客户对系统中的历史数据修改后，报表数据也要进行同步更改，同时，客户有时只更改某一列的数值，这要求我们需要选择合适的 Doris 模型来满足这些需求。我们在测试中发现，通过 Aggregate 聚合模型 + `Replace_if_not_null` 方式进行数据更新时，可以实现单独更新一列，代码如下：

```
drop table test.expamle_tbl2


CREATE TABLE IF NOT EXISTS test.expamle_tbl2
(
    `user_id` LARGEINT NOT NULL COMMENT "用户id",
    `date` DATE NOT NULL COMMENT "数据灌入日期时间",
    `city` VARCHAR(20) COMMENT "用户所在城市",
    `age` SMALLINT COMMENT "用户年龄",
    `sex` TINYINT COMMENT "用户性别",
    `last_visit_date` DATETIME REPLACE_IF_NOT_NULL COMMENT "用户最后一次访问时间",
    `cost` BIGINT REPLACE_IF_NOT_NULL COMMENT "用户总消费",
    `max_dwell_time` INT REPLACE_IF_NOT_NULL COMMENT "用户最大停留时间",
    `min_dwell_time` INT REPLACE_IF_NOT_NULL COMMENT "用户最小停留时间"
)
AGGREGATE KEY(`user_id`, `date`, `city`, `age`, `sex`)
DISTRIBUTED BY HASH(`user_id`) BUCKETS 1
PROPERTIES (
"replication_allocation" = "tag.location.default: 1"
);



insert into test.expamle_tbl2
values(10000,'2017-10-01','北京',20,0,'017-10-01 06:00:00',20,10,10);
select * from test.expamle_tbl ;

insert into test.expamle_tbl2 (user_id,date,city,age,sex,cost)
values(10000,'2017-10-01','北京',20,0,50);
select * from test.expamle_tbl ;
```

如下图所示，当写 50 进去，可以实现只覆盖 `Cost` 列，其他列保持不变。

```
23
24  insert into test.expamle_tbl2
25  values(10000,'2017-10-01','北京',20,0,'2017-10-01 06:00:00',20,10,10);
26  select * from test.expamle_tbl2 ;
27
```

信息 | Result 1 | 状态

user_id	date	city	age	sex	last_visit_date	cost	max_dwell_time	min_dwell_time
10000	2017-10-01	北京	20	0	2017-10-01 06:00:00	20	10	10

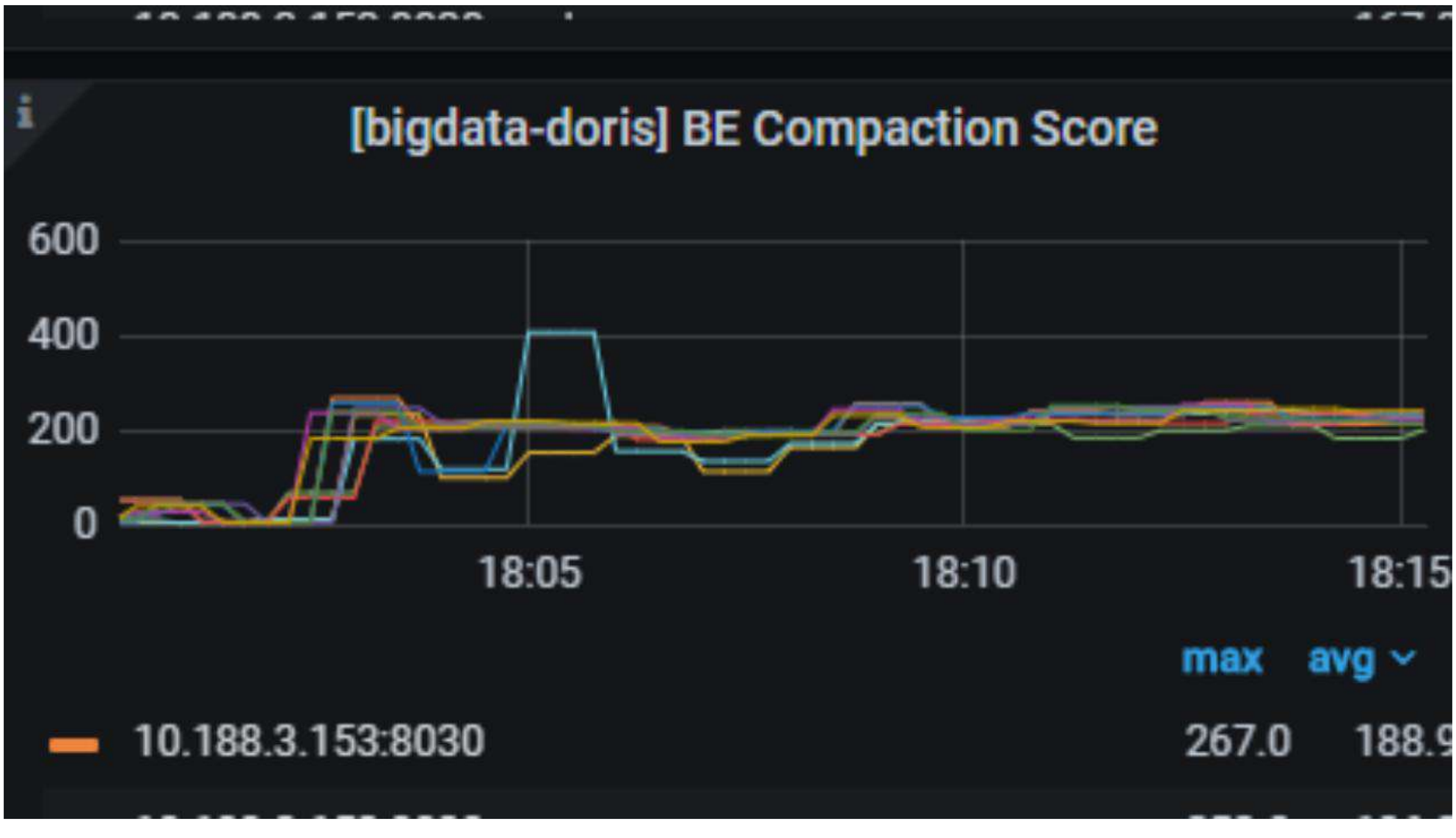
```
27
28  insert into test.expamle_tbl2 (user_id,date,city,age,sex,cost)
29  values(10000,'2017-10-01','北京',20,0,50);
30  select * from test.expamle_tbl2 ;
31
```

信息 | Result 1 | 状态

user_id	date	city	age	sex	last_visit_date	cost	max_dwell_time	min_dwell_time
10000	2017-10-01	北京	20	0	2017-10-01 06:00:00	50	10	10

Doris Compaction 优化

当 Flink 抽取业务库全量数据、持续不断高频写入 Doris 时，将产生了大量数据版本，**Doris 的 Compaction 合并版本速度跟不上新版本生成速度，从而造成数据版本堆积**。从下图可看出，BE Compaction Score 分数很高，最高可以达到 400，而健康状态分数应在 100 以下。

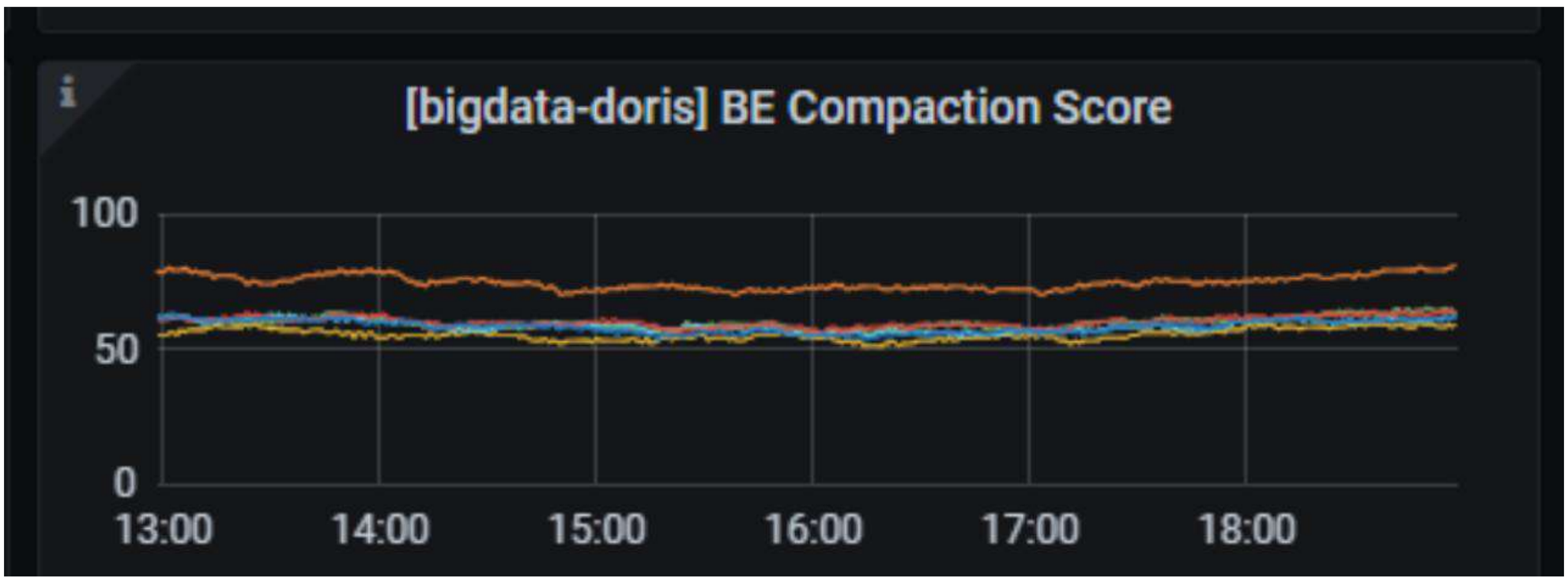


针对以上问题我们做了以下调整：

- 全量数据不使用实时写入的方式，先导出到 CSV，再通过 Stream Load 写入 Doris；

- 降低 Flink 写入频率，增大 Flink 单一批次数据量；该调整会降低数据的实时性，需与业务侧进行沟通，根据业务方对实时性的要求调整相关数值，最大程度的降低写入压力。
- 调节 Doris BE 参数，使更多 CPU 资源参与到 Compaction 操作中；
  - `compaction_task_num_per_disk` 单磁盘 Compaction 任务线程数默认值 2，提升后会大量占用CPU资源，阿里云 16 核，提升 1 个线程多占用 6% 左右 CPU。
  - `max_compaction_threads compaction` 线程总数默认为 10。
  - `max_cumulative_compaction_num_singleton_deltas` 参数控制一个 CC 任务最多合并 1000 个数据版本，适当改小后单个 Compaction 任务的执行时间变短，执行频率变高，集群整体版本数会更加稳定。

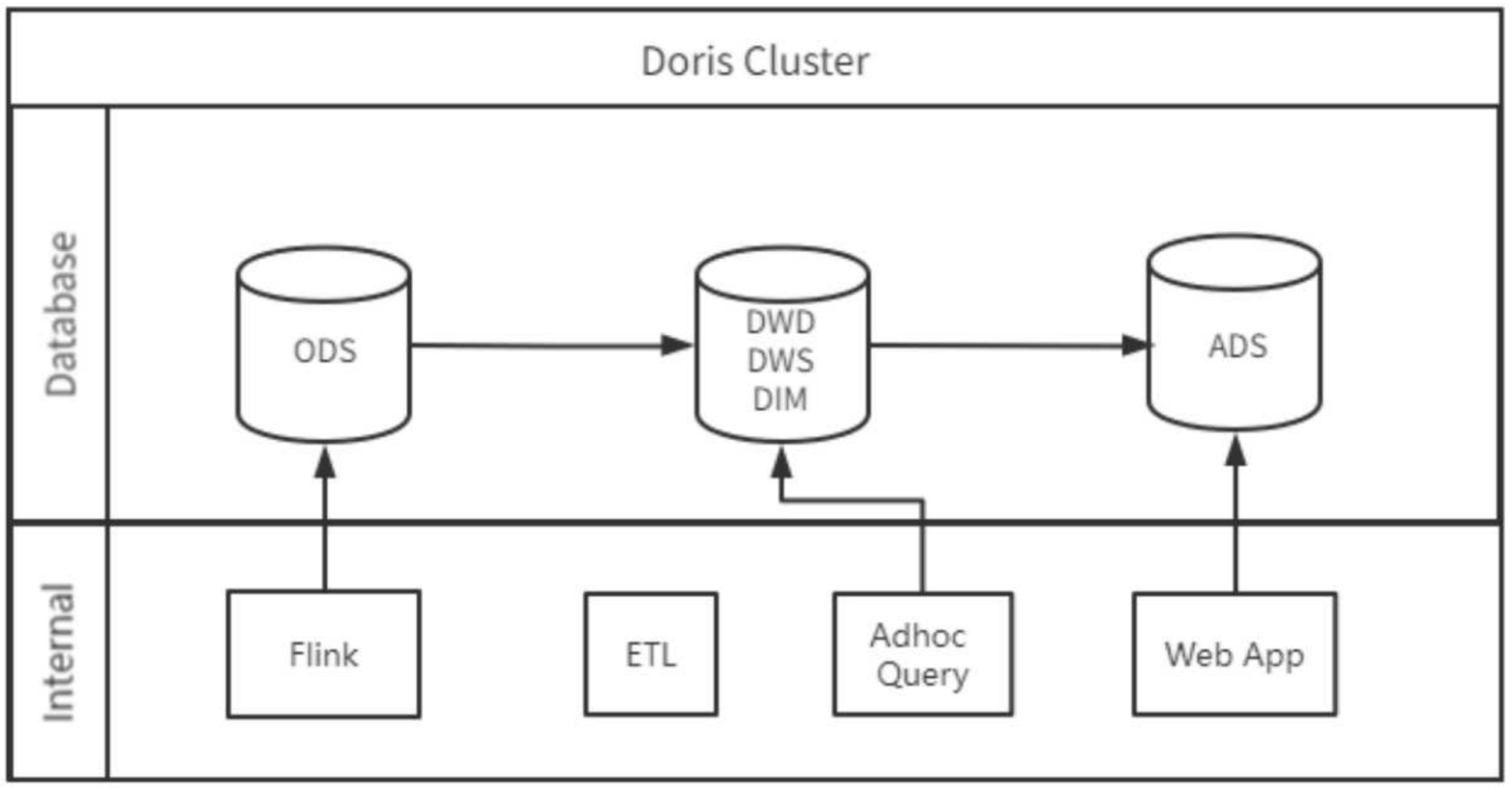
通过调整集群， Compaction Score 稳定在了 50–100，有效解决了版本堆积问题。



值得关注的是，在 **Doris 1.1 版本中对 Compaction 进行了一系列的优化**，在任务调度层面，增加主动触发式的 Compaction 任务调度，结合原有的被动式扫描，高效的感知数据版本的变化，主动触发 Compaction。在任务执行层面，对资源进行隔离，将重量级的 Base Compaction 和轻量级的 Cumulative Compaction 进行了物理分离，防止任务的互相影响。同时，针对高频小文件的导入，优化文件合并策略，采用梯度合并的方式，保证单次合并的文件都属于同一数量级，逐渐有层次的进行数据合并，减少单个文件参与合并的次数，大幅节省系统消耗。

负载隔离

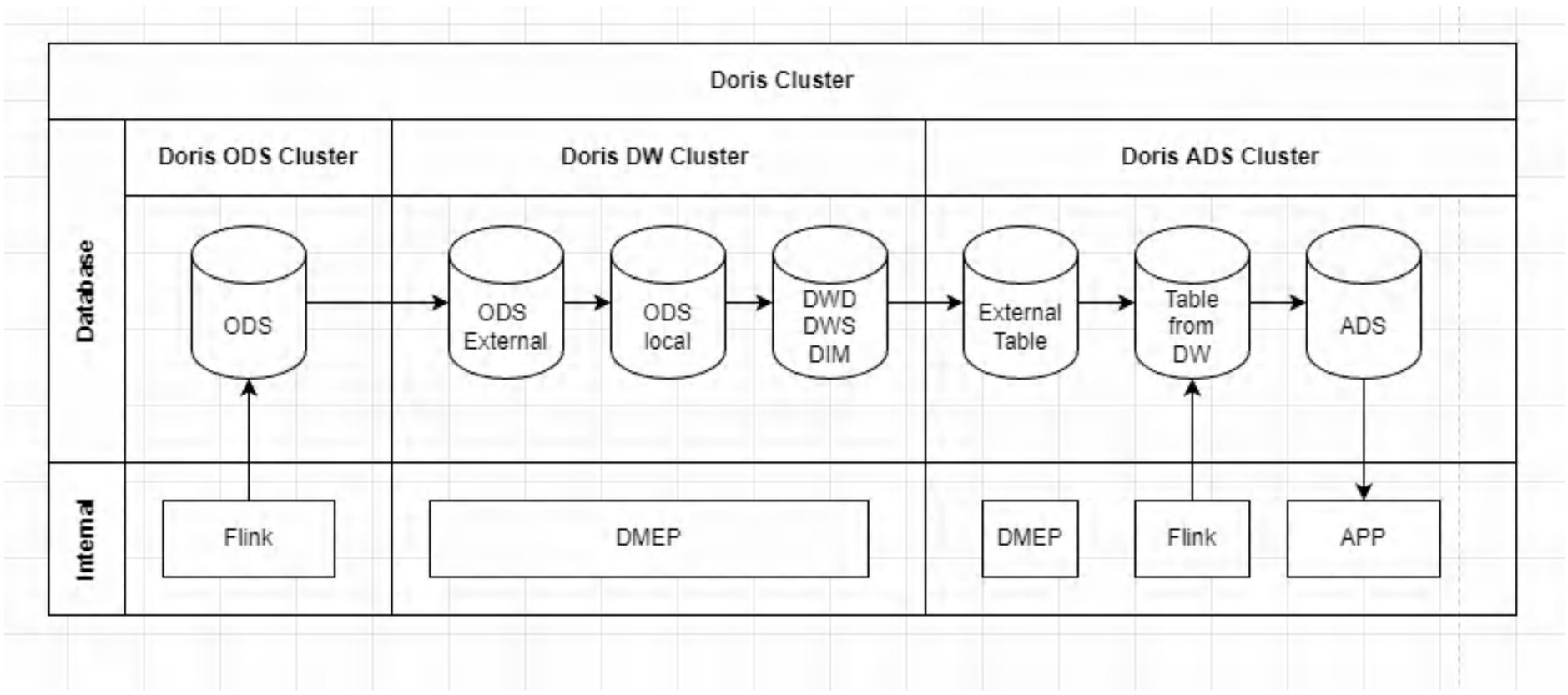
最初我们只有 1 个 Doris 集群，Doris 集群要同时支持高频实时写、 高并发查询、ETL 处理以及 Adhoc 查询等功能。其中高频实时写对 CPU 的占用很高，而 CPU 的上限决定高并发查询的能力，另外 Adhoc 查询无法预知 SQL 的复杂度，当复杂度过高时也会占用较高的内存资源，**这就导致了资源竞争，业务之间互相影响的问题。**为解决这些问题，我们进行了以下探索优化。



1. Doris 集群拆分

最初我们尝试对 Doris 集群进行拆分，我们把 1 个集群拆分为 3 个集群，分别为 ODS 集群、DW 集群、ADS 集群。我们将 CPU 负载最高的 ODS 层分离出去， ETL 时，通过 Doris 外表连接另一个 Doris 集群抽取数据；同时也将 BI 应用访问的集群分离出去，独立为业务提供数据查询。如下所示为各集群负责的任务：

- ODS 集群：数仓 ODS 层，Flink 写数据集中在此层进行。
- DW 集群：数仓 DW 层，DIM 层，主要负责 ETL 处理，Adhoc 查询任务。
- ADS 集群：数仓 ADS 层，主要支持 Web 应用的数据查询

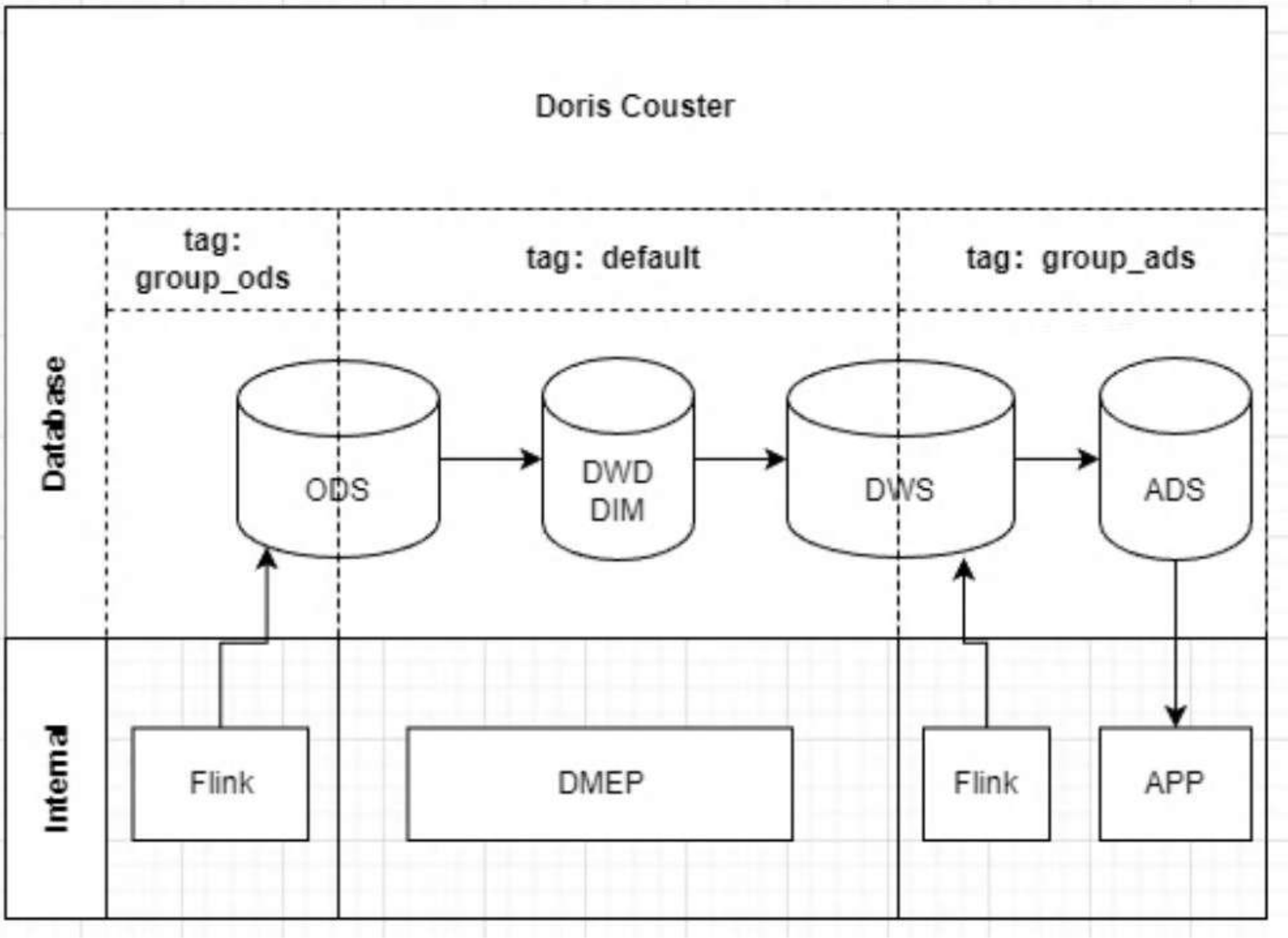


通过集群拆分，有效降低各个资源间的相互影响，保证每个业务运转都有较充足的资源。但是集群的拆分也存在集群之间数据同步 ETL 时间较长、从 ADS 到 ODS 跨 3 个集群的数据校验复杂度较高等问题。直到 Doris 0.15 发布后，这些问题也得到了相对有效的解决。

2. 资源隔离优化集群资源

Doris 0.15 版本新增了资源标签功能以及查询 Block 功能，资源标签功能允许 Doris 集群实现资源隔离，该功能有效减少集群之间同步数据的时间，降低了跨集群数据校验复杂度。其次查询 Block 功能的上线，可以对 SQL 进行查询审计，阻塞简单/不合规的查询语句，降低资源占用率，提升查询性能。除此之外，通过资源隔离的方式，我们将 3 个集群合并成 1 个集群，被合并的 6 个入口节点 FE 被释放掉，将节省的资源加到核心的运算节点上来。

升级到 Doris 0.15 后，我们将 ODS 表的副本修改为 `group_ods` 3份， `default` 3 份。Flink 写入时只写 `group_ods` 资源组的节点，数据写入后，得益于 Doris 内部的副本同步机制，数据会自动实时同步到 `default` 资源组。ETL 则可以使用 `default` 资源组的节点资源取用 ODS 数据，进行查询和数据处理。同理 ADS 也做了相同处理，原先需要通过外表进行数据抽取同步的表，均被做成了副本跨资源组的形式。此方式有效缩短了跨集群数据同步的 ETL 时长 。



离线 ETL 内存高

我们使用的是离线 ETL 方式直接在 Doris 上做 ETL 操作，在 Join 时，如果右表数据量比较大的情况下会消耗大量的内存，从而造成 OOM。在 1.0 版本之前内存跟踪能力较弱，容易造成 BE 节点超出 Linux 限制，导致进程被关闭 ，这时候会收到以下报错信息： `Host is down` 或者 `Fail to initialize storage reader` 。在 1.0 及更高版本中， Doris 由于优化了内存跟踪，则容易见到以下报错： `Memory exceed limit. Used: XXXX ,Limit XXXX.`

针对内存受限问题，我们开始寻找优化方案，另外由于公司内部资源受限，优化方案必须在不增加集群成本的情况下把超出集群负荷的任务跑通。这里为大家介绍 2 个解决方法：

1. 优化调整 Join 的方式：

Doris 内部 Join 分为 4 种，其内存开销以及优先级如下图所示：

	Colocated	Bucket shuffle	Broadcast	Shuffle
数据传输	无	右表	右表	左表 右表
内存开销	<=右表	<=右表	BE数*右表	右表
优先度	1	2	3	4

从上图可知，Join 类型优先级从左往右依次变低，Shuffle 的优先级最低，排在 Broadcast 之后。值得注意的是， Broadcast 内存开销非常大，它将右表广播到所有 BE 节点，这相当于每个 BE 节点会消耗一个右表的内存，这将造成很大的内存开销。针对 Broadcast 比较大的内存开销，我们通过 Hint 条件强制 Join 类型的方式，使 Join 语句跳过 Broadcast 到 Shuffle Join，从而降低内存消耗。

```
1   select * from a join [shuffle] b on a.k1 = b.k1;
```

2. 数据分批处理

我们尝试将数据按照时间分批，每批涵盖某一个或某几个时间段的数据，分批进行 ETL，有效降低内存消耗，避免 OOM 。

分批须知：需要将分批的标记列放在主键中，最大程度提升搜索数据的效率；注意分桶和分区的设置方式，保证每个分区的数据量都比较均衡，避免个别分区内存占用较高的问题。

总结

新架构收益

- 基于 Doris 的新数仓架构不再依赖 Hadoop 生态组件，运维简单，维护成本低。
- 具有更高性能，使用更少的服务器资源，提供更强的数据处理能力。
- 支持高并发，能直接支持 WebApp 的查询服务。
- 支持外表，可以很方便的进行数据发布，将数据推送其他数据库中。
- 支持动态扩容，数据自动平衡。
- 支持多种联邦查询方式，支持 Hive、ES、MySQL 等

得益于新架构的优异能力，我们所用集群从 **18 台** 16C128G 减少到 **12 台** 16C128G，集群资源较之前节**省了34%**，大大降低了投入成本；并且运算性能得到大幅提升，在 Kudu 上 **3 小时**即可完成的 ETL 任务， Doris 只需要 **1 小时**即可完成 。除此之外，高频更新的场景下，Kudu 内部数据碎片文件不能进行自动合并，表的性能会越来越差，需要定期重建；而 Doris 内部的 Compaction 机制可以有效避免此问题。

社区寄语

首先，**Doris 的使用成本很低**，仅需要 3 台低配服务器、甚至是台式机，就能相对容易的部署一套基于 Doris 的数仓作为数据中台基础；我认为对于想要进行数字化，但介于资源投入有限而又不想落后于市场的企业来说，**非常建议尝试使用 Apache Doris ，Doris 可以助力企业低成本跑通整个数据中台。**

其次，Doris 是一款国人自研的的 MPP 架构分析型数据库，这令我感到很自豪，同时其社区十分活跃、便于沟通，**Doris 背后的商业化公司 SelectDB 为社区组建了一支专职技术团队，任何问题都能在 1 小时内得到响应，近 1 年社区更是在 SelectDB 的持续推动下，推出了一系列十分抗打的新特性。**另外社区在版本迭代时会认真考虑中国人的使用习惯，这些会为我们的使用带来很多便利。

最后，感谢 Doris 社区和 SelectDB 团队的全力支持，也欢迎开发者以及各企业多多了解 Doris、使用 Doris，支持国产数据！

助力传统行业数字化革新，
Apache Doris 在美联物业的数据仓库应用实践

作者：美联物业数仓负责人，谢帮桂

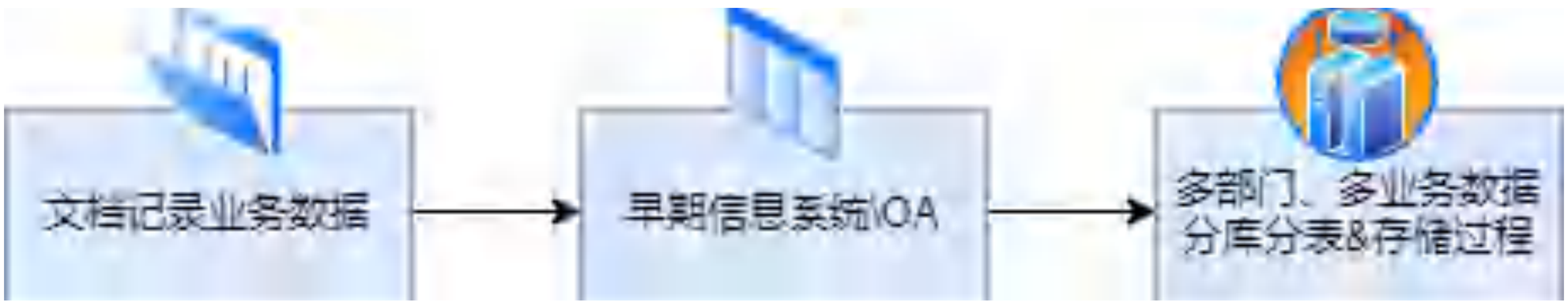
导读：传统行业面对数字化转型往往会遇到很多困难，比如缺乏数据管理体系、数据需求开发流程冗长、烟囱式开发、过于依赖纸质化办公等，美联物业也有遇到类似的问题。本文主要介绍关于美联物业在数据体系方面的建设，以及对数据仓库搭建经验进行的分享和介绍，旨在为数据量不大的传统企业提供一些数仓思路，实现数据驱动业务，低成本、高效的进行数仓改造。

美联物业属于香港美联集团成员，于 1973 年成立，并于 1995 年在香港联合交易所挂牌上市(香港联交所编号: 1200)，2008 年美联工商铺于主板上市（香港联交所编号: 459），成为拥有两家上市公司的地产代理企业。拥有 40 余载房地产销售行业经验，业务涵盖中小型住宅、豪宅及工商铺，提供移民顾问、金融、测量、按揭转介等服务，业务遍布中国香港地区、中国澳门地区和中国内地等多个重要城市。

本文主要介绍关于美联物业在数据体系方面的建设，以及对数据仓库搭建经验进行的分享和介绍，旨在为数据量不大的传统企业提供一些数仓思路，实现数据驱动业务，低成本、高效的进行数仓改造。（因考虑隐私政策，本文不涉及公司任何具体业务数据。）

业务背景

美联物业早在十多年前就已深入各城市开展房地产中介业务，数据体系的建设和发展与大多数传统服务型公司类似，经历过几个阶段时期，如下图所示。



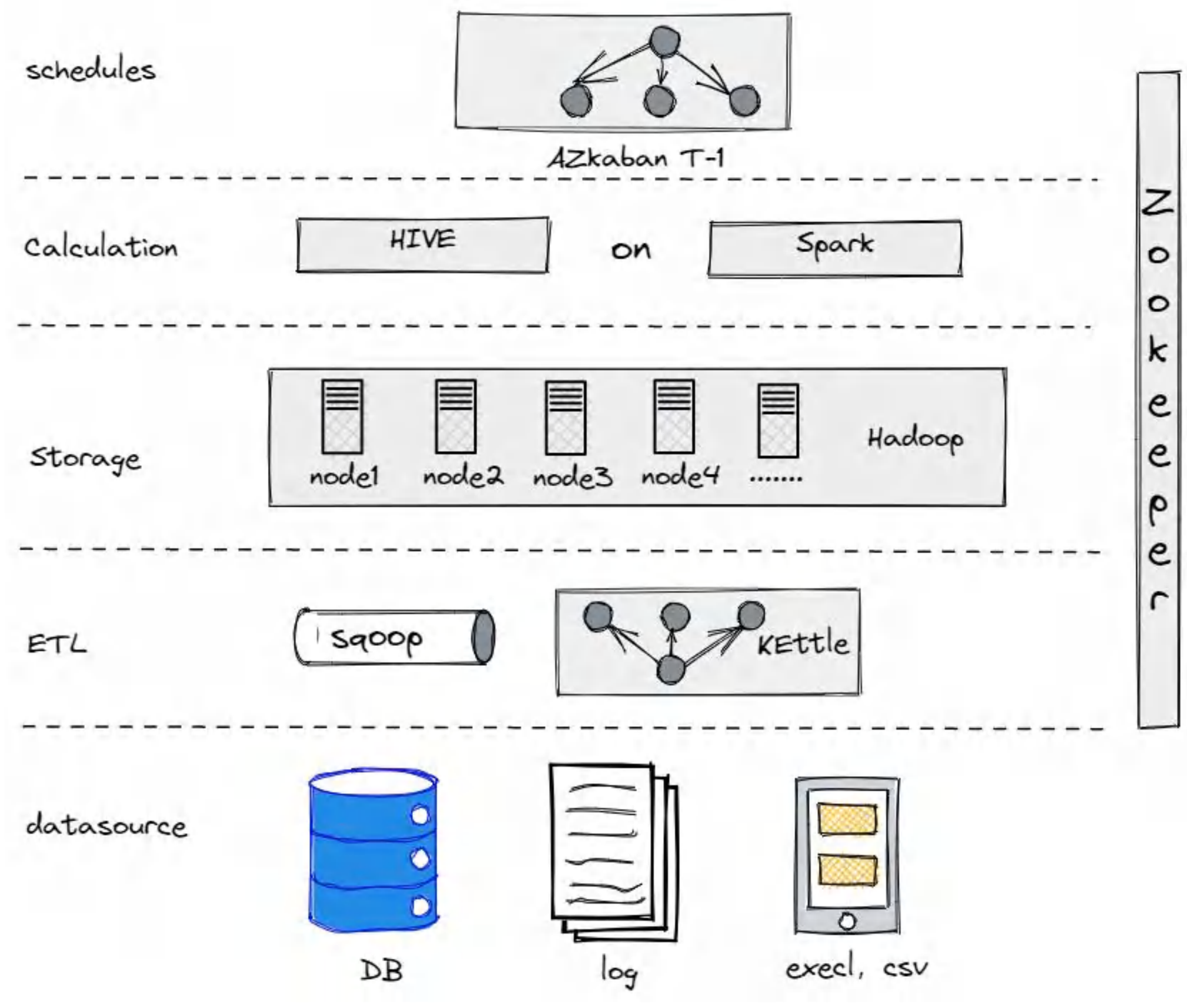
我们的数据来源于大大小小的子业务系统和部门手工报表数据等，存在历史存量数据庞大，数据结构多样复杂，数据质量差等普遍性问题。此外，早期业务逻辑处理多数是使用关系型数据库 SQL Server 的存储过程来实现，当业务流程稍作变更，就需要投入大量精力排查存储过程并进行修改，使用及维护成本都比较高。

基于此背景，我们面临的挑战可以大致归纳为以下几点：

- 缺乏数据管理体系，统计口径统一，已有数据无法降本复用。多部门、多系统、多字段，命名随意、表违反范式结构混乱；对同一业务来源数据无法做到多份报表复用，反复在不同报表编写同一套计算逻辑。
- 海量数据下性能不足，查询响应慢。历史大多数业务数据存储在关系型数据库中，分表分库已无法做到上亿数据秒级分析查询。
- 数据需求开发流程冗长、烟囱式开发。每当业务部门提出一个数据需求，数据开发就需要在多个系统之间进行数据兼容编写存储过程，从而导致存储过程的可移植性和可读性都非常差。
- 部门之间严重依赖文本文档处理工作，效率低下。由于长期的手工统计，用户已形成习惯，导致对信息系统的信任程度也比较低。

早期架构

针对上述的几个需求，我们在平台建设的初期选用了 Hadoop、Hive、Spark 构建最初的离线数仓架构，也是比较普遍、常见的架构，运作原理不进行过多赘述。



我们数据体系主要服务对象以内部员工为主，如房产经纪人、后勤人员、行政人事、计算机部门，房产经纪在全国范围内分布广泛，也是我们的主要服务对象。当前数据体系还无需面向 C 端用户，因此在数据计算和资源方面的压力并不大，早期基于 Hadoop 的架构可以满足一部分基本的需求。但是随着业务的不断发展、内部人员对于数据分析的复杂性、分析的效率也越来越高，该架构的弊端日益越发的明显，主要体现为以下几点：

- 过于笨重：传统公司的计算量和数据量并不大，使用 Hadoop 过于浪费。
- 效率低下：T+1 的调度时效和脚本，动辄需要花费 1 小时的计算时间导入导出，效率低、影响数据的开发工作。
- 维护成本高：组件过多，排查故障链路过长，运维成本也很高，且部门同事之间熟悉各个组件需要大量学习和沟通成本。

新数仓架构

基于上述业务需求及痛点，我们开始了架构升级，并希望在这次升级中实现几个目标：

- 初步建立数据管理体系，搭建数据仓库。
- 搭建报表平台和报表快速开发流程体系。
- 实现数据需求能够快速反应和交付（1小时内），查询延迟不超过 10s。
- 最小成本原则构建架构，支持滚动扩容。

技术选型

经过调研了解以及朋友推荐，我们了解到了 Apache Doris ，并很快与社区取得了联系，Apache Doris 的几大优势吸引了我们：

1. 足够简单

美联物业及大部分传统公司的数据人员除了需要完成数据开发工作之外，还需要兼顾运维和架构规划的工作。因此我们选择数仓组件的第一原则就是"简单"，简单主要包括两个方面：

- 使用简单：Apache Doris 兼容 MySQL 协议，支持标准 SQL，有利于开发效率和共识统一，此外，Doris 的 ETL 编写脚本主要使用 SQL 进行开发，使用 MySQL 协议登陆使用，兼容多数 MySQL 语法，提供丰富的数据分析函数，省去了 UDF 开发工作。
- 架构简单：Doris 的组件架构由 FE + BE 两类进程组成，不依赖其他系统，升级扩容非常方便，故障排查链路非常清晰，有利于运维成本的降低。

2. 极速性能

Doris 依托于列式存储引擎、自动分区分桶、向量计算、多方面 Join 优化和物化视图等功能的实现，可以覆盖众多场景的查询优化，海量数据也能可以保证低延迟查询，实现分钟级或秒级响应。

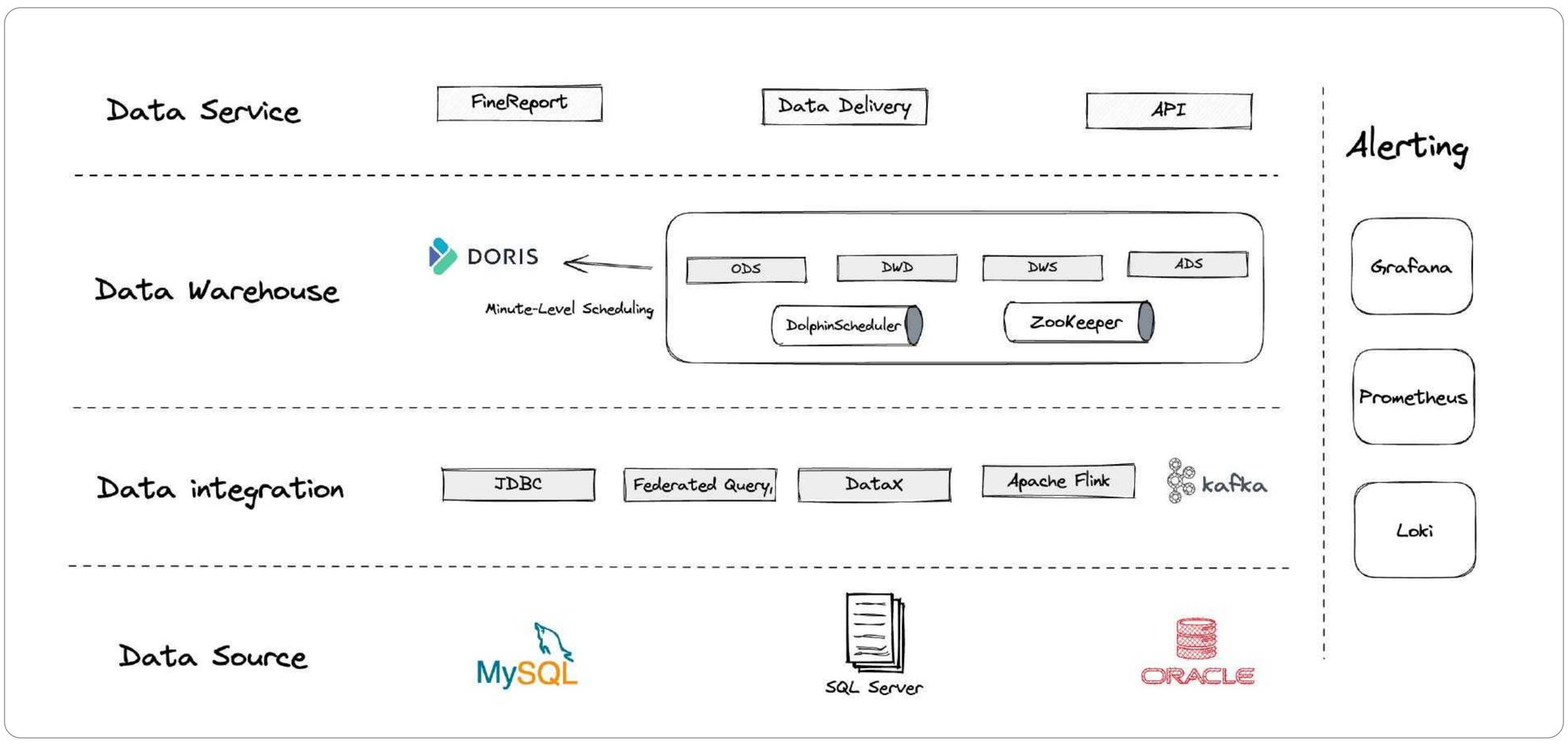
3. 极低成本

降本提效已经成为现如今企业发展的常态，免费的开源软件就比较满足我们的条件，另外基于 Doris 极简的架构、语言的兼容、丰富的生态等，为我们节省了不少的资源和人力的投入。并且 Doris 支持 PB 级别的存储和分析，对于存量历史数据较大、增量数据较少的公司来说，仅用 5-8 个节点就足以支撑上线使用。

4. 社区活跃

截止目前，Apache Doris 已开源数年，并已支持全国超 1500 企业生产使用，其健壮性、稳定性不可否认。另外社区非常活跃，SelectDB 为社区组建了专职的技术支持团队，任何问题均能快速反馈，提供无偿技术支持，使用起来没有后顾之忧。

运行架构



在对 Apache Doris 进一步测试验证之后，我们完全摒弃了之前使用 Hadoop、Hive、Spark 体系建立的数仓，决定基于 Doris 对架构进行重构，以 Apache Doris 作为数仓主体进行开发：

- 数据集成：利用 DataX、Flink CDC 和 Apache Doris 的 Multi Catalog 功能等进行数据集成。
- 数据管理：利用 Apache Dolphinscheduler 进行脚本开发的生命周期管理、多租户人员的权限管理、数据质量监察等。
- 监控告警：采用 Grafana + Prometheus + Loki 进行监控告警，Doris 的各项监控指标可以在上面运行，解决了对组件资源和日志的监控问题。
- 数据服务：使用帆软 Report 为用户提供数据查询和分析服务，帆软支持表单制作和数据填报等功能，支持自助取数和自助分析。

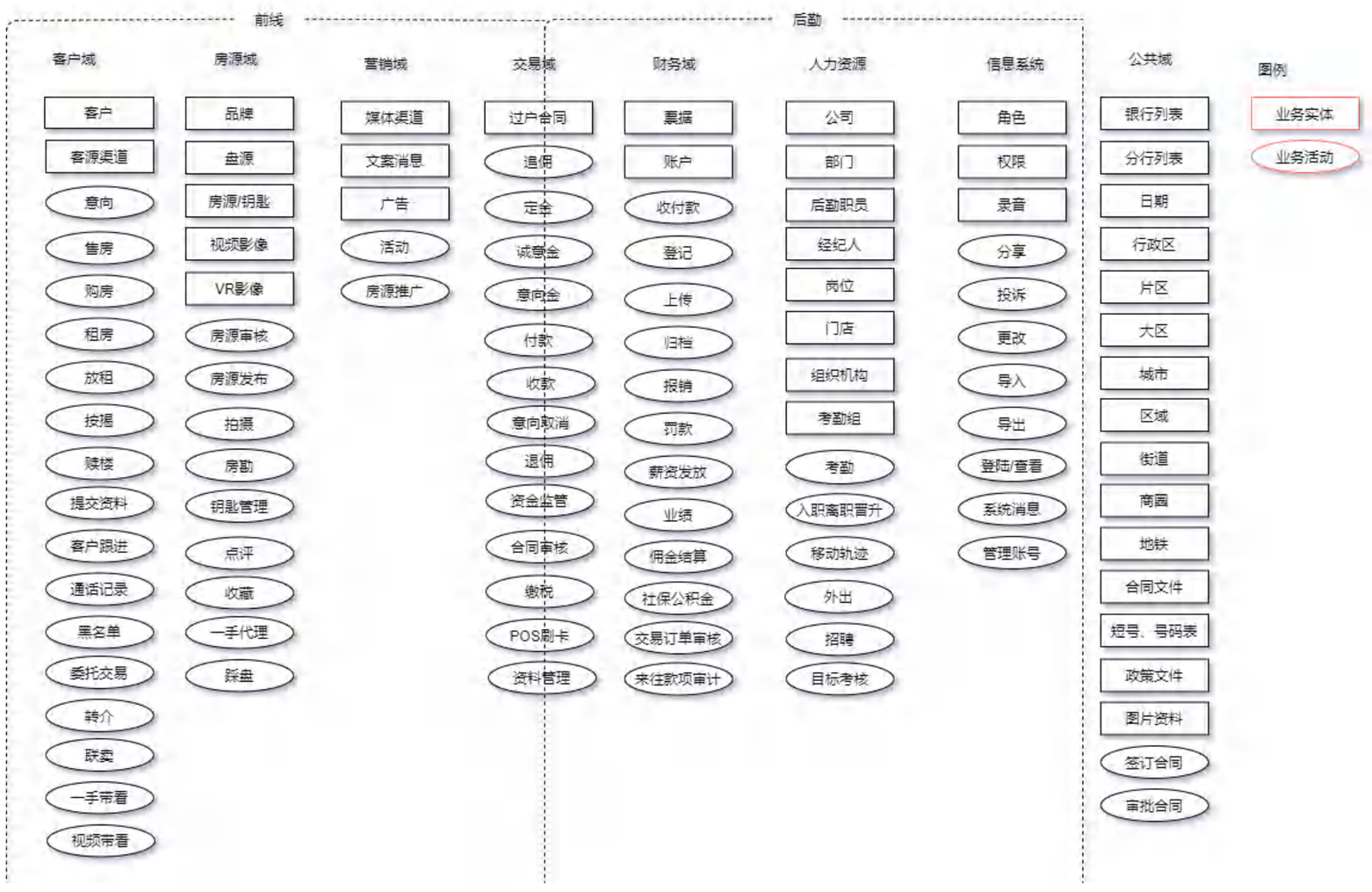
数据模型

1. 纵向分域

房地产中介行业的大数据主题大致如下，一般会根据这些主题进行数仓建模。建模主题域核心围绕"企业用户"、"客户"、"房源"、"组织"等几个业务实体展开，进行维度表和事实表的创建。



我们从前线到后勤，对业务数据总线进行了梳理，旨在整理业务实体和业务活动相关数据，如多个系统之间存在同一个业务实体，应统一为一个字段。梳理业务总线有助于掌握公司整体数据结构，便于维度建模等工作。下图为我们简单的梳理部分房地产中介行业的业务总线：



2. 横向分层

数据分层是最常见的 5 层结构主要是利用 Apache Doris + Apache DolphinScheduler 进行层级数据之间 DAG 脚本调度。

存储策略：我们在 8 点到 24 点之间采用增量策略，0 点到 8 点执行全量策略。采用增量 + 全量的方式是为了在 ODS 表因为记录的历史状态字段变更或者 CDC 出现数据未完全同步的情况下，可以及时进行全量补数修正。

层级	备注	存储策略
ODS	利用 Datax Doris Writer 进行数据导入,要注意参数设置 exec_mem_limit 参数大小，部分全量数据较大超过限制会导致 Stream Load 失败	8 - 24点增量策略、凌晨全量策略一次
DWD	部分事实表涉及缓慢变化维，需要进行拉链存储。	8 - 24点增量策略、凌晨全量策略一次、部分表拉链存储
DWS 与 ADS	注意数据校验问题，加入审计时间，如 etl_time,last_etl_time，create_time 等字段，方便后续故障排查	8 - 24点增量策略、凌晨全量策略一次
DIM	维度表存放层，基于业务实体，如客户，房源，职员，职级，岗位等	每小时整点全量

3. 增量策略

- where >= "业务时间-1天或-1小时"：增量的 SQL 语句不使用 `where ="业务时间当天"` 的原因是为了避免数据漂移情况发生，换言之，调度脚本之间存在时间差，如 23:58:00 执行了脚本，脚本的执行周期是 10 分钟/次，但是源库最后一条数据 23:59:00 才进来，这时候 `where ="业务时间当天"` 就会将该数据漏掉。
- 每次跑增量脚本前获取表中最大的主键 ID 存入辅助表，`where >= "辅助表记录ID"` ：如果 Doris 表使用的是 Unique Key 模型，且恰好为组合主键，当主键组合在源表发生了变化，这时候 `where >= " 业务时间-1天"` 会记录该变化，把主键发生变化的数据 Load 进来，从而造成数据重复。而使用这种自增策略可有效避免该情况发生，且自增策略只适用于源表自带业务自增主键的情况。

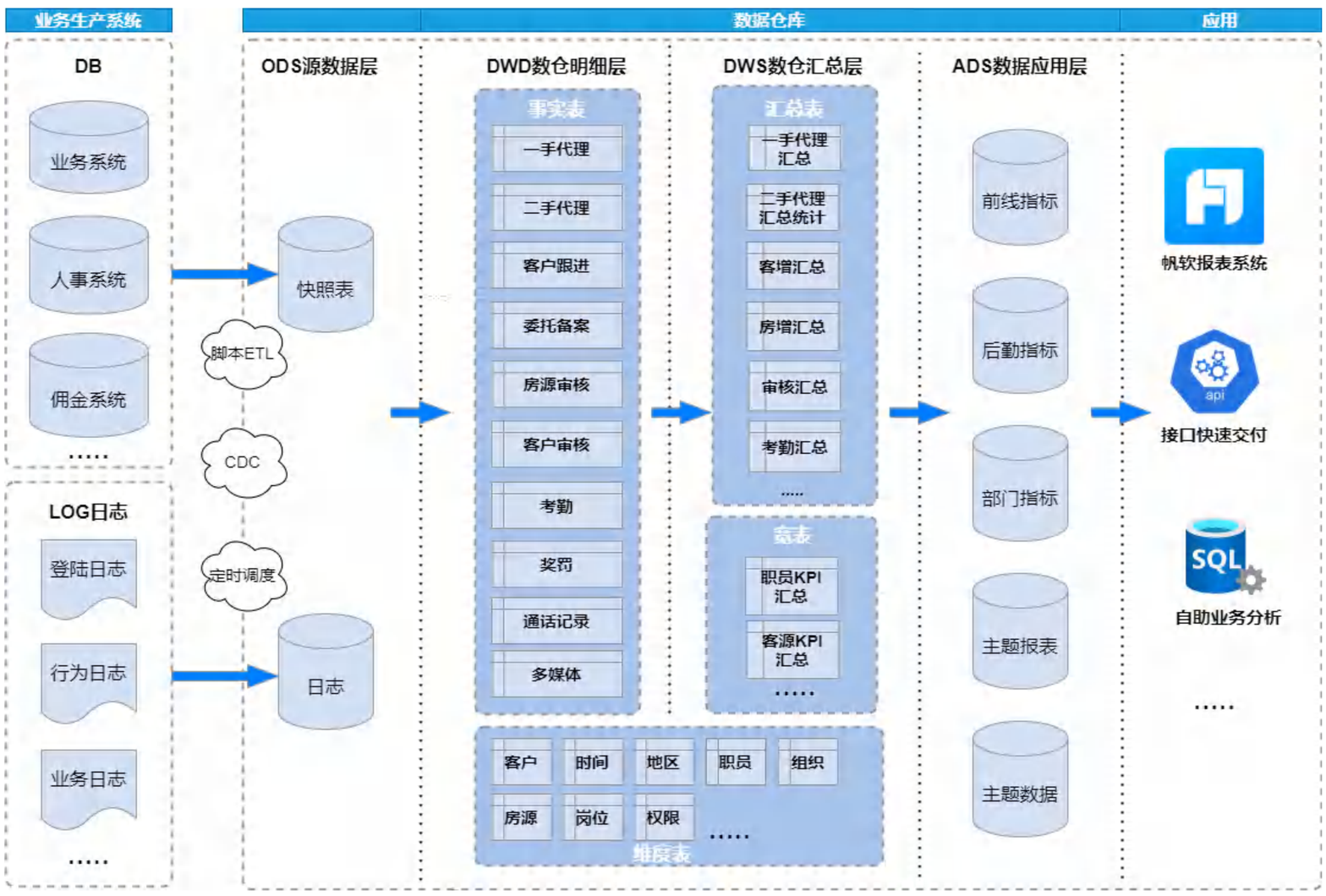
- 表分区：如面对日志表等基于时间的自增数据，且历史数据和状态基本不会变更，数据量非常大，全量或快照计算压力非常大的场景，这种场景需要对 Doris 表进行建表分区，每次增量进行分区替换操作即可，同时需要注意数据漂移情况。

4. 全量策略

1） Truncate Table 清空表插入：先清空表格后再把源表数据全量导入，该方式适用于数据量较小的表格和凌晨没有用户使用系统的场景。

2） `ALTER TABLE tbl1 REPLACE WITH TABLE tbl2` 表替换 ：这种方式是一种原子操作，适合数据量大的全量表。每次执行脚本前先 Create 一张数据结构相同的临时表，把全量数据 Load 到临时表，再执行表替换操作，可以进行无缝衔接。

应用实践



- 业务模型是分钟级调度 ETL
- 初次部署建议配置：8 节点 2FE * 8BE 混合部署
- 节点配置：32C * 60GB * 2TB SSD
- 对于存量数据 TB 级、增量数据 GB 级的场景完全够用，如有需要可以进行滚动扩容。

具体应用

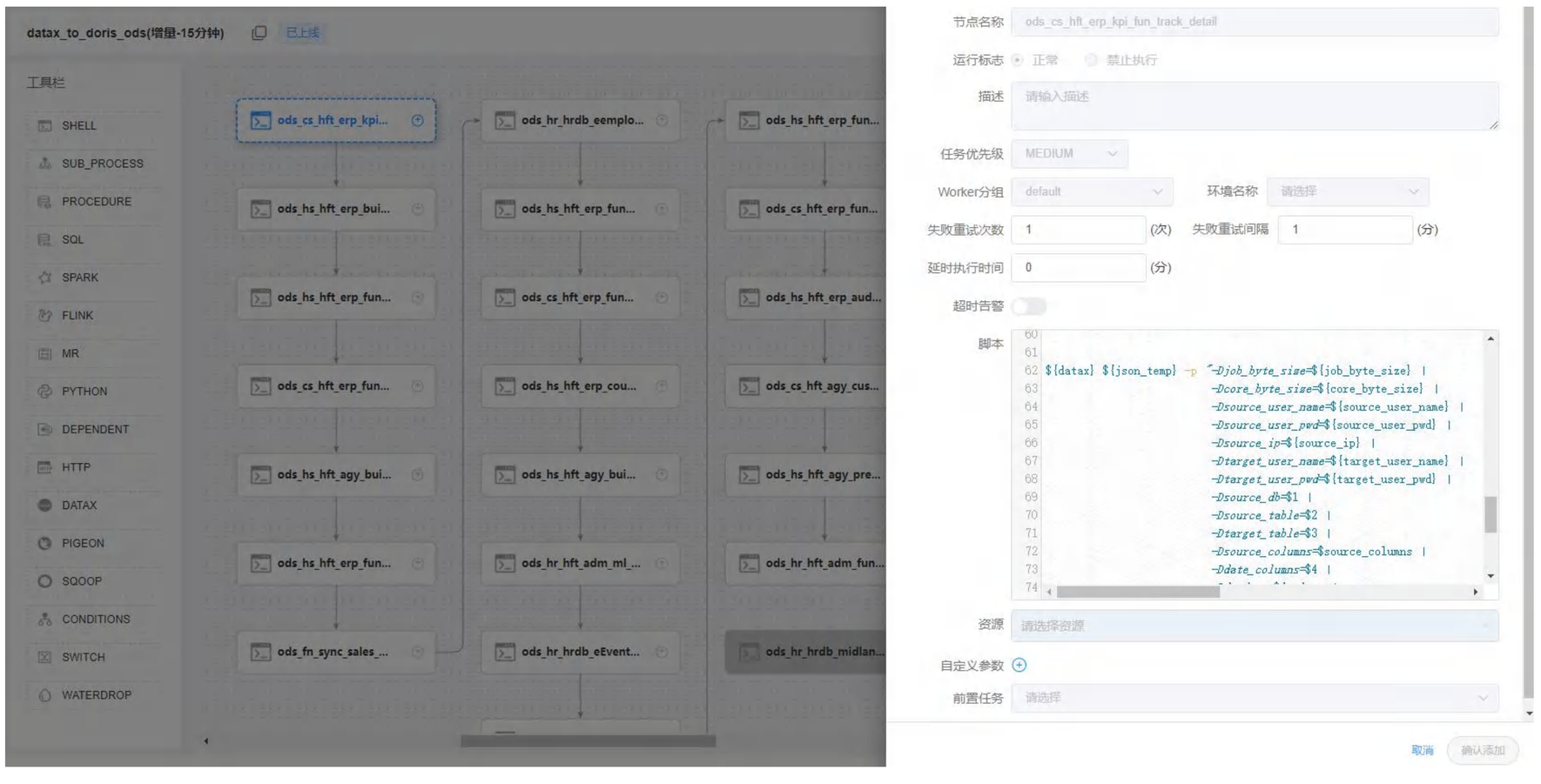
1. 离线数据和日志数据集成利用 DataX 进行增量和全量调度，Datax 支持 CSV 格式和多种关系型数据库的 Redear，而 Doris 在很早之前就提供了 DataX Doris Writer 连接器。

```
"writer": {
    "name": "doriswriter",
    "parameter": {
        "loadUrl": ["172.16.0.13:8030"],
        "loadProps": {
        },
        "column": ["emp_no", "birth_date", "first_name","last_name","gender","hire_date"],
        "username": "root",
        "password": "xxxxxx",
        "postSql": ["select count(1) from all_employees_info"],
        "preSql": [],
        "flushInterval":30000,
        "connection": [
          {
            "jdbcUrl": "jdbc:mysql://172.16.0.13:9030/demo",
            "selectedDatabase": "demo",
            "table": ["all_employees_info"]
          }
        ],
        "loadProps": {
            "format": "json",
            "strip_outer_array": true
        }
    }
}
```

2. 实时统计部分借助了 Flink CDC 对源表进行实时同步，利用 Doris 的物化视图或者 Aggregate 模型表进行实时指标的汇总处理，因我们只有部分指标需要实时处理，不希望产生过多的数据库连接和 Flink Job，因此我们使用 Dinky 的多源合并和整库同步功能，也可以自己简单实现一个 Flink DataStream 多源合并任务，只通过一个 Job 可对多个 CDC 源表进行维护。值得一提的是， Flink CDC 和 Apache Doris 新版本支持 Schema Change 实时同步，在成本允许的前提下，可完全使用 CDC 的方式对 ODS 层进行改造。

```
EXECUTE CDCSOURCE demo_doris WITH (
  'connector' = 'mysql-cdc',
  'hostname' = '127.0.0.1',
  'port' = '3306',
  'username' = 'root',
  'password' = '123456',
  'checkpoint' = '10000',
  'scan.startup.mode' = 'initial',
  'parallelism' = '1',
  'table-name' = 'ods.ods_*,ods.ods_*',
  'sink.connector' = 'doris',
  'sink.fenodes' = '127.0.0.1:8030',
  'sink.username' = 'root',
  'sink.password' = '123456',
  'sink.doris.batch.size' = '1000',
  'sink.sink.max-retries' = '1',
  'sink.sink.batch.interval' = '60000',
  'sink.sink.db' = 'test',
  'sink.sink.properties.format' ='json',
  'sink.sink.properties.read_json_by_line' ='true',
  'sink.table.identifier' = '${schemaName}.${tableName}',
  'sink.sink.label-prefix' = '${schemaName}_${tableName}_1'
);
```

3. 脚本语言采用 Shell + SQL 或纯 SQL 的形式，我们在 Apache DolphinScheduler 上进行脚本生命周期管理和发布，如 ODS 层，可以编写通用的 DataX Job 文件，通过传参的方式将 DataX Job 文件传参执行源表导入，无需在每一个源表编写不同的 DataX Job ，支持统一配置参数和代码内容，维护起来非常方便。另外我们在 DolphinsSheduler 上对 Doris 的 ETL 脚本进行管理，还可以进行版本控制，能有效控制生产环境错误的发生，进行及时回滚。





4. 发布 ETL 脚本后导入数据，可直接在帆软 Report 进行页面制作，基于登陆账号来控制页面权限，如需控制行级别、字段级别权限，可以制作全局字典，利用 SQL 方式进行控制。Doris 完全支持对账号的库表权限控制，这一点和 MySQL 的设置完全一样，使用起来非常便捷。



除以上之外，在容灾恢复、集群监控、数据安全等方面也有应用，比如利用 Doris 备份实现容灾恢复、Grafana + Loki 对集群进行指标规则告警、Supervisor 对节点组件进行守护进程监控，开启 Doris 审计日志对执行 SQL 效率进行监控等，因篇幅限制，此处不进行详细说明。

优化经验

1、数据导入

我们使用 DataX 进行离线数据导入，DataX 采用的是 Stream Load 方式导入，该方式可以通过参数控制导入批次流量，DataX 导入不需要借助计算引擎，开箱即用的特点非常方便。另外，Stream Load 导入是同步返回结果的，其他导入方式一般是异步返回结果，针对我们的架构来说，在 Dolphinscheduler 上执行异步导入数据会误以为该脚本已经执行成功，影响其正常运行。如采用其他异步导入方式，建议在 Shell 脚本中 执行 `show load` 再利用正则过滤状态进行判断。

2、数据模型

我们所有层级的表模型大部分采用 Unique Key 模型，该模型可有效保证数据脚本的结果幂等性，Unique Key 模型可以完美解决上游数据重复的问题，大家可以根据业务模式来选择不同的模型建表。

3、外部数据源读取

Catalog 方式可以使用 JDBC 外表连接，还可以对 Doris 生产集群数据进行读取，便于生产数据直接 Load 进测试服务器进行测试。另外，新版支持多数据源的 Catalog，可以基于 Catalog 对 ODS 层进行改造，无需使用 DataX 对 ODS 层进行导入。

4、查询优化

尽量把非字符类型（如 `int` 类型、 `where` 条件）中最常用的字段放在前排 36 个字节内，在点查表过程中可以快速过滤这些字段（毫秒级别），可以充分利用该特性进行数据表输出。

5、数据字典

利用 Doris 自带的 `information_schema` 元数据制作简单的数据字典，这在还未建立数据治理体系前非常重要，当部门人数较多的时候，沟通成本成为发展过程中最大的“拦路虎”，利用数据字典可快速对表格和字段的全局查找和释义，最低成本形成数仓人员的数据规范，减少人员沟通成本，提高开发效率。

架构收益

引入 Apache Doris 之后，美联物业实现了只用少数服务器、快速搭建一套数据仓库，成功实现降本赋能。在这段时间的运行中，给我们带来的部分收益如下：

- **自动取数导数：**数据仓库的明细表可以定时进行取数、导数，自助组合维度进行分析。
- **效率提升：**T+1 的离线时效降低至分钟级。
- **查询延迟降低：**面对上亿行数据的表，利用 Doris 在索引和点查方面的能力，即席查询延迟平均在到 1 秒内，复杂查询也能 5 秒内响应。
- **运维成本降低：**从数据集成到数据服务，只需维护少数组件就可以实现整个链路的高效管理。
- **数据管理体系初步形成：**Doris 数仓的实现使得数据资产得以规范化的沉淀。
- **存储资源节省：**Doris 超高的压缩比，将数据压缩了 70%，相较于 Hadoop 来说，存储资源的消耗大幅降低。

总结与规划

目前我们已经完成数仓建设的初期目标，未来我们有计划基于 Apache Doris 进行中台化的改造，同时 Apache Doris 在用户画像和人群圈选场景的能力十分强悍，支持 Bitmap 等格式进行导入和转换，提供了丰富的 Bitmap 分析函数等，后续我们也将利用这部分能力进行客户群体分析，加快数字化转型。

最后，感谢 Apache Doris 社区和 SelectDB 团队对美联物业的快速响应和无偿支持，希望 Doris 发展越来越好，也希望更多的企业可以尝试使用 Apache Doris。

查询提速 20 倍，Apache Doris 在 Moka BI SaaS 服务场景下的应用实践

作者：Moka 数据架构师，张宝铭

导读：MOKA 主要有两大业务线 MOKA 招聘（智能化招聘管理系统）和 MOKA People（智能化人力资源管理系统），MOKA BI 通过全方位数据统计和可灵活配置的实时报表，赋能于智能化招聘管理系统和人力资源管理系统。为了提供更完备的数据支持，助力企业提升招聘竞争力，MOKA 引入性能强悍的 Apache Doris 对早期架构进行升级转型，成就了 Moka BI 强大的性能与优秀的用户体验。

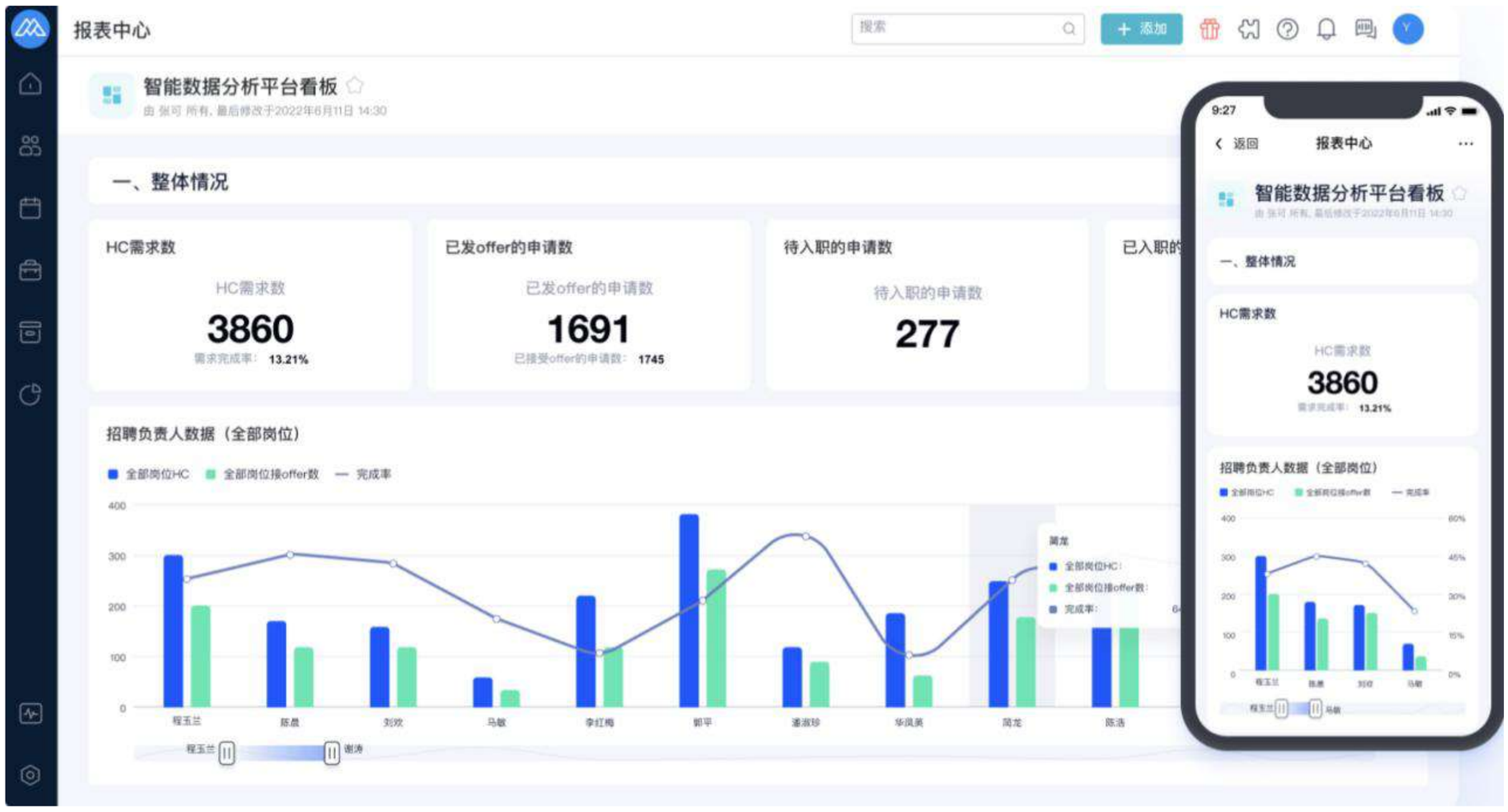
MOKA 是一家致力于为全员带来更好体验的 HR SaaS 服务商， 从 2015 年成立到现在已累计获得数十亿元融资，先后服务小米、万科、太平洋保险等各领域头部企业，累计服务超 2000 家客户。核心产品智能化招聘管理系统和人力资源管理系统为中大型企业提供从招聘到入职、组织人事、假勤、薪酬、绩效等人力资源全模块管理，为企业提供极致体验、数据驱动的智能化 HR SaaS 产品，致力于通过一流技术和服务赋能企业人才战略。

业务需求

MOKA 主要有两大业务线 MOKA 招聘（智能化招聘管理系统）和 MOKA People（智能化人力资源管理系统）。

- MOKA 招聘系统覆盖社招、校招、内推、猎头管理等场景，让 HR 获得更高效的招聘体验，更便捷的协作体验，让管理者获得招聘数据洞见，让招聘降本增效的同时，树立企业在候选人心目中的专业形象。
- MOKA People 覆盖企业所需要的组织人事、假期考勤、薪酬、绩效、审批等高频业务场景，打通从招聘到人力资源管理的全流程，为 HR 工作提效赋能。通过多维度数据洞见，助力管理者高效科学决策。全生态对接，更加注重全员体验，是一款工作体验更愉悦的人力资源管理系统。

而 MOKA BI 通过全方位数据统计和可灵活配置的实时报表，赋能于智能化招聘管理系统和人力资源管理系统。通过 PC 端和移动端的多样化报表展示，为企业改善招聘业务提供数据支持，全面提升招聘竞争力，从而助力科学决策。



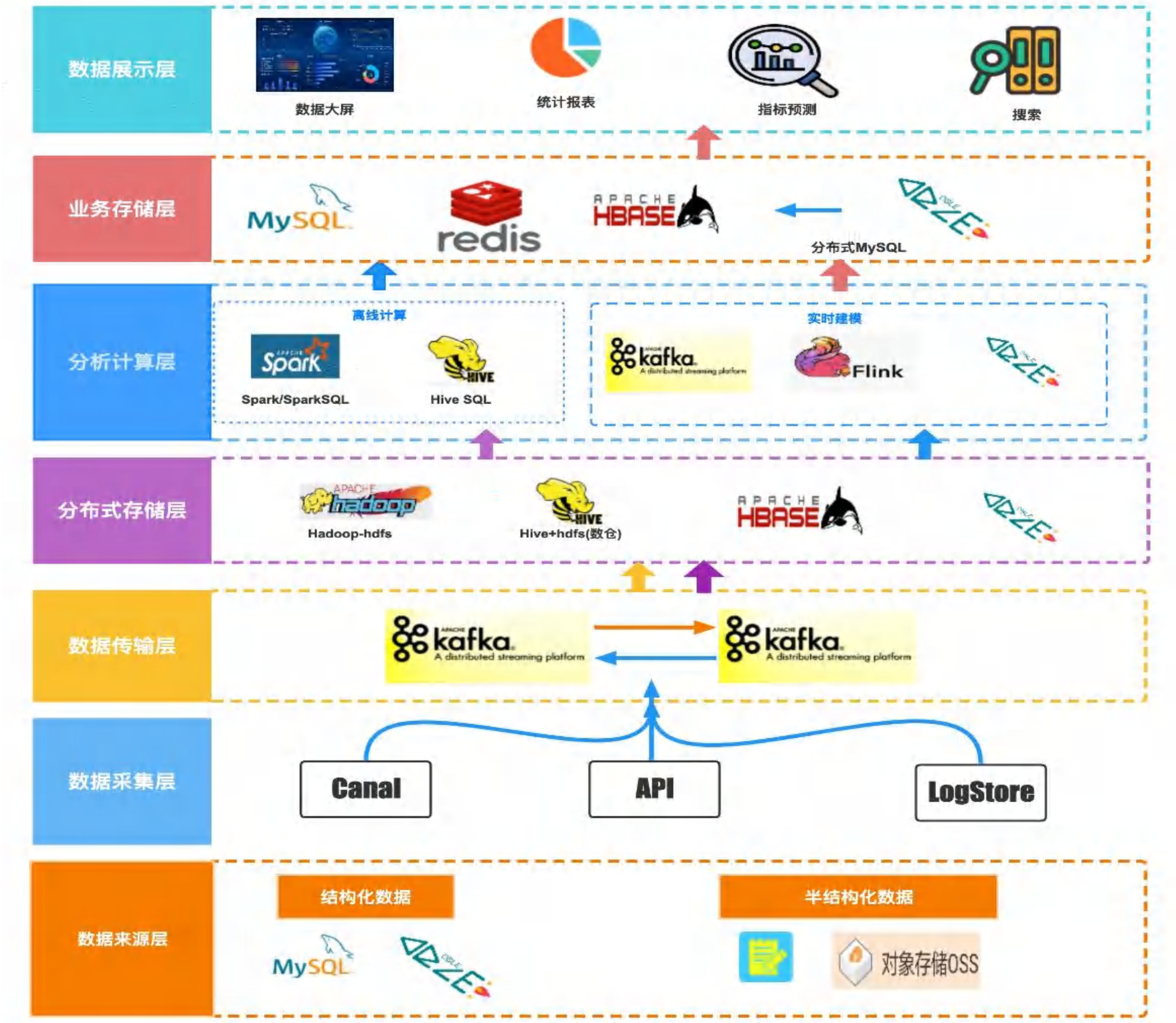
Moka BI 早期架构

架构设计

Moka BI 数仓早期架构是类 Lambda 架构，实时处理和离线处理并存。（架构图见下一页）

- 实时部分数据主要来源为结构化的数据，Canal 采集 MySQL 或 DBLE（基于 MySQL 的分布式中间件）的 Binlog 输出至 Kafka 中；未建模的数据按照公司分库，存储在业务 DBLE 中，通过 Flink 进行实时建模，将计算后的数据实时写入业务 DBLE 库，通过 DBLE 提供报表查询能力，支持数据大屏和实时报表统计。

- 离线部分涵盖了实时部分数据，其结构化数据来源于 DBLE 的 Binlog，明细数据在 Hbase 中实时更新，并映射成 Hive 表，非结构化数据通过 ETL 流程，存储至 Hive 中，通过 Spark 进行进行离线部分建模计算，离线数仓 ADS 层数据输出至 MySQL 和 Redis 支持离线报表统计，明细数据又为指标预测和搜索等外部应用提供数据支持。



现状与问题

在早期数仓架构中，为了实现实时建模以及实时报表查询功能，就必须要求底层数据库能够承载业务数据的频繁插入、更新及删除操作，并要求支持标准 SQL，因此当时我们选择 DBLE 作为数据存储、建模、查询的底层库。早期 Moka BI 灰度期用户较少，业务数据量以及报表的使用量都比较低，DBLE 尚能满足业务需求，但随着 Moka BI 逐渐面向所有用户开放，DBLE 逐渐无法适应 BI 报表的查询分析性能要求，同时实时与离线架构分离、存储成本高且数据不易维护，亟需进行升级转型。

技术选型

为匹配业务飞速增长的要求、满足更复杂的查询需求，我们决定引入一款性能突出的 OLAP 引擎对 Moka BI 进行升级改造。同时出于多样化分析场景的考虑，我们希望其能够支撑更广泛的应用场景。调研的主要方向包括 报表的实时查询能力、数据的更新能力、标准的查询 SQL 以及数据库的可维护性、扩展性、稳定性等。

OLAP引擎	数据导入延迟	实时数据 导入语义	数据实时 更新和删除	数据查询延迟	支持多维分析	支持 MySQL 协议	JOIN支持度
Greenplum	支持实时	at-least-once	支持	低 (亚秒~秒)	支持	否	一般
ClickHouse	支持实时	at-least-once	不支持	低 (亚秒~秒)	支持	否	一般
Doris	支持实时	exactly-once	支持	低 (亚秒~秒)	支持	是	非常好

确定调研方向后，我们首先对 Greenplum 展开了调研，其特点主要是数据加载速度、批量 DML 处理快，但受限于主从双层架构设计、存在性能瓶颈，且并发能力很有限、性能随着并发量增加而快速下降，同时其使用的是 PG 语法、不支持 MySQL 语法，在进行引擎切换时成本较高，因此在基本功能调研结束后便不再考虑使用。

随后我们对 ClickHouse 进行了调研，ClickHouse 在单表查询场景下性能表现非常优异的，但是在多表 Join 场景中性能表现不尽如人意，另外 ClickHouse 缺少数据实时更新和删除的能力，仅适用于批量删除或修改数据，同时 ClickHouse 对 SQL 的支持也比较有限，使用起来需要一定的学习成本。

紧接着我们对近几年势如破竹的 Apache Doris 进行了调研，在调研中发现，Doris 支持实时导入，同时也支持数据的实时更新与删除，可以实现 Exactly-Once 语义；其次，在实时查询方面，Doris 可以实现秒级查询，且在多表 Join 能力的支持上更加强劲；除此之外，Doris 简单易用，部署只需两个进程，不依赖其他系统，兼容 MySQL 协议，使用标准 SQL ，可快速上手，部署及学习成本投入均比较低。

Benchmark

在初步调研的基础之上，我们进一步将 Apache Doris 、Clickhouse 与当下使用的 DBLE 在查询性能上进行了多轮测试对比，查询耗时如下：

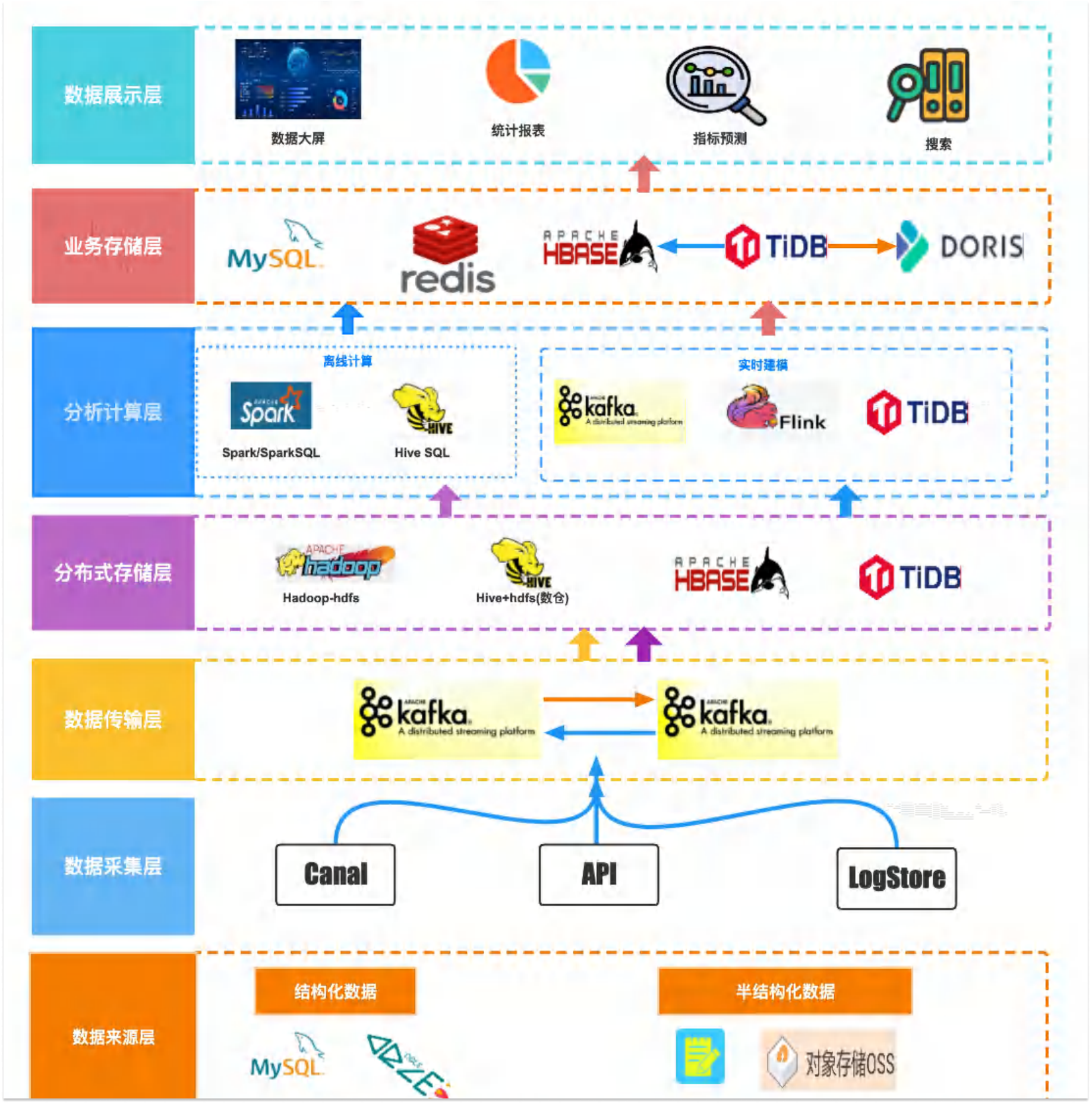
SQL Case/单位：秒	Doris	ClickHouse	DBLE
2个表join	0.63	2.708	27.31
4个表join	1.85	1.926	9.75
6个表join	2.38	11.72	23.25
9个表join	3.2	17.821	32.72
线上慢查询-SQL1	0.62	8.827	9.59
线上慢查询-SQL2	0.6	1.283	8.2
线上慢查询-SQL3	0.91	0.999	6.04
线上慢查询-SQL4	0.4	0.272	7.91
线上慢查询-SQL5	0.62	8.298	3.65
线上慢查询-SQL6	1.55	4.727	62.08

- **多表 Join：**随着 SQL Join 数量的增多，Doris 和 ClickHouse 性能表现差距越来越大，Doris 的查询延迟相对比较稳定，最长耗时仅为 3.2s；而 ClickHouse 的查询延迟呈现指数增长，最长耗时甚至达到 17.8s，**二者性能最高相差 5 倍**，DBLE 的查询性能则远不如这两款产品。
- **慢查询：**在线上慢查询 SQL 的对比测试中，Doris 的性能同样非常稳定，不同的 SQL 查询基本都能在 1s 内返回查询结果，ClickHouse 与之对比查询延迟波动较大、性能表现很不稳定，**二者相同 SQL 性能差距最大超过 10 余倍**。

通过以上调研对比，可以看出 Apache Doris 不管是在基本功能上、还是查询性能上表现都更胜一筹，因此我们将目标锁定了 Doris，并决定尽快引入 Apache Doris 作为 Moka BI 新一代数仓架构的查询引擎。

Moka BI 新版架构

在引入 Doris 之后，Moka BI 数仓架构的主要变化是将 OLAP 和 OLTP 进行分离，即使用 DBLE 支持数据的实时建模，数据来源于 Moka 系统的业务数据，包含了结构化和半结构化的数据，通过 Flink 读取 DBLE Binlog，完成数据去重、合并后写入 Kafka，Doris 通过 Routine Load 读取 Kafka 完成数据写入，此时 DBLE 仅作为数据建模合成使用，由 Doris 提供报表查询能力。(架构图见下一页)



基于 Doris 列存储、高并发、高性能等特性，Moka BI 报表采用自助方式构建完成，支撑客户根据需求灵活配置行、列、筛选的场景。与传统报表按需求定制开发方式对比，这种自助式报表构建非常灵活，平台开发与需求开发完全独立，需求完成速度得到极大的提升。

- **数据导入方面**，数据通过 Routine Load 定期批量导入到 Doris 数据仓库中，保证了数据的准实时同步。通过对系统数据收集与建模，及时向客户提供最新的业务数据，以帮助客户快速了解招聘情况，并做出有效的调整。

- **数据更新方面**，Doris 在大数据量（单表几十亿）的场景下，表现出了突出的数据更新和删除能力，Moka BI 读取的是业务库的 Binlog 数据，其中有大量的更新以及删除操作，Doris 可以通过 Routine Load 的 Delete 配置实现实时删除，根据 Key 实现幂等性写入，配合 Flink 可以做到真正的 Exactly-Once。在架构中增加了 Routine Load 后，数仓可以实现 1 分钟级别的准实时，同时结合 Routine load + Kafka 可以实现流量的削峰，保证集群稳定，并且可以通过重置 Kafka 偏移量来实现间数据重写，通过 Kafka 实现多点消费等。

- **数据查询方面**，充分利用 Doris 的多表 Join 能力，使得系统能够实现实时查询。我们将不同的数据表按照关联字段进行连接，形成一个完整的数据集，基于数据集可进行各种数据分析和可视化操作，同时可高效应对任意条件组合的查询场景以及需要灵活定制需求的查询分析场景，**在某些报表中，需要 Join 的表可能达到几十张，Doris 强大的 Join 性能，使 Moka BI 的报表查询可以达到秒级响应。**

- **运维管理方面**，Doris 部署运维简单方便，不依赖第三方组件，无损弹性扩缩容，自动数据均衡，集群高可用。Doris 集群仅有 FE 和 BE 两个组件，不依赖 Zookeeper 等组件即可实现高可用，部署、运维方便，相比传统的 Hadoop 组件，非常友好，支持弹性扩容，只需简单配置即可实现无损扩容，并且可以自动负载数据到扩容的节点，大大降低了我们引入新技术栈的难度和运维压力。

调优实践

在 Moka BI 报表查询权限场景中，同样配置的报表，有权限认证时查询速度比没有权限认证时慢 30% 左右，甚至出现查询超时，而超管权限查询时则正常，这一现象在数据量较大的客户报表中尤为明显。

人力资源管理业务的数据权限有着极为严格和精细的管控需求，除了 SaaS 业务自身对于不同租户间的数据隔离要求外，还需要针对业务人员的身份角色、管理部门范畴以及被管理人员的信息敏感程度对可见数据的范围进行进一步细分，因此在 Moka BI 权限功能模块的设计之时就考虑并实现了极为灵活的自定义配置化方案。例如 HRBP 与 PayRoll、HRIS 等角色的可见字段不同、不同职级或部门但角色一致用户的可见数据区间不同，同时针对部分敏感的人员信息还需要做数据过滤，或者出于管理授权的需求临时开通某一权限，甚至以上权限要求还会进行多重的交叉组合，以保证每一用户可查看的数据、报表、信息均被限制在权限范围以内。

因此当用户需要对数据报表进行查询时，会先在 Moka BI 的权限管理模块进行多重验证，验证信息会通过 `in` 的方式拼接在查询 SQL 中并传递给 OLAP 系统。随着客户业务体量的增大，对于权限管控的要求越精细、最终所产生的 SQL 就越复杂，部分业务规模比较大的客户报表会出现上千甚至更多的权限限制，因此造成 OLAP 系统的 `id` 过滤时间变长，导致报表查询延迟增加，给大客户造成了体验不佳。

客户报表	无权限验证延迟	有权限验证延迟
报表A	3.2S	5.6S
报表B	1.3S	4.5S
报表C	10S	超时

解决方案：

为适配该业务场景，我们通过查看官方的文档发现 Doris BloomFilter 索引的特性可以很好的解决该问题

```
select * from jobs a join users b on a.user_id=b.id where a.user_id in (1,2,3,...)
```

Doris BloomFilter 索引使用场景：

- BloomFilter 适用于非前缀过滤。
- 查询会根据该列高频过滤，而且查询条件大多是 `in` 和 `=` 过滤。
- 不同于 Bitmap，BloomFilter 适用于高基数列，比如 UserID。因为如果创建在低基数的列上，比如“性别”列，则每个 Block 几乎都会包含所有取值，导致 BloomFilter 索引失去意义。

客户报表	无权限验证延迟	有权限验证延迟
报表A	3.2S	4.3s
报表B	1.3S	3.8s
报表C	10S	15s

经过验证，可以通过上方对比报表看到，**将相关 ID 字段增加 BloomFilter 索引后，权限验证场景查询速度提升约 30% ，有权限验证的报表超时的问题也得到了改善。**

收益与总结

目前 Moka BI Doris 有两个集群，共 40 台服务器，数仓共维护了 400 多张表，其中 50 多张表数据量超过 1 亿，总数据量为 TB 级别。

引入 Apache Doris 改造了新的数据仓库之后，满足了日益增长的分析需求以及对数据实时性的要求，总体收益包含以下几点：

- **高性能数据查询：**Doris 基于列存储技术，能够快速处理大量的数据，并支持高并发的在线查询，解决了关系型数据库无法支持的复杂查询问题，复杂 SQL 查询的速度上升了一个数据量级。
- **数据仓库的可扩展性：**Doris 采用分布式集群架构，可以通过增加节点来线性提升存储和查询瓶颈，打破了关系型数据库数据单点限制问题，查询性能得以显著提升。
- **更广泛的应用：**基于 Doris 构建了统一的数据查询平台，应用不再局限于报表服务，对于离线的查询也有很好的支撑，可以说 Doris 的引入是构建数仓一体化的前奏。
- **实现自助式分析：**基于 Doris 强大的查询能力，我们引入了全新的报表构建方式，通过用户自助构建报表方式，能够快速满足用户的各种灵活需求。

在使用 Doris 的两年多时间里，Moka BI 与 Apache Doris 共同成长、共同进步，可以说 Doris 成就了 Moka BI 强大的性能与优秀的用户体验；也正是 Moka BI 特殊的使用场景，也丰富了 Doris 的优化方向，我们提的很多 Issue 与建议，经过版本更新迭代后使其更具竞争力。在未来的时间里，Moka BI 也会紧跟社区脚步，不断优化、回馈社区，希望 Apache Doris 和 SelectDB 发展越来越好、越来越强大。

从 ClickHouse 到 Apache Doris

腾讯音乐内容库数据平台架构演进实践

作者：腾讯音乐内容库数据平台，张俊、代凯、罗雷

导读：腾讯音乐内容库数据平台旨在为应用层提供库存盘点、分群画像、指标分析、标签圈选等内容分析服务，高效为业务赋能。目前，内容库数据平台的数据架构已经从 1.0 演进到了 4.0 ，经历了分析引擎从 ClickHouse 到 Apache Doris 的替换、经历了数据架构语义层的初步引入到深度应用，有效提高了数据时效性、降低了运维成本、解决了数据管理割裂等问题，收益显著。本文将为大家分享腾讯音乐内容库数据平台的数据架构演进历程与实践思考，希望所有读者从文章中有所启发。

腾讯音乐娱乐集团（简称“腾讯音乐娱乐”）是中国在线音乐娱乐服务开拓者，提供在线音乐和以音乐为核心的社交娱乐两大服务。腾讯音乐娱乐在中国有着广泛的用户基础，拥有目前国内市场知名的四大移动音乐产品：QQ音乐、酷狗音乐、酷我音乐和全民K歌，总月活用户数超过8亿。

业务需求

腾讯音乐娱乐拥有海量的内容曲库，包括录制音乐、现场音乐、音频和视频等多种形式。通过技术和数据的赋能，腾讯音乐娱乐持续创新产品，为用户带来更好的产品体验，提高用户参与度，也为音乐人和合作伙伴在音乐的制作、发行和销售方面提供更大的支持。

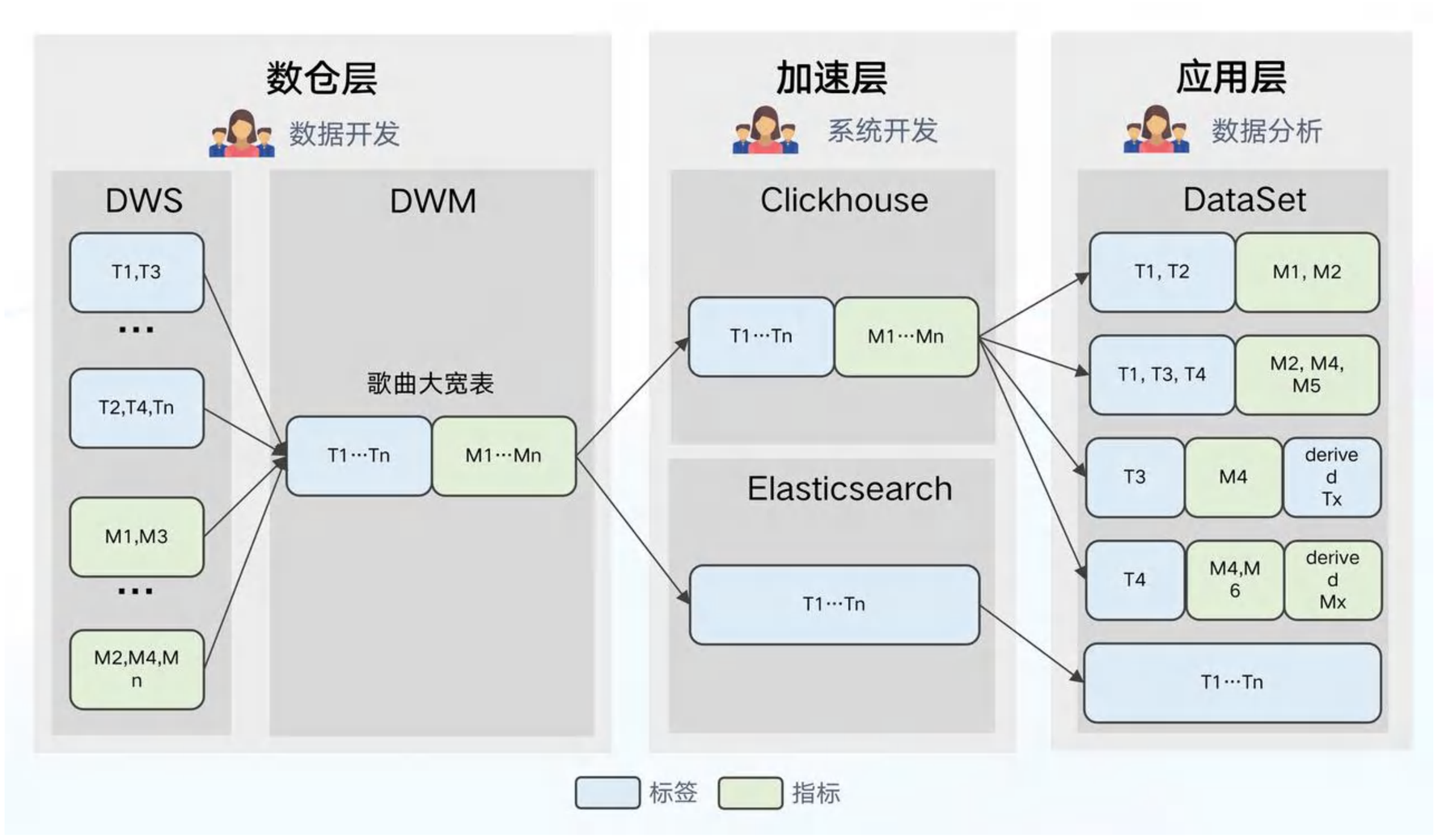
在业务运营过程中我们需要对包括歌曲、词曲、专辑、艺人在内的内容对象进行全方位分析，高效为业务赋能，**内容库数据平台旨在集成各数据源的数据，整合形成内容数据资产（以指标和标签体系为载体），为应用层提供库存盘点、分群画像、指标分析、标签圈选等内容分析服务。**



数据架构演进

TDW 是腾讯最大的离线数据处理平台，公司内大多数业务的产品报表、运营分析、数据挖掘等的存储和计算都是在TDW中进行，内容库数据平台的数据加工链路同样是在腾讯数据仓库 TDW 上构建的。截止目前，内容库数据平台的数据架构已经从 1.0 演进到了 4.0 ，**经历了分析引擎从 ClickHouse 到 Apache Doris 的替换、经历了数据架构语义层的初步引入到深度应用**，有效提高了数据时效性、降低了运维成本、解决了数据管理割裂等问题，收益显著。接下来将为大家分享腾讯音乐内容库数据平台的数据架构演进历程与实践思考。

数据架构 1.0



如图所示为数据架构 1.0 架构图，分为数仓层、加速层、应用层三部分，数据架构 1.0 是一个相对主流的架构，简单介绍一下各层的作用及工作原理：

- 数仓层：通过 ODS–DWD–DWS 三层将数据整合为不同主题的标签和指标体系， DWM 集市层围绕内容对象构建大宽表，从不同主题域 DWS 表中抽取字段。
- 加速层：在数仓中构建的大宽表导入到加速层中，Clickhouse 作为分析引擎，Elasticsearch 作为搜索/圈选引擎。
- 应用层：根据场景创建 DataSet，作为逻辑视图从大宽表选取所需的标签与指标，同时可以二次定义衍生的标签与指标。

存在的问题

- 数仓层：不支持部分列更新，当上游任一来源表产生延迟，均会造成大宽表延迟，进而导致数据时效性下降。
- 加速层：不同的标签跟指标特性不同、更新频率也各不相同。由于 ClickHouse 目前更擅长处理宽表场景，无区别将所有数据导入大宽表生成天的分区将造成存储资源的浪费，维护成本也将随之升高。
- 应用层：ClickHouse 采用的是计算和存储节点强耦合的架构，架构复杂，组件依赖严重，牵一发而动全身，容易出现集群稳定性问题，对于我们来说，同时维护 ClickHouse 和 Elasticsearch 两套引擎的连接与查询，成本和难度都比较高。

除此之外，ClickHouse 由国外开源，交流具有一定的语言学习成本，遇到问题无法准确反馈、无法快速获得解决，与社区沟通上的阻塞也是促进我们进行架构升级的因素之一。

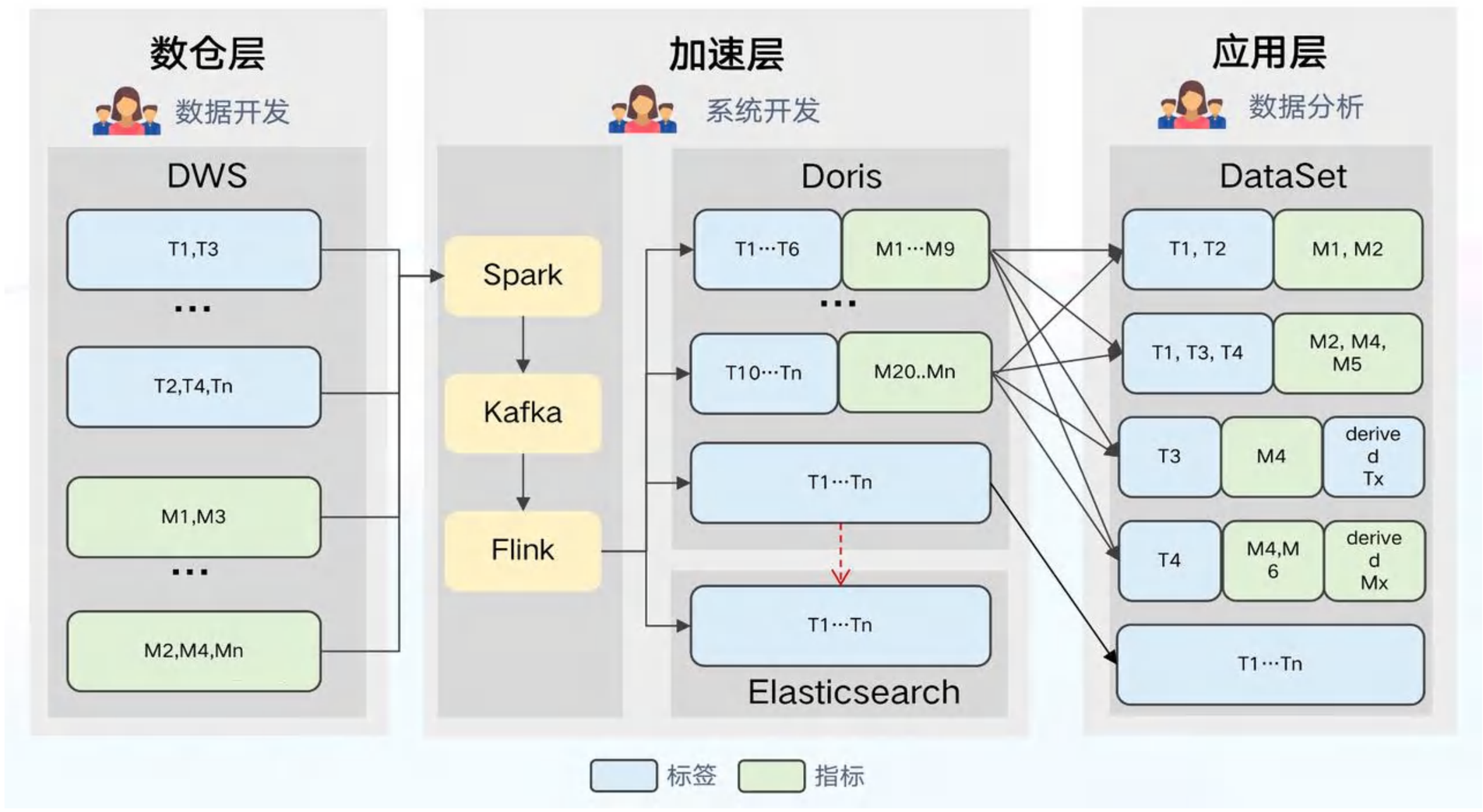
数据架构 2.0

基于架构 1.0 存在的问题和 ClickHouse 的局限性，我们尝试对架构进行优化升级（架构图见下一页），**将分析引擎 ClickHouse 切换为 Doris**，Doris 具有以下的优势：

Apache Doris 的优势：

- Doris 架构极简易用，部署只需两个进程，不依赖其他系统，运维简单；兼容 MySQL 协议，并且使用标准 SQL。
- 支持丰富的数据模型，可满足多种数据更新方式，支持部分列更新。

- 支持对 Hive、Iceberg、Hudi 等数据湖和 MySQL、Elasticsearch 等数据库联邦查询分析。
- 导入方式多样，支持从 HDFS/S3 等远端存储批量导入，也支持读取 MySQL Binlog 以及订阅消息队列 Kafka 中的数据，还可以通过 Flink Connector 实时/批次同步数据源（MySQL, Oracle, PostgreSQL 等）到 Doris。
- 社区目前 Apache Doris 社区活跃、技术交流更多，SelectDB 针对社区有专职的技术支持团队，在使用过程中遇到问题均能快速得到响应解决。



同时我们也利用 Doris 的特性，解决了架构 1.0 中较为突出的问题。

- 数仓层： Apache Doris 的 Aggregate 数据模型可支持部分列实时更新，因此我们去掉了 DWM 集市层的构建，直接增量到 Doris / ES 中构建宽表，**解决了架构 1.0 中上游数据更新延迟导致整个宽表延迟的问题，进而提升数据的时效性**。数据（指标、标签等）通过 Spark 统一离线加载到 Kafka 中，使用 Flink 将数据增量更新到 Doris 和 ES 中（利用 Flink 实现进一步的聚合，减轻了 Doris 和 ES 的更新压力）。
- 加速层：该层主要将大宽表拆为小宽表，根据更新频率配置不同的分区策略，减小数据冗余带来的存储压力，提高查询吞吐量。**Doris 具备多表查询和联邦查询性能特性，可以利用多表关联特性实现组合查询。**
- 应用层：DataSet 统一指向 Doris，**Doris 支持外表查询**，利用该特性可以对 ES 引擎直接进行查询。

架构 2.0 存在的问题：

- DataSet 灵活度较高，数据分析师可对指标和标签自由组合和定义，但是不同的分析师对同一数据的定义不尽相同、定义口径不一致，导致指标和标签缺乏统一管理，这使得数据管理和使用的难度都变高。

- Dataset 与物理位置绑定，应用层无法进行透明优化，如果 Doris 引擎出现负载较高的情况，无法通过降低用户查询避免集群负载过高报错的问题。

数据架构 3.0

针对指标和标签定义口径不统一，数据使用和管理难度较高的问题，我们继续对架构进行升级。数据架构 3.0 主要的变化是引入了专门的语义层，语义层的主要作用是将技术语言转换为业务部门更容易理解的概念，目的是将标签 (tag) 与指标 (metric) 变为“一等公民”，作为数据定义与管理的基本对象。

引入语义层的优势有：

- 对于技术来说，应用层不再需要创建 DataSet，从语义层可直接获取特定内容对象的标签集 (tagset) 和指标集 (metricset) 来发起查询。

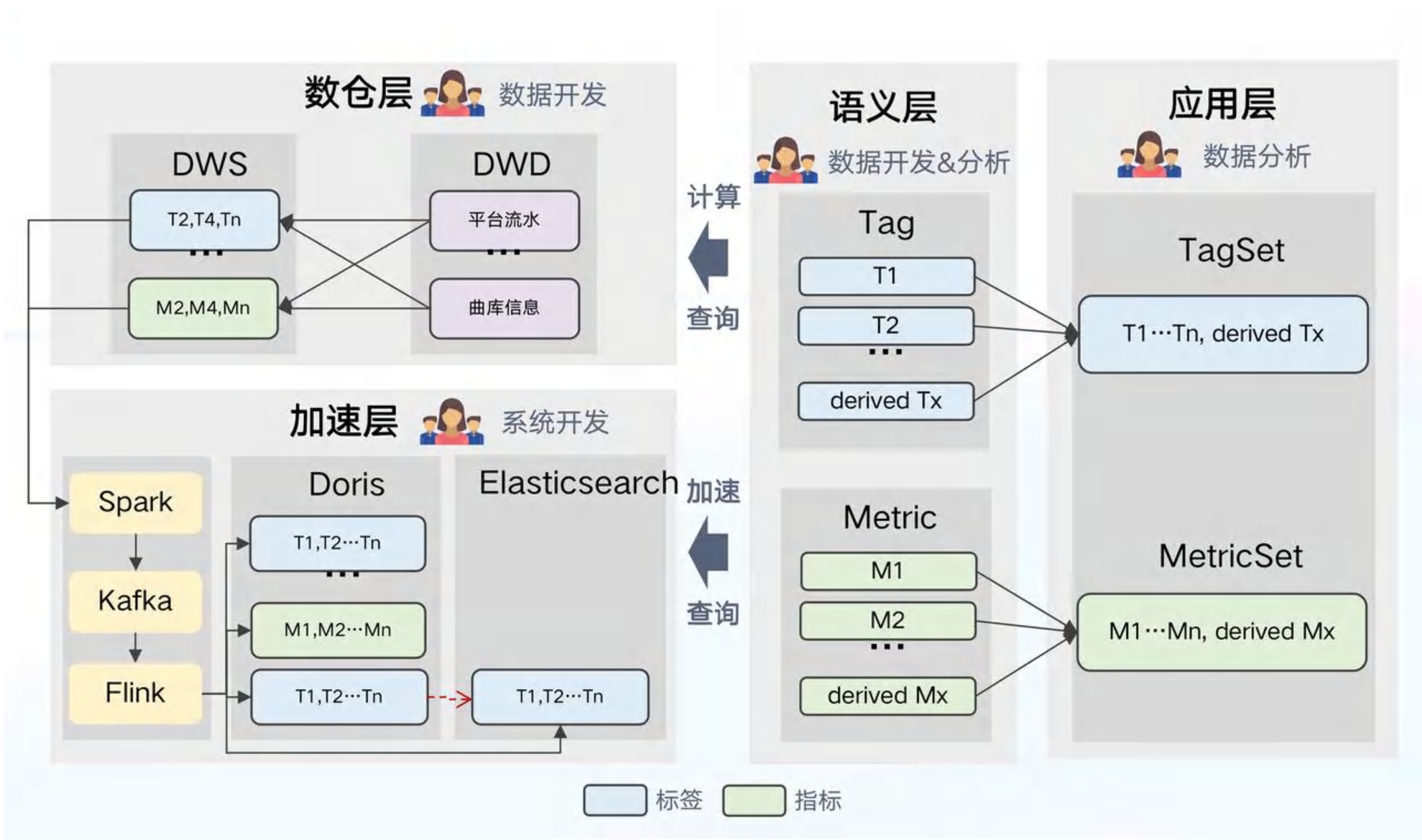
- 对于数据分析师来说，可统一在语义层定义和创建衍生的指标和标签，解决了定义口径不一致、管理和使用难度较高的问题。
- 对于业务来说，无需耗费过长时间考虑什么场景应选择哪个数据集使用，语义层对标签和指标透明统一的定义提升了工作效率、降低了使用成本。

存在的问题：

从架构图可知，标签和指标等数据均处于下游位置，虽然标签与指标在语义层被显式定义，但仍然无法影响上游链路，数仓层有自己的语义逻辑，加速层有自己的导入配置，**这样就造成了数据管理机制的割裂。**

数据架构 4.0

在数据架构 3.0 的基础上，我们对语义层进行更深层次的应用，在数据架构 4.0 中，我们将语义层变为架构的中枢节点，目标是对所有的指标和标签统一定义，从计算–加速–查询实现中心化、标准化管理，解决数据管理机制割裂的问题。



语义层作为架构中枢节点所带来的变化：

- 数仓层：语义层接收 SQL 触发计算或查询任务。数仓从 DWD 到 DWS 的计算逻辑将在语义层中进行定义，且以单个指标和标签的形式进行定义，之后由语义层来发送命令，生成 SQL 命令给数仓层执行计算。
- 加速层：从语义层接收配置、触发导入任务，比如加速哪些指标与标签均由语义层指导。
- 应用层：向语义层发起逻辑查询，由语义层选择引擎，生成物理 SQL。

架构优势：

- 可以形成统一视图，对于核心指标和标签的定义进行统一查看及管理。
- 应用层与物理引擎完成解耦，可进一步对更加灵活易用的架构进行探索：如何对相关指标和标签进行加速，如何在时效性和集群的稳定性之间平衡等。

存在的问题：

因为当前架构是对单个标签和指标进行了定义，因此如何在查询计算时自动生成一个准确有效的 SQL 语句是非常有难度的。如果你有相关的经验，期待有机会可以一起探索交流。

优化经验

从上文已知，为更好地实现业务需求，数据架构演进到 4.0 版本，其中 **Apache Doris 作为分析加速场景的解决方案在整个系统中发挥着重要的作用**。接下来将从场景需求、数据导入、查询优化以及成本优化四个方面出发，分享基于 Doris 的读写优化经验，希望给读者带来一些参考。

场景需求

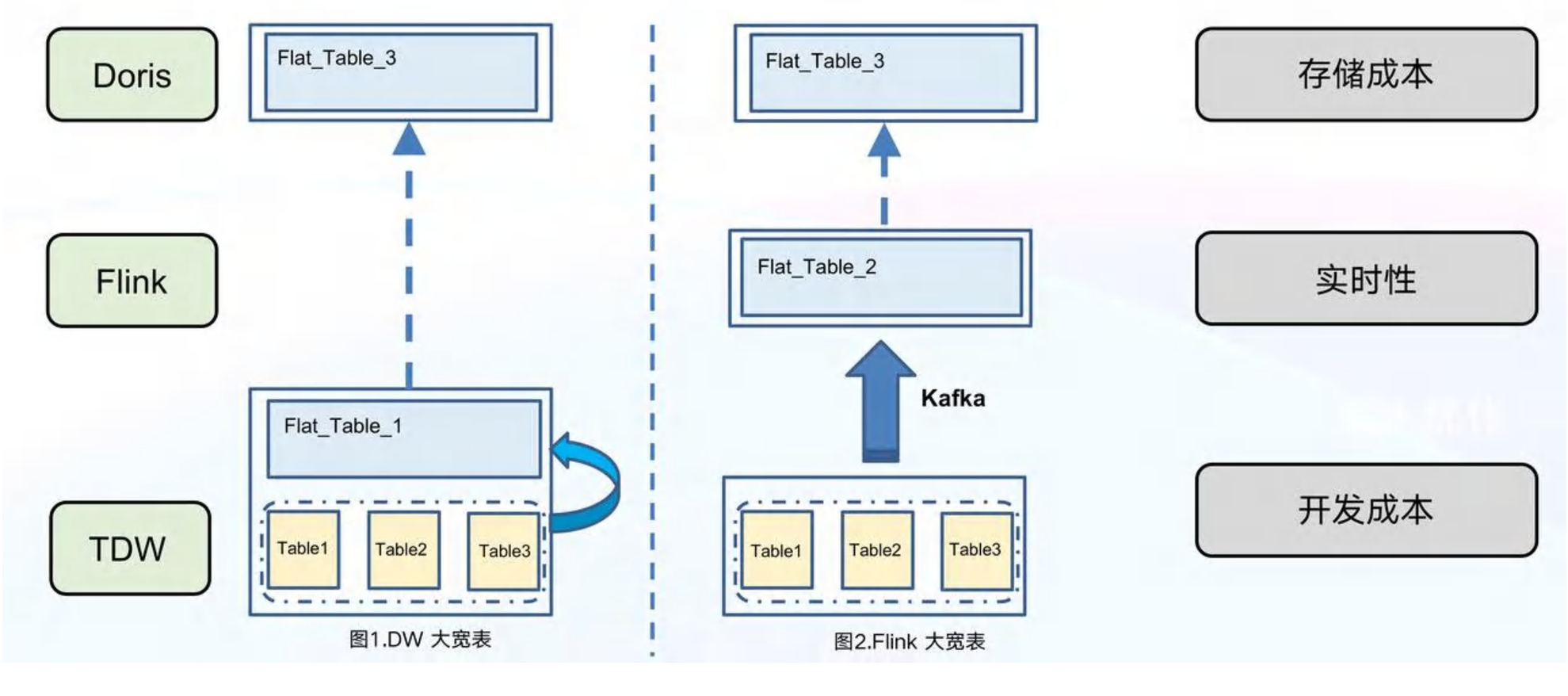
目前我们有 800+ 标签， 1300+ 指标，对应 TDW 中有 80 + Source 表，单个标签、指标的最大基数达到了 2 亿+（图见下页）。我们希望将这些数据从 TDW 加速到 Doris 中完成标签画像和指标的分析。**从业务的角度，需要满足以下要求：**

- **实时可用：**标签/指标导入以后，需实现数据尽快可用。不仅要支持常规离线导入 T+1 ，同时也要支持实时打标场景。
- **部分更新：**因每个 Source 表由各自 ETL 任务产出对应的数据，其产出时间不一致，并且每个表只涉及部分指标或标签，不同数据查询对时效性要求也不同，因此架构需要支持部分列更新。

- **性能高效：**具备高效的写入能力，且在圈选、洞察、报表等场景可以实现秒级响应。
- **控制成本：**在满足业务需求的前提下，最大程度地降低成本；支持冷热数据精细化管理，支持标签灵活上下架。



数据导入方案



为了减轻 Doris 写入压力，我们考虑在数据写入 Doris 之前，尽量将数据生成宽表，再写入到 Doris 中。**针对宽表的生成，我们有两个实现思路**：第一个是在 TDW 数仓中生成宽表；第二个是 Flink 中生成宽表。我们对这两个实现思路进行了实践对比，最终决定选择第二个实现思路，原因如下：

在 TDW 中生成宽表，虽然链路简单，但是弊端也比较明显。

- 存储成本较高， TDW 除了要维护离散的 80 +个 Source 表外，还需维护 1 个大宽表、2 份冗余的数据。
- 实时性比较差， 由于每个 Source 表产出的时间不一样，往往会因为某些延迟比较大的 Source 表导致整个数据链路延迟增大。
- 开发成本较高，该方案只能作为离线方式，若想实现实时方式则需要投入开发资源进行额外的开发。

而**在 Flink 中生成宽表，链路简单、成本低也容易实现**，主要流程是：首先用 Spark 将相关 Source 表最新数据离线导入到 Kafka 中， 接着使用 Flink 来消费 Kafka，并通过主键 ID 构建出一张大宽表，最后将大宽表导入到 Doris 中。如下图所示，来自数仓 N 个表中 ID = 1 的 5 条数据，经过 Flink 处理以后，只有一条 ID = 1 的数据写入 Doris 中，大大减少 Doris 写入压力。



图3.Flink 大宽表方案数据流图

通过以上导入优化方案，**极大地降低了存储成本**， TDW 无需维护两份冗余的数据，Kafka 也只需保存最新待导入的数据。同时该方案**整体实时性更好且可控**，并且大宽表聚合在 Flink 中执行，可灵活加入各种 ETL 逻辑，离线和实时可对多个开发逻辑进行复用，**灵活度较高**。

数据模型选择

目前我们生产环境所使用的版本为 Apache Doris 1.1.3，我们对其所支持的 Unique 主键模型、Aggregate 聚合模型和 Duplicate 明细模型进行了对比 ，相较于 Unique 模型和 Duplicate 模型，**Aggregate 聚合模型满足我们部分列更新的场景需求：**

Aggregate 聚合模型可以支持多种预聚合模式，可以通过 `REPLACE_IF_NOT_NULL` 的方式实现部分列更新。数据写入过程中，Doris 会将多次写入的数据进行聚合，最终用户查询时，返回一份聚合后的完整且正确的数据。

另外两种数据模型适用的场景，这里也进行简单的介绍：

- Unique 模型适用于需要保证 Key 唯一性场景，同一个主键 ID 多次导入之后，会以 append 的方式进行行级数据更新，仅保留最后一次导入的数据。在与社区进行沟通后，**确定后续版本 Unique 模型也将支持部分列更新**。
- Duplicate 模型区别于 Aggregate 和 Unique 模型，数据完全按照导入的明细数据进行存储，不会有任何预聚合或去重操作，即使两行数据完全相同也都会保留，因此 Duplicate 模型适用于既没有聚合需求，又没有主键唯一性约束的原始数据存储。

确定数据模型之后，我们在建表时如何对列进行命名呢？可以直接使用指标或者是标签的名称吗？

在使用场景中通常会有以下几个需求：

- 为了更好地表达数据的意义，业务方会有少量修改标签、指标名称的需求。
- 随着业务需求的变动，标签经常存在上架、下架的情况。
- 实时新增的标签和指标，用户希望数据尽快可用。

Doris 1.1.3 是不支持对列名进行修改的，如果直接使用指标/标签名称作为列名，则无法满足上述标签或指标更名的需求。而对于上下架标签的需求，如果直接以 drop/add column 的方式实现，则会涉及数据文件的更改，该操作耗时耗力，甚至会影响线上查询的性能。

那么，有没有更轻量级的方式来满足需求呢？接下来将为大家分享相关解决方案及收益：

- **为了实现少量标签、指标名称修改**，我们用 MySQL 表存储相应的元数据，包括名称、全局唯一的 ID 和上下架状态等信息，比如标签歌曲名称 `song_name` 的 ID 为 4，在 Doris 中存储命名为 a4，用户使用更具有业务含义 `song_name` 进行查询。在查询 Doris 前，我们会在查询层将 SQL 改写成具体的列名 a4。这样名称的修改只是修改其元数据，底层 Doris 的表结构可以保持不变。

- **为了实现标签灵活上下架**，我们通过统计标签的使用情况来分析标签的价值，将低价值的标签进入下架流程。下架指的是对元信息进行状态标注，在下架标签重新上架之前，不会继续导入其数据，元信息中数据可用时间也不会发生变化。

- **对于实时新增标签/指标**，我们基于名称 ID 的映射在 Doris 表中预先创建适量 ID 列，当标签/指标完成元信息录入后，直接将预留的 ID 分配给新录入的标签/指标，避免在查询高峰期因新增标签/指标所引起的 Schema Change 开销对集群产生的影响。经测试，用户在元信息录入后 10 分钟内就可以使用相应的数据。

值得关注的是，**在社区近期发布的 1.2.0 版本中，增加了 Light Schema Change 功能， 对于增减列的操作不需要修改数据文件，只需要修改 FE 中的元数据，从而可以实现毫秒级的 Schame Change 操作。**同时开启 Light Schema Change 功能的数据表也可以支持列名的修改，这与我们的需求十分匹配，后续我们也会及时升级到最新版本。

写入优化

接着我们在数据写入方面也进行了调整优化，这里几点小经验与大家分享：

- Flink 预聚合：通过主键 ID 预聚合，减少写入压力。（前文已说明，此处不再赘述）

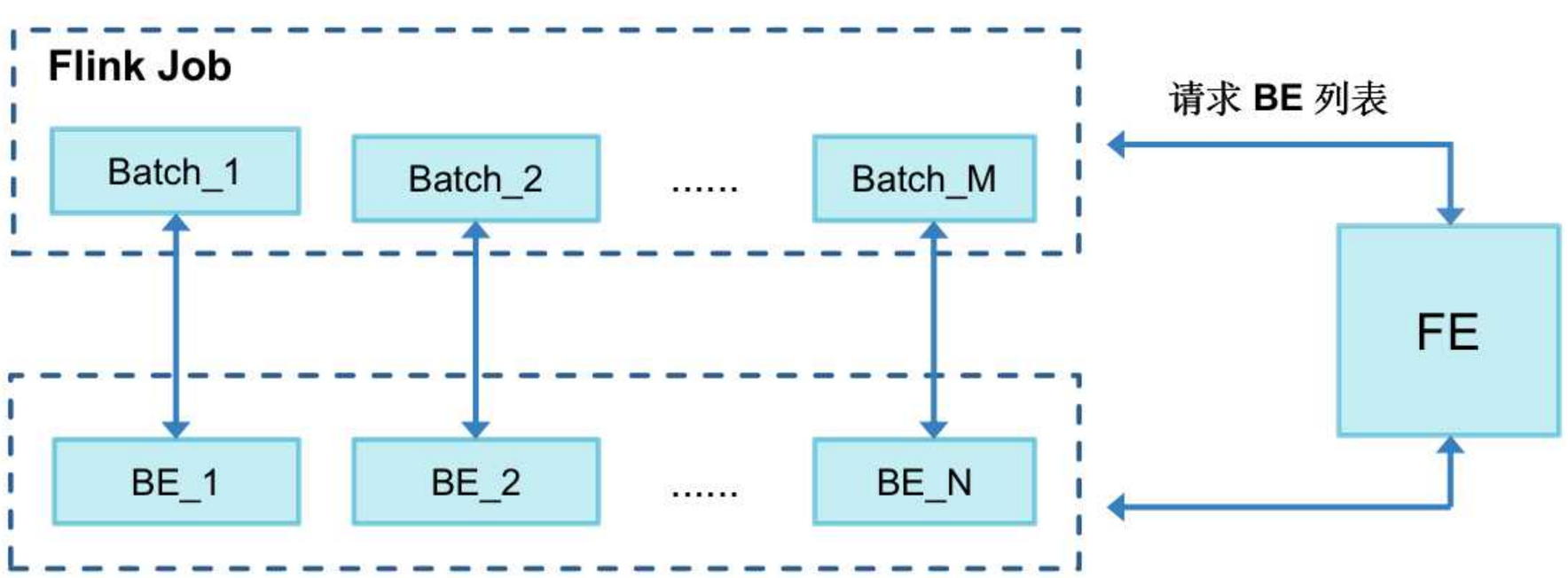
- 写入 Batch 大小自适应变更：为了不占用过多 Flink 资源，我们实现了从同一个 Kafka Topic 中消费数据写入到不同 Doris 表中的功能，并且可以根据数据的大小自动调整写入的批次，尽量做到攒批低频写入。

- Doris 写入调优：针对– 235 报错进行相关参数的调优。比如设置合理的分区和分桶（Tablet 建议1–10G），同时结合场景对 Compaction 参数调优：

```
max_XXXX_compaction_thread
max_cumulative_compaction_num_singleton_deltas
```

- 优化 BE 提交逻辑：定期缓存 BE 列表，按批次随机提交到 BE 节点，细化负载均衡粒度。

> 优化背景：在写入时发现某一个 BE 负载会远远高于其他的 BE，甚至出现 OOM。结合源码发现：作业启动后会获取一次 BE 地址列表，从中随机选出一个 BE 作为 Coordinator 协调者，该节点主要负责接收数据、并分发到其他的 BE 节点，除非作业异常报错，否则该节点不会发生切换。
>
> 对于少量 Flink 作业大数据场景会导致选中的 BE 节点负载较高，因此我们尝试对 BE 提交逻辑进行优化，设置每 1 小时缓存一次 BE 列表，每写入一个批次都随机从 BE 缓存列表中获取一个进行提交，这样负载均衡的粒度就从 job 级别细化到每次提交的批次，使得 BE 间负载更加的均衡，这部分实现我们已经贡献到社区，欢迎大家一起使用并反馈。
>
> - https://github.com/apache/doris-spark-connector/pull/59
> - https://github.com/apache/doris-spark-connector/pull/60
> - https://github.com/apache/doris-spark-connector/pull/61



通过以上数据导入的优化措施，使得整体导入链路更加稳定，每日离线**导入时长下降了 75%**，数据版本累积情况也有所改善，其中 cumu compaction 的**合并分数更是从 600+ 直降到 100 左右**，优化效果十分明显。

整体导入链路更加稳定，每日离线导入时长 75% ↓

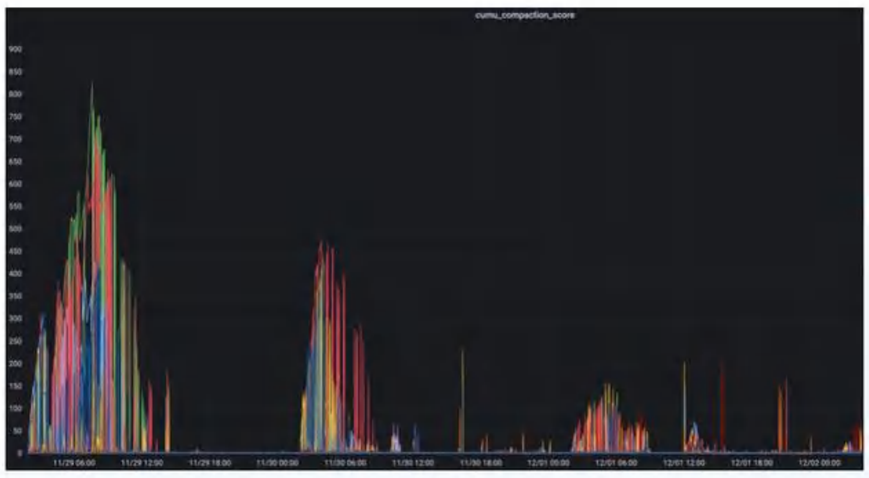


图5. cumu compaction score

600 + ⇨ 100

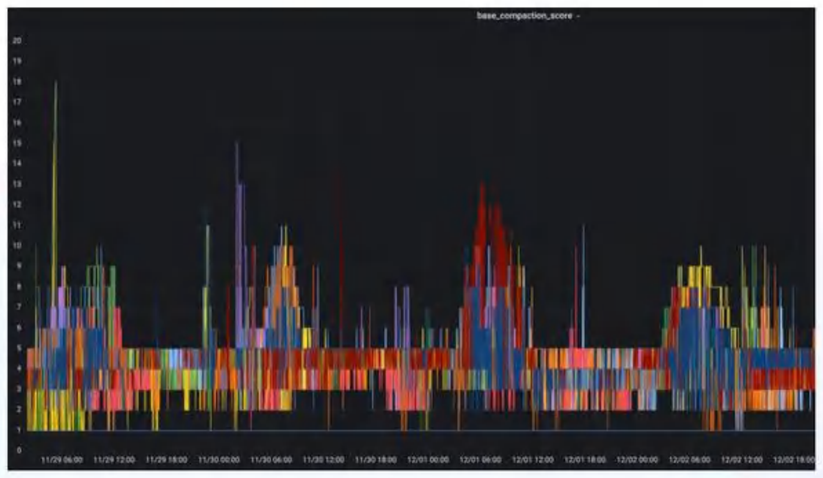


图6. base compaction score

18 ⇨ 10

查询优化

目前我们的场景指标数据是以分区表的形式存储在 Doris 中， ES 保留一份全量的标签数据。在我们的使用场景中，标签圈选的使用率很高，大约有 60% 的使用场景中用到了标签圈选，在标签圈选场景中，通常需要满足以下几个要求：

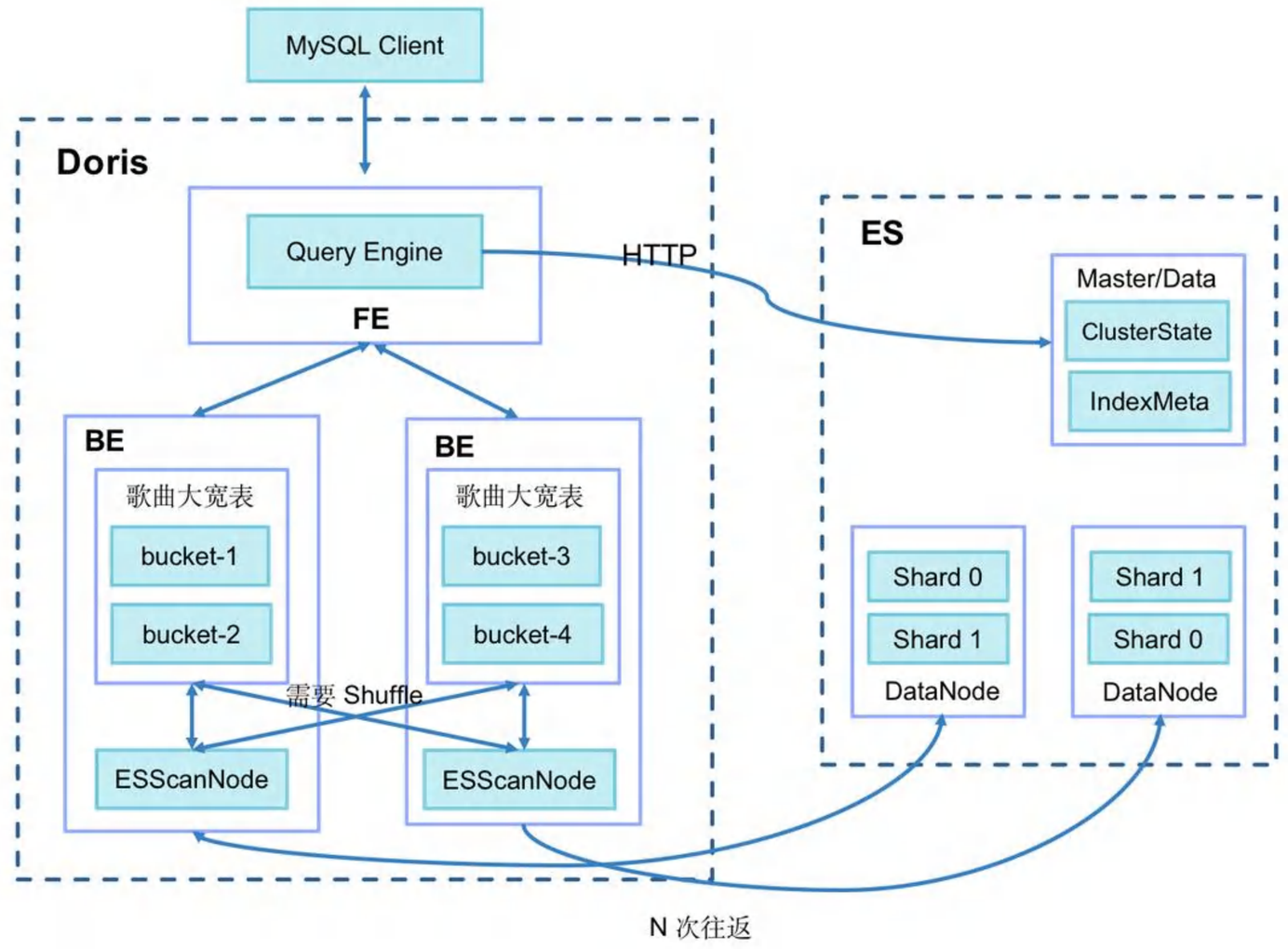
- 用户圈选逻辑比较复杂，数据架构需要支持同时有上百个标签做圈选过滤条件。
- 大部分圈选场景只需要最新标签数据，但是在指标查询时需要支持历史的数据的查询。
- 基于圈选结果，需要进行指标数据的聚合分析。
- 基于圈选结果，需要支持标签和指标的明细查询。

经过调研，我们最终采用了 **Doris on ES 的解决方案**来实现以上要求，将 Doris 的分布式查询规划能力和 ES 的全文检索能力相结合。Doris on ES 主要查询模式如下所示：

```
SELECT tag, agg(metric)
   FROM Doris
   WHERE id in (select id from Es where tagFilter)
   GROUP BY tag
```

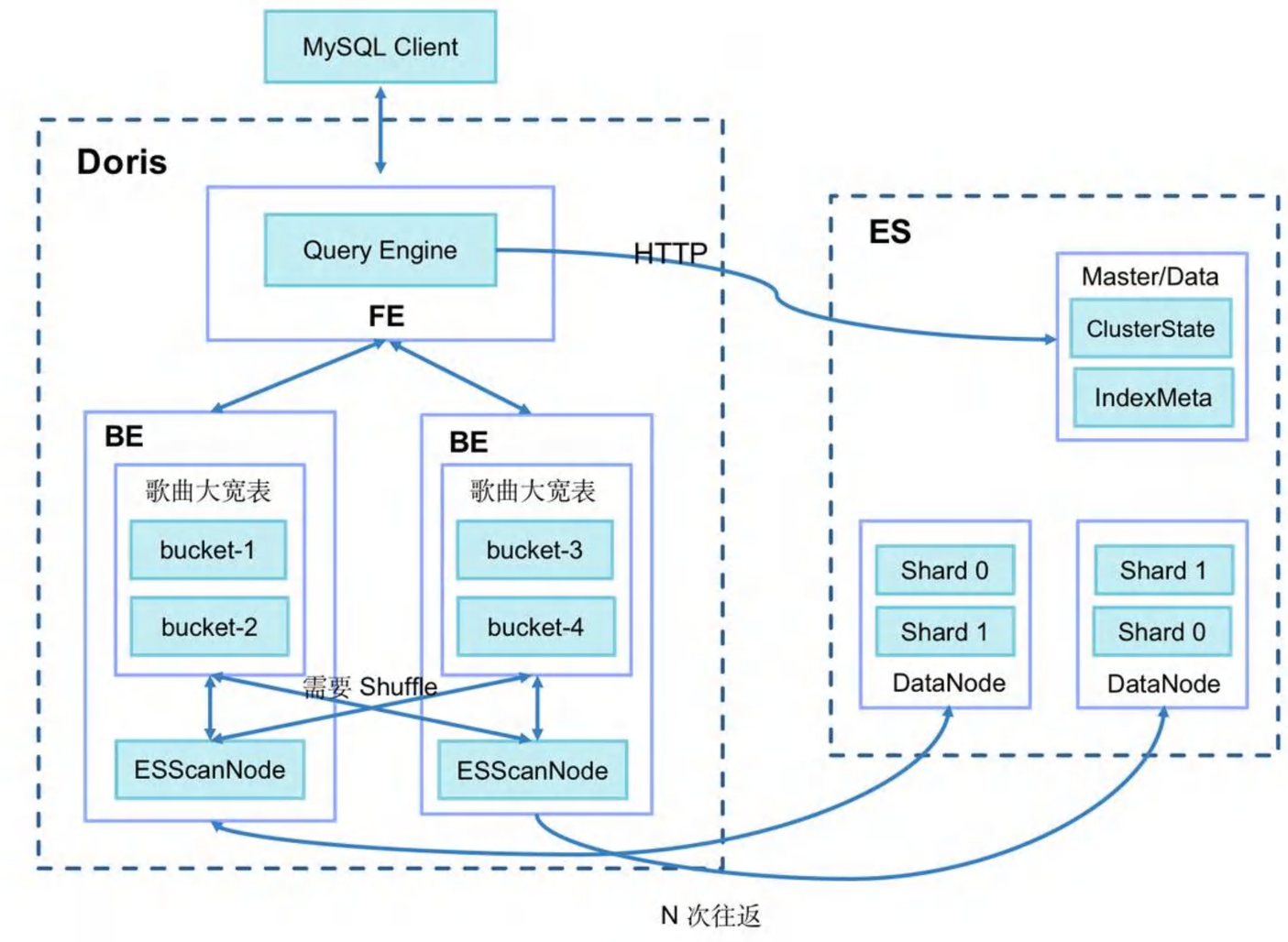
在 ES 中圈选查询出的 ID 数据，以子查询方式在 Doris 中进行指标分析。

我们在实践中发现，查询时长跟圈选的群体大小相关。如果从 ES 中圈选的群体规模超过 100 万时，查询时长会达到 60 秒，圈选群体再次增大甚至会出现超时报错。经排查分析，主要的耗时包括两方面：



- BE 从 ES 中拉取数据（默认一次拉取 1024 行），对于 100 万以上的群体，网络 IO 开销会很大。
- BE 数据拉取完成以后，需要和本地的指标表做 Join，一般以 SHUFFLE/BROADCAST 的方式，成本较高。

针对这两点，我们进行了以下优化：



- 增加了查询会话变量 `es_optimize` ，以开启优化开关；
- 数据写入 ES 时，新增 BK 列用来存储主键 ID Hash 后的分桶序号，算法和 Doris 的分桶算法相同（CRC32）；
- BE 生成 Bucket Join 执行计划，将分桶序号下发到 BE ScanNode 节点，并下推到 ES；
- ES 对查询出的数据进行 Bitmap 压缩，并将数据的多批次获取优化为一次获取，减少网络 IO 开销；
- Doris BE 只拉取和本地 Doris 指标表相关 Bucket 的数据，直接进行本地 Join，避免 Doris BE 间数据再 Shuffle 的过程。

通过以上优化措施，**百万分群圈选洞察查询时间从最初的 60 秒缩短到 3.7 秒**，性能显著提升！

经过与社区沟通交流，**Apache Doris 从 2.0.0 版本开始，将支持倒排索引**。可进行文本类型的全文检索；支持中文、英文分词；支持文本、数值日期类型的等值和范围过滤；倒排索引对数组类型也提供了支持，多个过滤条件可以任意进行 AND OR NOT 逻辑组合。由于高性能的向量化实现和面向 AP 数据库的精简优化，**Doris 的倒排索引相较于 ES 会有 3~5 倍性价比提升**，即将在 2 月底发布的 2.0 Preview 版本中可用于功能评估和性能测试，相信在这个场景使用后会有进一步的性能提升。

成本优化

在当前大环境下，降本提效成为了企业的热门话题，**如何在保证服务质量的同时降低成本开销**，是我们一直在思考的问题。在我们的场景中，**成本优化主要得益于 Doris 自身优秀的能力**，这里为大家分享两点：

1、冷热数据进行精细化管理。

- 利用 Doris TTL 机制，在 Doris 中只存储近一年的数据，更早的数据放到存储代价更低的 TDW 中；
- 支持分区级副本设置，3 个月以内的数据高频使用，分区设置为 3 副本 ； 3-6 个月数据分区调整为 2 副本；6 个月之前的数据分区调整为 1 副本；
- 支持数据转冷， 在 SSD 中仅存储最近 7 天的数据，并将 7 天之前的数据转存到到 HDD 中，以降低存储成本；

- 标签上下线，将低价值标签和指标下线处理后，后续数据不再写入，减少写入和存储代价。

2、降低数据链路成本。

Doris 架构非常简单，只有FE 和 BE 两类进程，不依赖其他组件，并通过一致性协议来保证服务的高可用和数据的高可靠，自动故障修复，运维起来比较容易；

- 高度兼容 MySQL 语法，支持标准 SQL，极大降低开发人员接入使用成本；
- 支持多种联邦查询方式，支持对 Hive、MySQL、Elasticsearch、Iceberg 等组件的联邦查询分析，降低多数据源查询复杂度。

通过以上的方式，**使得存储成本降低 42%，开发与时间成本降低了 40%**，成功实现降本提效，后续我们将继续探索！

未来规划

未来我们还将继续进行迭代和优化，我们计划在以下几个方向进行探索：

- 实现自动识别冷热数据，用 Apache Doris 存储热数据，Iceberg 存储冷数据，利用 Doris 湖仓一体化能力简化查询。
- 对高频出现的标签/指标组合，通过 Doris 的物化视图进行预计算，提升查询的性能。
- 探索 Doris 应用于数仓计算任务，利用物化视图简化代码逻辑，并提升核心数据的时效性。

最后，感谢 Apache Doris 社区和 SelectDB 的同学，感谢其快速响应和积极支持，未来我们也会持续将相关成果贡献到社区，希望 Apache Doris 飞速发展，越来越好！

Apache Doris

在小米亿级用户行为分析平台的实践

作者：小米数据智能部开发工程师，汤佳树

导读：过去 3 年时间里，Apache Doris 已经在小米内部得到了广泛的应用，支持了集团数据看板、广告投放/广告 BI、新零售、用户行为分析、A/B 实验平台、天星数科、小米有品、用户画像、小米造车等小米内部数十个业务，并且在小米内部形成了一套以 Apache Doris 为核心的数据生态 。本文将为大家分享小米用户行为分析平台基于 Apache Doris 向量化版本的改造实践，包括数据存储架构和查询服务架构的演进与改造经验分享。

小米用户行为分析统一平台是基于海量数据的一站式、全场景、多模型、多维度、自助式大数据智能洞察分析服务平台，对接各类数据源，进行加工处理、分析挖掘和可视化展现，满足各类用户在用户洞察场景下的数据分析应用需求，提供高效极致的分析体验。

业务需求

平台可以基于数据进行时间分析，留存分析，分布分析，漏斗分析等，业务方主要基于事件进行分析，事件是追踪或记录的用户行为或业务过程，可以是单个事件也可以是多个事件组合的虚拟事件。

数据来源于各业务的打点数据，且基于事件模型进行建模，用户在产品中的各种操作都可以抽象成 Event 实体，并且里面都会包含五要素：

- Who：即参与这个事件的用户是谁，例如用户的唯一 ID
- When：即这个事件发生的实际时间，例如 `time` 字段，记录精确到毫秒的事件发生时间
- Where：即事件发生的地点，例如根据 IP 解析出的省份和城市
- How：即用户从事这个事件的方式，例如用户的设备，使用的浏览器，使用的 App 版本等等

- What：描述用户所做的这件事件的具体内容，例如点击类型的事件，需要记录的字段有点击 URL，点击 Title，点击位置等

数据基于 OLAP 引擎 Doris 进行存储，随着接入业务不断增多，且接入的业务量不断膨胀，**Top 级应用可以达到 100 亿条/天，查询压力和时间相继增大，用户对查询时延的吐槽愈来愈多，我们急切的需要提升查询性能来提升用户的体验。**

痛点问题

针对于业务需求，我们总结了以下痛点问题：

- 为了实现复杂的业务需求，OLAP 分析引擎需要留存、漏斗等分析函数支撑。
- 增量数据 100亿/天，导入压力大，部分业务要求数据导入不丢不重。
- 业务接入不断增多，数据量膨胀，需要 PB 级的数据下的交互式分析查询达到秒级响应。

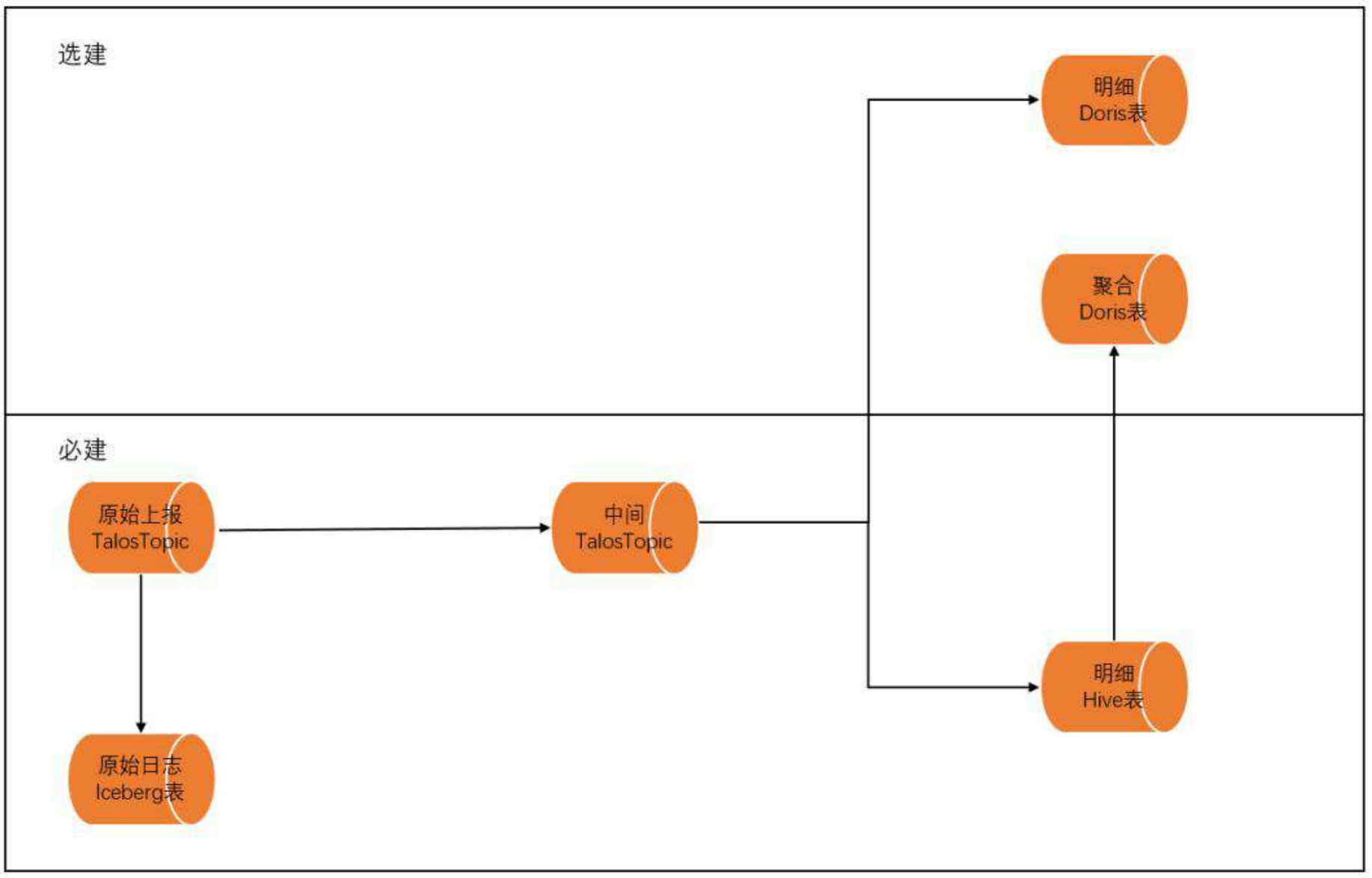
为了解决以上的痛点问题，**我们对比了多款 OLAP 分析引擎，最终选择了 Apache Doris。**Doris 提供了留存、漏斗分析等函数，极大程度的简化了开发的成本。在数据导入的过程中，我们尝试 Doris 刚推出的 Merge On Write Unique Key 导入模型，可以抗住 **100 亿/天**的增量数据压力。针对于向量化查询引擎的改造也是的性能较之前的版本有 **3–5 倍**的提升。

架构演进

一个优秀的系统离不开持续迭代与演进。为了更好的满足业务需求，我们在存储架构与查询引擎两个层面上不断进行尝试，小米用户行为分析系统在上线后，目前已完成 3 次改造，以下将为大家介绍改造历程。

数据存储结构：数据架构的演进

在小米的用户行为分析平台中，原始数据通过小米自研的消息队列 Talos，在 Flink 中清洗与建模后，被下游的 Doris 与 Hive 消费。全量的数据会存储在 Hive 中，进行批量 ETL 或历史数据召回的查询。实时增量数据被存储在 Doris 中，用来做热数据的查询操作。**基于冷热数据分离的架构，我们进行了 3 次架构的演进。**（架构图见下一页）



第一阶段：基于明细宽表的查询

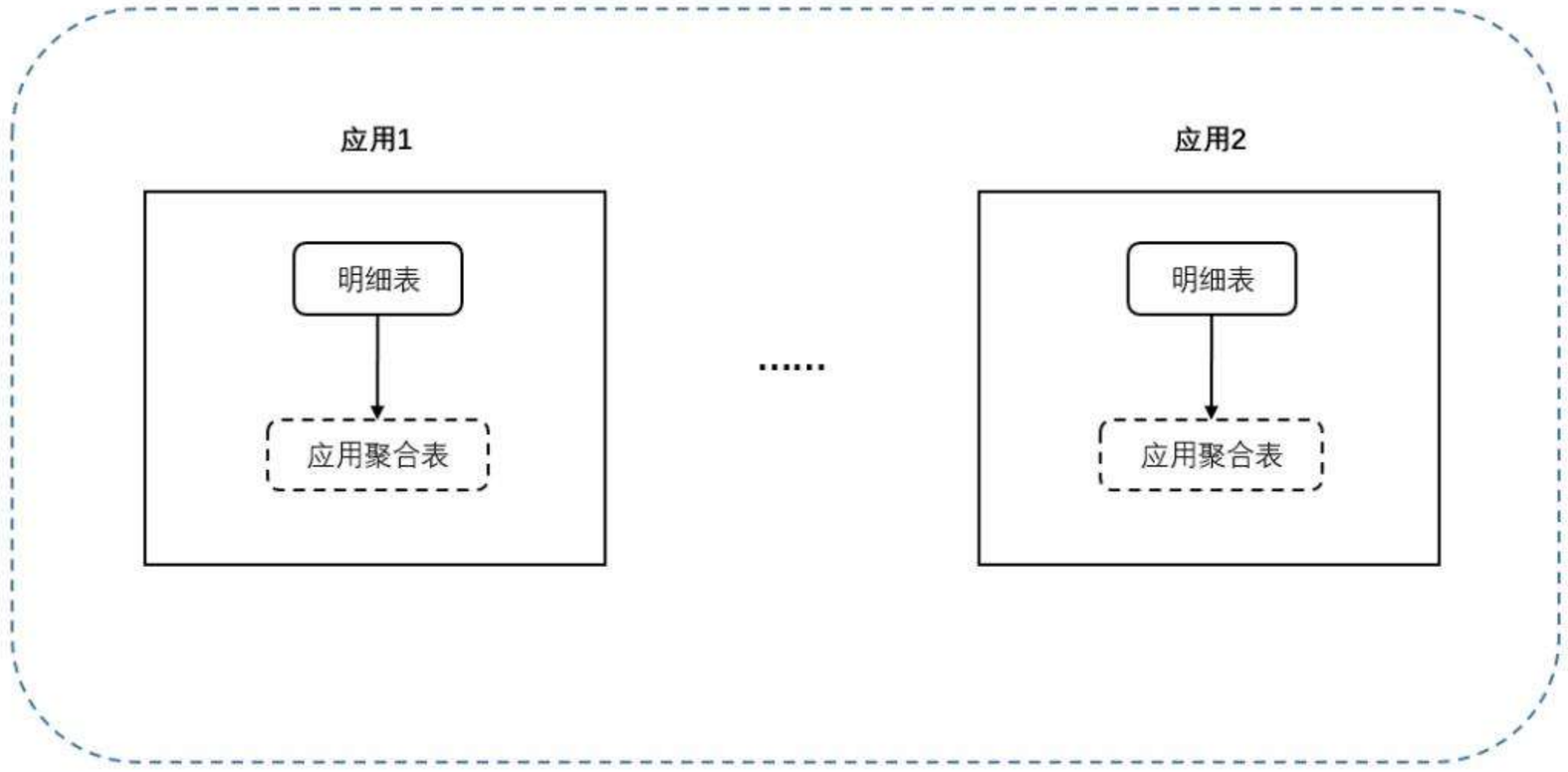
在最初的阶段我们使用了基于明细的宽表查询模式。为了处理灵活多样的分析请求，在系统中，我们配合统一埋点平台处理数据，接入的 OLAP 的数据是直接埋点的全字段展平。在入库之前，我们在 Flink 中将数据打平，以宽表的模式存储在 Doris 明细表中。根据查询的需求，我们将经常使用的列作为建表的维度列，利用前缀索引的特性进行查询加速。

但某些头部大数据量业务容易查询多天数据，一个大查询可能就会将集群资源占满甚至导致集群不可用，且查询耗时相当之久。

第二阶段：基于聚合模型的查询加速

在改造的第二阶段，我们使用了聚合模型对业务查询进行加速。我们对接入行为分析的应用进行统计分析，绝大多数接入行为分析的应用数据量在 1 亿/天数据量以内。对于部分使用频率较高的表，我们采用聚合表完成查询加速，对单天数据量超 10 亿且高频的头部应用做聚合表加速。具体流程为根据数据量挑选出头部应用，对其进行字段解析，并挑选出常用指标及维度，由 Hive 表数据进行聚合 T-1 产出数据，最后写入到 Doris 中，进行查询加速。

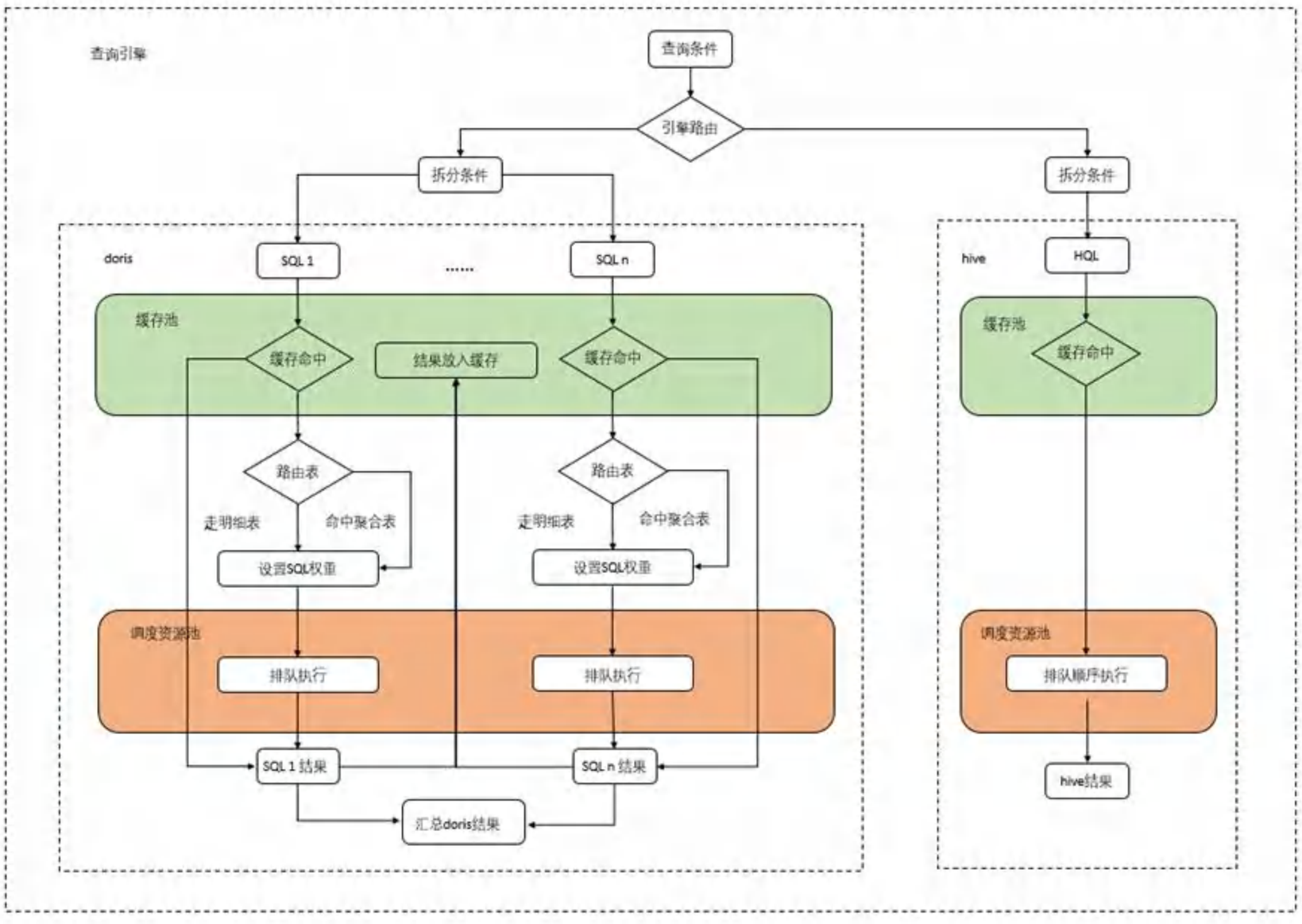
该阶段的改造解决了集群头部业务大查询的问题，此时虽然独立集群存储没问题，但由于其他业务接入后还会持续增加数据量和埋点字段 ，这样会导致元数据最先进入瓶颈。



第三阶段（当前阶段）：业务适配的建表改造

当前阶段，我们对业务需求进行深度解析后重新规划了建表结构。我们对某些应用的埋点字段进行分析，发现有些用户埋点字段多达 500+，但在行为分析里实际用到的可能只有 100+，这显然有所浪费。所以我们与用户沟通调研需求，配合行为分析平台侧的能力，用户可在平台对有用事件和属性进行筛选，同时设置字段映射和过滤逻辑，然后再进行建表。

查询服务架构：查询引擎的改造与演进



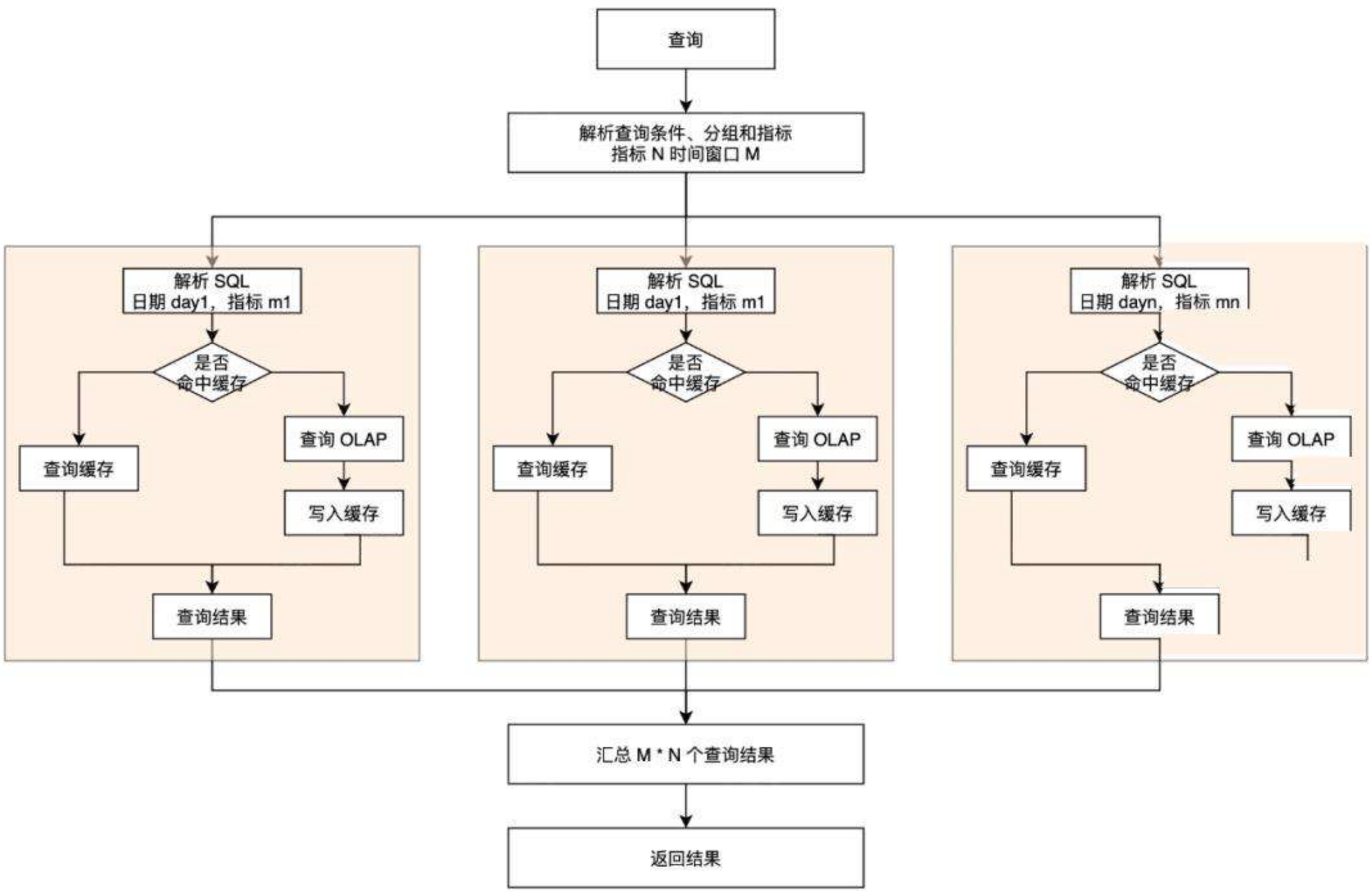
我们基于业务深度改造了查询的服务架构，构建了新的查询引擎架构，实现 SQL 的权重、路由、缓存和资源调度操作。根据查询条件，路由引擎会将 SQL 拆分成多条子查询，在 Doris 或 Hive 中执行后，将子查询的结果汇总，得到最终的结果。**针对查询引擎，我们也进行了 3 次技术架构的改造。**

第一阶段：基于集群粒度的查询资源管理

我们对集群粒度进行查询资源管理，在资源调度中，我们会给每一个 Doris 集群设置一个总的资源池大小（根据集群能力和测试进行量化），根据数据量大小和查询天数对每个 SQL 进行加权，并对资源池的最大最小并行 SQL 数进行限制，如果计算的 SQL 超过限制则进行排队。其次，还会利用 Redis 对数据进行 SQL 级别缓存。

第二阶段：基于 SQL 路由的改造

为适配聚合表加速做了路由层，提升缓存命中率和利用率，此阶段拆分原始提交 SQL，基于指标进行缓存，粒度更细，服务端可根据指标进行适当计算更易于缓存命中。值得一提的是排队时间往往会比较长，有些场景下可能会进行重复提交或拆分成同样的 SQL，为了提高效率会在 SQL 排队前和排队后各进行一次缓存校验。



第三阶段（当前阶段）：基于 SQL 权重的改造

整体架构方面，由于采取了筛选埋点字段而非全量字段导入 Doris，针对头部问题用户，我们会基于查询历史统计指标及维度，根据指定的某些规则进行默认初始化操作，并以此沟通用户并进行引导升级。

此外为了更精细的控制资源调度，本阶段对 SQL 内容进行加权，如含 `DISTINCT` ， `LIKE` ， `VARIANCE_SAMP` 等字样再加权。对于资源消耗较大的操作，如 `DISTINCT` ，会给予更高的权重，调度引擎在执行时会分配更多的资源。

实践应用

数据建模

对业务来讲，分析查询需要较高的灵活度，且是对用户粒度进行分析，所以需要保留较多的维度和指标，我们选用 Doris 作为存储查询引擎，且采用明细表建模，这样可以保证用户能够根据分析需求查出数据。另一方面，由于查询分析是一个延时要求较高的产品，对于数据量大、查询天数多、语句复杂的情况，查询延时会很高，所以对于头部应用，我们根据高频指标维度进行了聚合表模型建模。

```
CREATE TABLE `doris_XXX_event` (
  `olap_date` bigint(20) NOT NULL COMMENT "",
  `event_name` varchar(256) NOT NULL COMMENT "",
  `uniq_id` varchar(256) NOT NULL COMMENT "",
  `dim1` varchar(256) REPLACE NULL COMMENT "",
  `dim2` varchar(256) REPLACE NULL COMMENT "",
  ...
  `cnt` bigint(20) REPLACE NULL COMMENT "",
  `index1` double REPLACE NULL COMMENT "",
  `index2` double REPLACE NULL COMMENT "",
  ...
) ENGINE=OLAP
AGGREGATE KEY(`olap_date`, `event_name`, `uniq_id`)
COMMENT "OLAP"
PARTITION BY RANGE(`olap_date`)
```

数据导入

明细表部分，我们接入 Json 格式 TalosTopic，动态获取 Doris 表的 Schema 信息，通过双缓冲区循环攒批的方式，利用 StreamLoad 向 Doris 中写数据，如果在导入 Doris 时有出现失败的批次，重试 10 次仍然失败，会将数据按照应用粒度存入 HDFS，并在凌晨定时调度任务重新写入 T-1 未写入的数据。

聚合表部分，我们由 Talos 落盘的 Iceberg 表，每日进行 T-1 数据的聚合，根据服务端选取的维度和指标，以及聚合类型（ `count` ，`count distinct` ，`sum` ，`max` ，`min` ），进行聚合存入中间 Hive 表，再由统一导入 Doris 程序进行导入。

数据管理

明细数据和应用聚合表分库存储，TTL 均为 33 天。

数据表会有数据质量监控，如果总行数或者设置指标环比波动太大，会进行告警人工介入确认数据是否有误，视紧急程度进行回补处理。

数据查询及应用

绝大多数用户会锚定事件，进行含指标聚合，去重用户数（几乎占总查询的 50%）的事件行为分析，同时还会有留存分析，漏斗分析，分布分析等分析类型。

建表模型的维护

为了适配业务的变更，上游的埋点信息会周期性的更新。原有的表结构需要进行变更以适配埋点的增加。在过去的 Doris 版本中，Schema Change 是一项相对消耗较大的工作，需要对文件进行修改。在新版本中开启 Light Schema Change 功能后，对于增减列的操作不需要修改文件，只需要修改 FE 中的元数据，从而实现毫秒级的 Schame Change 操作。

应用现状

小米目前在 **300** 多个业务线上线了 Doris 集群，超过 **1.5PB** 的业务数据。在初期我们选择了两个使用较为频繁的集群进行向量化升级。

现迁移 Doris 向量化集群的行为分析业务有 2 个，7 天增量数据的平均值在**百亿左右**，存储空间占用 **7T/天**左右。同时在升级最新版本后，**存储资源有较大的节省，只需要原有集群约 2/3 的存储空间**。

	数据量（亿）/day	存储量-向量化（T）/day	存储量-原集群（T）/day	存储节约比率
业务一	28.87	1.77	2.16	18.03%
业务二	87.63	3.22	5.08	36.71%

机器	机器数量	单机 CPU (BE)	单机 内存	单机 磁盘 (BE SSD)
Doris (c3prc-abtest)	3 FE + 89 BE	2cpu*16core*32线程	32G*8[2400]	480G*8

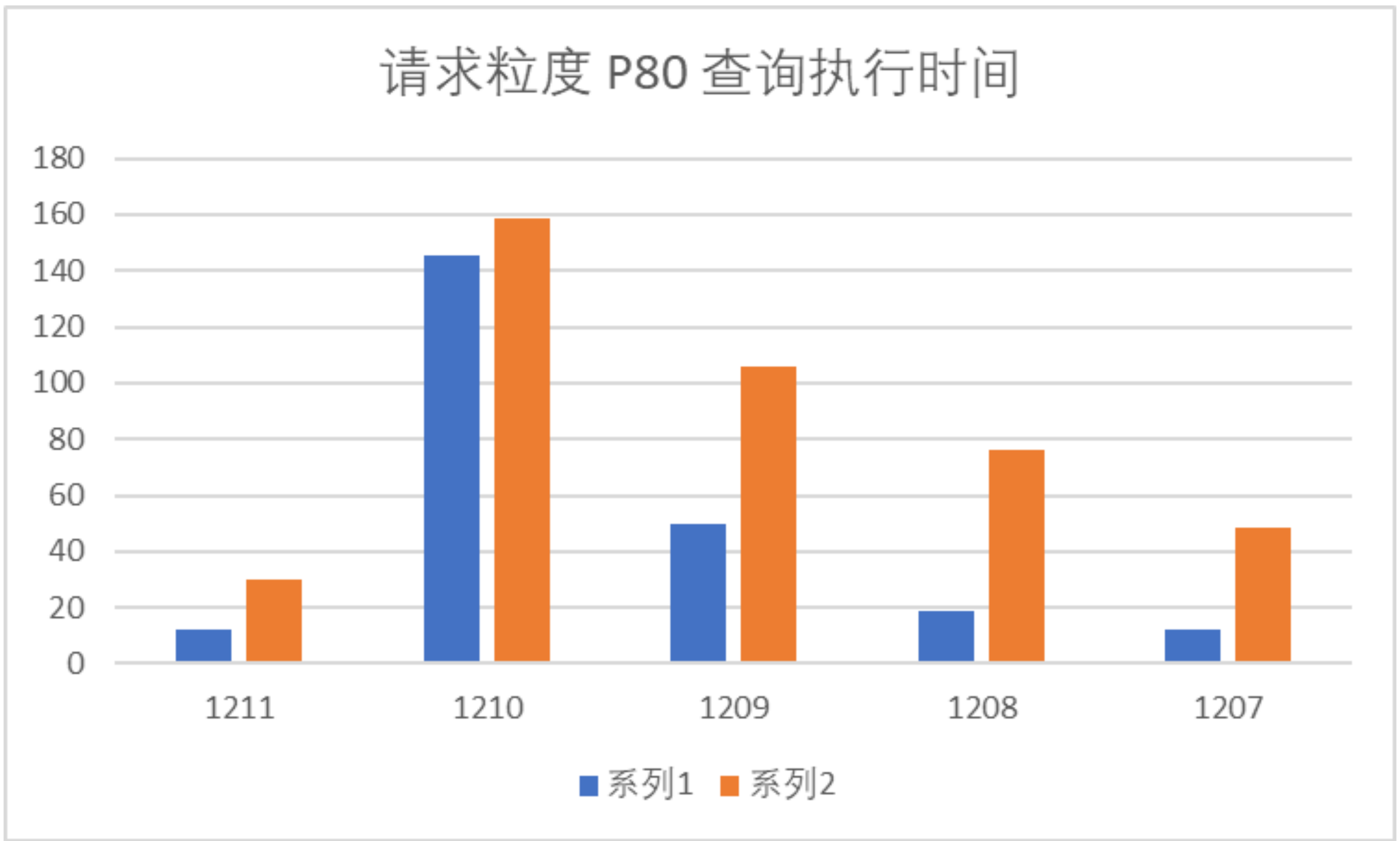
性能提升

请求粒度

升级 Doris 向量化版本后，行为分析平台以请求粒度统计查询耗时 P80 和均值，P80 耗时**下降 43%**，平均耗时**下降 27%**；

统计口径：汇总 12.07–12.11 期间，行为分析请求粒度查询执行时间。

日期	请求数	sql数	向量化p80	向量化平均	非向量化p80	非向量化平均	p80下降	平均下降
20221211	136	2245	12.161	16.709	29.677	24.994	-59.02%	-33.15%
20221210	29	447	145.214	65.07	158.307	52.163	-8.27%	24.74%
20221209	100	1340	49.573	45.587	105.539	72.147	-53.03%	-36.81%
20221208	107	792	18.772	13.815	76.062	41.681	-75.32%	-66.86%
20221207	184	800	12.471	24.396	48.606	36.265	-74.34%	-32.73%
汇总（平均）	111.2	1124.8	47.6382	33.1154	83.6382	45.45	-43.04%	-27.14%

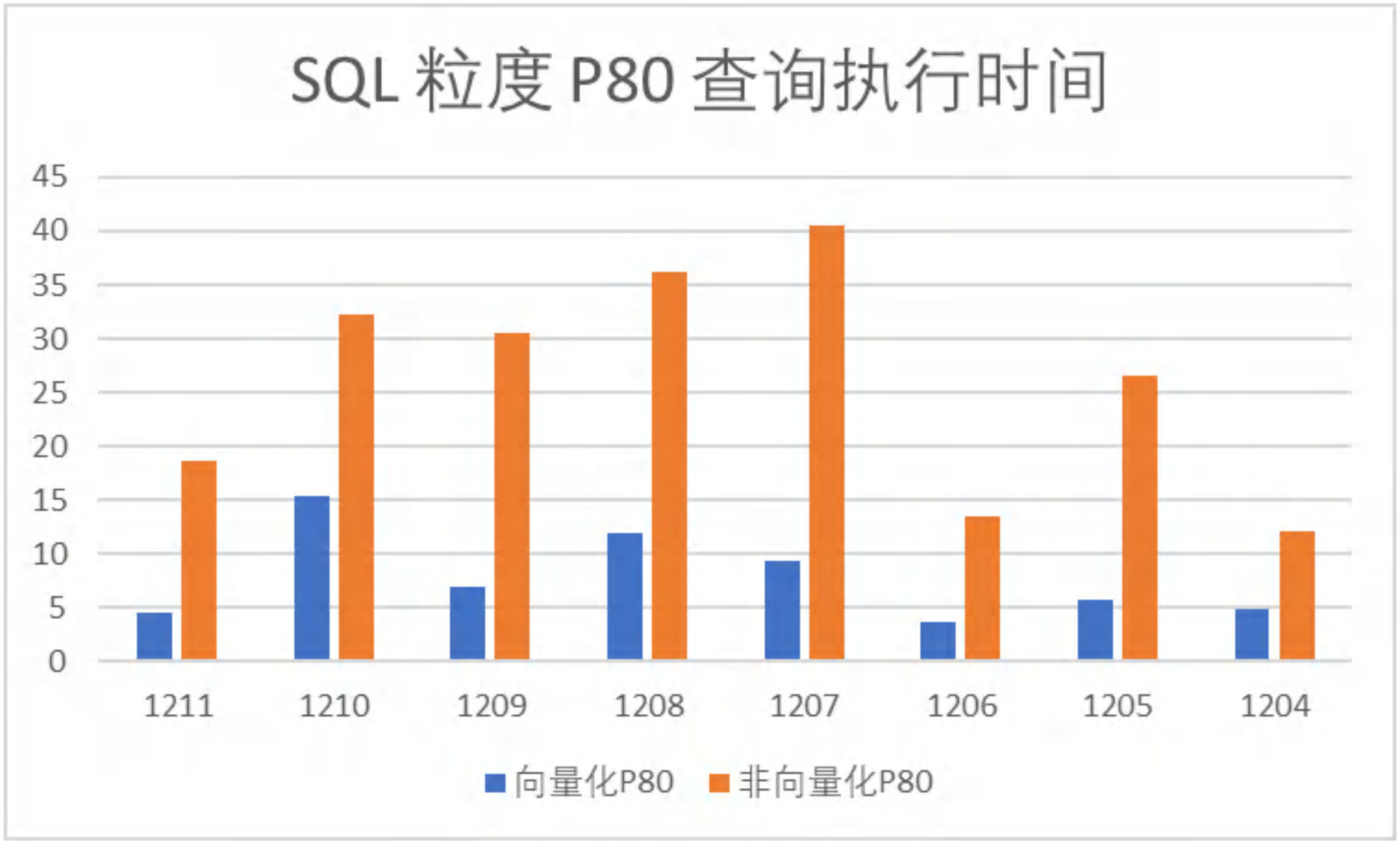


SQL 粒度

升级 Doris 向量化版本后，行为分析平台以 SQL 粒度来统计查询耗时 P80 和均值，耗时 P80 **下降 70%**，平均耗时**下降 54%**。

统计口径：汇总 12.04–12.11 期间，行为分析 SQL 粒度查询执行时间**（未含排队）**

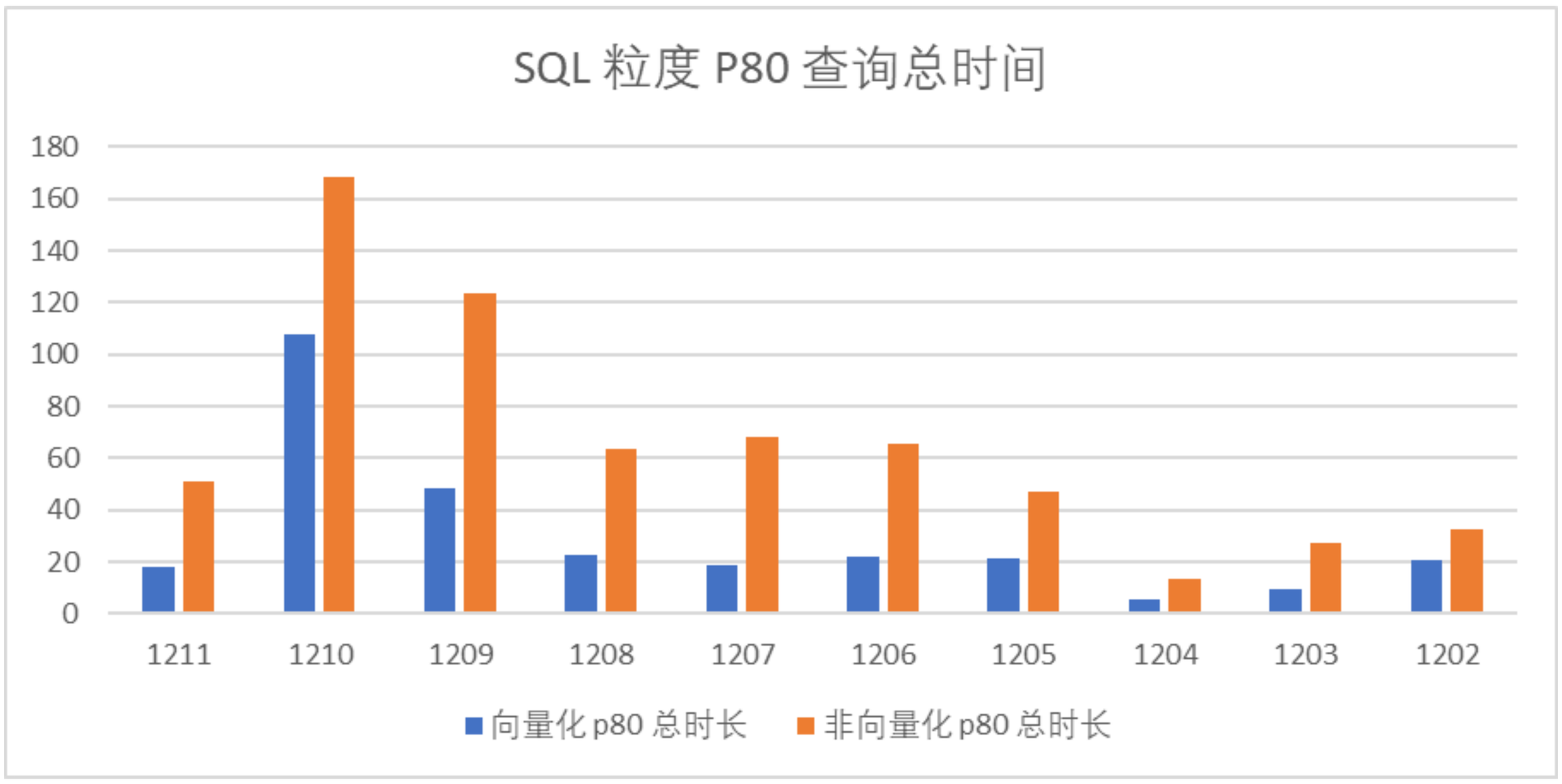
date	cnt	向量化_平均执行时长	向量化_p80执行时长	cnt	非向量化_平均执行时长	非向量化_p80执行时长	平均延时diff	P80延时diff
20221211	2245	5.274	4.558	2245	13.4	18.618	-60.64%	-75.52%
20221210	447	24.997	15.402	447	22.453	32.235	11.33%	-52.22%
20221209	1317	12.534	7.026	1317	31.374	30.474	-60.05%	-76.94%
20221208	792	6.845	11.956	792	23.943	36.174	-71.41%	-66.95%
20221207	800	7.256	9.341	800	22.836	40.507	-68.23%	-76.94%
20221206	2467	3.577	3.713	2467	9.88	13.452	-63.80%	-72.40%
20221205	1366	4.634	5.722	1366	18.769	26.508	-75.31%	-78.41%
20221204	200	2.932	4.92	200	6.385	12.176	-54.08%	-59.59%
汇总-平均	1204	8.506125	7.82975	1204	18.63	26.268	-54.34%	-70.19%



升级 Doris 向量化版本后，行为分析平台以 SQL 粒度统计查询耗时 P80 和均值，耗时 P80 **下降 56%**，平均耗时**下降 44%**；

统计口径：汇总 12.02–12.11，行为分析 SQL 粒度查询总时间**（含排队）**

date	cnt	向量化_平均总时长	向量化_p80总时长	cnt	非向量化_平均总时长	非向量化_p80总时长	平均延时 diff	P80延时 diff
20221211	2245	16.957	18.018	2245	51.24	51.24	-66.91%	-64.84%
20221210	447	69.331	107.85	447	97.121	168.445	-28.61%	-35.97%
20221209	1317	41.683	48.52	1317	72.264	123.349	-42.32%	-60.66%
20221208	792	14.92	22.561	792	63.311	63.311	-76.43%	-64.36%
20221207	800	18.341	18.558	800	41.894	68.224	-56.22%	-72.80%
20221206	2866	28.714	21.69	2866	37.741	65.747	-23.92%	-67.01%
20221205	1366	28.731	21.181	1366	31.528	47.325	-8.87%	-55.24%
20221204	200	3.497	5.484	145	7.197	13.242	-51.41%	-58.59%
20221203	313	9.957	9.509	313	16.713	27.341	-40.42%	-65.22%
20221202	1402	17.383	20.84	1402	26.971	32.722	-35.55%	-36.31%
汇总-平均	1175	24.9514	29.4211	1169	44.598	66.0946	-44.05%	-55.49%



去重的优化

在 ID-Mapping 的时候，通常需要针对 ID 进行去重操作。在最初我们使用了 `COUNT DISTINCT` 来完成去重。

```
SELECT      a.`olap_date` AS `time`,
            count(distinct a.`distinct_id`) AS distinct_id
FROM        analysis.doris_XXX_event a
WHERE       `a`.`olap_date` BETWEEN 20221218 AND 20221220 AND
            a.`event_name` IN(XXXX, XXX, XXX, XXX) AND
            ... ...
GROUP BY    1
ORDER BY    2 DESC
LIMIT       10000
```

在经过优化后，我们使用子查询 + GROUP BY 来替代 COUNT DISTINCT 的功能

```
SELECT      z.`time`,
            count(distinct_id) var1
FROM        (SELECT     a.`olap_date` AS `time`,
                        a.`distinct_id` AS distinct_id
            FROM        analysis.doris_XXX_event a
            WHERE       `a`.`olap_date` BETWEEN 20221218 AND 20221220 AND
                        a.`event_name` (XXXX, XXX, XXX, XXX) AND
                        ... ...
            GROUP BY 1, 2) z
GROUP BY    1
ORDER BY    2 DESC
LIMIT       10000
```

相较于原有的 COUNT DISTINCT ，使用子查询 + GROUP BY 的模式性能有 **1/3 的提升**。

未来规划

在过去的三年时间里，Apache Doris 已经在小米内部得到了广泛的应用，支持了集团数据看板、广告投放/广告 BI、新零售、用户行为分析、A/B 实验平台、天星数科、小米有品、用户画像、小米造车等小米内部数十个业务，并且在小米内部形成了一套以 Apache Doris 为核心的数据生态 。随着业务的持续增长，未来我们会进一步推动小米的其他业务上线向量化版本。

非常感谢 Apache Doris 社区与 SelectDB 公司的鼎力支持，小米集团作为 Apache Doris 最早期的用户之一，一直深度参与社区建设，参与 Apache Doris 的稳定性打磨，未来我们也会密切联系社区，为社区贡献更多的力量。

查询性能显著提升，Apache Doris 向量化版本在小米 A/B 实验场景的调优实践

作者：魏祚、赵立伟

导读：长期以来，Apache Doris在小米集团都有着广泛的应用。随着小米互联网业务的快速发展，用户对Apache Doris的查询性能提出了更高的要求，Doris 向量化版本在小米内部上线已经迫在眉睫。在 SelectDB 公司和 Apache Doris 社区的鼎力支持下，我们在小米 A/B 实验场景对 Doris 1.1.2 向量化版本进行了一系列的调优操作，使得查询性能和稳定性有了显著地提升。

业务背景

2019 年 9 月，为了满足小米互联网增长分析业务中近实时、多维分析查询的需求，小米集团首次引入了Apache Doris。在过去的三年时间里，Apache Doris 已经在小米内部得到了广泛的应用，支持了集团数据看板、广告投放、广告BI、新零售、用户行为分析、A/B 实验平台、天星数科、小米有品、用户画像、小米造车等小米内部数十个业务，并且在小米内部形成了一套以 Apache Doris 为核心的数据生态。小米集团作为 Apache Doris 最早期的用户之一，一直深度参与社区建设，参与 Apache Doris 的稳定性打磨。

为了保证线上服务的稳定性，小米内部基于 Apache Doris 社区的 0.13 版本进行迭代，为小米的业务提供稳定的报表分析和 BI 看板服务，经过业务的长时间打磨，内部 Doris 0.13 版本已经非常稳定。但是，随着小米互联网业务的发展，用户对 Doris 的查询性能提出了更高的要求，Doris 0.13 版本在某些场景下逐渐难以满足业务需求了。与此同时，**Apache Doris 社区在快速发展，社区发布的 1.1 版本已经在计算层和存储层全面支持了向量化，查询性能相比非向量化版本有了明显地提升，基于此，小米内部的 Apache Doris 集群进行向量化版本升级势在必行。**

场景介绍

小米的 A/B 实验平台对 Apache Doris 查询性能的提升有着迫切的需求，因此**我们选择优先在小米的 A/B 实验平台上线 Apache Doris 向量化版本，也就是 1.1.2 版本**。

小米的 A/B 实验平台是一款通过 A/B 测试的方式，借助实验分组、流量拆分与科学评估等手段来辅助完成科学的业务决策，最终实现业务增长的一款运营工具产品。在实际业务中，为了验证一个新策略的效果，通常需要准备原策略 A 和新策略 B 两种方案。随后在总体用户中取出一小部分，将这部分用户完全随机地分在两个组中，使两组用户在统计角度无差别。将原策略 A 和新策略 B 分别展示给不同的用户组，一段时间后，结合统计方法分析数据，得到两种策略生效后指标的变化结果，并以此来判断新策略 B 是否符合预期。

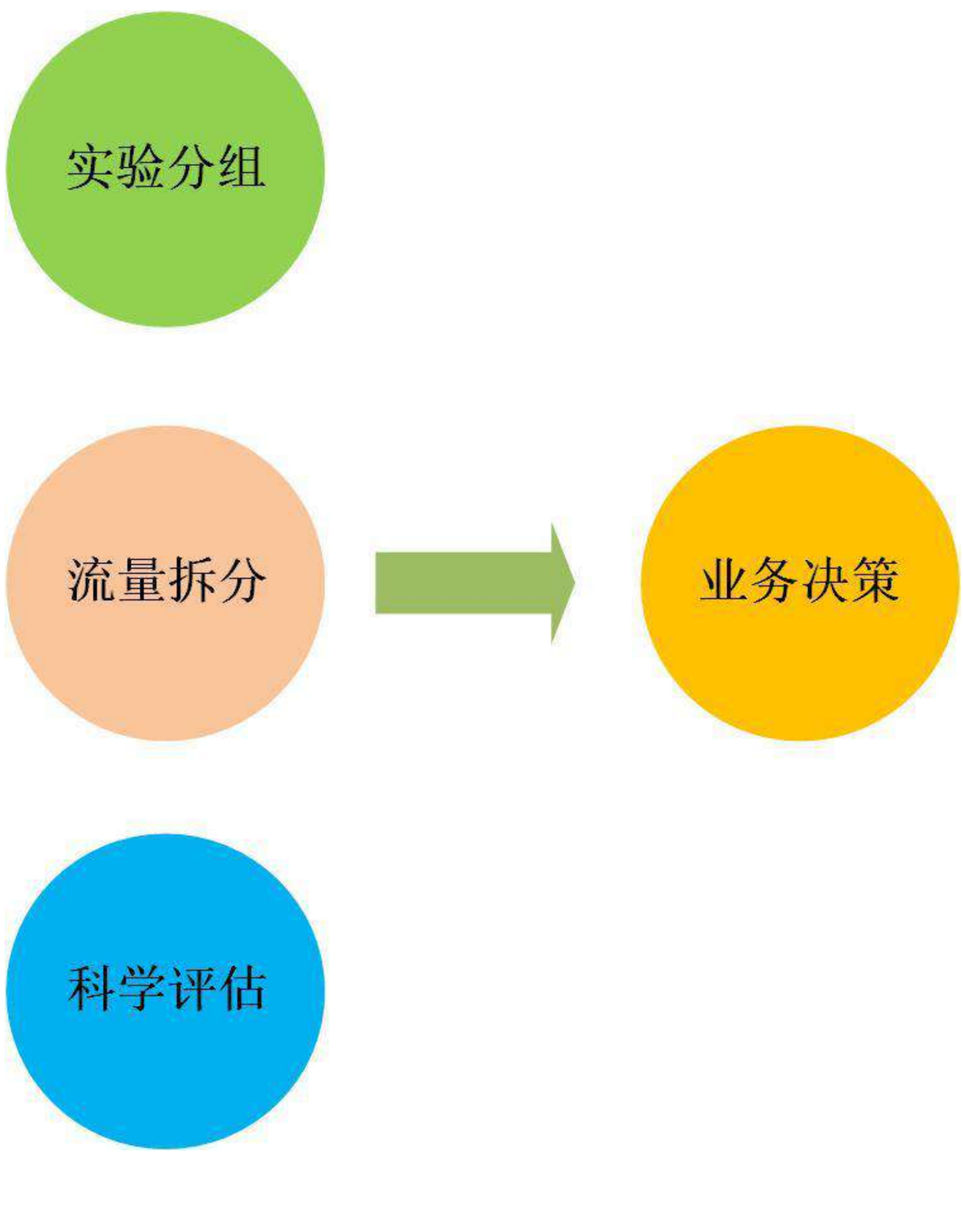


图1–小米的 A/B 实验简介

小米的 A/B 实验平台有几类典型的查询应用：用户去重、指标求和、实验协方差计算等，查询类型会涉及较多的 Count (distinct)、Bitmap 计算、Like 语句等。

上线前验证

我们基于 Apache Doris 1.1.2 版本搭建了一个和小米线上 Apache Doris 0.13 版本在机器配置和机器规模上完全相同的测试集群，用于向量化版本上线前的验证。验证测试分为两个方面：单 SQL 串行查询测试和批量 SQL 并发查询测试。在这两种测试中，我们在保证两个集群数据完全相同的条件下，分别在 Doris 1.1.2 测试集群和小米线上 Doris 0.13 集群执行相同的查询 SQL 来做性能对比。**我们的目标是，Doris 1.1.2 版本在小米线上 Doris 0.13 版本的基础上有 1 倍的查询性能提升。**

两个集群配置完全相同，具体配置信息如下：

- 集群规模：3 FE + 89 BE
- BE 节点CPU: Intel(R) Xeon(R) Silver 4216 CPU @ 2.10GHz 16核 32线程 × 2
- BE 节点内存：256GB
- BE 节点磁盘：7.3TB × 12 HDD

单 SQL 串行查询测试

在该测试场景中，我们选取了小米 A/B 实验场景中 7 个典型的查询 Case，针对每一个查询 Case，我们将扫描的数据时间范围分别限制为 1 天、7 天和 20 天进行查询测试，其中单日分区数据量级大约为 31 亿（数据量大约 2 TB），**测试结果如图所示**：

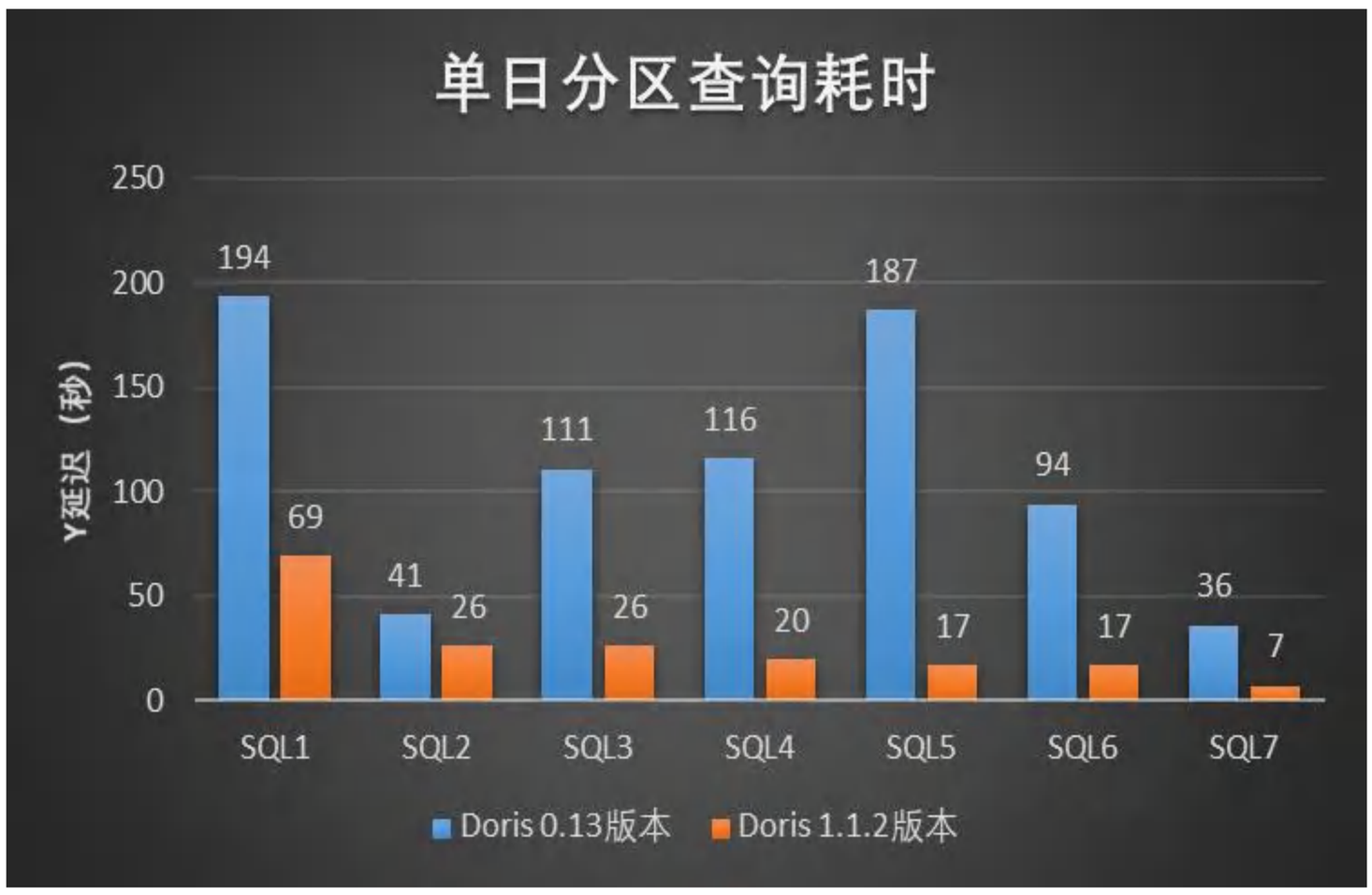


图2–单日分区查询耗时

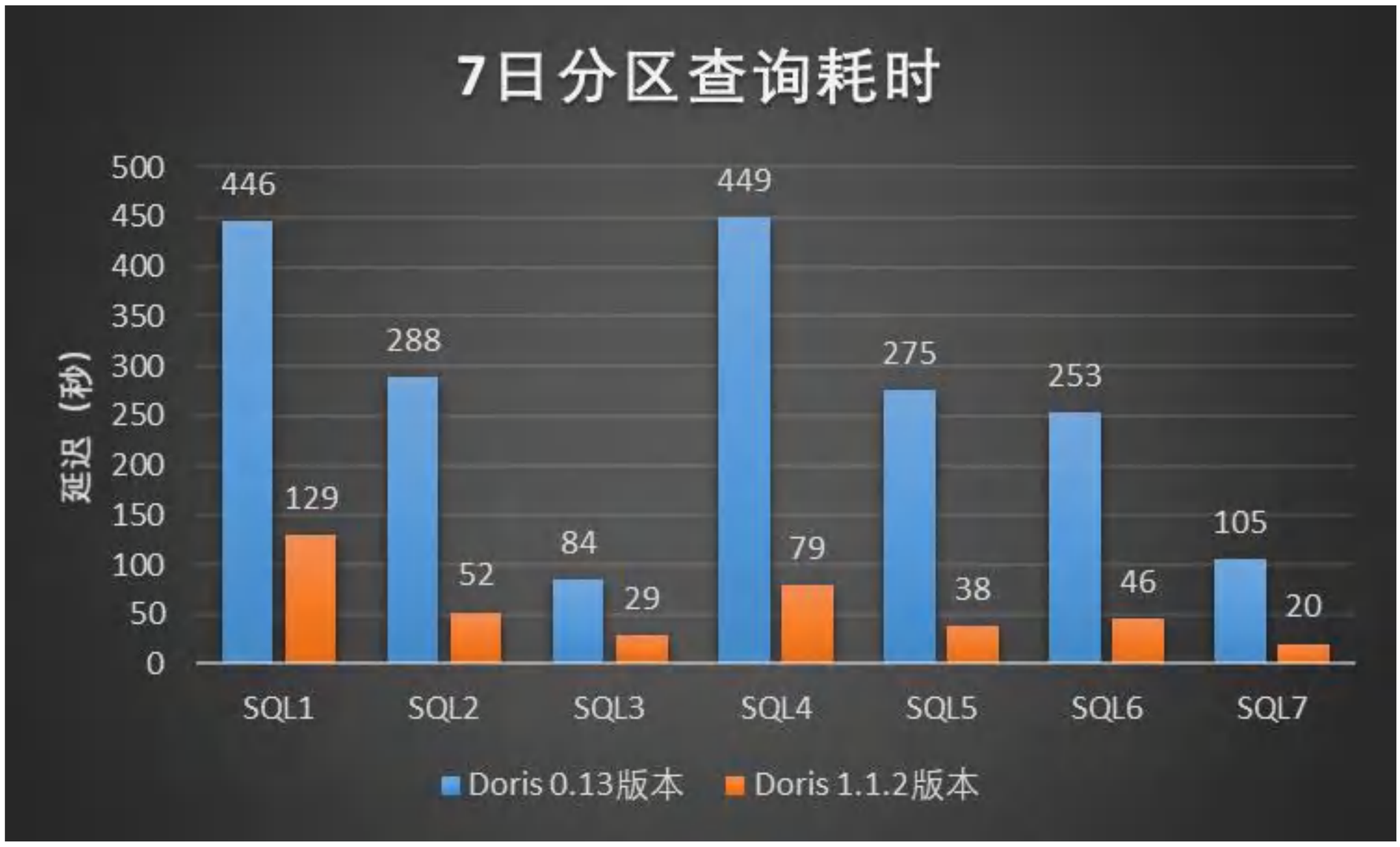


图3–7日分区查询耗时

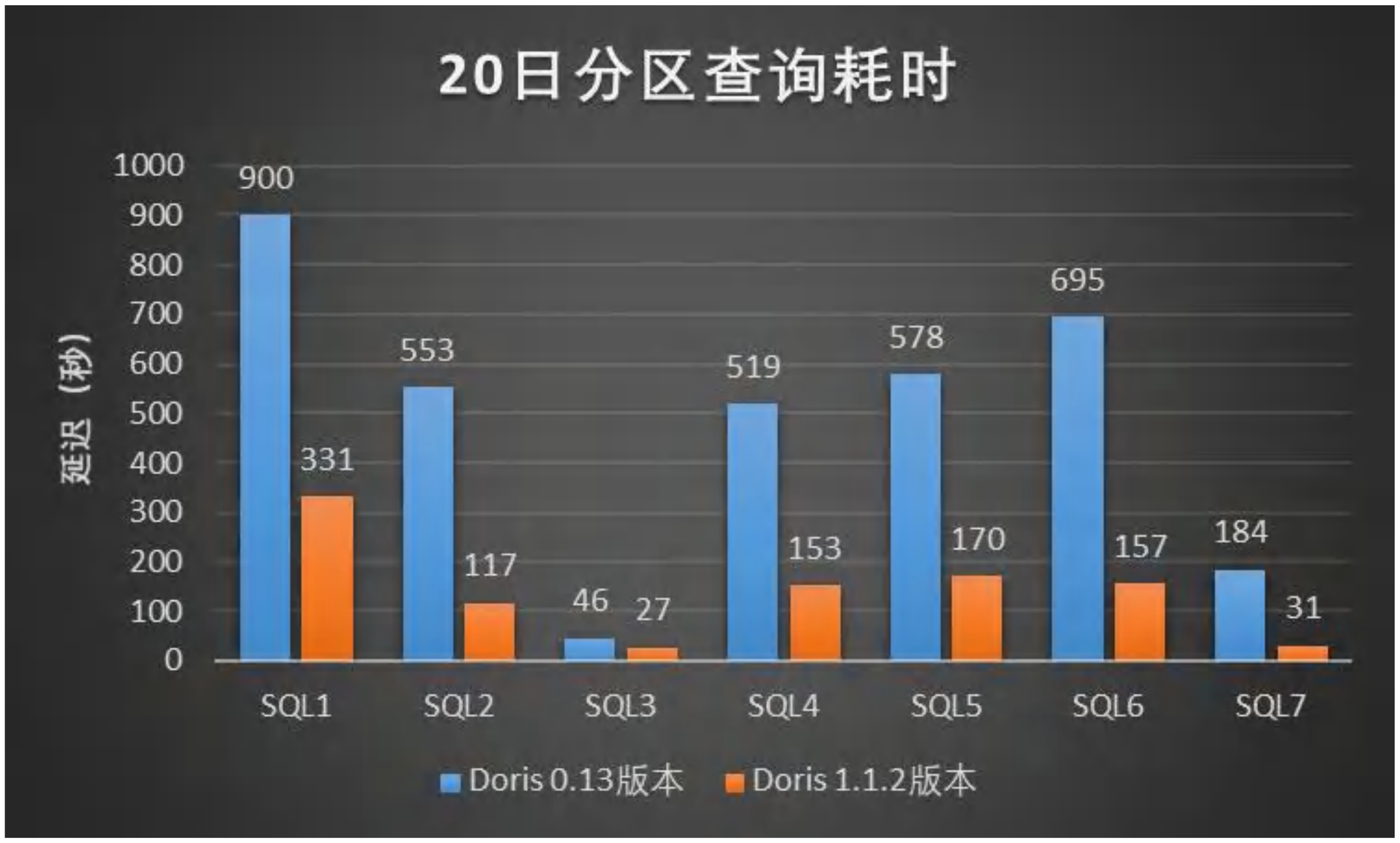


图4–20日分区查询耗时

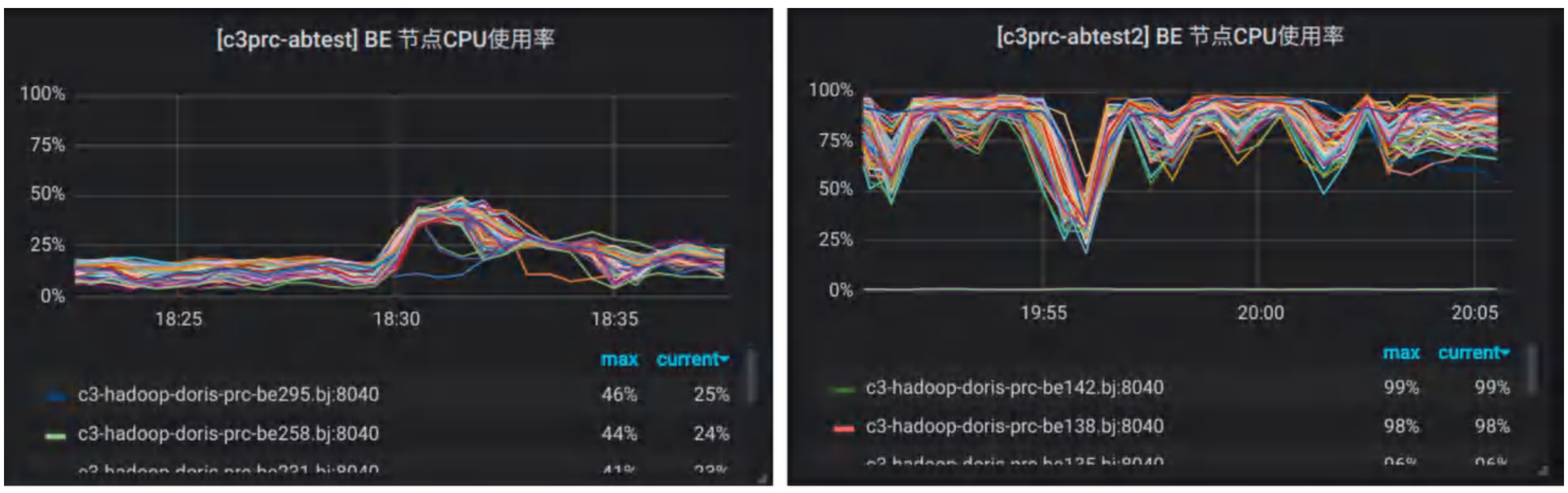
根据以上小米 A/B 实验场景下的单 SQL 串行查询测试结果所示，Doris 1.1.2 版本相比小米线上 Doris 0.13 版本至少有 **3~5 倍的性能提升**，效果显著。

批量 SQL 并发查询测试

在并发测试中，我们将小米 A/B 实验场景的查询 SQL 按照正常的业务并发分别提交到 Doris 1.1.2 测试集群和小米线上 Doris 0.13 集群，对比观察两个集群的状态和查询延迟。测试结果为，在完全相同的机器规模、机器配置和查询场景下，Doris 1.1.2 版本的查询延迟相比线上 Doris 0.13 版本整体上升了 1 倍，查询性能下降非常明显，另外，Doris 1.1.2 版本稳定性方面也存在比较严重的问题，查询过程中会有大量的查询报错。Doris 1.1.2 版本在小米 A/B 实验场景并发查询测试的结果与我们的预期差别较大。**并发查询测试过程中，我们遇到了几个比较严重的问题：**

1. CPU 使用率上不去

查询下发到 Doris 1.1.2 版本所在的集群，CPU 使用率最多只能打到 50% 左右，但是完全相同的一批查询下发到线上 Doris 0.13 版本的集群，CPU使用率可以打到接近 100%。因此推测 **Doris 1.1.2 版本在小米 A/B 实验场景中将机器的 CPU 利用不起来造成查询性能大幅度降低。**



(a)Doris 1.1.2 版本的 CPU 使用率　　(b) Doris 0.13 版本的CPU使用率

图5–Doris 1.1.2 版本和 Doris 0.13 版本 CPU 使用率对比

2. 查询持续报错

用户并发提交查询的时候会出现如下报错，后续的查询任务均无法执行，集群完全处于不可用的状态，只有重启 BE 节点才能恢复。

```
1   RpcException, msg: timeout when waiting for send fragments RPC. Wait(sec): 5, host: 10.142.86.26
```

用户提交查询的时候也会频繁出现如下报错：

```
detailMessage = failed to initialize storage reader. tablet=440712.1030396814.29476aaa20a4795e-b4dbf9ac52ee56be, res=-214, backend=10.118.49.24
```

3. Like 语句查询较慢

在小米 A/B 实验场景有较多的使用 Like 语句进行字符串模糊匹配的查询，在并发测试过程中，该类查询普遍性能较低。

4. 内存拷贝耗时较长

并发查询测试过程中，SQL 整体执行较慢，通过抓取查询过程中的 CPU 火焰图，发现读取字符串类型数据的时候内存拷贝会占用较多时间。



图6–CPU火焰图

调优实践

为了解决 Apache Doris 1.1.2 版本在小米 A/B 实验场景并发测试过程中暴露出的性能和稳定性问题，推动 Doris 向量化版本尽快在小米 A/B 实验平台上线，**我们和 SelectDB 公司以及 Apache Doris 社区一起对 Doris 1.1.2 版本进行了一系列的调优工作**。

提升 CPU 使用率

针对并发查询时 CPU 使用率上不去的问题，我们截取了查询过程中BE进程的函数调用栈，通过分析发现，有较多的内存分配和释放操作在等锁，这可能会造成 CPU 使用率上不去。

函数调用栈

```
#0  sys_futex (v3=0, a2=0x0, t=0x7f786c9e7a00, v=<optimized out>, o=128, a=0x560451827c48 <tcmalloc::Static::pageheap_lock_>) at /root/doris/doris/be/src/gutil/linux_syscall_support.h:2419
#1  SpinLockDelay (loop=1822369984, value=2, w=0x560451827c48 <tcmalloc::Static::pageheap_lock_>) at /root/doris/doris/be/src/gutil/spinlock_linux-inl.h:80
#2  base::internal::SpinLockDelay (w=w@entry=0x560451827c48 <tcmalloc::Static::pageheap_lock_>, value=2, loop=loop@entry=20) at /root/doris/doris/be/src/gutil/spinlock_linux-inl.h:68
#3  0x000056044cfd825d in SpinLock::SlowLock (this=0x560451827c48 <tcmalloc::Static::pageheap_lock_>) at src/base/spinlock.cc:118
#4  0x000056044f013a25 in Lock (this=<optimized out>) at src/base/spinlock.h:69
#5  SpinLockHolder (l=<optimized out>, this=0x7f786c9e7a90) at src/base/spinlock.h:124
#6  (anonymous namespace)::do_malloc_pages(tcmalloc::ThreadCache*, unsigned long) () at src/tcmalloc.cc:1360
...
```

```
#0  sys_futex (v3=0, a2=0x0, t=0x7f7494858b20, v=<optimized out>, o=128, a=0x560451827c48 <tcmalloc::Static::pageheap_lock_>) at /root/doris/doris/be/src/gutil/linux_syscall_support.h:2419
#1  SpinLockDelay (loop=-1803179840, value=2, w=0x560451827c48 <tcmalloc::Static::pageheap_lock_>) at /root/doris/doris/be/src/gutil/spinlock_linux-inl.h:80
#2  base::internal::SpinLockDelay (w=w@entry=0x560451827c48 <tcmalloc::Static::pageheap_lock_>, value=2, loop=loop@entry=2) at /root/doris/doris/be/src/gutil/spinlock_linux-inl.h:68
#3  0x000056044cfd825d in SpinLock::SlowLock (this=0x560451827c48 <tcmalloc::Static::pageheap_lock_>) at src/base/spinlock.cc:118
#4  0x000056044f01480d in Lock (this=<optimized out>) at src/base/spinlock.h:69
#5  SpinLockHolder (l=<optimized out>, this=0x7f7494858bb0) at src/base/spinlock.h:124
#6  (anonymous namespace)::do_free_pages(tcmalloc::Span*, void*) [clone .constprop.0] () at src/tcmalloc.cc:1435
...
```

Doris 内存管理机制

Apache Doris 中使用 TCMalloc 进行内存管理。根据所分配和释放内存的大小，TCMalloc 将内存分配策略分为小内存管理和大内存管理两类。

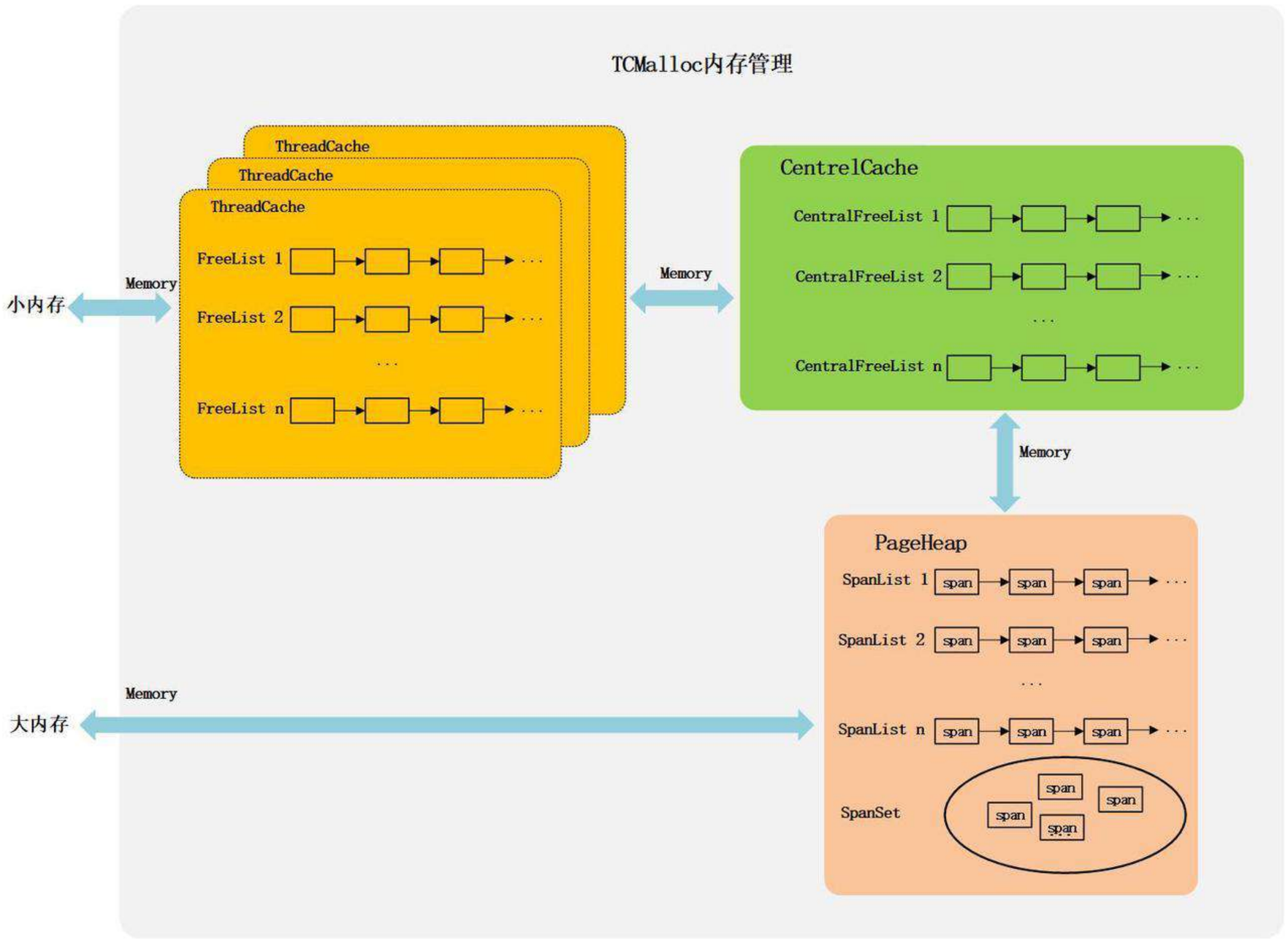


图7–TCMalloc内存管理机制

1）小内存管理

TCMalloc 使用了 ThreadCache、CentralCache 和 PageHeap 三层缓存来管理小内存的分配和释放。

对于每个线程，TCMalloc 都为其单独维护了一个 ThreadCache，每个 ThreadCache 中包含了多个单独的 FreeList，每个 FreeList 中缓存了 N 个固定大小的可供分配的内存单元。进行小内存分配时，会直接从 ThreadCache 中进行内存分配，相应地，小内存的回收也是将空闲内存重新放回 ThreadCache 中对应的 FreeList 中。由于每个线程都有自己独立的 ThreadCache，因此从 ThreadCache 中分配或回收内存是不需要加锁的，可以提升内存管理效率。

内存分配时，如果 ThreadCache 中对应的 FreeList 为空，则需要从 CertralCache 中获取内存来补充自身的 FreeList。CentralCache 中维护了多个 CentralFreeList 链表来缓存不同大小的空闲内存，供各线程的 ThreadCache 取用。由于 CentralCache 是所有线程共用的，因此 ThreadCache 从 CentralCache 中取用或放回内存时是需要加锁的。为了减小锁操作的开销，ThreadCache 一般从 CentralCache 中一次性申请或放回多个空闲内存单元。

当 CentralCache 中对应的 CentralFreeList 为空时，CentralCache 会向 PageHeap 申请一块内存，并将其拆分成一系列小的内存单元，添加到对应的 CentralFreeList 中。PageHeap 用来处理向操作系统申请或释放内存相关的操作，并提供了一层缓存。PageHeap 中的缓存部分会以 Page 为单位、并将不同数量的 Page 组合成不同大小的 Span，分别存储在不同的 SpanList 中，过大的 Span 会存储在一个 SpanSet 中。CentralCache 从 PageHeap 中获取的内存可能来自 PageHeap 的缓存，也可能是来自 PageHeap 向系统申请的新内存。

2）大内存管理

大内存的分配和释放直接通过 PageHeap 来实现，分配的内存可能来自 PageHeap 的缓存，也可能来自 PageHeap 向系统申请的新内存。PageHeap 向系统申请或释放内存时需要加锁。

TCMalloc 中的 `aggressive_memory_decommit` 参数用来配置是否会积极释放内存给操作系统。当设置为 `true` 时，PageHeap 会积极地将空闲内存释放给操作系统，节约系统内存；当该配置设置为 `false` 时，PageHeap 会更多地将空闲内存进行缓存，可以提升内存分配效率，不过会占用更多的系统内存；在 Doris 中该参数默认为 `true`

通过分析查询过程中的调用栈发现，有比较多的线程卡在 PageHeap 向系统申请或释放内存的等锁阶段，因此，我们尝试将 `aggressive_memory_decommit` 参数设为 `false`，让 PageHeap 对空闲内存进行更多的缓存。果然，调整完成之后，CPU 使用率可以打到几乎 100%。在 Doris 1.1.2 版本，数据在内存中采用列式存储，因此，会相比于 Doris 0.13 版本行存的方式有更大的内存管理开销。（社区相关的 PR：https://github.com/apache/doris/pull/12427）



图8–调优后Doris 1.1.2测试集群的CPU使用率

缓解 FE 下发 Fragment 超时的问题

在 Apache Doris 1.1.2 版本，如果一个查询任务的 Fragment 数量超过一个，查询计划就会采用两阶段执行(Two Phase Execution)策略。在第一阶段，FE 会下发所有的 Fragment 到 BE 节点，在 BE 上对 Fragment 执行相应的准备工作，确保 Fragment 已经准备好处理数据；当 Fragment 完成准备工作，线程就会进入休眠状态。在第二阶段，FE 会再次通过 RPC 向 BE 下发执行 Fragment 的指令，BE 收到执行 Fragment 的指令后，会唤醒正在休眠的的线程，正式执行查询计划。

```
1   RpcException, msg: timeout when waiting for send fragments RPC. Wait(sec): 5, host: 10.142.86.26
```

在用户执行查询时，会持续有上面的报错，并导致任何查询无法执行。通过截取进程的调用栈，分析发现大量的线程均在休眠状态，均阻塞在 Fragment 完成准备工作并休眠等待被唤醒的状态。排查发现，查询计划的两阶段执行机制中存在 Bug，如果执行计划被 FE 取消，BE 上已经完成 Fragment 准备工作并休眠等待的线程就不会被唤醒，导致 BE 上的 Fragment 线程池被耗尽，后续所有查询任务的 Fragment 下发到 BE 节点之后，因为没有线程资源都会等待直到 RPC 超时。

为了解决这个问题，我们从社区引入了相关的修复 Patch，为休眠的线程增加了超时唤醒机制，如果线程被超时唤醒，Fragment 会被取消，进而释放线程资源，极大地缓解了 FE 下发执行计划时 RPC 超时的问题。

该问题还未完全解决，当查询并发很大时还会偶发地出现。另外，我们还引入了 Apache Doris 社区相关的其他 Patch 来缓解该问题，比如：减小执行计划的 Thrift Size，以及使用池化的 RPC Stub 替换单一的 RPC Stub。

社区相关的 PR：

> https://github.com/apache/doris/pull/12392
>
> https://github.com/apache/doris/pull/12495
>
> https://github.com/apache/doris/pull/12459

修复 Tablet 元数据汇报的 Bug

在 Apache Doris 中，BE 会周期性地检查当前节点上所有 Tablet 是否存在版本缺失，并向 FE 汇报所有 Tablet 的状态和元信息，由 FE 对每一个 Tablet 的三副本进行对比，确认其中的异常副本，并下发 Clone 任务，通过 Clone 正常副本的数据文件来恢复异常副本缺失的版本。

```
1   detailMessage = failed to initialize storage reader. tablet=440712.1030396814.29476aaa20a4795e-b4dbf
    9ac52ee56be, res=-214, backend=10.118.49.24
```

在该报错信息中，错误代码 `res=-214` （OLAP_ERR_VERSION_NOT_EXIST）表示查询计划执行过程中在 BE 上初始化 Rowset Reader 的时候出现异常，对应的数据版本不存在。在正常情况下，如果 Tablet 的某一个副本存在版本缺失，FE 生成执行计划的时候就不会让查询落在该副本上，然而，查询计划在 BE 上执行的过程中却发现版本不存在，则说明 FE 并没有检测到该副本存在版本缺失。

通过排查代码发现，BE 的 Tablet 汇报机制存在 Bug，当某一个副本存在版本缺失时，BE 并没有将这种情况正常汇报给 FE，导致这些存在版本缺失的异常副本并没有被 FE 检测到，因此不会下发副本修复任务，最终导致查询过程中会发生 `res=-214` 的报错。

社区相关的 PR：

> https://github.com/apache/doris/pull/12415

优化 Like 语句性能

在 Apache Doris 1.1.2 版本中使用 Like 语句进行字符串模糊匹配查询时，Doris 底层其实是使用了标准库中的 `std::search()` 函数对存储层读出的数据进行逐行匹配，过滤掉不满足要求的数据行，完成 Like 语句的模糊匹配。通过调研和对比测试发现，GLIBC 库中的 `std::strstr()` 函数针对字符串匹配比 `std::search()` 函数有 1 倍以上的性能提升。最终我们使用 `std::strstr()` 函数作为 Doris 底层的字符串匹配算法，**将 Doris 底层字符串匹配的性能可以提升 1 倍。**

优化内存拷贝

在小米的场景中有很多字符串类型的查询字段，Apache Doris 1.1.2 版本使用 ColumnString 对象来存储内存中的一列字符串数据，底层使用了 PODArray 结构来实际存储字符串。

执行查询时，需要从存储层逐行读取字符串数据，在这个过程中需要多次对 PODArray 执行 Resize 操作来为列数据申请更大的存储空间，执行 Resize 操作会引起对已经读取的字符串数据执行内存拷贝，而查询过程中的内存拷贝非常耗时，对查询性能影响极大。

为了降低字符串查询过程中内存拷贝的开销，我们需要尽量减少对 PODArray 执行 Resize 操作的次数。鉴于小米 A/B 实验场景中同一列不同行的字符串长度相对比较均匀，我们尝试预先为需要读取的字符串申请足够的内存来减少 Resize 的次数，进而降低内存拷贝的开销。在数据扫描时，每个 Batch 需要读取的数据行数是确定的（假设为 n），当字符串数据读取完指定的前 m（在小米的场景中，该值配置为 100，m < n）行时，我们根据前 m 行的 PODArray 大小预估所有 n 行字符串数据需要的 PODArray 大小，并为其提前申请内存，避免后面逐行读取时多次执行内存申请和内存拷贝。

内存预估公式为：

```
1   所需PODArray总大小 = （当前PODArray总大小 / m）* n
```

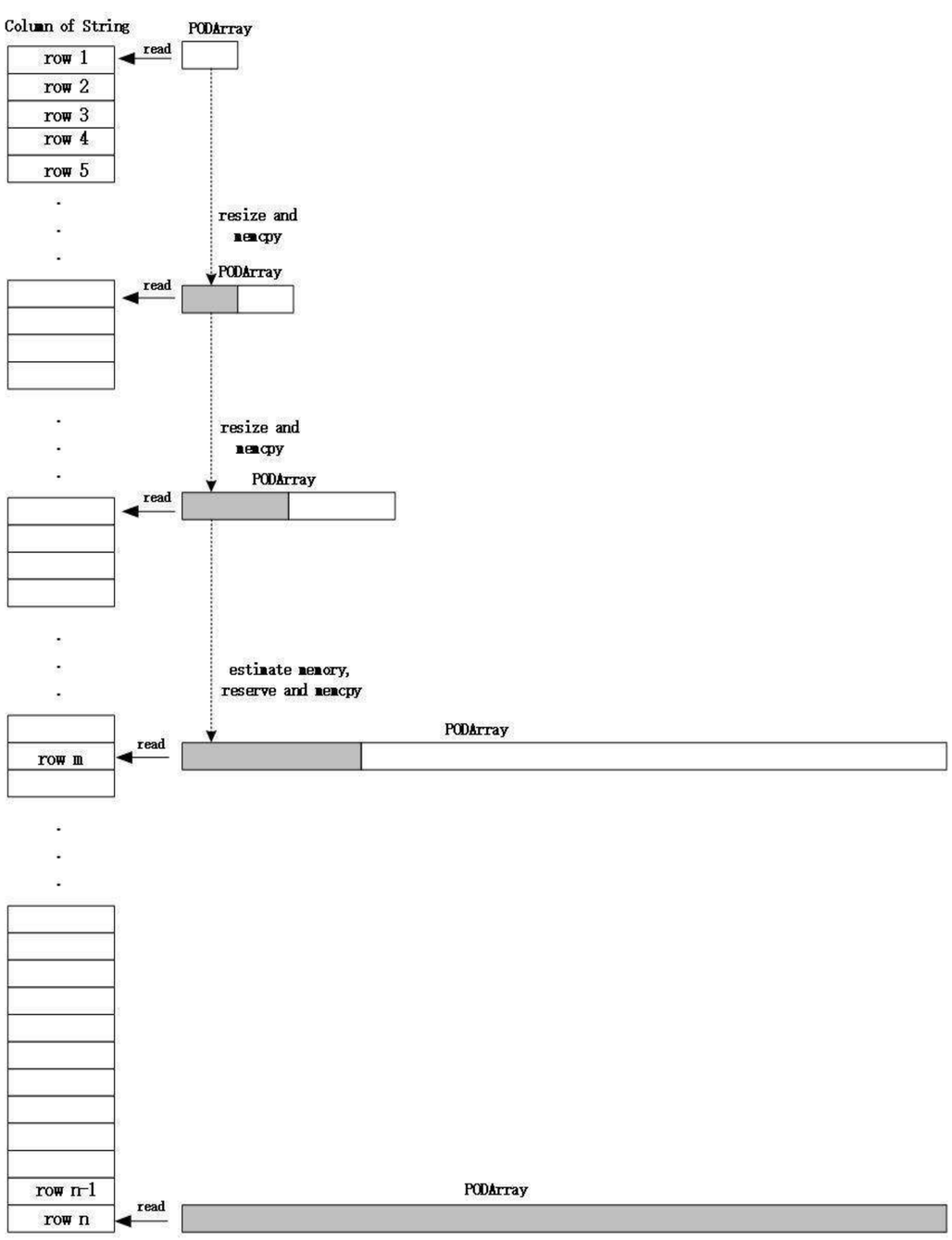


图9–优化内存拷贝开销

当然，该方法只是对所需的内存进行了预估，根据预估的大小提前申请了内存，减少了后面逐行读取字符串时大量的 Resize 操作，减少了内存申请和内存拷贝的次数，并不能完全消除字符串读取过程中的内存拷贝。该优化方案只对一列中字符串长度比较均匀的情况有效，内存的预估相对会比较接近实际内存。如果一列中字符串长度差别较大，该方法的效果可能不甚明显，甚至可能会造成内存浪费。

调优测试结果

我们基于小米的 A/B 实验场景对 Apache Doris 1.1.2 版本进行了一系列调优，并将调优后的 Doris 1.1.2 版本与小米线上 Doris 0.13 版本分别进行了并发查询测试。**测试情况如下：**

测试 1

我们选择了 A/B 实验场景中一批典型的用户去重、指标求和以及协方差计算的查询 Case（SQL 总数量为 3245）对两个版本进行并发查询测试，测试表的单日分区数据大约为 31 亿（数据量大约 2 TB），查询的数据范围会覆盖最近一周的分区。测试结果如图所示，Doris 1.1.2 版本相比 Doris 0.13 版本，总体的平均延迟降低了大约 48%，P95 延迟降低了大约 49%。**在该测试中，Doris 1.1.2 版本相比 Doris 0.13 版本的查询性能提升了接近 1 倍。**

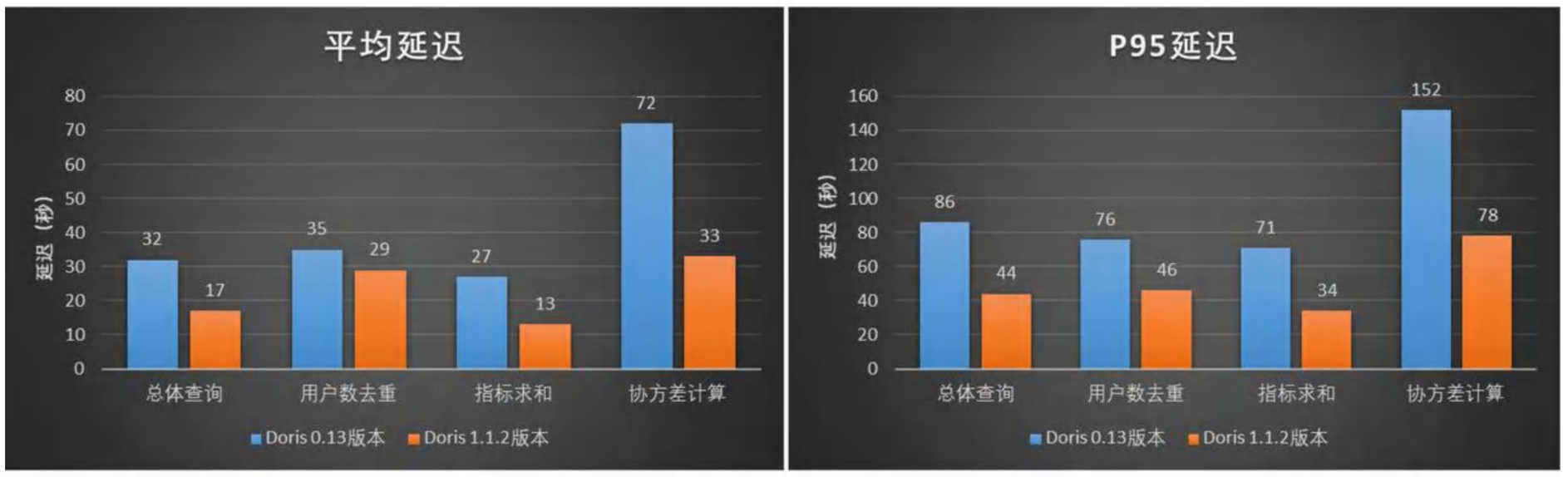


图10–查询平均延迟和P95延迟

测试 2

我们选择了 A/B 实验场景下的 7 份 A/B 实验报告对两个版本进行测试，每份 A/B 实验报告对应小米 A/B 实验平台页面的两个模块，每个模块对应数百或数千条查询 SQL。每一份实验报告都以相同的并发向两个版本所在的集群提交查询任务。测试结果如图所示，Doris 1.1.2 版本相比 Doris 0.13 版本，总体的平均延迟降低了大约 52%。**在该测试中，Doris 1.1.2 版本相比 Doris 0.13 版本的查询性能提升了超过 1 倍。**

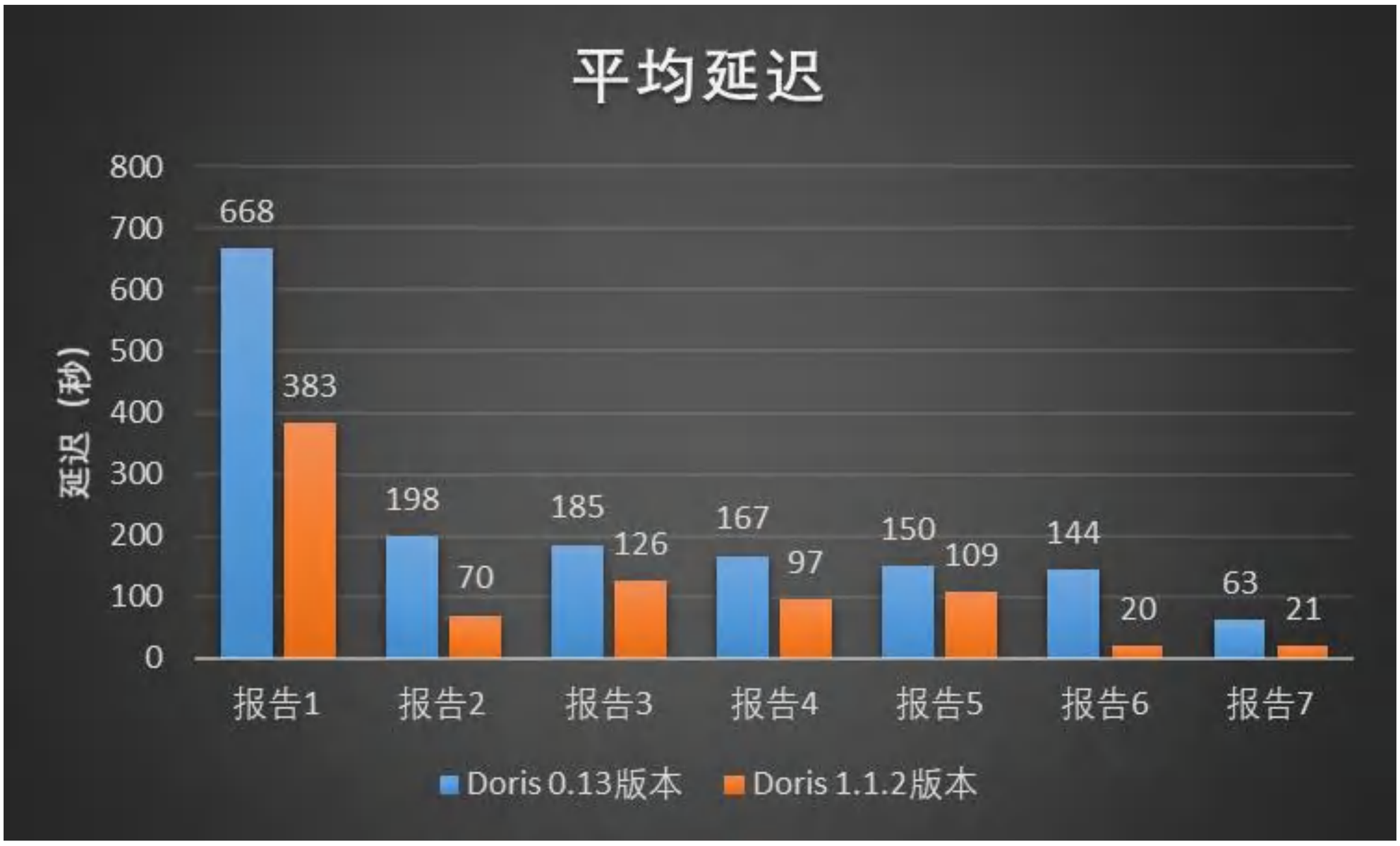


图11–查询平均延迟

测试 3

为了验证调优后的 Apache Doris 1.1.2 版本在小米 A/B 实验场景之外的性能表现，我们选取了小米用户行为分析场景进行了 Doris 1.1.2 版本和 Doris 0.13 版本的并发查询性能测试。我们选取了 2022 年 10 月 24 日、25 日、26 日和 27 日这 4 天的小米线上真实的行为分析查询 Case 进行对比查询，测试结果如图所示，Doris 1.1.2 版本相比 Doris 0.13 版本，总体的平均延迟降低了大约77%，P95 延迟降低了大约 83%。**在该测试中，Doris 1.1.2 版本相比 Doris 0.13 版本的查询性能有 4~6 倍的提升。**

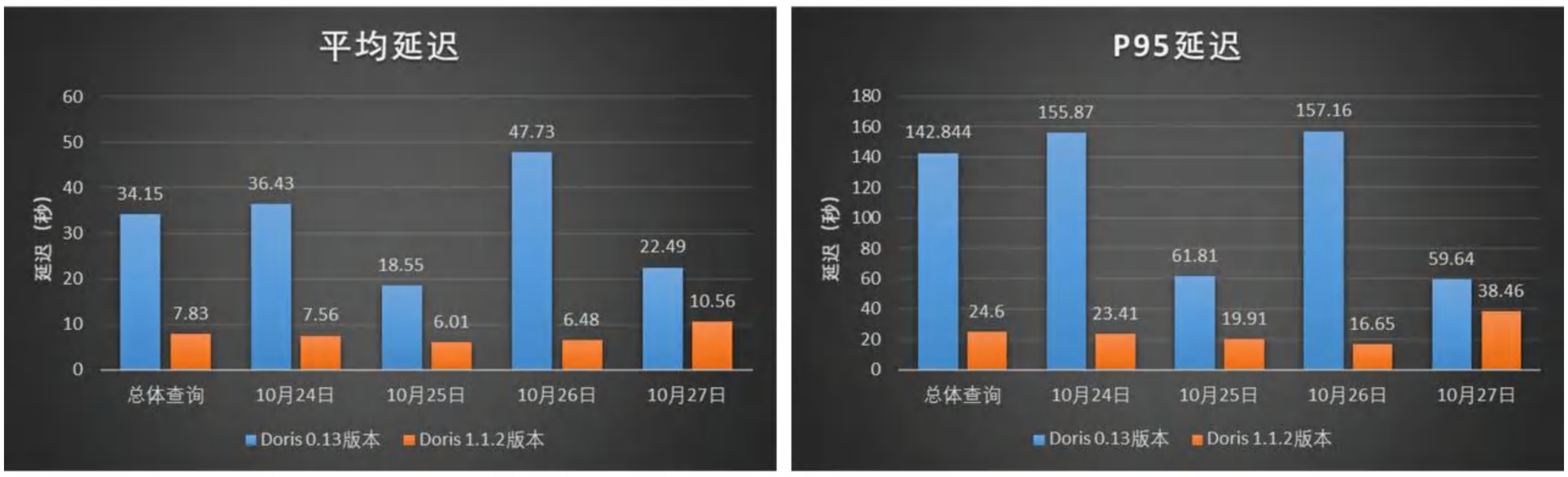


图12–查询平均延迟和P95延迟

结束语

经过一个多月的性能调优和测试，Apache Doris 1.1.2 版本在查询性能和稳定性方面已经达到了小米 A/B实验平台的上线要求，在某些场景下的查询性能甚至超过了我们的预期，希望本次分享可以给有需要的朋友一些可借鉴的经验参考。

最后，感谢 SelectDB 公司和 Apache Doris 社区对我们的鼎力支持，感谢衣国垒老师在我们版本调优和测试过程中的全程参与和陪伴。Apache Doris 目前已经在小米集团内部得到了广泛地应用，并且业务还再持续增长，未来一段时间我们将逐步推动小米内部其他的 Apache Doris 业务上线向量化版本。

小米 A/B 实验场景

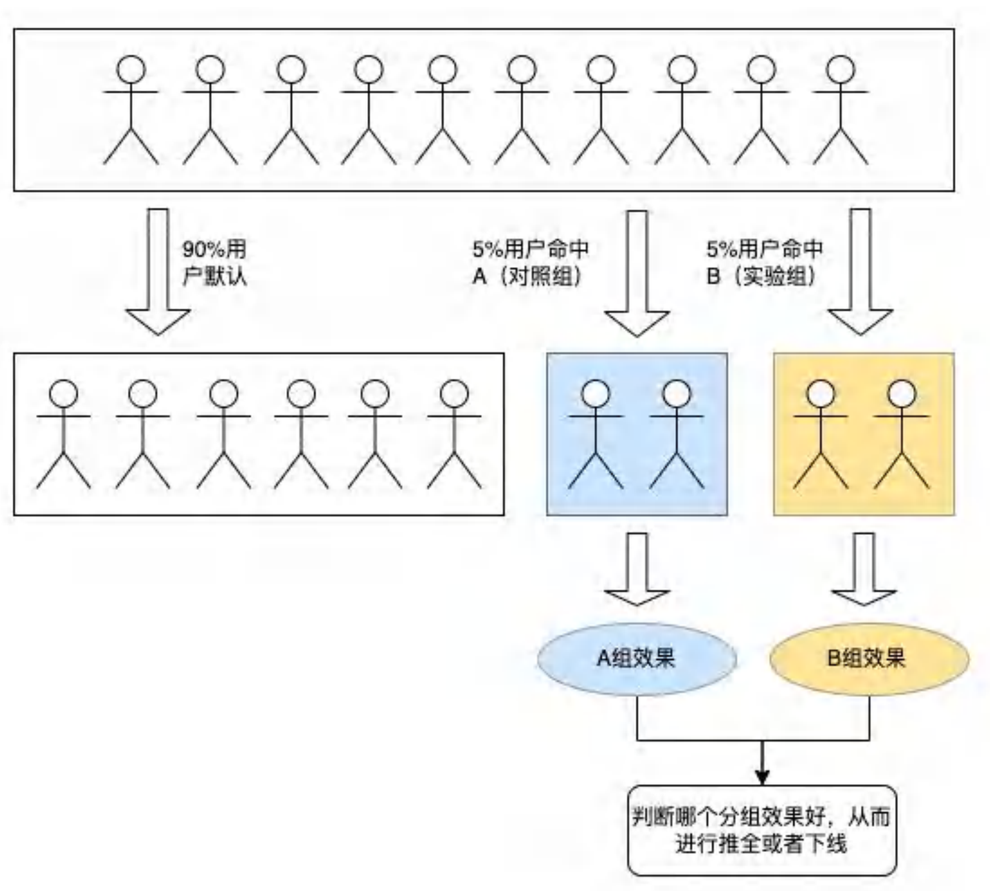
基于 Apache Doris 的查询提速优化实践

作者：小米集团大数据工程师，乐涛

导读：Apache Doris 是小米集团内部应用最为广泛的 OLAP 引擎之一，本文主要从数据的角度分析 A/B 实验场景查询的性能现状，探讨基于 Apache Doris 的性能优化的解决方案。经过一系列基于 Doris 的性能优化和测试，A/B 实验场景查询性能的提升超过了我们的预期。希望本次分享可以给有需要的朋友提供一些参考。

业务背景

A/B 实验是互联网场景中对比策略优劣的重要手段。为了验证一个新策略的效果，需要准备原策略 A 和新策略 B 两种方案。随后在总体用户中取出一小部分，将这部分用户完全随机地分在两个组中，使两组用户在统计角度无差别。将原策略 A 和新策略 B 分别展示给不同的用户组，一段时间后，结合统计方法分析数据，得到两种策略生效后指标的变化结果，并以此判断新策略 B 是否符合预期。

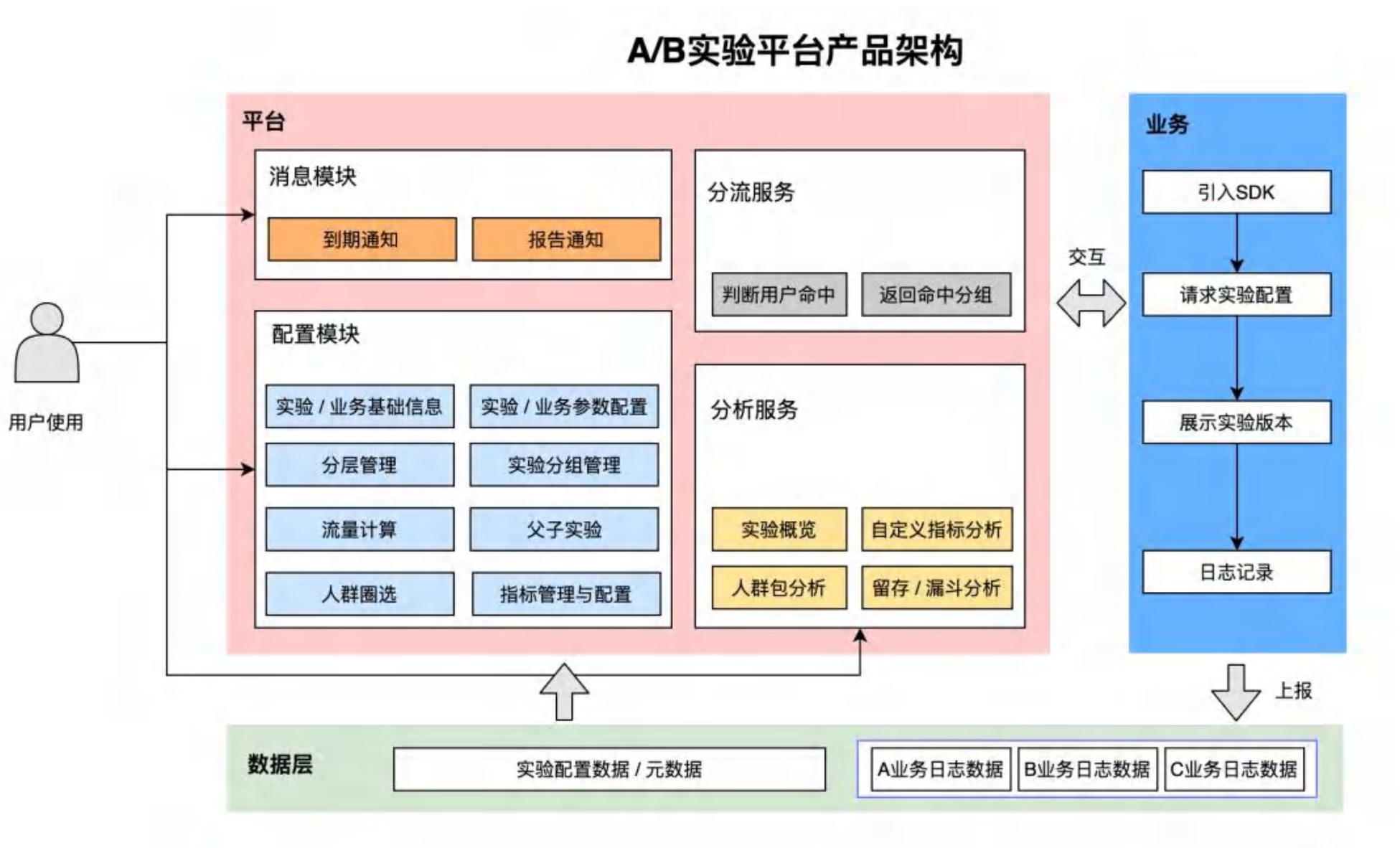


小米 A/B 实验平台是一款通过 A/B 实验的方式，借助实验分组、流量拆分与科学评估来辅助完成科学的业务决策，最终实现业务增长的一款运营工具产品。其广泛的应用于产品研发生命周期中各个环节：

角色	场景
产品	产品功能迭代、用户路径设计、UI&交互优化
运营	运营活动优化：落地页、素材...推送方案择优：推送时机、文案...
研发	推荐算法优化、排序算法优化...

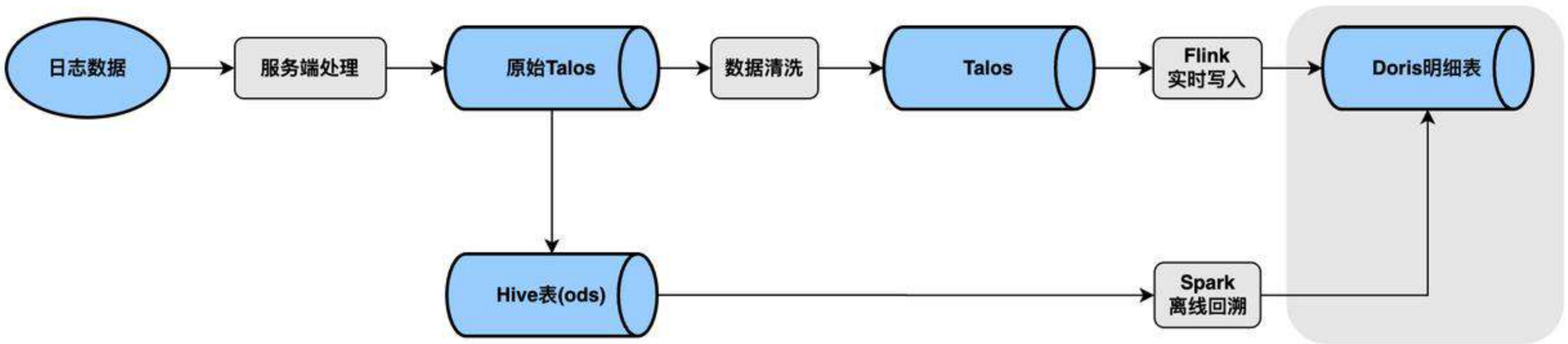
数据平台架构

本文主要从数据的角度分析 A/B 实验场景查询的性能现状，探讨一下基于 Apache Doris 的性能优化的解决方案。A/B 实验平台的架构如下图所示：



- 平台使用的数据主要包含平台自用的实验配置数据、元数据，以及业务方上报的日志数据。
- 由于业务方引入 SDK，并与分流服务进行交互，日志数据中包含其参与的实验组 ID 信息。
- 用户在实验平台上配置、分析、查询，以获得报告结论满足业务诉求。

鉴于 AB 实验报告各个业务方上报数据的链路都大体类似，我们就拿**头部业务方广告业务**举例，数据流程如下图所示：



从上图可知，整个数据链路并不复杂，日志数据传入后，经过必要的数据处理和清洗工作进入 Talos（小米自研消息队列），通过 Flink 任务以明细数据的形式实时写入到 Doris 表中，同时 Talos 数据也会同步到 Hive 表进行备份，以便问题排查和数据修复。

出于对高效写入以及字段增减需求的考虑，**Doris 明细表以 Duplicate 模型来建模：**

```
CREATE TABLE `dwd_xxxxxx` (
  `olap_date` int(11) NULL COMMENT "分区日期",
  `user_id` varchar(256) NULL COMMENT "用户id",
  `exp_id` varchar(512) NULL COMMENT "实验组ID",
  `dimension1` varchar(256) NULL COMMENT "",
  `dimension2` varchar(256) NULL COMMENT "",
  ......
  `dimensionN` bigint(20) NULL COMMENT "",
  `index1` decimal(20, 3) NULL COMMENT "",
  ......
  `indexN` int(11) NULL COMMENT "",

) ENGINE=OLAP
DUPLICATE KEY(`olap_date`, `user_id`)
COMMENT "OLAP"
PARTITION BY RANGE(`olap_date`)
(
PARTITION p20221101 VALUES [("20221101"), ("20221102")),
PARTITION p20221102 VALUES [("20221102"), ("20221103")),
PARTITION p20221103 VALUES [("20221103"), ("20221104"))
)
DISTRIBUTED BY HASH(`user_id`) BUCKETS 300
;
```

数据现状分析

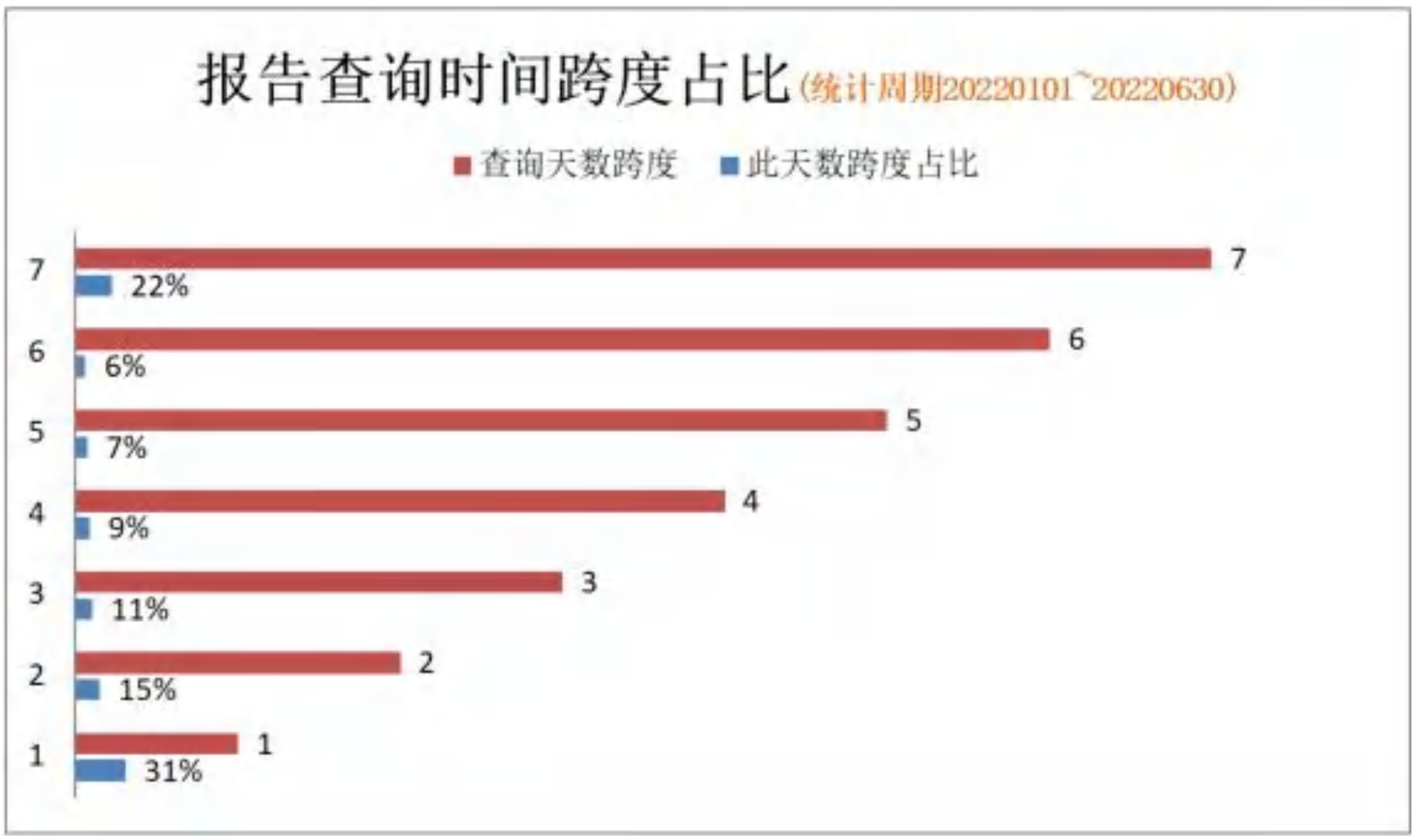
在提速之前，小米 A/B 实验平台完成实验报告查询的 P95 时间为小时级，实验报告使用数据的方式存在诸多的性能问题，直接影响业务部门做运营和决策的效率。

报告查询基于明细

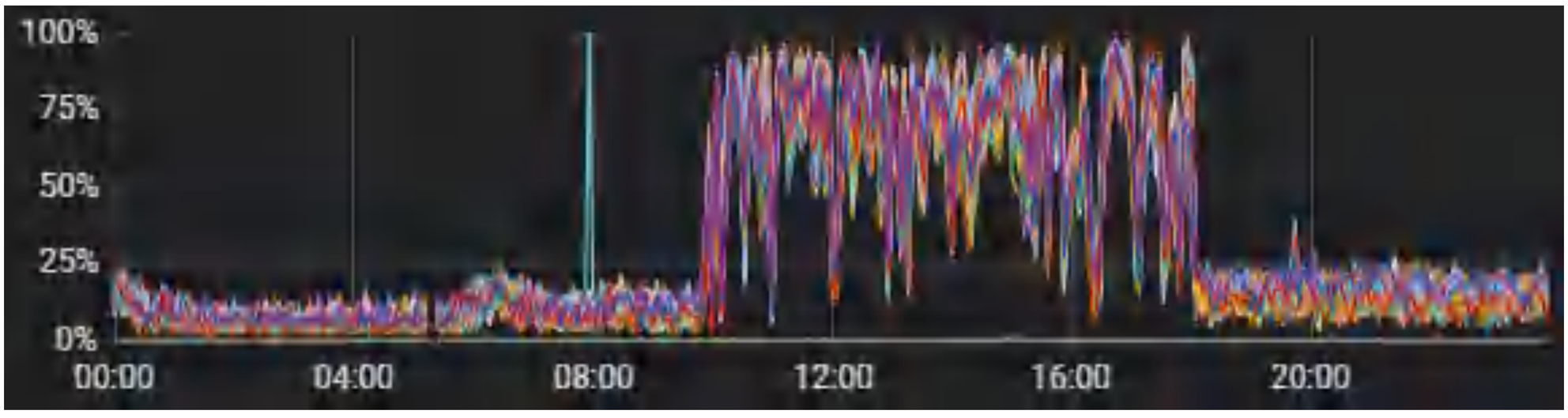
当前报告查询的数据来源为明细表，而明细表的数据量巨大：

单日记录数量级	单日存储量级(Doris replication_num=3)	字段数量
十亿级	TB级别	百余个

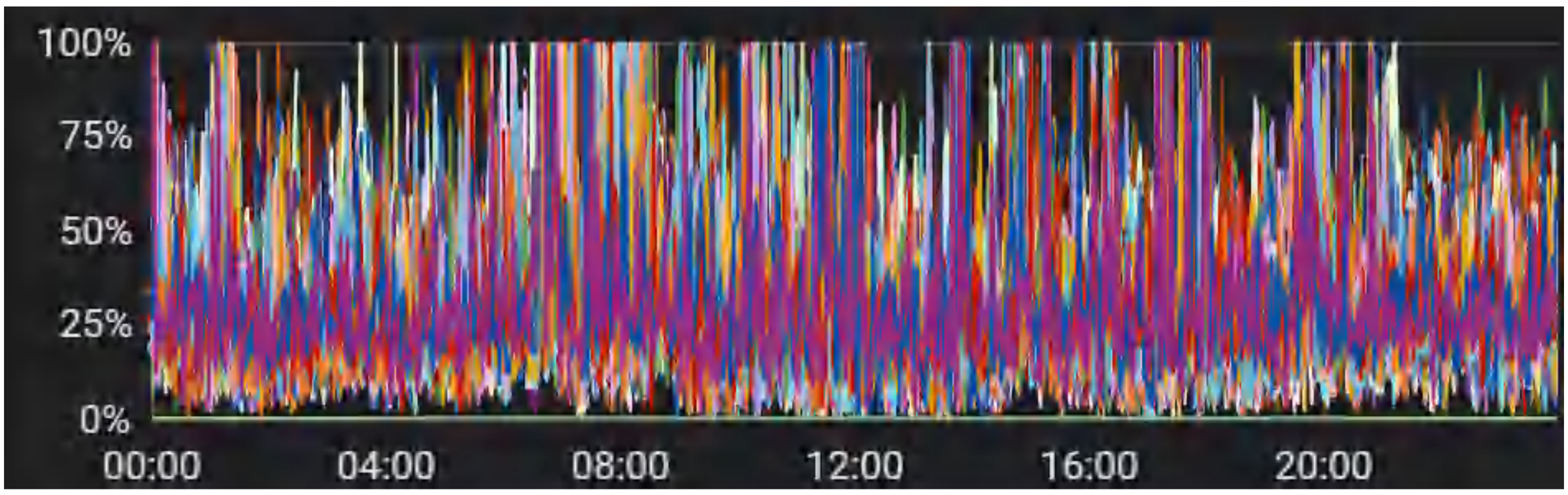
而且，实验报告的查询条件中时间范围常常横跨多天。基于历史查询报告统计，查询条件中时间范围大于一天的报告占比 69.1%，具体的时间跨度占比分布如下：



明细数据的巨大扫描量给集群带来了不小的压力，且由于报告查询存在并发以及 SQL 的拆分，如果一个 SQL 请求不能快速的返回结果释放资源，也会影响到请求的排队状况。因此在工作时间段内 Doris 集群BE节点 CPU 负载状况基本是持续满载，磁盘 IO 也持续处于高负荷状态，如下图所示：



BE节点CPU使用率



BE节点磁盘IO

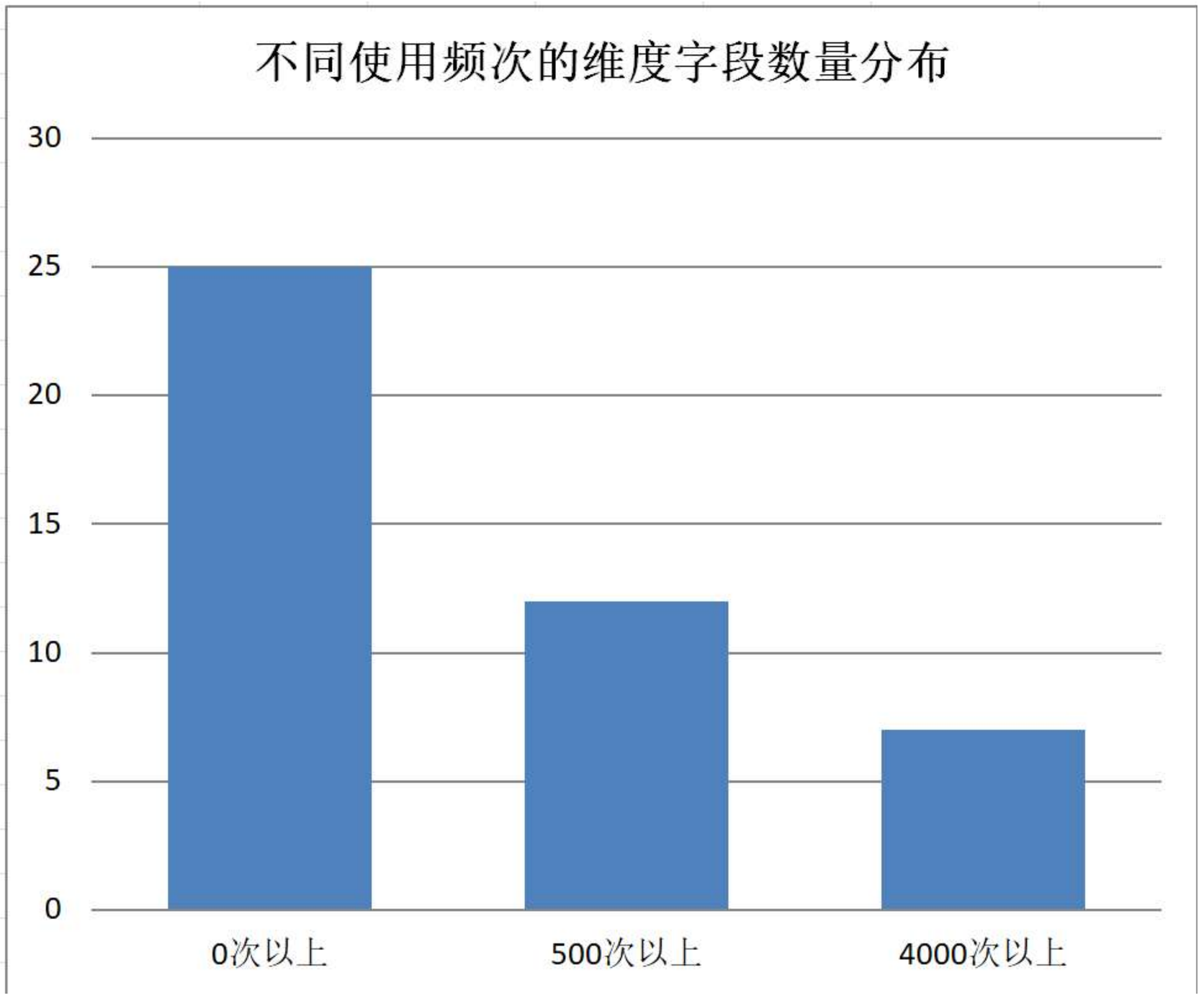
个人思考：

- 当前报告所有查询基于明细数据，且平均查询时间跨度为 4 天，查询扫描数据量上百亿。由于扫描数据量级大，计算成本高，给集群造成较大压力，导致数据查询效率不高。
- 如果通过对数据进行预聚合处理，控制 Scan Rows 和 Scan Bytes，减小集群的压力，查询性能会大幅提升。

字段查询热度分层分布

由于之前流程管控机制相对宽松，用户添加的埋点字段都会进入明细表中，导致字段冗余较多。统计历史查询报告发现，明细表中常用的维度和指标只集中在部分字段，且查询热度分层分布：

维度字段名	使用次数
exp_id	4000+
维度1	
维度2	
维度3	
维度4	
维度5	
维度6	
维度7	1000-4000
维度8	
维度9	
维度10	
维度11	
维度12	小于1000
维度13	
其余11个维度字段	



参与计算的指标也集中在部分字段，且大部分都是聚合计算(sum)或可以转化为聚合计算(avg)：

字段名	算子	使用次数
指标1	sum	1000+
指标2	sum	
指标3	sum	
指标4	sum	
指标5	avg(sum/cnt)	
指标6	sum	
指标7	sum	
指标8	avg(sum/cnt)	
指标9	sum	
指标10	sum	
指标11	sum	
指标12	avg(sum/cnt)	
指标13	uniq_user_avg(sum/count(distinct user_id))	

个人思考：

- 明细表中参与使用的维度只占 54.3%，高频使用的维度只占 15.2%，维度查询频次分层分布。
- 数据聚合需要对明细表中维度字段做取舍，选择部分维度进行上卷从而达到合并的目的，但是舍弃部分字段必然会影响聚合数据对查询请求的覆盖情况。然而维度查询频次分层分布的场景非常适合根据维度字段的热度做不同层次的数据聚合，同时兼顾聚合表的聚合程度和覆盖率。

实验组 ID 匹配效率低

当前明细数据的格式为：

user_id	exp_id
c7[redacted]752	208967,208987,114448,211555,172297,228698,6759,274327,1242,249580,298592,279713,286331
61[redacted]22d	208967,211556,206987,228698,172297,147513,6759,274327,1242,249580,298592,279713,286331
eb[redacted]146	297205,208967,303442,211557,228698,277446,187758,297986,203989,6759,1242,286331
53[redacted]5f4	208990,211557,187758,172297,228699,1241,6759,274327,249580,208966,298592,279713,286331
53[redacted]5f4	208990,211557,187758,172297,228699,1241,6759,274327,249580,208966,298592,279713,286331
95[redacted]90b	208967,211556,206987,241596,228698,277449,187758,1241,6759,298592,181617,286331
95[redacted]90b	208967,211556,206987,241596,228698,277449,187758,1241,6759,298592,181617,286331
95[redacted]90b	208967,211556,206987,241596,228698,277449,187758,1241,6759,298592,181617,286331
ff[redacted]a2b	297205,281083,211555,208988,147513,298423,124966,228699,6759,1242,206966,263383,297990,286331
ff[redacted]a2b	297205,281083,211555,208988,147513,298423,124966,228699,6759,1242,206966,263383,297990,286331

明细数据中的实验组 ID 以逗号分隔的字符串形式聚拢在一个字段中，而实验报告的每条查询语句都会使用到 `exp_id` 过滤，查询数据时使用 LIKE 方式匹配，查询效率低下。

个人思考：

- 将实验组 ID 建模成一个单独的维度，可使用完全匹配代替 LIKE 查询，且可利用到 Doris 索引，提高数据查询效率。
- 将逗号分隔的实验组 ID 直接打平会引起数据量的急剧膨胀，因此需要设计合理的方案，同时兼顾到数据量和查询效率。

进组人数计算有待改进

进组人数查询是实验报告的必查指标，因此其查询速度很大程度上影响实验报告的整体查询效率，当前主要问题如下：

- 当进组人数作为独立指标计算时，使用近似计算函数 `APPROX_COUNT_DISTINCT` 处理，是通过牺牲准确性的方式提升查询效率。
- 当进组人数作为复合指标的分母进行计算时，使用 `COUNT DISTINCT` 处理，此方式在大数据量计算场景效率较低。

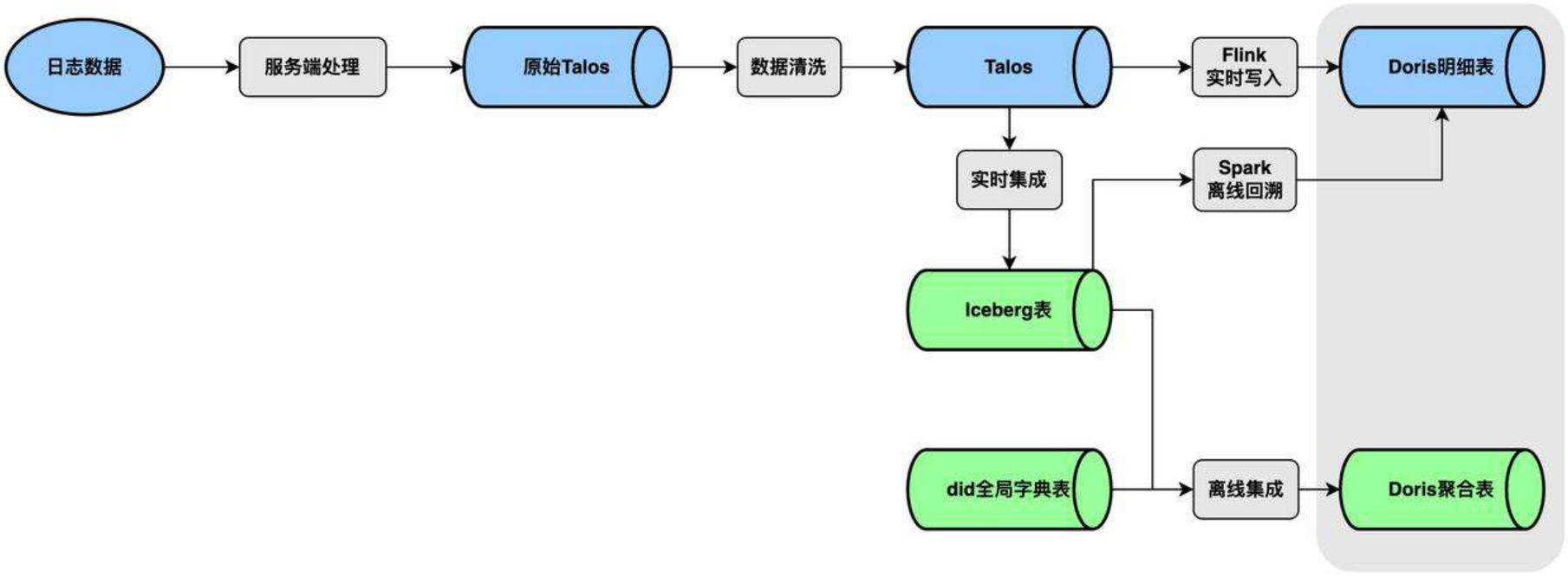
个人思考：

- AB 实验报告的数据结论会影响到用户决策，牺牲准确性的方式提升查询效率是不可取的，特别是广告这类涉及金钱和业绩的业务场合，用户不可能接受近似结果。
- 进组人数使用的 `COUNT DISTINCT` 计算需要依赖明细信息，这也是之前查询基于明细数据的重要因素。必须为此类场景设计新的方案，使进组人数的计算在保证数据准确的前提下提高效率。

数据优化方案

基于以上的数据现状，我们优化的核心点是将明细数据预聚合处理，通过压缩数据来控制 Doris 查询的 Scan Rows 和 Scan Bytes。与此同时，使聚合数据尽可能多的覆盖报告查询。从而达到，减小集群的压力，提高查询效率的目的。

新的数据流程如下图所示：

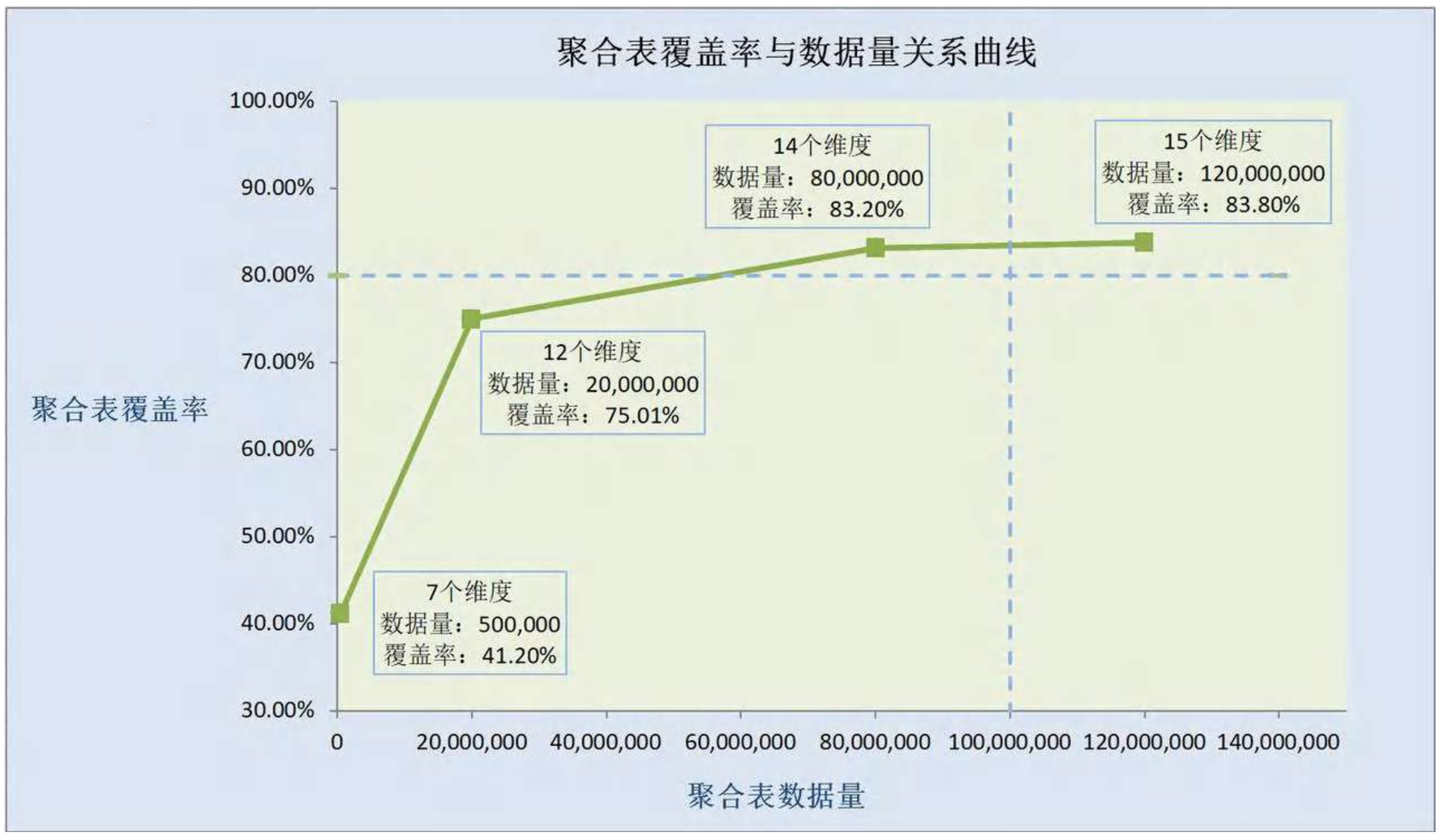


整个流程在明细链路的基础上增加聚合链路，Talos 数据一方面写入 Doris 明细表，另一方面增量落盘到 Iceberg 表中，Iceberg 表同时用作回溯明细数据以及生成聚合数据，我们通过工场 Alpha（小米自研数据开发平台）的实时集成和离线集成保证任务的稳定运行和数据的一致性。

选取高频使用维度聚合

在生成数据聚合的过程中，聚合程度与请求覆盖率是负相关的。使用的维度越少，能覆盖的请求就越少，但数据聚合程度越高；使用的维度越多，覆盖的请求也越多，但数据粒度就越细，聚合程度也越低。因此需要在聚合表建模的过程中取得一个平衡。

我们的具体做法是：拉取历史（近半年）查询日志进行分析，根据维度字段的使用频次排序确认进入聚合表的优先级。在此基础上得出聚合表的覆盖率和数据量随着建模字段增加而变化的曲线，如下图所示：



其中覆盖率根据历史请求日志代入聚合表计算得出。

我们的原则是：针对 OLAP 查询，聚合表的数据量应尽可能的控制在单日 **1 亿条**以内，请求覆盖率尽可能达到 **80%以上**。

因此不难得出结论：选择 14 个维度字段对聚合表建模比较理想，数据量能控制到单日 **8 千万**条左右，且请求覆盖率约为 **83%**。

使用物化视图

在分析报告历史查询日志时，我们发现不同的维度字段查询频次有明显的分层：

聚合表维度	使用此维度的报告占比
exp_id	100%
维度1	100%
维度2	99.29%
维度3	99.29%
维度4	99.12%
维度5	98.98%
维度6	98.67%
维度7	35.63%
维度8	35.63%
维度9	26.78%
维度10	23.66%
维度11	15.23%
维度12	2.31%
维度13	2.19%

Top7 维度字段几乎出现在所有报告的查询条件之中，对于如此高频的查询，值得做进一步的投入，使查询效率尽可能的提升到最佳。**Doris 的物化视图能够很好的服务于此类场景。**

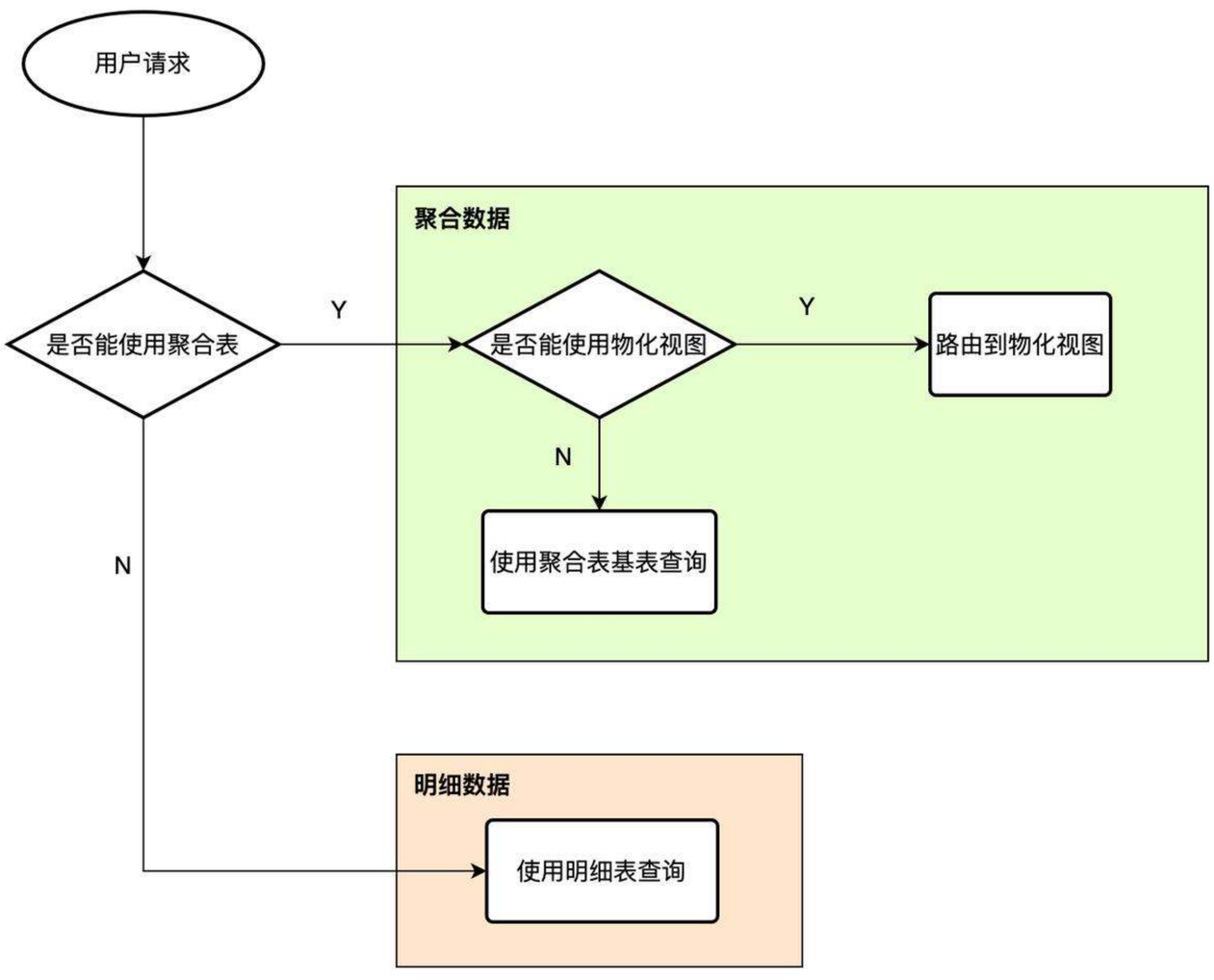
什么是物化视图？

物化视图是一种特殊的物理表，其中保存基于基表 (base table) 的部分字段进一步上卷聚合的结果。

虽然在物理上独立存储，但它是对用户透明的。为一张基表配置好物化视图之后，不需要为其写入和查询做任何额外的工作：

- 当向基表写入和更新数据时，集群会自动地同步到物化视图，并通过事务方式保证数据的一致性。
- 当对基表进行查询时，集群会自动判断是否路由到物化视图获取结果。当查询字段能被物化视图完全覆盖时，会优先使用物化视图。

因此我们的查询路由如下图所示：



用户的查询请求会尽可能的路由到聚合表物化视图，然后是聚合表基表，最后才是明细表。

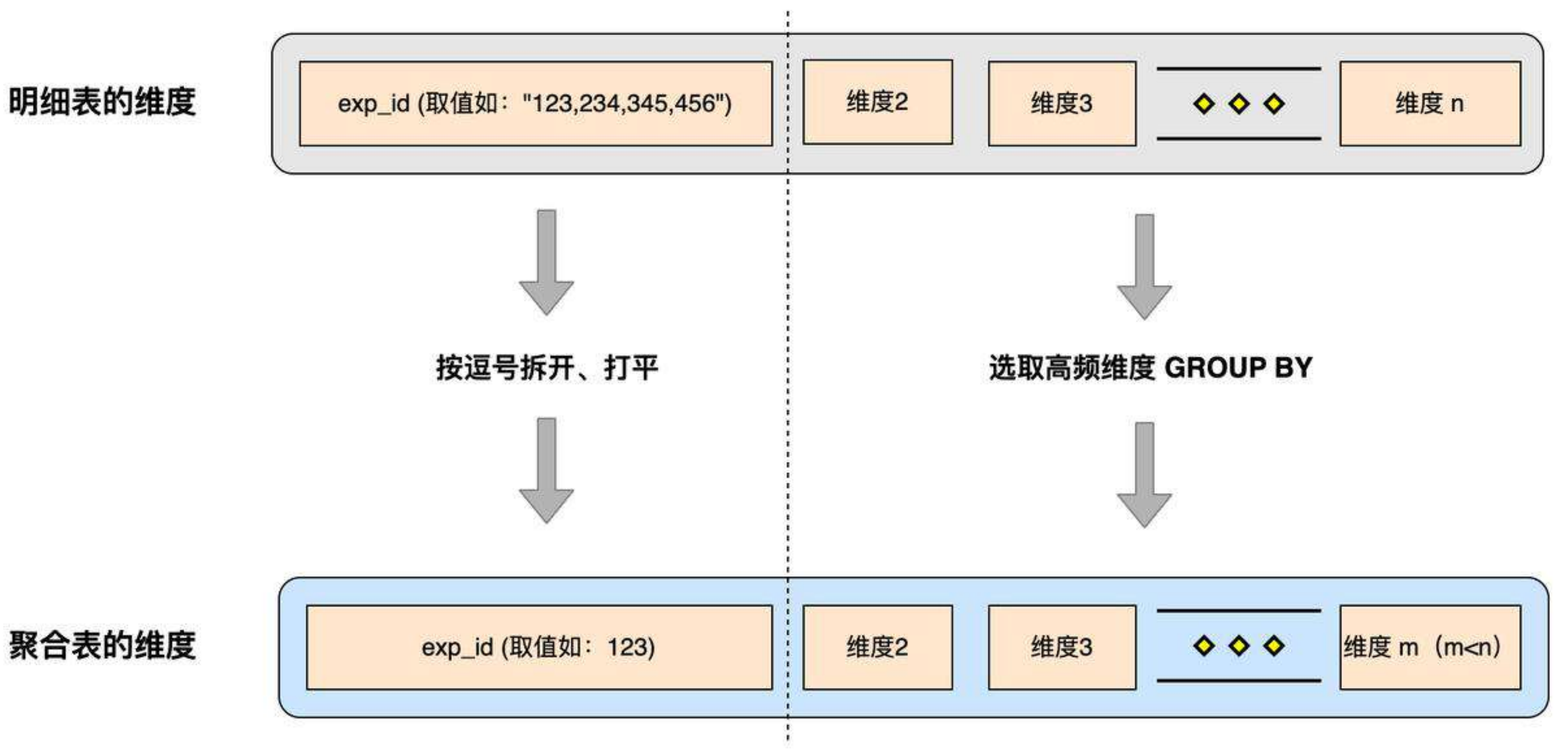
如此使用多梯度的聚合模型的配合来应对热度分层的查询请求，使聚合数据的效能尽可能的发挥到最大。

精确匹配取代 LIKE 查询

既然物化视图这么好用，为什么我们不是基于 Doris 明细表配置物化视图，而是单独开发聚合表呢？是因为明细数据中的实验组 ID 字段存储和查询方式并不合理，聚合数据并不适合通过明细数据直接上卷来得到。

上文中已经提到，`exp_id`（实验组 ID）在明细表中以逗号分隔的字符串进行存储，查询数据时使用 LIKE 方式匹配。作为 AB 实验报告查询的**必查条件**，这种查询方式无疑是**低效的**。

我们希望的聚合方式如下图所示：



我们需要将 `exp_id` 字段拆开，把数据打平，使用精确匹配取代 LIKE 查询，提高查询的效率。

控制聚合表数据量

如果只做拆分打平的处理必然会导致数据量的激增，未必能达到正向优化的效果，因此我们还需要想办法来压缩 `exp_id` 打平后的数据量：

- 聚合表选取维度字段建模的时候，除了上文提到的，以字段的使用频次热度作为依据之外，也要关注字段的取值基数，进行综合取舍。如果取值基数过高的维度字段进入聚合表，必然会对控制聚合表的数据量造成阻碍。因此，我们在保证聚合表请求覆盖量的前提下，酌情舍弃部分高基数（取值有十万种以上）的维度。
- 从业务的角度尽可能过滤无效数据（比如一个实验组的流量为 0% 或者 100%，业务上就没有对照的意义，用户也不会去查，这样的数据就不需要进入聚合表）。

经过这一系列步骤，最终聚合表的数据量被控制在单日约 8000 万条，并没有因为 `exp_id` 打平而膨胀。

值得一提的是，`exp_id` 字段拆分后，除了查询从LIKE匹配变为精确匹配，还额外带来了两项收益：

- 字段从 `String` 类型变为 `Int` 类型，作为查询条件时的比对效率变高。
- 能利用 Doris 的前缀索引和布隆过滤器等能力，进一步提高查询效率。

使用 BITMAP 去重代替 COUNT DISTINCT

要提速实验报告查询，针对进组人数（去重用户数）的优化是非常重要的一个部分。作为一个对明细数据强依赖的指标，我们如何在不丢失明细信息的前提下，实现像 Sum, Min, Max 等指标一样高效的预聚合计算呢？**BITMAP 去重计算可以很好的满足我们的需求。**

什么是BITMAP去重？

BITMAP 去重简单来说就是建立一种数据结构，表现形式为内存中连续的二进制位(bit)，参与去重计算的每个元素（必须为整型）都可以映射成这个数据结构的一个 bit 位的下标，如下图所示：

计算去重用户数时，数据以 `bit_or` 的方式进行合并，以 `bit_count` 的方式得到结果。更重要的是，如此能实现去重用户数的预聚合。BITMAP 性能优势主要体现在两个方面：

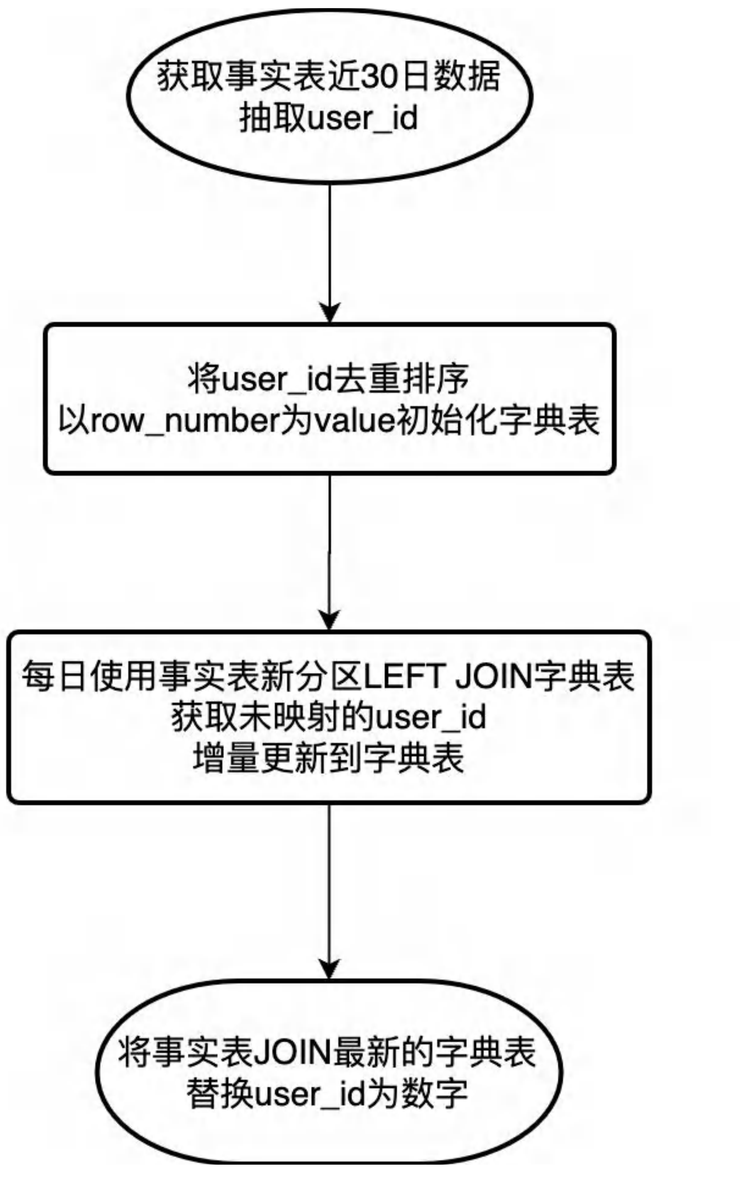
- 空间紧凑：通过一个 bit 位是否置位表示一个数字是否存在，能节省大量空间。以 Int32 为例，传统的存储空间为 4 个字节，而在 BITMAP 计算时只需为其分配 1/8 字节（1 个 bit 位）的空间。
- 计算高效：BITMAP 去重计算包括对给定下标的 bit 置位，统计 BITMAP 的置位个数，分别为 O(1) 和 O(n) 的操作，并且后者可使用 CLZ，CTZ 等指令高效计算。此外，BITMAP 去重在 Doris 等 MPP 执行引擎中还可以并行加速处理，每个节点各自计算本地子 BITMAP，而后进行合并。

当然，以上只是一个简化的介绍，这项技术发展至今已经做了很多优化实现，比如 RoaringBitmap，感兴趣的同学可以看看：https://github.com/RoaringBitmap/RoaringBitmap

全局字典

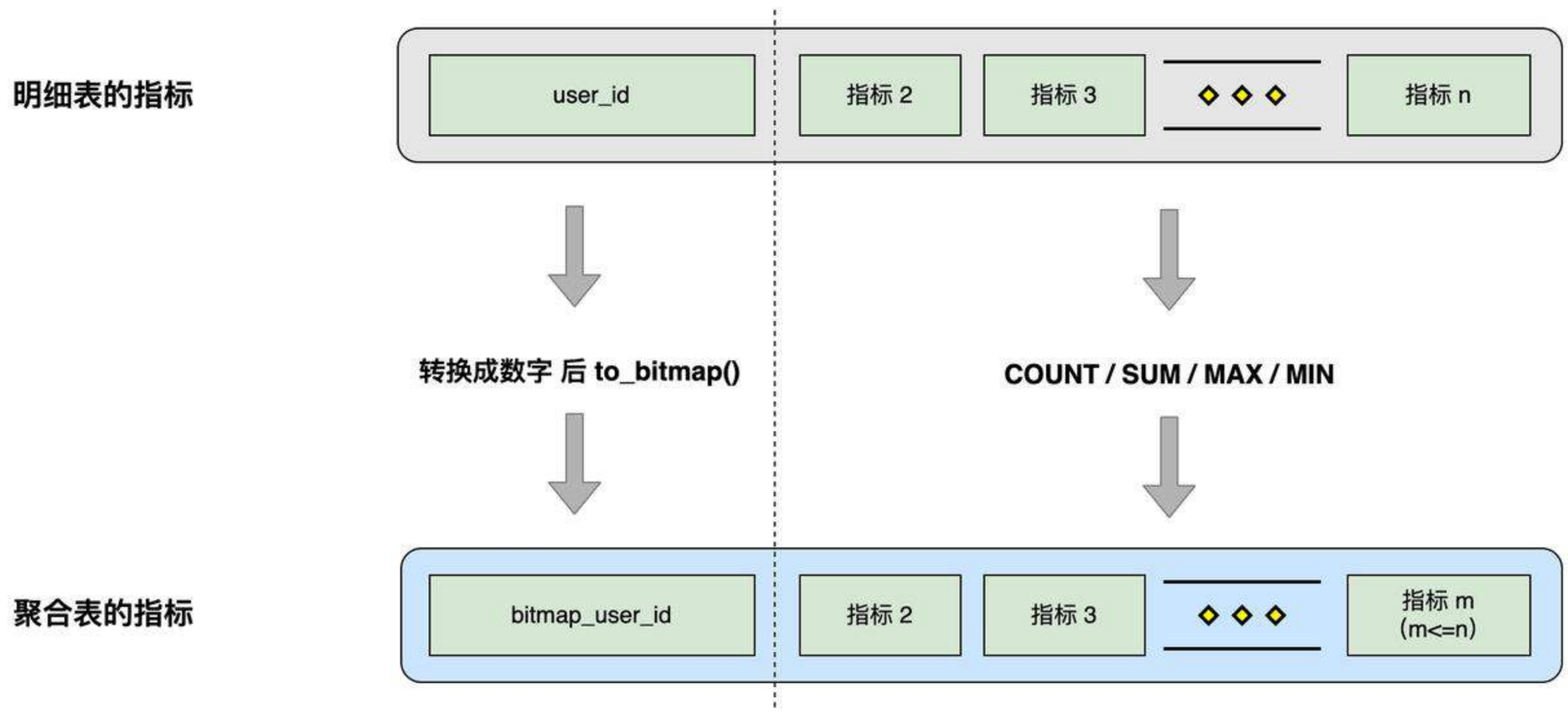
要实现 BITMAP 去重计算，必须保证参与计算的元素为 UInt32 / UInt64，而我们的 `user_id` 为 `String` 类型，因此我们还需设计维护一个全局字典，将 `user_id` 映射为数字，从而实现 BITMAP 去重计算。

由于聚合数据目前只服务于离线查询，我们选择基于Hive表实现全局字典，其流程如下：



指标聚合

生成 Doris 聚合表时，将 `user_id` 作为查询指标以 BITMAP 类型来存储，其他常规查询指标则通过 COUNT/ SUM/ MAX/ MIN 等方式聚合：



如此明细表和聚合表的指标计算对应关系如下：

指标计算	明细表	聚合表
[总数]	COUNT(*)	SUM(cnt)
[总和]	SUM($col)	SUM($col_sum)
[最大值]	MAX($col)	MAX($col_max)
[最小值]	MIN($col)	MIN($col_min)
[去重用户数]	SELECT COUNT(DISTINCT user_id) 或 SELECT COUNT(*) FROM (SELECT user_id FROM table GROUP BY user_id)	BITMAP_UNION_COUNT($bitmap_user_id)

优化效果

SQL视角

查询请求转换成 SQL 之后，在明细表和聚合表的表现对比如下：

<table>
<tr><th></th><th>优化前</th><th>优化后</th></tr>
<tr><td>[总数]</td><td>

```
select count(*)
from doris_xxx
where olap_date = 20221105
and event_name='VIEW'
and exp_id like '%154556%';
```

```
+-----------+
| count(*)  |
+-----------+
| 116068229 |
+-----------+
1 row in set (24.76 sec)
```

</td><td>

```
select sum(cnt)
from doris_xxx_agg
where olap_date = 20221105
and event_name='VIEW'
and exp_id = 154556;
```

```
+------------+
| sum(`cnt`) |
+------------+
|  116068229 |
+------------+
1 row in set (0.40 sec)
```

</td></tr>
<tr><td>[常规聚合]</td><td>

```
select sum(totalFee)
from doris_xxx
where olap_date = 20221105
and event_name='VIEW'
and exp_id like '%154556%';
```

```
+-----------------+
| sum(`totalFee`) |
+-----------------+
|    390079.57866 |
+-----------------+
1 row in set (56.00 sec)
```

</td><td>

```
select sum(totalFee)
from doris_xxx_agg
where olap_date = 20221105
and event_name='VIEW'
and exp_id = 154556;
```

```
+-----------------+
| sum(`totalFee`) |
+-----------------+
|    390079.57866 |
+-----------------+
1 row in set (0.80 sec)
```

</td></tr>
<tr><td>[进组人数]</td><td>

```
select count(distinct user_id)
from doris_xxx
where olap_date = 20221105
and event_name='VIEW'
and exp_id like '%154556%';
```

```
+----------+
| 16264806 |
+----------+
1 row in set (1 min 10.41 sec)
```

</td><td>

```
select bitmap_union_count(bitmap_user_id)
from doris_xxx_agg
where olap_date = 20221105
and event_name='VIEW'
and exp_id = 154556;
```

```
+----------+
| 16264806 |
+----------+
1 row in set (7.54 sec)
```

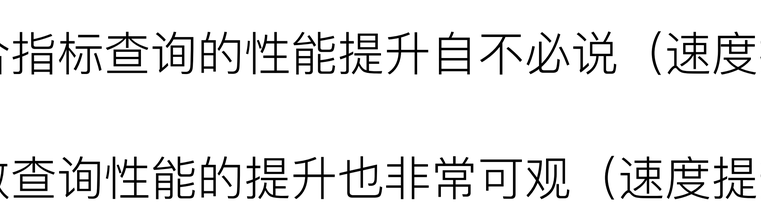
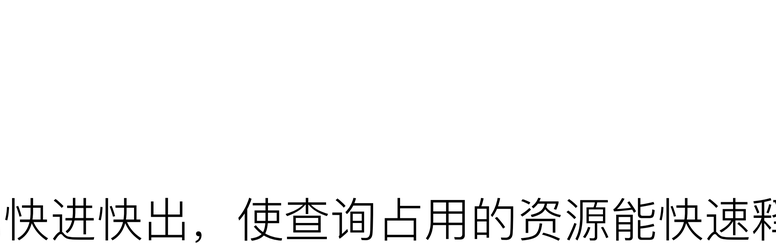
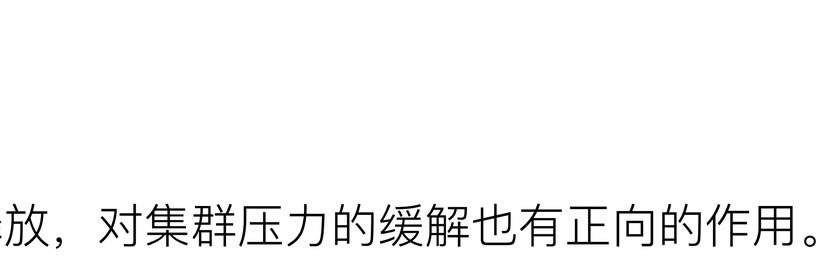
</td></tr>
</table>

- 常规聚合指标查询的性能提升自不必说（速度提升 50~60 倍）
- 进组人数查询性能的提升也非常可观（速度提升 10 倍左右）

集群视角

SQL 查询的快进快出，使查询占用的资源能快速释放，对集群压力的缓解也有正向的作用。

Doris 集群 BE 节点 CPU 使用情况和磁盘 IO 状况的改变效果显著：

	优化前	优化后
CPU 使用率		
磁盘 IO		

需要说明的是，集群状况的改善（包括实验报告查询 P95 提升）并不全归功于数据预聚合优化工作，这是各方合力协作（如产品业务形态调整，后端查询引擎排队优化，缓存调优，Doris 集群调优等）的综合结果。

小技巧

由于业务查询需求的多样，在查询明细表时，会出现一个字段**既作为维度又作为指标**来使用的情况。

如广告业务表中的 `targetConvNum`（目标转化个数)字段，此字段的取值为 0 和 1，查询场景如下：

```
--作为维度
select targetConvNum,count(distinct user_id)
from analysis.doris_xxx_event
where olap_date = 20221105
and event_name='CONVERSION'
and exp_id like '%154556%'
group by targetConvNum;

--作为指标
select sum(targetConvNum)
from analysis.doris_xxx_event
where olap_date = 20221105
and event_name='CONVERSION'
and exp_id like '%154556%';
```

如果这个字段被选取进入聚合表，应该如何处理呢?

我们的处理方式是：

- 在聚合表中把这类字段建模成维度
- 聚合表中需要一个计数指标 cnt，表示聚合表中一条数据由明细表多少条数据聚合得
- 当这类字段被作为指标查询时，可将其与 cnt 指标配合计算得到正确结果

明细表查询：

```
select sum(targetConvNum)
from analysis.doris_xxx_event
where olap_date = 20221105
and event_name='CONVERSION'
and exp_id like '%154556%';
```

对应的聚合表查询：

```
select sum(targetConvNum * cnt)
from agg.doris_xxx_event_agg
where olap_date = 20221105
and event_name = 'CONVERSION'
and exp_id = 154556;
```

结束语

经过这一系列基于 Doris 的性能优化和测试，A/B 实验场景查询性能的提升超过了我们的预期。值得一提的是，**Doris 较高的稳定性和完备的监控、分析工具也为我们的优化工作提效不少**。希望本次分享可以给有需要的朋友提供一些参考。

最后，**感谢 SelectDB 公司和 Apache Doris 社区对我们的鼎力支持**。Apache Doris 是小米集团内部应用最为广泛的 OLAP 引擎之一，目前集团内部正在推进最新的向量化版本升级工作。未来一段时间我们将会把业务优化工作和 Doris 最新的向量化版本进行适配，进一步助力业务的正向发展。

人群圈选效率提升 30 倍

云积互动基于 Apache Doris 构建统一数仓的实践

作者：云积互动大数据团队，王杰、蒙磊

导读：随着业务量快速增长，云积互动对数据的实时性及灵活性提出更高要求，早期基于 CDH 的大数据平台已经无法满足当前的难度以及复杂度较高的业务需求，因此云积互动于 2021 年引进 Apache Doris 在部分业务中使用，并在使用过程中逐渐发掘出 Apache Doris 更多强大之处以及优势，最终决定在 2022 年全面应用 Apache Doris ，基于 Apache Doris 来构建云积互动企业级实时离线统一数仓。

业务背景

云积互动，全称深圳市云积分科技有限公司，成立于 2014 年，是国内领先的 AI 驱动的消费者运营服务提供商，致力于发展消费者运营相关理论、技术、算法、模型及软件工具，为全球消费性企业提供基于 AI 的消费者运营系统及运营策略服务，打造消费者运营领域最佳服务和实践标准，帮助企业构建消费者运营核心能力，以应对当前及未来的场景化运营挑战。

目前已成为天猫、京东、抖音、腾讯等主流电商和社交平台深度合作伙伴，服务客户 2300+，其中世界 500 强客户超过 18 家，包括全球第一的美妆、日化、医药集团，并且均深度服务超过 7 年。

业务需求

云积互动的主要业务是以消费者运营为核心，包含了会员通，CRM，策略营销，数据资产等一系列业务，如下图所示。



早期云积互动的大数据需求较少，起步也比较晚，2019 年才开始搭建基于 CDH 的大数据平台，因此大数据平台的主要目的是为了满足早期较为单一的 BI 数据看板及报表功能。近年来，随着业务量快速增长，数据量的增长，业务对数据的实时性及灵活性提出更高的要求，大数据平台也从早期的只需要满足单一的 BI 服务需求，扩展到需要支持各业务线，包含圈人服务，人群分析，AI 智能数据等多种业务需求。早期基于 CDH 的大数据平台已无法满足当前难度以及复杂度较高的的业务需求。

大数据平台的迭代

早期数仓架构

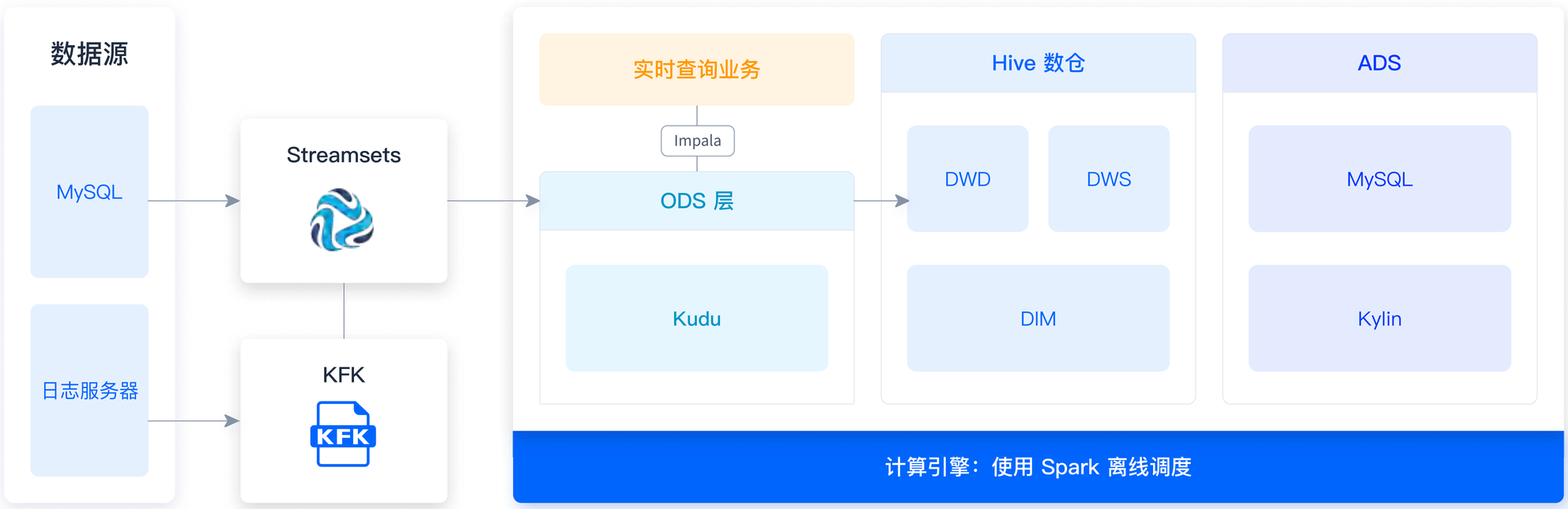
早期公司业务量较少，基于 Hive + Spark 构建的离线数仓即可满足早期大数据的需求。早期架构主要用于支持 BI 相关功能，数据大屏，自助报表等应用，大部分的指标仅要求 T+1 的指标。

下图为云积互动早期数仓架构，早期的数据源主要为业务数据库 MySQL 以及日志，数据通过 Streamsets 实时采集数据并经 ETL 后传入 ODS 层，存储到 Kudu 中，通过 Impala 对 ODS 层的数据进行处理，实现实时查询业务的需求。通过 Hive 构建了离线数仓的 DWD、DWS 以及 DIM 层，使用 Spark 进行离线任务的计算与调度，最终处理并计算完成的数据输出到 MySQL 和 Kylin 中，应用于上层业务应用及分析。（图片见下一页）

存在的问题

- 查询效率低：使用 Impala 多表查询速度太慢，亿级别表 Join 时，查询时间基本上在 3 分钟以上，部分复杂查询会在超过 3 分钟左右判定超时；同时使用 Impala 并行查询对内存消耗较大，影响其他任务运行。

- 存储成本太高：使用多个系统存储数据（Hive，Hbase，Kudu），存储成本较高，随着业务量的增长，数据量指数级的增多，存储成本更是成倍数增加。
- 开发难度大：数仓开发基于代码，不能满足灵活的指标需求，当分析需求越来越多时，存在多个场景组合查询、自定义等查询场景，面对这样的场景，必须进行再开发，开发和时间成本都很高。
- 数据链路长：较长的链路使得数据的一致性很难保证，数据在某一环节出现问题，排查难度高，运维成本也会增加。



技术选型

基于业务对数据实时性及灵活性更高的要求，我们在 2021 年初对当前市面上较为流行的分析引擎 ClickHouse 和 Apache Doris 进行了调研，调研中我们发现， Apache Doris 具有高性能、简单易用、实现成本低等诸多优势。基于此，我们决定在部分业务上开始使用 Apache Doris，在使用的过程中逐渐发掘出 Apache Doris 更多强大之处以及优势，Apache Doris 在很多方面十分贴合我们的诉求，因此，我们决定在 2022 年全面应用 Apache Doris 在数据仓库中，基于 Apache Doris 构建云积互动企业级数仓，选择 Apache Doris 的主要原因如下：

1. 该架构开发效率高，查询性能远高于 Hive。

- 数仓 ETL 由原来的 Spark 任务改为 Doris SQL 任务，使用 SQL 开发模式可进行快速迭代，开发效率提升了近一倍。
- Doris 查询支持物化视图索引加速，Doris 在 1.0 版本开始引入了向量化引擎，性能提升 2~3 倍，平均查询耗时降低了 60%。

2. 该架构对 OLAP 支持更好，支持更为灵活的查询。

- Doris 支持 Cube 函数，实现了 Kylin 的多维计算功能，极大的减少了 SQL 的开发量。
- Doris 支持 Bitmap 类型，可实现人群之间的快速交并差计算并落地新人群。

3. 支持主键唯一和聚合模型的表，极大的减少了开发难度。

- 使用主键唯一模型可做到依据主键数据覆盖，自动实现数据更新功能。
- 使用聚合模型的表，减少了 ETL 过程中的 Join 操作，同时解决了上层数据到达时间不一致而导致的数据关联不上的问题

新数仓架构

最开始使用新的数仓架构主要是要解决圈人速度慢的问题，圈人服务的核心在于人群圈选，通过 SQL 代码或标签取值组合等多种方式，实现人群查找，帮客户找到符合画像的人群，现在各行各业都会设计广告营销场景，其中也包括云积互动，而如何快速准确找到对的人推送广告就成了大数据场景需要解决的问题。当时我们只是在部分功能上使用了 Apache Doris，用 Apache Doris 替代了 Spark + Impala 来实现实时圈人功能，出乎意料的是，**Apache Doris 投入使用之后效果极佳，新版架构的实时圈人业务平均每个任务耗时由 3~5 分钟降低到 10 秒左右，并且在人群落地方面，使用存储更小的 Bitmap 代替原来的人群落地为表，不仅数据管理方便，而且磁盘空间占用减少了 80% 左右。**

除此之外，在一段时间的使用和学习中我们发现 Apache Doris 丰富的功能和核心优势，综上原因，我们产生了用 Apache Doris 数仓替代 Hive 数仓的想法，并迅速的付诸于实践。

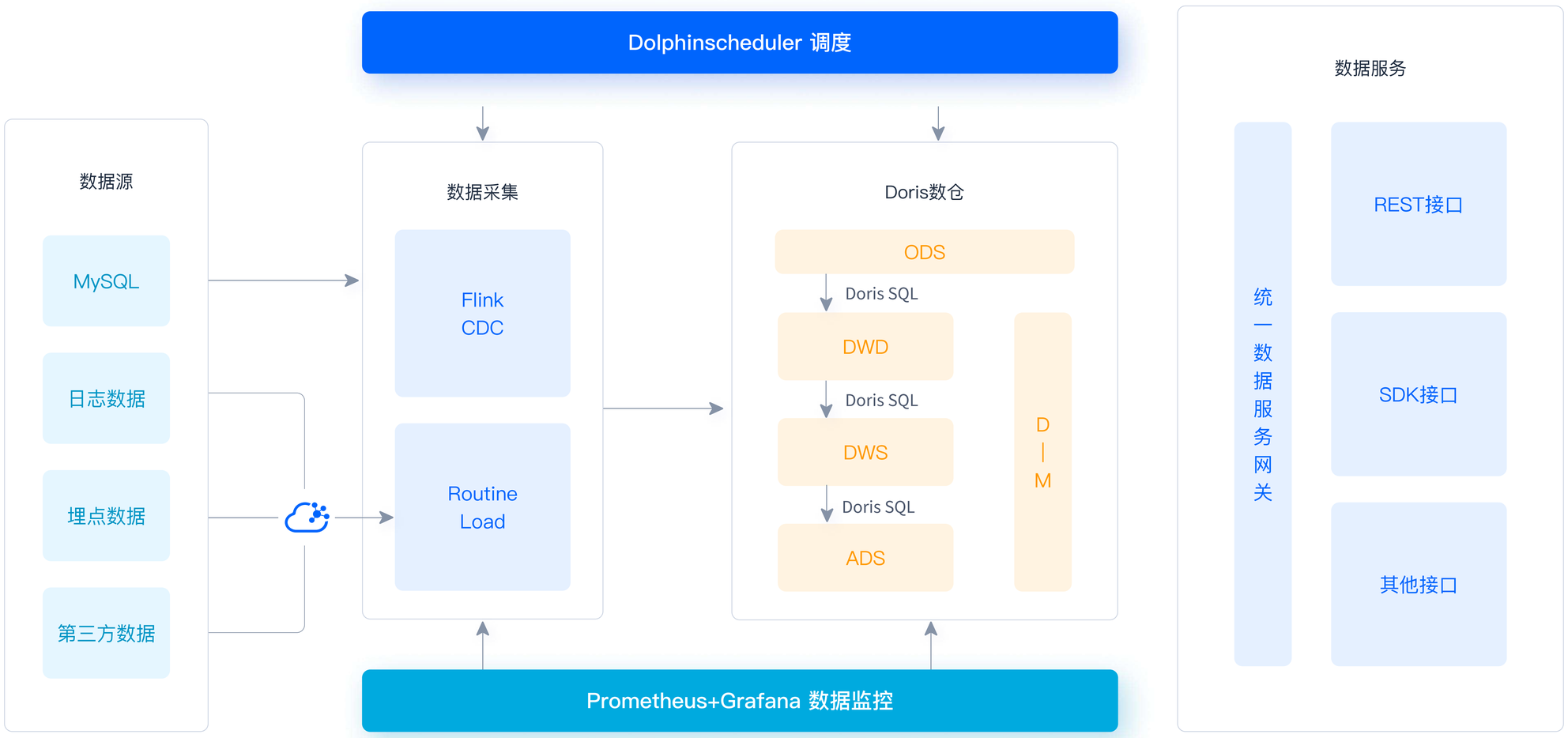
当确定数仓使用 Apache Doris 之后，结合当前的业务需求以及早期架构需要解决的问题，需要将多平台数据打通，构建统一数据口径和数据指标，我们将数据仓库构建分为：ODS 层，DWD 层，DWS 层和 ADS 层，下图为各个分层主要负责的数据类型。



新数仓的分层逻辑如下：

- **ODS 层：**从业务侧、日志系统和埋点系统等拉取过来的数据，按照原字段名存入 ODS 层。
- **DWD 层**：数据按照维度进行拆分，轻度聚合，将多个平台数据按照同一标准定义进行处理。
- **DWS 层：**主要负责数据的聚合、数据宽表、基于维度的一些计算指标等等。值得注意的是，DWS 层中的部分表使用了 AGGREGATE KEY 模型， AGGREGATE KEY 模型可以提前聚合数据，适合报表和多维度业务，可以有效避免数据汇总时的 Join 操作，部分指标可以使用该表特性实现，无需敲代码，降低了开发成本。
- **ADS 层：**各业务模块根据各自的需求，将数据从 DWS 层汇聚数据指标到 ADS 层。

新架构设计



数据接入：

业务数据 MySQL 存储在多台 RDS 中，因 Binlog 的留存时间较短，且数据存放于多服务器，同时还进行了分库存储，因此如何接入历史数据以及如何同时接入多个库的数据成了棘手的问题。在调研过程中我们发现 FlinkCDC 可以完美解决上述问题：FlinkCDC 可以在接入历史数据之后自动切换为读取 Binlog，且 2.x 版本已经支持断点续传，支持水平扩展，支持动态添加表。

日志和埋点数据我们采用 Kafka + Doris Routine Load 导入方式，Routine Load 支持支持用户提交一个常驻的导入任务，通过不断的从指定的数据源读取数据，将数据导入到 Doris 中，支持 Json 解析，并且可以做一些简单的 ETL，极大的减少了代码开发的工作。

数据加工：

数据加工采用了 Doris SQL、Insert into 的方式将增量计算完的结果导入到数仓分层中 (ODS/ DWD/ DWS/ ADS)。因业务需求对数据的实时性允许存在一点延迟性，因此 Dolphinscheduler 设置为每 5 分钟调度一次增量 SQL；同时设置数仓每一层错峰执行任务，避免任务堵塞。

对于数据量比较小的表可以用一个 SQL 完成导入，而对于数据量大的表，为避免 union，需要分成多个 insert into 来执行。但有些大表的逻辑是多个大表的 Join 结果，对于这种场景，我们应用 AGGREGATE KEY 模型的表来解决，利用表的聚合特性来代替 SQL 的 Join 操作。

任务调度：

任务调度一直沿用 Dolphinscheduler，页面化的操作简单方便，且对 SQL 的支持友好，整个大数据平台的任务都是通过该调度器完成。目前使用了2.x 以上的版本，支持使用钉钉报警和邮件报警的功能来监控任务，任务失败将通过钉钉或者邮件发送。

监控：

使用 Prometheus + Grafana 对 Doris 集群和 Flink 任务进行监控管理，页面化的监控，极大的减少运维成本。

优化方案

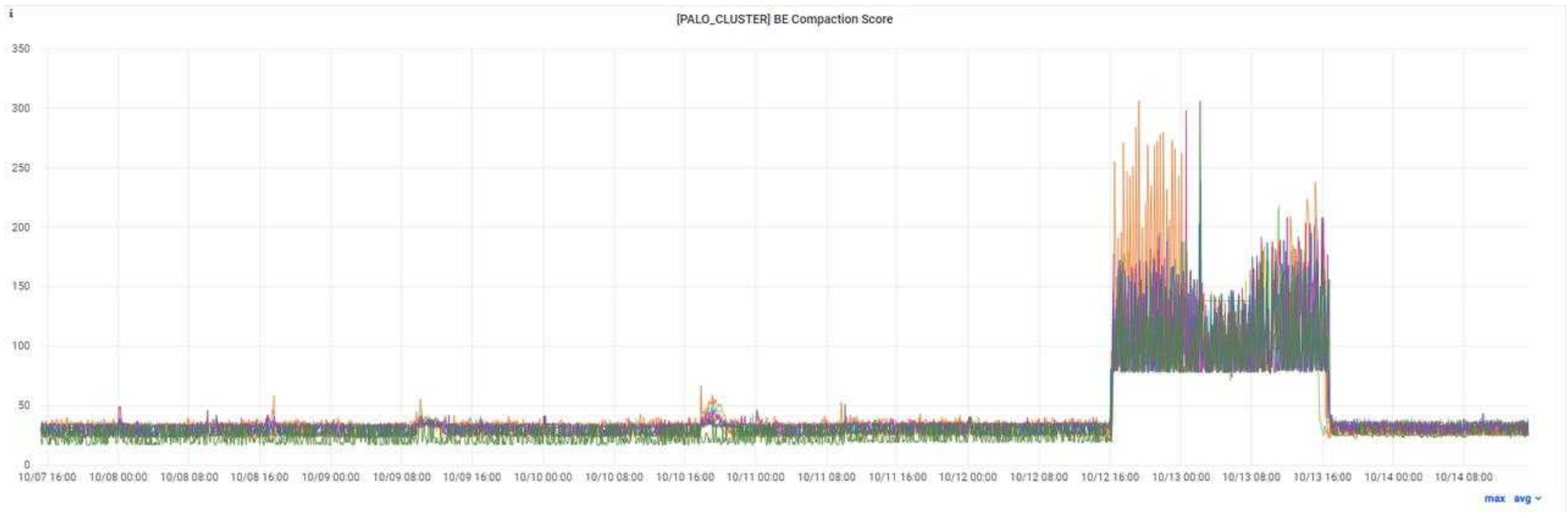
1. 使用 Flink CDC 启动多个同步任务后，磁盘 IO 飙升导致查询延迟变高。

- 对接多个数据源 CDC 任务同步数据时，中间数据写入 Kafka 进行合并和数据消峰，减少写入 Doris 的任务数。
- 调整 DorisSink 写入频率，控制每批次数据量。
- 优化部分表的分区分桶，降低数据分片数量。

2. Doris 1.1.0-rc05 版本偶发后台合并线程持续合并已删除的 Tablet, 合并持续失败且数据版本最多的那个 Tablet 的版本数量升高至 150 左右。

- 使用元数据管理工具 meta_tool 删除已失效的 Tablet 元数据，**版本数量显著下降，稳定在 30 左右。**

- 对大表的超过两年的数据做冷备份，**减少大表的 Tablet 数**，降低整个 Doris 集群对 Tablet 的管理压力。



3. Bitmap 存储散列用户 ID，使用 Bitmap 相关函数计算时性能较差。

- 用户唯一 ID 通过字符串转换生成，基数大且非常稀疏；使用全局字典生成紧凑的 ID 代替，优化后性能提高近五倍。

总结与收益

- Apache Doris 构建的离线+实时数仓一体化，采用 SQL 开发，并用 Dolphinscheduler 一键部署调度，**极大的降低开发难度和开发工作量**，可进行快速迭代以满足目前行业日益增长的数据需求。

- 新架构采用 Flink + Doris 的架构体系，FlinkCDC + StreamLoad 可以做到流批一体化数据接入，减少了组件的使用，解决了数据的冗余存储，**服务器资源节省了 30%，数据存储磁盘占用减少 40%，**同时组件的运维成本大大减少。

- Doris 的易用性极高，支持 MySQL 协议和标准 SQL，各业务线均可通过查询 MySQL 的方式进行数据查询，极大的减少了学习成本。

- 从 2021 年 Apache Doris 上线云积互动的第一个业务至今，**Apache Doris 在云积互动内部已成为大数据服务的基础，**承担了包括人群分析、报表查询、指标计算等场景下的在线/离线需求，在较小的集群规模下支持了每天近 2 万次的用户在线分析查询。

未来规划

目前新的数据仓库已经建设完成，基于 Apache Doris 较多优异特性以及与业务需求较高吻合性，当前团队已经在着手搭建基于 Apache Doris 的数据质量管理和数据血缘，后续我们计划基于 Apache Doris 搭建数据指标体系。下面简单分享一下我们在数据质量管理和数据管理的实现想法。

1. 数据质量管理：在 Apache Doris 的 ODS 层建表时设定一些非空主键，这些字段都是业务逻辑上的必须字段，当数据接入会给定一些默认值，这样就可以清晰的分类出这些数据，在质量分析中进行输出；在 ETL 中也会存在一些逻辑错误数据，这类数据会通过定时的 Doris SQL 脚本进行输出，同时也可以反馈到业务侧进行数据修复。

2. 数据血缘：依托 Doris 提供的 SQL 审计功能，使用采集工具 Filebeat / Logstash 持续采集审计日志发送到 Kafka，使用开源的 SQL 解析工具或者抽取 Doris 的 SQL 解析模块针对 DDL 或者 DML 进行解析，解析后的数据存入图数据库或者关系型数据库供业务端展示；该功能的实现对于数据问题排查、数据资产管理均有意义。

围绕着 Apache Doris 为核心的数据平台建设目前也在一直迭代发展，当然在使用中也发现了该产品的一些需要优化的地方，但不可否认它优秀的性能和丰富的功能，后续我们也将持续不断地进行优化，将优化方案贡献给 Apache Doris 社区。

一文揭秘字节跳动

基于 Apache Doris 的实时数仓探索

作者：字节跳动数据平台 E-MapReduce 团队，昭伟

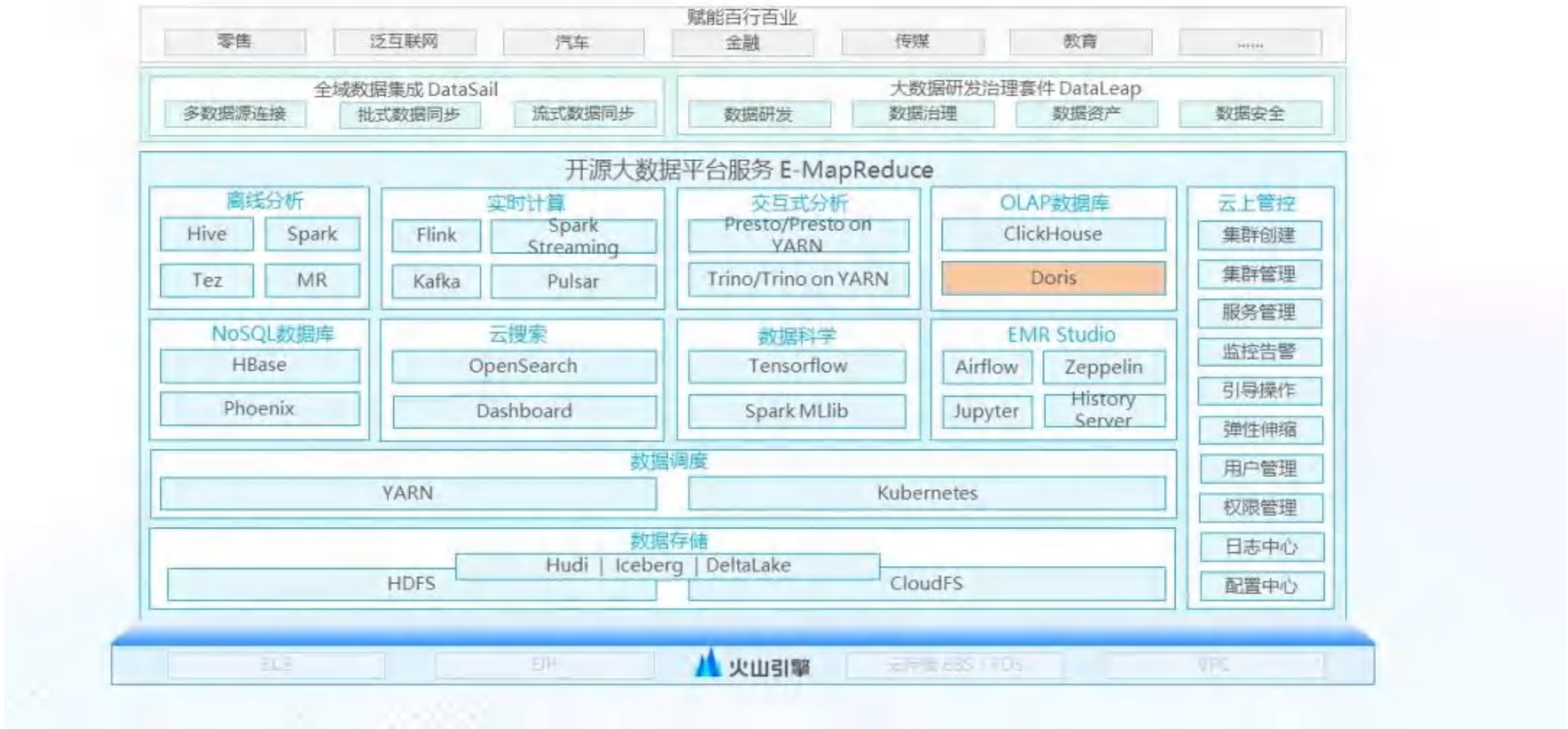
导读：火山引擎 EMR 作为一款云原生开源大数据平台产品，集成了包括 Hadoop、Spark、Flink 等引擎，并做到 100% 开源兼容。Apache Doris 作为 OLAP 领域中一款极具代表性的开源组件，也被集成到了火山引擎 EMR 产品生态中。本文来源于火山引擎 EMR 团队大数据工程师在 Doris Summit 2022 中的演讲实录，将为大家详细介绍火山引擎 EMR 是一款怎样的产品，火山引擎 EMR 团队对 Doris 社区做出了哪些贡献，火山引擎 EMR Doris 目前具备了哪些能力优化，以及后续的规划方向。

火山引擎是字节跳动旗下的云服务平台，将字节跳动快速发展过程中积累的增长方法、技术能力和工具开放给外部企业，提供云基础、视频与内容分发、数据平台 VeDI、人工智能、开发与运维等服务，帮助企业在数字化升级中实现持续增长。

火山引擎 EMR 是一款云原生开源大数据平台产品。首先，从开源大数据平台角度，火山引擎 EMR 集成了开源大数据生态的众多软件栈，包括 Hadoop、Spark、Flink 等引擎，并且做到 100% 开源兼容。**Apache Doris 作为一款 OLAP 领域极具代表性的开源组件，所以我们也将其集成在火山引擎 EMR 生态中。其次，从云原生角度，我们也会基于云的特性做深度的能力增强，例如弹性伸缩、存算分离等。**

我们在产品发布之初就已经集成了 Doris 引擎，它也是目前火山引擎 EMR 系统中的主力 OLAP 引擎之一。

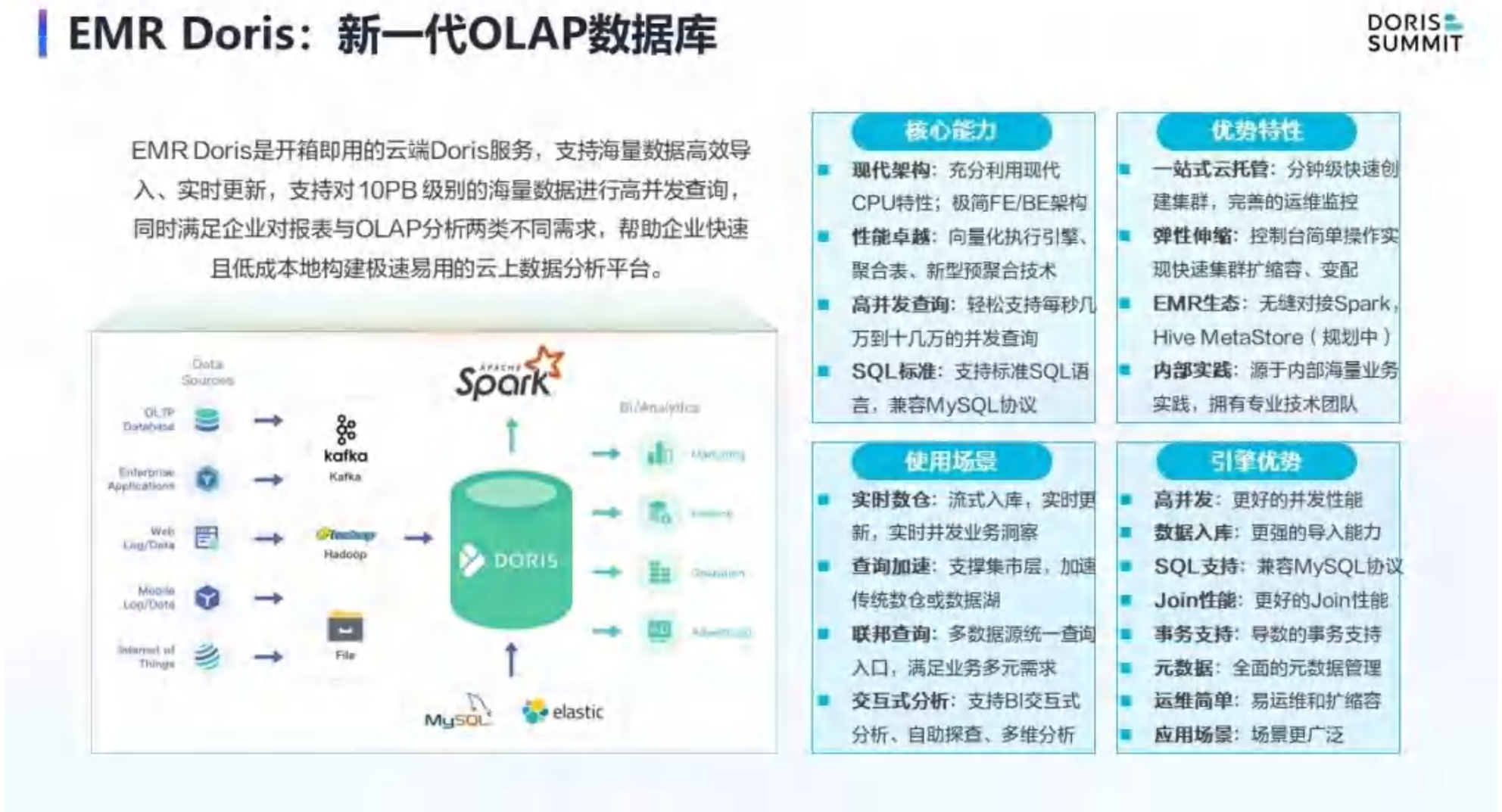
火山EMR: 云原生湖仓一体功能架构



EMR Doris 是一个开箱即用的云端 Doris 服务。支持海量数据的高效导入、实时更新，支持对 10PB 级别的海量数据进行高并发查询。我们认为 Doris 也是一个比较全面的 OLAP 引擎，不同于 ClickHouse 仅在大宽表的聚合上较为擅长，Doris 的各方面能力均比较出众。

- 支持向量化执行引擎，完备的 MPP 执行框架，多表关联能力更优异；
- 用户使用更友好，支持标准 SQL 以及 MySQL 协议；
- 支持预聚合表引擎，能方便快速地实现数据的聚合；
- 数据接入更便捷，提供多种数据导入方式以及异构数据源的访问能力；
- 具备物化视图的能力，能够实现查询自动路由，通过预计算来提高查询速度和并发。

因此，基于 Doris 全面的 OLAP 服务，火山引擎 EMR 很早对其进行了集成，进行了大量功能优化，并将其作为主力 OLAP 引擎之一推向了整个公有云市场。



基于 Doris 的能力优化

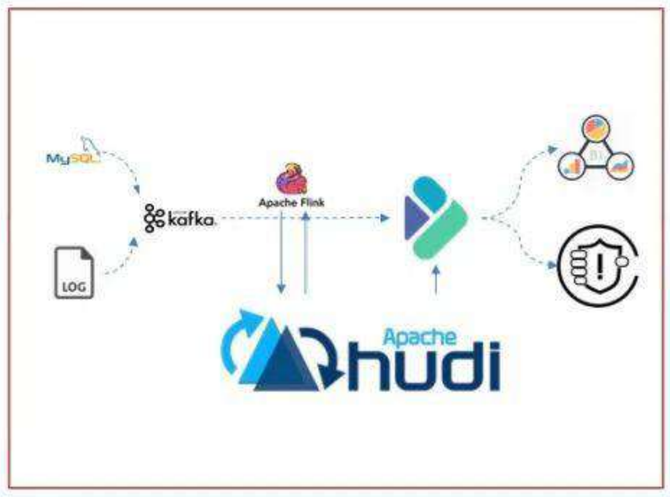
我们对 Doris 的研发主要以贡献开源社区为主，下面将为大家介绍下火山引擎 EMR 团队近期对于 Doris 社区的主要贡献。

基于 Hudi 的数据查询方案

在 22 年上半年，社区和火山引擎 EMR 团队一起做了基于 Iceberg 和 Hudi 的数据湖查询方案，火山引擎 EMR 团队主要负责 Hudi 的方案。

在此之前，查询 Hudi 需要通过一些导入工具，把 Hudi 的数据加载到 Doris 的内表中进行计算。但这样会带来一个坏处，数据链路相对来说会更长。因此我们把 Doris 作为一个数据服务层，和下层的 Hudi 做数据打通，**实现数据的直接查询，避免数据反复导入导出**。另外，通过打通 Hive Metastore，来直接访问 Hive Metastore 获取库表的元数据，而不是通过表映射来关联字段，**从而大大提升了数据开发的效率**。

通过以上优化，**Doris 实现无缝查询 Hudi 表**。当然，目前这一方案只支持 Hudi 中 CopyOnWrite (COW) 存储类型的表，对 MergeOnRead (MOR) 表的支持尚在规划中。



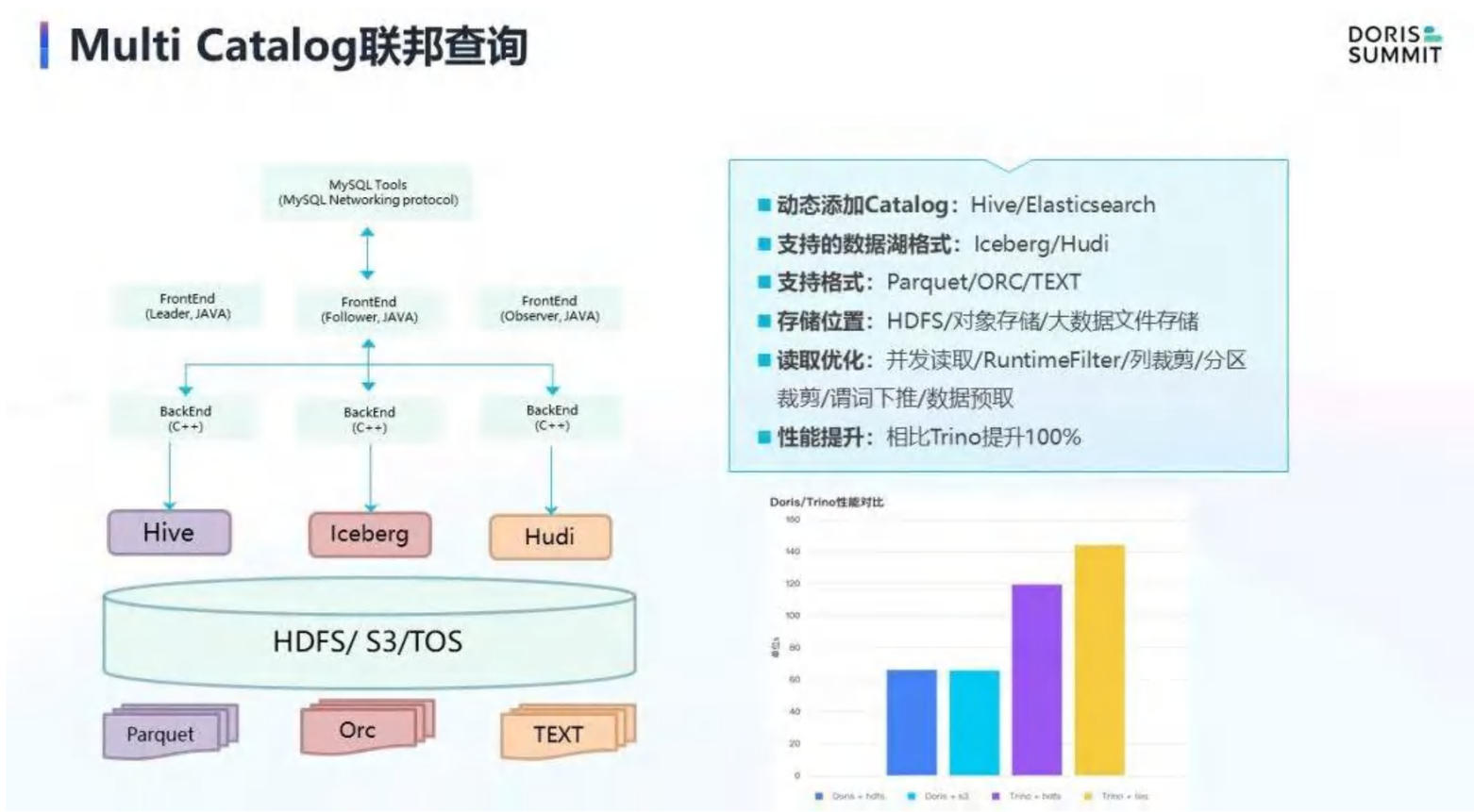
- 使用Doris作为数据服务层
- 打通Doris与数据湖Hudi
- 打通Doris与HiveMetaStore

Multi Catalog 联邦查询

22 年 6~7 月，我们和社区合作开启了 Multi Catalog 联邦查询的项目，目标是让 Doris 能像 Presto 一样有 Plugin 的能力，可以进行联邦查询，实现 ES、JDBC 等数据源的查询，以及最典型的 Hive、数据湖的联邦查询。

最终我们通过 Catalog 直接查询 Hive、Iceberg、Hudi 表。经过了两个月的开发，目前已经支持三大数据组织模式，也支持数据存放在 HDFS、S3 和 TOS 上，数据格式支持最常见的 Parquet、ORC、TEXT 等。

基于这些能力，我们在性能上也做了持续的优化。例如，我们做了 Table Scan 中最常见的几类优化，包括并发读取、RunTimeFilter、列裁剪、分区裁剪、Parquet 和 ORC 中的谓词下推、数据预取等。做了这些有效的优化以后，相对于 Trino， 在同样的场景下，也就是 Trino + HDFS 或者 Trino + S3 的模式，**对比发现 Doris 的查询性能相比 Trino 要提升了近一倍左右。**

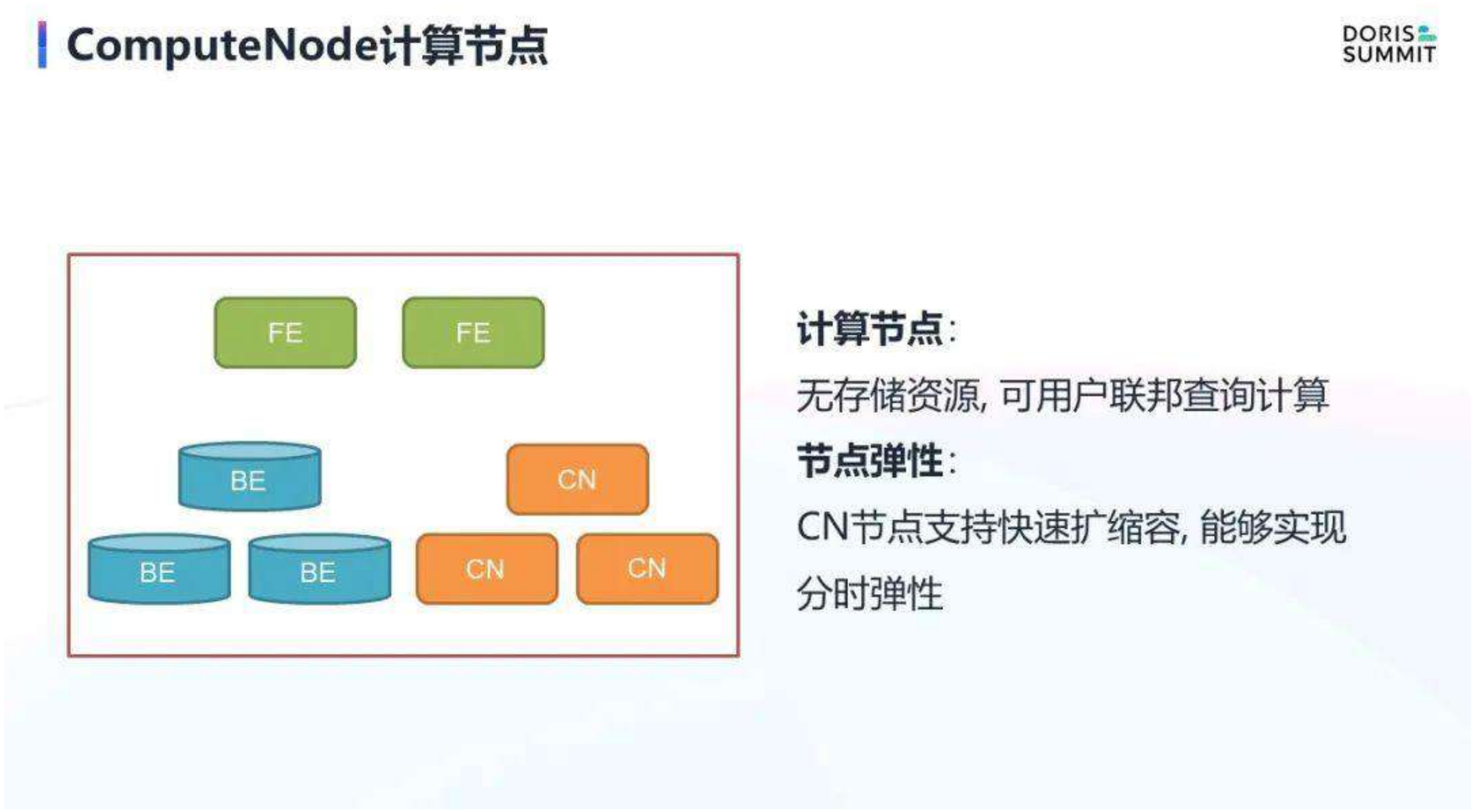


ComputeNode 计算节点

计算节点与联邦查询有很大的关联性。Doris 本身是典型的 Share-Nothing 架构，所以在它的 BE 节点上计算和存储是强绑定的，这样会带来几个影响：

- **扩容频繁，**计算资源不够则需要扩容，磁盘不够也需要扩容，只要满足一个条件，就必须要扩容。
- **弹性能力差，**因为每个节点都绑定了数据，一旦扩容就需要做数据的迁移。而一旦涉及到数据的迁移，时间相会比较长。而在联邦查询的场景下，用 BE 去承载联邦查询的计算相对来说比较厚重。

基于这种假设，我们做了 ComputeNode 计算节点的功能。**顾名思义，计算节点只有计算，没有存储，这样就非常适用于联邦查询场景。**因为联邦查询本身没有数据存储，数据都是从远端拉过来的。另外它能很好地支持弹性，扩缩容的时候也不需要做数据搬迁，只要进程起来可以执行任务，资源过多时也可以在一两分钟之内将资源回收。因此**计算节点可以实现非常好的弹性，可以支持分时弹性，这也是最常用的弹性策略。**



多表物化视图

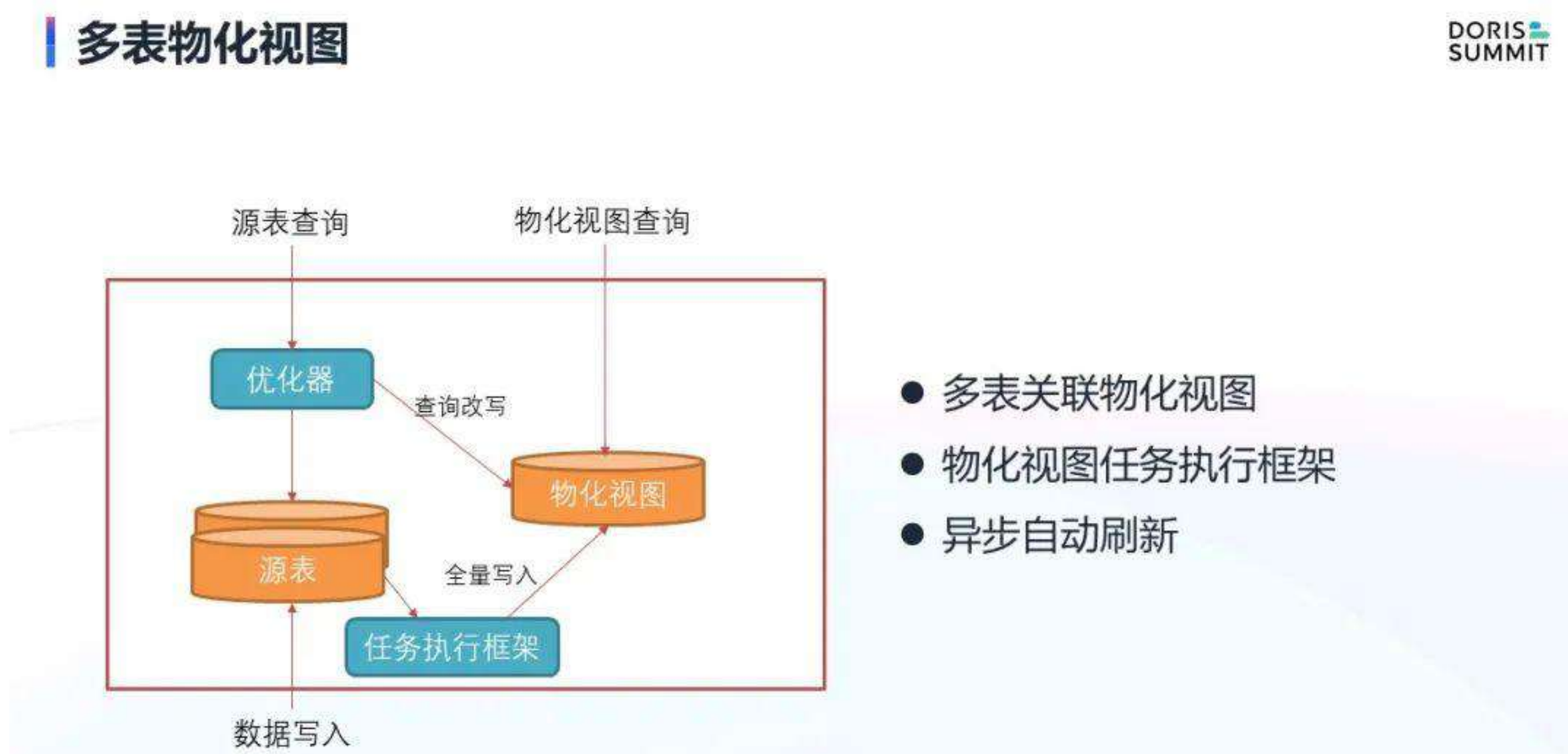
物化视图是一个典型的空间换时间的策略，通过预计算，配合查询时优化器的改写能力来直接查物化视图表，避免重复查询原表消耗过多的资源。

Doris 本身具有物化视图能力，但只针对单表的，可以对数据做简单聚合，因此我们也经常把它当做聚合索引。数据聚合后不再需要查原表，直接查物化视图性能会快很多，这也是它最大的使用场景。

但该功能目前存在一些限制：

- 仅支持简单聚合、不支持函数运算，例如当在 Sum 函数中嵌套加入 Case When 语法时，物化视图无法创建。因此在即将发布的 Apache Doris 2.0 中，社区对单表物化视图进行了增强，可以支持更加复杂的 SQL 表达。
- Doris 有比较好的 MPP 执行能力，经常会被用来做多表的关联查询，大宽表场景相对较少，因此如果物化视图只有单表时无法最大化发挥 Doris 的性能和场景优势。

基于以上需求，我们联合社区一起研发了多表物化视图，并在即将到来的 Apache Doris 2.0 版本发布这一重大功能。届时可以将带有 Join 的查询结果固化以供用户直接查询，支持定时自动或手动触发的方式进行全量更新查询结果，后续也将进一步支持更加完善的自动增量刷新。

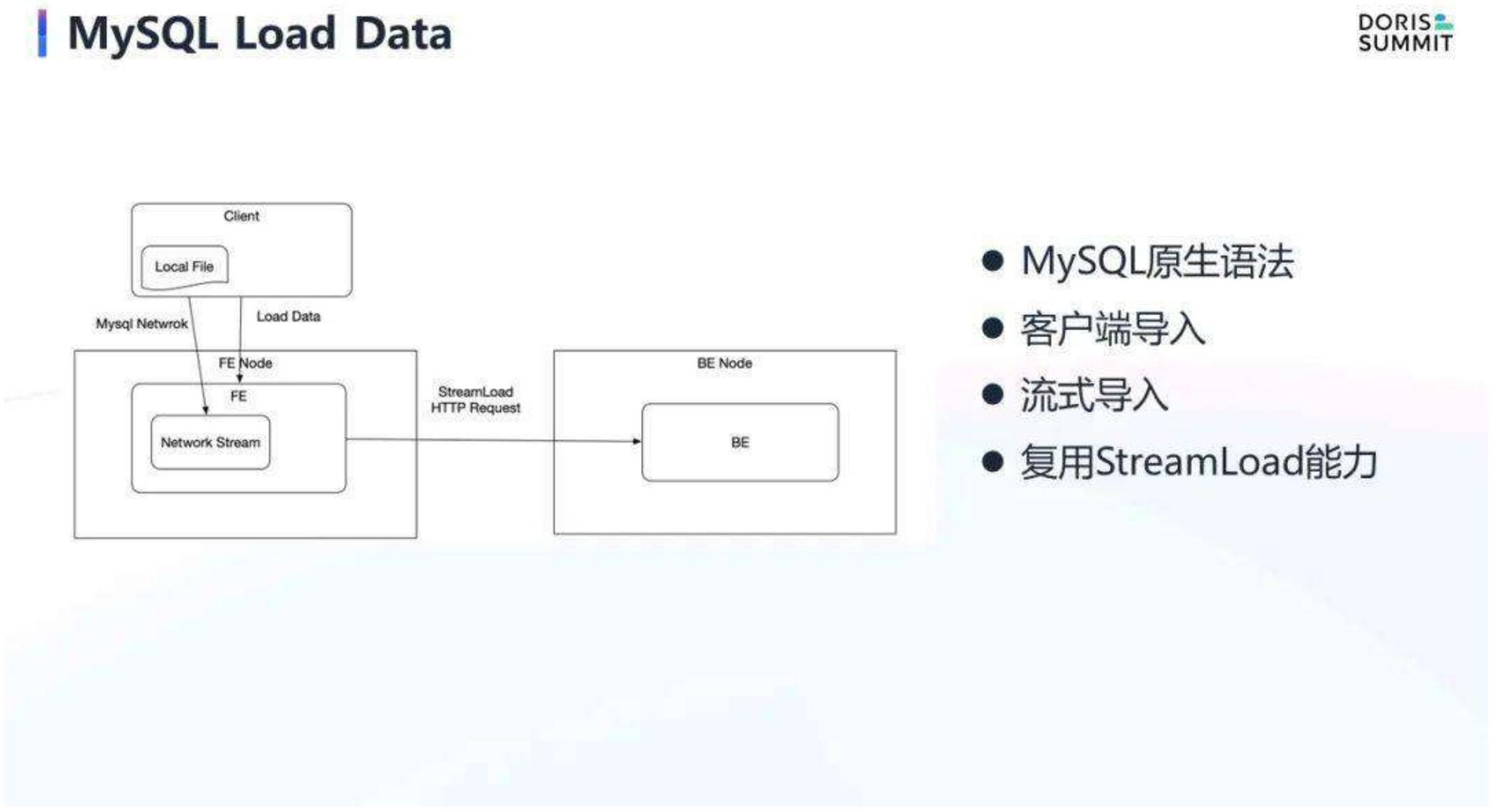


MySQL Load Data

Load Data 是 MySQL 中的原生的语法，可以将数据从客户端加载到服务端的，几乎所有的 MySQL 生态都支持该语法，也是一种标准的数据导入方式。

在 Doris 中可以导入本地数据的方式只有 Stream Load，它本身是 Shell 命令行，跟 MySQL 的语法有些差距，因此用户就需要做很大的改写。

基于这种背景，我们希望能支持 SQL 语法导入客户端本地文件的能力。在具体实现中，我们按照 MySQL 网络协议，通过客户端包发送方式，把在本地的数据通过 MySQL 客户端直接发送到 FE 节点，再通过流式的方式封装成 Stream Load HTTP Request 发给 BE，然后 BE 调用原来 Stream Load 的逻辑把数据导入进去。这个功能实现 MySQL 生态的 Load Data 能力，也是 Doris 支持 MySQL 生态比较大的一个点。



多流 Upset

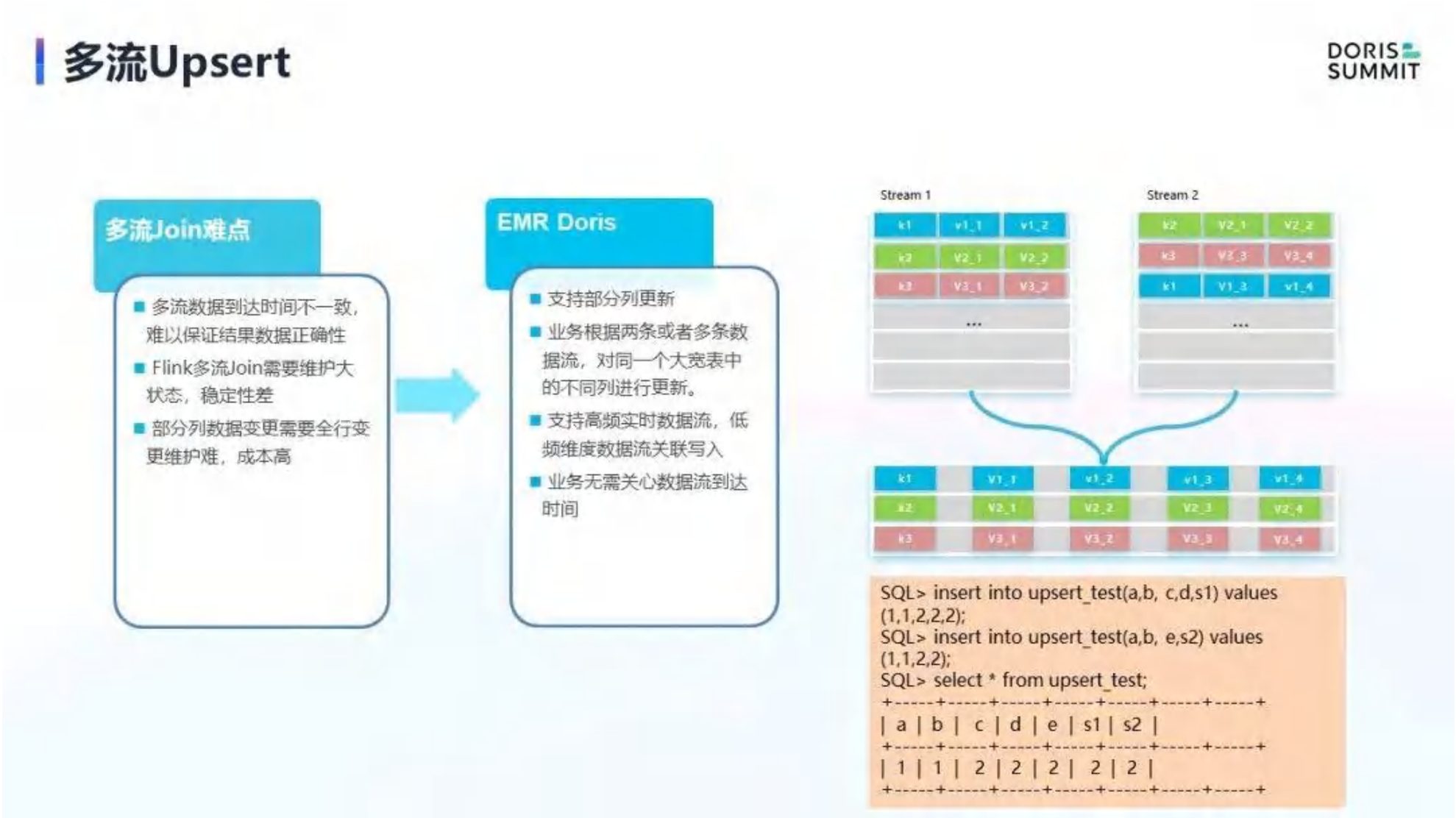
多流 Upsert 源自于 Flink 中的多流 Join，而多流 Join 需要维护比较大的状态，会导致集群不太稳定，因此很多 OLAP 引擎都支持部分列更新的能力、支持多流 Upsert。

我们基于原来的 Unique Key 引擎实现了部分列更新的能力。具体能力如图右侧所示（图片见下一页），有两个 Stream，它的主键是 K1、K2，数据也有可能是乱序的，在要进行 Merge 的时候，会根据 Key 值把数据按照它的顺序合并起来。

有该特性以后，数据的 Join 就变得比较简单,性能相对于原本使用 Flink 也有较大提升。节省了大量的回撤操作，所以在存储引擎去做这个能力还是非常不错的一种选择。

我们有位客户需要对 8 表做 Join 的一个案例，因为客户之前是用 Flink 写到 PostgreSQL中的，PostgreSQL 本身支持多流 Upsert，如果在 Flink 里面改写为 join 方式会非常困难。**而 Doris 支持多流 Upsert 后，用法就跟 PostgreSQL 的用法完全一致，效果也会好很多。**

在性能方面，如果数据量不太大的时候，性能表现很优异，而当数据量特别大的时候，目前的这套实现还不是特别好，主要原因是多流 Upsert 是基于旧版本 Merge-on-Read 实现的，在读取时要做大量的合并操作。目前社区 Unique Key 新的 Merge-on-Write 模式实现了性能的大幅优化，后续我们也将计划与社区一起进行全新 Unique Key 部分列更新。



基于 Doris 的应用实践

以上是火山 EMR 对社区的贡献，下面将介绍我们基于 Doris 到底做出了一个怎样的产品。

首先我们对客户提供的是云服务，所以除了引擎本身，引擎周边相对来说是我们重点布局的方面，例如下图右侧监控告警。如果是 IDC 用户，监控告警一般需要自己基于 Grafana 搭建。

而作为云上的一款产品，提供完善的运维监控体系就可以大大简化用户的搭建成本。我们将相关的运维相关的功能直接在控制台页面透出，例如日志查询，以前日志需要自己去采集，现在也是完全不用户操心，直接在日志中心里面看到所有的服务器日志，用户只要在这个页面上查询，跟原来用 ES 去做搜集和用 Kibana 做展示 的效果差不多的。

除了监控、日志以外，还有集群的扩缩容能力，这也是云上的这种服务化能力优势的体现。对于集群节点配置，用户也不需要手动管理，而是直接使用配置中心，通过控制台页面很方便的进行配置更改。



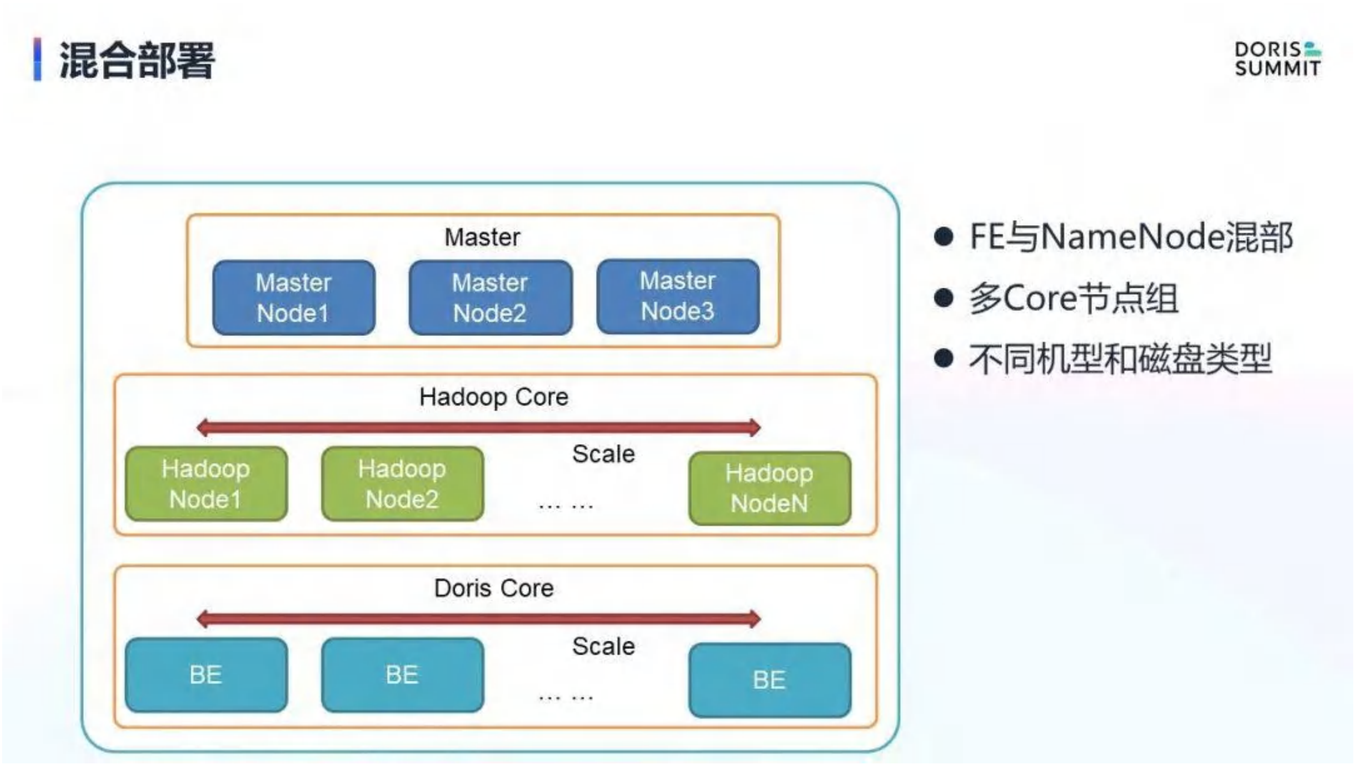
接下来的几个点，我们要谈谈火山引擎 EMR 相对于其他云上 EMR 产品， 在 OLAP 引擎有哪些差异点。

混合部署

在下图场景中，FE 和 NameNode 是在 Master 节点中混部的。实际上在计算的时候，只需要用到 BE 节点，但 FE 这三个节点又必须部署，不部署就无法正常工作。如果部署在 BE 中，又会受到隔离性的挑战。我们很多用户也有这种方面需求，希望能把 FE 节点和 Master 节点混部在一起。Hadoop 的 NameNode 也与之类似，它不参与计算，只是作为一个 RPC 的响请求响应，做一些主备功能，或者做元数据管理等。由于不承载数据的大规模吞吐，所以 NameNode 和 Doris FE 的定位比较像。

因此我们做了这样一个优化， 将 FE 都部署在 Master 节点上，完成了一个混部。但计算节点目前我们不推荐用户去混部，因为 Doris 和 Hadoop 对磁盘、机型的要求不太一样的。Hadoop 比较合适使用本地的 HDD 盘，但是 Doris 就是推荐本地 SSD 盘。

所以我们做了这样的多 Core 节点组策略，也就是 Hadoop Core 和 Doris Core 节点组，两者可以选择不同的机型和不同的磁盘类型，他们彼此独立的扩容节点，互不影响的情况下实现弹性能力。Master 节点是混部的，随着业务增长 Master 节点也支持升配，实现垂直扩容能力。

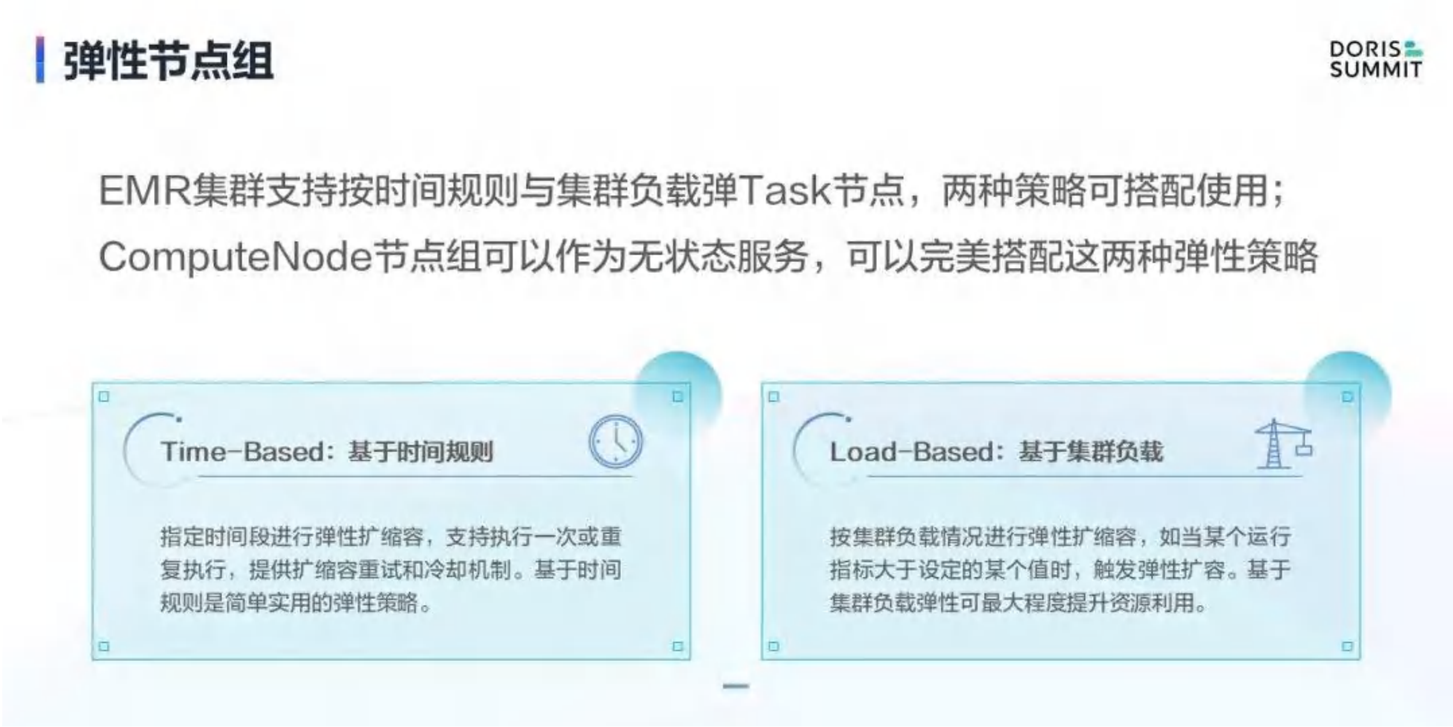


弹性节点组

Doris 本身不具备弹性的，而是有状态服务。但由于我们做了 ComputerNode 能力，也就使 Doris 支持了弹性能力。

有了 ComputerNode 后，CN 节点就可以按照要求就去做弹性策略了。我们可以完按照时间力度去做弹性，比如早上需要计算的数据多，就进行扩容，其他时间段，数据处理量不大就进行缩容。这也比较符合联邦查询的特点，一般都是白天负载会相对高一点。

另外一个是基于集群负载的弹性能力实现自动的扩缩容。我们也支持基于时间和基于集群负载的弹性策略混合使用。



自适应配置

Doris 目前的默认配置都是基于 IDC 的模式，一般都是按照大规格机器（128G 以上）去推荐设置的。这对于一些云上比较小的客户，比如 30G 以下的小规格的集群，会带来很多问题，我们之前遇到了很多 OOM 问题。

因此我们基于不同的数据规模，设计了不同的集群规格，并针对每种规格地进行不同的配置优化。

- 针对小规格，我们就将 page cache 关闭，把 buffer pool 调小，并调低 index cache 和 Load 内存配置，调小 session 内存。
- 针对大规格，我们主要是调大默认 session 内存和默认 batch_size 大小。
- 中规格相对来说比较中庸，我们调小了 page cache，调低了 load page 内存配置和 index cache。

通过这些配置的调整，我们完成了大中小规格的建设，减少用户的使用成本，也提高了整个集群的鲁棒性。

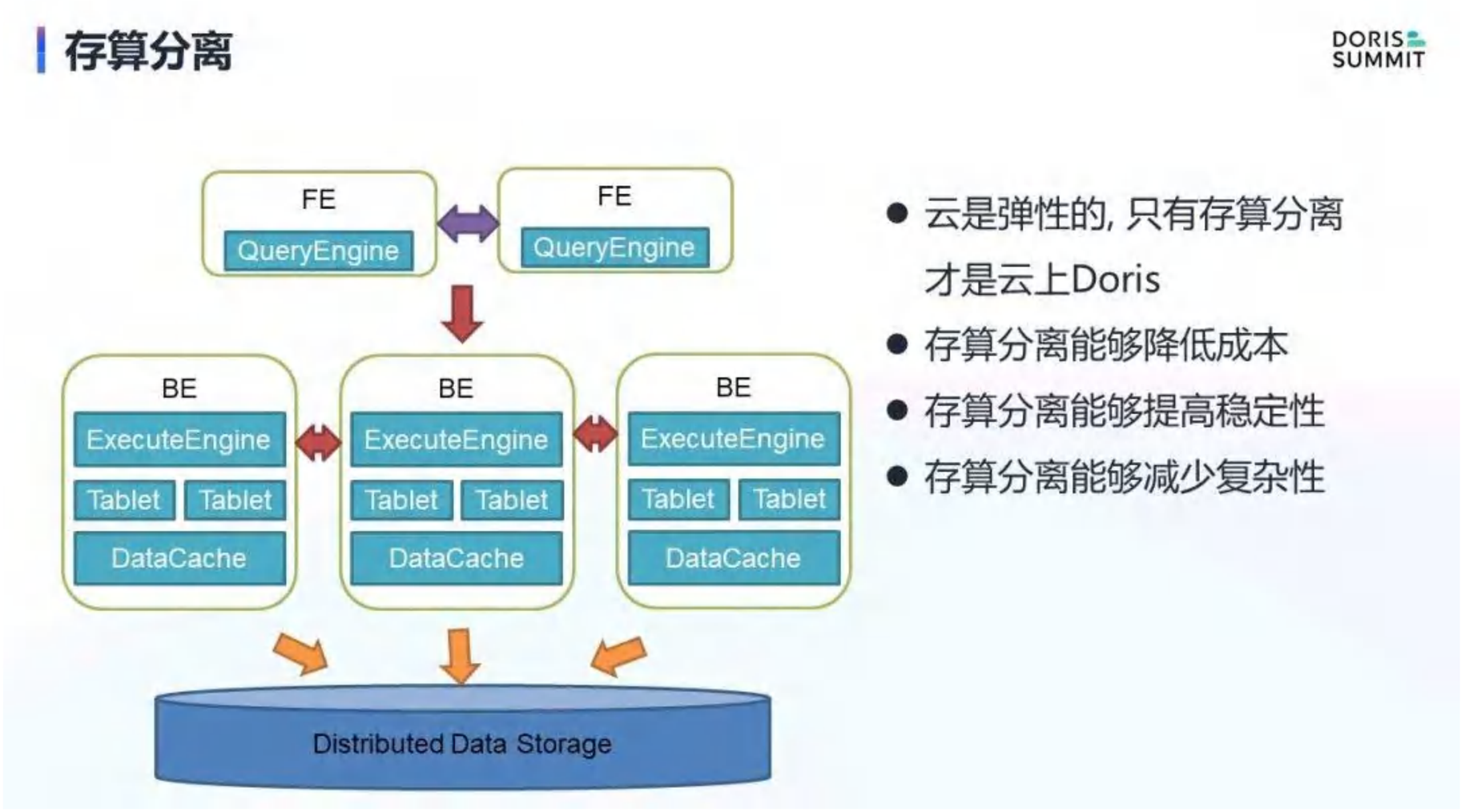


未来规划

1. 弹性能力，云产品最核心的价值是弹性能力，而 Doris 最初是基于 IDC 场景设计，弹性需求相对来说较低，所以短期内我们还需要对其进行大量的改造，来实现高弹性化。

2. 数据存储方面，因为 Doris 数据是自身进行管理，通过 Tablet 副本实现数据的高可用性。但在云时代，Doris 仍旧自己管理数据其实没什么太大的必要，因为云上有 S3、 TOS 这些对象存储产品，它们能保证非常高的可用性，能达到了十几个 9 级别，这是通过 Tablet 副本很难实现的，这也是我们做存算分离的一个初衷。

3. 存算分离方面，存算分离是把数据放到远端持久化存储中，通过缓存缓解一部分的查询性能压力。在此之上，我们整个集群就有了更好的弹性，剥离开数据之后的扩容也变的更加灵活方便，相当于做了一个 stateless 的 BE，存算分离也能帮助用户进一步降低成本。另外在稳定性上也会有很大的提升，不需要自己通过 Tablet 副本实现高可用能力了，可用性可以直接交给 S3 和 TOS 等对象存储产品来实现。整体系统复杂度也会大大降低，这也是存算分离能带来的一个非常大的优势。



4. MetaServer，完成了数据的存算分离，数据弹性问题得到了解决，但元数据问题依旧存在。现在 Doris 的源数据是通过 bdb 做选主, 通过本地做数据持久化, 通过 3 副本做高可用，这种模式依然依赖于 3 节点来实现高可用。**我们希望能做一个 MetaServer，把元数据接到一个外部 MetaServer 中去，这样有几个好处：**

- 突破 Master 节点内存限制。现在的元数据全在内存中，当 Tablet 个数超过几千万的时候，内存消耗比较大。我们之前测过差不多 2000 万个 Tablet 的时候，内存达到将近 20G，这 20G 的数据完全在内存里，没办法用磁盘去做溢出。如果有了 MetaServer，比如基于 MySQL RDS，那就只将一些热点缓存在 FE 中，其他的有需要的时候再去拉取，时延也和现在的模式不会有太大差别。

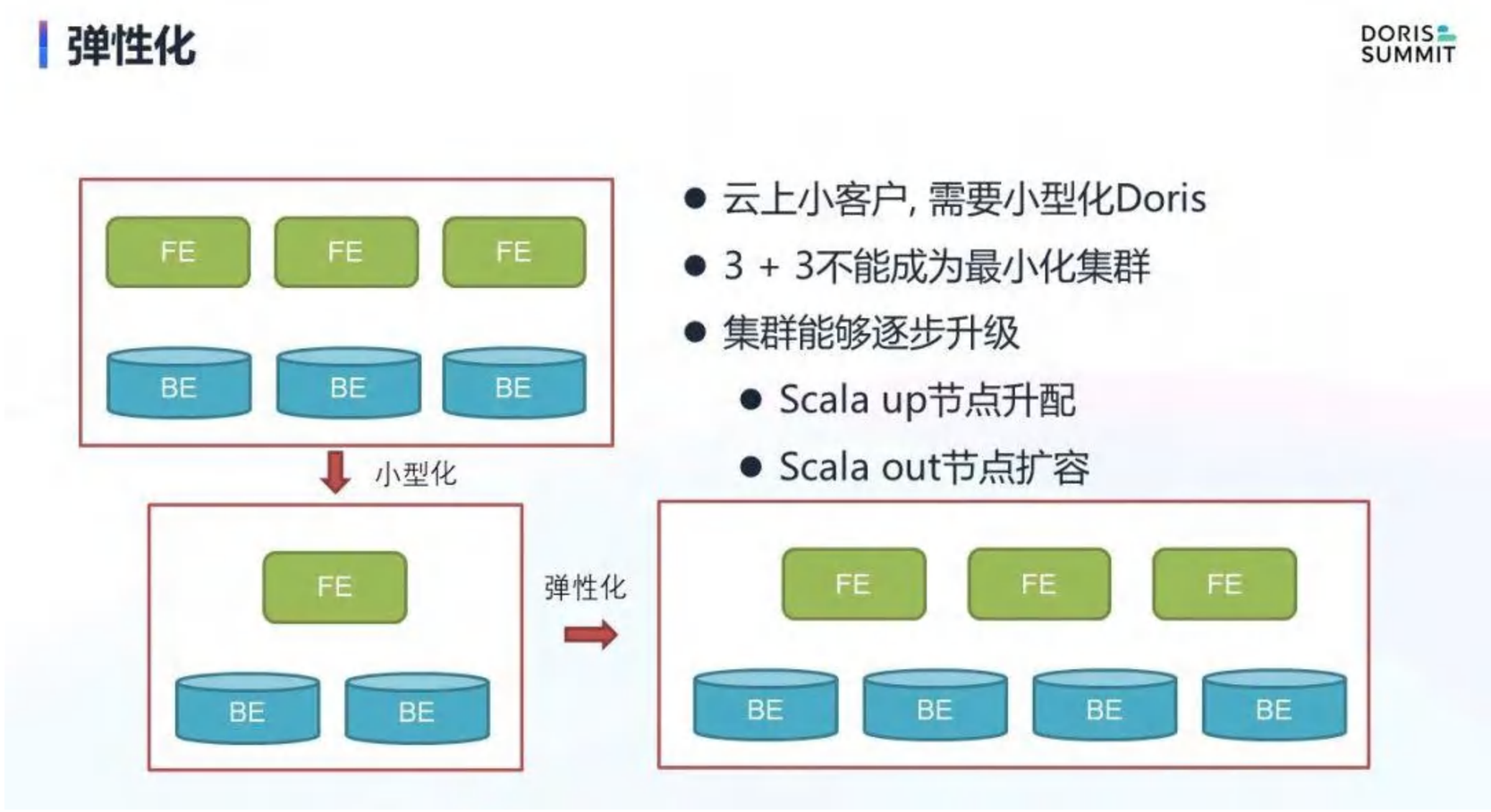
- 元数据的高可用性。MetaServer 产品可以做到跨 AZ 级别，甚至跨 region 级别的高可用，但通过 FE 来说实现是非常困难的，这也是云上和云下的一个巨大的差异点，云上可以通过依赖标准的云产品的能力来实现自己能力，而在云下这些都需要自建。

- 节约成本。现在 FE 要通过三节点实现高可用，如果有了 MetaServer，只要一节点就可以，成本也就随之降低。

- 支持更多类型的外部元数据存储，比如 RDS，KV 数据库等。

如果说前面提到的存算分离、MetaServer 都是手段，那我们真正的目的是希望做到 Doris 的极致弹性化。

针对弹性化，集群规模可以跟随业务发展不断升级，这也是个云的基本能力。扩容有两个方向，一个 scale up，节点升降配都是比较常规的能力了。另外一个 scale out，这是本身 Doris 就有的，但是如果做了存量分离和 MetaServer 后， scale out 能力会进一步加强。有了这两个能力，我们才可以说 Doris 成为了一个真正意义上的云上 Doris。



打破数据孤岛，Apache Doris 助力纵腾集团快速构建流批一体数仓架构

作者：纵腾集团数据技术架构师，张彬华

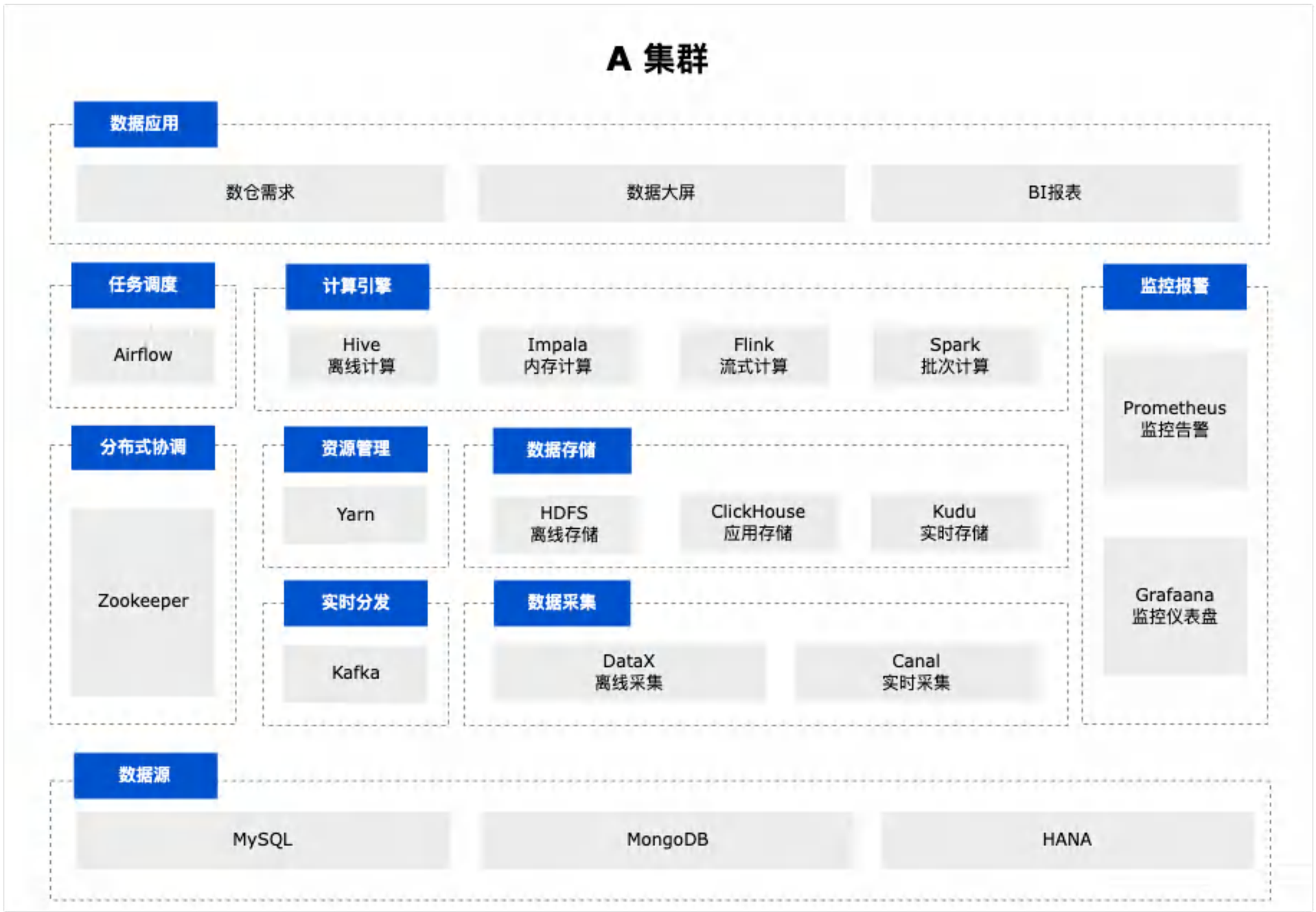
导读：福建纵腾网络有限公司（简称“纵腾集团”）成立于 2009 年， 以“全球跨境电商基础设施服务商”为企业定位，聚焦跨境仓储与物流， 为全球跨境电商商户、出口贸易企业、出海品牌商提供海外仓储、商业专线物流、定制化物流等一体化物流解决方案， 旗下拥有谷仓海外仓 、云途物流 、WORLDTECH 等知名品牌 。

随着纵腾集团业务的快速发展，各产品线提出的数据需求越发严格，而早期基于多套 CDH 大数据架构的技术栈和组件繁杂，开发和运维难度高、效率低，数据质量和时效难以保障，已无法满足当下数据分析需求，严重影响相关工作的开展。因此，**纵腾集团在 2022 年正式引入 Apache Doris，基于 Apache Doris 构建了新的流批一体数据架构，同时建立了以 Apache Doris 为核心的数据中台。**构建过程中对读写时效性、服务的稳定性及高并发读写等多方面进行了优化，在这一过程中我们也积累了诸多实践经验，在此总结分享给大家。

早期架构

早期数仓架构主要分为两套基于 CDH 的大数据集群，这两套架构用于不同产品线的数仓需求、数据大屏和 BI 报表等应用。

这两套架构（架构图见下一页）为独立的数据管道，具有耦合度低，集群间相互独立等特点，便于精细化管理。但随着业务需求的不断变化，这样的特点也引发出许多新的问题。


A 集群
数据应用
数仓需求
数据大屏
BI报表
任务调度
Airflow
计算引擎
Hive
离线计算
Impala
内存计算
Flink
流式计算
Spark
批次计算
监控报警
Prometheus
监控告警
分布式协调
Zookeeper
资源管理
Yarn
数据存储
HDFS
离线存储
ClickHouse
应用存储
Kudu
实时存储
Grafaana
监控仪表盘
实时分发
Kafka
数据采集
DataX
离线采集
Canal
实时采集
数据源
MySQL
MongoDB
HANA


B 集群
数据应用
数仓需求
数据大屏
BI报表
任务调度
Azkaban
计算引擎
Hive
离线计算
Presto
内存计算
Flink
流式计算
监控报警
Prometheus
监控告警
分布式协调
Zookeeper
资源管理
Yarn
数据存储
HDFS
离线存储
HBase
列式存储
TiDB
实时存储
Grafaana
监控仪表盘
实时分发
Kafka
数据采集
DataX
离线采集
Canal
实时采集
数据源
MySQL
SQL Server
HANA

遇到的问题

- 元数据和数据质量缺乏管控，数据质量无法得到保证
- 不同业务数据独立存储维护导致数据孤岛，不利于数据整合
- 每个集群的机房分布不一，维护成本非常高
- 集群间的技术栈和组件较多且存在差异性，对统一开发运维和数据整合都极具挑战性

架构选型

为了解决早期架构的痛点、更好满足日益严苛的数据需求，我们希望能有一款产品帮助我们快速构建流批一体的数仓架构、构建数据中台服务。



我们对传统数仓、 实时数仓和数据湖进行了对比。从上图可知，传统数仓可以支撑超 PB 级的海量数据，但是交互查询性能相对差一些，偏离线场景，不满足我们对数据实时性的要求；数据湖可以支撑超海量的数据，支持数据更新，查询性能适中，但是数据湖近两年才开始应用，成熟度较低，使用风险较大；实时数仓适用 PB 级数据存储，支持数据更新且查询性能非常好。结合我们的要求，**实时数仓与我们的使用和需求场景都比较贴合，因此我们最终决定选择实时数仓作为数据底座。**

接着我们对市面上较为流行的三款实时数仓：ClickHouse、Apache Druid、Apache Doris 进行了选型对比，对比图如下：

产品	优点	缺点	社区活跃度	运维成本
ClickHouse	向量化引擎成熟 单表查询性能强悍 支持较多的表引擎 具有较多的聚合函数 提供庞大的优化参数选项	数据导入无事务保证 不支持标准SQL语言 几乎不支持分布式 JOIN 扩容缩容复杂，繁琐且不支持在线操作	中	高
Apache Druid	支持流式和批量数据的导入 支持倒排索引和位图索引优化查询 可进行数据预计算，提高查询速度 基于时间分区，加快查询速度提供性能	查询类型比较单一 实时的数据摄取较慢 无法关联查询（新版支持但性能差）	高	高
Apache Doris	独立主从架构且简单 通过外表机制可快速实现联邦查询 多样的导入方式可应对不同的导数场景 丰富的数据模型可应对不同的数仓层级 兼容 MySQL 协议和标准 SQL，JOIN 支持全面	产品年轻 问题出现后较难找到可直接复制代码的解决方案（但社区活跃，响应极快）	高	低

对比可知，Apache Doris 优势明显、性价比更高，具有独立主从架构简单、运维更灵活便捷、丰富的数据模型、优秀的查询性能和周全的生态规划等诸多优势，对比这三个产品，Apache Doris 最符合我们的选型要求。

新数据架构



新数据架构基于 Apache Doris 简化了数据采集、存储和计算的流程：

- 结合 DataHub 实现自研元数据采集和周期管理
- 通过 Seatunnel 集成 Flink Doris Connector 稍加改造实现全量加增量数据的一体化采集
- 简化存储媒介，**对 ClickHouse、Kudu、HBase 等技术栈进行收敛，由 Apache Doris 进行流批数据的统一存储**
- 以 **Apache Doris 为核心数据底座**，结合 Apache Kyuubi 的 JDBC 引擎直连查询（自研）和 Spark 引擎中的 Spark Doris Connector 进行 ETL 开发（原生），统一计算引擎管理、权限管控和对外服务。

基于上述几点进行了数据应用开发及对外提供数据服务，构建了数据中台。

数据中台

我 们 以　Apache　Doris 为核心底座创建了数据平台，核心功能包括：指标中心、元数据中心、基础配置中心、即席分析和数据接口服务中心，其中指标中心和即席分析的数据主要来源于 Aapche Doris ，当前已上线几百个指标。



数仓建模

我们结合 Apache Doris 的特性重新对数仓进行了建模，数仓分层与传统数仓类似，其中 ODS 数据为存量加增量一体的导入模式，同时为防止出现[随机查询结果问题]，**ODS 层最终选用 Unique 数据模型**，相比于 Aggregate 模型可以实现写时合并（Merge-on-Write），有效提高数据实时性，且 Aggregate 模型查询性能更接近于 Duplicate 模型，对于 ODS 层是非常好的选择。

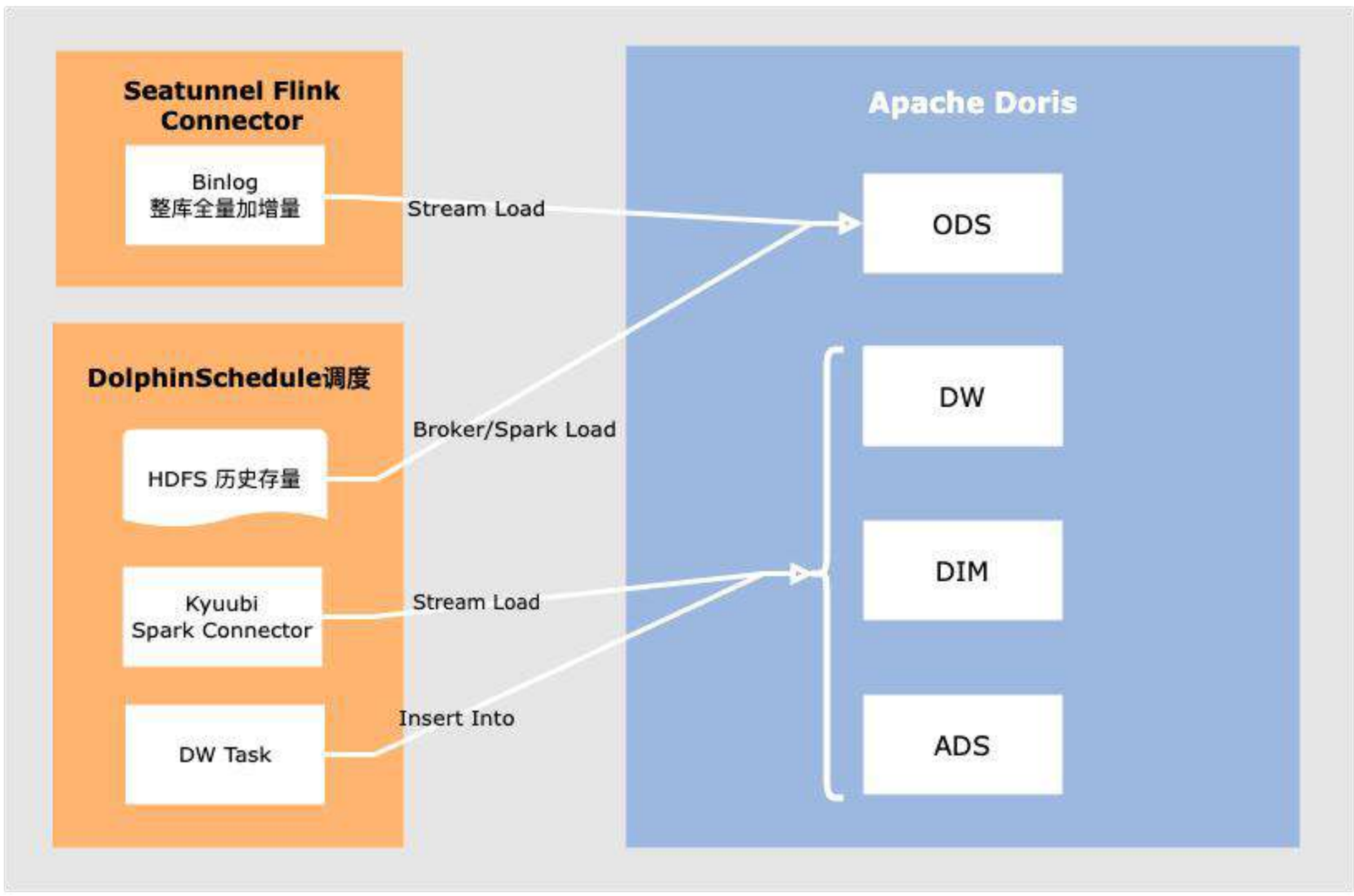
DIM/DED/DWS/ADS 层主要选用 Aggregate 数据模型；Aggregate 数据模型提供的四种聚合方式可以在大部分场景下达到事半功倍的效果，帮助我们快速应对不同的需求场景。

- **SUM：**能够高效实现 PV 类指标计算，但对于 UV 类的指标需要考虑预去重。
- **MAX/MIN：**常用于最大最小运单时间节点类指标或包裹体积/重量最大最小值的指标计算。
- **REPLACE_IF_NOT_NULL：**可以自动地过滤空值，非常便捷地实现仅记录最后一条数据，适用于大部分 DW 场景。

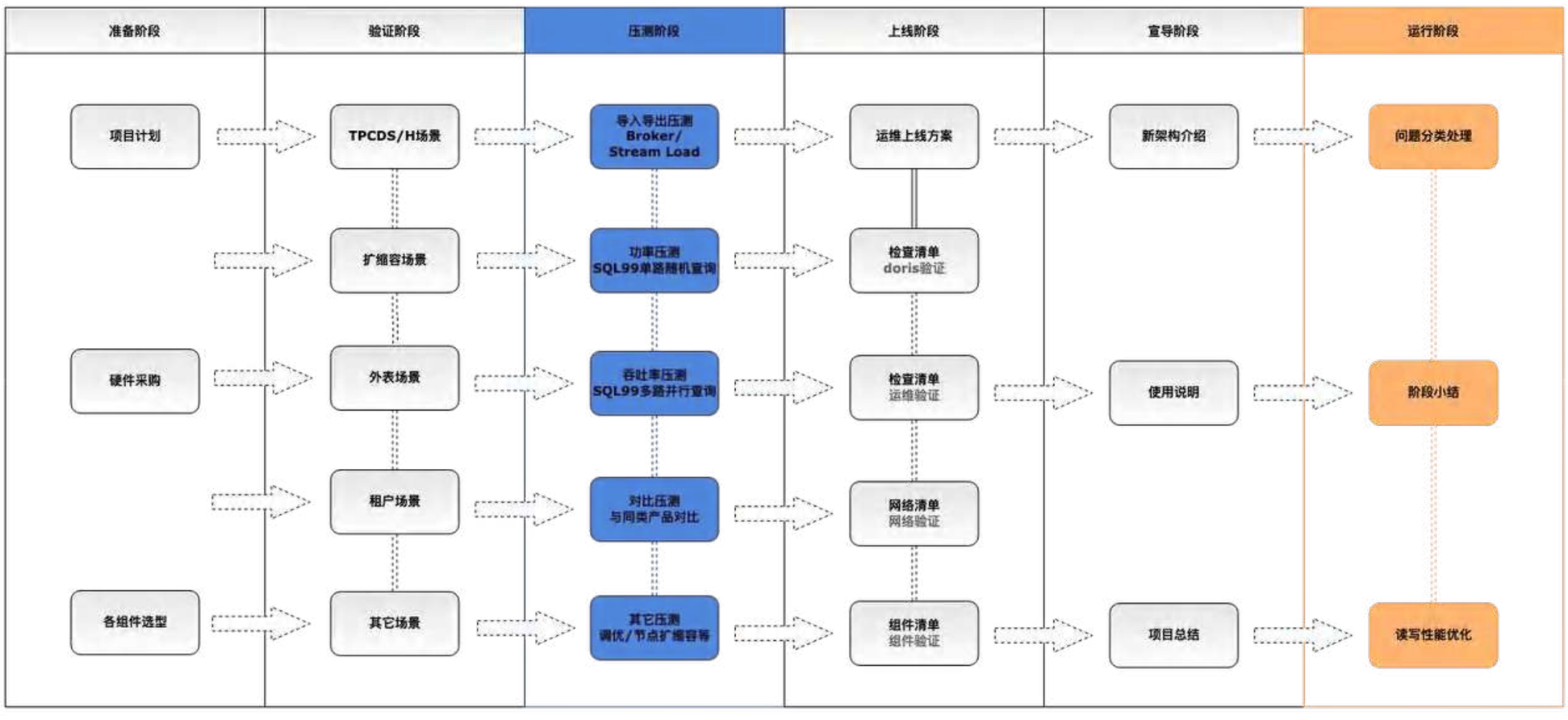


数据导入

ODS 层的数据导入目前主要以 Stream Load 为主，在 HDFS 上的历史存量数据也会通过 Broker Load 或 Spark Load 导入。**DW 层数据主要以 insert into 方式导入**，同时为减轻 Doris 内存压力，我们将部分 ETL 任务放到 Kyuubi On Spark 引擎上去计算，目前在 DolphinScheduler 每天平稳调度 Doris DW 任务有上万个，其中大部分为 T+1 任务，小部分为小时级任务。



实践经验



对于以 Apache Doris 为核心的新数据架构，我们规划了 6 个阶段进行运行测试，直至可以上线运行。（重点关注压测阶段和运行阶段，有一些调试优化经验分享给大家）

1、准备阶段

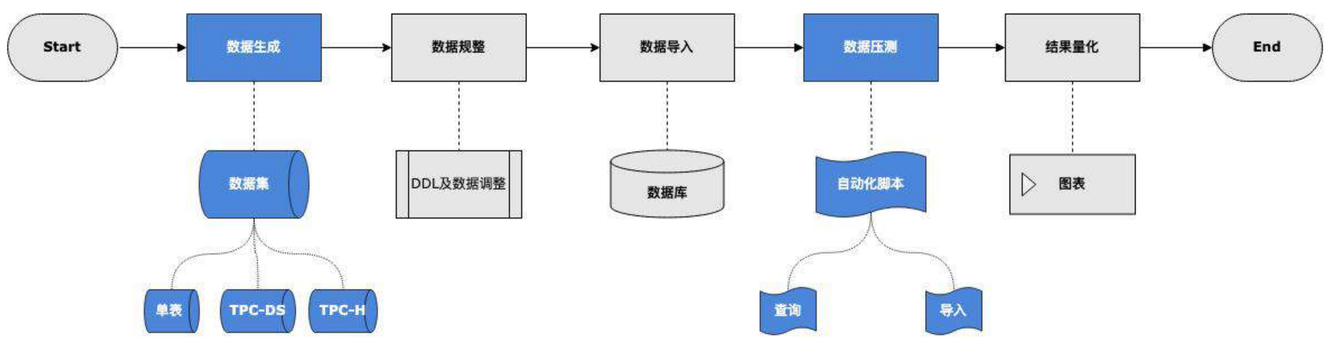
引入 Apache Doris 时是 2022 年 2 月，因此选择当时最新版本 Apache Doris 0.15 Release 版本进行应用，主要考虑维度如下：

- 支持事务性插入语句功能
- 支持 Unique Key 模型下的 Upsert
- 支持 SQL 阻塞 List 功能，可以通过正则、哈希值匹配等方式防止某些 SQL 的执行
- 官方不支持跨两位版本号进行升级，而 0.15 为当时最新的 Release 版本，选用该版本利于后期版本升级
- 可通过资源标签的方式将一个 Apache Doris 集群中的 BE 节点划分为多个资源组，实现多租户和资源隔离
- 该版本提供了官方认可的 Flink-Doris-Connector/Spark-Doris-Connector/DataX Doriswriter 等插件，利于 ETL 流程建设

2、验证阶段

该阶段主要是为了**二次验证**官方文档中介绍的功能是否满足我们的实际运用场景，比如生态扩展中的 Connector、外表联邦查询、各种 Load 方式、多租户隔离及物化视图等。

3、压测阶段



压测阶段首先进行数据生成，数据集选用的是 TPC-DS 数据，接着根据 Doris 的特性对 DDL 和 SQL 等规则进行对应调整，最后通过脚本将数据导入到 Apache Doris 存储中，再通过自动化脚本进行查询及导入压测，最终将压测结果输出到 MySQL 表中，量化为图表进行展示。**下方为本阶段的基本配置及压测过程介绍：**

– 硬件环境

- 内存：256G
- CPU：96C
- 硬盘：SSD 1.92T * 8

– 软件环境

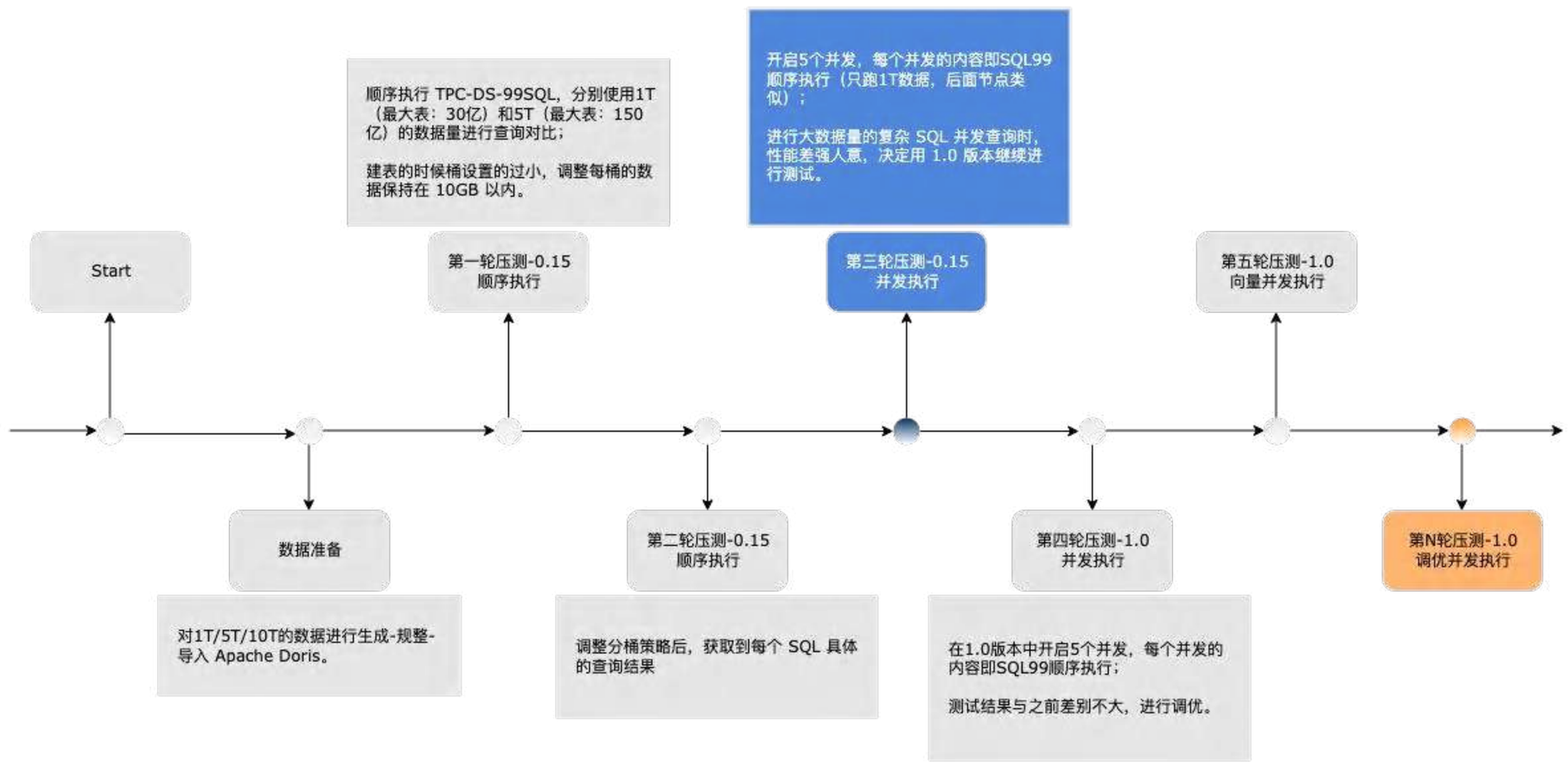
- Apache Doris 版本：0.15-release/1.0-release（该阶段进行时，1.0-release 版本刚好发布）
- Apache Doris 集群：3 FE + 9 BE
- 系统：CentOS Linux release 7.9.2009

– 数据集信息

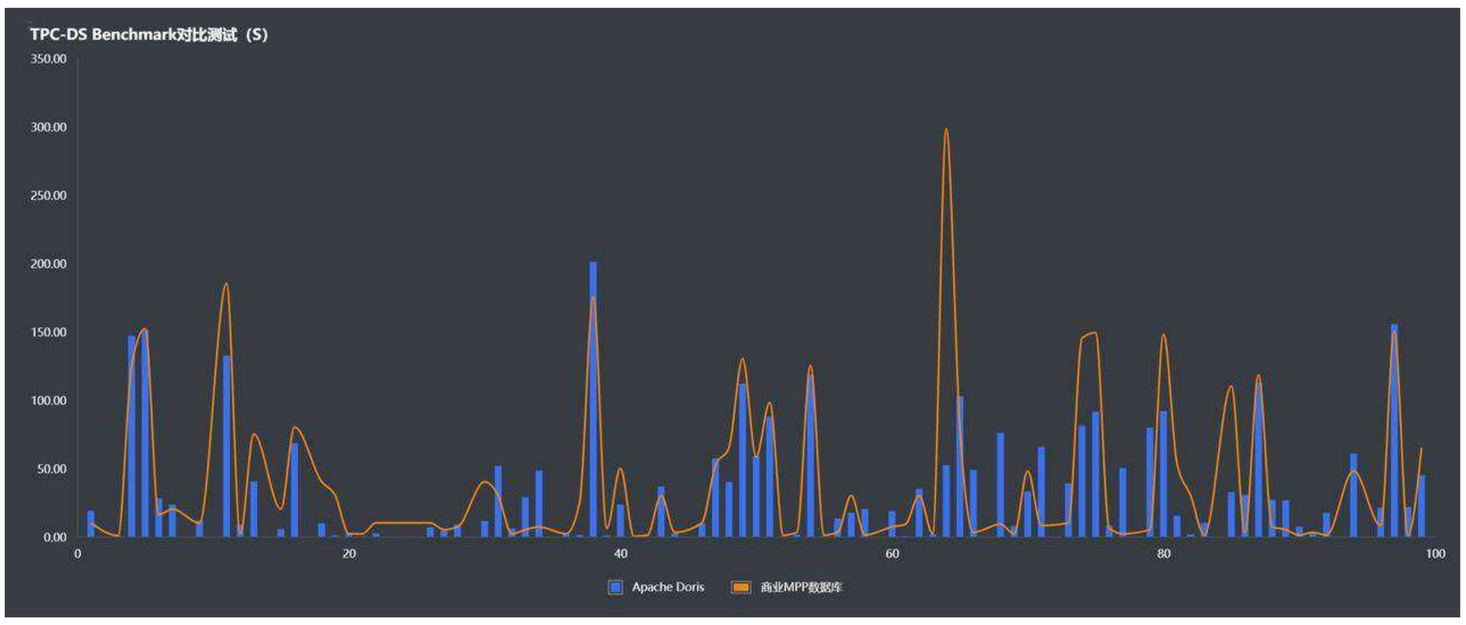
我们生成了 1T、5T、10T 的 TPC-DS 数据集，1T 的数据集约有 30 亿数据量。

Table	Sample Row Count Of These TB			Dimension/Fact Table	Notes
	1T	5T	10T		
store_sales	2949109737	14745548685	29491097370	Fact Table	订单信息
此处省略其它tpcds明细和维表信息					
item	18000	90000	180000	Dimension Table	商品信息

查询压测



压测过程中，最初使用 0.15-release 版本进行测试，正巧 1.0-release 版本发布，后决定更换为 1.0-release 版本进行后续的压测。下图是基于 1T 的 TPC-DS 数据在同等硬件配置环境下和某商业 MPP 数据库的对比结果：

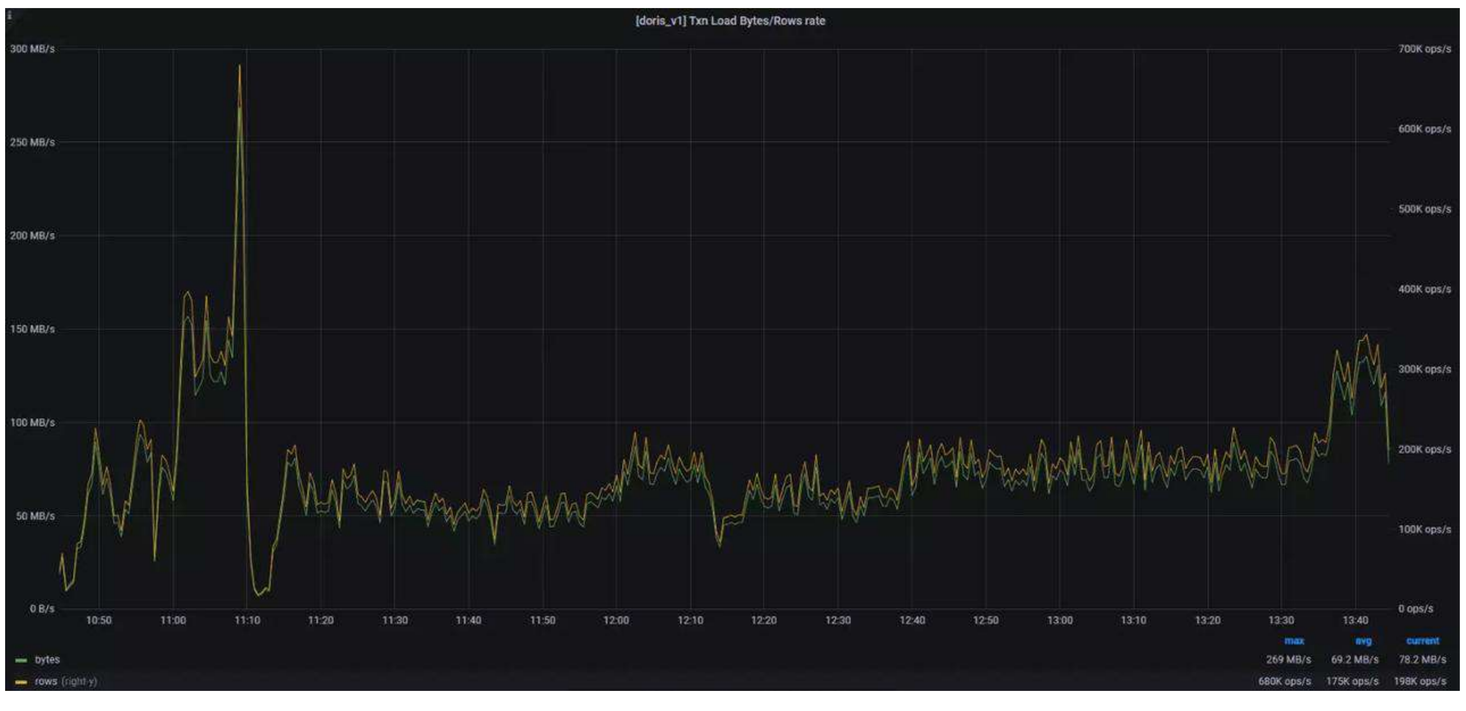


如图所示，Apache Doris 的查询压测性能优异，有着明显的性能优势，**作为开源产品能够达到这样的效果是非常优秀也是十分不易的。**

导入压测

- 导入方式：通过 DataX Doriswriter 以 Stream Load 方式进行写入压测
- 数据来源：为避免因 Source 端原因影响写入时效，选择 100 张相同大表，即 100 个并发从内网 Hive 中导入（例如 tpcds-ds 的 store_sales_1t 表）
- 数据模型：选用 Unique 模型（模拟 ODS 层），同时为充分考虑 Compaction 性能及小文件场景，每张表设置 70 个 Tablet

经调整优化后，**最大写入时效为 269 MB/S & 680K ops/s，平均写入时效 70 MB/S & 180K ops/s，写入时效大幅提升。**



4、上线阶段

该阶段主要是确认 Apache Doris 上线需要的检查清单、预调参数、BE 资源组规划及用户权限的划分。

- 检查清单：包括但不限于 FE & BE 端口、网络检查及 Apache Doris 的一些功能性验证，例如读写是否正常等。
- 预调参数：确认优化后的 FE & BE 参数是否配置，是否开启 `global enable_profile` 、动态分区以及数据盘保存位置是否有误等。
- BE 资源组：由于我们需要通过 Apache Doris 的多租户特性对不同的用户进行资源隔离，所以需要提前规划好每个 BE 节点对应的资源组。
- 用户权限：对于不同的用户群体需要提前规划好权限范围，比如分析师开发只需要 `SELECT_PRIV` 权限，而 ETL 工程师需要 `SELECT_PRIV` 、 `LOAD_PRIV` 和 `CREATE_PRIV` 权限。

5、宣导阶段

该阶段主要是输出前面各阶段的 TimeLine、总结以及上线后使用 Apache Doris 的注意事项说明，比如我们用到多租户隔离，那么 DDL 建表时则需要在 Properties 中显示指定各副本对应的资源组：

```
1  create table zt_table
2  ......
3  properties(
4      "replication_allocation"="tag.location.group_a:1, tag.location.group_b:1, tag.location.group_c:
   1"
5  )
```

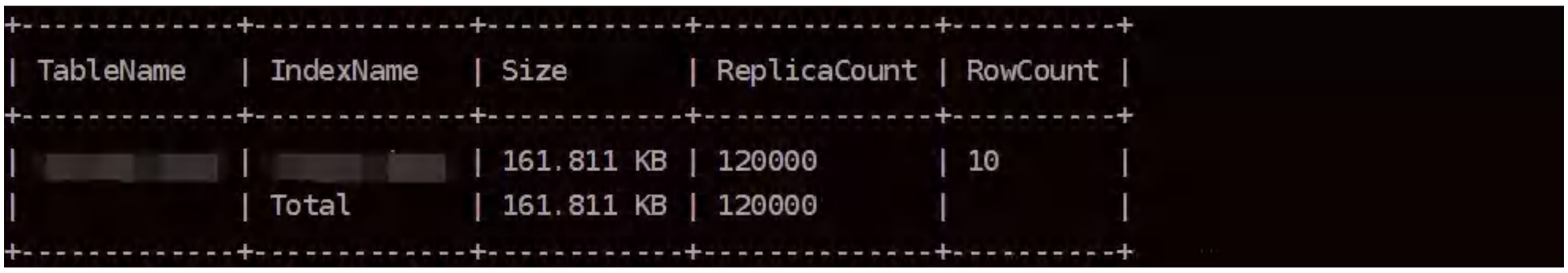
6、运行阶段

Tablet 规范问题

问题描述：上线运行一段时间后，随着越来越多的数据增长，集群每次重启后一周左右，读写就会开始变得越来越慢，直到无法正常进行读写。

问题处理：

- 经过对生产和 UAT 环境的对比测试以及对数仓表的 Schema 的分析，我们发现有些表数据并不大，但是 Bucket 却设置的非常大。



TableName	IndexName	Size	ReplicaCount	RowCount
[redacted]	[redacted]	161.811 KB	120000	10
	Total	161.811 KB	120000	

- 结合 `show data from database` 命令，我们将整个集群所有表的 Bucket 信息罗列出来，明确了大部分表的 Bucket 设置的不合理；而当前集群共 20T 左右数据，平均 1T 数据近 10W 个 Tablet，这就会导致小文件过多，造成 FE 元数据负载过高，从而影响导入和查询性能。

- 定位原因后与社区小伙伴二次确认，并根据官方建议将 Bucket 设置不合理的表全部调整，调整后集群逐步恢复读写正常。（即将发布的 **Apache Dorie 1.2.2 版本将推出 Auto Bucket 动态分桶推算功能**，可以根据历史数据和机器数目自动推算新建 Partition 的分桶个数，保证分桶数始终保持在合理范围内，可有效解决上述问题）

问题小结：

- Tablet 数 = 分区数 * 桶数 * 副本数
- 1TB 数据的 Tablet 数量控制在 8000 个左右（三副本控制到 2.4W 左右）
- 建议大表的单个 Tablet 存储数据大小在 1G–10G 区间，可防止过多的小文件产生
- 建议百兆左右的维表 Tablet 数量控制在 3–5 个，保证一定的并发数也不会产生过多小文件

集群读写优化

问题描述：1.1.3 release 版本中，高并发的同时进行 Stream Load、Broker Load、insert into 和查询时，读写会变得非常慢，如下图 11/01 19:00 并发上来后的 Txn Load 所示：



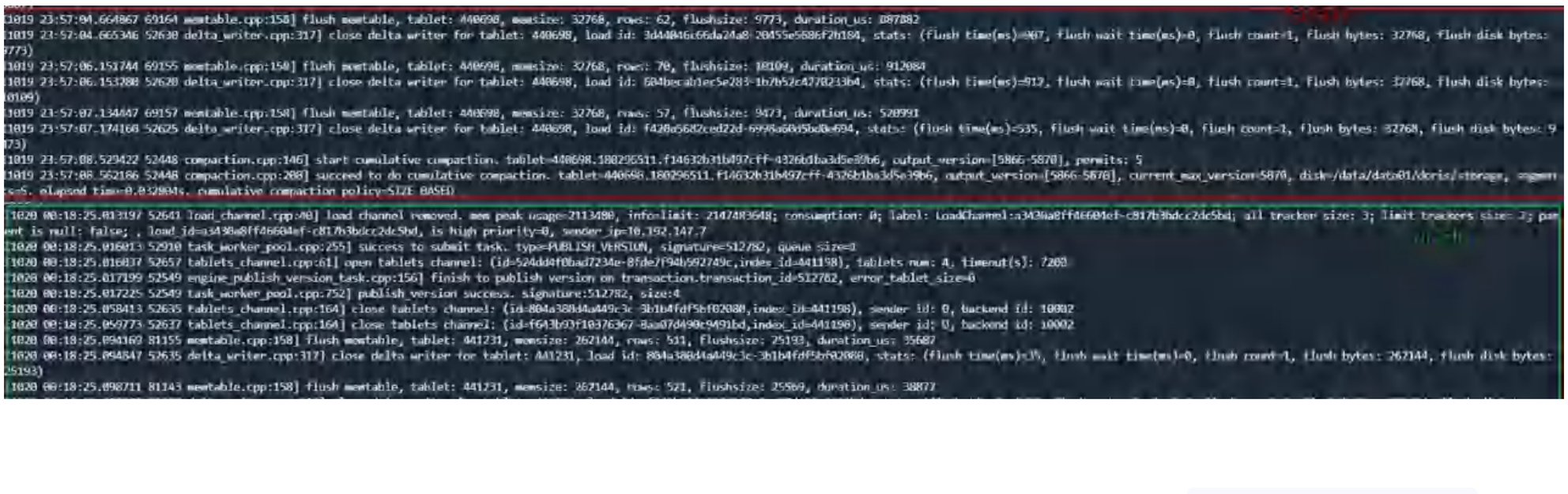
问题处理：

1. 我们进行了十几轮对比测验，结论如下：

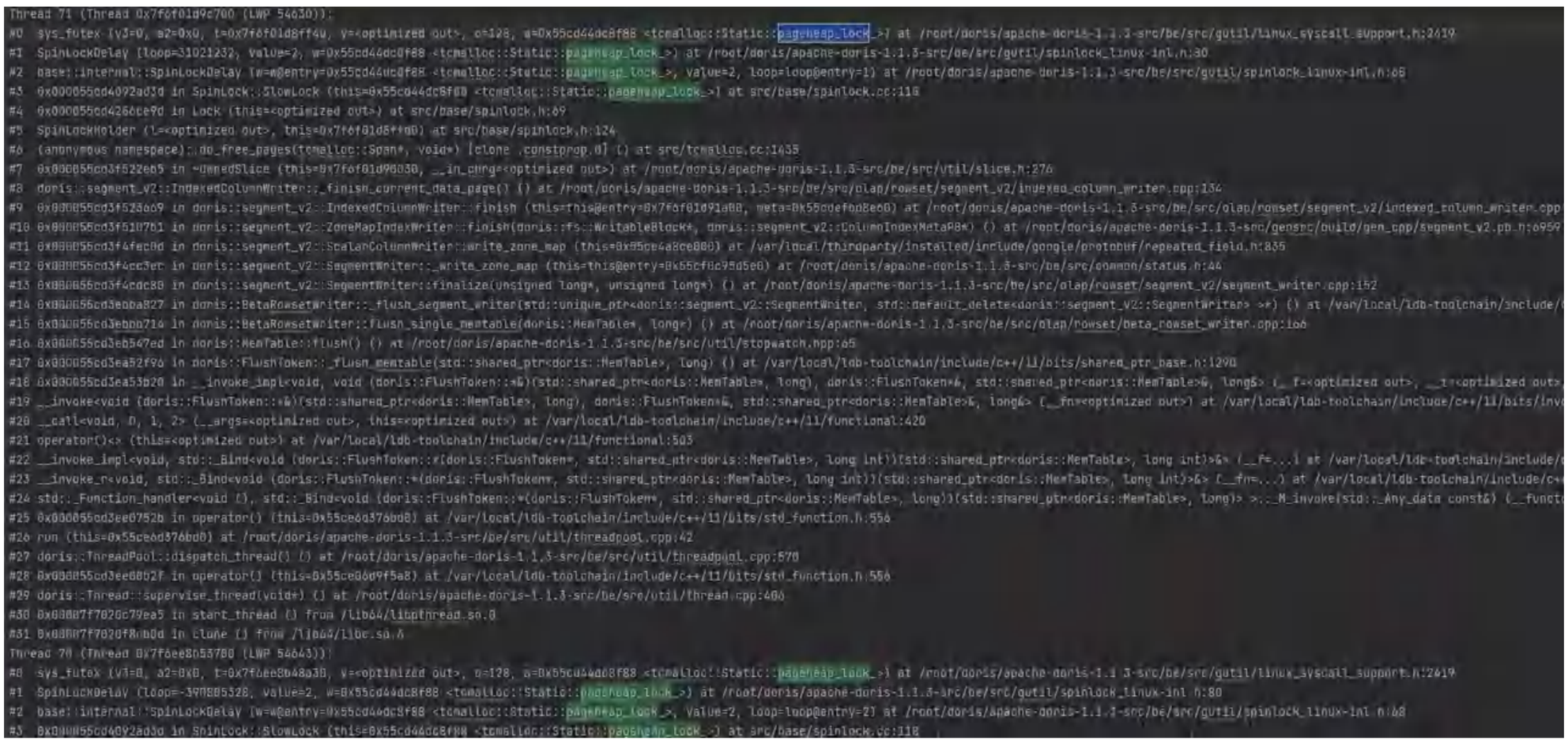
- 写入速度与并发的增长成反比（但不会骤变，而是缓慢变化）
- 单表 Bucket（Tablet）设置过大会导致集群写入速度骤减；例如 A 库的 TA 表，设置 80 个 Bucket 时，启动相关 Flink Sink Job 就会导致集群整体写入速度迅速变慢，降低 Bucket（9~10个）时写入恢复正常。
- `insert into select` 的 ETL 任务与 Stream Load 写入任务会进行资源抢占，同时并发运行会使整个集群读写变慢。

测验维度	测验方式	结果	附件
	评估标准：datax单线程实时写5000条一批数据（streamload）时效小于500ms即为正常		
并发上限	停止所有dw etl任务，10~并发↓	单线程任务写入时效无改善	
并发上限	停止所有flink per job任务，180并发↓	写入速度明显提升，2719ms/685ms/202ms；300KB~1M/s+，9500 r/s	
并发上限	停止所有flink session任务，130并发↓（当前共0并发）	写入完全恢复（停止所有写入任务），平均时效170ms+	
并发上限	起A和B两个整库同步的flink per job任务，16并发↑（当前共16*2并发）	写入时效非常慢，50KB/s, 500 r/s	
并发上限	**停止A整库同步flink per job任务，16并发↓（当前共16并发）**	写入完全恢复，平均时效170ms+	
并发上限	再起一个A整库同步flink per job任务，16并发↑（当前共16*2并发）	写入变慢，平均时效1000ms+	
并发上限	停止A整库同步任务，16并发↓（当前共16并发）	3分钟后写入完全恢复，平均时效170ms+	
并发上限	再起一个C整库同步任务，16并发↑（当前共16*2并发）	写入正常速度不变，平均时效170ms+	
并发上限	再起一个D整库同步任务，16并发↑（当前共16*3并发）	整体写入速度微浮下降，平均时效2000ms~	
并发上限	再起一个A整库同步任务，16并发↑（当前共16*4并发）	写入时效非常慢，50KB/s, 500 r/s	
并发上限	**停止A整库同步任务，16并发↓（当前共16*3并发）**	整体写入速度恢复，平均时效2000ms~	
并发上限	再起一个A整库同步任务（其中一张表TA的bucket由80调成9个），16并发↑（当前共16*4并发）	整体写入速度不变，平均时效2000ms~	
并发上限	0点后etl任务与ods任务同时并发运行	stream load任务写入时效逐渐降低，直到50KB/s, 500 r/s	

2. 通过 `be.INFO` 发现，80 个 Bucket 表写入某个 Tablet 的 `memsize/rows/flushsize/duration` 数值比 10 个 Bucket 写入时的数值呈数倍之差，即 80 个 Bucket 表的数据写入时效无论 Memsize 还是 Flushsize 都非常小、但花费时间却很长。



3. 同时收集 Pstack 日志，经过分析可以确定，Tcmalloc 在频繁地寻找 `pageheap_lock` ，导致高频竞争锁从而降低了读写性能。



4. 于是，进行如下参数调整：

```
减少doris_be进程内存返回给linux系统的频率，从而减少tcmalloc频繁竞争锁的情况
tc_use_memory_min = 207374182400
tc_enable_aggressive_memory_decommit = false
tc_max_total_thread_cache_bytes=20737418240
```

5. 调参并滚动重启 BE 后，集群状况如下图所示：

18:50 前将 Broker Load、insert into 和查询任务同时开启，18:50 后将 Stream Load 任务也开启（包括 80 bucket的表），集群整体的读写性能不仅没有下降，反而 Stream Load 时效突破了压测阶段的最大值 **269 MB/S&680K /ops/s** ，并且持续稳定。



问题小结：

使用 Apache 1.1.3 及以上版本，非常推荐调整 Tcmalloc 相关参数，减少 `doris_be` 进程与系统之间的内存申请回收过程，可明显减少锁竞争的现象，大大提升读写性能和集群稳定性。（从 Apache Doris 1.1.5 版本开始，**增加了Tcmalloc 简化配置**，可将众多 Tcmalloc 参数归约到参数 `memory_mode` 中，compact 为节约内存模式，performance 为性能模式，用户可根据实际需求进行调整）

总结收益

当前 Apache Doris 的生产集群为 3 FE + 9 BE 组合， 已导入集团存量和增量数据的 60%以及部分 DW 数据生成，3 副本共占 44.4TB 的存储。



依赖 Apache Doris 自身优异特性及其生态圈帮助我们快速构建了一套新的流批一体数据架构，平均每天实时入库的**数据量达到上亿规模**，同时支持**上万个调度任务**平稳运行，相比早期架构单表**查询效率提升近 5 倍**，数据导入效率**提升近 2 倍**，内存资源使用率显著减少。除此之外，Apache Doris 以下优势也是我们快速构建数据架构的重要推动力：

- 扩展表：联邦查询的设计，便于集成其它存储
- 数据表设计：丰富的数据模型，可快速应对不同的数据需求。

- 数据查询：不同的 Join 算子结合自身完善的优化器，让查询快而稳。
- 架构设计：架构清晰明了且运维简单，大大地降低了我们的运维成本。
- 数据导入：各种 Load 方式及 Connector 的扩展，基本涵盖大部分的数据同步场景应用。
- 活跃度：社区高度活跃，SelectDB 为 Apache Doris 社区组建了一支专职技术支持团队，疑难杂症基本能在 12H 内快速响应并有社区小伙伴跟进和协助解决。

未来规划

结合当下业务场景的考虑，未来我们将引入数据湖进行非结构化和结构化数据一体存储，进一步完善流批一体架构。同时也会将 Apache Doris 回归它最本质的定位，专注于 OLAP 分析场景，并通过 Apache Doris 统一湖仓查询引擎层，发挥其最大的功效。



最后，非常感谢 Apache Doris 社区和 SelectDB 团队的张家锋、曲率和杨勇强等小伙伴对我们无私的技术支持，未来我们也将持续参与 Apache Doris 社区建设中，贡献绵薄之力。祝 Apache Doris 社区和 SelectDB 越来越好，日臻完善！

感谢您的下载与阅读！

如果你是 Apache Doris 的爱好者或使用者
欢迎扫码加入社区技术交流群！