

# 实时数仓 技术入门一本通

MC-Hologres带你搞定10+个实战场景



- 多位资深技术专家深度解析
- 融合实时数仓&报表、数据仓库、传统OLAP迁移等多个热门场景
- 助力实现服务和分析一体化实时数仓





Hologres 用户交流群



Hologres 官网



Hologres 开发者社区技术圈



阿里云开发者“藏经阁”  
海量免费电子书下载

# | 目录

Hologre 产品介绍与技术揭秘	4
快速上手 Hologres	16
Hologres+Flink 实时数仓详解	26
MaxCompute+Hologres 数据仓库详解	33
开源 OLAP 升级 Hologres 详解	46

# Hologre 产品介绍与技术揭秘

概要：近年来，随着数据实时化的诉求加剧，催生了一系列的实时数仓架构，Lambda 架构也应运而生，但是随着场景的复杂度和业务多维需求，Lambda 架构的痛点也越来越明显。HSAP 的理念则是服务分析一体化，在本文中，来自阿里巴巴的资深技术专家将会深度剖析 HSAP 技术实现 Hologres 的设计原理，解读其产品典型场景。

作者 | 仙隐（金晓军） 阿里巴巴资深技术专家

## 一、传统数据仓库

目前来说，大数据相关的业务场景一般有实时大屏、实时 BI 报表、用户画像和监控预警，如下图所示。

- 实时大屏业务，一般用在公司领导做决策的辅助工具，在对外展示，比如实时成交额等场景也会经常用到，是一种展示公司实力的方式。
- 实时 BI 报表是运营和产品经理经常用到的一个业务。
- 用户画像常用在广告推荐场景中，通过更详细的算法给用户贴上标签，使得推荐算法更加有针对性，更加有效。
- 预警监控，比如对网站、APP 进行流量监控，在达到一定阈值的时候可以进行报警。

### 典型业务场景列举



实时大屏



实时BI报表

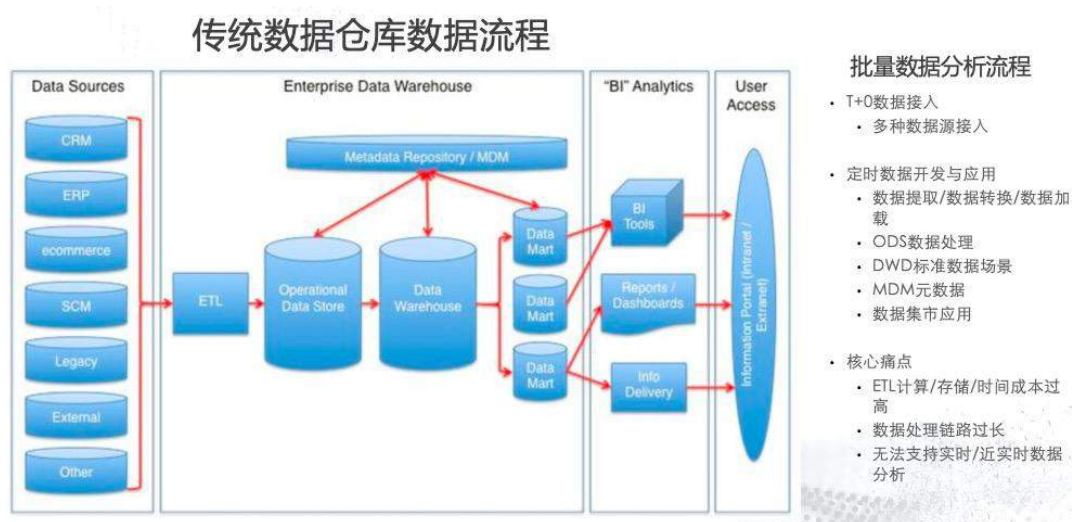


用户画像



监控预警

对于上面这些大数据业务场景,在很早之前业界就开始通过数据仓库的建设来满足这些场景的需求,比较传统的是如下图所示的离线数据仓库,其大致流程就是:首先,将各类数据收集起来;然后经过 ETL 处理,再经过层层建模对数据进行聚合、筛选等处理;最后在需要的时候通过应用层的工具对数据进行展现,或者生成报表。



上面这种方式虽然可以对接多种数据源,但是存在一些很明显的痛点:

- ETL 逻辑复杂, 存储、时间成本过高;
- 数据处理链路非常长;
- 无法支持实时/近实时的数据, 只能处理 T+1 的数据。

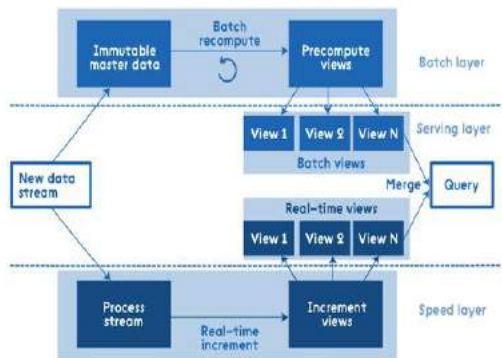
## 二、Lambda 架构

随着实时计算技术的兴起,出现了 Lambda 架构。Lambda 架构的原理如下图所示,其思路其实是相当于在传统离线数仓的基础上再加上一个处理实时数据的层,然后将离线数仓和实时链路产生的数据在 Serving 层进行 Merge,以此来对离线产生的数据和实时产生的数据进行查询。

从 2011 年至今, Lambda 架构被多数互联网公司所采纳,也确实解决了一些问题,但是随着数据量的增大、应用复杂度的提升,其问题也逐渐凸显,主要有:



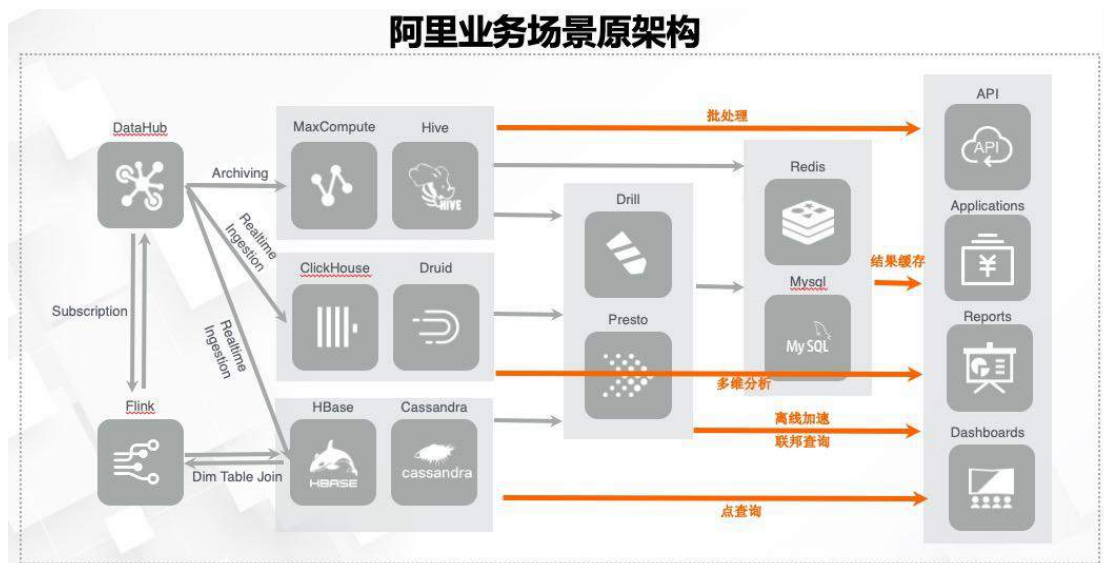
- 由多种引擎和系统组合而成，开发和维护成本高，学习成本高；
- 数据在不同的 View 中存储多份，空间浪费，数据一致性的问题难以解决；
- 从使用上来说，Batch，Streaming 以及 Merge Query 等处理过程中均使用不同的 language，使用起来并不容易；
- 学习成本非常高，增大了应用成本。



#### Lambda架构的问题：

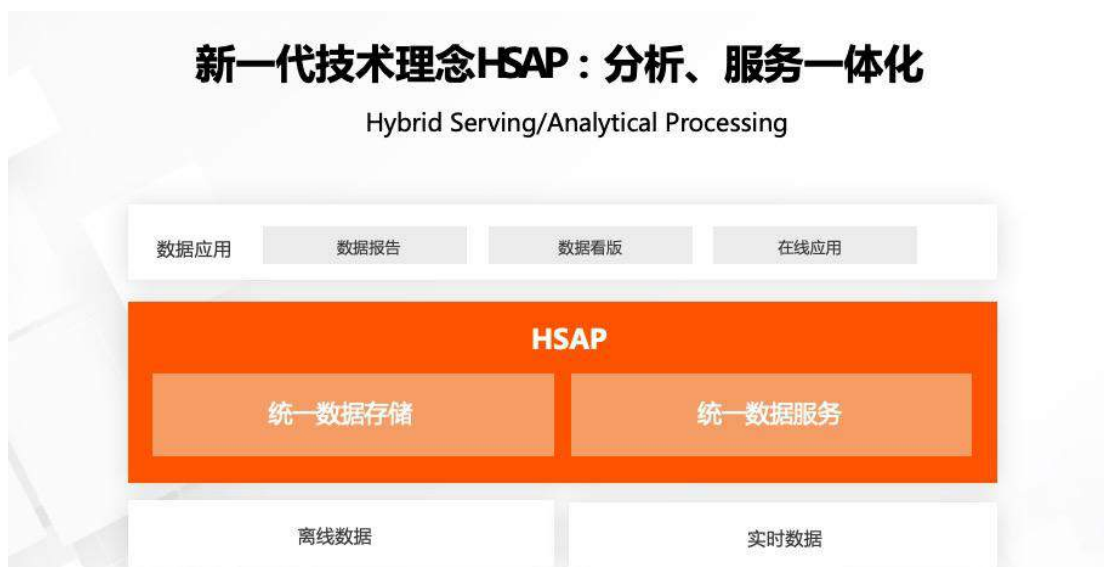
1. 由多种引擎和系统组合而成，开发和维护成本高，学习成本高
2. 数据在不同的View中存储多份，空间浪费，数据一致性的问题如何解决
3. 从使用上来说，Batch，Streaming及Merge Query均使用不同的language，使用起来并不容易

下图是阿里巴巴在 2011 年到 2016 年间沉淀下来的一套实时数仓架构，其本质上也是 Lambda 架构，然而随着业务量的增长，随着数据的增长，关系复杂度越来越大，成本急剧增加。上面讲到的问题，在阿里内部也都遇到了，因此，我们迫切的需要一种更优雅的方案去解决类似的问题。



### 三、HSAP：分析、服务一体化

基于上述背景，我们提出了 HSAP 解决方案，HSAP 是指 Hybrid serving and analytical processing，我们的理念是能够支持这种很高 QPS 的 Serverless 场景的查询写入，并且将复杂的分析场景在一套体系里面完成。那么其核心是什么呢？首先，要有一套非常强大的存储，能够把实时的数据和离线的数据存储进来，实现数据的通存，同时还要有一种高效的查询服务，能够支持高 QPS 的查询，支持复杂的分析以及联邦查询和分析，这样的话就可以把离线数据和实时数据都导入到系统里去，然后将前端的数据应用，比如 BI 报表和一些在线服务，对接到系统中去。如此，上面提到的架构复杂的问题，其实就可以迎刃而解。我们把这样的设计理念叫做 HSAP 设计理念。



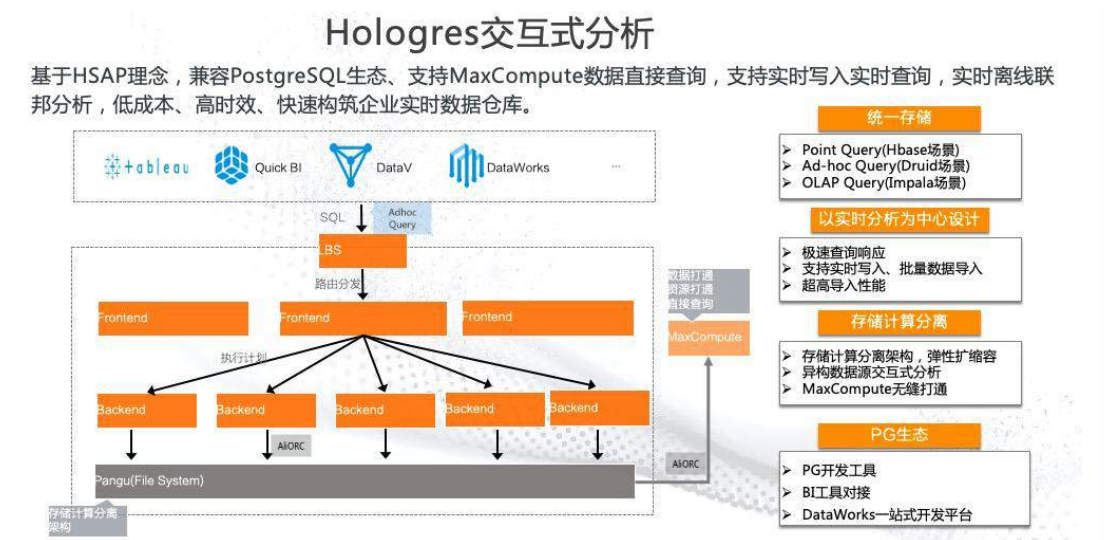
### 四、Hologres 简介

有了上面 HSAP 的设计理念，我们要去做相应的产品来实现这种理念，于是有了 Hologres。Hologres 这个词是 holographic 和 Postgre 的组合，Postgre 指的是兼容 PostgreSQL 生态，holographic 指的是全息，也就是全部信息，我们希望通过 Hologres 对数据进行全息的分析，并且能够兼容 PostgreSQL 生态。

用一句话来总结，Hologres 就是基于 HSAP 理念，兼容 PostgreSQL 生态、支持 MaxCompute 数据直接查询，支持实时写入实时查询，实时离线联邦分析，低成本、高时效、快速构筑企业实时数据仓库，其架构图如下所示。

整体来看，其架构非常简单，是存储计算分离的架构，数据全部存在一个分布式文件系统里面，在阿里内部指的是盘古分布式文件系统；服务节点叫 Backend，真正去接收数据，存储和查询，并且能够支持数据的计算；Frontend 接收路由分发过来的 SQL，然后生成真正的物理执行计划，发布到 Backend 做分布式的执行；在接入端由 LBS 来做相应的负载均衡任务。

下图中黄色部分均部署在容器中，整个分布式系统可以做到高度容错。同时，因为兼容 PostgreSQL 生态，所以一些开源或者商业化的 BI 工具，或者 WebIDE 可以直接跟 Hologres 进行对接。



Hologres 有着强大的性能，主要包括：

## 1) 统一存储

Hologres 能够满足多种场景中的数据存储方式，支持 Point Query (Hbase 场景)、Ad-hoc Query (Druid 场景) 和 OLAP Query (Impala 场景) 等。

## 2) 以实时分析为中心设计

Hologres 的设计理念就是为了快，支持压秒级的数据实时分析，极速查询响应，还支持实时写入、批量数据导入，拥有超高导入性能。



### 3) 存储计算分离

Hologres 采用存储计算分离架构，用户可以根据需求进行弹性扩缩容，如果存储用的比较多可以多买存储，CPU 用的比较多可以多买 CPU。另外，Hologres 支持异构数据源交互分析以及离线数据和实时数据的联邦查询。Hologres 已经和 MaxCompute 无缝打通，能够直接在 Hologres 中对 MaxCompute 表进行查询。

### 4) PG 生态

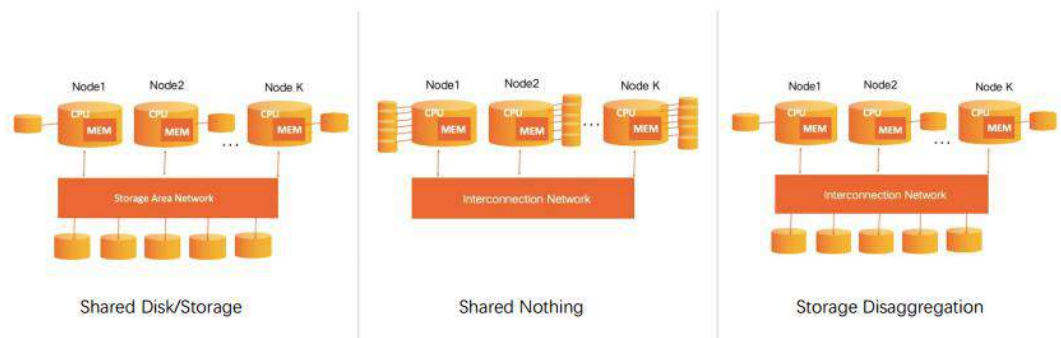
Hologres 兼容 PostgreSQL 生态，能够与 PG 开发工具、BI 工具进行对接，同时在阿里云提供了一套原生的开发平台，能够在 WebIDE 中进行 SQL 的开发，并且支持任务的调度。

#### (一) 存储计算分离

在传统的分布式系统，尤其是分布式存储中，常用的架构有如下三种。其中 Shared Disk/Storage 就是在存储集群上挂载了很多盘，每个计算节点都可以访问这些盘；Shared Nothing 架构就是每个计算节点自己挂载存储，节点之间可以通信，但是各个节点之间的盘不共享，存在资源浪费的情况；Storage Disaggregation 就是相当于把存储集群看做一个大的磁盘，每个计算节点都可以访问，且每个计算节点都有一定的缓存空间，可以对缓存数据进行访问，也无需关心存储集群的管理，这种存储计算分离的架构便于灵活扩容，能够有效节省资源，Hologres 就是采用的这种架构。

阿里云 开发者社区

## 存储计算分离

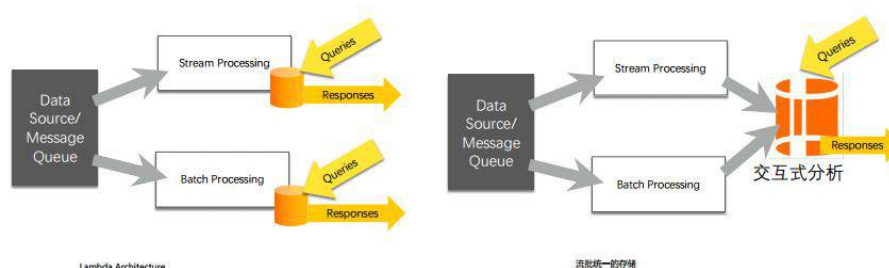


#### (二) 流批统一的存储

Hologres 定位是能够做离线数据和实时数据的存储。对于典型的 Lambda 架构，是将实时数据通过实时数据的链路写入到实时数据存储中，离线数据通过离线数据的链路写入到离线存储中，然后将不同的 Query 放到不同的存储中，再做一个 Merge。对于 Hologres，如下图所示，数据收集之后可以走不同的处理链路，但是处理完成之后的结果都可以写入 Hologres 中，这样就解决了数据的异质性问题，也不需要去区分离线表和实时表，降低了复杂度，也大大降低了使用者的学习成本。

阿里云 开发者社区

## 流批统一的存储



Hologres 底层支持行存储和列存储两种文件格式，对于两者的处理也有略微不同，具体如下图所示。数据写入的时候先写 log，log 是存储在分布式文件系统上的，保证整个服务的数据不会丢失，因为即便服务器挂掉也可以从分布式系统中恢复。Log 写完之后再写 MemTable，就是内存表，这样子才认为是数据写入成功。MemTable 有一定的大小，写满了之后会将其中的数据逐渐 Flash 到文件中，文件是存储在分布式系统中的。而对于行存储和列存储的区别就在 Flash 到文件的这个过程中，这个过程会将行存表 Flash 成行存储的文件，列存表会 Flash 成列存文件。在 Flash 的过程中会产生很多小文件，后台会将这些小文件合并成一个大文件，这里也会有所不同，大家可以自行查阅文档了解。

阿里云 开发者社区

## 流批统一的存储

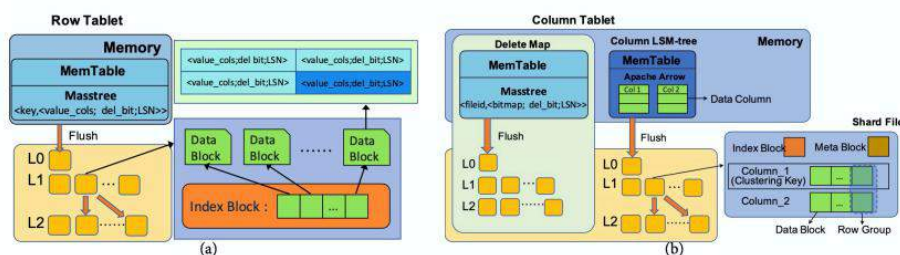


Figure 4: (a) The structures of a row tablet, and (b) the structures of a column tablet

### （三）极致查询性能

如果希望得到一个快速的系统，除了高性能的存储，更少不了高性能的查询。Hologres 其自身有着极致的查询性能，主要包括：

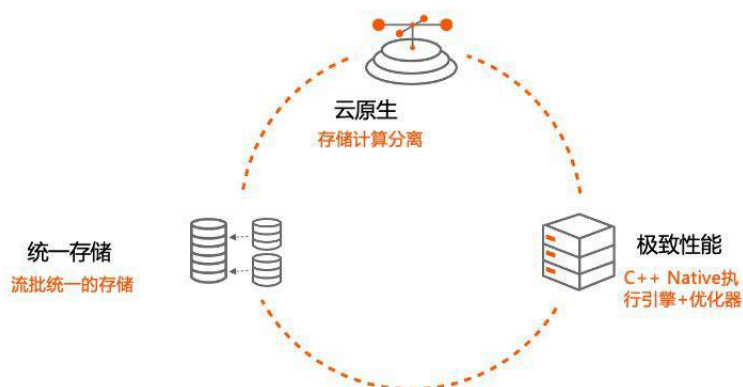
- 同时支持行存列存；
- 向量化等执行层优化；
- 高并发充分利用计算资源；
- 纯 C++ 实现保证稳定的低延迟；
- 优化的调度保证 SLA；
- 基于成本的优化器，针对存储特点高度优化。

## 极致查询性能



总的来说，Hologres 的技术亮点包括云原生、统一存储和极致性能，具体的细节大家可以参考 <http://www.vldb.org/pvldb/vol13/p3272-jiang.pdf>。

## 技术亮点



## 五、Hologres 典型应用场景

从 Hologres 的设计理念来看，其典型的应用场景主要有以下三类：

### （一）离线数据查询加速

这种场景下，就是将以前在离线系统里面的数据和查询通过 Hologres 来进行加速，来达到量级提升的加速效果。通过 Hologres，对离线数据秒级交互式查询响应，无需额外 ETL 工作，便捷地把冷数据转换成易于理解的分析结果，提升企业决策效率，降低时间成本。

#### 典型应用场景-传统数仓查询加速方案



#### MaxCompute数仓查询加速解决方案

阿里云大数据计算服务MaxCompute经过十年磨砺，已成为阿里巴巴集团数据中台的计算核心和阿里云大数据的基础服务。通过交互式分析引擎加速Maxcompute里的海量数据进行高性能低延迟的分析查询，为业务发展寻找新的突破点。

#### 客户收益

**分析报表实时响应**  
MaxCompute数据秒级交互式查询响应，无额外ETL工作，便捷地把冷数据转换成易于理解的分析结果。

**低成本**  
直接连接访问Maxcompute项目，去除传统方案中不必要的数据导出操作，降低存储成本和维护成本。

**简单易用**  
兼容PostgreSQL，上手快，分析工具可无缝对接。

### （二）实时数仓

在下图所示场景中，实时数仓通过搭建用户洞察体系，实时监测平台用户情况，并从不同视角对用户进行实时诊断，进而采取针对性的用户运营策略，从而达到精细化用户运营目的，助力实时精细化运营。这种场景下，离线数据和实时数据都可以写入到 Hologres 中，不需要再通过 Lambda 架构做多个系统中数据的 Merge，减少异质性，降低学习成本。

#### 典型应用场景-实时数仓方案



#### 实时数仓解决方案

实时数仓平台旨在通过搭建用户洞察体系，实时监测平台用户情况，并从不同视角对用户进行实时诊断，进而采取针对性的用户运营策略，从而达到精细化用户运营目的。

#### 客户收益

**分析报表实时响应**  
数据实时采集、实时清洗，交互式分析引擎提供毫秒级交互式查询响应。

**实时业务洞察**  
实时用户大屏，实时用户圈定和定向投放，精准触达。

**简单易用**  
兼容PostgreSQL，上手快，分析工具可无缝对接。

### （三）实时离线联邦计算

如下图所示场景，基于实时计算引擎 RealtimeCompute、离线数仓 MaxCompute 和交互式分析，从商业逻辑出发，实现离线数据分析实时化，实时离线联合分析，构筑实时全链路精细化运营。

#### 典型应用场景-离线+实时联邦分析方案



#### 实时、离线分析引擎打造精细化运营解决方案

企业的搜索推荐和广告系统积累了海量用户行为数据和交易数据，为快速响应多元化的运营诉求，平滑应对电商业务各类大促的数据洪峰，基于阿里实时计算引擎 RealtimeCompute、离线数仓 MaxCompute 和交互式分析引擎，从企业商业逻辑出发，构筑实时、离线融合的全链路精细化运营方案。

#### 客户收益

**统一的实时多维分析方案**  
使用阿里云大数据产品，构建统一实时多维分析平台，大幅降低资源消耗，提升数据开发效率。

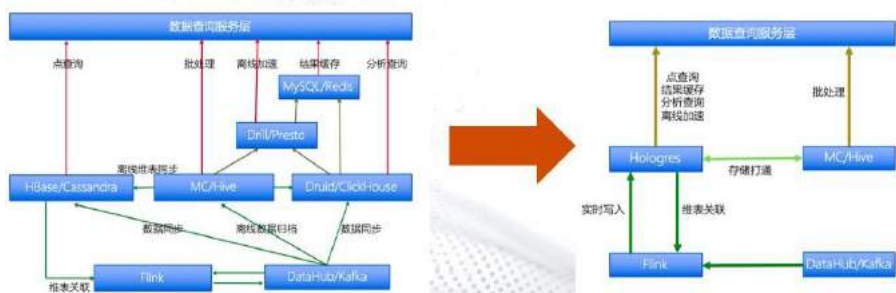
**提升业务洞察能力**  
解决PB级数据实时即席分析性能问题，亿级数据量下的UV计算秒级响应，轻松达成用户洞察分析和增长目标。

**快速响应业务需求**  
轻松实现全渠道、全触点、全洞察、全链路数据打通，快速提升数据分析需求的“随机应变”能力。

## 六、典型客户案例

下图所示的是阿里数据业务基于 Hologres 的架构演进，从下图（左）可以看出，在没有 Hologres 之前，整个数据链路以及架构非常的复杂，比如图中所示的 5 条红线代表的 5 个链路；在有了 Hologres 之后，整个数据链路就变得清晰明了，数据进入系统之后通过 Flink 做实时的 ETL，然后直接写入到 Hologres 中，之后可以通过 Hologres 进行归档、历史数据的加速等，数据服务也非常简单，点查询、分析查询等操作都可以通过 Hologres 来完成，整个链路就变的非常简单，大大降低了架构复杂度，降低了成本。

#### 阿里数据业务基于Hologres的架构演进





下面三张图展示的分别是 Hologres 在基于实时分析引擎搜索推荐实时分析和算法应用、基于实时分析引擎行业精细化运营、基于实时分析引擎构建安全风控系统实时分析场景中的应用。

### （一）基于实时分析引擎搜索推荐实时分析和算法应用

本案例是阿里巴巴的一个实时搜索推荐的场景，其特点是数据量相当大，单日数据达到了 PB 级别，且写入数据量也非常大，QPS 也非常高。该场景对数据的灵活性要求非常高，分析场景多样化，在应用了 Hologres 之后，可以将原来众多的业务数据通过实时 ETL 导入 Hologres 中（离线数据通过离线 ETL 导入离线数仓再加载到 Hologres 中），然后再通过 JDBC 查询进行数据的应用，比如分析报表、实时大屏等。通过 Hologres 的引入，开发效率大大提升，且成本降低了一半，整个业务系统的性能也得到了很大的提升，数据生产到消费端到端秒级实时，交互式查询毫米级返回。

#### 典型客户案例1

**荐**

基于实时分析引擎搜索推荐实时分析和算法应用

##### > 业务特点

- 数据量大，单日PB级存储，单表总条数10000亿+；
- QPS高，峰值写入TPS 1亿+，峰值查询qps 200+；
- 数据灵活性要求高，分析场景多样化，固定条件高频分析、非固定条件多维查询

##### > 效率

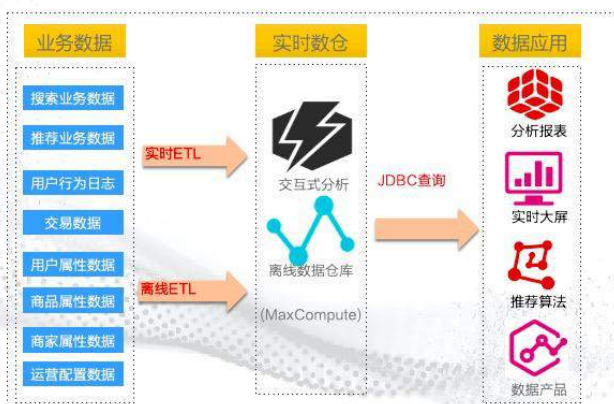
- 开发资源有限，从数据开发、校验、压测到上线，仅一个月

##### > 成本

- 计算资源有限，一套套数据多路应用，成本降低50%

##### > 性能

- 数据生产到消费端到端秒级实时；
- 交互式查询毫米级返回(含UV计算)



### （二）基于实时分析引擎行业精细化运营

该案例针对的是行业精细化运营的场景，使用对象一般是运营小二或者商家。这种场景下，通过将业务数据数据通过 ETL 导入到 Hologres 中，然后去做多维的筛选，提供给用户分析报表或者进行实时大屏的展现，解决了之前各种 OLAP 产品 UV 计算性能差，使用 Blink 实时作业计算 UV 成本高、业务灵活性差等痛点，实现了 UV 计算的秒级响应。

## 典型客户案例2



基于实时分析引擎行业精细化运营



### (三) 基于实时分析引擎构建安全风控系统实时分析

本案例是一个风控场景，通过将实时、离线数据写入到 Hologres 中，加速查询的过程，最终产生相应的分析报表或者实时预警的规则，通过 Hologres 的引入，节约了大量成本，大大降低了学习成本，且与 BI 工具可以无缝对接。

## 典型客户案例3



基于实时分析引擎构建安全风控系统实时分析

- 业务特点**
- 超大规模、冷数据、低频&高时效性查询需求
- 面临的问题**
- MaxCompute -> Hbase，导入性能和导入作业维护无法忍受
- 改造收益**
- 无冗余存储，无导出数据作业，节约大量成本
  - 兼容PostgreSQL，学习成本低，BI工具无缝对接



上面三个场景各有各的业务特点，也有各自的痛点问题，通过 Hologres 可以针对性的解决所面临的问题，达到降低成本，节约资源的效果。

目前，Hologres 已经在阿里内部众多场景得到应用来解决面临的问题，并且 Hologres 也已经完成了商业化，欢迎大家使用，希望能够给云上的客户带来业务上的实质性帮助。

# 快速上手 Hologres

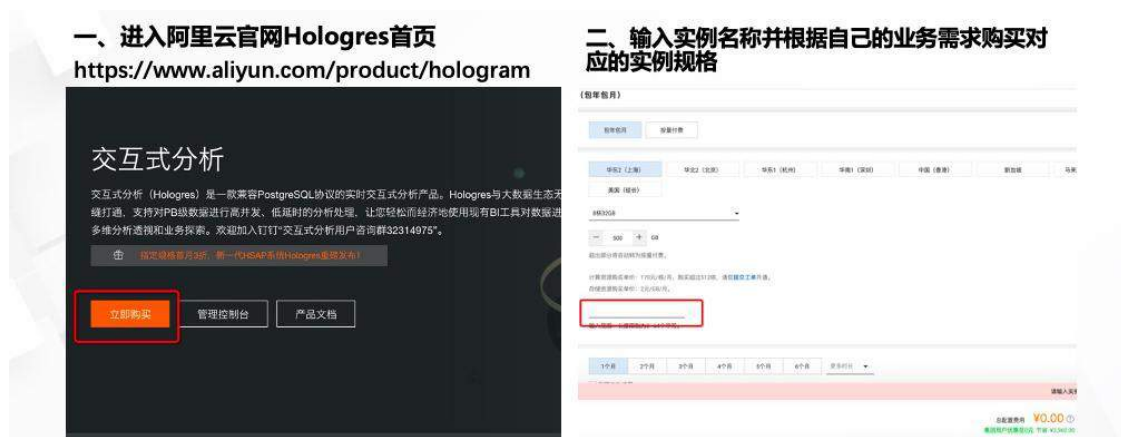
概要：本文将会为您介绍如何开通并使用 Hologres，并为大家详细介绍在 Hologres 中使用 SQL 开发以及性能调优。

作者 | 清芬（余骏） 阿里巴巴技术专家

## 一、实例购买与管理

### （一）购买 Hologres

进入阿里云官网，大家搜索交互式分析产品或者 Hologres，就可以进入到[产品详情](#)界面，点击立刻购买，就会进入到产品的购买页面；在产品购买页面中我们需要输入实例名称，然后根据实际需求和填写选择商品类型、地域等信息，即可以完成购买。



### （二）管理实例

实例购买完成之后，我们可以在阿里云界面上点击管理控制台进入 Hologres 管控台。Hologres 管控台是专属用于管控 Hologres 实例，如下图（左）所示。我们可以在管控台查看购买的实例详情，还可以针对每一个实例进行实例内对象管控，例如新建 DB、授权用户等，同时管控台还提供监控告警指标，方便您对实例运行情况做更加细致的管控。整个管控台的界面清晰明了，简单易用，对新手小白也非常的友好，易上手。



## 二、SQL 开发教程

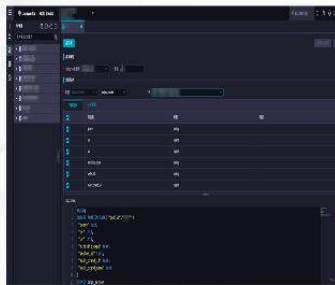
在学习 Hologres SQL 的使用之前我们需要了解 Hologres 实例内的相关概念：

- 实例：使用和管理数据库存储服务的实体，一个实例可以看作是多个数据库的合集。
- 数据库：一个模式的合集，用户所有的操作，包括表、函数等都是在数据库中完成的。系统会在用户完成实例申请后默认创建一个“postgres”的数据库，该 DB 仅用于运维管理，实际业务需要新建 DB。
- 表：表是数据存储的单元。它在逻辑上是由行和列组成的二维结构，列的数量和顺序是固定的，并且每一列拥有一个名字；行的数目是变化的，它反映了在一个给定时刻表中存储的数据量。
- 外表：外表是数据实际存储在其他系统里，但是通过 Hologres 来访问的一类表。Hologres 完全兼容 postgres 的 [Foreign Data Wrapper](#)，目前内部支持直接访问 MaxComputer 中的数据。

### （一）连接开发工具

Hologres 是一个完全兼容 Postgres 生态的产品，因此原则上任何能够直接访问 Postgres 的工具都能够直接连接 Hologres，比如 JDBC、ODBC 等工具。同时，Hologres 也提供了 HoloStudio、HoloWeb 等开发平台，方便用户根据不同的业务需求连接 Hologres 编写 SQL 任务，加快开发过程。

Hologres兼容PostgreSQL生态，提供JDBC/ODBC Driver，可以连接PG的开发工具都能直接连接Hologres，也提供HoloStudio和HoloWeb开发平台。



HoloStudio



HoloWeb



PostgreSQL生态工具

## (二) Hologres SQL

Hologres SQL 目前兼容开源的 PostgreSQL 11，因此用户可以参照 [Postgres 官方文档](#)来编写查询任务，另外，若是在开发过程如果遇到一些问题，也可以很容易的从搜索引擎上找到问题答案。

当前 Hologres 并未完全兼容 Postgres，以下将会介绍 Hologres 已经兼容的重点 SQL 语法和功能等。

### 1) 创建表

Hologres 建表的语句是 PG (Postgres) 的一个子集，使用起来非常方便，一个简单的创建表的例子如下图所示。

```
CREATE TABLE IF NOT EXISTS holo_test
(
    name text,
    ds text,
    age text,
    price float
    PRIMARY KEY (name)
);
```

### 2) 分区表的创建

分区表方便用户管理数据，并能通过分区裁剪来加快数据的查找。Hologres 中分区表的创建依然兼容 Postgres 语法，使用简便，但是有以下几点需要注意事项：



- 不能向父表中插入任何数据；
- 只有 text/varchar 类型才能作为分区键；
- Partition by 类型仅支持 list，切分 Partition list 只能有一个值；
- 分区表的数据不会自动删除，需要用户自己管理生命周期。

下图是一个创建分区表的实例：

```
begin;
drop table if exists HOLO_MULTI_PART;
create table HOLO_MULTI_PART(id text not null, pt text not null) PARTITION BY LIST (pt);
create table HOLO_MULTI_PART_0 partition of HOLO_MULTI_PART FOR VALUES IN ('0');
create table HOLO_MULTI_PART_1 partition of HOLO_MULTI_PART FOR VALUES IN ('1');
create table HOLO_MULTI_PART_2 partition of HOLO_MULTI_PART FOR VALUES IN ('2');
create table HOLO_MULTI_PART_3 partition of HOLO_MULTI_PART FOR VALUES IN ('3');
create table HOLO_MULTI_PART_4 partition of HOLO_MULTI_PART FOR VALUES IN ('4');
commit;
```

### 3) 数据类型

目前 Hologres 支持大部分 Postgres 支持的数据类型，如下图所示，并且 Hologres 支持的数据类型在不断的增加，以便能够满足大家更多的场景需要。关于数据类型的支持情况，可以参考文档[数据类型](#)。

数据类型	别名	是否支持	存储大小	范围	说明
integer	int, int4	支持	4 字节	-2147483648 到 +2147483647	常用的整数
bigint	int8	支持	8 字节	-9223372036854775808 到 +9223372036854775807	大范围整数
boolean	bool	支持	1 字节	True / False	布尔类型
float	float8	支持	8 字节	15 位十进制数字精度	可变精度，不精确
double precision		支持	8 字节	15 位十进制数字精度	可变精度，不精确
text	varchar	支持	可变长		可变长度字符串
timestamp with time zone	timestamptz	支持	8 字节	4713 BC 到 294276 AD	时间戳，包含时区，解析度为1微秒/14位数。 示例: '2004-10-19 10:23:54+02'
date	date	支持	4 字节	4713 BC 到 5874897 AD	单位是一天
decimal	numeric	支持	可变长	小数点前 38 位；小数点后 38 位	需要指定precision, scale信息。

## 4) 存储属性设置

Hologres 与标准 PG 有一个不太一样的地方,就是 Hologres 有一个额外的属性设置,如下图所示,这种属性设置类似其他数据库中的索引,能够起到查询加速的效果。

```
call set_table_property('table_name', 'orientation', '[column | row]');
call set_table_property('table_name', 'clustering_key', '[columnName{:[desc|asc]}[,...]]');
call set_table_property('table_name', 'segment_key', '[columnName[,...]]');
call set_table_property('table_name', 'bitmap_columns', '[columnName[,...]]');
call set_table_property('table_name', 'dictionary_encoding_columns', '[columnName[,...]]');
call set_table_property('table_name', 'time_to_live_in_seconds', '<non_negative_literal>');
call set_table_property('table_name', 'distribution_key', '[columnName[,...]]');
```

Hologres 目前通过设置 set\_table\_property 语法来指定表的额外属性,合理设置 table property 对于查询的性能影响极大,但是我们需要注意 set\_table\_property 的调用需要 create table 在同一事务中执行,且 set\_table\_property 对于同一个 key 不能重复调用,下面将会来详细介绍在 Hologres 中属性的设置以及其适用场景和使用限制。

### 1. 存储类型设置

Hologres 目前支持按行存储和按列存储两种存储类型,我们可以通过 orientation 来指定存储类型。行存储对于高 QPS 的基于 primary key 的点查询,例如 where pk=abc,其余场景都应该选用列存方式。下图是两个相应的例子。

```
begin;
create table tbl (a int not null, b text not null, PRIMARY KEY(a));
call set_table_property('tbl', 'orientation', 'row');
commit;

-- 适合的查询
select * from tbl where a = 100;

begin;
create table tbl (a int not null, b text not null, PRIMARY KEY(a));
call set_table_property('tbl', 'orientation', 'column');
commit;

-- 适合的查询
select sum(a) from tbl group by b;
```

## 2. 聚簇索引 Clustering Key

聚簇键类似于传统数据库中的聚簇索引。如果用户设置了 clustering key，指定一些列作为聚簇索引，Hologres 会在指定的列上将建立聚簇索引，并且在聚簇索引上对数据进行排序。建立聚簇索引能够加速用户在索引列上的 range 和 filter 查询。

- clustering key 创建的时候数据类型不能为 float/double，
- 每个表最多只能有一个 clustering key。

```
begin;
CREATE TABLE tbl (a int not null, b text not null);
call set_table_property('tbl', 'clustering_key', 'a');
commit;

-- 适合的查询
select sum(a) from tbl where a > 100 and a < 200;
```

## 3. 分段键 Segment key

聚簇索引主要是对数据进行排序，而分段键 Segment key 主要是用来帮助 Hologres 进行一些文件的快速筛选和跳过。用户可以通过设置 Segment key 指定一些列（例如，时间列）作为分段键，当查询条件包含分段列时，查询可以通过 segment key 快速找到相应数据对应的存储位置。

- segment key 要求按照数据输入自增，一般只有时间类型的字段(timestamptz)适合设置为 segment key，其他场景基本不需要设置；
- 只有列存表支持分段键设置。

```
begin;
create table tbl (a int not null, ts timestamp not null);
call set_table_property('tbl', 'segment_key', 'ts');
commit;

-- 适合的查询
select sum(a) from tbl where ts > '2020-01-01' and ts < '2020-03-02';
```

#### 4. 比特编码列设置

比特编码列 `bitmap columns` 也是对 Hologres 性能来说非常重要的一个属性，通过 `bitmap_columns` 指定比特编码列，Hologres 会在这些列上构建比特编码，相当于把数据与对应的行号做一个映射。

- `bitmap` 可以对 `segment` 内部的数据进行快速过滤，因此建议把 `filter` 条件的数据建成比特编码。
- 目前 Hologres 会默认所有 `text` 列都会被隐藏式地设置到 `bitmap_columns` 中。
- 但是只有列存表支持比特编码列。

```
begin;
create table tbl (a int not null, b text not null);
call set_table_property('tbl', 'bitmap_columns', 'a');
commit;

--适合的查询
select * from tbl where a = 100;
```

#### 5. 字典编码列设置

字典编码主要是对一些字符串类型的列生成字典编码。用户通过设置 `dictionary_encoding_columns` 指定字典编码列，Hologres 将为指定列的值构建字典映射。字典编码可以将字符串的比较转成数字的比较，加速 `group by` 查询，因此建议用户将 `group by` 的字段都建成 `dictionary_encoding_columns`，但是不建议将基数高的列建为 `dictionary_encoding_columns`，会导致查询性能变差。Hologres 默认所有 `text` 列都会被隐式地设置到 `dictionary_encoding_columns` 中，另外需要注意只有列存表支持字典编码列。

```
begin;
create table tbl (a int not null, b text not null);
call set_table_property('tbl', 'dictionary_encoding_columns', 'b');
commit;

--适合的查询
select sum(a) from tbl group by b;
```

## 6. 分布键 distribution key

Hologres 是一个分布式的计算引擎，如果没有设置分布键，数据库表默认为随机分布形式，数据将被随机分配到各个 shard 上；如果用户指定了分布列，数据将按照指定列，将数据 shuffle 到各个 shard，同样的数值肯定会在同样的 shard 中。

当用户以分布列做过滤条件时，Hologres 可以直接筛选出数据相关的 shard 进行扫描；当用户以分布列做 join 条件时，Hologres 不需要再次将数据 shuffle 到其他计算节点，直接在本节点 join 本节点数据即可，可以大大提高执行效率；同时如果用户 group by 的 key 是分布列也可以减少一次数据 shuffle，对整个查询的性能带来非常大的提升。

- 对于有 pk 的表，其分布键默认就是 pk，如果不想 pk 字段作为分布键，可以指定 pk 字段的子集，但是不能随意指定。
- 可以通过 shard\_count 来指定表的 shard 数，如果不指定的话每个数据库都有一个默认的 shard 数，一旦指定了一个表的 shard 数，其他的表如果想要和这个表做 local join，就必须指定 colcate with 这个表。下图所示是一个通过分布键设置来加速两个表做 join 的场景。

```
begin;
create table tmp(a int, b int, c int);
call set_table_property('tmp', 'distribution_key', 'a');
commit;

begin;
create table tmp1(a int, b int, c int);
call set_table_property('tmp1', 'distribution_key', 'b');
commit;

select count(1) from tmp join tmp1 on tmp.a = tmp1.b -- 将分布列设为 join key
```

## 7. 数据生命周期管理 time\_to\_live\_in\_seconds

Hologres 目前提供了 time\_to\_live\_in\_seconds 来帮助管理数据的生命周期，其默认单位是秒，必须是非负数字类型。

表数据的 TTL 并不是精确的时间，当超过设置的 TTL 后，系统会在某一个时间自动删除表数据，因此业务逻辑不能强依赖 TTL，以免带来不必要的损失。



```
begin;
create table tbl (a int not null, b text not null);
call set_table_property('tbl', 'time_to_live_in_seconds', '3.14159');
commit;

-----

begin;
create table tbl (a int not null, b text not null);
call set_table_property('tbl', 'time_to_live_in_seconds', '86400');
commit;
```

### 三、性能调优

下面是一个两个表关联查询的优化案例。这里我们有 tmp1 和 tmp2 两个表，其中 tmp1 有 1000 条记录，tmp2 有 1 亿条记录，两个表 join 查询的 query 如下：

```
select count(1) from tmp1 join tmp2 on tmp1.a = tmp2.a
```

在查询之前，我们通过 explain 来查看执行计划，如下图所示：

```
it=# explain select count(1) from tmp join tmp1 on tmp.a = tmp1.b;
               QUERY PLAN
-----
Partial Aggregate (cost=0.00..10.11 rows=1 width=8)
-> Gather Motion (cost=0.00..10.11 rows=10 width=8)
-> Partial Aggregate (cost=0.00..10.11 rows=10 width=8)
-> Hash Join (cost=0.00..10.11 rows=1 width=1)
    Hash Cond: (tmp.a = tmp1.b)
-> Parallelism (Gather Exchange) (cost=0.00..5.04 rows=1000 width=1)
-> DecodeNode (cost=0.00..5.04 rows=1000 width=1)
-> Seq Scan on tmp (cost=0.00..5.01 rows=1000 width=1)
-> Hash (cost=5.04..5.04 rows=1000 width=1)
-> Parallelism (Gather Exchange) (cost=0.00..5.04 rows=1000 width=1)
-> DecodeNode (cost=0.00..5.04 rows=1000 width=1)
-> Seq Scan on tmp1 (cost=0.00..5.01 rows=1000 width=1)

Optimizer: HQO version 0.8.0
(13 rows)
```

通过查看执行计划，我们可以对整个查询过程更加了解，发现其中可以进行优化的地方，比如从上图可以看出整个查询计划并没有拿到表的相应统计信息，这种情况下会导致整个查询计划性能较差。

通过实验，上面的查询计划在不进行调优的情况下整个过程需要约 8.5 秒的时间。接下来我们通过 analyze 来帮助 Hologres 生成优化器需要的统计信息，其操作非常简单，只需要执行如下两行命令即可：

```
analyze tmp1;  
analyze tmp2;
```

在生成了统计信息之后我们再次通过 explain 查看执行计划：

```
--  
Partial Aggregate (cost=0.00..327.34 rows=1 width=8)  
-> Gather Motion (cost=0.00..327.34 rows=10 width=8)  
    -> Partial Aggregate (cost=0.00..327.33 rows=10 width=8)  
        -> Hash Join (cost=0.00..327.33 rows=1000 width=1)  
            Hash Cond: (tmp2.a = tmp1.a)  
                -> Parallelism (Gather Exchange) (cost=0.00..318.27 rows=1000000000 width=8)  
                    -> DecodeNode (cost=0.00..299.84 rows=1000000000 width=8)  
                        -> Seq Scan on tmp2 (cost=0.00..98.75 rows=1000000000 width=8)  
                -> Hash (cost=5.01..5.01 rows=44000 width=8)  
                    -> Broadcast Motion (cost=0.00..5.01 rows=44000 width=8)  
                        -> Parallelism (Gather Exchange) (cost=0.00..5.00 rows=1000 width=8)  
                            -> DecodeNode (cost=0.00..5.00 rows=1000 width=8)  
                                -> Seq Scan on tmp1 (cost=0.00..5.00 rows=1000 width=8)  
Optimizer: HQO version 0.8.0  
(14 rows)
```

从上图中可以看到，在 analyze 之后，整个查询计划已经发生了改变：

- 拿到了整个表的真实行数，并且对整个查询计划做了相应调整；
- join 顺序从 tmp1 join tmp2 变成了 tmp2 join tmp1，降低了资源消耗。

在经过上面的优化之后，再次执行相同的查询，发现整个查询过程只用了 500ms，整个查询性能大大提升，而我们所做的操作也仅仅是生成了整个表的查询信息，就获得了 10 倍以上的优化效果。这也说明整个查询的统计信息对查询性能有着至关重要的影响，因此大家今后在导入数据之后可以生成一下统计信息，以便更好的执行查询计划。

更多关于性能调优的方式，大家可以参考文档[性能调优](#)。

# Hologres+Flink 实时数仓详解

概要：本次内容将会介绍使用 Flink 和 Hologres，实现可扩展的、高效的、云原生实时数仓。

作者 | 王华峰 阿里云技术专家

## 一、Hologres 生态

从前面几篇的内容，相信大家已经了解到 Hologres 是一款兼容 PostgreSQL 协议的实时交互式分析产品。在生态的兼容性上，Hologres 有着非常庞大的生态家族，如下图所示：

- 对于开源大数据领域，Hologres 支持当下最流行的大数据开源组件，其中包括；
- 对于埋点类数据，支持 Blink/Flink/Spark/数据集成等大数据工具进行高性能的实时导入和批量导入；
- 对于数据库类的数据，通过和 Dataworks 数据集成（DataX 和 StreamX）共建实现方便高效的数据库整库实时镜像到 Hologres 中，并满足金融企业对可管理性、监控、网络等的需求。

无论是实时数据，还是离线数据接入 Hologres 之后，接下来就能使用 Hologres 对数据进行分析。最常见的就是使用 JDBC 或者 ODBC 对数据进行查询、分析、监控，然后承接上游的业务，比如说大屏、报表、应用等各种场景。



同时再为大家介绍一下 DataWorks，它是阿里云的一个数据开发平台，提供了数据集成、数据地图、数据服务等功能。数据集成主要功能可以将数据库的数据导入 Hologres，其中同步的方式包括离线同步和实时同步，离线同步支持几十种异构数据源同步至 Hologres，而实时同步当前主要支持以下几种：

- Mysql Binlog：通过订阅 Biblog 的方式将 mysql 数据实时写入 Hologres。
- Oracle CDC：全称是 Change Data Capture，也是一个类似 Mysql Binlog 的用来获取 Oracle 表 change log 的方式。
- Datahub：是阿里巴巴自研的一个分布式高性能消息队列，值得一提的是，Datahub 自身也提供了直接将数据实时导入至 Hologres 的功能，无需经过 Dataworks。
- PolarDB：是阿里巴巴自主研发的关系型分布式云原生数据库。

## Dataworks数据集成支持实时输入

- Mysql Binlog
- Oracle CDC
- Datahub
- Kafka
- PolarDB

## 二、Hologres 实时导入接口介绍

接下来为大家介绍一下 Hologres 提供的一个实时导入的接口，以及接口的技术原理。

### 1) 实时导入接口

Hologres 实时导入接口的具备以下特性：

- 行存&列存都支持；
- 支持根据主键去重 (Exactly once)；
- 支持整行数据局部更新；
- 导入即可见，毫秒级延迟；
- 单 Core 2W+ RPS (TPCH PartSupp 表)；

- 性能随资源线性扩展；
- 支持分区表写入。

## 2) 实时导入原理

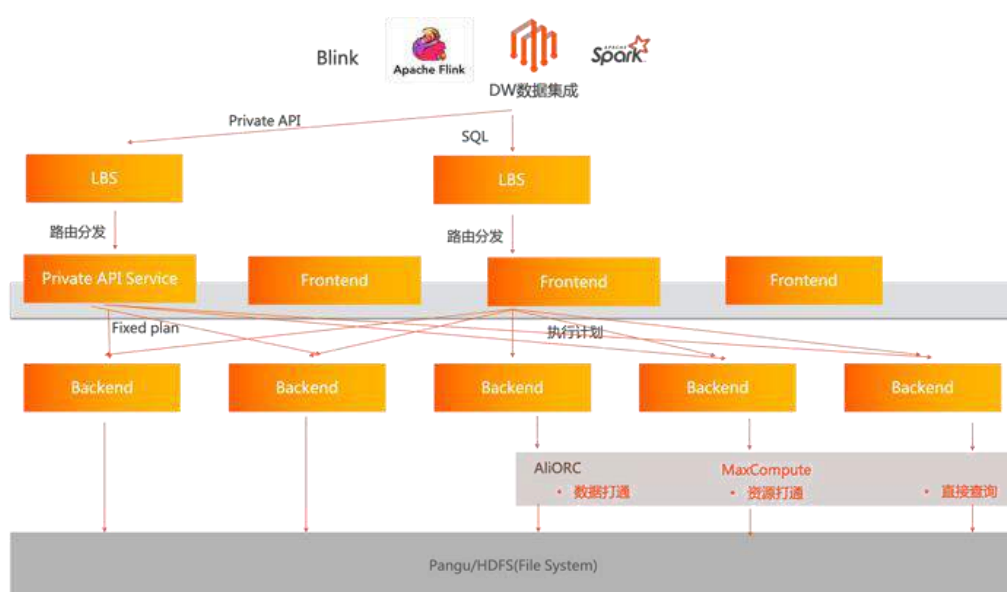
实时导入的原理如下图所示，首先我们看一下该图的最上面的几个节点，代表了数据的上游，也就是业务层。如何将数据导入 Hologres，主要有两种场景：

- 使用 SQL 进行数据的导入（最常见）

例如使用 JDBC 执行 insert 语句，该 insert 语句会经过一个负载均衡服务器路由分发至我们的 Frontend 节点，对该 insert 语句进行 SQL 的解析优化，然后生成一个优化后的执行计划，并将该执行计划分发至后端的 worker 节点。worker 节点收到该执行计划之后，就会将该数据完成写入。

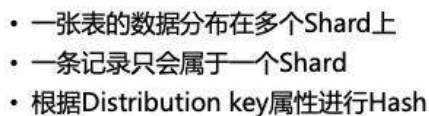
- Connector 写入

另外一条链路为左边的 Private API 链路，也就是当前 Apache Flink 或者 Apache Spark Connector 所使用的 Hologres 的实时导入接口。该 Private API 提供的数据接口和普通 sql 请求不一样，而是我们称之为 Fixed Plan 的请求接口，这些请求被分发至负载均衡服务器之后，负载均衡服务器会将数据路由分发至一个叫做 Private API Service 的节点。该节点将数据写入请求分发至 worker 节点，也就是后端的节点。当 worker 节点收到，无论是 Fixed Plan，还是执行计划之后，会对数据进行持久化，最终数据完成写入。

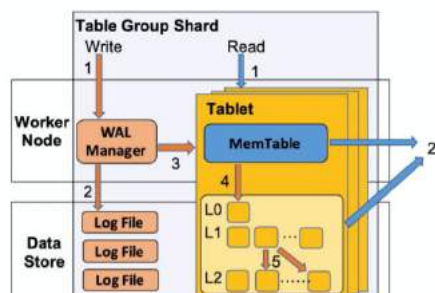




## Hologres实时导入实现原理



- Log Structured Merge Tree(LSM);
- 全异步框架，协程(Coroutine);
- 基于 Masstree 的 Memtable。



- 同时上面也提到通过 SQL 来进行数据的写入是最常见的场景，Hologres 也在后端优化了整个 SQL 的写入链路。例如对于 Insert into values, Insert into on conflict do update, Select from where pk = xxx 等场景简单的 SQL，Hologres 会进行优化，减少 SQL 的解析和优化过程，提升整个数据写入和查询的性能。

### 三、Hologres 实时读写场景

前面介绍了 Hologres 通过 connector 写的原理，下面将会介绍 Flink+Hologres 最常见的写入场景。

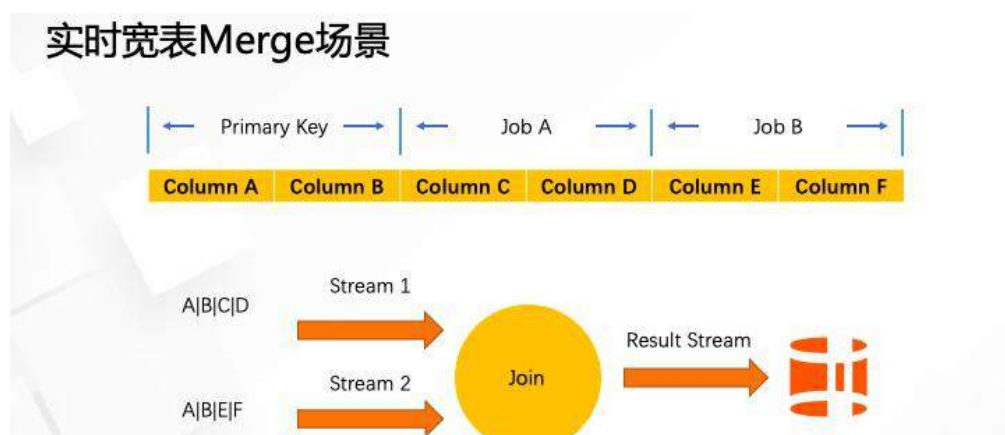
#### 1) 实时写入场景

最常见的第一种就是实时写入场景。实时写入分为几种。

- 第一种，Hologres 的结果表没有设置主键，这样 Flink 实时接入就是一种 Append Only 的模式进行写入。当上游数据发生重复，或者 Flink 任务作业失败，上游数据会需要进行回溯，这时候下游数据录入到 Hologres 中就会产生重复的数据。这种情形对于日志型数据是比较合理的，因为用户并不需要关心数据是否需要进行去重。
- 第二种，Hologres 的结果表设置了主键。Flink 或者其它实时写入就会按照行的主键进行更新。主键更新的意思就是说对于相同主键的两行数据，后到的数据会完全覆盖掉之前已经到达的数据。
- 第三种，是按照主键去重。就是说后到的数据会被忽略掉，只保留最早到的一条记录。这种场景用户并不关心主键的更新情况，只需要保证主键的去重。

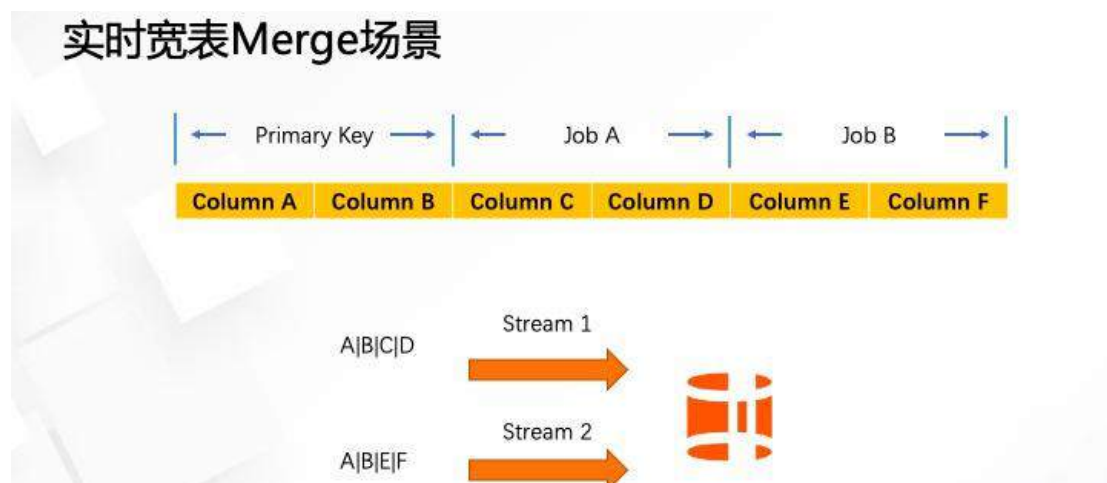
#### 2) 宽表 Merge 场景

例如一个用户的结果表有非常多的字段，会有上百列，而该表的许多字段可能同时分布在不同的数据上游，例如，Column C 和 D 分布在一个 kafka 的 topic A 上面，Column E 和 F 分布在 kafka 的 topic B 上面，用户希望消费两个 kafka topic，并将数据 merge 成 Hologres 的一张结构表。最常见的解决办法是，进行流场景的一个双流 Join。这种实现对于开发人员来说相对比较复杂，需要实现一个双流 Join，而且理论上来说会对计算资源要求非常大，也加剧了运维人员的负担。



而 Hologres 针对这种场景是如何实现的呢？

Hologres 支持局部更新的功能。如下图所示，按照这种实现方式，只需要两个流各自写入 Hologres 结果表。第一个流消费 ABCD 四个字段，将数据写入到最终的结果表中。第二个流消费 ABEF 四个字段，最终将数据写入到结果表，并不需要进行双流的 Join，最终 Hologres 会自己进行一个数据的组装。第一个流写入 ABCD 的时候并不会去更新已经存在的 EF 字段，同样，第二个流写入 ABEF 字段的时候，C 和 D 字段已经存在，不会被更新，最终达到完整的一个数据 Merge 的功能。使用这种功能，可以大大提升流作业的开发效率，以及减少流作业所需要的资源消耗，也能够更容易的维护各个流作业。



### 3) 实时维表 Join 场景

除了写场景，Hologres 也支持读场景，最常见的是使用 Hologres 的行存表来进行点查。如下图所示，是一个实时维表的 Join 场景。主要逻辑是生成一个数据源，会不停的生成一个数据流，和 Hologres 的维表进行 Join，打宽数据流，最终将数据写入到一个结果表中。在实际业务中，这种使用场景通常会用来替换 HBase，以达到更好的性能和更低的成本。

最后一个场景是 Hologres 的 Binlog 场景，如下图所示，以消息队列方式读取 Hologres 数据的 Change log。

```
1 CREATE TABLE src(  
2   a INT,  
3   b BIGINT,  
4   c STRING,  
5   'proc_time' AS PROCTIME()  
6 ) with (  
7   'connector' = "datagen"  
8 );  
9  
10 create table dim (  
11   a int,  
12   b VARCHAR,  
13   c VARCHAR,  
14   PRIMARY KEY (a, b), PERIOD FOR SYSTEM_TIME  
15 ) with (  
16   'connector' = 'hologres',  
17   ...  
18 );  
19  
20 CREATE TABLE print_sink(  
21   a INT,  
22   b STRING,  
23 ) with (  
24   'connector' = 'print'  
25 );  
26  
27 insert into print_sink select T.a,H.b  
28 FROM src AS T JOIN dim FOR SYSTEM_TIME AS OF T.proc_time "  
29   "AS H ON T.a = H.a"
```

#### 4) Hologres Binlog 场景

如下图所示，以消息队列方式读取 Hologres 数据的 Change log。其中：

- Binlog 系统字段，表示 Binlog 序号，Shard 内部单调递增不保证连续，不同 Shard 之间不保证唯一和有序；
- Binlog 系统字段，表示当前 Record 所表示的修改类型；
- UPDATE 操作会产生两条 Binlog 记录，一条更新前，一条更新后的。订阅 Binlog 功能会保证这两条记录是连续的且更新前的 Binlog 记录在前，更新后的 Binlog 记录在后。

### Hologres Binlog场景

- 以消息队列方式读取Hologres数据Change log
- Binlog系统字段
  - hg\_binglog\_lsn
  - hg\_binglog\_event\_type
  - hg\_binglog\_timestamp\_us

# MaxCompute+Hologres 数据仓库详解

概要：介绍如何基于 Hologres 和 MaxCompute 产品组合，支撑高并发、快响应的数据服务化场景，替换 HBase 开发模式，实现数据资产服务化在线化能力。

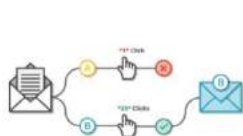
作者 | 刘一鸣 阿里云高级产品专家

## 一、传统数据库痛点

传统的数据仓库开发方式，比如在一些用户行为的分析，点击流的分析场景，一般来说是通过一些报表的形式去展现。这类分析背后往往需要很复杂的计算，需要计算引擎具备很强大的计算能力。另外一个场景，也是一个很强的数据驱动的场景，常见的像推荐系统的场景。推荐背后完全依赖对数据或者说对人的行为的理解，基于历史行为的统计信息，我们来智能化的推荐什么事情是他当下最感兴趣的。这类场景并不是传统的分析场景，但它也是一个数据决策的场景，今天我们把它们放在一起，看一看我们的数据仓库是如何支持不同的数据决策场景。

从行为分析到实时推荐，这两者之间有一些关系。传统的报表分析是一种查询相对比较复杂，查的也比较快，要交互式体验。但是对于实时推荐的场景，查询往往更简单，往往就是一张单表的查询，但是对性能会非常的敏感，一般来说需要毫秒级响应。这两件事过去往往是由不同的系统来做的。

### 分析与服务典型场景



复杂分析类查询，要快



简单查询，要非常快



传统上，我们是如何做数据加工以及数据服务的呢？最常见的方式就是下图所示，数据链路也左向右发展，有加工平台负责加工，结果数据导出到服务平台对外提供服务。这个架构在最开始应用的时候还是比较顺利的，大部分公司里面 90% 的场景下是这个架构。但是随着业务越来越多，越来越复杂，每天都有新的报表，每天都有新的业务场景，就会发现每一个业务调整的时候，都要从源头一步步调整，包括表结构，加工脚本，历史数据重刷等等，最后造成整个数据加工的链路会变得数据冗余、成本高、开发周期长、甚至数据不一致。于是我们就需要开始思考这个架构还能不能再有优化的空间，甚至是再简化一些。



要解决上述问题，可以看到市面上的选项大致分为 3 个系统，以及它们之间的综合。

第一个是直接用事务系统做分析，Transaction 类型，支持随机读写、事务、可靠，主要面向 DBA。

第二个系统是专业的分析系统，Analytics，适合大规模数据扫描、过滤、汇总，面向分析师。

第三个系统是在线服务的系统，Serving，支持高并发、简单、快速，面向 API。

三个系统各有擅长，最终落地解决问题的时候，往往多个系统一起使用，数据在系统间频繁交换，也引起了刚刚提到的各种冗余、复杂、不一致的问题。

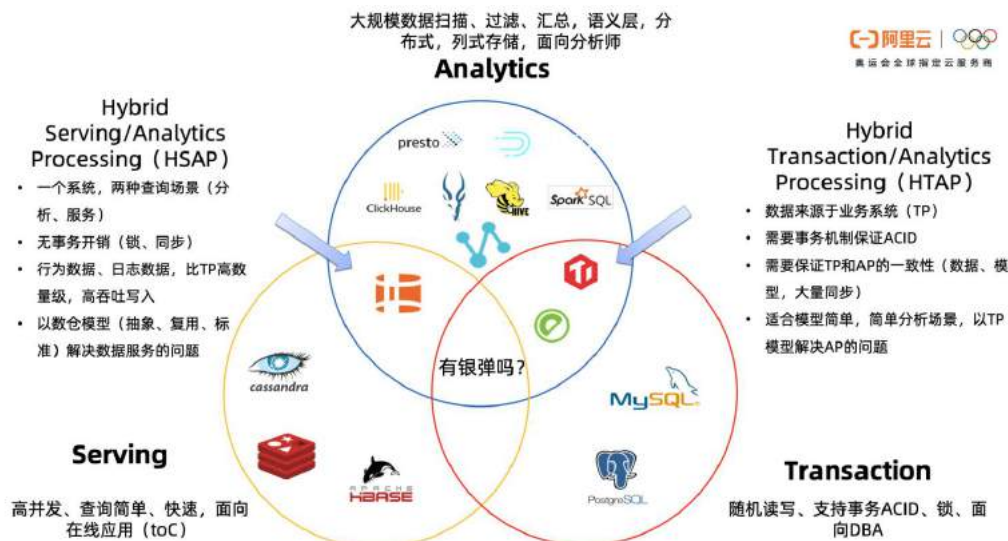
我们首先想到的是，能否一个系统支持尽量多的场景，简化运维，少一些数据移动。

首先出现的是红线和蓝线交界部分，让一个系统既可以支持分析，又可以支持事务，也就是我们常说的 HTAP，它有一定的适用场景：

- 数据来源于业务系统（TP）。
- 需要机制保证 TP 和 AP 的一致性（数据、模型，大量同步）。
- 模型简单，用于简单分析场景。
- 局限是，事务系统的数据模型是无法直接用于分析场景的，数据需要被加工、抽象才能用于数据分析师。

在另一侧，蓝色和黄色交界的部分也有一个场景，分析和事务也可以做成一个系统，也有它的适用场景：

- 统一实时、离线存储引擎。
- 没有事务需求，减少针对事务场景的开销（锁、同步）。
- 埋点数据、机器数据，比 TP 高数量级。
- 为多场景设计可复用数仓。



## 二、HSAP：服务分析一体化

我们提出 HSAP（Hybrid Serving & Analytics Processing）的理念，目标做到服务分析一体化，背后的技术挑战是非常大的：首先，离线数据和实时数据都要支持。然后我们希望数据是统一的，不要把实时和离线割裂。也希望接口是统一的，提供一套统一的接

口。我们可以看到 SQL 接口相对来说是现在市面上表达能力最强的一种查询语言，所以我们认为在 HSAP 这种理念下，如果重新做一个系统，它应该支持统一的查询接口，一般是 SQL，统一的访问控制；然后统一的存储，实时、离线都可以存在这样一个系统里。

## 下一代大数据数仓理念HSAP：分析、服务一体化

**HSAP: Hybrid Serving & Analytical Processing**

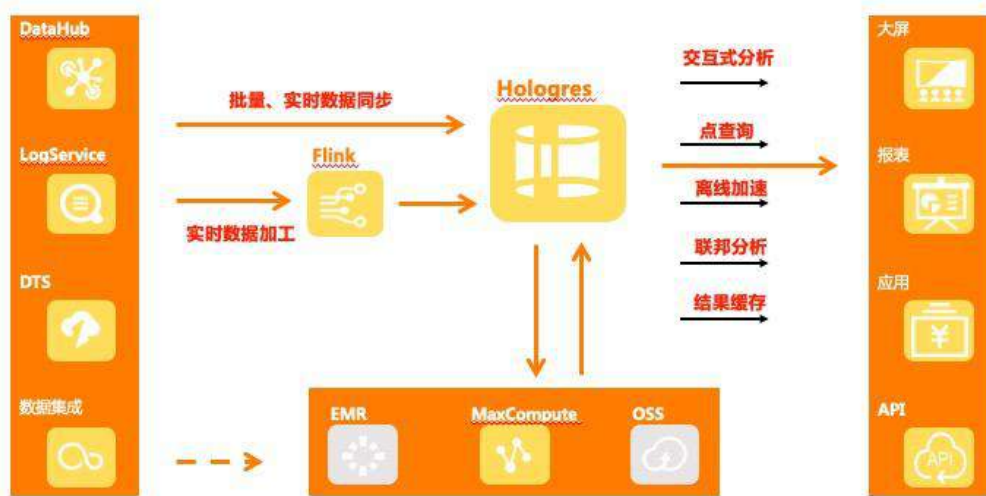


### 三、实时数仓：Hologres

Hologres 是针对 HSAP 场景设计的系统，它支持数据以离线和实时的方式，从业务系统、日志系统进入 Hologres。进入过程中，一种是批量导入，这种方式的数据量是很大的。另一部分实时数据会通过 Flink 实时计算，通过计算前置的方式，把一些计算逻辑规则通过 Flink 提前算好，比如流数据关联，窗口加工，轻度汇总等，然后把计算结果存在 Hologres 的表里面，然后通过 Hologres 作为一个服务平台对接应用层，不管是一个多维分析的报表，还是一个实时推荐的应用，都可以支撑服务。

为了实现实时和离线的一体化，Hologres 与 MaxCompute 做到了底层打通，Hologres 隶属 MaxCompute 产品家族，这两个产品合在一起为用户提供实时、离线一体化的解决方案。数据在 Hologres 和 MaxCompute 之间以一种原生的方式互相打通，互相可以查询对方的数据，互相可以看到对方的表，互相可以有很简单的方式做一些数据的迁移。

## 分析、服务一体化实时数仓Hologres



上面提到 Hologres 既能支持分析，又能支持服务。这是因为 Hologres 底层会有两种不同的存储模式：行存和列存，这也就意味着我们既能支持查询的量，又能支持查的快。下图是 Hologres 行存和列存两种模式下的对比：

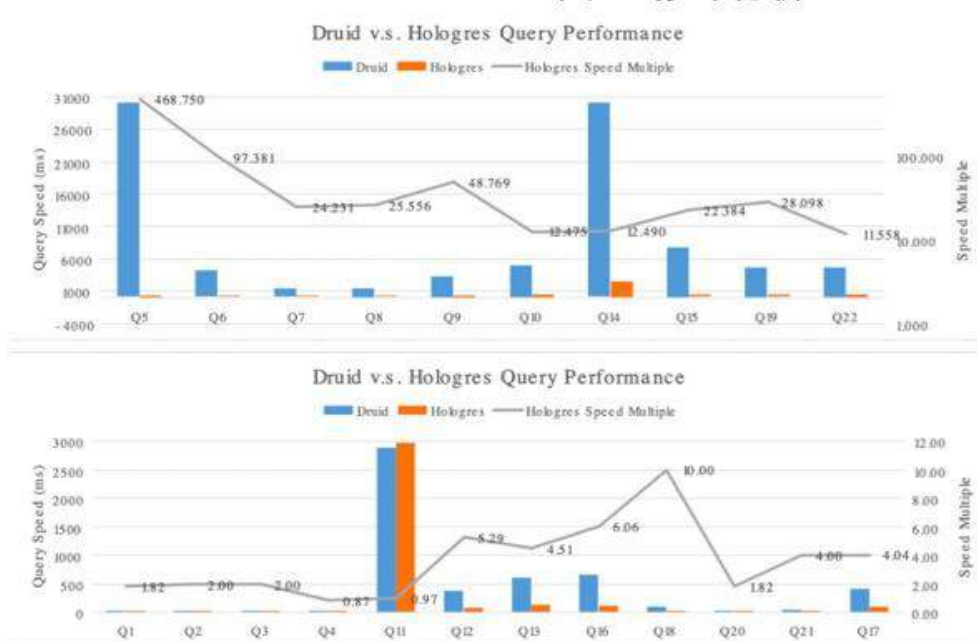
阿里云 开发者社区

### 行存 Vs 列存

	行存	列存
使用场景	点查, Point Query, 面向应用, 面向 API	多维、过滤、聚合, OLAP, 面向分析师, 面向报表
存储考虑	Key Value结构	列组结构
索引优化	整行连续存放, 形成Block, 包含Data Block和Index Block ( Range和 Offset )	列数据连续存放, 聚合高效, 形成Block, 同时维护Index Block ( Range、Offset、编码、压缩、统计 ) 和Meta Block。
更新方式	内存更新标志位, 异步Flush文件, 减少IO操作	引入Delete Map, 提供更新能力
接口标准	MaxC SQL	PostgreSQL
SQL Demo	select * from table where primary_key = XXX	select count(), sum() from table group by A
IO开销	K to MB	MB to GB
QPS	1K-100K+	10-1K

同时，下面也可以给大家分享 Hologres 与 Druid 的性能对比（对比基于 TPC-H 测试集），可以从图中直观的看到 Hologres 绝大多数场景下性能更优。

## Benchmarks – OLAP 交互式多维分析



Hologres 的另外一个场景是 Serving 场景，主要是对标 HBase。我们看到，在同等的硬件条件下，99%的查询延迟 Hologres 相对来说都是非常的稳定，小于一微秒。而对于 HBase 来说，查询延迟随着吞吐量的变高，延迟有很大的扩大。Hologres 的定位是同时支持分析和服务两个场景，同时我们也有信心把这两个场景做的都比只做一个场景的那些过去的系统做得更好。对于一个新技术而言，必须要有更高的要求。

## Benchmarks – Serving 高并发点查询

### YCSB (Yahoo Cloud Serving Benchmark)

- Hardware: 192 core

Throughput	Hologres 99th Percentile Latency (us)	HBase 2.2.4 99th Percentile Latency (us)	Diff
100,000	355	12431	35x
500,000	554	33385	60x
1,000,000	885	116889	132x

现在绝大多数业务都是用 HBase 来做服务的场景，在 Hologres 里使用行存表的效果是优于 HBase 的。这边给大家比较 Hologres 和 HBase 的一些差异点，主要关注红色的部分：



## 1) 看产品定位和高层设计

		Apache HBase	阿里云Hologres	差异
产品定位		分布式面向列簇的开源数据库	支持HSAP能力的云原生实时数仓	Hologres同时支持Serving和OLAP两个场景
高层设计	系统架构	存储计算耦合，存储依赖底层HDFS。HDFS集群需要手动扩容。	存储计算分离，Shared Nothing的MPP架构	
	存储引擎	仅支持行存，以Key Value方式存储	行存+列存	Hologres针对不同的场景使用不同的存储引擎，把分析（列存）和服务（行存）各自发挥到极限。
	Schema支持	弱schema	强schema，丰富的类型	【Hologres】强Schema可以保证开发的效率，在数据质量不可靠，数据接口不明确的情况下，更易于通过Schema排查开发问题。
	SQL支持	通过Phoenix扩展支持，功能弱，不支持Join。 受限KV存储模式，SQL性能差。	PostgreSQL协议，兼容PG 11	PG的生态更丰富
	实时写入	支持，写入即可查。写入TPS受限于compaction性能	支持，写入即可查。高TPS写入。	【Hologres】更高的QPS和TPS，Hbase存在写入热点问题，造成Region Server不稳定，宕机。
	数据分片	支持，支持预先分片和自动分片的模式	支持，集群模式下2种分片模式：哈希（hash）、随机	
	开发语言	Java为主	C++为主	【Hologres】执行效率更高

## 2) 存储引擎和查询引擎的差异

		Apache HBase	阿里云Hologres	差异
存储引擎	压缩	压缩比差，冗余信息多	高压缩比，特别是列存时，压缩比非常高	
	排序	全局排序	局部排序	【Hologres】可配置的聚簇索引
	存储能力	基于分布式文件系统HDFS，用户自行维护集群，集群会自动同步数据至多副本，存储能力与集群规模有关，支持线性扩展，LSM-Tree数据结构，多种压缩算法。	基于分布式文件系统Pangu/HDFS，存储能力与集群规模有关，支持线性扩展，单表最大容量3PB+；多种存储模式和多种压缩算法赋能存储。	
查询引擎	事务能力ACID	有限支持，支持行级数据的ACID	有限支持，支持行级数据的ACID	
	查询语言	Java API（需要与其他框架共同使用，例如Apache Phoenix）	PG SQL，支持全Join关联查询	
	分析能力	原生仅支持点查(GET)和扫描(SCAN)。点查QPS高，SCAN性能差。Phoenix SQL通过coprocessor支持，性能差，不支持复杂计算	百亿级数据，实时查询及分析亚秒级响应；千亿/万亿级数据，实时查询及分析秒级响应；Join能力强大。点查QPS高。	【Hologres】海量数据的查询及分析实时响应；Join能力强大。
	并发任务数-复杂OLAP	不支持OLAP场景	支持，较高QPS	【Hologres】全异步架构，查询QPS和TPS高
	在线数据服务(QPS)	点查：高QPS(5w+/s)	点查：高QPS(5w+/s)	

3) 扩展性, 运维, 生态, 适用场景, 以及开发方式的差异

	Apache HBase	阿里云Hologres	差异
扩展性	存储计算耦合, 需同时扩展。	存储与计算能够独立线性扩展, 提高并行能力。	【Hologres】 业务方无需担忧因存储或计算其中之一先达到瓶颈需要扩容而导致另一资源浪费 企业无需频繁更新机器配比 企业无需担忧扩容带来的大量数据迁移的问题。
运维	用户需要自行运维。	全托管。 系统自动化感知集群的拓扑信息变化, 用户侧无感知。	【Hologres】 全托管,免运维。
生态	HBase兼容Hadoop生态	Hologres高度兼容PG生态	
适用场景	海量存储, 非结构化存储, 单点查询性能优异, 写密集型数据库。	实时数仓。可全面替代HBase产品。 联通数据孤岛, 海量数据实时查询及分析, 弹性扩展集群。完整SQL支持	
开发方式	应用开发复杂, 需要将业务分析的指标、维度、表、聚合等概念, 转化为存储的Key/Value概念, 将应用层查询过滤场景翻译为对Key的字节过滤操作, 系统效率严重依赖Key设计的质量。整个系统从数据录入到数据分析查询等复杂多样的场景, 依赖应用层对Key/Value基础接口的使用。	应用开发简单, 面向Table开发, 使用SQL标准语句, 适用于复杂多维分析, 嵌套查询, 关联查询等场景。提供JDBC、ODBC接口, 面向数据主题建模开发。	【Hologres】 从HBase的面向指标, 面向宽表开发, 转化为Hologres面向主题域建模, 减少了数据模型在采集端、处理端、分析端的异构信息衰减, 减少了数据加工的层次, 提高了数据使用的灵活性。

四、典型应用案例

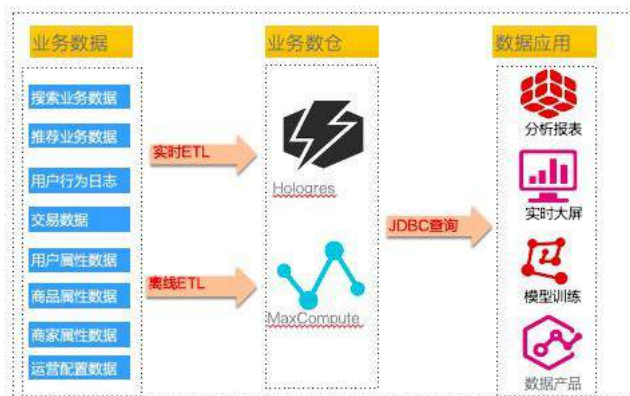
下面可以来介绍 Hologres 落地具体业务场景的最佳实践。

1) 阿里巴巴搜索推荐实时分析和算法应用

阿里淘宝系列里边实时的搜索推荐的场景下, 也在使用 Hologres 和 MaxCompute 的组合。一方面数据量非常的大, 我们会首先基于 MaxCompute 建立整个离线的数仓, 它能够处理离线全量的数据, 有调动几百上千台机器这样并发的能力。其次推荐场景下, 也是对实时要求非常高的。推荐场景的业务方会很关心用户当前对什么产品感兴趣, 当前搜索什么样的关键词, 当前是通过什么网络、通过什么设备、在什么地域连接了淘宝的一个系统等信息。一些实时特征都会影响搜索的一个结果, 对实时特征的计算也非常的重要, 所以需要有一个实时数仓的场景, 通过 Flink 加工, 把这些实时信息加工成用户和商品的一些特征, 然后对接应用使用。最后加工的结果一部分面向分析师, 一部分面向机器学习。根据刚才学到的一些知识, 我们知道要完成全部的场景会利用到 Hologres 的列存, 也会利用行存。

## 搜索推荐实时分析和算法应用

- 数据量大，单日PB级存储
- 单表总条数**千亿+**
- RPS高，Flink峰值写入RPS **千万+**
- 峰值查询QPS **200+**
- 数据灵活性要求高，分析场景多样化，固定条件高频分析、非固定条件多维查询



列存的场景，一般是面向报表的场景，一般会把我们的数据结果变成我们的数据产品，提供给我们的分析师。分析师选择相关的维度做一些过滤组合，得到相关的一些转化率。这相当于是把我们的数据产品作为一种服务。

## 数据产品，自助式分析



### 2) 友盟

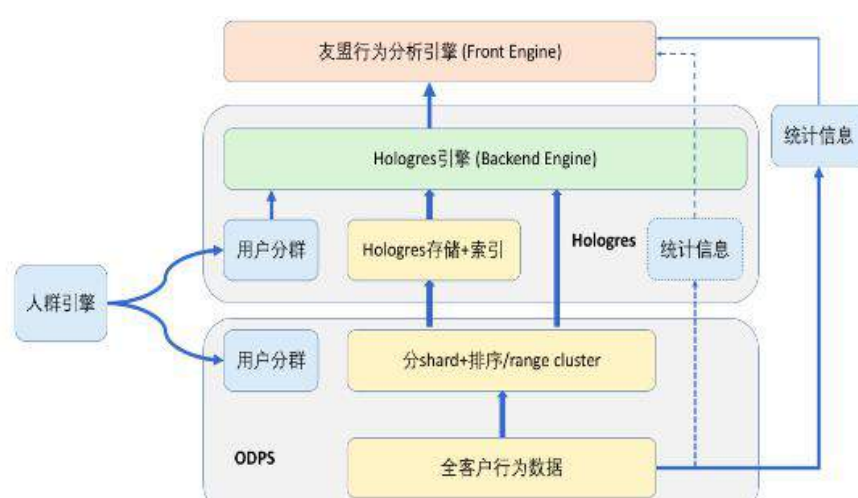
友盟+是国内最大的移动应用统计服务商，其统计分析产品 U-App&U-Mini & U-Web 为开发者提供基本报表统计及自定义用户行为分析服务，支持精细化运营。业务痛点包括：

- 业务数据量大，年新增行为数据 10PB 级，个性化、自定义地交互式用户行为分析强需求；
- 基于 MaxCompute 提供异步离线的 adhoc 分析和优化、以及自研引擎开发尝试均无法满足业务需求；

- 导出到 mysql/Hbase 方案的二次开发和数据导出链路长、成本高、操作不灵活。

在使用 Hologres 和 MaxCompute 后，客户得到的收益包括：

- PB 级数据毫秒级查询响应；
- 与 MaxCompute 深度集成，能够利用 range cluster 索引加速，实时离线联邦查询，同时也可以实现冷热数据混合查询，有利于成本性能平衡；
- 计算资源弹性伸缩，可兼顾扩展性、稳定性、性能、成本。



### 3) 菜鸟的智能物流

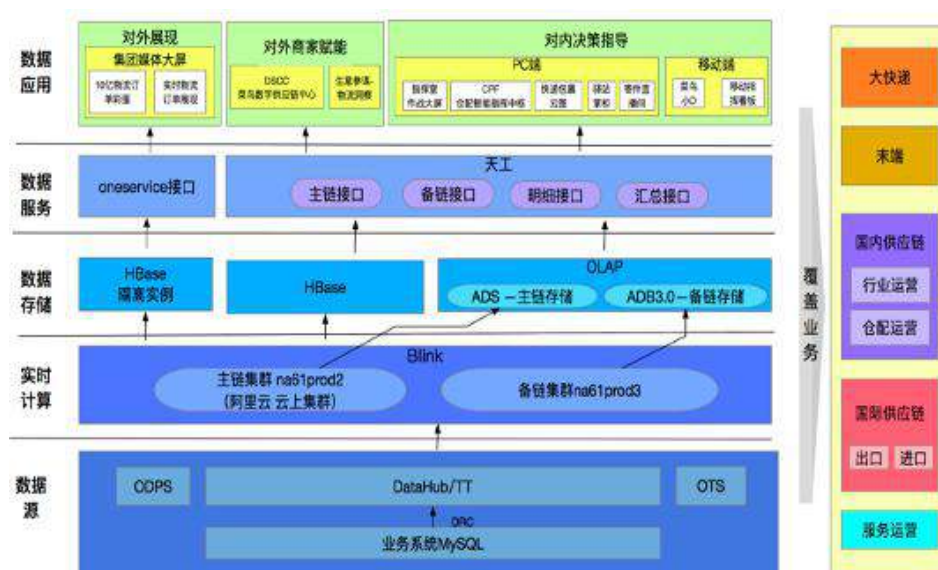
菜鸟智能物流分析引擎是基于搜索架构建设的物流查询平台，日均处理包裹事件几十亿，承载了菜鸟物流数据的大部分处理任务。客户的需求包括：

- HBase 的架构下维表数据导入耗时长、资源浪费严重、成本高；
- HBase 不能同时满足 PointQuery 和 OLAP 分析，数据导入导出引发数据孤岛、数据同步负担、冗余存储、运维成本和数据不一致等问题。

在引入 Hologres 交互式分析后，客户得到的收益包括：

- 整体硬件资源成本下降 60%+；
- 更快的全链路处理速度（2 亿记录端到端 3 分钟）；
- 一个系统，满 KV 和 OLAP 两个场景，没有数据冗余；
- 解决大维表实时 SQL 查询需求；
- 强 Schema，有效避免潜在错误，节省时间。





#### 4) 实时推荐场合

刚才讲的几个场景，都是偏分析的场景。这边给大家介绍一个推荐的服务的场景。这个场景是实时推荐（特征查询、实时指标计算、向量检索召回），提高广告留存率，用到的主要技术有 PAI+Hologres(with Proxima)。客户收益包括：

- 支持 2000 万日活用户快速向量检索，千万级 u2u, i2i 均可以 20ms 返回；
- 通过 SQL 描述业务逻辑，无需手工编码；
- 加工逻辑简化，无需额外集群。

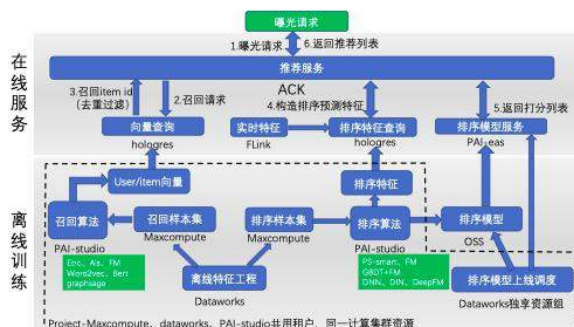
这是一个很典型的把数据作为服务的场景。而且数据并没有离开我们的数据平台，还是在我们这套数仓平台里面，还是以表的形式，所以在运维上、开发效率上都会提升很多。

### 实时推荐、API as service

实时推荐（特征查询、实时指标计算、向量检索召回），提高广告留存率，Flink+PAI+Hologres with Proxima

#### 客户收益

- 支持2000万日活用户快速向量检索，千万级u2u,i2i 均可以20ms返回
- 通过SQL描述业务逻辑，无需手工编码 (select a, b, proxima\_distance(c) as distance from table order by distance desc limit k;)
- 加工逻辑简化，无需额外集群 (Redis)





## 五、MaxCompute+Hologres 最佳实践

通过上述典型场景的应用，有些人可能就会有些疑问，是不是就不需要 MaxCompute 了呢？答案是否定的。其实 MaxCompute 和 Hologres 是最好的搭档。这两个产品在技术架构定位场景上是有些差异的，如下图所示，两个放在一起可以实现最好的效果。

	MaxCompute	MC-Hologres
使用场景	ETL	交互式分析Analytics、数据产品服务化Serving
用户使用	异步的OdpsJob/Instance/Task	同步的Query
集群资源	共享大集群	独享集群、共享只读大集群（MC加速）
计算引擎	基于Stage和File设计的，持久化的，基于内存的，超快速响应的SQL可扩展SQL Engine	Engine，计算不落盘
调度方式	进程级别，运行时分配	轻量级线程，资源预留
扩展性	几乎不受限制	复杂查询尽量避免跨多节点数据shuffle
存储格式	列式	行式、列式共存，面向不同场景
存储成本	基于Pangu，成本低	基于Pangu，利用SSD做缓存加速，成本相对高
接口标准	MC SQL	PostgreSQL

最后给大家讲一下 MaxCompute 和 Hologres 怎么放在一起，实现最好的一个加工服务一体化的效果。一般来讲，数仓分层为 ODS、DWD、DWS、ADS，如图所示。下面两层放在 MaxCompute 是比较好的，上面两层可以放在 Hologres。总的来说，有如下特点：

- 面向主题的开发，提供可复用的数仓模型；
- 加工服务一体化：减少数据移动，减少数据孤岛；
- 数仓建模敏捷化：减少数据层次，敏捷适应需求变化，面向 DWS、DWD 的应用开发。



一般来讲，在 Hologres 中加速查询 MaxCompute 有两种方式：

- 创建外表（数据还是存储在 MaxCompute 中），性能会有 2-5 倍的提升；
- 导入内表，性能相比外表大约有 10-100 倍的提升。

当我们设计的数仓，想要它跑的快，至少做好两件事：查询引擎优化和索引设计优化。

直接查外表的方式实际上是利用查询引擎的优化能力来提高效率的，但是没有利用到索引能力。所以当把外表导到内表的时候，可以根据查询的方式指定内表的索引结构。通过这些索引能力带来更高的查询性能。这就是外表导入内表，内表的性能更好的原因，就是要充分发挥数仓里边索引优化的能力。

# 开源 OLAP 升级 Hologres 详解

概要：本文将会为您介绍开源 OLAP 如 ClickHouse/Druid/Presto 等架构同 Hologres 的深度对比，并介绍如何如何平滑迁移到 Hologres 技术体系，实现更稳定，更可扩展，更多功能的 HSAP 架构。

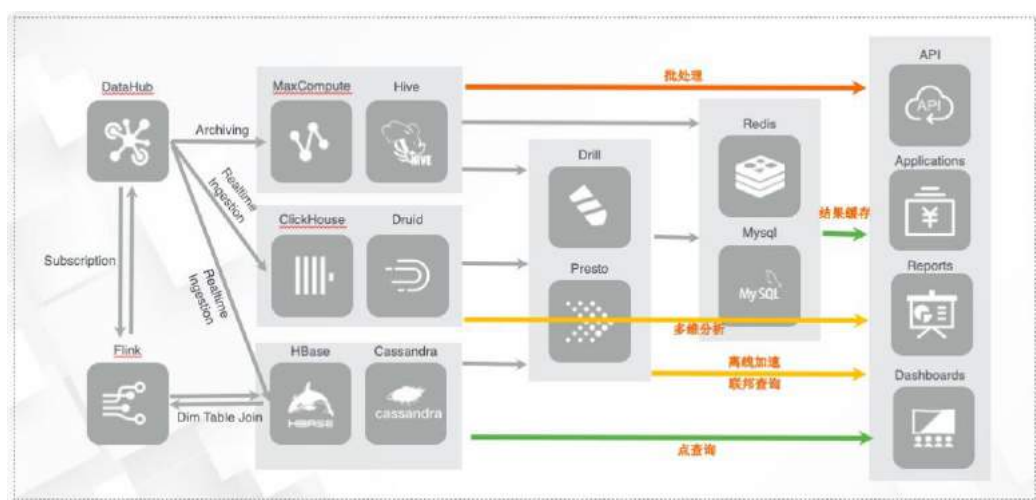
作者 | 夏晨 阿里云资深技术专家

## 一、OLAP 系统

从技术影响力的角度来说，Github Star 排名前 5 的系统也是目前市面上比较流行的开源 OLAP 系统，它们的开源时间以及产品的定位各不相同，我们将会选取排名前 3 的系统来与 Hologres 做深度的对比。

开源OLAP系统现状				
系统	Github Star	开源时长	语言实现	系统分类
ClickHouse	12k	4年	C++	存储+计算
Presto	11k	7年	Java	计算引擎
Druid	10k	8年	Java	存储+计算
Greenplum	4k	5年	C	存储+计算
Impala	3k	7年	C++	计算引擎

另外再从业务的视角来看一下大数据架构中 OLAP 系统的一个定位和功能，以及它在业务系统中的作用，可以看到，在整个大数据架构中，为了解决不同的问题，需要用到不同的系统，在大数据架构中，OLAP 系统百家争鸣、万花齐放。



## 二、开源系统对比

将会从以下几个方面来深度对比 Hologres 和开源 OLAP 系统。

### 1) 存储计算角度

在存储方面，Hologres 支持毫秒级的写入可见性，Druid 和 ClickHouse 支持的是秒级。同时 Hologres 支持写入更新。

在计算方面，Hologres 同 Druid 和 ClickHouse 都采用了向量化/SIMD 执行器，值得一提的是，因 Hologres 底层技术原理可以支持联邦查询和高 QPS 的点查，这些都是 Druid 和 ClickHouse 无法做到的。

		Druid	ClickHouse	Hologres
存储	写入可见性	秒级	秒级	毫秒级
	明细存储	不建议	支持	支持
	可更新	不支持	不完备	支持
	索引	bitmap/dictionary	primary key/clustering Key	bitmap/dictionary/segment/primary/cluster
计算	优化器	RBO	RBO	CBO
	执行器	-	向量化/SIMD	向量化/SIMD
	联邦查询	不支持	支持	支持
	预聚合	支持	支持	支持 (Flink)
	高QPS点查	不支持	不支持	支持

## 2) SQL、高级功能和生态

在 SQL 方面，Hologres 是兼容 Postgres 语法，开箱即可用，学习成本低，而 Druid 和 ClickHouse 的语法都是较复杂，难以上手。并且对于 update、delete、join 等，Druid 和 ClickHouse 的支持都是比较有限。

在高级功能上，Hologres 支持向量检索、空间数据等，支持更加丰富的业务场景，并且对于安全管控方面，Hologres 也有着非常严苛的权限管理，例如 RAM、IP 白名单等。

在生态上 Hologres 支持会更加丰富，Hologres 提供 JDBC 接口，且能通过 Flink 或者 DataX、StreamX 写入，实现多种异构数据源轻松导入 Hologres。

		Druid	ClickHouse	Hologres
SQL	DDL	复杂	语法复杂	语法简单
	Update/Delete	不支持	有限支持，语法复杂，非ANSI标准	支持
	JOIN	有限支持	支持，性能较差	支持
	WINDOW	不支持	不支持	支持
高级功能	向量检索	不支持	不支持	支持
	空间数据	不支持	不支持	支持
	安全管理	不支持	不支持	丰富的权限控制，RAM子账号，ip白名单
	扩缩容	分钟级	小时级，复杂	分钟级
生态	数据接入	Kafka/HTTP	Kafka/JDBC/DataX	Flink/JDBC/DataX
	BI工具	部分	部分	10+ 主流BI工具

## 三、迁移指南

对比完主流的开源 OLAP 引擎之后，肯定就会有人问，要是想把 OLAP 引擎迁移到 Hologres，应该怎么做呢？它们的不同点在哪里呢？下面将会一一讲述。

### 1) Presto 迁移

#### • Presto 简介及应用场景

Presto 是 Facebook 推出的一个分布式的 SQL 查询引擎，基于内存的并行计算，它可以提供比较快速的一个交互式查询能力。应用场景有离线加速，联邦查询，数据湖，以及实时数仓。



从 Presto 迁移到 Hologres 后，基本上功能可以无缝的得到支持，而且 Presto 不支持实时数仓，而 Hologres 支持实时数仓，提供写入和更新能力，并支持高 QPS 的查询能力。

## Presto简介及应用场景

Presto:

Facebook推出的分布式SQL查询引擎 基于内存的并行计算 提供交互式查询功能

应用场景	Presto	Hologres
离线加速	Hive加速	MaxCompute/Hive加速
联邦查询	MySQL/MongoDB等丰富的connector扩展	MaxCompute/Postgres等丰富的Postgres FDW扩展
数据湖	S3/OSS	OSS
实时数仓 (写入/更新/高QPS)	不支持	支持

### • 迁移方案

可以分别从数据模型，数据类型，SQL，数据接入，BI生态来看。数据模型上2者概念类似，只不过 Presto 的 Catalog 需要变成 Hologres 的 Database。数据类似方面 Hologres 同 Presto 一致都支持基本以及复杂类型，无需做修改。SQL 层面，2者都是支持 ANSI SQL，学习成本也比较低。对于数据接入和生态而言，Hologres 同 Presto 都提供 JDBC 接口，无需更换工具，只需要更换对应的连接信息就能立即使用

总结来看，迁移的成本是非常低的，基本上可以做到无缝的迁移。

## Presto迁移Hologres实践

功能	Presto	Hologres
数据模型	Catalog/Schema/Table	Database/Schema/Table
数据类型	基本类型/复杂类型	基本类型/复杂类型
SQL	ANSI SQL(Presto)	ANSI SQL (Postgres)
数据接入	客户端/JDBC	PSQL客户端/JDBC/HoloWeb
BI生态	Tableau/帆软等主流BI	Tableau/帆软主流BI+云QuickBI

**两者均支持ANSI SQL，迁移成本低，基本无缝**

## 2) Druid 迁移

- Druid 简介及应用场景

Druid 自带存储引擎，支持实时数据，并提供亚秒级查询的 OLAP 引擎。

Druid 的适用场景有：

- 时序组织数据（点击，曝光，监控）
- 数据质量要求不高（有可能重复/丢失）
- 海量 append only 数据（不支持更新）
- 单表聚合类查询

- Druid 不适用的场景

- 高 QPS 明细场景
- 有一致性要求的场景
- 需要更新的场景
- 需要 join 的场景

从 Druid 迁移到 Hologres 后，可以支持一些原来 Druid 不适用的场景。

- 迁移方案

下面来看实践，首先是数据模型的迁移。Druid 所有数据都存储在 dataSource 中，dataSource 类似于 Hologres 的表。Druid 通过 json 的方式来描述一个 dataSource 的 schema，dataSource schema 与 Hologres 的概念映射，如下图所示。

## 数据模型迁移

Druid所有数据都存储在dataSource中，dataSource类似于hologres的表，本章节介绍druid中的数据模型和Hologres的数据模型之间的关系以及如何迁移。

### Data Schema

Druid通过json的方式来描述一个dataSource的schema，dataSource schema与Hologres的概念映射如下：

Druid概念	Hologres概念	说明
<a href="#">dataSource</a>	Table	
<a href="#">timestampSpec</a>	Column	Druid中的timestamp在hologres中就是一个普通列，切分segment的属性通过在表中添加segment key的方式来指定。
<a href="#">dimensionsSpec</a>	Column	维度列在hologres中是一个普通列，在druid中维度列一般用来做过滤和聚合，用户可以通过在hologres中添加bitmap或dictionary的方式来加速用户的查询。
<a href="#">metricsSpec</a>	Column	指标列在hologres中是一个普通列。
<a href="#">granularitySpec</a>	-	无需配置
<a href="#">transformSpec</a>	-	在数据Ingestion到Hologres中时，用户通过SQL where子句指定需要过滤的数据。

#### • 迁移示例

下图是一个具体的从 Druid 迁移 Hologres 的示例。

就 DDL 而言，如上面所述，Datasource 修改为 table，其余的参数都配置为 Hologres 的字段信息即可。

## Druid迁移Hologres实践 - DDL

Druid Datasource	Hologres Table
<pre> {   "dataSource": "wikipedia",   "timestampSpec": {     "column": "ts",     "format": "auto"   },   "dimensionsSpec": {     "dimensions": [       { "type": "string", "name": "page" },       { "type": "string", "name": "language" },       { "type": "long", "name": "userId" }     ]   },   "metricsSpec": [     { "type": "doubleSum", "name": "bytes added sum", "fieldName": "bytes_added" },     { "type": "doubleSum", "name": "bytes deleted sum", "fieldName": "bytes_deleted" }   ] } </pre>	<pre> BEGIN; CREATE TABLE wikipedia (   ts timestamptz,   page text,   language text,   user_id bigint,   bytes_added_sum double precision,   bytes_deleted double precision ); CALL set_table_property('test', 'segment_key', 'ts'); -- 分段列，设成为druid的timestampSpec中指定的列 CALL set_table_property('test', 'bitmap_columns', 'page, language, user_id'); -- bitmap索引，设成druid的维度列 CALL set_table_property('test', 'dictionary_encoding_columns', 'page,language'); -- dictionary索引，设成druid维度列中的string 列 COMMIT; </pre>

再来看 Query 相关的 Query 迁移，Druid 当前版本支持两种查询方式：SQL 查询和 Native 查询，SQL 查询和 Hologres 的 SQL 查询方式类似，在此主要介绍 Native query 到 Hologres 的迁移方式。

Scan 查询修改为 Hologres 中的 Select 字段即可。

### Query 迁移

Druid 当前版本支持两种查询方式：SQL 查询和 Native 查询，SQL 查询和 Hologres 的 SQL 查询方式类似，在此主要介绍 Native query 到 Hologres 的迁移方式

Druid	Hologres
Scan 查询 { "queryType": "scan", "dataSource": "dataSourceName", "columns": ["column1", "column2"], "intervals": ["0000/3000"] }	SELECT column1, column2 FROM dataSourceName

TOPN 查询修改为 Sum 和 Group by 查询即可。

Druid	Hologres
TOPN 查询： { "aggregations": [ { "fieldName": "L_QUANTITY_longSum", "name": "L_QUANTITY_", "type": "longSum" } ], "dataSource": "tpch_year", "dimension": "l_orderkey", "granularity": "all", "metric": "L_QUANTITY_", "queryType": "topN", "threshold": 2 }	SELECT SUM(L_QUANTITY_longSum) AS L_QUANTITY_ FROM tpch_year GROUP BY l_orderkey ORDER BY L_QUANTITY_ LIMIT 2

Timeseries 查询修改 where 条件的 sum 查询。

Druid	Hologres
Timeseries 查询： { "queryType": "timeseries", "dataSource": "sample_datasource", "granularity": "day", "descending": "true", "filter": { "type": "and", "fields": [ { "type": "selector", "dimension": "sample_dimension1", "value": "sample_value1" }, { "type": "or", "fields": [ { "type": "selector", "dimension": "sample_dimension2", "value": "sample_value2" }, { "type": "selector", "dimension": "sample_dimension3", "value": "sample_value3" } ] } ] }, "aggregations": [ { "type": "longSum", "name": "sample_name1", "fieldName": "sample_fieldName1" }, { "type": "doubleSum", "name": "sample_name2", "fieldName": "sample_fieldName2" } ] }, "intervals": [ "2012-01-01T00:00:00.000/2012-01-03T00:00:00.000" ] }	SELECT SUM(sample_fieldName1) as sample_name1, SUM(sample_fieldName2) as sample_fieldName2 FROM sample_datasource WHERE sample_dimension1 = 'sample_value1' AND (sample_dimension2 = 'sample_value2' OR sample_dimension3 = 'sample_value3' )

GroupBy 查询在 Hologres 中也是 Group by，无需做大幅度的修改。

Druid	Hologres
GroupBy查询： <pre>{   "type": "query",   "query": {     "queryType": "groupBy",     "dataSource": "site_traffic",     "intervals": ["0000/3000"],     "granularity": "all",     "dimensions": ["page"],     "aggregations": [       { "type": "count", "name": "hits" }     ]   } }</pre>	<pre>SELECT   page,   COUNT(*) AS hits FROM   site_traffic GROUP BY   page</pre>

### 3) ClickHouse 迁移

#### • ClickHouse 简介及应用场景

ClickHouse 自带存储引擎，支持实时数据，并提供亚秒级查询，C++实现的 OLAP 分布式列式数据库。

适用场景有：

- 海量明细数据存储
- 单表聚合类查询
- 写入可见性/一致性要求不高

不适用的场景有：

- 高 QPS 查询
- 高 QPS 更新
- 需要 join 的场景

#### • 迁移方案

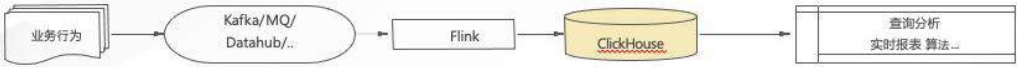
常见实时数仓写入链路如下图所示，通过 Flink 将实时日志数据写入到 ClickHouse 中。这里将会涉及到两种数据迁移。



首先是增量数据迁移，Hologres 紧密结合 Flink 生态，如果客户之前是通过 Flink 来向 ClickHouse 写入实时数据，那么通过 Hologres Connector 就能可以方便地将数据导入 Hologres，将增量数据迁移过来。

然后是存量数据迁移，由于 DataX 等插件，暂不支持 ClickHouse Reader。如果已经在 ClickHouse 中存储了存量数据，需要搬迁至 Hologres，目前可以用 ClickHouse -> COPY OUT -> Data File -> COPY IN -> Hologres 的链路，需要确定好字符集、分隔符和 NULL 标记等规范就能迁移。

### 数据写入链路迁移



常见实时数仓写入链路如上图，通过Flink将实时日志数据写入到实时数仓系统中。

#### 增量数据迁移

Hologres紧密结合Flink生态，如果客户之前是通过Flink来向ClickHouse灌入实时数据，那么通过集成Hologres Connector，可以方便地将数据导入Hologres，将增量数据迁移过来。

#### 存量数据迁移

由于DataX等插件，暂不支持ClickHouse Reader。如果已经在ClickHouse中存储了存量数据，需要搬迁至Hologres，目前可以用ClickHouse -> COPY OUT -> Data File -> COPY IN -> Hologres的 链路，需要确定好字符集、分隔符和NULL标记等规范

迁移过程中，数据模型的对应关系，如下图所示。

	ClickHouse	Hologres
DB	create database xx;	create database xx;
Table Group/Shard	对所有的<SHARD>，<REPLICA>： ENGINE = ReplicatedMergeTree ( '/path/to', 'cluster01-<SHARD>-<REPLICA>', dt, (cmd, dt, log_timestamp), 8192)	一键设置 begin; create table xx (a int); Call set_table_property('xx', 'shard_count', '100'); commit;
TABLE	create table xx (a int) ENGINE = engine;	无需engine设置
	默认列存	默认列存，支持行存
	表级TTL	表级TTL
COLUMN	列级TTL	不支持
	CONSTRAINT	支持primary key, default , not null等
VIEW	支持Materialized view	支持VIEW，不支持Materialized view

## • 迁移示例

迁移的 DDL 示例如下图所示，建表语句可以保持一致，但是索引的创建需要改成 call set 语句。

ClickHouse	Hologres
<pre>CREATE TABLE test (   'a' Int64 DEFAULT CAST(0, 'Int64'),   'b' Float64 DEFAULT 0.,   'c' String DEFAULT "",   'd' DateTime DEFAULT CAST('0000-00-00 00:00:00',     'DateTime'), ) ENGINE = ReplicatedMergeTree('/clickhouse/tables/{layer}- {shard}/test', '{replica}') PARTITION BY formatDateTime(d, '%D') ORDER BY d SETTINGS index_granularity = 8192;</pre>	<pre>BEGIN; CREATE TABLE test (   a bigint DEFAULT 0,   b double precision DEFAULT 0.,   c text DEFAULT "",   d timestampz DEFAULT 0. ); CALL set_table_property('test', 'clustering_key', 'd'); -- 聚簇列（段内 排序列），与ClickHouse的ORDER BY类似 CALL set_table_property('test', 'segment_key', 'd'); -- 分段列，与ClickHouse的PARTITION BY类似，注：PARTITION多的 表，也可以改造成Holo的分区表 CALL set_table_property('test', 'bitmap_columns', 'c'); -- bitmap索引 CALL set_table_property('test', 'shard_count', '100'); -- shard_count根据数据量而定 COMMIT;</pre>

对于一些典型查询 Query，只需要修改部分的语法即可达到更好的效果。

	ClickHouse	Hologres
<u>Identifier quote</u>	`, "" 支持反引号，双引号	"" 只支持双引号
QUANTILE	quantile(level)(expr)	approx_percentile(level) WITH GROUP (ORDER BY expr)
	quantileDeterministic(level)(expr, determinator)	-
	quantileExact(level)(expr)	percentile_cont(level) WITH GROUP (ORDER BY expr)
DISTINCT	uniq(x)	approx_count_distinct(x)
	uniqCombined(x)	
	uniqCombined64(x)	
	uniqHLL12(x)	
	uniqExact(x)	
WINDOW	不支持	支持（参考PG或Holo官方文档）
JOIN	支持	支持（参考PG或Holo官方文档）
其他 - 可参考官方文档		

## 四、案例分享

从 Hologres 诞生到赋能阿里巴巴集团内大多数核心业务再到云上商业化,已经沉淀无数开源 OLAP 成功迁移 Hologres 的案例。下面将会介绍典型的成功迁移案例。

### 1) 搜索业务

在最开始的时候,阿里巴巴的搜索推荐的业务,用到的是一个非常复杂的架构,如下图所示。我们把这个业务架构仔细拆开来,它具有需要系统具备多种能力:

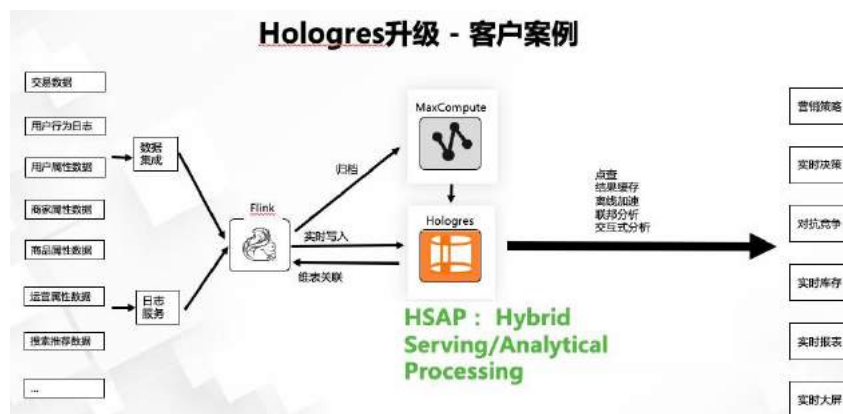
第一, KVStore: Redis/Mysql/Hbase/Cassandra 存储能力。

第二, 交互式计算能力: Presto/Drill 计算能力。

第三, 实时数仓: Clickhouse/Druid 存储+计算。



正是因为这样一些复杂的业务架构,我们做了一个 HSAP 的系统,进行了业务的架构升级,把多种能力有机统一于一个引擎。这样,用户只需要通过 Hologres 就可以解决问题,让用户的学习成本降低非常多,不需要再去学习多个系统。升级后的架构:



下图是具体的迁移表现。迁移到 Hologres 后，资源节省 60%。平均查询性能大幅提升，开发周期显著加快。

阿里集团 - 搜索推荐业务

指标	Druid	Hologres
资源	-	节省60%
平均查询性能	300ms	复杂查询100ms，点查5ms
写入可见性	5s	<1ms
写入QPS	千万	1.3亿
查询量	千万	1.5亿
开发周期	周	小时

## 2) 某社交网站替换 ClickHouse

在阿里云上也有成功的迁移案例，例如某个社交网站的客户成功从 ClickHouse 迁移到 Hologres，资源也节省了几百 core，原来只能存储 7 天的数据，现在能无限制存储并支持 7-15 天的明细数据复杂查询，相比之前的实时写入 QPS 也有大幅提升。

公共云 - 某社交网站客户

指标	ClickHouse	Hologres
资源	1320core	1024core
存储	只能存7天	无限制
复杂查询	只能查3天	7-15天
实时写入QPS	<30k/s	> 40k/s
写入可见性	秒级	毫秒级
查询性能	-	提升2-5倍

## 3) 某游戏客户替换 Redis

下面是公共云上某个游戏客户替换 Redis 后的收益，首先是成本下降了 50%，也支持了复杂查询，学习成本也降低了，无需特别维护，就能开箱即用。

公共云 - 某游戏客户

指标	Redis/Faiss	Hologres
成本	-	节省50%
扩展性	单机	分布式
复杂查询	不支持	支持
学习成本	高	低
业务耦合度	高	低
易用性	低	高

五、总结

Hologres 作为 HSAP 服务分析一体化的最佳落地实践，。支持联邦查询，通过 Hologres 完全无缝的加速离线场景。同时也能解决 AP 常见的能力，同时比 AP 系统的性能更好。并且还能解决 HBASE 这种点查系统的能力，可以解决实时推荐的一些场景的能力。

在整个大数据架构中，通过 Hologres 完成了非常好的业务架构的整合，让业务能力得到非常大的扩张，用户可以非常方便的使用一套系统来完成各种各样的业务。







Hologres 用户交流群



Hologres 官网



Hologres 开发者社区技术圈



阿里云开发者“藏经阁”  
海量免费电子书下载