火山引擎

VeDI Volcengine Data Intelligence

云原生数据仓库 ByteHouse 技术白皮书

目录

第一章
ByteHouse简介

第一章 ByteHouse简介

ByteHouse是火山引擎自主研发的云原生数据仓库产品，在开源ClickHouse引擎之上做了技术架构重构，实现了云原生环境的部署和运维管理、存储计算分离、多租户管理等功能。在可扩展性、稳定性、可运维性、性能以及资源利用率方面都有巨大的提升。

截至2022年2月，ByteHouse在字节跳动内部部署规模超过1万8000台，单集群超过2400台。经过内部数百个应用场景和数万用户锤炼，并在多个外部企业客户中得到推广应用。

1.1 产品特性

ByteHouse以提供高性能、高资源利用率、高稳定性、低运维成本为目标，进行了优化设计和工程实现，产品特性和优势如下：

1.存储计算分离：解决了全局元数据管理，过多小文件存储性能差等等技术难题。在最小化性能损耗的情况下，实现存储层与计算层的分离，独立扩缩容。

2.新一代 MPP 架构：结合 Shared-nothing 的计算层以及 Shared-everything的存储层，有效避免了传统 MPP 架构中的 Re-sharding 问题，同时保MPP并行处理能力。

3.数据一致性与事务支持。

4.计算资源隔离，读写分离：通过计算组(VW)概念，对宿主机硬件资源进行灵活切割分配，按需扩缩容。资源有效隔离，读写分开资源管理，任务之间互不影响，杜绝了大查询打满所有资源拖垮集群的现象。

5.ANSI-SQL：SQL兼容性全面提升，支持ANSI-SQL 2011标准TPC-D测试集100%通过率。

6.UDF：支持Python UDF/UDAF创建与管理，补足函数的可扩展性。(JavaUDF/UDAF已在开发中)

7.自研优化器：自研Cost-Based Optimizer，优化多表JOIN等复杂查询性能，性能提升若干倍。

产品能力上，在引擎外提供更加丰富的企业级功能和可视化管理界面：

1.库表资产管理：控制台建库建表，管理元信息。

2.多租户管理：支持多租户模型，租户间互相隔离，独立计费。

RBAC权限管理：支持库、表、列级，读、写、资源管理等权限。通过角色进行管理。

3.VW自动启停，弹性扩展：计算资源按需分配，闲时关闭。降低总成本，提高资源使用率。

4.性能诊断：提供Query History和Query Profiler功能，帮助用户自助地排查慢查询的原因。

1.2 适用场景

ByteHouse定位为一款数据仓库产品，主要用于OLAP查询和计算场景。在实时数据接入、大宽表聚合查询、海量数据下复杂分析计算、多表关联查询场景下有非常好的性能。

主要的的应用场景如下：

场景分类	场景	描述	特点
交互式查询	用户自定义查询	支持多维查询分析的数据应用	自由维度、多表关联、响应快
	自助式报表	支持Tableau等BI工具	自由维度、多表关联、响应快
	用户画像分析	支持DMP等圈人画像平台	自由维度、多表关联、响应快
	营销效果分析	支持流量效果漏斗分析	多表关联、实时
	行为日志分析	支持日志探索分析	日志检索、数据量大
实时数据看板	实时业务监控大屏	支持DataV等可视化大屏	实时
	直播数据统计看板	支持实时报表	实时
	业务仪表盘	支持报表工具	统计、响应快
	系统链路监控	支持实时监控应用	实时
实时数据仓库	实时数据接入	支持实时数据写入、更新	实时数据写入，立即可见
	准实时ETL计算	支持复杂计算，数据清洗	混合负载

第二章
技术趋势和挑战

第二章 技术趋势和挑战

2.1 业务需求

企业级数据仓库场景中，需要融合来自多个业务系统数据库的业务数据，主要是交易记录，例如银行存取记录、用户订单记录等，通常是数千万至数亿条规模；用户行为日志是数据量最大的数据源，包括用户访问日志、用户操作记录等，这部分数据记录数量通常是业务数据的数百倍。

ByteHouse需要支持海量数据的实时接入、无限扩展存储、实时合并计算和关联聚合查询。

随着大数据应用的深入发展，最核心的业务需求如下：

1.提高分析的实时性:最近10年，以hadoop技术体系为代表的大数据平台大规模部署，大大小小的企业和政府部门都搭建了大数据平台和分析应用，以隔天和小时级数据延迟的应用得到了普及；以Flink为代表的实时计算引擎解决了数据统计场景的时效性问题。

随着业务的发展和技术的进步，业务部门不再满足于T+1的分析需求和固化的实时统计，希望业务发生后秒级/分钟级延迟就能看到统计结果；希望能交互性探查分析数据，要求毫秒/秒级返回结果保持良好的用户体验。

在新的企业级数据架构中，对于已经构建大数据平台的企业，对时效性要求高的业务，用云原生数据仓库构建实时数据仓库，作为hadoop平台的补充；在数据量低于1PB，没有构建hadoop等大数据平台的企业，直接以云原生数据仓库构建轻量级数据仓库。

2.成本可控:大数据应用逐步从互联网企业和政府部门，并深入到工业企业，先后进行了业务数据的大集中、用户行为数据和IOT数据的广泛采集存储，企业和政府单位的数据量每年呈现30%以上的增长速度。

在过去集中式架构的数据仓库方案中，建设成本与数据总量正相关，成本居高不下；采用基于分布式架构的大数据方案中，由于存储计算耦合，为了满足存储空间膨胀，需要采购越来越多的服务器。

实时的数据采集和存储，导致数据量持续高速增长。

在新的云原生数据仓库方案中，既要解决数据和应用增长带来的扩展性问题，同时要解决成本问题，将数据存储和计算成本处于可控范围。

3.支持业务上云:根据智库报告的研究，目前业务上云已经形成趋势，除游戏视频电商等泛互联网企业之外，在政务、金融、制造业正在以私有云和混合云的方式持续上云，从而实现数据上云。

政务云和金融云是两大主要的行业云，平台建设水平较高，同时制造业、医疗卫生、交通等领域的行业云也在加速变革和加快建设行业云平台大规模建设和升级，实现数字化管理和运营。

制造业设备上云和云化改造能够实现制造业企业的数据互通和业务互联，支撑形成以数据驱动的智能化制造、实现供应链和上下游业务的网络化协同，以及实现对业务和设备的数字化管理等制造业发展新模式，引领制造业数字化转型。

业务上云从而数据上云，也在推动数据处理平台的云原生升级。

2.2 技术趋势

近年来，以Snowflake为代表的云原生数据仓库得到了客户的认同，市场上取得了巨大的成功。其核心功能和技术点是云原生的架构设计，利用IAAS的高可用和资源池化特性，通过存储计算分离、多租户隔离、容器化技术，提供数据仓库的扩展性、稳定性、可维护性和易用性，整体上提高资源利用率。

国际上，除了Snowflake之外，谷歌的BigQuery、AWS的RedShift、Azure的Synapse都实现了云原生的架构升级，实现了存储计算分离和多租户管理。Databricks、Firebolt等新生的厂商及产品如雨后春笋一样涌现出来。

在国内，阿里云、华为云、腾讯云都推出了自己的云原生数据仓库产品；PingCap的TiDB、鼎石科技的StarRocks等独立产品也选择了云原生道路。

OLAP产品有如下几个技术趋势：

1.云原生的整体架构:基于公共云、私有云或混合云的架构设计，应用容器化和微服务等云原生技术，实现敏捷开发、敏捷运维，天然解决扩展性问题。

2.存储服务化:对数据存储层进行统一抽象，灵活采用HDFS分布式存储或S3等对象存储作为数据存储载体，最终实现存储服务化，便于解决存储扩展性、读写吞吐瓶颈问题、数据一致性问题，同时能大幅降低存储成本。

此外，实现存储服务化后，对于产品的跨云兼容和多云部署带来方便。

3.计算资源池化:由于OLAP应用负载的波动特点，特别在支持多租户的场景下，通过计算资源池化，根据实时负载进行计算资源统一调度管理，实现资源隔离的同时，又能支持资源共享和实时弹性扩缩。从而提高集群整体利用率。

4.支持混合负载:在企业级应用中，OLAP场景可以细分为交互查询和批量计算，前者要求毫秒/秒级响应并支持高并发查询，后者可以接受分钟/小时级延迟，但要求计算性能的稳定性和较好的failover机制。自适应支持多场景的混合负载是OLAP产品的核心能力。

5.其他:OLAP平台中的计算资源、内存、网络带宽是最宝贵的资源，系统资源利用率通常围绕这三个资源进行优化。很多产品开始在计算Serverless化、分布式内存等方向进行探索。

2.3 技术挑战

ClickHouse是近几年最热门的开源大数据产品，以其优异的查询性能引人瞩目，在全球得到了大量的推广和应用。字节跳动从2017年开始大规模使用ClickHouse，总部署规模超过1万8000台，投入巨大的研发团队，对ClickHouse进行了大量的优化和

化和改进，积累了丰富的应用场景和应用经验。

ByteHouse的云原生技术方案也遇到了非常多的技术挑战，主要表现在几个方面：

1.数据实时写入性能:在大批量实时数据写入场景下，需要平衡数据一致性与写入吞量的矛盾，特别在存储计算分离后，远程数据访问网络开销加大，写入性能问题会显得更加突出，需要有新的解决方案。

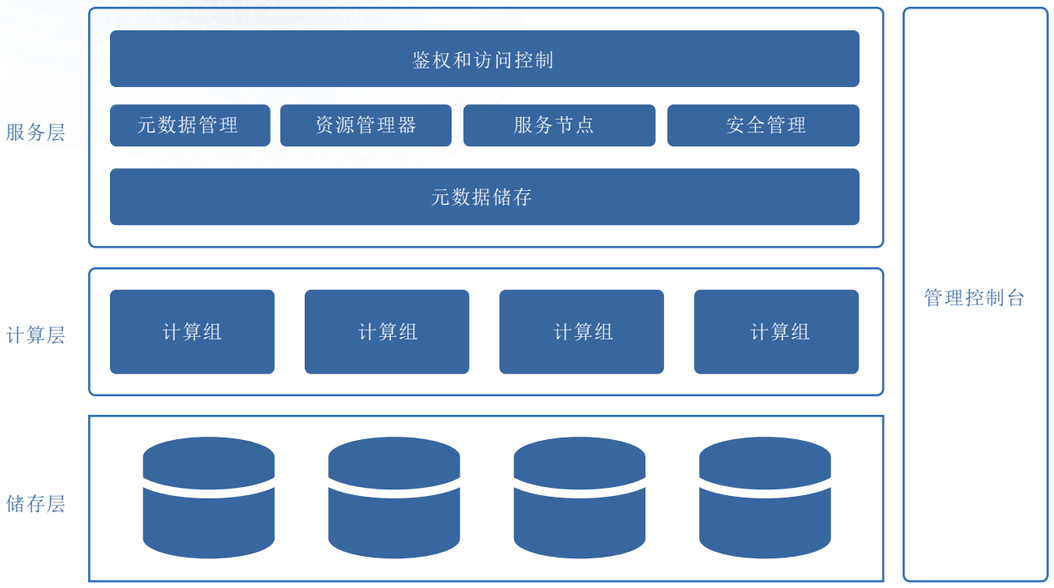
2.多场景下查询性能:ClickHouse以单表查询性能好著称，但在多表关联查询方面性能不理想，极大地限制了ClickHouse的应用场景。ByteHouse定位为综合性能强的云原生数仓，需要兼顾多种应用场景下都能把持优异的性能。

3.资源弹性和隔离:ByteHouse旨在提高整个集群的资源利用率，从而降低平台建设成本。由于OLAP应用的负载通常具有峰谷特性和随机性，要求具备资源弹性共享和资源隔离的能力，在保证性能和SLA的情况下降低资源成本。

4.提升产品易用性：ByteHouse需要提供计算资源管理、数据资源管理、数据接入、数据应用的可视化管理功能，降低维护和应用成本，成为真正SaaS化的云原生数据仓库产品。

第三章
整体架构设计

第三章 整体架构设计



ByteHouse整体架构图

云原生数据仓库ByteHouse总体架构图如上图所示，设计目标是实现高扩展性、高性能、高可靠性、高易用性。从下往上，总体上分服务层、计算层和存储层。

• 3.1 服务层

服务层包括了所有与用户交互的内容，包括用户管理、身份验证、查询优化器，管理、安全管理、元数据管理，以及运维监控、数据查询等可视化操作功能。

服务层主要包括如下组件：

1.资源管理器：资源管理器（Resource Manager）负责对计算资源进行统一的管理和调度，能够收集各个计算组的性能数据，为查询、写入和后台任务动态分配资源。同时支持计算资源隔离和共享，资源池化和弹性扩缩等功能。资源管理器是提高集群整体利用率的核心组件。

2.服务节点：服务节点（CNCH Server）可以看成是Query执行的master或者是coordinator。每一个计算组有1个或者多个CNCH Server，负责接受用户的query请求，解析query，生成逻辑执行计划，优化执行计划，调度和执行query，并将最终结果返回给用户。服务节点是无状态的，意味着用户可以接入任意一个服务节点（当然如果有需要，也可以隔离开），并且可以水平扩展，意味着平台具备支持高并发查询的能力。

3.元数据服务：元数据服务（Catalog Service）提供对查询相关元数据信息的读写

Metadata主要包括2部分：Table的元数据和Part的元数据。表的元数据信息主要包括表的Schema，partitioning schema，primary key，ordering key。Part的元数据信息记录表所对应的所有data file的元数据，主要包括文件名，文件路径，partition，schema，statistics，数据的索引等信息。

元数据信息会持久化保存在状态存储池里面，为了降低对元数据库的访问压力，对于访问频度高的元数据会进行缓存。

元数据服务自身只负责处理对元数据的请求，自身是无状态的，可以水平扩展。

4.安全管理：权限控制和安全管理，包括入侵检测、用户角色管理、授权管理、访问白名单管理、安全审计等功能。

• 3.2 服务层

通过容器编排平台（如 Kubernetes）来实现计算资源管理，所有计算资源都放在容器中。

计算组是计算资源的组织单位，可以将计算资源按需划分为多个虚拟集群。每个虚拟集群里包含0到多台计算节点，可按照实际资源需求量动态的扩缩容。

一个租户内可以创建1个或多个计算组，计算资源扩缩容的方式有两种，一种是调整计算组的CPU核数和内存大小实现快速的纵向扩缩容，另一种方式是增减计算组的数量实现水平扩容，在存储计算分离的架构下，计算资源与存储资源是解耦的且无状态的，扩缩容过程不需要迁移和平衡数据，因而可以实现快速弹性扩缩容。

计算节点主要承担的是计算任务，这些任务可以是数据写入、用户查询，也可以是一些后台任务。用户查询和后台任务，可以共享相同的计算节点以提高利用率，也可以使用独立的计算节点以保证严格的资源隔离。用户可以根据计算任务的特性、优先级和业务类别不同，构建多个计算组，并设置不同的资源弹性策略，提高计算效率降低成本。

• 3.3 存储层

采用HDFS或S3等云存储服务作为数据存储层，用来存储实际数据、索引等内容。

数据表的数据文件存储在远端的统一分布式存储系统中，与计算节点分离开来。底层存储系统可能会对应不同类型的分布式系统。例如HDFS，Amazon S3，Google cloud storage，Azure blob storage，阿里云对象存储等等。

不同的分布式存储系统，例如S3和HDFS有很多不同的功能和不一样的性能，会影响到功能的设计和实现。例如hdfs不支持文件的update，S3 object move操作时重操作需要复制数据等。

通过存储的服务化，对计算层提供统一的抽象文件系统接口，存储层采用S3还是HDFS对计算层透明；计算层可以支持ByteHouse自身的计算引擎之外，将来还可以便捷地对接其他计算引擎，例如Presto、Spark等。

采用块存储或对象存储作为共享的存储层，带来的好处是多方面的：

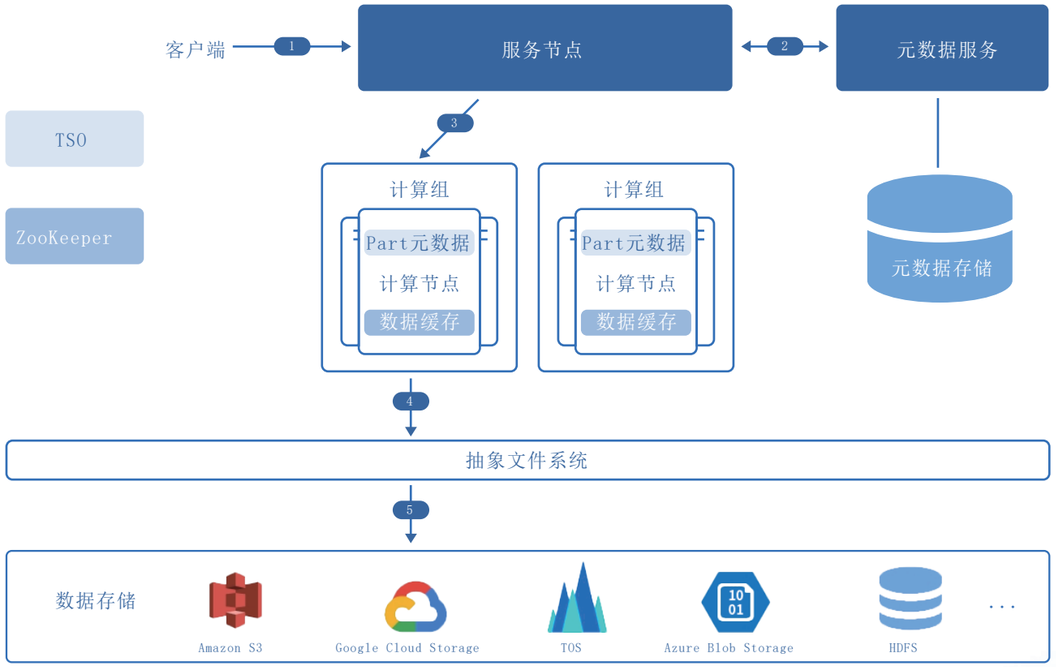
1.首先底层存储是天然支持高可用
2.存储容量可以无限扩缩
3.扩容时无需做数据均衡

3.4 作业执行流程

ByteHouse中的作业按照响应优先级分为3大类：Read query、Write query和Background的作业。不同类型的作业，按照前面所述，可以运行同一个工作节点上，也可以分离开来。

3.4.1 数据查询流程

服务节点负责响应和接受用户查询请求，并调度到相应的计算组中去执行，并回传结果给服务节点。各个计算节点执行完子查询之后， 很多时候会有相应计算结果要集中处理，如果希望这一层有计算组的隔离，务节点的部分功能例如聚合最终结果需要下放到计算组中的计算节点中去。



Read Query模块交互图

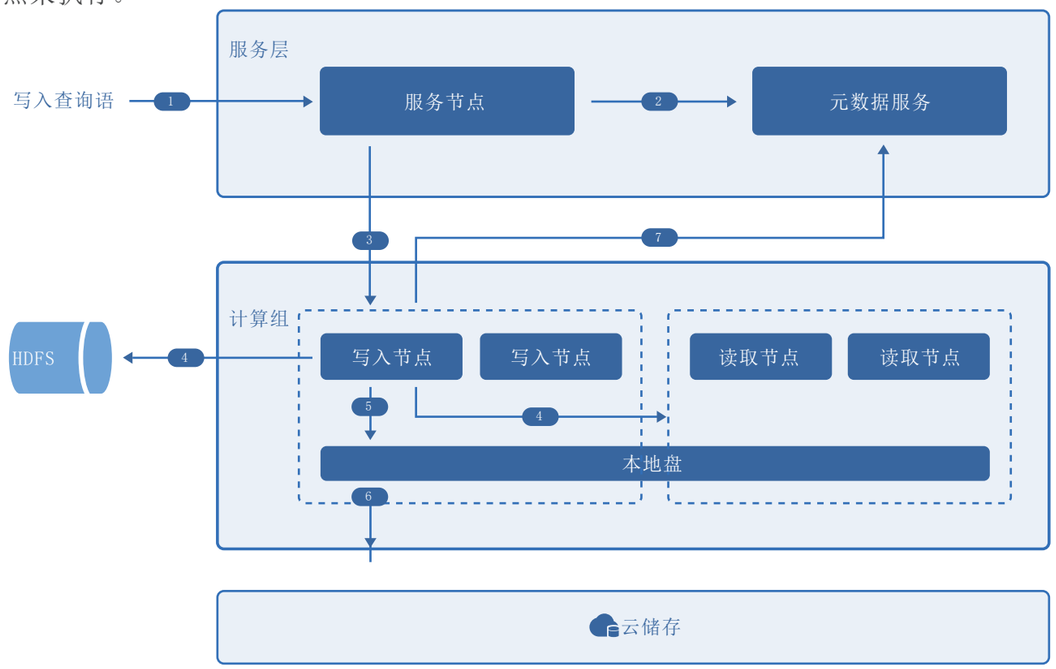
Query的执行过程：

1.用户提交Query到服务节点。
2.从元数据服务获取需要的元数据信息，对Query进行Parse，Planning，Optimize，生成执行计划。
3.服务节点对Query进行调度。

4.计算节点接收到Query子查询。
5.Query从远程文件系统获取原始数据，并根据Query的执行计划在计算节点上执行，并发回计算结果给服务 节点汇总。

3.4.2 数据写入流程

ByteHouse实现了读写分离，有单独写入节点来执行写入请求，写入请求分为几类：insert values，insert infile， insert select，insert values可能包含大量数据集，为避免网络传输开销直接由服务节点本地执行insert而无需转发给写入节点来执行。



Write Query模块交互图

Query的执行过程：

1.用户提交Write Query到服务节点。
2.服务节点从元数据服务获取需要的元数据信息，对Query进行parse，planning，optimize，生成执行计划，根据写入类型分为以下两种模式来执行：
a.Local模式：insert values操作直接由服务节点跳转到步骤四直接执行。
b.分布式模式：对于insert infile/select模式直接将执行计划信息分发给一个写入
节点执行。
3.服务节点对写入请求根据调度策略选择合适的写入节点执行。
4.写入节点从读取节点(insert select)或者外部存储(insert infile hdfs)读取数据流。

7.写入节点 更新元数据。

3.4.3 数据写入流程

为了更好的查询性能，会有一些作业在后台对写入的数据进行更进一步的处理。ByteHouse中主要包括如下3种后台任务：

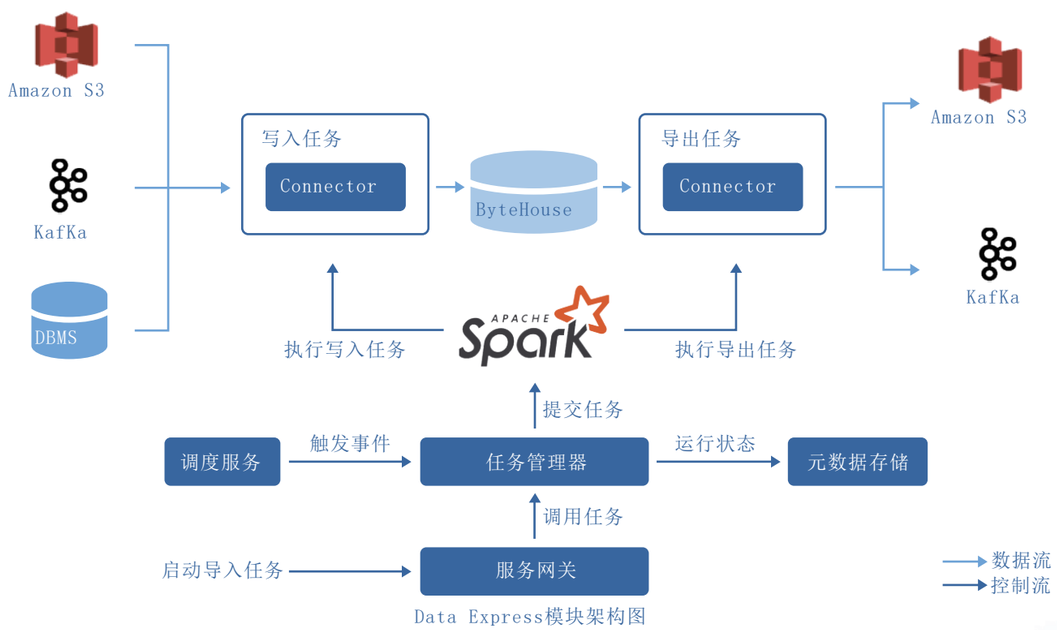
Merge：将不同的parts文件按Primary Key做排序合并成一个大的part文件。

Checkpoint： 对表的任意更新，例如元数据的改变，数据字典等异步构建操作会产生新的增量数据文件，这部分新产生的增量和原有的数据文件会在后台合并成一个新的数据文件。

GC：空间回收，当数据文件中的垃圾空间超过一定阈值后，会触发后台作业回收空间。

• 3.5 数据导入导出

ByteHouse包括一个数据导入导出（Data Express）模块，负责数据的导入导出工作。



Data Express模块架构图

Data Express 为数据导入/导出作业提供工作流服务和快速配置模板，用户可以从提供的快速模板创建数据加载作业。

DataExpress 利用 Spark 来执行数据迁移任务。

主要模块：

JobServer

导入模板

导出模板

JobServer 管理所有用户创建的数据迁移作业，同时运行外部事件触发数据迁移任务。启动任务时，JobServer 将相应的作业提交给 Spark 集群，并监控其执行情况。作业执行状态将保存在我们的元存储中，以供 Bytehouse 进一步分析。

ByteHouse支持离线数据导入和实时数据导入。

3.5.1 离线导入

离线导入数据源：

Object Storage：S3、OSS、Minio

Hive (1.0+)

Apache Kafka /Confluent Cloud/AWS Kinesis

本地文件

RDS

离线导入适用于希望将已准备好的数据一次性加载到 ByteHouse 的场景，根据是否对目标数据表进行分区，ByteHouse 提供了不同的加载模式：

全量加载：全量将用最新的数据替换全表数据。

增量加载：增量加载将根据其分区将新的数据添加到现有的目标数据表。ByteHouse 将替换现有分区，而非进行合并。

支持的文件类型：

ByteHouse的离线导入支持以下文件格式：

- Delimited files (CSV, TSV, etc.)
- Json (multiline)
- Avro
- Parquet
- Excel (xls)

3.5.2 实时导入

ByteHouse 能够连接到 Kafka，并将数据持续传输到目标数据表中。与离线导入不同，Kafka 任务一旦启动将持续运行。ByteHouse 的 Kafka 导入任务能够提供 exactly-once 语义。您可以停止/恢复消费任务，ByteHouse 将记录 offset 信息，确保数据不会丢失。

支持的消息格式：

ByteHouse 在流式导入中支持以下消息格式：

- Protobuf
- JSON

更多的导入数据源以及导出功能正在不断完善。

• 3.6 多租户管理

多租户管理架构图

ByteHouse的计算资源、数据资源、作业任务和用户权限都用租户进行隔离，所有的数据对象和资源都在一个租户内部进行管理。

不同的业务团队可以建立各自的租户，按额度申请所需的计算资源，便于进行资源管理和结算。计算资源隔离在租户内部，屏蔽租户之间的资源争抢。

数据库、数据表、视图等对象都在租户内部进行管理和授权，数据安全限制在租户内部。

数据查询、数据导入任务也在各自租户中，增加了任务代码安全性。

多租户管理功能适应了整个企业资源集中统一管理、按需按份额使用、兼顾资源共享和数据安全要求，同时可以为SaaS应用提供支撑，能按需为新用户申请资源，做到即开即用，又能满足不同用户资源和数据隔离性需求，实现一套系统服务所有用户。

• 3.7 运维监控管理

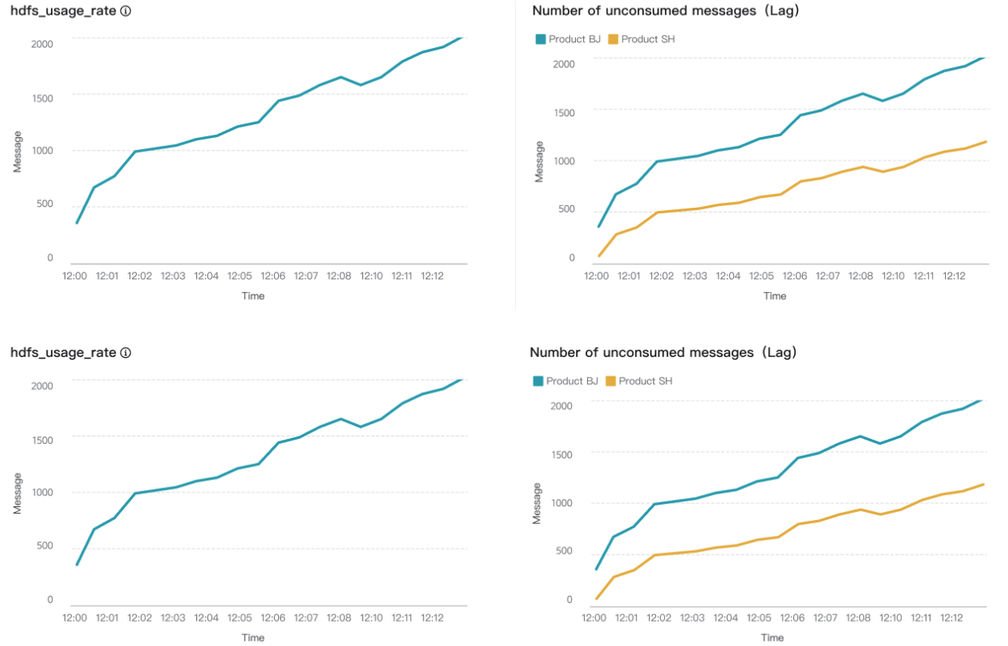
ByteHouse的私有化部署版本包含一个可视化的资源监控和管理平台，提供资源、负载监控仪表盘，直观地展现集群整体状况，同时提供租户管理、报警监控、审计日志、扩缩容、系统升级、故障节点替换等核心功能，让运维人员通过白屏化操作，降低运维成本和操作风险。

集群管理维护模块包括对物理资源的配置、节点重启、故障节点一键替换、滚动升级、滚动重启等功能，实现可视化运维管理。

通过仪表板对集群健康度进行宏观监控，集群资源饱和度监控能实时查看存储计算的当前应用情况和增长趋势，方便进行扩缩容；节点健康度监控能实时监控节点实时的响应情况；集群负载监控能实时反应集群总体负载水位；提供Grafana对各个组件运

行状态进行细粒度监控。HDFS对计算层透明；计算层可以支持ByteHouse自身的计算引擎之外，将来还可以便捷地对接其他计算引擎，例如Presto、Spark等。

采用块存储或对象存储作为共享的存储层，带来的好处是多方面的：



运维监控模块示意图

监控报警模块提供与第三方报警平台对接能力，支持对CPU、内存、存储资源使用量指标、技术组件健康度指标、计算任务状态指标、集群负载和性能指标进行监控，并通过短信、电话等方式通知值班员。

第四章
核心技术解析

第四章 核心技术解析

4.1 元数据管理

元数据管理（Catalog Service）的功能主要是对读写请求的元数据进行读写操作。元数据服务是一个非常关键的服务，需要保证其自身的高可用和元数据的一致性，元数据服务的扩展性影响整个平台的扩展性，此外元数据读写的性能也影响整个读写过程的性能。

元数据管理需要重点考虑下面几个方面的问题，元数据的持久化，和利用缓存对元数据层的加速。

4.1.1 元数据持久化

元数据的持久化，可以有很多不同的存储后端可供选择，例如KV型数据库，传统数据库，New SQL。经过综合考虑，最后决定选择KV数据库，目前采用字节内部产品ByteKV，外部开源的FoundationDB也是其他产品常见选择。

对于KV数据库里面需要存储的元数据信息主要有版本、统计信息、事务信息、数据的 Schema、Partition信息、Part的信息等。

4.1.2 元数据缓存

由于我们将Part级元数据存储在 ByteKV 中，因此在查询大数据范围时，是 KV 数据库的 Scan 操作，获取Part元数据的时间较长，且给 ByteKV 带来很重的负担。因此通过增加一个缓存层提高性能、降低负载。

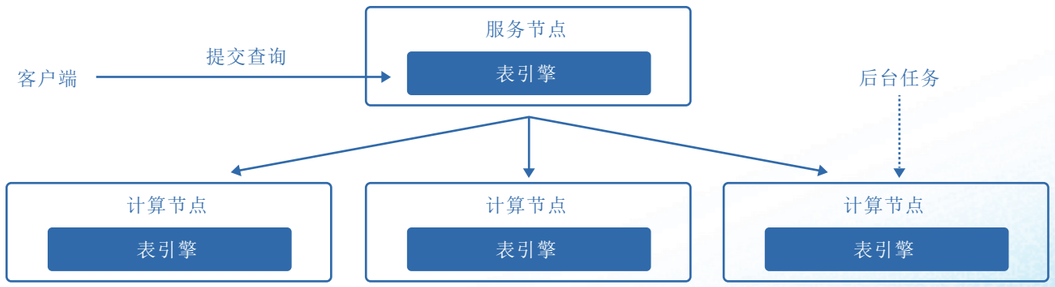
因为 Insert/Select 语句会在任意的 Coordinator 节点上执行，为保Read-Commited 语义，需要确保不同 Coordinator 进程间一致的 元数据读取，采用：

1.Leader Selection 机制保证唯一的 Master

2.Master 维护全局一致的拓扑图

3.所有 Coordinator 采用相同的选主机制保证每一张表有唯一的主节点

4.2 自研表引擎



表引擎抽象架构图

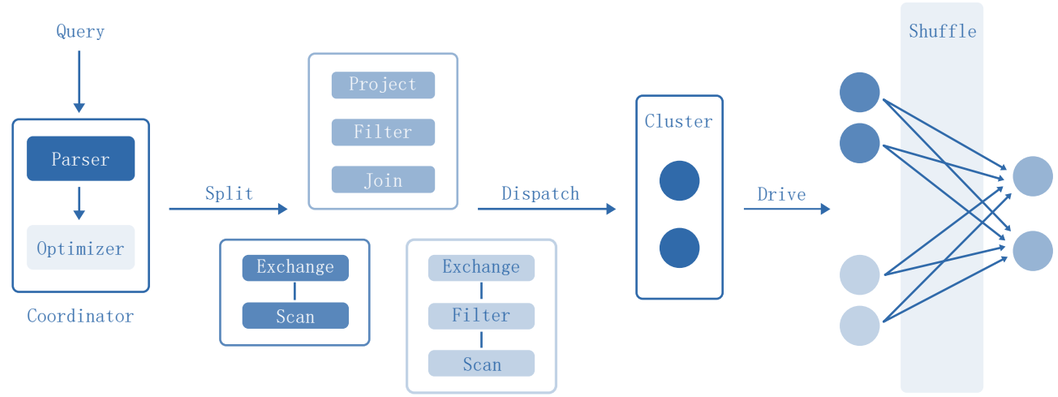
在ByteHouse中，复用了开源ClickHouse存储计算引擎，但开源ClickHouse诸多的表引擎不利于维护和应用。ByteHouse自研了一个CnchMergeTree表引擎，用户只需要感知CnchMergeTree引擎，原来ClickHouse分布式表和local表不再直接暴露给用户，降低了应用成本。

服务节点从元数据服务获取Part信息之后，需要分配到多个节点去执行。针对不同的需求和负载，支持不同的part metadata分配策略，可以根据需求和workload选择最适合的策略。

为了最小化节点变化带来的重新分配的开销，其中一个分配策略是通过一致性Hash算法对Part 元数据的进行分配。一致性hash算法可能会带来元数据的分配不均，针对这个问题，我们提出了一个新的Bounded Consistent Hash算法进行优化，在满足一定均匀性的前提下，尽可能减少拓扑变化带来的元数据的Rebalance。

4.3 复杂查询执行模型

分析型查询可以分为简单查询和复杂查询，简单查询通常是单表检索聚合、大表与小表Join查询，查询响应快；复杂查询指的是有大表、多表关联和复杂的逻辑处理，通常查询响应慢消耗资源多。ByteHouse在复杂查询上进行了优化设计。



复杂查询执行模型图

简单的查询可以采用两阶段执行模型，计算节点上面执行的partial阶段，服务节点上面执行的final阶段，一旦我们需要执行一个复杂的多个聚合或者join和查询，两阶段的执行模型灵活性非常低，也让查询的优化变得棘手。为了更好的支持分布式查询，方便执行优化器产生的执行计划，我们引入了支持多轮分布式执行模式的复杂查询。

复杂查询的执行流程如下：

- Query SQL String 通过parser解析为AST；
- AST做改写和优化，产生执行计划。
- 启用优化器的时候，通过优化器产生执行计划。

将执行计划切分为多个PlanSegment：

- PlanSegment即分布式执行过程中的一个执行片段，它包含：
  - 执行需要的AST片段，或者一个由PlanNode构成的逻辑执行计划片段；
  - PlanSegment执行的节点信息；
  - PlanSegment的上下游信息，这些信息包括上游的输入流，下游需要的输入流。
- 引擎的调度器会将这些PlanSegment构成一个DAG，然后下发给集群中的所有节点；
- 每个节点收到PlanSegment之后，开始驱动PlanSegment执行：
- 包含数据源的PlanSegment开始读取数据，将数据按照一定的shuffle 规则分发到下游的各个节点上；
- 包含exchange输入的PlanSegment等待上游的数据，如果需要继续做shuffle则会继续将数据下发给各个节点；
- 多轮stage完成之后，结果会返回到服务端。

4.4 列式储存设计

通常事务型数据库采用行存便于支持事务和高并发读写，分析型数据库采用列存减少IO和便于压缩。ByteHouse采用行列混存的方式，综合列存和行存优势，既能保证读写性能、支持事务一致性，又适用大规模的数据计算。

4.4.1 Data layout

表数据物理上按Partition Key切分为多个Parts存储在统一的云存储的逻辑存储路径下，每个Part的大小有数据量。和行数限制，计算组根据服务节点分配的策略（预先分配和实时分配）获得其对应的部分parts。

4.4.2 Part Delta

Part数据最初构建之后是一个行列混合存储的Part 数据文件，随着DML/数据字典/Bitmap index等构建工作的进行Part存在增量数据，这部分数据可以有以下两种存储方式：

1.每次构建都会Rewrite Part数据

2.生成增量数据，后台异步合并成一个大的Part文件

4.4.4 Compaction

ByteHouse支持将一个part文件拆分为多个小文件，通过配置Part的最大Size和最大行数，Compact之后的Part需要满足这个限制。

ByteHouse中的Compaction是在全局做的，与前面提高的全局的block ID保持一致。

4.5 事务和并发控制

4.5.1 事务概览

在ByteHouse里，为了保证数据质量，我们提供了事务语义的支持。每条SQL 语句都会转换为一个事务去执行，事务提供了原子性、一致性、隔离性和持久性（ACID）属性的保证，旨在在并发读写，软件异常，硬件异常等各种情况下仍然可以保证数据的正确性和完整性。

原子性（Atomicity）保证每一个事物被视为一个单元，事物要么完全成功要么彻底失败。在事务成功之前，写入的数据不可见，不会出现部分数据可见的情况。事物失败之后，会把写入的部分数据自动清理掉，不会导致垃圾数据的残留。ByteHouse在各种情况下等会保证原子性，包括掉电，错误和宕机等各种异常情况。

一致性（consistency）保证数据库只会从一个有效的状态变成另外一个有效的状态，任何数据的写入必须遵循已经定义好的规则。

隔离性（isolation）确保数据库SQL并发执行（例如，同一时刻读写同一张表）的正确性，确保数据库的状态在并发场景下能等价于某种顺序执行的状态，事务之间互不影响。隔离性是并发控制的目标，可以有多种隔离级别的实现，ByteHouse为用户提供的是read committed（rc）隔离级别的支持。未完成的事务的写入对于其他事务是不可见的。

持久性（Durability）保证数据的高可用性。一旦事务成功提交，其写入的数据会被持久化，及时在出现各种系统failure的情况下不丢失。ByteHouse采取的存储计算分离结构，利用了成熟的高可用分布式文件系统或者对象存储（例如hdfs，S3），保证成功事务所提交数据的高可用。

4.5.2 技术选型

ByteHouse是一款分析型数据库（OLAP），跟事务型数据库（OLTP）在事务上的需求是不同。分析型在事务上针对高吞吐低延迟的场景，相反，事务性数据库针对的是高QPS实时的场景。除了基本的ACID属性需要保证，ByteHouse在事务实现选型上主要有3个特别的需求。首先，ByteHouse单个事务可能涉及到海量数据（例如，上亿行级别），事务对数据吞吐和写入性能有较高要求，并且需要保证其原子性。其次，分析型数据库的workload中读的比例高于写，事务需要保证读workload不会被写workload影响和阻塞。最后，事务需要具备灵活可控的并发控制的功能，ByteHouse里除了需要处理用户侧并发的workload，还需要处理并发的后台任务。

ByteHouse事务处理主要是对用户数据的元数据进行管理，元数据包括用户的db，table和part（part是数据文件的元数据，包括了part名字，columns，行数，状态，版本，提交时间等信息）。随着数据的增长，元数据本身数量级也会线性增长，不能丢失并且需要高可用，所以需要一个分布式存储/数据库的方案。我们选择了成熟的分布式key-value数据库的作为ByteHouse的元数据的存储方案，通过抽象元数据读写API，后端适配了字节自研的ByteKV和苹果公司开发的FoundationDB。

4.5.3 分布式时钟

事务在分布式系统中的执行需要在分布式不同节点中进行时钟同步。ByteHouse采取了简单实用的Timestamp Oracle（TSO）方案。其优点首先简单易懂，采取中心授时，能够确定唯一时间。然后是性能好，通常一个tso节点能支持1m+的QPS。缺点是不适合跨数据中心的场景，所有事务从tso获取时间延迟较高。由于TSO是中心化授时方案，ByteHouse为其提供了高可用服务。

TSO使用混合逻辑时钟，时钟由物理部分和逻辑部分组成，64位表示一个时间。为了避免TSO宕机导致的时间戳丢失，需要对时间戳持久化。但是如果每次授时都持久化将会降低性能，所以TSO会预申请一个可分配的时间窗口（例如3s）申请成功之后，TSO可以在内存中直接分配3秒窗口之内的所有时间戳。客户端请求时间戳，逻辑时钟部分随着请求递增。如果出现逻辑部分溢出情况，会睡眠50ms等待物理时钟被推进。TSO会每50ms检查时钟，如果当前TSO的物理时钟已经落后于当前时间，需要更新TSO的物理时钟部分为当前物理时间。如果逻辑时钟部分过半，也会增加TSO的物理时钟，一旦物理时钟增长，逻辑时钟清零。如果当前时间窗口已经用完，需要申请下一个时间窗口。同时更新持久化的窗口上界。

4.5.4 事务处理

Atomicity（原子性）

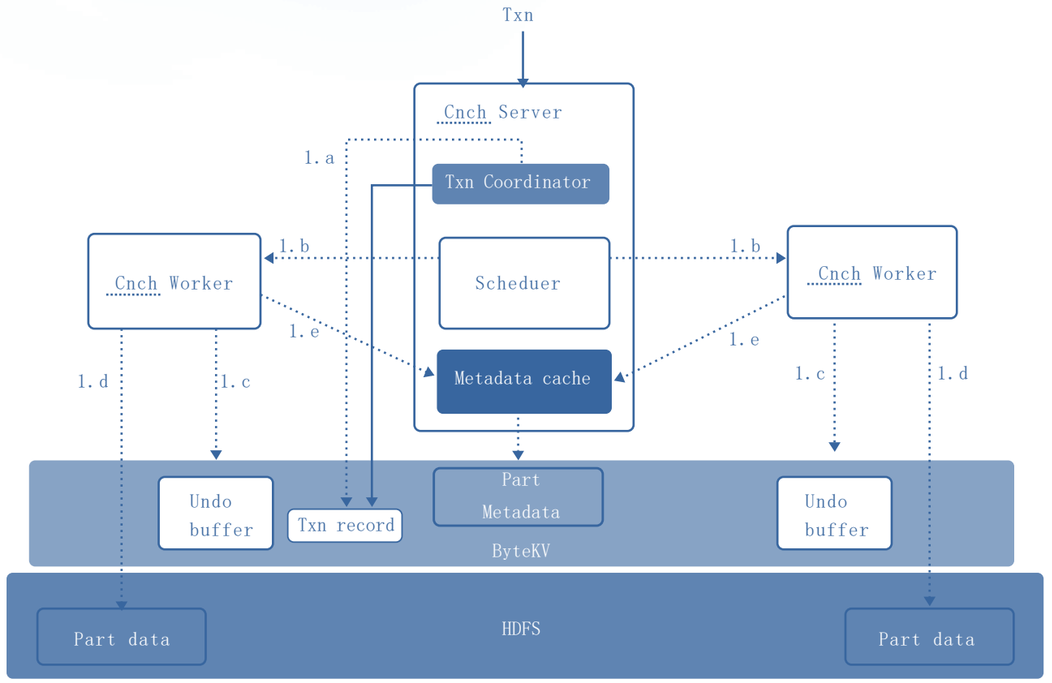
ByteHouse单个事务在元数据管理上有高吞吐读写的需求，由于分布式key-value数据库（例如ByteKV，FoundationDB）对单次原子写入的value都有大小限制（例如10MB），ByteHouse自己在分布式key-value存储之后实现了2阶段，使得单次写入大小不受限并且更加灵活可控。在第一阶段可以分批多次写入任意数据，并且不可见。第二阶段对事务进行提交，提交成功之后所有写入的数据同时可见。下面以一个insert sql为例，描述了2阶段原子提交的一个详细流程。

阶段1

- 1. a：在kv里写入事务记录（txn record），唯一标识当前事务；
- 1. b：解析insert sql并执行；
- 1. c：在远端文件系统或者对象存储写入数据之前，先把要写入数据的位置信息写入undo buffer（供失败情况下清理使用）；
- 1. d：把数据写入到远端文件系统或者对象存储；
- 1. e：提交数据的元信息part，写入到kv中；

阶段2

- 提交事务，并更新事务记录的提交时间；
- 异步更新part数据的提交时间为事务的提交时间（part未更新提交时间之前，需要反查事务记录的提交时间；



事务提交详细流程图

Consistency（一致性）

ByteHouse选择的分布式key-value存储系统，ByteKV和Foundation已经提供了一致性的支持，直接复用即可。

Isolation（隔离性）

ByteHouse对用户提供Read Committed（RC）隔离级别的支持。每个事务初始化的时候会从TSO服务获取一个timestamp作为其id和开始时间，提交的时候会再从TSO服务获取一个提交时间，在事务提交的时候更新kv里事务记录的提交时间并异步更新part的提交时间。读事务可以读取到已经提交成功（对应事务提交即成功）并且提交时间小于读事务开始时间的part元数据信息，从而实现RC语意。相比更加严格的隔离级别，RC隔离级别可以最大化读性能。而更严格的隔离级别例如Serializable Snapshot Isolation（SSI），读可能会被写入block。

Durability（持久性）

ByteHouse元数据持久到ByteKV或者FoundationDB中，2个分布式key-value存储提供了持久化和高可用的保障。

4.5.5 并发控制

ByteHouse利用多版本和锁来保证并发读写场景下数据的正确性。ByteHouse除了来自用户的workload，内部还有后台任务（merge/alter 任务和唯一键表的去重任务）的并发读写需要处理。ByteHouse选择了RC隔离级别，对于新的写入（例如insert），由于不可见，可以无锁执行。对于已有数据，在并发读写时，需要进行并发控制。对于并发读和写这种场景，ByteHouse利用多版本解决了读和写冲突，提供了读写性能。对于并发写写的场景（例如merge和唯一键表的去重任务），利用了加锁来保证数据的正确性。

多版本

每个part的元数据除去其原有基本信息之外，都有一个对应的版本（version），每次对已有数据进行变更，都会产生一个新的版本，而不是直接在原有数据上进行更新。对于RC隔离级别，已经开始的读事务，仍然继续读取旧的版本，新版本对其不可见，这样读和写互相不影响，最大化读写性能。

Data part

Key: Partname_version	value: {columns,rows_count,min_max...}	{txn_id}	{deleted}	{commit_time}

锁

- 分布式KV锁

  ByteHouse对于DDL提供了全局KV排他锁避免并发的对table schema进行变更，分布式kv锁是全局共享，不同的节点都可以共享。

- 内存读写锁
  - 支持共享锁和排他锁
  - 支持等待
  - 支持不同粒度

  ByteHouse提供了多级细粒度DML读写锁的支持，DML相关的任务可以根据需求在不同粒度持不同类型的锁。

```
1                Table
2              /       \
3          bucket       \
4          /              \
5    partition          partition
```

4.5.6 垃圾回收

ByteHouse对于不可见的part和版本会定期进行回收，例如merge任务生成新的part之后，对于旧的part，当不再被查询引用之后，就会进行回收，释放空间，降低成本。

4.6 自研优化器

优化器是数据库系统的核心，优秀的优化器能极大提高查询性能，特别是在复杂查询场景下优化器能带来数倍至数百倍的性能提升。ByteHouse 自研优化器基于四个大的优化方向提供极致优化能力：

RBO：基于规则的优化能力。支持：列裁剪、分区裁剪、表达式简化、子查询解关联、谓词下推、冗余算子消除、Outer-JOIN 转 INNER-JOIN、算子下推存储、分布式算子拆分等常见的启发式优化能力。

CBO：基于代价的优化能力。支持：Join Reorder、Outer-Join Reorder、Join/Agg Reorder、CTE、物化视图、Dynamic Filter 下推、Magic Set 等基于代价的优化能力。并且面向分布式计划融合了 Property Enforcement。

DBO：基于数据依赖的优化能力。支持：唯一键、functional dependency、Order dependency、Inclusion dependency 等基于数据依赖关系的优化能力。

HBO：基于查询反馈的优化能力。支持：基数估计动态调整、并行度动态调整、执行计划动态调整等基于历史执行反馈的优化能力。

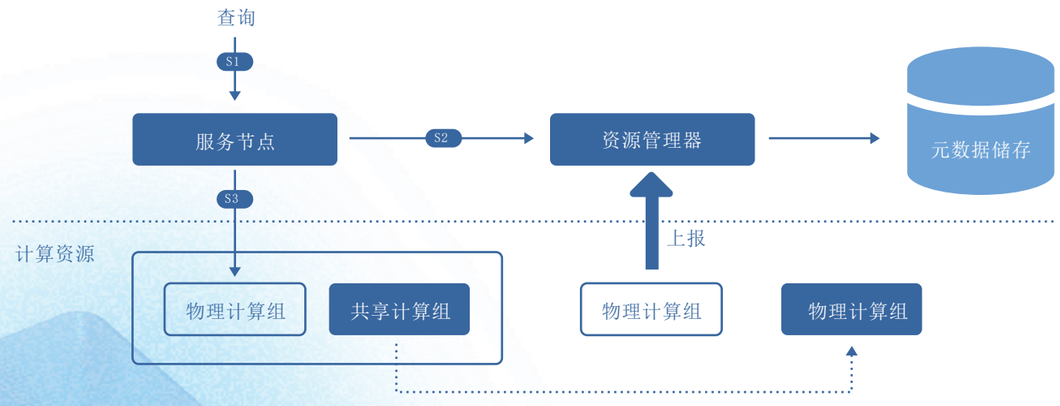
4.7 资源管理器

资源管理器（Resource Manager，简称RM）组件用于对计算资源进行统一调度和管理，是实现资源弹性和提高资源利用率的核心组件。

ByteHouse资源管理器的核心设计目标如下：

- 能够管理、调度计算资源，提升资源利用率。
- 能够收集 VW 的监控数据，便于基于负载进行资源调度。
- 能够为查询、INSERT、各种后台任务提供调度功能。
- 能够协调不同VW并动态调配 VW 的资源。
- 实现资源池化，弹性扩缩功能。

资源管理器的简化架构图如下：



资源管理器的简化架构图

4.7.1 资源收集和服务发现垃圾回收

1. 计算节点启动后，会运行一个后台线程定期上报心跳（每秒一次），心跳中包含CPU/MEM/Query Count等各种信息；
2. RM 也会定期清理不活跃的 计算节点。

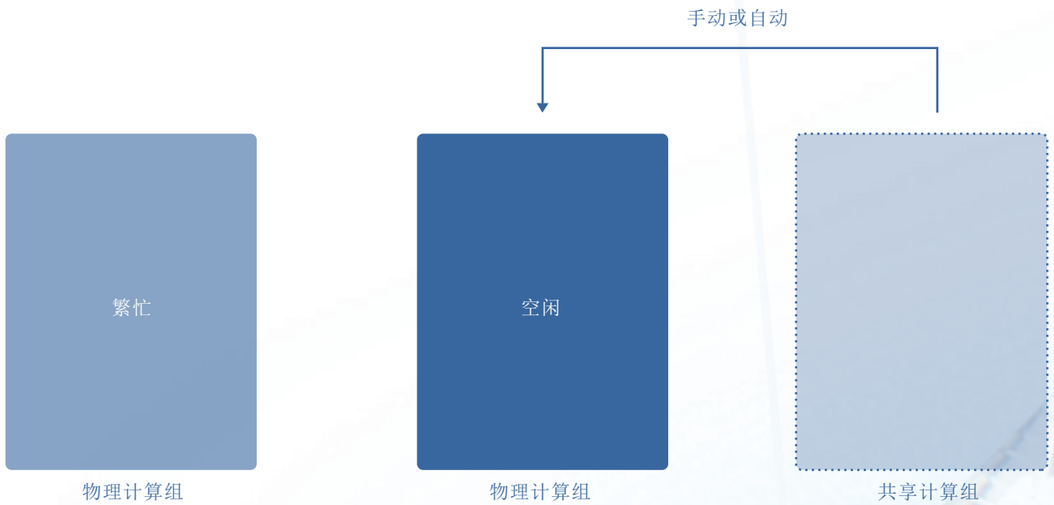
4.7.2 资源管理器自身高可用 & 兼容升级

1. 采用简单的俩副本主从配置；
2. 服务节点的请求和计算节点的心跳只会发到主节点；
3. 当主节点宕机时，由备节点接替服务，并从 KV数据库同步必要的元数据；
4. 由于服务节点有计算组和计算节点的缓存，即便 RM 短时间内宕机，服务节点仍然能够使用缓存的数据继续运行。

4.7.3 计算资源弹性共享

在一个数据仓库系统中，计算资源永远是最珍贵的资源。ByteHouse计划通过计算资源弹性共享来计提升算资源利用率，核心实现如下：

1. 引入 物理/ 共享计算组的概念，物理计算组即 k8s StatefulSet 对应的计算组，而共享计算组则是某一个物理计算组的引用；
2. 用户可以手动创建共享计算组，当请求发送至共享计算组时，实际上使用的是其对应的物理计算组；
3. 当某个物理计算组整体使用率较低时，允许基于该物理计算组创建临时的共享计算组，从而其计算资源可以被共享利用。



计算资源弹性共享示意图

第五章
总结和展望

第五章 总结和展望

云原生数据仓库ByteHouse的最大特性在于存储和计算的分离，允许以经济高效的成本独立扩展；用户只需为其使用的容量和性能付费。ByteHouse可以更有效地利用资源，既保持高性能的特性，又带来更低的运维成本。

实时数据分析是 ByteHouse 的优势场景，结合字节跳动实时数据场景的特点，我们积累了非常多的经验，并将这些能力沉淀到了ByteHouse 上。ByteHouse 基于自研技术优势和超大规模的使用经验，为企业大数据团队带来新的选择和支持，以应对复杂多变的业务需求，高速增长的数据场景。未来，ByteHouse将不断以字节和外部最佳实践输出给行业用户，帮助企业更好地构建交互式大数据分析平台和云原生数据仓库。

火山引擎