一种基于管算存分离的内存数据库实现技术

张延松[1),2),3)] 韩瑞琛[1),2)] 刘 专[4)] 张 宇[5)]

[1)](中国人民大学数据工程与知识工程教育部重点实验室，北京 100872)

[2)](中国人民大学信息学院，北京 100872)

[3)](中国人民大学中国调查与数据中心，北京 100872)

[4)](英特尔(中国)研究中心有限公司，北京 100190)

[5)](国家卫星气象中心，北京 100081)

摘 要 在多核处理器、大内存、非易失内存等新硬件技术的支持下，异构存储与计算平台成为主流的高性能计算平台.传统的数据库引擎采用一体化设计，新兴数据库则采用存算分离和算子下推技术以更好适应新型分布式存储架构.提出了一种新颖的基于管算存分离方法的内存数据库实现技术，在存算分离技术的基础上进一步根据数据库模式、数据分布与负载计算特征将数据集划分为元数据集和数值集，将统一的查询引擎分解为元数据管理引擎、计算引擎和存储引擎，将包含语义信息的元数据管理抽象为独立的管理层，将无语义的数值存储和计算抽象为计算存储层，其中计算密集型负载定义为计算层，数据密集型负载设计为存储层，并根据硬件平台的不同分离或合并计算与存储层.内存数据库的实现技术分为几个层次:1)模式优化，实现数据库存储中“数(数值)”与“据(元数据)”的分离，根据数据的内在特性选择不同的存储与计算策略;2)模型优化，采用Fusion OLAP模型，实现在关系存储模型上的高性能多维计算;3)算法优化，通过代理键索引、向量索引支持优化的向量连接、向量聚集算法，提高OLAP性能;4)系统设计优化，通过数据库引擎分层技术实现管理与计算分离、存储与计算分离以及多维计算算子下推到存储层.实验结果表明，管算存分离计算模型可以灵活地支持CPU-GPU异构计算平台、DRAM-PM(Persistent Memory，持久内存)异构存储平台和外部存储平台，采用开源的Arrow内存列存储引擎作为数据库“数”的存储引擎，以及应用多维计算算子下推到Arrow存储引擎技术的OLAP实现技术在SSB基准测试中与存算结合的内存OLAP实现技术性能相当，查询性能优于主流内存数据库Hyper和OmniSciDB，以及基于Arrow存储的GPU数据库PG-Strom.

关键词 内存数据库;数据分离;存算分离;管算分离;向量索引

中图法分类号 TP311 **DOI号** 10.11897/SP.J.1016.2023.00761

An In-memory Database Implementation Technique based on Separation of Management, Computation and Storage

ZHANG Yan-Song[1),2),3)] HAN Rui-Chen[1),2)] LIU Zhuan[4)] ZHANG Yu[5)]

[1)](*Key Laboratory of Data Engineering and Knowledge Engineering* (*Renmin University*), *Ministry of Education*, *Beijing* 100872)

[2)](*School of Information*, *Renmin University of China*, *Beijing* 100872)

[3)](*National Survey Research Center at Renmin University of China*, *Beijing* 100872)

[4)](*Intel China Research Center Ltd*, *Beijing* 100190)

[5)](*National Satellite Meteorological Centre*, *Beijing* 100081, *China*)

Abstract Heterogeneous storage/computing platform has been main-stream high performance computing platform with the support of multicore processors, big memory and non-volatile

收稿日期:2022-04-27;在线发布日期:2022-11-03. 本课题得到国家自然科学基金项目(61732014,61772533)、北京市自然科学基金项目(4192066)资助. **张延松**，博士，副教授，主要研究领域为数据仓库、内存数据库、GPU数据库和新硬件数据库技术. E-mail:zhangys_ruc@hotmail.com. **韩瑞琛**，博士研究生，主要研究领域为内存数据库和新硬件数据库技术. **刘 专**，硕士，主要研究领域为GPU数据库和新硬件数据库技术. **张 宇**(通信作者)，博士，高级工程师，主要研究领域为数据仓库、GPU数据库. E-mail:yuzhang@cma.gov.cn.

memory techniques. The traditional database engines are co-designed for storage and compute, the emerging databases employ separation of storage and compute and pushdown compute techniques for novel distributed storage infrastructure. This paper introduces a novel in-memory database implementation based on separation of manage, compute and storage technique, based on separation of storage and compute, it further separates the dataset into meta dataset and value dataset according to the characteristics of database schema, data distribution and workload. The unified query engine is divided into meta data management engine, computing engine and storage engine. The meta data with semantic information management is abstracted as independent management layer, the non-semantic value storage and compute are abstracted as compute and storage layers, and the compute-intensive workload is further defined as compute layer, the data-intensive workload is defined as storage layer, the compute layer and storage layer can be combined or separated according to different hardware configurations. The implementation of in-memory database is designed as following levels: 1) schema optimization, separating value and meta data in database to choose different storage and compute strategies according to the inner data features; 2) data model optimization, the Fusion OLAP model supports the high performance multidimensional compute on relational storage model; 3) algorithm optimizations, using surrogate key index and vector index to support the optimal vector join and vector aggregation for higher OLAP performance; 4) system design optimizations, the layered database engine separates the mange and compute, storage and compute, and pushdown multidimensional compute to storage layer. The experimental results show that the separation of manage, compute and storage model can flexibly support hybrid CPU-GPU computing platform, hybrid DRAM-PM (Persistent Memory) storage platform and external storage platform, by employing the open-source in-memory column store Arrow as data storage engine for database and pushing down multidimensional compute to Arrow storage engine, the OLAP implementation proves to be equal performance as OLAP implementation co-designed for storage and compute in Star Schema Benchmark, the OLAP performance outperforms the leading in-memory databases Hyper, OmniSciDB and Arrow based GPU database PG-Strom.

Keywords in-memory database; separation of data and meta data; separation of storage and compute; separation of manage, compute and storage; vector index

1 引 言

数据库系统经过几十年的发展已经逐渐成熟和完善,大部分研究工作主要集中于索引、缓冲区、日志以及事务并发控制、查询优化等传统的核心领域,并结合新硬件特性发展出云数据库、内存数据库、GPU数据库等新兴的数据库系统.

传统的数据库是一个相对封闭的系统,存储引擎与查询处理引擎设计为紧耦合的功能以提高数据访问的局部性和查询处理性能,这种设计思想的基本假设是存储与计算、软件与硬件同步发展,紧密的功能模块封装带来更高的性能收益.近年来,随着多核处理器、GPU、大内存、非易失性内存、分布式存储等新型计算和存储硬件技术的发展与普及,存储与计算、软件与硬件发展的平衡被打破,传统数据库黑盒般的一体化设计逐渐转向存算分离架构,如原生的云数据库通常采用计算与存储分离的设计方法,通过独立的云存储取代传统数据库紧耦合结构中的存储引擎,通过将核心的存储与计算功能分层实现各功能层独立扩展,按需计算,其中最具代表性的存算分离数据库为Amazon Aurora[1-2]、阿里巴巴的PolarDB①以及华为的Taurus[3].随着存储技术的发展,集中式数据库同样产生存算分离的需求,大内存推动了分析型内存数据库及开放的内存存储技术

① 云原生关系型数据库PolarDB, https://www.alibabacloud.com/zh/product/polardb, 2022,3,15

的发展，以开源的Apache Arrow[①]为代表的内存列存储提供了跨平台的数据交换格式，可以为不同系统提供极低成本的内存数据共享资源. 对于数据库系统而言，Arrow不具备完整的存储引擎功能，但Arrow提供了不同数据库系统之间统一的共享数据存储访问接口，可以为一些计算型任务提供统一的数据存储访问功能，实现了相对于数据库系统的存储独立性，也为传统的磁盘数据库提供了灵活的内存列存储支持，减少数据库存储引擎的开发代价. 对于数据库来说，Arrow可以退化为计算数据集，虽然相对于数据库的存储引擎功能单一，但可以提供数据库系统之外的数据存储访问能力，将数据库的复杂计算从数据存储中剥离，实现存算分离.

当前的存算分离技术展示了具有独立技术路线的存储与数据库技术的结合方法，也将传统数据库紧耦合的引擎设计进行分层，使存储与计算技术的发展走向可扩展化、分工化、专业化. 当前的存算分离技术主要聚焦于系统层面的功能分离，在结合数据库内在的模式特征、数据分布特征、负载特征时可以进一步细化哪些数据适合存算一体的传统数据库模式，哪些数据适合存算分离的计算架构，哪些数据适合算子下推，实现以数据为中心的存算分离设计.

与通用数据库不同，OLAP(On-line Analytical Processing，联机分析处理)数据库以多维模型组织多维数据集，多维计算具有显著的数据局部性特征，多维元数据管理与多维计算不同的数据和计算特征可以进一步优化存算分离技术. 本文以内存数据库为应用背景，讨论了基于OLAP多维模型和多维分析负载特征的管算存分离技术，即将数据库的多维数据集按负载特征划分为管理负载、计算与存储访问负载，将传统的一体化OLAP查询处理过程分离为功能独立的元数据管理层、计算数据层和存储访问层，不同层次可以独立扩展并结合负载特征匹配最优的实现技术. 本文提出了一种新颖的内存数据库管算存分离计算技术，通过模式、存储、计算三个维度的优化技术对数据库的数据和计算进行分层，根据数据特征优化设计各层的存储与计算，本文的主要贡献体现在以下几个方面：

(1) 模式优化：基于数据库多维数据模型将数据库模式转换为优化的星形模型，维、层次及描述性分组属性全部上推到维表，用作多维数据集的元数据，事实表中只存储无语义的数值型度量数据，将描述型数据与计算型数据通过模式优化分离，实现“数(值)”与“据(元数据)”的分离，从而将不同类型的管理和计算负载分离到不同的数据集中，达到数据管理与数据存储的分离(管存分离). 在存储层可以进一步根据硬件的特征和模式的特征将计算密集型负载和数据密集型负载分离，用特定的硬件加速计算负载，实现管算存分离.

(2) 存储优化：模式上的管存分离支持不同数据集上采用不同的存储管理方法，较小维表上的元数据管理需要支持通用的增、删、改操作，可以使用传统的数据库引擎；较大事实表上的计算可以匹配优化的存算分离技术，定制计算接口，以增强存储的开放性与独立性、提高存储效率、降低存储成本、提高存内计算性能为目标. 通过存储与模式的优化实现对OLTP与OLAP负载不同的数据管理和计算功能，提供了OLTP模式与OLAP模式的映射能力，支持了HTAP(Hybrid Transaction Analytical Processing)计算架构.

(3) 计算优化：基于多维数据模型设计了存储端的多维计算操作符(存内计算)，实现在单一事实表上的连接、分组、聚集操作，将高收敛性的多维计算算子下推到事实表存储层，实现最大化的近数计算(NDP，Near-Data Processing).

本文第2部分对相关技术进行对比分析；第3部分描述基于管算存分离的内存数据库实现技术；第4部分通过实验给出基于管算存分离计算模型的内存OLAP基准测试性能；最后对本文进行总结.

2 相关工作

传统数据库的设计中存储引擎与查询处理引擎的设计紧密结合，在底层存储(表存储、索引存储、日志存储等)、缓冲区管理、索引、查询优化等核心功能模块中存储访问与计算采用一体化设计，以存储访问为基础的I/O代价模型也是关系数据库查询优化重要的基础技术. 随着云计算技术的发展，云计算平台提供了更高性价比的数据存储、数据安全和资源弹性. 云计算平台数据库系统将传统数据库的存储层交付给云平台提供的可靠性存储解决，实现计算与存储分离的弹性扩容与缩容. 这种技术演变反映了外部存储技术发展超越数据库内部存储技术的

① Apache Arrow，A cross-language development platform for in-memory analytics，https://arrow.apache.org/，2022,5,6

现状，推动数据库将封闭的存储完全委托给底层可靠的云存储系统，数据库更聚焦于计算功能，实现存储与计算分离、存储与计算技术单独扩展. 存算分离技术依赖于成熟的云存储技术以及高性能网络数据传输带宽高于本地磁盘I/O带宽的硬件技术，但存算分离架构下数据库存储层以数据页面向上层计算节点返回数据，较大的存储访问延迟使存储层计算资源通常处于闲置状态而浪费计算资源. 云数据库采用将部分计算下推到存储层的方法达到近数计算的效果[4-5]，减少网络传输代价并提高存储层的计算资源利用率. 简而言之，存算分离是基于现代云存储技术将数据库的存储与计算功能分层，将数据库的计算功能与存储层的存储访问与存储端计算功能剥离，可以由不同的技术团队独立开发与功能扩展，构建一个开放的数据库平台.

从业务场景来看，现代内存数据库具有HTAP特征，即通过内存计算的高性能实现内存事务处理与内存大数据分析处理的统一平台. 与传统磁盘数据库相比，内存数据库没有缓冲区管理，也较少依赖全局索引加速查询性能，更适于存算分离技术的实现. 现代数据库越来越多地使用开源的Apache Arrow作为统一的共享内存数据交换格式，Arrow是一种内存列存储格式，面向CPU和GPU硬件优化，支持不同平台间的零拷贝(zero-copy)数据共享访问，可以进一步扩展为面向不同数据库的独立内存存储平台，分离内存数据库的存储层功能. 独立的存储层上可以独立扩展存储端计算能力，如Oracle Exadata数据库一体机中在存储服务器上的SmartScan①支持连接过滤、投影、选择等操作下推到存储服务节点，PushdownDB在云端共享存储S3支持的SELECT、PROJECT以及不带GROUP-BY的聚合操作的基础上进一步将算子扩展到JOIN、GROUP-BY以及TOP-K查询，通过基于小表的布隆过滤器实现大表连接过滤；通过两阶段的GROUP-BY投影和不同值下推到存储节点实现聚合计算；通过随机抽样计算阈值在存储层过滤数据实现TOP-K查询. 通用数据库的存算分离是一种粗粒度的分层方法，结合数据库内在的特征，可以进一步细化存算分离的应用策略.

数据库将数据组织为多维模型，维表存储多维元数据，如维属性、层次属性、描述属性等，数据量较小但数据类型多样；事实表存储用于聚集计算的度量数据，数据量大以数值型数据为主. 从模式特征和负载特征来看，维表对应管理型负载，事实表对应计算型负载，小数据上的管理型负载适合数据库平台，而大数据上的计算型负载适于存算分离架构，可以从数据库中完全分离出去，在存储层完成80%以上的计算任务. Fusion OLAP[6]模型在关系存储模型上建立逻辑的多维模型，将OLAP查询处理分解为维表上的维映射计算，事实表上的多维索引计算和聚集计算三个阶段，优化匹配不同的计算平台. 其中维映射计算可以看作是多维元数据上的管理型负载计算任务，其余两个阶段是面向不同硬件定制的计算和存储层计算任务.[7]进一步将OLAP负载从计算的特性划分为维表数据管理、GPU多维索引计算和CPU内存聚集计算三个与硬件优化匹配的异构计算框架，探索了在新硬件异构平台上实现管理与计算分离，存储与计算分离和存内计算的方法.[7]主要的优化目标是标准星形模型的OLAP计算性能，初步探索了管算存分层计算模型，但对OLTP负载的集成、复杂模式(如TPC-H雪花形模式)的集成未做进一步探索，本文进一步扩展[7]的研究目标，将HTAP负载优化、新硬件查询优化技术纳入统一的管算存技术框架，为数据库提供统一的理论模型.

将庞大的事实表从数据库中分离为存储层计算型负载后，核心的优化问题是如何将OLAP操作中具有强收敛性的算子下推到存储层实现存内计算，减少数据库端的计算代价. 在OLAP算子中，选择、投影、聚集操作易于下推到存储端计算并具有较好的收敛性，连接与GROUP-BY操作较为复杂，当前是基于简单的布隆过滤器技术实现连接过滤，减少存储端返回的数据大小. 在OLAP操作中，GROUP-BY属性通常存储于维表，需要通过连接操作才能完成对事实数据的分组操作，优化存储端的连接分组操作是实现高效算子下推的基础.

当前学术界主流的连接算法为哈希连接，尤其以面向cache优化的PRO[8](基于Radix分区的哈希连接)算法为性能的代表. 但PRO算法需要物化连接中间结果，并不适合OLAP负载中典型的多表星形连接任务[9]. 哈希连接是一种无模式知识的连接算法，而数据库的多维数据模型支持维映射特征的内置连接索引技术[10]，通过将维表代理键索引映射为事实表连接索引简化连接算法.[11]给出了一

① Smart Scan (Cell Offloading) in Exadata | Magic Feature of Oracle Exadata, https://ittutorial. org/smart-scan-cell-offloading-in-exadata-magic-feature-of-oracle-exadata/, 2019,12,30

种数组化的OLAP实现技术，通过基于代理键索引的扩展位图索引实现向量索引，将维表的GROUP-BY属性压缩到维向量中，在事实表外键列上的向量连接操作中实现分组操作下推到事实数据存储端，将收敛性最强的分组聚集计算下推到存储层.

在内存数据库的优化技术上，存算分离是在计算维度上的优化设计，通过分离数据库端计算与存储端计算来适应新型存储技术的发展；管算存分离是在数据维度上的优化设计，基于数据库数据和负载的特征实现多维元数据管理负载上推到数据库端，OLAP多维计算下推到存储端，将多维索引计算进一步抽象为计算层，在传统数据库中将数据管理、计算与存储访问功能分层，将计算从数据库核心功能中分离，独立扩展存储与存内计算功能，也使数据库和存储及硬件团队可以更加聚焦于不同层次功能的实现与优化工作.

3　内存数据库管算存分离计算架构

存算分离计算架构是数据库技术与云存储技术融合而产生的，根据数据存储的物理层次划分存储和计算功能. 管算存分离计算架构是面向多维数据模型而优化设计的，数据按多维模型的语义逻辑划分：维代表多维数据的元数据，用于管理多维数据结构和查询映射；事实数据代表多维空间中存储的度量数据，用于分析计算；维度与事实数据之间的映射对应关系模型中事实表外键列，用于实现将维度映射到事实数据的多维计算. 管算存三个层次分别对应多维数据集的逻辑结构，根据不同类型数据不同的存储、计算需求特征难以在统一的存储和计算平台获得较高的效率而实现从数据模型、语义、功能上的分层，并可以与不同的硬件存储/计算层次、OLTP-OLAP业务层次等优化匹配，实现数据库核心功能面现不同体系结构的解构与优化设计. 简而言之，存算分离是面向存储迁移客观情况的被动分层技术，而管算存分离是面向多维数据内在特征的主动分层技术，通过分层而使数据管理、数据计算、数据存储访问获得最优的硬件资源，提高硬件利用率.

内存数据库管算存分离计算架构是一种面向大内存平台的分层内存数据库实现技术，它首先基于数据库的多维模型将数据划分为“数(数值)”与“据(元数据)”，分别对应无语义的事实数值和多维元数据，将数据上无语义的数值计算与基于语义的元数据管理分离，从而分离存储系统与数据库系统，提高存算分离的效率.

如图1所示，理想的OLAP数据库模型为星形模型，多个维表与一个事实表构成多维数据集，带有语义的描述信息全部存储于维表，负责查询的解析与数据映射，生成无语义信息的维向量，用作系统的元数据管理层；事实表全部由无语义的数值构成，可垂直划分为外键表和度量表，外键表通过星形连接进行多维分组计算，生成向量索引用于度量表上的分组聚集计算，用作系统的多维索引计算层；在事实表外键生成的向量索引的基础上在较大的度量数据上执行分组聚集计算，完成存内计算过程，作为系统的存储层. 管算存分离计算模型是指从模式语义逻辑上将OLAP数据集划分为多维元数据集、多维计算数据集和事实数据集，在存储模型上独立为三个数据集选择适合的存储设备、存储模型和表结构，在计算模型上通过向量索引将三个不同的计算阶段整合为粗粒度的流水线，支持在异构存储/计算设备、异构系统之间的分层计算，将传统紧耦合的查询处理引擎解耦合为异构的、松耦合的协同计算.

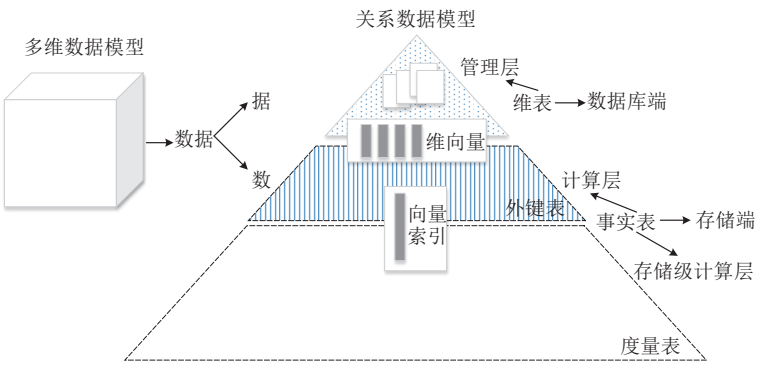


图1　基于管算存分离架构的内存数据库系统

维表元数据由数据库管理，简单结构数值类型的事实表由专用的存储层管理，外键连接计算可以通过专用的硬件加速器实现定制化计算，存内计算下推到存储端执行，最小化与数据库端的数据交互. 当存储为同构设备时，计算层和存储层可以合并为存储层，计算合并为存储内计算（存内计算），计算模型简化为管存分离计算模型；当存储层为异构存储时，可以设置独立的计算层和存储层，计算模型为管算存三级计算模型.

管算存（管存）分离与存算分离计算模型的区别主要体现在以下几个方面：

（1）存算分离计算模型的“存”指的是独立的存储层，执行的是近数计算，任务是通过算子下推优化技术减少向上一级计算层传输的数据量；

（2）存算分离的“算”指的是数据库查询处理引擎，执行在过滤后数据上的查询处理任务，是主要的计算节点；

（3）管算存（管存）分离的“管”指的是OLAP数据集的多维元数据管理，任务是存储语义数据、元数据维护、查询映射和无语义计算结果解析，最小化数据库查询处理负载；

（4）管算存（管存）分离的“存”指的是独立的存储层，执行的是面向OLAP定制的多维计算任务（“算”指的同构存储时的存储内计算和异构存储时的外键连接计算），实现无语义事实数据上的存储级计算，将OLAP核心的多维分析处理任务完全下推到事实数据存储层，将OLAP查询的计算负载完全从数据库查询处理引擎中分离出来.

3.1 模式优化

模式优化可以简化管算存分离计算模型并提高计算效率. 模式优化的目标是分离“数”与“据”，实现数据库端与存储端的管存分离，简化算子实现技术，提高存储效率和存储端算子下推效率. 下面以TPC-H基准数据模式为例讨论管存分离架构对于非标准数据仓库模型和包含OLTP和OLAP混合负载应用需求下的模式优化方法.

3.1.1 主外键优化

将维表主键通过代理键索引（连续的1，2，3…整数序列）映射为维度的连续偏移地址，在内存列存储中维表主键对应维表记录偏移地址，外键通过直接地址映射访问维表记录，替代传统的哈希连接方法. 维表主键代理键索引机制扩展了传统数据库理论中主键的实体完整性约束，在非空值和唯一值约束的基础上增加了面向多维数据模型的值-地址映射约束，同样地，在主外键参照完整性约束的基础上进行扩展，将外键值在参考表主键上的唯一等值约束扩展为值-地址映射完整性约束，即外键值在维表主键列的地址映射关系. 通过对数据库完整性约束的扩展，数据库从关系模型扩展为关系-多维模型，在关系存储模型的基础上增加了多维计算模型，丰富了OLAP计算模型. 代理键索引的自增性和无语义特点不影响插入（insert，新值采用自增的唯一键值）与更新操作（update，不修改无语义代理键值），删除操作（delete，事实记录不删除时不能删除相应的维记录）在数据库中受维表与事实表的参照完整性约束而较少执行，维表删除操作可以周期性地通过代理键复用机制而低代价实现[9].

3.1.2 维优化

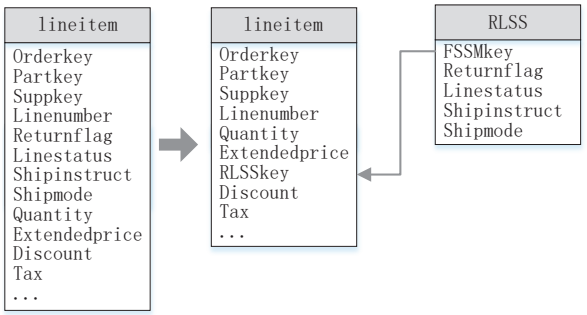
除已有维表Part，Supplier和Customer外，根据分析任务的需求增加额外的日期维表Date，以及由事实表中低势集退化维度（只有单一属性的维）的笛卡尔积构成的退化维表RLSS（如图2所示TPC-H的Lineitem表中由退化维度属性L_RETURNFLAG：3，L_LINESTATUS：2，L_SHIPINSTRUCT：4，L_SHIPMODE：7笛卡尔乘积168条记录组成的维表，主键为代理键FSSMkey）. 维优化的目标是从事实表中分离出具有描述语义的数据并上推到维表层，将相关的操作上推到元数据处理层，使事实表中只存储代表外键的整型和代表度量的数值型数据，在事实表存储上实现“数”与“据”的分离，简化事实表存储和存储端的计算，消除存储端语义解析.

3.1.3 事实表优化

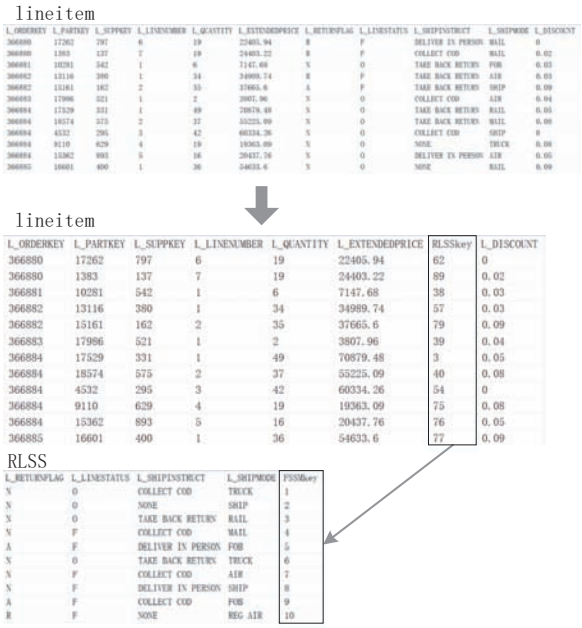
根据分析需求，可以将TPC-H的三个事实表Partsupp，Lineitem和Orders合并为SSB基准中统一的Lineorder事实表，也可以将Partsupp中用于分析的PS_SUPPLYCOST属性物化到Lineitem表中，消除Partsupp与Lineitem表的连接操作，同时将Orders表的主键更改为代理键，并更新Lineitem表中的外键，使Lineitem表记录在内存列存储中可以直接映射访问Orders表相应记录，降低Lineitem表与Orders表的连接代价.

3.1.4 管算存分离

TPC-H是典型的三范式结构，适合于业务处理场景，原始的模式适用于数据管理层，即使用数据库端处理维表的增、删、改和事实表数据写缓存上的更新操作. 数据库管理端负责SQL解析、转换，并作为事实记录行结构写缓存delta表. 存储端可以使用专用的存储层技术，如Arrow存储insert-only模式



（A）退化维度逻辑结构



（B）退化维度物理结构

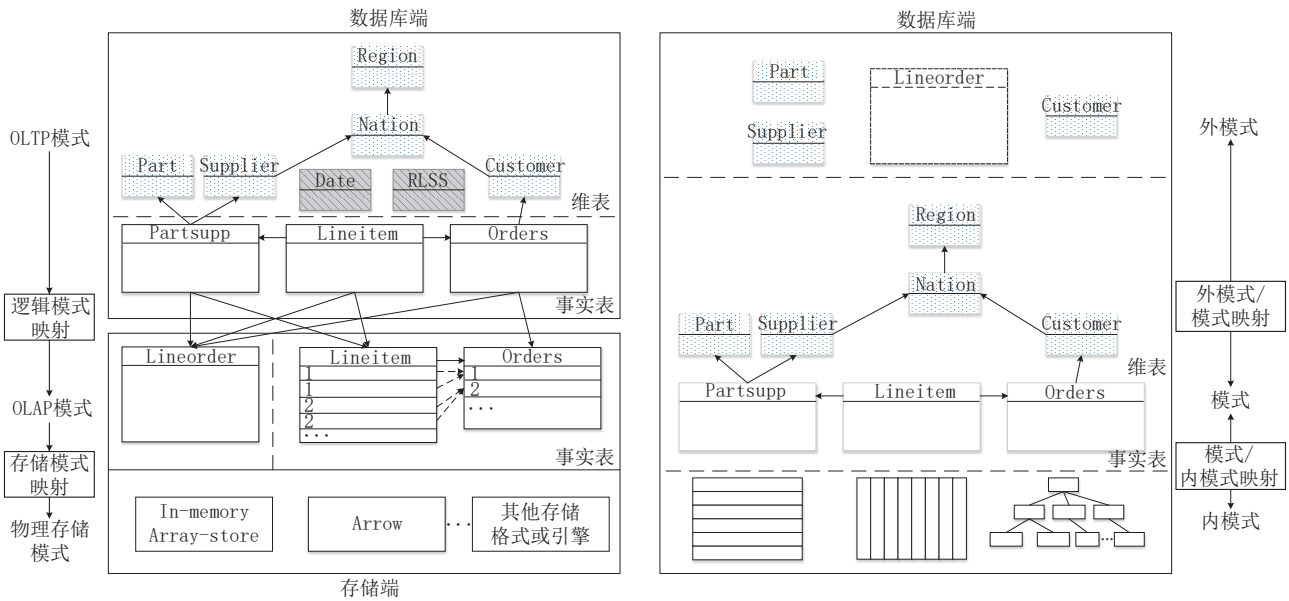
图2　退化维度优化

的只读数据，实现在存储端的算子下推，最大化存储端计算效率．为提高计算性能，可以进一步将事实表外键列及其上的多表连接计算抽象为专用的计算层，通过内存计算或硬件加速器进一步提高计算性能．

如图3所示，整体架构可以定义为扩展的三级模式-二级映射．三级模式分别为OLTP模式、OLAP模式和物理存储模式，二级映射为OLTP模式向OLAP模式的物理模式映射，和OLAP模式向物理存储模式的存储模式映射．

如图3(B)所示，传统数据库的三级模式-二级映射是在一个统一的数据库系统中，在模式层设计数据库的逻辑结构，在内模式层设计存储、索引等存储结构，在外模式层定义模式的子集作为用户视图．当外模式层面对OLAP负载时，逻辑视图机制难以独立选择优化的存储模型．

图3(A)为扩展的三级模式-二级映射机制，主要解决HTAP负载中不同模式、不同存储模型、不同存储平台的差异化问题．首先，数据划分为数据库端和存储端两个独立的存储平台，支持存算分离架构；其次，OLTP模式和OLAP模式可以采用不同的模式和物理存储模式，分别优化OLTP和OLAP负载；最后，通过对查询负载特征的优化，可以将OLAP负载中元数据管理功能分离到OLTP负载中，避免维表的冗余存储和管理．从管算存分离计算模型的视角来看，数据库端对应元数据管理层，采用OLTP模式，使用数据库内置优化的行存储模式，管理较小的维表和事实表增量delta数据，数据存储和处理的压力较小，既可以使用传统一体化设



（A）扩展三级模式-二级映射　　（B）传统三级模式-二级映射

图3　扩展三级模式-二级映射和传统三级模式-二级映射机制对比

计的数据库作为存储和处理引擎，也可以进一步通过存算分离技术借助云平台优化性能. 数据库端负责元数据的更新，事实表新记录的delta数据缓存，及OLAP查询中的维映射操作. 存储端对应存储计算层，包含OLAP模式和物理存储模式，采用最优的列存储模型和可扩展的存储引擎，实现将分析型数值集从数据库专用的内部存储引擎中分离出来，采用适合OLAP的优化模式，并通过底层独立的存储引擎或存储平台存储大数据，通过定制的计算实现存储端算子下推，将部分或大部分近数计算任务从数据库端转移到存储层，在最接近物理存储的位置完成存内计算. 扩展的三级模式-二级映射机制支持多模式和多存储结构，适应HTAP应用中OLTP与OLAP负载不同的存储访问需求，提高数据库管理和计算的灵活性，避免为兼容OLTP和OLAP负载而产生的不同存储模型的数据冗余存储代价.

3.2 存储优化

不同场景下数据库模式优化的目标不同，对于HTAP应用而言，不仅要实现行-列存储格式的转换，还需要实现面向TP(Transaction Processing)业务模式和面向AP(Analytical Processing)业务模式的转换，以适应TP与AP不同的应用需求.

最简单的OLAP模型是星形模型，单一的事实表易于分片及分布式存储与计算. 存储优化的两个基本方法是数值化和单一结构化，简化存储与计算. 下面以代表性的分析型负载TPC-H、SSB和TPC-DS为例讨论存储优化技术.

TPC-H模式来源于业务处理场景，Partsupp表中PS_AVAILQTY为实时库存数量，在分析处理中不具有较高的应用价值，PS_SUPPLYCOST可用于分析处理中的利润计算，适合于物化到Lineitem表中，通过增加一列减少大表连接操作. Orders表中仅有一个度量列O_TOTALPRICE，可以由Lineitem表中订单明细记录计算得出，在分析处理中可以被Lineitem表数据替代，Orders表中的O_ORDERDATE提供了一个日期维度，可以物化到Lineitem表中，三个低势集的退化维度可以整合为笛卡尔积退化维度表(O_ORDERSTATUS，O_ORDERPRIORITY，O_SHIPPRIORITY)，压缩存储空间. Orders表在分析处理中主要的作用是连接Customer维表，其外键可以物化到Lineitem表中以减少Lineitem与Orders大表之间的连接代价. 由于Orders表和Lineitem表中的主外键O_ORDERKEY/L_ORDERKEY具有偏序关系，将Orders表主键O_ORDERKEY转换为代理键后，Lineitem与Orders表可以看作是基于外键地址映射的虚拟宽表，也可以在保留Lineitem与Orders表结构的基础上通过优化的连接操作达到与物化的单一事实表相近的存储与计算效率.

SSB模式是将TPC-H模式物化为星形模型，专用于OLAP负载性能测试. 根据只读分析的需求，共享的Nation和Region表中属性被物化到Supplier和Customer表中，并增加了额外的Date维表用于不同日期层次上的分析任务. 在单一事实表Lineorder上除数值型的度量属性外还包含三个包含语义信息的低势集描述属性LO_ORDERPRIORITY，LO_SHIPPRIORITY，LO_SHIPMODE，可以进一步整合为退化维度以简化事实表数据类型与计算.

TPC-DS包含3个不同的渠道，每个渠道包含一个Sales表和Returns事实表以及多个共享维度，事实表只包含无语义的数值数据，Sales表和Returns事实表包含共享的外键以减少Returns事实表通过Sales事实表与维表的间接连接，减少大表连接代价. Sales和Returns事实表中的记录具有1∶1关系，可以通过在Sales表建立代理键索引实现事实表之间记录的直接地址映射连接，优化表连接性能并且统一数据分片策略.

综上所述，管算存分离的计算架构中，数值化存储和逻辑或物理单一的表结构可以简化存储层设计，将语义信息分离使从数据库中分离出的独立的存储层中的数据不易被解读，提高数据存储安全.

3.3 计算优化

管算存分离从模式上对数据库进行逻辑分层，从存储结构上对数据进行物理分层，从架构上对数据库的功能进行分层，将计算型数据和负载从数据库管理系统中分离到存储层，并进一步通过计算负载特征进行细化，简化数据库功能. 通过独立的、定制化的存内计算提高计算的局部性和开放性，使存储层可以服务于不同的数据库系统，应用不同的计算平台和存储技术，提高数据库系统应用的灵活性.

本文内存数据库的实现技术探索了采用独立的内存列存储平台，如Arrow提供面向不同系统的可共享访问的数据平台. 在存储平台提供定制化的多维计算算子，为数据库提供核心的OLAP计算功能.

3.3.1 多维计算模型

在多维模型中，OLAP查询是一种在多维数据

空间中的多维计算，在多维空间中检索数据并将其映射到由GROUP-BY构成的数据立方体中进行聚集计算. OLAP查询是一种模式定义的计算(Schema Defined Compute)，维结构定义了多维计算的维度，不同的查询共享统一的多维计算模型. 图4显示了多维计算模型的逻辑结构，数据库端的维表作为多维元数据用于解析数据和查询任务，查询任务分解在各维表上生成用于计算的维向量，维向量是与维表等长的向量索引，存储维表上的选择操作和GROUP-BY属性投影操作结果，并通过字典表压缩构建等价的GROUP-BY Cube，将分组立方体的多维坐标映射到相应的维向量压缩编码中[12]. 事实表外键列与维向量通过向量连接算法迭代地将分组立方体的多维地址计算结果存储于向量索引中，向量索引存储GROUP-BY Cube对应的一维分组向量地址，通过向量分组操作将事实表度量列上相关的数据映射到分组向量中进行聚集计算，最后分组向量通过数据库端维向量压缩字典解析出对应的GROUP-BY属性值，生成查询结果.

在多维计算模型中，事实表数据上的计算为多维计算，输入为模式所定义的维表对应的维向量，输出为查询中GROUP-BY语句所定义的相应长度的分组向量，输入与输出都是无语义的、压缩的数值型向量，计算也是在无语义数值上的代数计算，存储端只负责计算任务但无法感知及解析计算的语义，从而保证存储端数据的安全性和隐私性. 不同的查询任务使用相同的输入输出数据结构，对应不同的数值，使用相同的计算模型，事实表上的计算完成OLAP查询中绝大部分计算任务，输入的维向量较小，输出为分组长度的向量，最小化了查询输出结果集大小和与数据库端的数据传输代价. 在实现层面，需要在存储层按数据库模式定制一个计算接口，将OLAP查询中的多维计算下推到存储层.

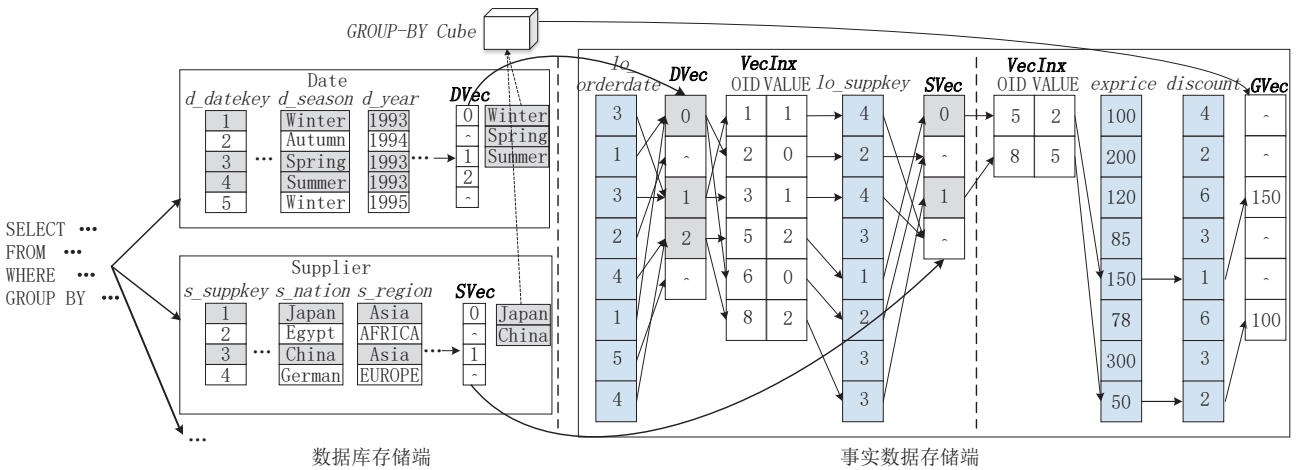


图4　多维计算模型

3.3.2　多维计算算子实现

以下面查询为例说明多维计算算子实现方法：

```
SELECT d_season,s_nation,sum(exprice* discount)
FROM lineorder,date,supplier
WHERE lo_orderdate = d_datekey
AND lo_suppkey=s_suppkey
AND d_year = 1993
AND s_region='ASIA'
GROUP BY d_season,s_nation;
```

在数据库端，SQL命令解析后分解为在维表Date和Supplier上的选择和投影操作，投影出的GROUP-BY属性在维向量中被压缩并替换为压缩编码，Date表中*d_seasons*中包含3个成员，Supplier表中*s_nation*中包含2个成员，GROUP-BY属性构成一个3×2多维数组，维向量中的压缩编码表示该记录的属性值在多维数组中的分量坐标. 图4中维向量为***DVec***和***SVec***，向量索引VecInx采用压缩结构，由OID和VALUE组成，OID记录满足连接条件的记录的偏移地址，VALUE为连接中迭代计算的GROUP-BY多维数组的一维坐标(分组向量下标). 外键列*lo_orderdate*将值映射到维向量DVec相应的位置，用非空单元值计算出分组向量下标值，记录到压缩向量索引VecInx中. 然后顺序扫描压缩向量索引VecInx，按OID值访问下一个连接外键列*lo_suppkey*，与维向量***SVec***执行向量连接操作，将映射的非空单元值与压缩向量索引中存储值迭代

计算分组向量下标值.经过连接操作后,压缩向量索引中存储了最后参与聚集计算的度量记录的OID和其在分组向量中的下标值,通过OID访问度量列,完成聚集计算后将结果映射到分组向量*GVec*中进行聚集计算,最终的分组向量*GVec*为存储端返回数据库端的计算结果.

在实现中进一步应用向量化查询处理方法将事实表列划分为适合L1 cache大小的向量组(长度约1024),每个向量组数据分片的处理采用流水线方式,数据与向量索引的物化产生在L1 cache中,消除了列存储数据处理时中间结果的内存物化代价,提高查询处理性能和内存利用率.

从计算效率视角来看,多维计算算子包含了传统查询处理中的多表星形连接和分组聚集操作,由于向量索引中集成了压缩分组编码,因此可以在类似布隆过滤器的连接过滤操作中完成分组计算,从而将OLAP查询中主要的选择、投影、连接、分组、聚集操作下推到存储层,实现存储端计算最大化,存储端与数据库端数据交互最小化.从数据库系统功能分层的视角来看,数据库端存储多维数据集的元数据,负责查询解析与计算任务分派,在较小数据集上的管理和查询处理成本较低,存储端存储大量数据,多维计算的数据局部性较强,能够最大化发挥存储端CPU资源的计算效率,而且存储层只负责无语义的数值计算,无法了解计算的语义信息,使独立的存储层计算功能具有安全性,减少扩展时的数据耦合性,减少存储分离产生的信息泄露风险.

事实表数据上的计算以向量索引为分界划分为两部分:外键列上的多表连接操作生成向量索引;基于向量索引的事实表度量列上的分组聚集计算.当使用同构的内存存储时,这两个计算过程可以合并到统一的向量化查询处理模型中实现最大化内存局部性计算,当使用不同的存储设备时,如使用GPU显存存储事实表外键列并加速多表连接操作时,事实表上的多维计算算子进一步划分为GPU加速的计算层和基于内存存储的存内计算层.

3.4 管算存分离计算架构实现技术

管算存分离计算架构是根据数据库模式、数据的负载特征对数据库的管理功能和存储功能进行分层,将元数据管理、查询解析、任务调度等数据库核心功能分派给数据库端,将数据存储及存储端计算功能分派给存储层,将大数据上的数值计算功能从数据库层中分离出去,以便于独立扩展与开发.数据库的多维模型中元数据与数值存储边界清晰,OLAP操作中多维计算适合于下推到存储端执行,是理想的管算存分离计算架构实现案例.

3.4.1 元数据管理端实现技术

按照模式和存储优化策略,内存OLAP数据集分解为维表和事实表两部分,对应管理层(数据库端)与存储层(存储端),数据库端对输入的SQL命令进行解析和改写,生成存储端多维计算接口需要的维向量,存储端根据输入的维向量调用定制的多维计算接口,完成存储端计算任务后将分组向量返回数据库端,由数据库端对分组向量进行解析输出.

以3.3节示例SQL命令为例说明数据库端的处理过程.原SQL命令在维表Date和Supplier上有GROUP-BY属性,需要在其上构建维向量.将SQL命令改写为如下两个SQL命令,在维表上根据WHERE子句投影出GROUP-BY属性,由两个维表的GROUP-BY属性构建查询的多维分组,在维表端对GROUP-BY属性字典表压缩并编码,字典表连续的编码用于构建GROUP-BY分组的两个维坐标.

```
SELECT d_season, d_datekey
FROM date
WHERE d_year = 1993
```

```
SELECT s_nation, s_suppkey
FROM supplier
WHERE s_region='ASIA'
```

如图5所示,传统的查询计划中维表选择、投影出来的属性用于构建哈希表,如将满足WHERE条件的Date表的*d_datekey*,*d_season*属性构建用于连接操作的哈希表.本文采用维向量映射方法,将满足WHERE条件的*d_season*属性投影为维向量,维向量记录每条记录输出的分组属性,空值表示该记录无输出.输出的分组属性*d_season*通过哈希表进行字典压缩,当在哈希表未找到该属性值时,向全局自增序列申请编码值作为该分组属性值的编码,并将编码写入维向量;当分组属性在哈希表中找到相同值时,直接将该属性值在哈希表中的编码写入维向量相应单元中.最终,维表输出压缩的维向量,表示SQL命令在该维表上的GROUP-BY属性编码.

在数据库的元数据管理端需要增加扩展的维向量生成器接口,在传统的哈希表基础上进行功能扩

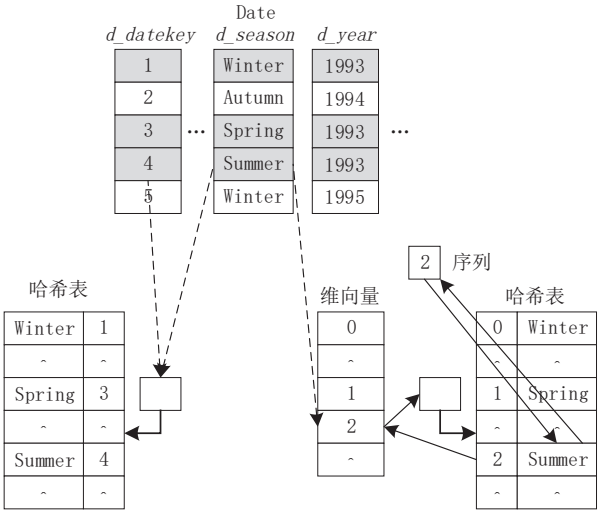


图 5　维哈希表与维向量生成过程

充，利用维表的代理键索引特性用简单的向量代替哈希表，通过自增序列和哈希表对GROUP-BY属性进行压缩编码.

维向量生成是在较小维表上的低势集GROUP-BY属性压缩编码的过程，需要latch机制保证字典编码的唯一性，在实现中有较高的并发访问控制开销，适合于较低的线程并发度.

3.4.2　存储端实现技术

优化的星形模型和维向量映射技术简化了存储端的多维计算. 如图6所示，事实表存储于专用的存储平台Arrow，存储端需要定制专用的多维计算接口执行下推的OLAP多维计算算子. 多维模型在查询中分解为若干查询相关的维向量，并根据维向量的分组哈希表构建分组向量，事实表记录依次通过各维向量进行多维过滤并计算过滤记录的多维分组地址，过滤后的记录映射到分组向量中聚集计算. 多维计算是一种模式定义计算(Schema Defined Compute)，多维数据集的结构确定了统一的多维计算模型，不同OLAP查询使用相同结构的维向量和分组向量，不同查询只是动态改变维向量中数值的分布(由WHERE子句和GROUP-BY属性)及分组向量的长度(由GROUP-BY属性决定). 存储层采用编译的执行代码，采用“推”(push)模型提高计算中的数据局部性，事实数据“推”向各维向量过滤器执行计算.

多维计算是一个基于共享维向量只读访问的处理过程，分组向量较小，在多线程并行算法设计中可以作为线程私有向量使用，消除线程间并发访问控制代价，多维计算适合于较高的线程并发度来提高CPU的利用率.

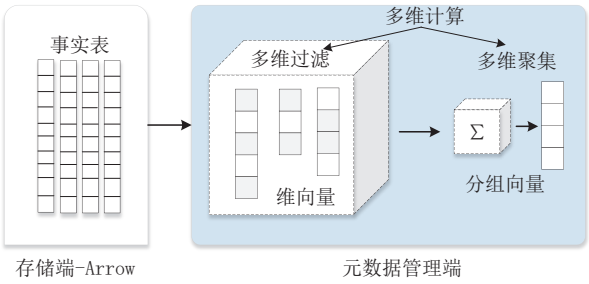


图 6　存储端实现技术

多维计算操作算法描述如下：

算法1.　多维计算MDCompute.

输入：维向量列表***DVecList***，聚集表达式*AggExp*

输出：分组向量***GVec***

GVec=*init*(***DVeclist***)//根据维向量初始化分组向量

BEGIN

　FOR EACH ***DVec*** IN ***DVeclist*** DO

　　FKCol=*addressof*(***DVec***, ***VecInx***)//获取相应外键列地址

　　VecInx←*VecStarJoin*(***DVec***, ***FKCol***)；

//基于维向量执行星形向量连接操作，更新向量索引

　END FOR

GVec=*VecAgg*(***VecInx***, *AggExp*)//执行向量聚集计算

Return ***GVec***

END

图7显示了一个OLAP查询执行示例. 传统的查询树转换为一系列编译的执行代码，在数据库管理端执行维表上将满足选择条件记录的GROUP-BY属性压缩到维向量的功能，在存储端执行基于向量索引的多维计算功能，最后将压缩的计算结果转换为查询输出结果.

通过管算存分层，数据库端与存储端功能模块的扩展可以独立进行，在存储端执行的多维计算可以进一步通过GPU等硬件加速器加速计算性能，也可以结合非易失内存技术进一步提高数据存储容量，降低存储成本. 管算存分离将OLAP查询处理过程分解为两个(或三个)层次上的由数据驱动的计算过程，当查询任务较重时可以在查询间流水并行，即在存储层执行第n个查询任务的多维计算任务时数据库端执行第$n+1$个查询任务的解析以及维向量映射计算，提高两个不同层次的计算总体效率.

集中式内存数据库可以使用Arrow作为存储平台. 分布式数据库可以应用云存储技术提高存储的扩展性，通过将巨大的事实表划分为适当大小的分片，将多维计算任务下推到各数据分片完成并行多

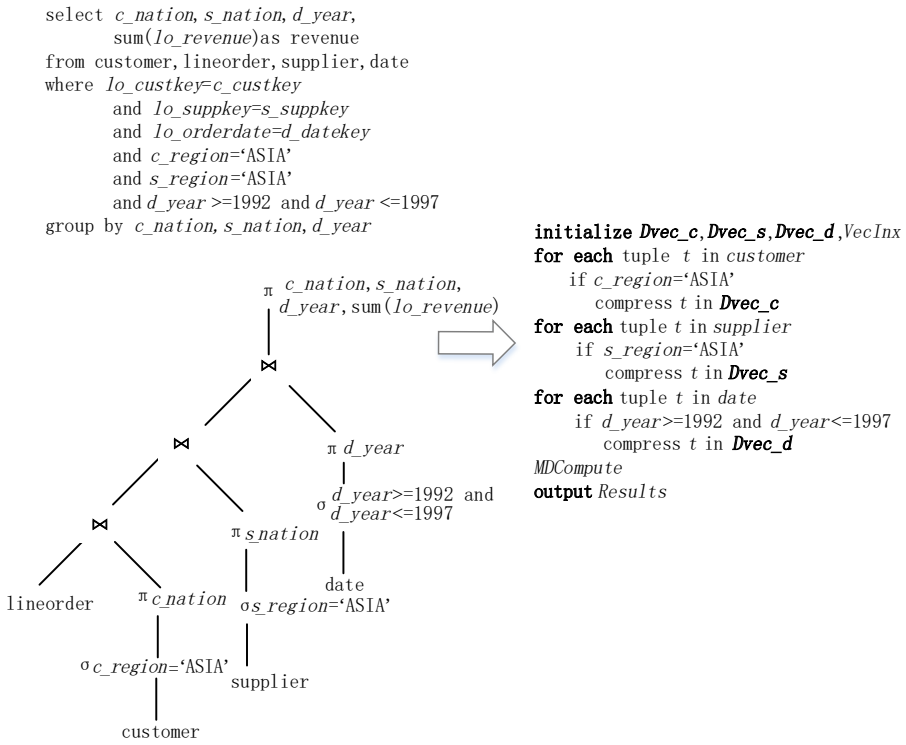


图 7 查询执行示例

维计算任务，并将各分片对应的分组向量归并为全局分组向量，而较小的维表可以集中存储于数据库节点，仅执行较轻负载的查询解析、任务调度、归并计算等任务. 为进一步提高云计算的安全性，管算存分离架构可以将较小的多维元数据管理层从云平台迁移到企业私有数据库平台[13]，只将大数据量的无语义事实数据存储于云平台，查询在企业私有数据库平台进行解析并生成维向量，较小的维向量以较低的网络传输延迟发送到云平台进行多维计算，计算结束后多维计算生成的较小的分组向量再返回企业私有数据库平台进行解析，向用户返回查询结果语义.

存算分离一方面是为适应存储技术的发展而将数据库分层并分而治之地独立扩展存储与计算功能，另一方面也通过独立的存储层将传统数据库通过SQL接口的共享方式扩展为数据访问的共享，使不同数据库系统共享相同的数据存储成为可能，不同的数据库可以更加聚焦于数据库端查询优化技术的发展，同时，存储层的独立和存储端计算的开放性有助于具有不同技术背景的技术力量进入细分的数据库开发领域，共同推动数据库技术的发展. 管算存分离技术进一步细化了OLAP应用在存算分离架构下的实现技术，通过基于多维模型特征的模式及存储优化技术，实现“数”与“据”分层，强化存算分离架构下数据库端的元数据管理功能，同时加强存储层无语义数值存储和计算功能，在提高存储效率的同时将数据中包含的语义信息上推到元数据层，降低存储层数据的解读能力，增强在数据库外部存储层上的数据安全. 管算存分层结构定义了开放的OLAP计算框架，可以结合不同的硬件技术、不同的存储设备、不同的技术团队独立扩展及提升数据库管理、多维计算及存内计算的功能和性能，以更好地适合不断变化的硬件发展趋势.

4 实验结果与分析

本文实验的主要目标是验证三个目标：(1)基于管算存分离的OLAP技术在CPU-GPU混合计算平台和DRAM-PM异构存储平台的实现与性能，验证不同负载分离模式的性能特征；(2)基于独立的内存Arrow存储的OLAP实现及性能；(3)对比分析基于管算存分离的不同OLAP实现技术与代表性的内存数据库及GPU数据库的性能差异，分析基于管算存分离的OLAP技术面向未来新型硬件异构计算平台的实现技术.

实验平台为一台2U服务器，配置有2块Intel Xeon Gold 6138 @ 2.00 GHz 20核心CPU，共80个物理线程，27.5 MB L3 cache，512 GB DDR4内存，

操作系统为CentOS 7，Linux版本为3.10.0-514.16.1.el7.x86_64，gcc版本为4.8.5. 服务器配置了1块NVIDIA Tesla V100 GPU，集成了5120个流处理器，显存容量为32 GB，显存带宽达到900 GB/s. DRAM-PM异构存储平台配置为双路Intel Xeon Platinum 8368(ICX)28核处理器，57 MB L3 cache，512 GB DDR4内存，2 TB OPTANE持久内存.

实验中使用SSB数据集①(*SF*=100)验证基于管算存分离的OLAP实现技术的查询处理性能，主要性能指标为查询执行时间(毫秒)和平均查询执行时间(毫秒). 本文基于管算存分离的OLAP技术采用编译代码执行方式，由编译的C++语言代码实现面向维表元数据管理的维向量生成、面向事实表外键列的多表连接和面向事实表度量列的分组聚集计算，算法实现采用了向量化查询处理技术以提高在CPU平台的cache效率和性能，与当前主流内存数据库采用的JIT实时编译技术、向量化查询处理技术具有相同的优化级别. 实验中对比了学术界性能最优的内存数据库Hyper和开源OmniSciDB在相同数据量下的SSB查询性能，测试查询连续执行三次取最短时间作为最优的查询性能指标，消除内存数据库和GPU数据库第一次查询时的数据加载时间及可能的CPU负载波动的影响，最小化数据库查询执行时的系统开销和实时编译代价，OmniSciDB支持CPU和GPU两种模式，可以提供当前主流CPU-GPU异构平台上的高性能数据库性能参照.

4.1　面向异构硬件平台的管算存分离OLAP技术

基于管算存分离的OLAP技术采用三级存储和三级计算模型：元数据存储，存储维表数据；连接列存储，存储事实表外键列作为多表连接操作集；事实数据存储，存储事实表度量列用于分组聚集计算. 维向量作为元数据存储的中间交换数据，向量索引作为显式或隐式的连接操作中间交换数据，分组向量作为分组聚集计算的结果集. 三级计算包括：维表元数据上根据SQL命令参数生成维向量；连接列根据维向量生成向量索引；事实数据上根据向量索引生成分组向量. 三级存储可以存储于不同的存储引擎、存储平台或存储设备上，三级计算依赖不同的处理器对应不同的存储层，灵活配置存储与计算资源以达到提高存储效率或查询处理性能的目标，也使OLAP计算在面向异构高性能计算平台时具有灵活性和适配性.

如图8所示，本小节演示了管算存分离OLAP计算模型在CPU-GPU异构计算平台和DRAM-PM异构存储平台上的实现技术，三种线型代表细分的三种不同的计算模型.

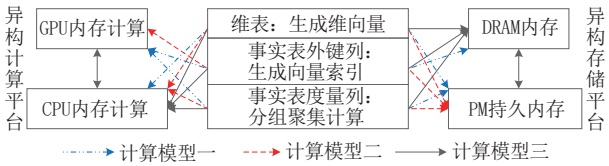


图8　异构计算平台管算存分离OLAP实现技术

在CPU-GPU异构计算平台实现了三种OLAP计算模型：CPU-host计算模型，GPU-host计算模型和GPU加速模型，分别对应CPU平台上的管存分离、CPU-GPU混合平台的管存分离和管算存分离计算模型.

在三种计算模型中，维表数据具有数据量小、数据类型多样、数据可更新等特征，适合CPU处理，因此统一设置为CPU端计算模块. 将SQL命令中各维表上的选择、投影、分组属性及表达式作为输入参数，调用维向量生成API，为OLAP计算创建相应的维向量.

在CPU-GPU异构计算平台中，CPU-host计算模型将连接列和事实数据合并存储于主存，通过向量化查询处理技术将连接列上的计算与度量列上的计算合并，实现cache内的向量索引生成和基于向量索引的分组聚集计算，生成最终的分组向量结果，达到优化内存访问和提高查询性能的目标.

GPU-host计算模型将连接列和事实数据合并存储于GPU显存，同样采用向量化查询处理技术，在GPU的shared memory中通过向量化处理优化向量索引的存储访问，消除向量索引对显存的占用并降低计算延迟.

GPU加速模型将较小的连接列存储于GPU显存，较大的事实表度量列存储于主存，GPU端执行代价较高的多表连接计算，生成向量索引并传回CPU主存，由CPU根据向量索引在事实表度量列存储层上执行计算代价较小的向量索引分组聚集计算. GPU加速模型的第一、三级计算代价较轻的负载分布在CPU，第二级计算代价较高的负载分布在GPU，一方面可以利用GPU强大的并行计算性能加速核心计算负载，另一方面在多查询模式可以进

① Star Schema Benchmark, https://www.cs.umb.edu/~poneil/StarSchemaB.PDF, 2021,9,17

一步将CPU端与GPU端的计算流水并行，在GPU端执行第n个查询的连接计算时，CPU端执行第$n-1$个查询分组聚集计算和第$n+1$个查询的维向量计算，通过流水处理掩盖PCIe传输延迟和处理器空闲，提高系统的整体处理效率.

图9显示了CPU-host、GPU-host和GPU加速模式的SSB查询平均执行时间. 在CPU平台维向量生成时间相对较短，元数据管理层的计算代价较小，主要代价集中于事实表存储层上的连接和分组聚集计算. GPU平台加速了事实数据上的连接和分组聚集计算性能，较CPU平台有4.2倍的性能提升，并且低于CPU端的维向量生成时间，GPU强大的计算性能和较小的显存容量使GPU-host计算模式出现较大的性能偏斜. 在SSB数据集中，维表数据量占比约为1%，事实表外键列数据量占比约为20%，事实表数据量占比约80%，GPU加速模式只需要在GPU显存存储数据集总量20%的外键列数据用于加速多表连接计算，而GPU-host计算模式需要在GPU显存中存储全部数据，因此GPU加速模式中GPU显存存储数据对应的数据集总量大约是GPU-host计算模型的4倍，GPU加速模式比GPU-host模式存储更小比例的数据子集，从而可以支持更大的数据集进行GPU计算加速. 实验中采用的是单查询CPU、GPU端分阶段顺序执行模式，图9显示了各执行阶段执行时间和占比. 维表虽然数据量占比仅1%左右，但其多样化的数据类型及维向量创建阶段的latch并发访问控制机制适用于较低的线程并发度，在执行时间上高于GPU显存上数据量占比超过98%的事实表处理阶段. GPU加速模式中GPU端事实表外键多表连接计算占比略低于50%，与CPU端维向量生成和分组聚集计算时间占比接近，GPU的计算资源未充分利用. 从查询平均执行时间指标来看，顺序执行的GPU加速模式比GPU-host模式性能低2.5倍，甚至低于CPU-host模式，但若采用CPU与GPU端的流水并行处理，则平均查询预期执行时间可以缩短到100 ms，高于GPU-host模式的39 ms但低于CPU-host模式的106 ms. 从CPU-GPU异构计算平台的利用率角度来看，完全GPU显存副本模式的OLAP查询处理具有最优的性能，如文献[12]将维表数据通过压缩编码方式转换为GPU存储数据，通过原子操作的分组属性直接映射低延迟创建哈希表使SSB平均查询执行时间缩小到20 ms，但这种全数据复制方式失去维表元数据管理的功能和与OLTP负载的融合能力. CPU端维表元数据管理的维向量映射计算相对于GPU端事实表多维计算时间较长但使用线程数量较少，在并发查询处理应用场景中可以进一步通过在CPU端并发执行多个查询的维向量映射计算来充分利用CPU的多线程计算资源，构建多路流水线，使GPU显存事实表上的多维计算消除CPU端维映射计算的等待时间，通过并发查询队列连续执行多维计算任务，提高GPU计算资源利用率.

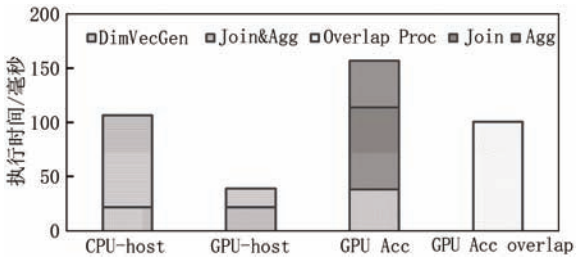


图9 CPU-GPU平台SSB分段平均查询执行时间

图10显示了CPU-host、GPU-host、GPU加速和GPU overlap模式在SSB测试中的查询性能和平均查询时间.

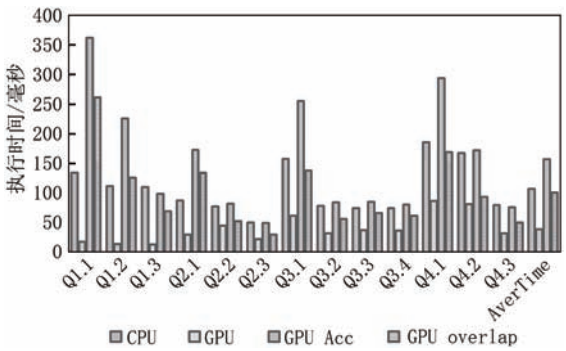


图10 CPU-GPU平台SSB平均查询执行时间

在CPU-host的管存分离模式下，事实数据上的计算代价为查询主要代价. GPU-host模式显示了GPU在存内计算上卓越的性能，在HBM2高带宽显存和高并行计算性能的支持下不仅GPU存储端平均计算时间缩小到17 ms，而且不同选择率的查询时间差异减少，表现了较高的性能和稳定性. GPU加速模式对应异构存储与异构计算平台，体现了异构存储模式下的管算存分离的OLAP计算架构. GPU内存外键列数据量中等但计算负载较高，实现将计算层负载用GPU进行加速的目标，当采用CPU和GPU端计算流水并行的GPU overlap策略时，总体查询时间可以进一步降低.

在DRAM-PM异构存储平台实现了三种

OLAP计算模型：DRAM-host计算模型，PM-host计算模型和DRAM优化模型，分别对应DRAM平台上的管存分离、PM混合平台的管存分离和DRAM-PM异构存储平台管算存分离计算模型. DRAM-host计算模型采用全DRAM存储、向量化查询处理方法，PM-host计算模型采用全PM存储、向量化查询处理方法，DRAM-PM异构计算模型采用事实表外键列DRAM存储、物化向量索引(DRAM-optimized)或向量化查询处理(DRAM-vecoptimized)方法.

图11显示了DRAM-PM异构存储平台上管算存分离OLAP模型不同实现方法的分段平均查询时间. 由于PM在内存访问性能上低于DRAM，因此PM-host模式的执行时间为DRAM-host的3.58倍，性能差距较大. DRAM-optimized模式和DRAM-vecoptimized模式将事实表外键列存储于较小但性能较高的DRAM中，前者使用物化向量索引技术，后者则直接在DRAM和PM之间使用向量化查询处理技术，实验结果表明二者性能差距较小，相对于PM-host模式性能有较大的提升，接近DRAM-host模式，实现了管算存分离OLAP计算模型向DRAM-PM异构存储的映射，实现将计算密集型负载与高性能DRAM存储的优化匹配.

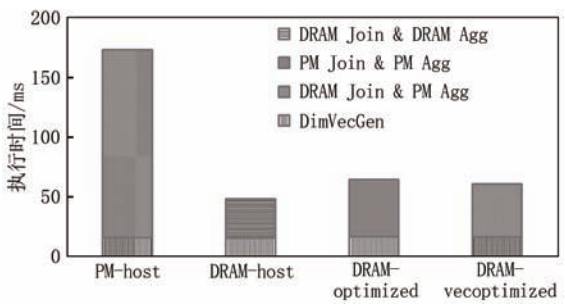


图11　DRAM-PM平台SSB分段平均查询执行时间

图12显示了四种查询处理方法在13个查询上的时间对比，总体而言，当选择率较高时，DRAM-host实现方法的性能较其他三种方法性能优势较大，当选择率低时则性能较为接近，这种现象主要原因是低选择率的cache miss访问延迟削弱了DRAM与PM内存访问延迟的影响. 总体而言，通过较小容量、较高性能DRAM优化“算”的负载能够有效地提升OLAP查询整体性能.

4.2　基于Arrow存储的OLAP实现技术

Apache Arrow已经成为一个成熟的工业标准内存列存储格式，主要应用于内存分析处理领域.

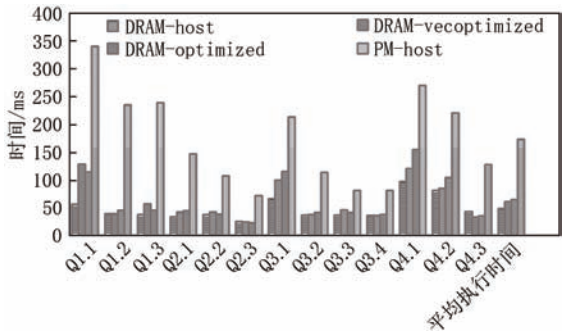


图12　DRAM-PM平台SSB平均查询执行时间

Arrow为不同的系统提供了统一的内存数据列式访问接口，为内存数据库的存算分离提供了存储支持，本节探索基于Arrow的管存分离架构OLAP实现技术以及基于Arrow存储的存内计算实现技术.

在优化的OLAP模式中，事实表主要由数值型的连接外键列和度量属性组成，数据结构简单，数据量和增量较大，主要提供列扫描及基于位图的按位置访问操作. Arrow基于C++ vector向量类实现，封装了field、schema、table对象表示列、模式和表，提供了对表数据的存储和列数据访问接口，可以用作数据库扩展的内存列存储外部表使用，为传统数据库扩展内存列存储访问能力. Apache Arrow基本存储格式是数组(Array)，能为OLAP分析处理提供良好的列式访问数据能力. Apache Arrow提供了Tabular Data存储格式，Tabular Data存储格式中的arrow::table实例由两部分数据组成：以行存形式组织的Table元数据和以列存形式组织的Table数据. 我们定义了DataBase::DB_Opera类来导入、管理和读取四个维表part表、date表、supplier表以及customer表数据. DataBase::DB_Opera类里封装了内存数据访问接口访问Apache Arrow中的arrow::table实例中列式存储数据，Get_element_data_string_arrow()接口根据表名、列名以及偏移地址返回具体的std::string类型值，主要用于访问维表数据，Get_array_data_int_arrow()接口根据表名和列名返回数据类型为int的数组地址，主要用于访问数值存储的事实表. 我们在实验中将SSB的tbl数据文件加载到Arrow中，作为共享访问的内存数据集，模拟独立的内存列存储平台. 在Arrow存储层上创建多维计算层，用于将事实表上的多维计算任务下推到Arrow存储层，实现基于Arrow的存内计算，维表上的处理模拟数据库查询处理引擎功能，构建一个基于Arrow存储的管存分离OLAP计算架构. 设计了维表上的维向量映射计算接口和事实表上多维计算接口，

分别模拟本文管算存计算模型中维表管理端功能(维表采用行列混合存储,模拟数据库的内存行表存储结构)和事实表存储端(采用列存储)的存内计算功能.

宏观上,我们将Arrow设计为存储计算层,实现OLAP多维计算任务的存储级计算功能,在微观上,我们在Arrow上扩展了一个轻量的多维计算层,Arrow作为插件式存储引擎仅提供内存列存储、内存列顺序扫描、内存列按位置访问等基础功能,由计算层完成基于底层Arrow存储服务的OLAP多维计算功能.当不使用Arrow存储时,可以通过列存储接口转换为其他候选的存储引擎,如内存数组表、vector表、数据库内置内存列存储引擎等,实现图2所示物理存储模式的灵活选择.

从SSB查询平均时间来看,如图13所示,基于数组存储和Arrow存储的OLAP查询处理性能非常接近,反映了Arrow存储较好的内存访问性能,接近甚至略优于最简的内存数组存储访问性能,可以作为传统的磁盘数据库系统扩展的高性能内存列存储引擎.在查询执行时间中,维向量生成时间占比低于25%,事实表数据上的多维计算执行时间超过75%,事实表数据上的存内计算是查询优化的重点.

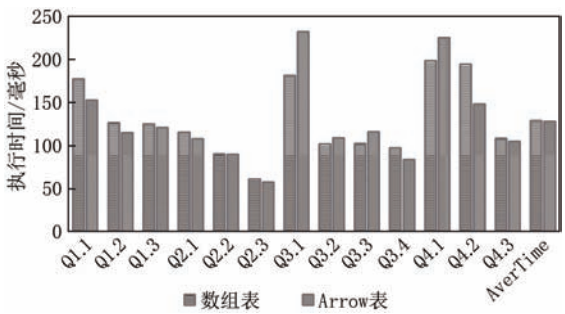


图13 数组表与Arrow表SSB性能对比

图14显示基于Arrow存储的OLAP加速引擎在SSB基准测试中的性能,其中PostgreSQL、PG-Strom均采用内存模式,将数据加载到tmpfs内存文件系统.从查询平均时间指标来看,基于Arrow存储的OLAP加速引擎优于当前最优的内存数据库Hyper,OmniSciDB的GPU以及CPU版本,相对于基于Arrow存储的GPU数据库PG-Strom性能有较大的优势.从当前内存数据库的技术路线来看,专用的内存数据库采用一体化设计,在内存列存储引擎上的向量化查询处理和JIT实时编译技术的支持下具有较高的性能,但底层算子优化技术与存储引擎绑定较紧密,当存储或处理器技术发生改变或技术升级时产生较高的系统维护成本.

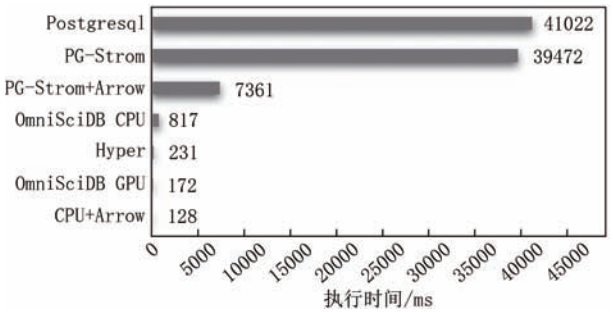
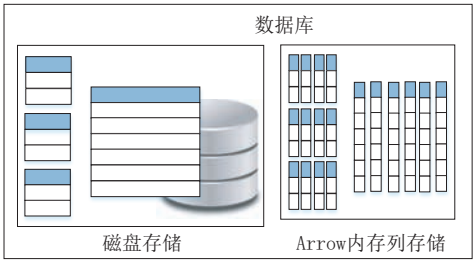


图14 基于Arrow的OLAP性能与数据库对比

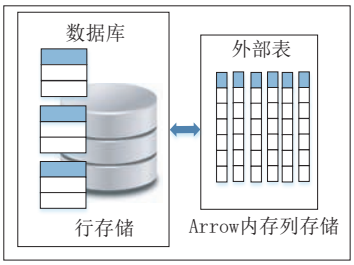
如图14所示,磁盘数据库PostgreSQL和PG-Strom在磁盘模式下有一倍的性能差距,但在内存模式下性能差距不显著,PG-Strom的GPU加速仅有较小的性能收益,性能主要受内存page结构数据访问效率影响.基于Arrow存储的GPU数据库PG-Strom相对PG-Strom采用了Arrow内存列存储结构,通过外部表Arrow提高存储访问性能,通过GPU提高查询处理性能,可以看作是一种存算分离计算架构,相对于PostgreSQL和PG-Strom性能有显著的提升,但系统仍然采用传统PostgreSQL的查询处理引擎访问Arrow数据,无法应用现代内存数据库优化的向量化查询处理技术以及优化的内存连接、分组等算法,综合性能仍远远低于专用的内存数据库和GPU数据库系统,传统磁盘数据库查询处理引擎仅提升存储访问性能难以有效提高OLAP综合性能.

本文所述管算存分离计算架构适用于面向HTAP负载的混合数据库技术,适用于不同的数据库应用场景.图15(A)显示了传统磁盘数据库基于Arrow存储的内存OLAP加速引擎方案:双存储引擎方案是在传统磁盘数据库中集成Arrow内存列存储引擎,创建磁盘行存储数据的内存列存储副本,并通过Arrow上定制的OLAP加速模块加速数据库的OLAP分析处理性能.图15(B)显示了外部OLAP加速引擎方案:扩展行列混合存储方案是由数据库内部的传统磁盘存储引擎或内存行存储引擎管理维表元数据,提供面向OLTP负载的事务处理能力,Arrow作为外部表存储事实表数据,数据库负责多维数据的元数据管理和SQL查询解析、改写、任务调度,创建查询维向量发送给外部Arrow存储引擎执行本地化的多维计算任务,再将计算结果传回数据库解析输出,为数据库提供一个完全自治的外部存储平台和多维计算服务,卸载数据库在OLAP查询中的计算负载.本文所述异构CPU-GPU内存计算和DRAM-PM异构存储OLAP计算

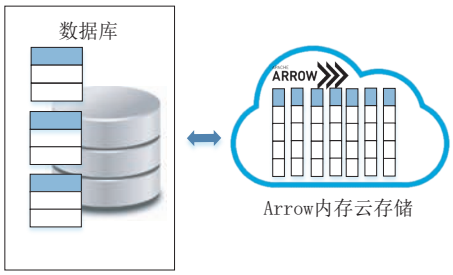
都可以应用于独立的Arrow存储平台．图15(C)显示了将存储层扩展为独立的云存储计算平台方案：数据库＋云存算一体化平台方案根据未来软/硬件发展趋势，将Arrow扩展为独立的存算一体化平台，提供高性能、可扩展内存云存储能力和存算一体化设计，与数据库管理引擎协同支持基于高可扩展、高性能、弹性的内存云计算平台的HTAP数据库．



（A）双存储引擎结构



（B）扩展行列混合存储结构



（C）数据库+云存算一体化平台结构

图15　基于Arrow列存储引擎的扩展存储方案

基于Arrow的外部独立存储计算平台可以进一步与新硬件技术相结合，如通过CPU-GPU异构计算平台提高多维计算性能，通过DRAM-PM异构存储平台提高内存计算性价比．在与云计算技术结合方面，数据库经历了云部署、基于存算分离的云原生数据库技术方案，进一步通过本文介绍的管算存分离计算架构实现数据库＋云存算一体化平台方案实现数据库与云计算平台基于数据和模式特征的优化分布和协同计算，为数据库提供插件化、高性能、可扩展的大数据多维计算能力．

本文的技术方案不仅借助于开源的Arrow提供内存列存储数据平台，而且通过在Arrow存储的事实表上定制的存储级计算最大化了Arrow数据的本地存储访问和计算性能，使Arrow不仅可以成为数据库动态扩展的外部表，还提供了存储内计算能力，在数据库存算分离的基础上进一步验证了数据管理与存储级计算分离的实现方案．

5　结束语

存储和处理器等新硬件技术的发展为提高数据库的性能提供了硬件基础，但同时也增加了数据库系统存储与查询优化技术升级的复杂性和系统开发成本．本文提出的管算存分离计算模型实现元数据管理和无语义数据存储的分离，将数据库OLAP负载中事实表的存储和计算从数据库引擎中分离出去，根据硬件平台定制优化的存储和计算，通过新硬件技术加速核心的事实数据上的计算性能，实现存储/计算功能与数据库管理功能的独立，增强数据库面向新硬件的扩展能力．另一方面，管算存分离的计算模型通过元数据与事实数据的分离进一步丰富了HTAP实现技术，在行-列存储模型转换、CPU-GPU平台负载分布[14]等技术的基础上通过三级模式-二级映射实现元数据管理端OLTP模式、存储端OLAP模式和物理存储端不同存储模型和引擎的异构匹配，为不同层次选择最优的模式、存储模型和算法实现，支持更加灵活的HTAP应用支持．同时，通过管算存分离计算模型，传统的数据库内置存储引擎可以扩展为更加独立的存储平台，开发面向新型硬件的存储和计算技术，如面向非易失持久内存的Arrow存储技术和面向大容量显存GPU平台的OLAP计算技术等，为数据库提供多样化的大数据存储和计算服务支持，也可以使高性能计算和存储背景的技术人员摆脱复杂的数据库系统的开发工作，更加关注于面向新硬件特征的存储平台开发与存内计算优化设计，更好地发挥各自的特长．

致　谢　感谢华为鲲鹏应用实验室的陆云飞、阙鸣健对本文的技术支持．

参　考　文　献

[1] Verbitski A, Gupta A, DSahaet al. Amazon Aurora: Design Considerations for High Throughput Cloud-Native Relational Databases//Proceedings of the 2017 ACM SIGMOD, Chicago, USA, 2017: 1041-1052

[2] Verbitski A, Gupta A, DSahaet al. Amazon Aurora: On Avoiding Distributed Consensus for I/Os, Commits, and Membership Changes//Proceedings of the 2018 ACM SIGMOD, New York, USA, 2018: 789-796

[3] Depoutovitch A, Chen C, Chen Jet al. Taurus Database: How to be Fast, Available, and Frugal in the Cloud//Proceedings of the 2020 ACM SIGMOD, Portland OR, USA, 2020: 1463-1478

[4] Yu X, Youill M, MWoiciket al. PushdownDB: Accelerating a DBMS Using S3 Computation//Proceedings of 2020 IEEE 36th Intl. Conf. on Data Engineering (ICDE), Dallas, USA, 2020: 1802-1805

[5] Cao W, Liu Y, Cheng Zet al. POLARDB Meets Computational Storage: Efficiently Support Analytical Workloads in Cloud-Native Relational Database//Proceedings of the 18th USENIX Conf. on File and Storage Technologies, Santa Clara, USA, 2020:29-41

[6] Zhang Yansong, Zhang Yu, Wang Shan, Lu Jiaheng: Fusion OLAP: Fusing the Pros of MOLAP and ROLAP Together for In-Memory OLAP[J]. IEEE Transactions on Knowledge and Data Engineering, 2019, 31(9):1722-1735

[7] Zhang Yansong, Zhang Yu, Lu Jiaheng, Wang Shan, Liu Zhuan, Han Ruichen: One size does not fit all: accelerating OLAP workloads with GPUs. Distributed Parallel Databases, 2020,38(4): 995-1037

[8] Balkesen C., Teubner J., Alonso G., and Ozsu T. Main-memory hash joins on multi-core cpus: Tuning to the underlying hardware//Proceedings of the International Conference on Data Engineering(ICDE). Brisbane, Australia, 2013: 362-373

[9] AnilShanbhag, SamuelMadden, Yu Xiangyao. A Study of the Fundamental Performance Characteristics of GPUs and CPUs for Database Analytics. SIGMOD Conference, Portland, USA, 2020: 1617-1632

[10] Zhang Yansong, Wang Shan, Lu Jiaheng. Improving Performance by Creating a Native Join-Index for OLAP. Frontiers of Computer Science in China, 2011, 5(2):236-249

[11] Zhang Y, Zhou X, Zhang Y, Zhang Y, Su M, Wang S. Virtual Denormalization via Array Index Reference for Main Memory OLAP. IEEE Transactions on Knowledge and Data Engineering (TKDE). 2016,28(4): 1061-1074.

[12] Zhang Yansong, Zhang Yu, Wang Shan. A Novel In-Memory OLAP Star Join Acceleration Technique with Vector Index. Chinese Journal of Computers, 2019, 42(8): 1686-1703 (in Chinese)
张延松,张宇,王珊. 一种基于向量索引的内存OLAP星型连接加速新技术,计算机学报,2019,42(8):1686-1703

[13] Zhang Yan-song, Zhang Yu, Wang Shan. One Method of Data Warehouse Safe OLAP on in-memory Cloud Computing Platform. CN Patent 201610016726.3. ,2019
张延松,张宇,王珊. 一种内存云计算平台上的数据仓库安全OLAP方法.中国授权, 201610016726.3.,2019

[14] Lee R, Zhou M, Li C, Hu S, Teng J, Li D, Zhang X. The Art of Balance: A RateupDB Experience of Building a CPU/GPU Hybrid Database Product. Proc. VLDB Endow. 14(12): 2999-3013 (2021)

ZHANG Yan-Song, Ph. D., associated processor. His current research interests include data warehouse, main-memory database, OLAP and new hardware databases.

HAN Rui-Chen, Ph. D. candidate. His research interests include In-memory databases, hardware-conscious optimizations.

LIU Zhuan, master. His research interests include GPU databases and new hardware database optimizations.

ZHANG Yu, Ph. D., senior engineer. Her research interests include data warehouse, GPU database.

Background

As more and more new storage and compute techniques are involved in database implementation and optimization, how to make database adaptive to different hardware configurations and workloads are challenging issues. The emerging trend of separation of storage and compute and pushdown compute techniques are adopted for distributed storage infrastructure, while this method can be further explored for in-memory databases with more considerations for schema, workloads and data distributions. Another trend is hybrid OLTP and OLAP workloads for in-memory databases, most of the solutions focus on hybrid row-wise storage model and column-wise storage model, hybrid CPU-GPU or DRAM-PM platforms for OLTP and OLAP workloads, and concurrent control between OLTP and OLAP data, etc. Beyond these solutions, we can further explore how to design a unified framework with schema perspective.

For OLAP workloads, the data are designed as multidimensional dataset, the meta data describe the dimensions, hierarchies and other attributes, the n-D space is

filled with values which are described by meta data. For relational database, the well-designed dimension tables are meta data for multidimensional dataset with semantic information, and the fact tables are only values without semantic descriptions. In this paper, we separate the meta data and values from OLAP dataset, the meta data and workloads on them (update, query rewriting, query schedule, …) are defined as management layer, the non-semantic values on fact tables and compute on them are defined as storage layer which can further divided into compute layer and storage layer. With the separation of management, compute and storage, we can independently design the storage model and computing model for each layer, how to deploy workload with specified hardware, and how to combine OLTP and OLAP workloads in the unified framework.

In this paper, we discuss the management-compute-storage framework for nowadays in-memory databases. By separation of meta data management workloads from the OLAP workloads, we can move computing workloads from database engine with more flexible external storage engine or platforms to independently match the optimal configurations. The compute on big fact table is accelerated by GPU and CPU, or external Arrow with in-storage computing in experiments, the experimental results show that the layered computing framework can achieve high performance for OLAP benchmark.

This work is supported by National Natural Science Foundation of China (61732014, 61772533) and Natural Science Foundation of China Beijing (4192066). The target of research is to exploit how to design a flexible framework for new storage and hardware era. We combine the data and compute for specified hardware for in-storage compute to achieve higher performance and locality, the management-compute-storage framework enables the independent compute for different layers.