软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2024,35(1):405−429 [doi: 10.13328/j.cnki.jos.006916]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

支持混合事务和分析处理的数据库管理系统综述*

王嵩立[1,2], 荆一楠[1,2], 何震瀛[1,2], 张 凯[1,2], 王晓阳[1,2]

[1](上海市数据科学重点实验室, 上海 200433)

[2](复旦大学 计算机科学技术学院, 上海 200438)

通信作者: 荆一楠, E-mail: jingyn@fudan.edu.cn

摘 要: 数据库管理系统根据应用场景分为事务型 (OLTP) 系统和分析型 (OLAP) 系统. 随着实时数据分析需求增长, OLTP 任务和 OLAP 任务混合的场景越来越普遍, 业界开始重视支持混合事务和分析处理 (HTAP) 的数据库管理系统. 这种 HTAP 数据库系统除了需要满足高性能的事务处理外, 还需要满足实时分析对数据新鲜度的要求. 因此, 对数据库系统的设计与实现提出了新的挑战. 近年来, 在工业界和学术界涌现了一批架构多样、技术各异的原型和产品. 综述 HTAP 数据库的背景和发展现状, 并且从存储和计算的角度对现阶段的 HTAP 数据库进行分类. 在此基础上, 按照从下往上的顺序分别总结 HTAP 系统在存储和计算方面采用的关键技术. 在此框架下介绍各类系统的设计思想、优劣势以及适用的场景. 此外, 结合 HTAP 数据库的评测基准和指标, 分析各类 HTAP 数据库的设计与其呈现出的性能与数据新鲜度的关联. 最后, 结合云计算、人工智能和新硬件技术为 HTAP 数据库的未来研究和发展提供思路.

关键词: 数据库系统; 混合事务和分析处理; 查询处理; 数据库存储; 存储模型; 事务处理

中图法分类号: TP311

中文引用格式: 王嵩立, 荆一楠, 何震瀛, 张凯, 王晓阳. 支持混合事务和分析处理的数据库管理系统综述. 软件学报, 2024, 35(1): 405–429. http://www.jos.org.cn/1000-9825/6916.htm

英文引用格式: Wang SL, Jing YN, He ZY, Zhang K, Wang XY. Survey on Database Management Systems Supporting HTAP. Ruan Jian Xue Bao/Journal of Software, 2024, 35(1): 405–429 (in Chinese). http://www.jos.org.cn/1000-9825/6916.htm

Survey on Database Management Systems Supporting HTAP

WANG Song-Li[1,2], JING Yi-Nan[1,2], HE Zhen-Ying[1,2], ZHANG Kai[1,2], WANG Xiao-Yang[1,2]

[1](Shanghai Key Laboratory of Data Science, Shanghai 200433, China)

[2](School of Computer Science, Fudan University, Shanghai 200438, China)

Abstract: Database management systems are divided into transactional (OLTP) systems and analytical (OLAP) systems according to application scenarios. With the growing demand for real-time data analysis and the increasing popularity of mixed OLTP and OLAP tasks, the industry has begun to focus on database management systems that support hybrid transactional/analytical processing (HTAP). An HTAP database system not only needs to meet the requirements of high-performance transaction processing but also supports real-time analysis for data freshness. Therefore, it poses new challenges to the design and implementation of database systems. In recent years, some prototypes and products with diverse architectures and technologies have emerged in industry and academia. This study reviews the background and development status of HTAP databases and classifies current HTAP databases from the perspective of storage and computing. On this basis, this study summarizes the key technologies used in the storage and computing of HTAP systems from bottom to top. Under this framework, the design ideas, advantages and disadvantages, and applicable scenarios of various systems are introduced. In addition, according to the evaluation benchmarks and metrics of HTAP databases, this study also analyzes the relationship between the

* 基金项目: 国家自然科学基金 (62072113)
收稿时间: 2022-07-22; 修改时间: 2022-11-01, 2022-12-11; 采用时间: 2023-02-06; jos 在线出版时间: 2023-08-09
CNKI 网络首发时间: 2023-08-10

design of various HTAP databases and their performance as well as data freshness. Finally, this study combines cloud computing, artificial intelligence, and new hardware technologies to provide ideas for future research and development of HTAP databases.

Key words: database system; hybrid transactional/analytical processing (HTAP); query processing; database storage; storage model; transactional processing

1 引 言

数据库管理系统根据应用场景可以分为两类: 事务型系统和分析型系统. 事务型系统主要处理的任务被称为在线事务处理 (online transaction processing, OLTP). OLTP 一般适用于交易类场景, 以写为主、以读为辅. 写强调事务的一致性, 而读以点查询 (point query) 为主. 这种查询一般访问数据量小, 以单条记录或少量记录为主; 而且强调低时延和高吞吐, 以支持高并发的操作. 分析型系统, 即传统意义上的数据仓库, 常常作为商业智能 (business intelligence) 的重要支撑, 帮助数据分析师进行大规模数据分析处理, 产生各种报表和报告, 辅助决策者总结过去的经验并预测未来的趋势. 分析型系统更关注数据在高层次的聚集整合, 主要负责再造、合成、管理数据, 也被称作在线分析处理 (online analytical processing, OLAP). OLAP 以读为主、写少甚至没有. 读以范围查询 (range query) 伴随各种聚集 (aggregation) 操作为主, 这类查询往往涉及大量数据的访问, 所以读开销大, 时延一般较长. 即便有写操作, 也主要以追加 (append) 为主.

进入大数据时代, 数据的产生方式、规模和速度都发生了巨大的变化. 与此同时, 随着云计算等技术的发展, 数据使用门槛降低, 实时数据分析需求猛增成为新常态. 例如在当今的营销场景中, 从消费者到卖家, 各个销售环节的参与者都有数据分析的需求. 在线下, 零售店的调研员可以通过手持终端设备随时随地了解产品销售情况和预测销售趋势, 进而根据数据做出相应决策; 在线上, 电商卖家会随时通过后台数据监测店铺营业情况, 分析交易数据, 进而调整营销策略. 在这种情况下, 个人既是数据的生产者, 同时也是消费者, 驱动 OLTP 场景与 OLAP 场景互相渗透, 事务型系统和分析型系统之间的界限变得模糊. 更进一步的, 随着“直播带货”成为热门行业, 在直播间, 商家需要瞬时对消费者的下单、修改订单等操作做出反应, 并实时调整直播间的商品信息. 这说明了在数据分析需求增长的同时, 数据分析对实时性、数据新鲜度的要求也越来越高, 从新鲜的数据中获取即时的数据洞察 (data insight) 对于决策支持、分析能力的提升是显著的[1]. 另一个典型的例子是在金融市场, 基金经理需要随时根据客户资产净值、交易频次变化、金融产品销售情况等一系列数据服务, 来有针对性地进行营销决策. 这些决定必须要在几分钟甚至几秒钟内完成, 因此快速、即时的数据分析逐渐成为行业刚需. 除此之外, 在欺诈侦测、健康管理、个性化推荐等场景下也都存在 OLTP 和 OLAP 高度混合的情况.

在上述需求下, 借助 ETL 技术[2]定期从在线事务型系统将数据全部拷贝转移到数据仓库的传统做法已经难以满足高时效数据分析的要求. 首先, ETL 过程需要经过复杂的软硬件操作, 数据传输和更新慢, 且消耗大量的网络带宽; 其次, 为了不影响事务系统的吞吐量, ETL 过程通常在闲时进行, 这会进一步加大当前分析所用的数据与最近更新的数据之间的时间延迟. 在需求的驱动下, 一个能够高性能处理事务负载同时兼顾实时数据分析的数据库管理系统的重要性不言而喻[3]. Gartner 公司[4]最早在 2014 年提出了混合事务和分析处理 (hybrid transactional/analytical processing, HTAP) 这一概念, 将其定义为在一个系统上同时处理事务和实时分析, 这是一种打破了 OLTP 数据库和 OLAP 型数据仓库之间壁垒的全新技术, 它的出现使得更可靠和实时的决策变得可能. Gartner 在后续报告中也将 HTAP 称为增广的事务 (augmented transactions)[5], 其他咨询公司如 451 Group 将其描述为“混合操作型和分析型处理 (hybrid operational and analytical processing, HOAP)”. 综上所述, 狭义 HTAP 的定义是以事务处理为主, 并增强对分析任务的支持能力. 在最新的实践中, 数据库厂商和学术界将其含义进一步拓宽, 对于分析和事务处理的一体化解决方案均可称为广义的 HTAP, 这也是本文讨论的重点.

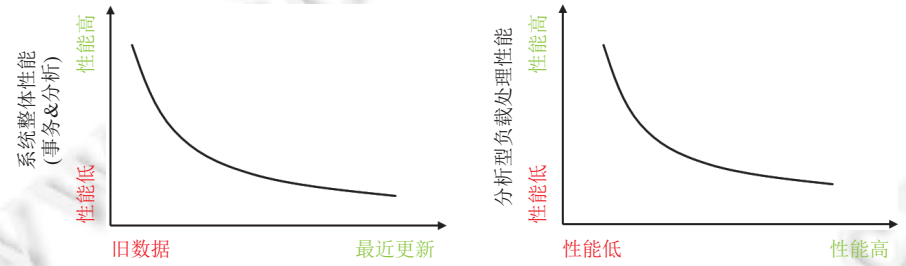
本文包括 5 个部分, 首先从设计目标和发展概述介绍了 HTAP 的总体背景, 其次提出了一种从设计架构角度对 HTAP 系统分类的方法, 并据此对学术界和产业界的主流 HTAP 系统进行了分类. 本文随后详细分析了 HTAP 数据库系统的核心关键技术, 并介绍了几种较为典型的评估 HTAP 数据库性能的测试基准. 最后, 本文就 HTAP 系统领域的未来研究方向进行了展望.

2 背 景

2.1 HTAP 数据库系统的主要设计目标

HTAP 数据库系统希望能在高效处理事务的基础上提供更即时的分析能力. 试想在金融行业, 如果为了追求数据分析能力而降低日常交易系统的吞吐量, 那么交易系统阻塞造成的后果将是无法承受的[4]. 因此, HTAP 的两大核心要素是事务处理的“高性能”和数据分析的“实时性”. 支持 HTAP 的数据库管理系统的主要设计目标即是如何能够在提升实时数据分析能力的同时尽量减少对事务处理的影响, 以其在两者之间取得恰当的平衡. 具体来说, 衡量一个 HTAP 数据库系统的主要指标是 OLTP 负载的吞吐量和分析型系统的数据新鲜度. 数据新鲜度通常可以通过比较事务型系统和分析型系统的数据重合率, 或者数据同步的时间间隔[6]等来进行评价.

HTAP 要求在数据产生后马上就可以进入分析场景, 其中面临的最大挑战是如何把 OLTP 和 OLAP 两类工作负载更好放在一个系统上运行. 这本身是一对矛盾的需求: OLTP 负载以写为主; OLAP 负载以读为主, 经常需要执行全表扫描, 并且在扫描的基础上做统计、聚合等操作. 这种情况下, 分析任务常需要占据大量计算资源, 使交易型事务吞吐量严重下降. Psaroudakis 等人[7]的研究表明, 一个系统处理事务和分析的资源是相互冲突的, 相互抢占硬件资源会导致系统整体的吞吐量都受到影响. 在 Sirin 等人[8]的研究中, 3 个被试数据库系统在执行 HTAP 任务时, OLTP 负载的吞吐量最多可下降 42%. 而且, 若新数据的生成速度快于 HTAP 系统同步数据的速度, 则分析型查询的执行时间将成倍增加. 上述研究表明, 为了确保分析所用的数据新鲜度, 系统需要更频繁地进行同步任务, 这会占用系统资源, 导致整体性能下降; 同时随着分析能力消耗的资源增大, 事务处理的吞吐量会相应下降, 如图 1 所示.



(a) 分析能力 (b) 处理能力

图 1 事务处理能力和分析能力的权衡

概括地说, HTAP 的优化问题是计算和存储资源的分配问题, 需要解决事务和分析处理相互抢占资源的矛盾、数据新鲜度和系统性能的矛盾. 针对上述问题, 一种直观的做法是延续 ETL 的思想, 在物理层面将事务系统和分析系统分离. 这样做的好处是避免了对事务系统的侵入, 从而减少对事务性能的影响; 但坏处也很明显, 即数据同步会产生额外的时间开销和存储冗余, 影响分析所用数据的新鲜度. 另一种做法是完全摒弃 ETL 过程, 基于同一个底层存储, 进行逻辑隔离[4]. 这样的做法最大限度地保证了数据新鲜度, 却需要建立更复杂的系统来对系统资源进行管理. 如何权衡各种做法的利弊, 如何通过系统优化来尽量兼顾两方面的要求, 是近年来该领域学术研究和工业实践的主要课题.

2.2 HTAP 数据库的发展概述

在 HTAP 的概念提出以前, 事务型系统和分析型系统相对独立, 多家软件巨头都推出了数据库产品, 如微软的 SQL Server[9]、Oracle Database[10]、IBM 的 DB2[11]等, 其均属于事务型数据库, 旨在满足企业的数据管理及日常业务的可靠、高效执行, 数据存储通常使用 NSM (*n*-ary storage model), 即行存储模型. 后来随着数据分析需求增长, 产生了一些分析型数据库, 如 MonetDB[12], Vertica[13]等, 其主要面向数据分析需求, 针对高速聚合、整合不同来源数据而进行优化, 通常使用 DSM (decomposition storage model) 存储, 也称为列存储模型. 在相当长的时间里,

由于这两类系统在系统架构、底层存储、查询处理和优化逻辑上大相径庭, 且鲜有真正混合 OLTP 和 OLAP 任务的业务场景, 所以几乎没有厂商尝试将两类业务在同一系统中实现.

随着业界逐渐认识到 HTAP 能力的重要性, 并且借助存储和计算技术的发展和成熟, HTAP 登上了历史舞台, 迈入黄金发展期. 这些技术包括列存储[12]、LSM 树 (log-structured merge-tree)[14]、向量查询引擎等软件技术; 以及固态硬盘、高性能 GPU 在硬件平台的广泛使用等.

学术界对 HTAP 的研究起步较早, 2011 年德国慕尼黑大学开发的 HyPer[15]是一款基于主存的, 兼顾事务和分析处理的数据库原型系统, 其设计目标就是构建一个能够处理混合负载的高效系统. 该系统在数年间不断演进, 如后续推出的支持多版本并发控制 (multi-version concurrency control, MVCC) 的版本[16], 在学界和产业界取得了较高的认可度. 洛桑理工学院的 Caldera 系统[1]借助硬件异质性新型架构 H^2TAP, 充分发挥新型硬件能力且专攻混合事务和分析处理. 卡耐基梅隆大学的数据库小组也在自研的 Peloton 数据库[17]上设计了混合行列存储系统以提升 HTAP 性能, 该系统可以感知工作负载来自适应地调节底层存储以加速对 OLAP/OLTP 混合负载的处理能力.

在产业界, 数据库厂商主要在两种道路上探索 HTAP 的解决方案.

(1) 改良方案, 将分析能力作为事务的扩展, 让事务系统适应分析任务. 2010 年, Microsoft SQL Server 11 (代号 Denali)[18]中引入了基于聚集索引的列存储功能, 极大提升了分析性能; 几乎同时期, Oracle 12c 加入了内存中组件 (in-memory component), 使得 Oracle 数据库具有了双模式数据存放方式, 从而支持 HTAP; 内存数据库 SingleStore (前身为 MemSQL) 在原有的事务型存储的基础上增加了用于分析的列存储系统[19], 以支持对大量历史数据查询的分析业务. 这些改良方法使得原本事务型的数据库向 HTAP 进行了有效的靠拢, 可以说是一种通过改进原有系统的“适应性策略”. 其优点是在原有系统上进行扩展成本相对较低, 用户的迁移成本也低; 部分系统上分析能力是可选功能, 用户可以根据需求进行定制化选择. 这种方案的缺陷也很明显, 那就是原本系统的设计和架构限制了方案的选择, 如 SQL Server 的数据同步更新依然依赖于内存中的行存系统作为缓存.

(2) 改革现有的数据库系统架构, 在一个全新的系统上同时兼容事务和分析. Tableau 公司在收购 HyPer 系统后, 使其成为了一款面向 HTAP 的商业化产品. IBM 在 DB2 的设计基础上研制了基于 PAX 存储技术的 HTAP 原型系统 Wildfire[20]. 与 HyPer 一样, 这类数据库在单一系统上通过逻辑隔离存储和计算资源来支持混合负载. OceanBase[21]通过基于 LSM 树[14]的存储系统将基线数据和增量数据分别储存在磁盘和内存中, 通过多节点分布式部署来支持 HTAP 查询. Greenplum[22]最初是基于 PostgreSQL 开发的侧重 OLAP 的数据库系统. 经过多年的发展, Greenplum 利用大规模并行处理 (massively parallel processing, MPP) 架构的优势, 加入多级分区、行列混合存储等新技术以满足 HTAP 场景. 不同于在一个系统上进行逻辑隔离, Google 在 2020 年提出了将松耦合的分析系统和事务型系统以组件的形式结合在一起的 F1 lightning[23]方案, 该模式最小化了两种业务之间的相互影响, 将优化重点放在数据同步和传输上. PingCap 公司延续了 F1 lightning 的设计优势, 开发了 TiDB[24], 它由事务型系统 TiKV 和分析型系统 TiFlash 组成完整的 HTAP 功能, 利用 RAFT 协议[25]同步数据, 使用 LSM 树提升更新写入性能. 这类分离架构在物理层面使得事务系统和分析系统解耦, 更好地保证了隔离性, 但是增加了数据一致性保证的压力, 也一定程度上牺牲了分析型系统的数据新鲜度. 用全新架构替代传统架构的方案从硬件层到软件层的设计都可以更灵活, 更适应现代软硬件环境, 但相应的, 高额的开发成本、客户对其性能的不确定性、学习与部署成本都是不容忽视的.

HTAP 数据库发展到现在, 该领域的一个新的关注点是如何建立完整的技术生态. 搭建一个良好的开源生态有利于安全可靠保障、用户反馈收集与经验积累、开发和运维工具链构建、用户与产业链培育等, 这些都是支撑 HTAP 数据库发展的核心要素. 现阶段 HTAP 数据库生态主要靠既有开源数据库生态支撑, 包括分布式架构 Hadoop[26]和 Spark[27]生态下的 HBase[28], 以及历史悠久的 MySQL、PostgreSQL 等开源生态. 近年来国产的 HTAP 数据库企业如 PingCAP、阿里云等国内厂商已经开始通过开放源码来构建多方参与的社区. 同时, 数据库厂商也广泛地通过项目形式与高校和研究所合作, 将成果在开源社区共享, 在这个基础上建立信任, 共同成长, 从而快速建立属于 HTAP 数据库的生态.

3 HTAP 数据库系统分类

3.1 分类依据

目前拥有 HTAP 能力的数据库管理系统采用了多种多样的技术路线, 其主要原因是不同类别的 HTAP 系统针对不同的应用: 如一些交易场景需要首先保证事务处理, 而另一些决策应用则将分析处理置于优先地位. 除此之外, 还会根据系统本身考虑可用性、可扩展性、系统性能和数据新鲜度的要求进行适应性设计. 近年来对于 HTAP 系统的综述[29,30]大多是从存储角度出发对 HTAP 数据库系统进行分类, 重点关注 HTAP 数据库系统的设计架构和实现效果, 但对不同设计架构下计算与存储的结合以及对应关键技术分析并不全面. 因为计算和存储的设计与结合方式对系统整体的性能影响是显著的[31]. 所以本文从计算、存储两个视角将其分为 4 大类, 并试图总结出一些实践中行而有效的“共同道路”, 如图 2 所示.

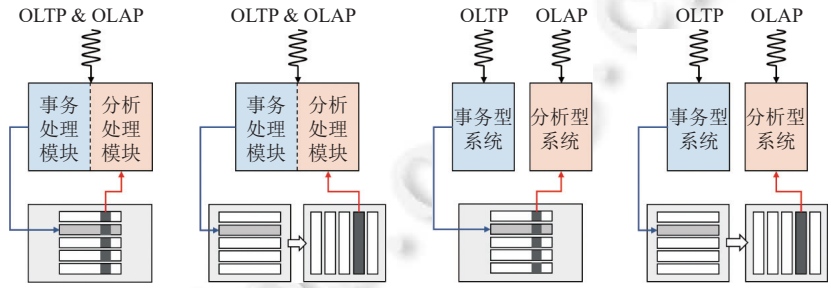


图 2 HTAP 数据库系统分类

从计算的角度考虑一个 HTAP 系统, 主要关注 OLTP 和 OLAP 负载是如何被处理调度的、计算资源是如何分配给不同的工作负载的. 现有的 HTAP 系统在计算架构上可以分为“单系统”和“双系统”两类. 单系统方案由一个大而全的系统同时处理两类负载, 这种方案的优点是数据可见度高, 因为不需要做数据的同步导入导出或数据转换. 另一种是双系统, 这类系统是由异构的两类子系统组成, 它们分别处理 OLTP 和 OLAP 负载, 由调度组件将负载分配到对应的系统进行处理. 其主要优势是通过系统解耦将设计关注点分离, 异构的事务系统和分析系统可以分别有针对性地优化自己的领域. 同时, 这种结构也提高了系统整体的可扩展性, 如在 F1 database[32]中, 就可以将其分析型系统 F1 lightning[23]与多种事务系统进行组合; 在 TiDB 中, 事务系统 TiKV 和分析系统 TiFlash[24]由 PD 节点统一调度协作处理 HTAP 事务; 还有 IBM Wildfire[20], 通过有状态和无状态工作节点组合运行, 其分别对应事务处理和分析负载.

另一个角度是存储模型. 存储模型主要考虑数据如何在存储介质 (硬盘、主存) 中组织, 目前有两大方案: “单拷贝”存储和“双拷贝”存储. 单拷贝存储指的是 OLAP 查询和 OLTP 查询在一份数据上执行, 通过快照机制来做隔离. 双拷贝存储指的是 OLTP 查询和 OLAP 查询在两份独立数据上执行, 借助 ETL 或类 ETL 的方式实现数据同步. 在双拷贝系统中存在更大量的数据传输和同步需求, 其硬盘读写和网络传输的代价是高昂的, 因此一些硬件技术对性能提升有非常重要的驱动作用. 在存储介质方面, 固态硬盘 (SSD) 能够提供非常高的 IO 吞吐量. 这一硬件基础能力的提升, 为解决 OLTP 系统和 OLAP 系统存储结构差异导致的数据同步延迟问题做好了准备. 除此之外, 对于采用 MPP 模式和分布式架构的系统, 随着并发事务增多, 网络传输开销是不容忽视的. 在这两大分类下, 又根据存储技术的具体细节而有所差异. 需要注意的一点是, 在一些系统上也会存在一份数据的多个副本, 如 MVCC 维护的多个版本, 这并不能称为“双拷贝系统”, 因为这些副本并不是专门为了分离事务和分析操作而产生的. 本文中的“双拷贝”特指专门为了将事务和分析操作分离而从一份数据产生的多个副本拷贝.

3.2 系统分类

根据上述两个角度, 我们可以将产业界和学术界的 HTAP 数据库分为 4 大类系统, 分别是: 单系统单拷贝、单系统双拷贝、双系统单拷贝、双系统双拷贝, 如表 1 所示.

表 1 HTAP 数据库分类

从存储视角分类	从计算视角分类	
	单系统	双系统
单拷贝	HyPer[15,16], SAP HANA[33], SingleStore (MemSQL)[19], Peloton[17], OceanBase[21], Caldera[1], HBase[28], Greenplum[22]	IBM Wildfire[20]
双拷贝	BatchDB[34], Microsoft SQL Server[18], Oracle Dual-Format[35]	F1 lightning[23], TiDB[24], IBM IDAA[36]

(1) 单系统单拷贝

最主流的 HTAP 数据库类型是单系统单拷贝, 这类数据库用单一系统同时处理事务和分析查询, 确保了数据的新鲜度, 但处理两种负载时相互抢占资源, 对性能的影响较大. 为了实现事务处理和分析查询并行执行, 系统在处理 OLAP 查询的时候通常产生并维护一个临时快照以供执行.

Caldera[1]和 HyPer 的早期版本[15]利用操作系统的 CoW (copy-on-write) 机制[37]创建快照. HyPer 完全使用内存, 从而能够以单线程串行执行事务, 这种做法消除了事务层级并发控制的成本, 并且避免了多线程处理过程中上下文切换、索引维护和加解锁等成本. 当执行 OLAP 查询时, 系统为该进程创建一致性快照来执行分析查询. 实现上, HyPer 使用 UNIX fork (基于 CoW) 在分析查询到达时启动一个新的子进程, 从而为分析进程提供对新数据的即时访问. Caldera 管理快照的方式与 HyPer 类似, 并使用硬件加速, 将分析事务调度到 GPU 上执行, 以最大化异构处理器的优势.

SingleStore、Peloton 以及 Greenplum 使用 MVCC 对不同版本的数据进行快照隔离, 在细粒度逻辑分片上执行查询.

SAP HANA 和 OceanBase 使用 Delta-Versioning 技术[21], OLTP 事务修改的新数据先存在 Delta 中, 然后合并到主存储区, 使用类似 MVCC 版本控制的方法给 Delta 存储进行快照隔离. Delta 存储区更适合 OLTP 而主存储区更适合 OLAP. HBase 也采用这一思想, 但实现上略有不同: 它在不同的节点上访问 HDFS[38], 数据更新首先在分区本地完成, 然后再定期合并到共享存储中.

整体而言, 快照机制最大程度保证了分析查询使用数据的新鲜度, 代价是 OLTP/OLAP 任务不同程度的性能损失. 在 CoW 机制中, 当事务引擎更新记录时, 它必须首先进行整页拷贝, 是牺牲 OLTP 性能换取分析性能. 而在 MVCC 机制下, 处理 OLAP 查询时必须遍历各个版本, 这是增加 OLAP 任务随机内存访问次数来保证 OLTP 的性能. Delta-Versioning 是对 OLTP 和 OLAP 性能的折衷, 对 OLTP 保证了读取最新数据的性能, 而对 OLAP 保证了访问已经合并到主存储区的数据的性能.

(2) 单系统双拷贝

在单系统双拷贝类型的 HTAP 数据库中, 事务和分析查询运行在相互独立的数据上. 微软的 SQL Server 和 Oracle 系统[35]就采用了这一架构. 双拷贝存储使用 ETL 同步代替快照策略, 从事务存储区传输同步数据到分析存储区. 这种做法相比单系统单拷贝, 提高了存储的隔离程度, 却牺牲了分析性能. 这是因为每次处理 OLAP 查询时都需要从事务快照抽取数据. 另一个学术研究型内存数据库 BatchDB[34]也采用了单系统双拷贝的思路, 包含主次副本, 主副本为 OLTP 负载服务, 次副本为 OLAP 负载服务. 除了单机部署外, BatchDB 也支持利用 NUMA 技术[39]分布式部署数据副本, 分析型系统定期从事务系统提取最新版本的快照进行同步.

(3) 双系统单拷贝

双系统单拷贝类型的 HTAP 数据库有 IBM 的 Wildfire[20]原型, 从技术细节来看, 该系统为多级单拷贝存储. 作为一个分布式系统, 其节点分为两类: 有状态节点和无状态节点. 有状态的节点处理 OLTP 请求, 无状态的节点处理 OLAP 请求. 和 HBase 类似, OLTP 的数据不会直接写入到共享文件系统里, 而是写入到有状态节点组成的局

部集群, 按照表分片的模式进行数据的快速写入. 有状态节点会定期将数据导入到共享文件系统里, 主要供 OLAP 查询使用. 分析型系统继承自 DB2 的 BLU 系统. 分区同步策略注定会造成未同步的数据延迟, 且这种时延不容小觑, 所以双系统双拷贝存储模式开始受到学界和业界的关注.

(4) 双系统双拷贝

双系统双拷贝的一个经典实现是 Google 的 F1 lightning 系统. F1 lighting 是一个松耦合的组件, 作为 F1 关系型数据库由 OLTP 转向 HTAP 的解决方案. Google 采用双系统双拷贝设计的宗旨是尽量减少或避免对事务型系统进行改动, 这种方案建立在 OLTP 数据库已经暴露了一个相对完备的更改回放 (change replay) 接口上. Lightning 的工作模式由 3 大部分构成: (1) F1 query 查询层, 是 F1 系统下的分布式查询系统; (2) 一个事务型系统, 如 F1 DB; (3) F1 lightning 维护分析查询所需数据的存储. 在 F1 lightning 发布后, 这样的双拷贝存储模式极大启发了其他的数据库厂家. PingCap 公司的 TiDB 正是沿用了其设计优势, 通过组合事务系统 TiKV 和分析型列存储系统 TiFlash 来增强其 HTAP 能力. IBM 公司最初也为 DB2 开发了一个松耦合的 HTAP 解决方案 IDAA (IBM Db2 analytics accelerator)[36], 将数据仓库挂载到事务型系统, 通过轻量级的集成同步方案优化同步延迟问题. IDAA 把读取数据、数据处理、转存等操作从数据源端转移到传输目的端执行, 而目的端能够原生地识别从 DB2 里传输过来的数据.

综上所述, 4 种方案各有侧重和优劣. 系统耦合程度越高, 数据的新鲜度就越高, 但在版本控制、资源调度上的额外开销导致的性能降低就越明显; 系统解耦程度越高, 隔离性越好, 资源相对独立可以保证事务和分析的性能, 但是也由于高昂的数据传输成本牺牲了分析用数据的新鲜度.

4 HTAP 数据库系统核心关键技术

按照上文对 HTAP 的分类角度, 本节分别从存储和计算两个角度介绍目前工业界和学术界在设计实现 HTAP 系统时采用的关键技术.

4.1 存储部分的主要技术

4.1.1 存储模型

存储模型指的是表数据在存储时的组织顺序, 基本分类有行存储模型和列存储模型两种, 而根据数据分区布局模式的不同, 又衍生出更为复杂的模型, 如图 3 所示.

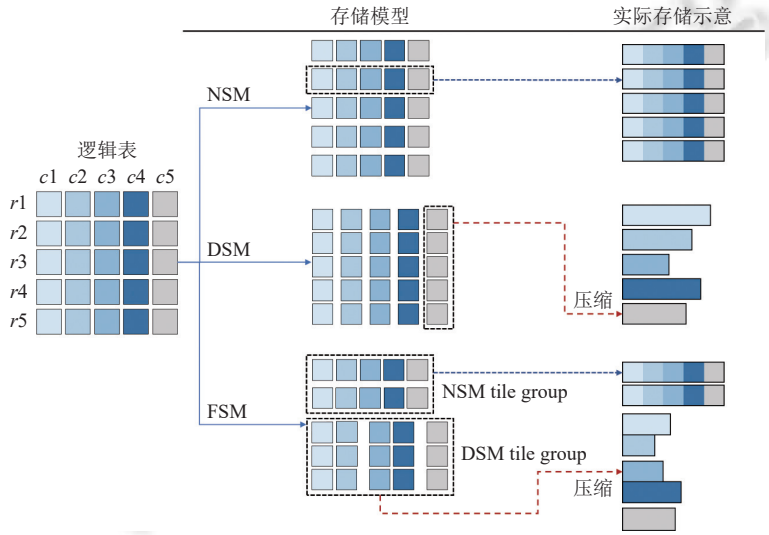


图 3 3 种存储模型

行存储模型 (*n*-ary storage model, NSM)[17]是最早被提出且应用最广泛的. 将表的每一行连续存储, 通过一次访问可以读取到一个元组 (tuple) 所包含的全部属性. 行存储更适合增删改等以元组为单位的操作, 对事务的支持较好, 但并不适合针对单一属性的大范围查询或者是聚合操作等 OLAP 操作. 同时, 读取过程中的大面积加锁也会严重降低数据库的并发性能.

列存储模型 (decomposition storage model, DSM)[40]以列为基础存储单位, 将表中的不同属性集中存储, 可以直接读取全部元组同一属性的值, 这种存储模型更适合 OLAP 查询. 同时, 每个属性可以应用不同的压缩算法以减少存储开销, 在单个属性上排序和索引也可以更好地加速查询. 在提高并行度方面, 其整齐的数据格式更有利于数据库完成向量化操作. 列存储模型也存在弊端, 由于一列数据集中存放, 对小规模的数据进行读取时琐碎的跨多列拼接会严重降低性能.

结合两种存储模型的优势, CMU 提出一种所谓灵活存储模型 (flexible storage model, FSM)[17]. 综合考虑行存储模型和列存储模型各自的优缺点, FSM 模型走了一种更灵活的“中间道路”, 他将属性 (列) 分组, 一个元组因此被划分为若干个片 (tile), 每个片包含多个属性. 其存储的逻辑单元是片的组合, 称为片组 (tile group). 且每个片组可以选择以行式或列式存储, 相互独立.

除了基本的存储模型, 现代数据库系统会在其基础上对数据表进行更细粒度的分区 (partitioning), 产生不同的数据布局 (data layout), 进而衍生出更复杂的行列混合存储模型. 如图 4 所示, 数据分区方法可以概括为水平分区、垂直分区、多层分区、不规则分区 4 种. 在一个工作负载中, 每条查询的结果在表中是不规则分布的, 图 4 中 $Q1$, $Q2$, $Q3$ 对应逻辑表中不同颜色的区域, 根据分区方案的不同, 对于工作负载的查询性能也会相应有差异.

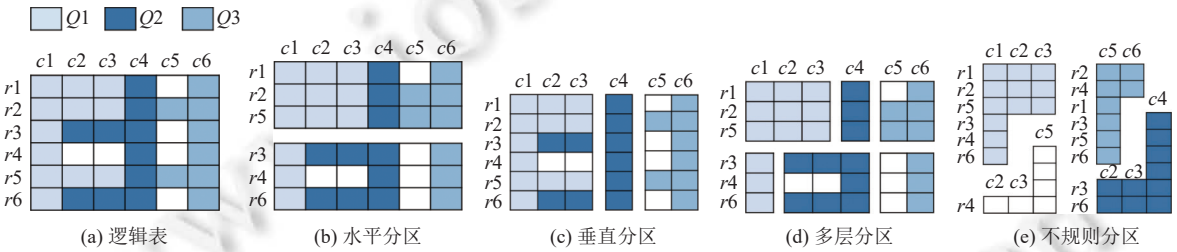


图 4 对逻辑表的不同分区方案

(1) 水平分区 (horizontal partitioning)

水平分区是将表水平方向切分, 通过给分区添加索引, 可以有效减少扫描的元组数量. Ailamaki 等人[41]在 2001 年提出了著名的 PAX (partition attributes across) 模型. PAX 模型针对缓存优化, 将表按缓存大小分为多个页进行存储. 每一页中存有一定行数的数据, 在该页内使用列式存储, 同时针对可变长度的数据结构根据缓存对齐进行优化. 对数据的读取修改操作只会影响 PAX 所在的有限个页, 而不是整列数据. PAX 模型一定程度上弥补了列存储对缓存的不友好以及插入数据时写放大的问题, 并具有更好的空间局部性.

基于 PAX 的列存储模型和水平分区方法, 在 Hadoop 生态下, Cloudera 与 Twitter 推出了 Parquet[42]. Parquet 格式将数据水平切分为多个行组 (row group), 每个行组是部分列的集合. 在单个行组内部, 数据按列存放, 称为列块 (column chunk). 由于每个列块的数据类型相同, 可以根据数据类型使用不同的压缩策略. 在分布式架构的支持下, 多个行组可以实现并行操作.

Facebook 在 2011 年发布了同样基于 PAX 的 RCFile[43]. RCFile 在 Hive 之中作为很好的列存储模型被广泛使用, 很好地提升了 Hive 的工作性能. 2013 年, HortonWorks 在 RCFile 的基础之上开发出了 ORCFile[44], 并且成为 Apache 的顶级项目. 其基本思路也是将行组与列存结合, 若干行数据称为一个 Stripe, 这与 Parquet 的行组如出一辙. Stripe 作为逻辑处理单元, 可由一个任务单独处理. 每个 Stripe 含有 3 个数据域: 索引域、数据域和尾部域. 索引域存储每列的最大值、最小值等信息方便定位, 数据域以列式存储组织数据, 尾部域存储了每列数据在数据域中的位置、编码方式等.

总体而言, 基于 PAX 方案的应用很广泛, 包括 Spanner[45]、F1 lightning[23]以及 IBM Wildfire[20]等数据库系统. TiDB[24]将表水平切分为不同的区域 (region), 并且在多个节点上存有同一区域的若干副本, 这样可以减小扫描开销, 并提升并行处理能力. Greenplum 采用 FSM 的思想, 将表切分为细粒度分区后, 针对每个分区可以指定行存或列存, 以分别支持不同类型的工作负载.

(2) 垂直分区 (vertical partitioning)

垂直分区是将表垂直切分, 每个分区包含一个或多个属性, 即若干列的集合. 这种方案可以有效减少查询处理中读取多余的属性. 同一列中的垂直分区只会被单次 I/O 操作访问一次. H2O 系统[46]将这种切分后的属性集合称为列组, 它可以根据工作负载动态调整存储布局, 用基于规则的策略确定每个垂直分区的大小并选择最佳数据顺序.

(3) 多层分区 (hierarchical partitioning)

多层分区可以视为水平和垂直分区方法的一种结合, 首先对表进行水平切分为组, 然后再将行组进一步进行垂直分区, 最终得到大小不一的矩形块. 这种分区方法 IO 效率更高, 但是由于粒度小, 也存在投影时重构元组开销大的缺点. Peloton 数据库系统基于 FSM 根据数据的冷热分类动态地将行数据转变为列存储[17], 在 OceanBase 的列组混合存储[21]系统中也可以看到该方案的影子.

(4) 不规则分区 (irregular partitioning)

不规则分区是对于多层分区方案进一步的优化, 摒弃了分区总是矩形区域的思想, 是一种最自由的布局. 在 JIGSAW 系统[47]中第一次提出, 该方法通过登山算法将矩形分区通过分裂、合并成不规则形状的分区. 这样的布局对特定负载适配性更高, 但是会存在一定的过拟合以及额外的投影重建难度. 存储顺序和数据布局并没有直接对应关系, 可以通过技术组合适应不同类型的工作负载. 但由于该分区方法对 OLTP 的数据更改操作不友好, 所以目前主要在分析型系统中应用.

新的分布式 HTAP 数据库原型 Proteus[48]实现了自适应存储分区策略. 它可以根据工作负载对数据进行管理, 包括数据分区、调整分区的存储格式或介质、应用存储优化策略等, 使得执行混合负载所需要的延迟最小. 除此之外, Proteus 还针对这套存储系统定制了计算系统, 可以生成独特的物理执行计划, 并充分利用存储感知运算符来高效执行事务.

综上所述, 行存储与列存储在不同的领域有不同的表现, 前者更偏好于 OLTP 任务, 后者更偏向于 OLAP 任务. PAX 模型、FSM 模型等行列混合模型也是在两者之间进行权衡的产物, 并没有一种存储模型是“完美的”. 因此, 目前大量的 HTAP 系统为一份数据分别维护行存与列存副本, 以行存副本支持 OLTP 事务处理, 并借助列存副本支持 OLAP 查询处理.

4.1.2 存储介质及数据同步

数据与文件都需要存储在物理介质之中, 数据从获取到提交 CPU 处理需经历磁盘、内存以及缓存这一路径[49]. 从存储介质看, 一般而言越高效的存储器成本越高. 虽然计算机的运算性能迅猛增长, 但磁盘相对慢的 IO 性能始终是数据库应用的性能瓶颈[50]. HTAP 数据库的数据在存储介质上的分布上主要分为 3 类: 基于磁盘的、基于内存的, 以及组合使用内存与磁盘的混存数据库, 如表 2 所示, 其中单拷贝系统用粗体表示, 其余则是双拷贝系统. 当数据由事务更新时, 变更数据和持久化数据通常被存储在不同的区域 (如分别存在磁盘或内存中), 后续, 变更数据会由不同的策略同步到持久化存储区, 这个过程也被称为变更合并 (delta-merge)[33]. 传统上, 在 TP 系统里更新一行数据的事务时延是毫秒, 甚至微秒级的, 而列式存储因为稀疏索引和压缩、排序等算法, 更新一列数据可能就需要数十毫秒的时间, 更新一行的延迟比行存引擎高一个数量级. 而有了更大内存之后, 列存数据和索引可以全部放置于内存里, 而内存更新速度是纳秒级别的[51]. 在 TP 请求行存更新毫秒级的时延基础上, 列式存储增加的每一列几微秒级别的时间, 对 OLTP 事务整体延时不会带来显著的增加. 即更多内存、更快的 SSD 可以在同时维护行列数据一致性的同时保证 OLTP 系统的低延时.

基于磁盘的数据库系统将内存作为高一级的缓冲区, 主要数据存放在磁盘中. 元数据, 包括数据分区、数据起始地址、索引等信息存入内存以快速定位到磁盘的数据位置[52]. 许多数据库将变更数据缓存到内存中, 以加快对热点数据的读取与修改. 基于磁盘进行存储的设计主要考虑如何提高内存命中率, 减少磁盘 I/O 以降低延迟. 在数

据存放上将经常被访问的数据对象存放在同一个磁盘分区内[53]. 同时控制读写行为, 通过批处理写入[54]等技术获得更高的磁盘 I/O 效率.

表 2 HTAP 数据库的存储介质和数据同步策略

分类	数据库管理系统	数据分布		同步策略
		磁盘	内存	
基于磁盘的	**IBM Wildfire**[20], **Greenplum**[22]	主要存储区	变更数据	定期批量合并
	HBase[28], **OceanBase**[21], F1 lightning[23], TiDB[24]	主要存储区 (LSM树)	增量数据区 (LSM树)	基于日志的变更合并
基于内存的	**HyPer**[15], **Peloton**[17]	持久化数据	主要存储区	无需同步
	SAP HANA[33]	持久化数据	主要存储区 变更数据区	变更合并 (内存中)
组合内存与磁盘	SQL Server[18]	主要存储区 (冷数据)	主要存储区 (热数据)	变更合并 (内存-磁盘)
	SingleStore[19]			重拷贝
	Oracle Dual-Format[35]			变更合并(内存-磁盘); 重拷贝

基于磁盘的数据库系统有: (1) IBM Wildfire[20], Greenplum[22]等分布式系统. 在 Wildfire 中新数据首先写进事务私有的集群, 然后定期执行批量合并操作完成数据同步, 合并到共享文件系统中. 其数据以 PAX 块的格式进行存储, 粒度较小. 同时, 由节点到共享存储的合并式结构充分利用了基于磁盘的多级存储思想, 并行的局部更新和合并操作极大提高了 I/O 效率, 减少了各个节点的 I/O 开销和需要处理的数据量, 让系统整体的吞吐量得到提升. 在 Greenplum 中, 节点之间不存在数据共享 (shared-nothing), 设计重点在于利用各种策略将数据分布到不同节点上, 包括哈希、随机等. 其目标是做到数据的均匀分布, 避免出现短板效应, 来最大化并行处理的能力. (2) F1 lightning, HBase, OceanBase, TiDB 等基于 LSM 树的存储系统. 这类系统将增量 (变更) 数据先写入内存, 这样可以充分保证 OLTP 的性能, 然后利用 LSM 树优秀的写性能将变更数据逐步与 LSM 树下层合并. 同步策略采用基于日志的变更合并[14], 读取过程结合内存与磁盘, 在降低对 OLTP 事务性能侵入的同时能够保证数据的新鲜度.

内存数据库[55]相对于磁盘存储读写性能更快. 由于随机读写受数据存储顺序影响较小, 其主要性能瓶颈是数据访问延迟和数据管理算法的处理器消耗. 随机访问内存中的数据会把多个不同的内存块重复的装入内存和处理器之间的多级缓存中, 而这一操作在 HTAP 任务中尤为频繁. 因此在 HTAP 内存数据库系统的存储方式设计上, 不仅要考虑到内存访问数据的方式, 也要考虑到缓存对内存访问的性能影响. 目前比较典型的内存数据库有: (1) HyPer. 其充分利用了内存随机读写的能力, 串行执行事务, 这样就避免了加锁、缓冲区管理、日志管理等操作. 在传统的系统中, 这些操作消耗系统 80% 左右的资源[15]. OLAP 查询为事务的子进程, 在快照上执行. 由于内存的较高性能, 一个事务串行执行的耗时只有微秒级别. HyPer 的一个后续版本[16]使用 MVCC 进行快照隔离, 进一步提升了事务和分析的并行度. (2) SAP HANA. 这是一款比较成熟的基于内存的 HTAP 数据库系统, 它在内存中采用变更-合并的存储结构. 为了适应逐渐增大的数据量, 2019 年 SAP HANA 推出了 NSE (native storage extension)[56]的存储扩增选项. 当内存资源不足时, 可以由用户指定将热点数据页加载到内存中, 而剩余数据则保存到磁盘中. (3) Peloton. 作为 CMU 数据库小组的一个学术型数据库, 使用 FSM 存储模型在内存中进行管理, 通过冷热判别, 将有可能同时访问到的数据保存在同一个分区[17]来减少读取时的额外开销, 提高效率.

组合使用内存和磁盘是一种更为灵活的存储方式, 这类系统的共同特点是内存不再是单纯作为磁盘的缓冲区, 而是由数据库系统统一管理, 内存和磁盘都是可选可控的主要数据存储位置. 对于主要用于分析但不参与或很少参与 OLTP 查询的数据, 可以将它们存放在磁盘之中, 等待分析负载对它们进行读取. 使用这种架构的数据库有: (1) SQL Server. 其支持根据数据的冷热程度, 将热点数据优先存在内存中[18], 以提升事务处理效率. 同步上采用变更合并的策略从内存中将变更数据同步到磁盘. (2) Oracle Dual-Format[35]. 该系统具有双模式数据存放方式,

允许用户指定表数据的优先级, 再由系统根据资源空闲情况加载到内存中. 磁盘中以行存的形式存储, 在内存中以列存进行存储. 同步采用变更合并策略, 当数据变化超过阈值时, 则从磁盘重新拷贝. (3) SingleStore[19]. 其前身是基于内存的数据库, 随着列存储引擎的推出, 成为了组合使用内存和磁盘的 HTAP 数据库. SingleStore 的内存默认使用行存储, 磁盘为列存. 磁盘中的列存储数据服务于 OLAP 任务, 而内存中的行存数据主要用于高性能事务处理, 同步采用重拷贝方式. 内存中, SingleStore 使用 MVCC 和无锁执行组合来强化 OLTP 性能, 仅在同一行发生写-写冲突的情况下才使用锁. 除此之外, SingleStore 还尽可能将单行更新查询优化为简单的原子操作.

4.1.3 存储引擎

存储引擎是数据库底层负责数据管理、存储、压缩等操作的组件, 数据库管理系统使用存储引擎进行创建、查询、更新和删除数据. 不同的存储引擎提供不同的存储机制、索引选择、事务隔离等功能. 在 HTAP 数据库中, 存储引擎的架构和优化对性能影响深远. 目前该领域主流的存储引擎根据数据模型可以分为关系型[57]和非关系型 KV 存储[58]两大类, 如图 5 所示.

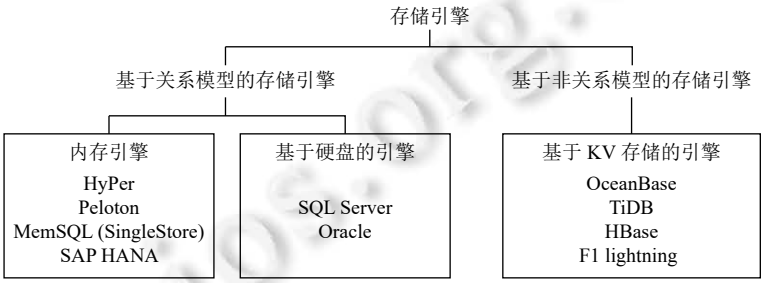


图 5 HTAP 数据库的存储引擎

(1) 基于关系型模型的存储引擎

在 HTAP 数据库系统中, 有相当一部分是由传统事务型数据库演变而来, 所以其底层的存储引擎沿用了关系型数据库存储引擎. 关系型数据库最典型的逻辑数据结构是表以及表之间的联系所组成的一个数据组织. 数据更新采用原地更新的方式, 常采用 B+树等索引结构. 基于关系型模型的引擎由于格式一致, 易于维护; 借助 SQL 语言规范通用的优点, 可用于复杂查询. 但缺点是海量数据、高并发读写下硬盘 I/O 是一个很大的瓶颈. 所以使用关系型模型的 HTAP 系统致力于通过开发内存空间来降低读写延迟.

HyPer, Peloton 和 SAP HANA 等数据库采用全内存引擎, 利用数据冷热程度来规划存储策略. HyPer 存储引擎基于内存中的快照[15], 通过内存管理单元去区分数据的访问热度. 热数据放在内存中的较小页面上; 冷数据则压缩后存储在较大页面上. 这种做法的好处是更新代价比较小, 同时在做 OLAP 查询的时候, 在大页面上会有比较好的性能. Peloton 存储引擎使用 FSM 存储模型在内存中进行管理, 根据数据的冷热程度进行行存储到列存储的转换[17]. 当元组被添加时定义为热数据, 更有可能执行 OLTP 操作; 随着时间的推移, 数据则会逐渐“降温”, 主要被 OLAP 查询访问. 根据这个基本假设, FSM 系统将热数据按行存储, 随着数据块变为冷数据之后, 将一些相关性强的属性以小粒度的分片的形式列式存放在一起. 整体而言, 数据随着时间的推移, 渐渐地从行存转变成列存. SAP HANA 存储引擎使用变更-合并的存储结构分别存储冷热数据, 称为主存储区 (main store) 和变更存储区 (delta store)[33]. 主存储区使用列存储模型, 主要服务于 OLAP 查询; 变更存储区分为两层: L1 层使用行存储模型, 在该区域直接发生数据更改, 存储无压缩的热数据; L2 层采用的是轻量级压缩的列存储, 由 L1 层转换而来. 变更存储区还会定期地执行合并操作, 把数据合并到主存储区里. 由于内存的读写性能高, 在合并操作的过程中生成的变更版本也能用于读写, 数据加锁的行为只是发生在缓冲区切换阶段.

SingleStore 原本是完全基于内存的数据库, 随着其推出了列存储引擎, 也成为了组合使用内存和磁盘的 HTAP 数据库. SingleStore 的内存默认使用行存, 磁盘为列存. 在使用中需要管理员指定表为列存储模式即可开启磁盘存储. 磁盘中的列存储数据服务于 OLAP 任务, 而内存的行存数据主要用于高性能 OLTP.

SQL Server, Oracle 等数据库在传统基于磁盘的存储引擎基础上优化了内存对 HTAP 的支持. SQL Server 集成了基于内存的关系型存储引擎 Hekaton[59]. 该内存引擎采用无锁数据结构和乐观的多版本并发控制技术, 从而实现了非常高的并发和比先前版本 10 倍以上的性能提升. Hekaton 重新设计实现了常驻内存的数据索引以替代传统的 B+树索引, 索引相关的操作不会记录到日志, 日志和检查点机制只用来保证数据的持久性. 在系统需要恢复的时候, Hekaton 会通过最新的检查点和日志来重建表和索引. Oracle 存储引擎在磁盘中以行存储格式存储数据. 在此基础上开辟了 in-memory 空间, 用户可指定部分表数据的优先级, 再由系统根据资源空闲情况加载到内存中. 数据在内存中以列存储格式进行存储[35], 由 1M 池和 64K 池两部分构成. 1M 池用于保存原始数据, 基本的存储单位是内存压缩单元 (in memory compression unit, IMCU); 64K 池用于保存和内存压缩单元相对应的元数据信息, 基本单位是快照元数据单元 (snapshot metadata unit, SMU).

(2) 基于 KV 存储的存储引擎

KV 存储, 也即键值存储, 是一种非关系型存储模型, OceanBase, TiDB, HBase 等系统均采用这一模型. 这种存储模型通常采用 LSM 树结构进行存储, 使用日志更新, 将内存作为写缓冲区, 逐层向下合并到持久存储中. 其优点是日志更新无锁, 不会引入并发问题, 能够保证高效写入, 并且没有空间碎片[14]. 缺点是 LSM 树结构在扫描时需要读取多层的数据, 导致读路径变长, 称为读放大问题. 除此之外, LSM 树需要进行归并 (compaction) 操作, 该操作会导致大量数据的重新写入, 也称为写放大问题. 目前基于 KV 存储的引擎都致力于解决上述由 LSM 树带来的读写放大问题.

HBase 是在 Hadoop 生态下的数据库系统, 所以存储引擎基于 HDFS 文件系统工作, 使用类似 Parquet 的基于列簇的列式存储模型[28]. 在处理数据写入时, 在内存中保留的时间以及数据量阈值可以设定. 由于磁盘顺序写速度较快的特性, 数据会先在写缓冲区中进行快速排序, 等到达刷写时机才会刷写到磁盘中.

TiDB 的事务型子系统 TiKV 采用 RocksDB[24]作为其底层存储引擎. RocksDB 是一个基于 LSM 树结构的 KV 存储引擎. TiKV 将数据按照范围切割成了多个切片 (region), 同一个节点的所有切片数据存储在同一个 RocksDB 实例上. TiKV 采用 multi raft-group[24]的副本机制, 每一个切片会有多个副本提供读写服务. TiKV 通过总控节点对这些切片的副本进行调度, 以保证数据和读写负载均匀地分散在各个节点上, 保证了整体负载均衡和良好的可扩展性. 分析型子系统 TiFlash 的存储引擎也是基于 LSM 树, 包括变更层和持久层两层, 分别对应 LSM 树的 L0 和 L1 层. 最新的数据会首先被写入缓冲区, 写满之后才会被刷入磁盘上的变更层. 当变更层写满之后, 会与持久层做一次合并操作 (delta merge) 得到新的持久层, 其存储格式 DTFile 为文件夹标准文件格式.

F1 lightning server[23]通过分区加 LSM 树的方式来进行数据存储. F1 lightning 主要依赖的是变更数据捕获 (change data capture, CDC) 层面的复制, 这种复制得以屏蔽前端系统不同带来的复杂度. ChangePump 是处于 lightning server 之外的服务, 这个针对不同 OLTP 引擎定制化实现的服务可以帮助将 OLTP 日志同步到 lightning 服务器. Lightning 通过 ChangePump 捕获 OLTP 数据库的更新, 以订阅的方式把数据分发到订阅者. ChangePump 内部还开发了一个适配器, 将不同源的 CDC 模式转换成统一的格式. Lightning 的主存存储是以行存储的模式存在, 采用 B 树索引, 在数据写入磁盘时才把行存储转换成列存储. 上层查询通过快照的机制读取可见范围的相关数据.

4.2 计算部分的关键技术

4.2.1 查询引擎

(1) 针对混合存储的优化器

在应用行列混存的双拷贝类型 HTAP 数据库中, 一份数据可能在多个机器上存有不同存储格式的拷贝, 如 TiDB 的同一张表可能在 TiKV 上存有行存副本, 同时在 TiFlash 上有列存副本. 在这类系统上, 传统的数据库优化器会面临两大问题: 首先, 其通常只针对单一行存或列存系统, 由于行列存储的访问方式、索引策略等都大相径庭, 所以势必会影响优化器的性能. 在执行计划生成阶段, 传统策略将很难进行准确的基数估计和代价估计, 无法很好支持这类混合存储的系统. 其次, 一些系统支持在不同副本上联合执行查询, 将算子下发到不同的存储格式上

执行, 因此如何在不同拷贝上分配、调度算子, 并生成对应的执行计划成为了一个新需求.

针对这两大问题, 目前业界的解决方案较为局限. 一些数据库如 TiDB 设计了一套增加对混合负载支持的增强优化器[24], 其主要职责是判断一个查询应该被调度到行存执行还是列存执行. 对于创建了列存副本的表, 优化器会自动根据代价估算选择是否使用该副本. 这里的代价估算包括启发式算法和基于经验的规则. 但是由于基数估计和代价估计本身的误差, 以及行列混合存储本身的复杂性, 该类优化器需要配合用户设置的白名单以及统计信息来保证较稳定的效果.

(2) 向量化计算加速

在 OLAP 场景下, SQL 中经常会包含很多涉及一个或者多个值、运算符、函数组成的计算过程. 这些表达式的求值是一个计算密集型的任务, 因此表达式的计算效率是影响整体性能的一个关键的因素. 传统的以 MySQL 为代表的表达式计算体系以行为单位进行逐行运算, 称为迭代器模型. 由于迭代器对整张表进行了抽象, 整个表达式实现为一个树形结构, 所以整个处理的过程非常清晰. 但这种抽象会同时带来性能上的损耗, 因为在迭代器进行迭代的过程中, 每一行数据的获取都会引发多层的函数调用, 同时逐行地获取数据会带来过多的 I/O. MySQL 采用树形迭代器模型, 是受到存储引擎访问方法的限制, 这导致其难以对复杂的逻辑计算进行优化. 在列存格式下, 由于每一列的数据都分别顺序存储, 涉及某一个特定列上的表达式计算过程都可以批量进行. 在不少的系统如 HyPer, SAP HANA, MemSQL 等均有实现这种向量化计算加速. IBM Wildfire 对接 Spark 执行器进行无状态向量化执行. SQL Server 借助延迟物化[60]、SIMD 指令集[61]等技术, 针对列存储提供批量执行的算子以加速分析操作. Oracle 的内存组件也针对聚合操作进行 SIMD 的向量化加速.

(3) 大规模并行处理 (massively parallel processing, MPP)

在 MPP 数据库非共享集群中, 每个节点都有独立的磁盘存储系统和内存系统, 业务数据根据数据库模型和应用特点划分到各个节点上, 每台数据节点通过专用网络或者商业通用网络互相连接, 彼此协同计算, 作为整体提供数据库服务. 非共享数据库集群有完全的可伸缩性、高可用、高性能、优秀的性价比、资源共享等优势. 简单来说, MPP 是将任务并行的分散到多个服务器和节点上, 在每个节点上计算完成后, 将各自部分的结果汇总在一起得到最终的结果. 利用 MPP 提升 HTAP 效率的核心在于数据分布合理、能够适应 HTAP 的工作负载. Greenplum[22]是典型的 MPP shared-nothing 架构. 通过多级分区, 数据被切分为细粒度的分区, 底层使用不同的存储格式 (行存或列存) 保存不同的分区, 来降低每次 SQL 处理要扫描的数据量.

4.2.2 索引支持

索引是对数据库表中属于同一数据域的值进行快速定位的一种结构, 其主要目的是加速检索, 是一种使用空间换时间的策略. 传统的索引包括位图索引、哈希索引、B/B+树索引、跳表 (skip list)、ART 索引等[62]. 这些索引方案在实践中得到了广泛的应用, 如 B+树索引已经在关系数据库系统中使用了几十年.

由于 HTAP 系统中的存储模式不尽相同, 所以其选用的索引方法也各有千秋. 索引的性能与工作负载强相关. 例如, 在范围查询较多的情况下, B+树索引的性能显然强于哈希索引、位图索引等支持点查的索引结构[63]. 在 HTAP 的研究领域, 涌现出了各种新的索引方案来强化对混合负载的支持. 有些根据工作负载组合使用多种索引[18], 如根据某张表在工作负载中更多服务于 OLTP 或是 OLAP 创建相应的 B+树索引或是列存索引. 另一些是针对系统自身架构特点提出的全新索引结构. 一些数据库管理系统跨多个区域 (multiple zone) 以不同的方式组织数据: 最新的数据位于 OLTP 友好的区域, 而较旧的数据位于 OLAP 友好的区域. 根据冷热程度、访问频率等因素, 数据会动态地从一个区域被转移到另一个区域. 这种多区域设计应用于许多 HTAP 系统, 包括 SAP HANA、MemSQL 等. 对于这类系统, 为跨区域的大量数据提供统一的索引对于支持快速点查询和范围查询至关重要.

(1) 组合使用 B+树和列存储索引

SQL Server 支持在同一个表上建立 B+树或列存储索引[63]. 该列存储索引是一种聚集索引, 它们可以作为主索引建立在表中所有列上; 也可以作为冗余二级索引, 建立在一部分选定的列上. 聚集索引承担了查找数据和存储数据的双重任务. 在 SQL Server 中, B+树被用于 OLTP 任务, 列存储索引被用于 OLAP 任务.

(2) 定制化字典和存储索引

Oracle 的列存系统[35]在内存中, 内存压缩单元 (IMCU) 里每一个列都会包含对应的字典信息和存储索引信息. 每个列的不同值编写成一个字典, 之后为该列的每一行数据指定一个键值, 来代替具体的值, 这样做的优点是节省空间, 支持 SIMD. 除了字典外, IMCU 中每个列的头信息当中都会保存这个列在对应的 IMCU 当中的最大值和最小值, 以及他们所对应的偏移量, 称为存储索引 (storage index). 可以在查询数据时通过对比最大和最小值的方式快速过滤掉不满足条件的数据, 而且一旦数据改变可以快速定位到对应的位置.

(3) Umzi 索引

Wildfire 系统采用一种多版本、多区域的类 LSM 索引方法 Umzi[64]. 面向 HTAP 中由于事务而不断发生变化的数据, 它提供了统一的索引视图. Umzi 采用灵活的索引结构, 将哈希和排序技术结合在一起, 以支持点查询和范围查询. 此外, 它还充分利用分布式集群环境 (内存、SSD 和分布式共享存储) 中的存储层次结构来提高索引效率. 为了实现最大的并发性, Umzi 中的所有索引维护操作都被设计为非阻塞和无锁查询, 这样的好处是并发的索引修改只会产生最小的锁定开销.

(4) LSM 树存储结构下的索引支持

LSM 树[14]作为一种新型的存储格式, 由于其极高的写性能, 被多种 HTAP 数据库使用. 在 LSM 树中, 为了能够加速其查询性能, 需要在内存和磁盘中辅助使用其他索引结构. LSM 树的索引一般指的是内存里面 L0 层所用的索引, 比较常见的做法是 B+树和跳表, 同时附加多级缓存来优化索引效果. 除了索引之外, 由于 LSM 树存储的有序性, 可以通过维护一个检索表来记录 SSTable 内键的最小最大值来快速定位到候选 SSTable.

4.2.3 事务的保证

数据库事务包含了一系列的对数据库的读写操作. 并且为数据库操作序列提供了一个从失败中恢复到正常状态的方法, 同时提供了数据库即使在异常状态下仍能保持一致性的方法. 当多个应用程序在并发访问数据库时, 事务可以在这些应用程序之间提供一个隔离方法, 以防止彼此的操作互相干扰.

在 HTAP 实际场景中, 数据库中的数据不仅在应用层要被多个用户共同访问、在系统层也有可能被多个面向不同任务的系统访问. 在多方同时操作一份数据对象时, 可能会出现一些事务的并发问题, 具体可概括为 5 种: 脏读 (dirty read)、不可重复读 (nonrepeatable read)、虚读/幻读 (phantom read)、丢失更新 (lost update) 和写偏斜 (write skew)[65].

针对并发问题, 定义了 4 种事务隔离级别, 包括 (1) 读未提交 (read uncommitted), (2) 读已提交 (read committed), (3) 可重复读取 (repeatable read), (4) 可序列化 (serializable). SQL 标准定义的这 4 个隔离级别, 只适用于基于锁的事务并发控制. Berenson 等人[66]提出一种新的隔离级别: 快照隔离 (snapshot isolation, SI). 该隔离级别通常由 MVCC 多版本控制支持, 通过维护数据的多个版本, 确保事务读取与自己的时间戳一致的数据.

上述这 5 种事务的隔离级别和允许发生的并发问题如图 6 所示. 不同的隔离级别对应的并发度和一致性有所不同, 事务的隔离级别越高, 越能保证数据库的完整性和一致性, 但对并发性能的影响也越大.

在 HTAP 数据库中, 事务和分析的并发能力往往是影响性能的重要因素. 因此, 如何加锁、如何进行隔离都至关重要. HTAP 数据库中的事务保证主要包含两种类型.

第 1 种类型是基于 MVCC 的, 依赖于 MVCC 协议和日志记录来处理事务. 这种方案会为新的数据创建新的版本, 具有开始时间戳和结束时间戳, 旧版本被标记为删除. 因为 DML 操作主要在内存中执行, 所以事务处理是高效的. Oracle Dual-Format, SQL Server, SAP HANA 等都是使用这一策略. 然而, 在传统的 MVCC 策略中 OLTP 和 OLAP 负载共用一条版本链, 版本搜索和版本清理的代价比较高昂. Kim 等人在 2022 年提出了一种新的 MVCC 系统 Diva (decoupling index from version data)[67], 利用解耦的思想使 MVCC 更适合 HTAP 的应用场景. 其一, 由于版本搜索和版本清理的过程之间存在干扰, 所以 Diva 把两个流程解耦, 以便独立地进行模块的优化; 其二, 对于故障恢复而言, 往往只需要最新的已提交版本, 所以 Diva 将最新已提交版本和正在处理的版本归为一类, 而剩余的大量历史版本则放置在额外的存储空间. 这样既可以提升恢复的效率, 又为模块各自优化带来可能. 第二种类型是基于两阶段提交 (2PC) 和 RAFT 协议的, 主要应用于分布式数据库, 如 TiDB[24]. 事务在分布式节点上采

用两阶段提交协议、基于 RAFT 的一致性算法和预写日志技术进行处理. 领导节点接收来自 SQL 引擎的请求, 然后在本地追加日志, 并将日志异步发送给跟随者节点. 如果多数节点成功添加日志, 则领导者提交请求并在本地应用该请求. 在性能上与第 1 种类型相比, 第 2 种类型由于分布式事务处理而具有较低的效率.

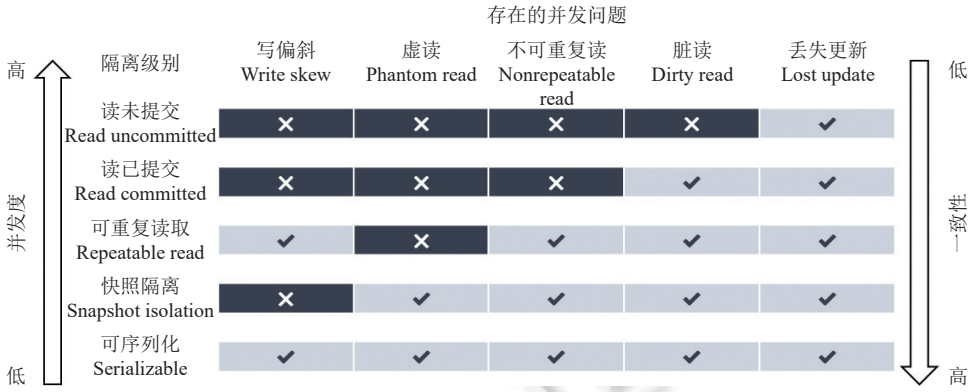


图 6 隔离级别和存在的并发问题

在隔离级别上, “可序列化”级别可以有效避免并发问题, 但可能导致大量的超时现象和锁竞争, 在实际应用中很少使用. 在快照隔离模式下不会发生上述四种并发问题, 且读操作不会被阻塞, 对于读多写少的应用是非常好的选择, 所以被众多 HTAP 系统所采用, 如 SQL Server, OceanBase, Peloton, IDAA, TiDB.

为了更灵活地处理 HTAP 负载, 现有的 HTAP 数据库都提供了可选的隔离级别, 如 SAP HANA 的默认隔离级别为“读已提交”, 最高级别为“可序列化”. 这样的设计保证了用户可以根据性能需求进行调整. Oracle 提供了“读已提交”和“可序列化”两种隔离选项, 默认值为较高的“可序列化”. 除此之外还有一种附加模式“只读”, 它不是 SQL 标准的一部分, 类似于“可序列化”隔离级别, 但不允许在事务中修改数据. 读取一致性是通过从撤销 (undo) 语句重建数据来实现的.

5 HTAP 数据库系统的性能评价

5.1 性能评测测试基准

为了筛选适合其业务运营的最佳数据库系统选择, 必须根据期望的性能目标评估这些 HTAP 系统. OLTP 和 OLAP 分别都有实行几十年的行业基准, 如 TPC-H 和 TPC-C 基准在业界被广泛使用. 这些基准通过模拟与行业高度相关的数据、事务和查询精确地评估数据库系统性能.

针对混合负载处理性能, 一种直观的做法是将 TPC-C 和 TPC-H 基准测试都部署在现代 HTAP 数据库系统上, 以分析它们的事务和分析处理能力. Hrubaru 等人[68]就用此方法对 3 个内存数据库系统进行了性能评估, 这些系统使用 TPC-H 基准测试处理混合工作负载, 指标为查询处理时间和吞吐量. 然而, 在这种评估中, 有一个关键的方面被忽略了, 那就是在并发处理 OLAP 工作负载时对同一数据库系统上处理的 OLTP 工作负载的影响, 正是这种缺陷驱动了专用于 HTAP 的测试基准的设计与开发. 测试基准的设计要点包括: (1) 合理的模式 (schema), 能够支持业务要求, 且划分符合逻辑. (2) 仿真的工作负载 (workload), 混合事务和分析工作负载. (3) 客观的评价指标 (metric). 目前业界较为流行的 HTAP 数据库测试基准主要有 3 个——CH-benCHmark[69]、HTAPBench[70]和 HATtrick[31], 如表 3 所示.

(1) CH-benCHmark

该基准在用于 OLTP 的 TPC-C 测试基准和用于 OLAP 的 TPC-H 测试基准的单一工作负载套件之间架起了一座桥梁, 并执行复杂的混合工作负载: 基于 TPC-C 的订单输入处理的事务性工作负载和相应的 TPC-H 对应的 OLAP 查询套件在单个数据库系统中的相同表上并行运行. 由于它源自这两个最广泛使用的 TPC 基准, 因此

CH-benCHmark 产生的结果与单工作负载系统高度相关. CH-benCHmark 采用组合数据库模式, 它包含 5 种 OLTP 查询和 22 种 OLAP 查询. 这两类工作负载存储在基准测试部署前独立进行配置.

表 3 HTAP 数据库系统测试基准

HTAP测试基准	数据模式和工作负载	技术核心	工作负载是否可配置	资源访问交叉程度
CH-benCHmark[69]	TPC-C+TPC-H	通过基准参数控制的OLAP/OLTP组合; 独立的OLAP/OLTP指标	OLAP和OLTP均可配置	由于较小的访问交叉, 资源干扰较低
HTAPBench[70]	TPC-C+TPC-H	固定OLTP吞吐量; 动态控制OLAP	OLTP吞吐量固定, 由系统自动生成; OLAP可配置	
HATtrick[31]	基于SSB的扩展	引入两个新的性能指标: 吞吐边界 (throughput frontier)和面向查询的新鲜度 (query-driven freshness)	OLAP和OLTP均可配置	合理配置下干扰程度高, 资源竞争激烈

数据库模式包含 12 个表: 9 个 TPC-C 和 3 个 TPC-H 表 (供应商、国家和地区表). 对于 OLTP 能力评估, “新订单、付款、交付、订单状态和库存水平”这 5 类 TPC-C 交易均整合到 CH-benCHmark 中, 这些操作仅在 9 张 TPC-C 表上执行, 不会影响其余 3 个表. 对于 OLAP 能力的评估, CH-benCHmark 使用了 22 个与 TPC-H 中相同的查询, 并适配了 TPC-C 的数据模式及上下文. 其整体的实现方式简单, 但是由于 OLAP 的扫描和 OLTP 的修改在数据访问空间上的不一致, 较小的访问交叉使得遇到读写冲突的概率较低, 两类负载在处理上存在的资源干扰较低.

(2) HTAPBench

Coelho 等人[70]于 2017 年提出了另一个基准, 用于评估具有混合事务和分析工作负载的现代 HTAP 数据库系统——HTAPBench. 它的设计架构类似于集成 TPC-C 和 TPC-H 基准的 CH-benCHmark. 但在核心设计中额外引入了客户端均衡器, 它使用反馈控制机制来确保预先设定的 OLTP 吞吐量阈值这是因为实际场景中, HTAP 系统在处理分析查询时通常会要求不能过高地牺牲 OLTP 吞吐量. 在 HTAPBench 的核心设计架构中, 客户端平衡器是动态控制 OLTP/OLAP 比例的核心. 在实际运行时, 客户端均衡器每分钟启动 1 个 OLAP 客户端, 直到 OLTP 吞吐量低于设置的阈值. 反馈控制机制不断向客户端均衡器报告注册的吞吐量, 如果 OLTP 吞吐量持续高于设置的阈值, 则释放 1 个 OLAP 客户端.

HTAPBench 的数据库模式与 CH-benCHmark 一致, 包括 TPC-C 基准的 9 个表和 TPC-H 的 3 个表. 同样的, TPC-H 部分以非侵入性的方式集成到 TPC-C 工作负载中, 与 TPC-C 的负载访问的数据不发生冲突. HTAPBench 中的动态查询生成器生成不断变化的工作负载. 这是首次在更新的数据库上生成查询的方法, 可确保分析操作在最新数据上进行. OLTP 处理主要受配置文件中设置的目标 TPS (每秒事务数) 的影响. 用户可以通过配置最佳仓库数量和最大允许客户数量来满足 OLTP 性能预期.

(3) HATtrick

HATtrick[31]是威斯康辛大学的 Milkai 等人在 2022 年提出的针对 HTAP 数据库的全新测试基准. CH-benCHmark 和 HTAPBench 两个基准都无法测量隔离程度和新鲜度, 且无法识别 HTAP 数据库的设计类别. HATtrick 的设计者认为, 不同任务之间的隔离性和控制新鲜数据的访问是 HTAP 数据库实现中面临的主要挑战. 据此, 该基准提出了两个新的性能指标: 吞吐边界 (throughput frontier) 和面向查询的新鲜度 (query-driven freshness), 如图 7 所示. 吞吐边界线位于对角线之上越接近边界框 (bounding box) 隔离性能越好, 越接近对角线 (proportional line) 表明事务负载和分析负载之间的代价越均衡, 低于对角线越接近坐标轴表示事务负载和分析负载之间的干扰程度越高, 资源竞争越激烈. 吞吐边界刻画了被测系统中事务负载和分析负载之间的代价权衡和资源竞争程度, 而新鲜度则由分析所用的数据版本相较于事务修改的数据版本之间的差距表征.

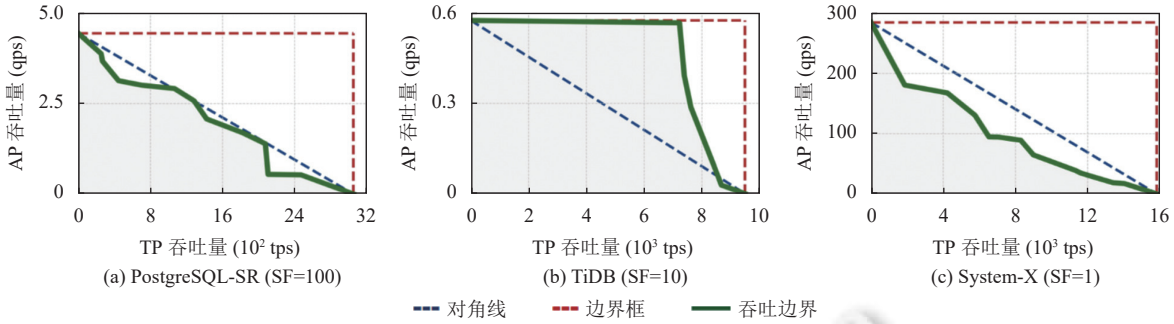


图 7 HATtrick[31]的吞吐边界 (throughput frontier) 示意图

HATtrick 的数据库模式和工作负载也与之前的 TPC-C+H 组合不同, 从 SSB[71]的模式中扩展而来, 新增了历史记录表、新鲜度记录表以及部分字段. 工作负载方面, OLTP 型负载受 TPC-C 启发使用自建的下单事务、付款事务和订单计数事务, 分析型负载使用 13 个调整后的 SSB 查询.

5.2 主要的评价性能指标

评价一个 HTAP 数据库系统的基本性能指标如下.

(1) 响应时间

响应时间指的是在请求提交后, 返回结果的平均时间. 响应时间直观体现了数据库的调度效率以及处理能力. 在混合事务和分析处理环境下, 响应时间可能受引擎、存储等多种设计因素影响, 可以帮助对比不同架构的调度处理能力. 除此之外, 可以设置超时阈值, 计算超时请求的数量, 以衡量系统的可靠性和稳定性.

(2) 数据新鲜度

对数据新鲜度的评估业内并没有一个统一的指标. 在 HATtrick 中, OLAP 查询的新鲜度 f_{A_q} 被定义为用于 OLAP 查询的数据快照与被 OLTP 修改的快照之间的时间差, 该数值越大说明数据从产生到用于分析的时延越长, 数据新鲜度越低[31], 如公式 (1) 所示:

$$f_{A_q} = \max\left(0, t^{S}_{A_q} - t^{fns}_{A_q}\right) \tag{1}$$

其中, $t^{S}_{A_q}$ 表示 OLAP 查询的执行开始时间, $t^{fns}_{A_q}$ 表示首个不可见 (first-not-seen) 的 OLTP 事务的提交时间.

(3) 吞吐量

吞吐量指数据库系统单位时间能处理请求的量. 可量化为每秒处理的事务数. 通过增加集群数量或是优化架构均可以提高数据库的吞吐量. 吞吐量越大说明数据库在应对大规模请求时的性能越好. 可以细分为 OLTP 吞吐量、OLAP 吞吐量、整体吞吐量等以综合考量 HTAP 系统的性能.

在 CH-benCHmark 中, OLTP 吞吐量的度量通过 (1) TPC-C 的度量 *tpmC* 和 (2) TPC-H 的度量标准 *QphH*. 直观上这两个指标彼此成反比: 更高的事务吞吐量意味着, 随着数据大小的增加, 需要扫描的数据量不断增加, 查询处理性能会恶化. 除此之外, 它还取决于数据库系统特有的其他关键因素, 如存储布局、索引、事务隔离级别、查询并行机制等.

对于 HTAPBench 而言, 考虑到事务导致的数据量的增长最终会降低 OLAP 性能, 所以在测试设计中缩小了这一研究差距, 引入了一种受监管的事务执行机制. 这保证了持续的、预先确定的数据量增长. 由于新数据的出现率保持不变且可预测, 因此无需对 OLAP 结果进行标准化, 从而使统一指标的路径无障碍.

$$QpHpW = \frac{QphH}{\#\ OLAPworkers} @tpmC \tag{2}$$

公式 (2) 描述了该基准的统一指标. 它是一个综合计算指标, 使用 TPC-C 的 *tpmC* 和 TPC-H 的 *QphH* 指标, 以及客户机平衡器在整个测试执行期间释放的 OLAP 客户端的数量. 统一度量标准为 *QpHpW*, 即“每个工作客户端每小时被处理的 TPC-H 类查询数量”, 可以表示单个 OLAP 客户端在维持事务处理能力为 *tpmC* 时能够被执行的

查询数. 因此, 它是在保证一定事务处理能力下衡量 OLAP 负载的效率标准.

5.3 不同设计对性能的影响及权衡

为了直观表述不同的设计架构 (单/多系统, 单/双拷贝) 对于 OLTP 查询和 OLAP 查询的影响, 定义 HTAP 系统对性能权衡的评价维度: 性能损失和数据新鲜度.

性能损失 (performance degradation) 是对于 OLTP 事务处理而言的. 通常情况下, 确保事务的处理性能是基本要求, 所以性能损失定义为同时处理 OLAP 事务后, 对于 OLTP 事务的干扰程度. 性能损失的比例越高, 说明为了分析能力牺牲了过多的事务性能, 可能会影响较为重要的日常事务. 在金融业等高时效行业, 这样的性能损失评估是非常重要的.

数据新鲜度 (freshness) 是对于 OLAP 而言的, 在大部分的数据库中, 除非共享一份底层数据, 否则数据都是由事务系统流向分析系统. 数据新鲜度即可用分析时使用的数据较事务系统最新数据版本的时间差距来量化表示 (即数据发生更新后到被分析系统使用的最大时延).

部分 HTAP 数据库系统按照以上维度的性能表现如图 8 所示. 图中各系统在数据延迟和事务性能损失两个象限上的定位主要参考了各系统公开的性能结果. 需要说明的是, 本文希望通过此图对各类 HTAP 系统的性能进行概括性的定性比较, 而非绝对客观的定量比较. 因为, 在具体系统测试时, 不同的软硬件配置和优化措施都会导致系统性能结果有较大差异. 此外, 前文讨论的一些系统由于没有公开的性能结果, 所以无法在图 8 中进行完整陈列.

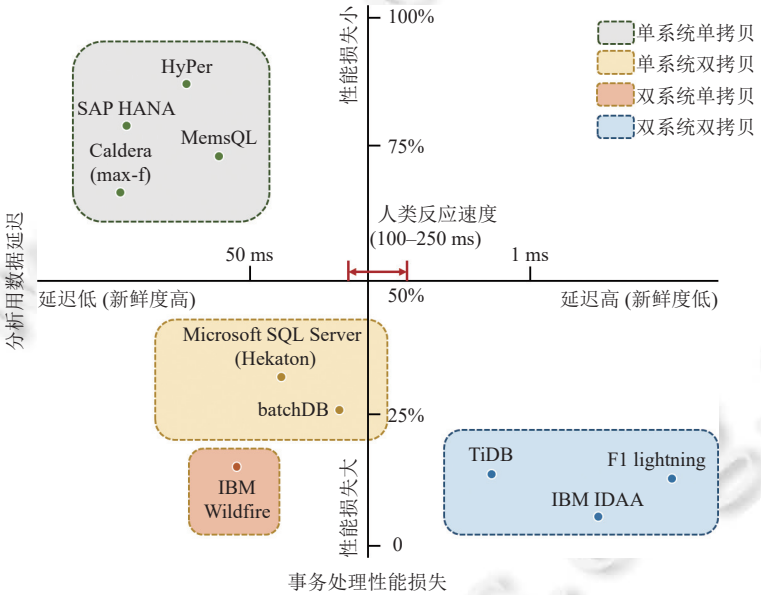


图 8 不同 HTAP 架构的 OLTP 性能和 OLAP 数据新鲜度权衡

5.3.1 系统设计对混合负载的查询性能影响

单系统数据库在执行混合负载的情况下, 由于 OLTP 负载和 OLAP 负载相互干扰, 抢占同一资源, 所以互相干扰最为严重. 可以看到 HyPer, SAP HANA, Caldera, SingleStore 等单拷贝存储下的单系统数据库普遍 TP 性能损失在 50% 以上[72]. 在这类系统中通常会根据工作环境严格控制 OLTP 和 OLAP 负载的比重, 以尽量减少性能降低引发的问题. 在双拷贝存储的数据库中, 单系统方案, 如 SQL Server 和 BatchDB 相较于双系统方案, 如 TiDB, F1 lightning 等系统, 事务处理性能的损失明显会更高.

5.3.2 存储架构对混合负载的影响对比

观察图 8 中不同架构的 3 个区域不难发现, 单拷贝存储的一大特点是新鲜度远远高于双拷贝存储, 这是符合直觉的, 快照机制的延迟远小于系统间的数据传输和同步. 但是这样的优势带来的明显缺陷是由于数据加锁、CPU 调度、内存占用等并行引起的开销较大, 导致系统整体吞吐量下降, 特别是对于事务系统, 如果不加以控制,

可能会有明显的性能降低.

在双拷贝存储中, 单系统的数据新鲜度更高, 这是由于双系统数据库的系统间同步数据涉及网络传输等额外开销. 双拷贝存储的一大优点是相互干扰小, OLTP 负载和 OLAP 负载几乎能够以完整性能同时被处理. TiDB, IBM IDAA 和 F1 lightning 都是采用这一设计, 在应用中效果也令人满意. 除非是实时系统, 如预测、系统监测、在线游戏、股价监控等任务可能受到数据更新延迟的影响, 其余场景几乎都能在兼顾性能的前提下满足数据新鲜度的需求.

6 HTAP 数据库系统的研究展望

6.1 云原生支持下的弹性资源调度

随着云计算技术的发展, 业界开始思考云原生架构对 HTAP 系统的帮助. 前文提到, HTAP 的核心问题是资源分配问题. 目前主要有两种资源调度的方案: 新鲜度驱动的 (freshness-driven) 资源调度方法[30]和工作负载驱动 (workload-driven) 的方法[8,73]. 前者依赖于基于规则的方法来控制执行模式, 但没有考虑工作负载. 后者根据工作负载特点调整事务处理和分析处理的线程数, 但忽略了新鲜度因素. 目前还缺乏一个综合考虑新鲜度和工作负载的调度方案.

除此之外, 云原生灵活的系统架构能否为 HTAP 的设计带来便利是一个备受学界关注的问题. 云技术的一个重点就是对于资源的隔离和弹性分配. 在此基础上, 业界和学术界已经开始思考存算分离和弹性算力能否根据 HTAP 数据库的业务特点缓解资源竞争. 因为这两条技术下的系统, 在资源竞争和数据可见度上的表现各有优缺点, 都是在数据延迟和资源争用上做取舍. 对 HTAP 的资源隔离和竞争更深入的理解和定义有助于选择更适合的技术路线为 HTAP 优化服务. 以计算资源为例, OLTP 请求与 OLAP 请求的资源使用特征差异较大, OLTP 大多是对数据的修改和小基数的查询, 主要占用 IO 资源, 而 OLAP 顺序扫描大量数据并做运算, 会在 IO 资源上对 OLTP 类型请求的处理带来干扰, 并占用 CPU 时间. 如果能够根据不同客户端的历史信息, 查询特征等对客户端类型进行识别、分类, 并相应地分配系统资源, 并对两类客户端的资源进行有效隔离, 就可以尽可能地减少云计算资源的浪费, 同时最大化系统的整体性能. 此外, 云环境下丰富的存储资源池也能够在混合事务中针对不同的数据模态、不同的业务特征进行划分, 从而降低用户的整体成本. 其中, 对于客户端业务类型的识别、建模和相应的资源调度都是难点所在.

6.2 人工智能赋能 HTAP

传统的基于经验的数据库优化技术, 如代价估计、基数估计、执行计划优化、索引等, 在如今 HTAP 数据库高度复杂的系统和存储架构下, 已经难以满足高性能、高可用的需求了. 而人工智能技术则借助计算机硬件技术和大数据的支持得到了飞速发展. 近年来, 不少基于机器学习的优化方法已经逐步进入数据库优化领域.

在索引领域, Kraska 等人[74]提出一种用 AI 模型来代替传统索引数据结构的方案, 称为学习型索引 (learned index), 启发了一系列数据库领域的广泛研究[75−77]. 许多 HTAP 数据库都使用了基于 LSM 树的存储系统, 如 HBase, OceanBase, TiDB 等. Dai 等人[78]基于学习型索引的思想, 设计了一款适用于键值分离的 LSM 树存储结构的学习型索引 Bourbon, 并且提出了若干条面向该类存储系统的索引建议. 该领域的拓展研究可以着眼于如何提高模型的可靠性和实用性, 如对于建立学习型索引进行更全面的成本效益分析、对于通用字符串的支持和提供不同的模型选择等.

在查询优化器方面, 以往的基于机器学习的优化器, 无论是基于数据 (data-driven) 的[79−81], 或是基于工作负载 (workload-driven) 的[82,83], 都存在局限性. 研究表明[84], 基于机器学习的方法在准确率上相比于传统方法有比较大的提升. 然而相比于传统方法, 机器学习方法需要更多的时间去训练模型, 同时模型推理时间也更长. 此外, 许多端到端的方法基于底层设计是透明的假设, 在训练模型的时候并不关注具体的存储架构, 但是在 HTAP 问题下, 系统架构更丰富复杂, 如果不对底层的设计加以考虑, 那么模型的鲁棒性和普适性会大打折扣.

在物理设计 (physical design) 方面, 为了有效地存储和查询, DBMS 必须人工选择存储和索引. 这个过程包括决定数据应该存储在行存或列存中、内存中还是磁盘上, 以及需要建立索引的列. 对 HTAP 系统来说这些选择极

具挑战性. Abebe 等人提出的 Tiresias 系统[85]可以预测数据访问成本以及在不同存储场景下的延迟. Tiresias 通过观察查询延迟和查询历史, 在线构建预测模型, 从而自适应地调整存储和索引.

而在数据库系统优化的其他环节, 一些跨领域的 AI 模型也有取得了相当可喜的效果. 如在数据库配置 (configuration tuning) 领域, BERT 等 NLP 模型被用于识别工作需求, 对数据库配置进行自动调优[86,87]. 如何将新兴的 AI 技术和数据库, 特别是 HTAP 数据库有机结合, 未来仍然有非常大的发展空间.

6.3 基于新硬件的优化

新硬件技术极大促进了数据库的发展, 例如发展迅猛的 GPU、FPGA、NVM[88]技术已经深入到数据库优化领域. 对于计算处理能力, 以 FPGA、GPU 为代表的异构并行计算体系架构打破了同构计算模式处理能力和数据增长速度之间的壁垒.

GPU 日益增长的计算能力使其覆盖的任务比先前更为广泛, 英伟达 (NVIDIA) 公司 2016 年提出的 Pascal GPU 比前代的 Tesla 架构具有 16 倍的处理能力和 13.3 倍的内存容量, 而后续的 Turing, Ampere 架构更是有数十倍的计算能力提升. 随着 GPU 的内存容量不断增加、可编程性提高, GPU 已经被用于与 CPU 协作处理 HTAP 负载. 基于 H^2TAP 架构的 Caldera[1]使用 GPU 来处理 OLAP 查询, 大大减轻了对 CPU 资源的占用, 整体提升了系统性能.

现场可编程逻辑门阵列 (field programmable gate array, FPGA) 作为一种可编程芯片, 在流水线并行计算、响应延时等方面优于 CPU, 因此在数据库领域中得到应用. Sukhwani 等人[89]提出了一种无需修改原有数据布局和应用程序的 FPGA 调用机制, 用以加速分析型查询. Owaida 等人[90]提出了一种 CPU+FGPA 的异构架构 Centaur, 并集成到列存储数据库 MonetDB 上, 它允许在执行计划上动态分配 FPGA 操作符, 以达到在 CPU 和 FPGA 上混合运行的目的.

在双拷贝 HTAP 系统中, 大量变更数据的传输是一大性能瓶颈, 特别是双拷贝 HTAP 系统. 针对该问题, 新接口技术也能提供强大的助力. 高速串行总线技术从 PCIe1.0 到如今 PCIe4.0, 性能进步了超过 8 倍. 在最高 X16 通道下, PCIe4.0 的最大带宽达到了 32 GB/s, 是上一代 PCIe3.0 技术的两倍. 在涉及 GPU 的通信上, 英伟达也提出了 NVLink[91], 这是一种节能、高带宽的 GPU 到 CPU 或 GPU 到 GPU 的互联组件, 可以提供至少 5 倍于 PCIe3.0 总线的带宽. 这些超高速的传输接口可以使得大规模数据在不同存储设备、不同主机上的同步更迅速, 而这种低延迟、异步低占用的数据传输正是 HTAP 系统亟需的.

在存储上, 更高速、更大容量的存储介质提供了更广的设计选择. NVM, 即非易失性存储器是近年来新出现的一系列存储介质, 包括 PCM 相变存储器[92]、STT-MRAM[93]、ReRAM[94]等. 第 1 代 NVM 产品在 2019 年由英特尔公司推出, NVM 从性质上看介于磁盘和内存两者之间, 性能明显优于外存, 存储密度高于 DRAM, 成本也更低. 基于现有的内存数据库或者重新定制 NVM 原生的数据库, 它可以接近内存数据库的性能的同时大大扩展内存空间, 同时也可以降低成本, 解决内存数据库数据量受限的瓶颈, 拓宽其适用范围. 不少 HTAP 数据库都采用了组合内存和硬盘的存储架构, 将变更数据存储在内存中, 而分析用的相对静态的数据存储在硬盘中, 借助 NVM 的支持可以加快静态数据的访问效率, 同时也可以增加从内存转移到磁盘的传输效率.

7 总 结

本文概述了 HTAP 技术的产生契机, 发展过程和现状. 从计算和存储两个角度对现有 HTAP 数据库进行了技术路线的分类与总结. 目前业界对 HTAP 系统的设计还处于百家争鸣的状态, 在学界和工业界涌现出一批优秀的设计和优化方案. 本文从底层存储、查询处理、引擎设计等角度介绍了最新研究进展. 并介绍了 3 个使用最广泛的测试基准和常用的性能评价方法. 最后根据不同设计方案的特性, 结合公开的性能数据分析其优缺点, 为后续的 HTAP 系统研究和开发提供建议.

致谢 感谢所有匿名审稿人对本文提出的宝贵修改意见. 感谢“高性能数据库系统关键技术校企联合研究中心”对本文研究工作提供的支持.

References:

[1] Appuswamy R, Karpathiotakis M, Porobic D, Ailamaki A. The case for heterogeneous HTAP. In: Proc. of the 8th Biennial Conf. on Innovative Data Systems Research. Chaminade: CIDR, 2017. 1–11.

[2] El-Sappagh SHA, Hendawi AMA, El Bastawissy AH. A proposed model for data warehouse ETL processes. Journal of King Saud University-computer and Information Sciences, 2011, 23(2): 91–104. [doi: 10.1016/j.jksuci.2011.05.005]

[3] Edjlali R, Feinberg D, Rayner N, Pezzini M. How to enable digital business innovation via hybrid transaction/analytical processing. 2016. https://www.gartner.com/en/documents/3352419

[4] Pezzini M, Feinberg D, Rayner N, Edjlali R. Hybrid transaction/analytical processing will foster opportunities for dramatic business innovation. 2014. https://www.gartner.com/en/documents/2657815

[5] Feinberg D, Ronthal A. Hype Cycle for Data Management, 2019. Gartner, 2019.

[6] Bouzeghoub M. A framework for analysis of data freshness. In: Proc. of the 2004 Int'l Workshop on Information Quality in Information Systems. Paris: ACM, 2004. 59–67. [doi: 10.1145/1012453.1012464]

[7] Psaroudakis I, Wolf F, May N, Neumann T, Böhm A, Ailamaki A, Sattler KU. Scaling up mixed workloads: A battle of data freshness, flexibility, and scheduling. In: Proc. of the 6th TPC Technology Conf. on Performance Characterization and Benchmarking. Traditional to Big Data. Hangzhou: Springer, 2015. 97–112. [doi: 10.1007/978-3-319-15350-6_7]

[8] Sirin U, Dwarkadas S, Ailamaki A. Performance characterization of HTAP workloads. In: Proc. of the 37th IEEE Int'l Conf. on Data Engineering (ICDE). Chania: IEEE, 2021. 1829–1834. [doi: 10.1109/ICDE51399.2021.00162]

[9] MacLennan J, Tang ZH, Crivat B. Data Mining with Microsoft SQL Server 2008. Indianapolis: John Wiley & Sons, 2011. 40–43.

[10] Greenwald R, Stackowiak R, Stern J. Oracle Essentials: Oracle Database 12c. 5th ed., Sebastopol: O'Reilly Media Inc., 2013. 2–10.

[11] Schiefer B, Valentin G. DB2 universal database performance tuning. IEEE Data Engineering Bulletin, 1999, 22(2): 12–19.

[12] Idreos S, Groffen F, Nes N, Manegold S, Mullender SK, Kersten ML. MonetDB: Two decades of research in column-oriented database. IEEE Data Engineering Bulletin, 2012, 35(1): 40–45.

[13] Bear C, Lamb A, Tran N. The vertica database: SQL RDBMS for managing big data. In: Proc. of the 2012 Workshop on Management of Big Data Systems. San Jose: ACM, 2012. 37–38. [doi: 10.1145/2378356.2378367]

[14] O'Neil P, Cheng E, Gawlick D, O'Neil E. The log-structured merge-tree (LSM-tree). Acta Informatica, 1996, 33(4): 351–385. [doi: 10.1007/s002360050048]

[15] Kemper A, Neumann T. HyPer: A hybrid OLTP&OLAP main memory database system based on virtual memory snapshots. In: Proc. of the 27th IEEE Int'l Conf. on Data Engineering. Hannover: IEEE, 2011. 195–206. [doi: 10.1109/ICDE.2011.5767867]

[16] Neumann T, Mühlbauer T, Kemper A. Fast serializable multi-version concurrency control for main-memory database systems. In: Proc. of the 2015 ACM SIGMOD Int'l Conf. on Management of Data. Melbourne: ACM, 2015. 677–689. [doi: 10.1145/2723372.2749436]

[17] Arulraj J, Pavlo A, Menon P. Bridging the archipelago between row-stores and column-stores for hybrid workloads. In: Proc. of the 2016 Int'l Conf. on Management of Data. San Francisco: ACM, 2016. 583–598. [doi: 10.1145/2882903.2915231]

[18] Larson PÅ, Clinciu C, Hanson EN, Oks A, Price SL, Rangarajan S, Surna A, Zhou QQ. SQL server column store indexes. In: Proc. of the 2011 ACM SIGMOD Int'l Conf. on Management of Data. Athens: ACM, 2011. 1177–1184. [doi: 10.1145/1989323.1989448]

[19] Chen J, Jindel S, Walzer R, Sen R, Jimsheleishvilli N, Andrews M. The MemSQL query optimizer: A modern optimizer for real-time analytics in a distributed database. Proc. of the VLDB Endowment, 2016, 9(13): 1401–1412. [doi: 10.14778/3007263.3007277]

[20] Barber R, Huras M, Lohman G, Mohan C, Mueller R, Özcan F, Pirahesh H, Raman V, Sidle R, Sidorkin O, Storm A, Tian YY, Tözun P. Wildfire: Concurrent blazing data ingest and analytics. In: Proc. of the 2016 Int'l Conf. on Management of Data. San Francisco: ACM, 2016. 2077–2080. [doi: 10.1145/2882903.2899406]

[21] Yang ZK. The architecture of OceanBase relational database system. Journal of East China Normal University (Natural Science), 2014(5): 141–148, 163 (in Chinese with English abstract).

[22] Lyu ZH, Zhang HH, Xiong G, Guo G, Wang HZ, Chen JB, Praveen A, Yang Y, Gao XM, Wang A, Lin W, Agrawal A, Yang JF, Wu H, Li XL, Guo F, Wu J, Zhang J, Raghavan V. Greenplum: A hybrid database for transactional and analytical workloads. In: Proc. of the 2021 Int'l Conf. on Management of Data. ACM, 2021. 2530–2542. [doi: 10.1145/3448016.3457562]

[23] Yang JC, Rae I, Xu J, Shute J, Yuan Z, Lau K, Zeng Q, Zhao X, Ma J, Chen ZY, Gao Y, Dong QL, Zhou JX, Wood J, Graefe G, Naughton J, Cieslewicz J. F1 lightning: HTAP as a service. Proc. of the VLDB Endowment, 2020, 13(12): 3313–3325. [doi: 10.14778/3415478.3415553]

[24] Huang DX, Liu Q, Cui Q, Fang ZH, Ma XY, Xu F, Shen L, Tang L, Zhou YX, Huang ML, Wei W, Liu C, Zhang J, Li JJ, Wu XL, Song LY, Sun RX, Yu SP, Zhao L, Cameron N, Pei LQ, Tang X. TiDB: A Raft-based HTAP database. Proc. of the VLDB Endowment, 2020, 13(12): 3072–3084. [doi: 10.14778/3415478.3415535]

[25] Ongaro D, Ousterhout J. In search of an understandable consensus algorithm. In: Proc. of the 2014 USENIX Conf. Philadelphia: USENIX Association, 2014. 305–320.

[26] O'Driscoll A, Daugelaite J, Sleator RD. 'Big data', Hadoop and cloud computing in genomics. Journal of Biomedical Informatics, 2013, 46(5): 774–781. [doi: 10.1016/j.jbi.2013.07.001]

[27] Tang SJ, He BS, Yu C, Li YS, Li K. A survey on spark ecosystem: Big data processing infrastructure, machine learning, and applications. IEEE Trans. on Knowledge and Data Engineering, 2022, 34(1): 71–91. [doi: 10.1109/TKDE.2020.2975652]

[28] Vora MN. Hadoop-HBase for large-scale data. In: Proc. of the 2011 Int'l Conf. on Computer Science and Network Technology. Harbin: IEEE, 2011. 601–605. [doi: 10.1109/ICCSNT.2011.6182030]

[29] Li GL, Zhang C. HTAP databases: What is new and what is next. In: Proc. of the 2022 Int'l Conf. on Management of Data. Philadelphia: ACM, 2022. 2483–2488. [doi: 10.1145/3514221.3522565]

[30] Raza A, Chrysogelos P, Anadiotis AC, Ailamaki A. Adaptive HTAP through elastic resource scheduling. In: Proc. of the 2020 ACM SIGMOD Int'l Conf. on Management of Data. Portland: ACM, 2020. 2043–2054. [doi: 10.1145/3318464.3389783]

[31] Milkai E, Chronis Y, Gaffney KP, Guo ZH, Patel JM, Yu XY. How good is My HTAP system? In: Proc. of the 2022 Int'l Conf. on Management of Data. Philadelphia: ACM, 2022. 1810–1824. [doi: 10.1145/3514221.3526148]

[32] Shute J, Vingralek R, Samwel B, Handy B, Whipkey C, Rollins E, Oancea M, Littlefield K, Menestrina D, Ellner S, Cieslewicz J, Rae I, Stancescu T, Apte H. F1: A distributed SQL database that scales. Proc. of the VLDB Endowment, 2013, 6(11): 1068–1079. [doi: 10.14778/2536222.2536232]

[33] Färber F, Cha SK, Primsch J, Bornhövd C, Sigg S, Lehner W. SAP HANA database: Data management for modern business applications. ACM Sigmod Record, 2011, 40(4): 45–51. [doi: 10.1145/2094114.2094126]

[34] Makreshanski D, Giceva J, Barthels C, Alonso G. BatchDB: Efficient isolated execution of hybrid OLTP+OLAP workloads for interactive applications. In: Proc. of the 2017 ACM Int'l Conf. on Management of Data. Chicago: ACM, 2017. 37–50. [doi: 10.1145/3035918.3035959]

[35] Lahiri T, Chavan S, Colgan M, Das D, Ganesh A, Gleeson M, Hase S, Holloway A, Kamp J, Lee TH, Loaiza J, Macnaughton N, Marwah V, Mukherjee N, Mullick A, Muthulingam S, Raja V, Roth M, Soylemez E, Zait M. Oracle database in-memory: A dual format in-memory database. In: Proc. of the 31st IEEE Int'l Conf. on Data Engineering. Seoul: IEEE, 2015. 1253–1258. [doi: 10.1109/ICDE.2015.7113373]

[36] Ballard C, Beaton A, Ketchie M, Ketelaars F, Noor A, Parkes J, Rangarao D, Shubin B, van Tichelen W. Smarter Business: Dynamic Information with IBM InfoSphere Data Replication CDC. New York: IBM Redbooks, 2012. 1–4.

[37] Smith JM, Maguire GQ Jr. Effects of copy-on-write memory management on the response time of UNIX fork operations. Computing Systems, 1988, 1(3): 255–278.

[38] Karun AK, Chitharanjan K. A review on Hadoop—HDFS infrastructure extensions. In: Proc. of the 2013 IEEE Conf. on Information & Communication Technologies. Thuckalay: IEEE, 2013. 132–137. [doi: 10.1109/CICT.2013.6558077]

[39] Lameter C. NUMA (Non-Uniform Memory Access): An overview: NUMA becomes more common because memory controllers get close to execution units on microprocessors. Queue, 2013, 11(7): 40–51. [doi: 10.1145/2508834.2513149]

[40] Copeland GP, Khoshafian SN. A decomposition storage model. ACM SIGMOD Record, 1985, 14(4): 268–279. [doi: 10.1145/971699.318923]

[41] Ailamaki A, DeWitt DJ, Hill MD, Skounakis M. Weaving relations for cache performance. In: Proc. of the 27th Int'l Conf. on Very Large Data Bases. San Francisco: Morgan Kaufmann Publishers Inc., 2001. 169–180.

[42] Vohra D. Apache parquet. In: Practical Hadoop Ecosystem. Berkeley: Apress, 2016. 325–335. [doi: 10.1007/978-1-4842-2199-0_8]

[43] He YQ, Lee R, Huai Y, Shao Z, Jain N, Zhang XD, Xu ZW. RCFile: A fast and space-efficient data placement structure in MapReduce-based warehouse systems. In: Proc. of the 27th IEEE Int'l Conf. on Data Engineering. Hannover: IEEE, 2011. 1199–1208. [doi: 10.1109/ICDE.2011.5767933]

[44] Kumar R, Kumar N. Improved join operations using ORC in HIVE. CSI Trans. on ICT, 2016, 4(2–4): 209–215. [doi: 10.1007/s40012-016-0128-6]

[45] Corbett JC, Dean J, Epstein M, *et al*. Spanner: Google's globally distributed database. ACM Trans. on Computer Systems, 2013, 31(3): 1–22. [doi: 10.1145/2518037.2491245]

[46] Alagiannis I, Idreos S, Ailamaki A. H_2O: A hands-free adaptive store. In: Proc. of the 2014 ACM SIGMOD Int'l Conf. on Management of Data. Snowbird: ACM, 2014. 1103–1114. [doi: 10.1145/2588555.2610502]

[47] Kang DH, Jiang RC, Blanas S. Jigsaw: A data storage and query processing engine for irregular table partitioning. In: Proc. of the 2021 Int'l Conf. on Management of Data. ACM, 2021. 898–911. [doi: 10.1145/3448016.3457547]

[48] Abebe M, Lazu H, Daudjee K. Proteus: Autonomous adaptive storage for mixed workloads. In: Proc. of the 2022 Int'l Conf. on Management of Data. Philadelphia: ACM, 2022. 700–714. [doi: 10.1145/3514221.3517834]

[49] White RM. Disk-storage technology. Scientific American, 1980, 243(2): 138–148. [doi: 10.1038/scientificamerican0880-138]

[50] Biliris A. The performance of three database storage structures for managing large objects. ACM SIGMOD Record, 1992, 21(2): 276–285. [doi: 10.1145/141484.130324]

[51] Brandt SA, Miller EL, Long DDE, Xue L. Efficient metadata management in large distributed storage systems. In: Proc. of the 20th IEEE/the 11th NASA Goddard Conf. on Mass Storage Systems and Technologies. San Diego: IEEE, 2003. 290–298. [doi: 10.1109/MASS.2003.1194865]

[52] Fitzpatrick B. Distributed caching with memcached. Linux Journal, 2004, 2004(124): 5.

[53] Agrawal S, Narasayya V, Yang B. Integrating vertical and horizontal partitioning into automated physical database design. In: Proc. of the 2004 ACM SIGMOD Int'l Conf. on Management of Data. Paris: ACM, 2004. 359–370. [doi: 10.1145/1007568.1007609]

[54] Lith A, Mattsson J. Investigating storage solutions for large data: A comparison of well performing and scalable data storage solutions for real time extraction and batch insertion of data [MS. Thesis]. Göteborg: Chalmmers University of Technology, 2010.

[55] DeWitt DJ, Katz RH, Olken F, Shapiro LD, Stonebraker MR, Wood DA. Implementation techniques for main memory database systems. In: Proc. of the 1984 ACM SIGMOD Int'l Conf. on Management of Data. Boston: ACM, 1984. 1–8. [doi: 10.1145/602259.602261]

[56] Sherkat R, Florendo C, Andrei M, Blanco R, Dragusanu A, Pathak A, Khadilkar P, Kulkarni N, Lemke C, Seifert S, Iyer S, Gottapu S, Schulze R, Gottipati C, Basak N, Wang YH, Kandiyanallur V, Pendap S, Gala D, Almeida R, Ghosh P. Native store extension for SAP HANA. Proc. of the VLDB Endowment, 2019, 12(12): 2047–2058. [doi: 10.14778/3352063.3352123]

[57] Codd EF. Relational database: A practical foundation for productivity. In: Mylopolous J, Brodie M, eds. Readings in Artificial Intelligence and Databases. Los Altos: Morgan Kaufmann, 1989. 60–68. [doi: 10.1016/B978-0-934613-53-8.50009-1]

[58] Seeger M. Key-value stores: A pracitical overview. Computer Science and Media, Ultra-Large-Sites SS09, 2009, 9: 1–21.

[59] Diaconu C, Freedman C, Ismert E, Larson PA, Mittal P, Stonecipher R, Verma N, Zwilling M. Hekaton: SQL server's memory-optimized OLTP engine. In: Proc. of the 2013 ACM SIGMOD Int'l Conf. on Management of Data. New York: ACM, 2013. 1243–1254. [doi: 10.1145/2463676.2463710]

[60] Abadi DJ, Myers DS, DeWitt DJ, Madden SR. Materialization strategies in a column-oriented DBMS. In: Proc. of the 23rd IEEE Int'l Conf. on Data Engineering. Istanbul: IEEE, 2007. 466–475. [doi: 10.1109/ICDE.2007.367892]

[61] Zhou JR, Ross KA. Implementing database operations using SIMD instructions. In: Proc. of the 2002 ACM SIGMOD Int'l Conf. on Management of Data. Madison: ACM, 2002. 145–156. [doi: 10.1145/564691.564709]

[62] Leis V, Kemper A, Neumann T. The adaptive radix tree: ARTful indexing for main-memory databases. In: Proc. of the 29th IEEE Int'l Conf. on Data Engineering (ICDE). Brisbane: IEEE, 2013. 38–49. [doi: 10.1109/ICDE.2013.6544812]

[63] Dziedzic A, Wang JJ, Das S, Ding BL, Narasayya VR, Syamala M. Columnstore and B+ tree—Are hybrid physical designs important? In: Proc. of the 2018 Int'l Conf. on Management of Data. Houston: ACM, 2018. 177–190. [doi: 10.1145/3183713.3190660]

[64] Luo C, Tözün P, Tian YY, Barber R, Raman V, Sidle R. Umzi: Unified multi-zone indexing for large-scale HTAP. In: Proc. of the 22nd Int'l Conf. on Extending Database Technology. Lisbon, 2019. 1–12. [doi: 10.5441/002/edbt.2019.02]

[65] Yang Y, Zhu J W. Write skew and zipf distribution: Evidence and implications. ACM Trans. on Storage, 2016, 12(4): 21. [doi: 10.1145/2908557]

[66] Berenson H, Bernstein P, Gray J, Melton J, O'Neil E, O'Neil P. A critique of ANSI SQL isolation levels. ACM SIGMOD Record, 1995, 24(2): 1–10. [doi: 10.1145/568271.223785]

[67] Kim J, Yu J, Ahn J, Kang S, Jung H. Diva: Making MVCC systems HTAP-friendly. In: Proc. of the 2022 Int'l Conf. on Management of Data. Philadelphia: ACM, 49–64. [doi: 10.1145/3514221.3526135]

[68] Hrubaru I, Fotache M. On the performance of three in-memory data systems for on line analytical processing. Informatica Economica, 2017, 21(1): 5–15. [doi: 10.12948/issn14531305/21.1.2017.01]

[69] Cole R, Funke F, Giakoumakis L, Guy W, Kemper A, Krompass S, Kuno H, Nambiar R, Neumann T, Poess N, Sattler KU, Seibold M, Simon E, Waas F. The mixed workload CH-benCHmark. In: Proc. of the 4th Int'l Workshop on Testing Database Systems. Athens: ACM, 2011. 1–6. [doi: 10.1145/1988842.1988850]

[70] Coelho F, Paulo J, Vilaça R, Pereira J, Oliveira R. HTAPBench: Hybrid transactional and analytical processing benchmark. In: Proc. of the 8th ACM/SPEC on Int'l Conf. on Performance Engineering. L'Aquila: ACM, 2017. 293–304. [doi: 10.1145/3030207.3030228]

[71] O'Neil P, O'Neil E, Chen XD, Revilak S. The star schema benchmark and augmented fact table indexing. Performance Evaluation and Benchmarking. Lyon: Springer, 2009. 237–252. [doi: 10.1007/978-3-642-10424-4_17]

[72] Shen SJ, Chen R, Chen HB, Zhang BY. Retrofitting high availability mechanism to tame hybrid transaction/analytical processing. In:

Proc. of the 15th USENIX Symp. on Operating Systems Design and Implementation. Santa Clara: USENIX Association, 2021. 219–238.

[73] Sikka V, Färber F, Lehner W, Cha SK, Peh T, Bornhövd C. Efficient transaction processing in SAP HANA database: The end of a column store myth. In: Proc. of the 2012 ACM SIGMOD Int'l Conf. on Management of Data. Scottsdale: ACM, 2012. 731–742. [doi: 10.1145/2213836.2213946]

[74] Kraska T, Beutel A, Chi EH, Dean J, Polyzotis N. The case for learned index structures. In: Proc. of the 2018 Int'l Conf. on Management of Data. Houston: ACM, 2018. 489–504. [doi: 10.1145/3183713.3196909]

[75] Ding JL, Minhas UF, Yu J, Wang C, Do J, Li Y, Zhang HT, Chandramouli B, Gehrke J, Kossmann D, Lomet D, Kraska T. ALEX: An updatable adaptive learned index. In: Proc. of the 2020 ACM SIGMOD Int'l Conf. on Management of Data. Portland: ACM, 2020. 969–984. [doi: 10.1145/3318464.3389711]

[76] Galakatos A, Markovitch M, Binnig C, Fonseca R, Kraska T. FITing-tree: A data-aware index structure. In: Proc. of the 2019 Int'l Conf. on Management of Data. Amsterdam: ACM, 2019. 1189–1206. [doi: 10.1145/3299869.3319860]

[77] Li PF, Lu H, Zheng Q, Yang L, Pan G. LISA: A learned index structure for spatial data. In: Proc. of the 2020 ACM SIGMOD Int'l Conf. on Management of Data. Portland: ACM, 2020. 2119–2133. [doi: 10.1145/3318464.3389703]

[78] Dai YF, Xu YE, Ganesan A, Alagappan R, Kroth B, Arpaci-Dusseau AC, Arpaci-Dusseau RH. From WiscKey to bourbon: A learned index for log-structured merge trees. In: Proc. of the 14th USENIX Conf. on Operating Systems Design and Implementation. Berkeley: USENIX Association, 2020. 155–171.

[79] Hilprecht B, Schmidt A, Kulessa M, Molina A, Kersting K, Binnig C. DeepDB: Learn from data, not from queries! Proc. of the VLDB Endowment, 2020, 13(7): 992–1005. [doi: 10.14778/3384345.3384349]

[80] Yang ZH, Liang E, Kamsetty A, Wu CG, Duan Y, Chen X, Abbeel P, Hellerstein JM, Krishnan S, Stoica I. Deep unsupervised cardinality estimation. Proc. of the VLDB Endowment, 2019, 13(3): 279–292. [doi: 10.14778/3368289.3368294]

[81] Yang ZH, Kamsetty A, Luan SF, Liang E, Duan Y, Chen X, Stoica I. NeuroCard: One cardinality estimator for all tables. Proc. of the VLDB Endowment, 2020, 14(1): 61–73. [doi: 10.14778/3421424.3421432]

[82] Sun J, Li GL. An end-to-end learning-based cost estimator. Proc. of the VLDB Endowment, 2019, 13(3): 307–319. [doi: 10.14778/3368289.3368296]

[83] Kipf A, Kipf T, Radke B, Leis V, Boncz PA, Kemper A. Learned cardinalities: Estimating correlated joins with deep learning. In: Proc. of the 9th Biennial Conf. on Innovative Data Systems Research (CIDR). Asilomar, 2019.

[84] Wang XY, Qu CB, Wu WY, Wang JN, Zhou QQ. Are we ready for learned cardinality estimation? Proc. of the VLDB Endowment, 2021, 14(9): 1640–1654. [doi: 10.14778/3461535.3461552]

[85] Abebe M, Lazu H, Daudjee K. Tiresias: Enabling predictive autonomous storage and indexing. Proc. of the VLDB Endowment, 2022, 15(11): 3126–3136. [doi: 10.14778/3551793.3551857]

[86] Trummer I. Database tuning using natural language processing. ACM SIGMOD Record, 2021, 50(3): 27–28. [doi: 10.1145/3503780.3503788]

[87] Trummer I. DB-BERT: A database tuning tool that "Reads the Manual". In: Proc. of the 2022 Int'l Conf. on Management of Data. Philadelphia: ACM, 2022. 190–203. [doi: 10.1145/3514221.3517843]

[88] Chen A. A review of emerging non-volatile memory (NVM) technologies and applications. Solid-state Electronics, 2016, 125: 25–38. [doi: 10.1016/j.sse.2016.07.006]

[89] Sukhwani B, Min H, Thoennes M, Dube P, Iyer B, Brezzo B, Dillenberger D, Asaad S. Database analytics acceleration using FPGAs. In: Proc. of the 21st Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT). Minneapolis: IEEE, 2012. 411–420.

[90] Owaida M, Sidler D, Kara K, Alonso G. Centaur: A framework for hybrid CPU-FPGA databases. In: Proc. of the 25th IEEE Annual Int'l Symp. on Field-programmable Custom Computing Machines (FCCM). Napa: IEEE, 2017. 211–218. [doi: 10.1109/FCCM.2017.37]

[91] Foley D, Danskin J. Ultra-performance Pascal GPU and NVLink interconnect. IEEE Micro, 2017, 37(2): 7–17. [doi: 10.1109/MM.2017.37]

[92] Liu Z, Wang B, Carpenter P, Li D, Vetter JS, Yu WK. PCM-based durable write cache for fast disk I/O. In: Proc. of the 20th IEEE Int'l Symp. on Modeling, Analysis and Simulation of Computer and Telecommunication Systems. Washington: IEEE, 2012. 451–458. [doi: 10.1109/MASCOTS.2012.57]

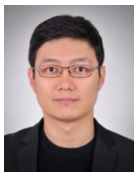
[93] Song YJ, Lee JH, Han SH, *et al*. Demonstration of highly manufacturable STT-MRAM embedded in 28 nm logic. In: Proc. of the 2018 IEEE Int'l Electron Devices Meeting (IEDM). San Francisco: IEEE, 2018.18.2.1–18.2.4. [doi: 10.1109/IEDM.2018.8614635]

[94] Chen YY. ReRAM: History, status, and future. IEEE Trans. on Electron Devices, 2020, 67(4): 1420–1433. [doi: 10.1109/TED.2019.2961505]

附中文参考文献:

[21] 阳振坤. OceanBase关系数据库架构. 华东师范大学学报(自然科学版), 2014(5): 141–148, 163.

王嵩立(1998－), 男, 硕士生, 主要研究领域为数据库系统, 查询优化, 机器学习.



张凯(1986－), 男, 博士, 副教授, CCF 专业会员, 主要研究领域为并行与分布式计算, 网络系统.

荆一楠(1978－), 男, 博士, 副教授, 博士生导师, CCF 专业会员, 主要研究领域为大数据分析, 时空数据管理, 数据库系统.

王晓阳(1960－), 男, 博士, 教授, 博士生导师, CCF 会士, 主要研究领域为大数据分析, 数据安全.

何震瀛(1977－), 男, 博士, 副教授, CCF 专业会员, 主要研究领域为数据库系统, 数据分析.