软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software, 2023,34(2):761−785 [doi: 10.13328/j.cnki.jos.006713]

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

HTAP 数据库关键技术综述*

张 超, 李国良, 冯建华, 张金涛

(清华大学 计算机科学与技术系, 北京 100084)

通信作者: 李国良, E-mail: liguoliang@tsinghua.edu.cn

摘 要: 混合事务与分析处理(hybrid transactional analytical processing, HTAP)技术是一种基于一站式架构同时处理事务请求与查询分析请求的技术. HTAP 技术不仅消除了从关系型事务数据库到数据仓库的数据抽取、转换和加载过程, 还支持实时地分析最新事务数据. 然而, 为了同时处理 OLTP 与 OLAP, HTAP 系统也需要在系统性能与数据分析新鲜度之间做出取舍, 这主要是因为高并发、短时延的 OLTP 与带宽密集型、高时延的 OLAP 访问模式不同且互相干扰. 目前, 主流的 HTAP 数据库主要以行列共存的方式来支持混合事务与分析处理, 但是由于该类数据库面向不同的业务场景, 所以它们的存储架构与处理技术各有不同. 首先, 全面调研HTAP数据库, 总结它们主要的应用场景与优缺点, 并根据存储架构对它们进行分类、总结与对比. 现有综述工作侧重于基于行/列单格式存储的 HTAP 数据库以及基于 Spark 的松耦合 HTAP 系统, 而这里侧重于行列共存的实时 HTAP 数据库. 特别地, 凝炼了主流 HTAP 数据库关键技术, 包括数据组织技术、数据同步技术、查询优化技术、资源调度技术这 4 个部分. 同时总结分析了 HTAP 数据库构建技术与评测基准. 最后, 讨论了 HTAP 技术未来的研究方向与挑战.

关键词: HTAP 数据库; 行列共存; 数据组织; 查询优化; 数据同步; 资源调度

中图法分类号: TP311

中文引用格式: 张超, 李国良, 冯建华, 张金涛. HTAP 数据库关键技术综述. 软件学报, 2023, 34(2): 761–785. http://www.jos.org.cn/1000-9825/6713.htm

英文引用格式: Zhang C, Li GL, Feng JH, Zhang JT. Survey of Key Techniques of HTAP Databases. Ruan Jian Xue Bao/Journal of Software, 2023, 34(2): 761−785 (in Chinese). http://www.jos.org.cn/1000-9825/6713.htm

Survey of Key Techniques of HTAP Databases

ZHANG Chao, LI Guo-Liang, FENG Jian-Hua, ZHANG Jin-Tao

(Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China)

Abstract: Hybrid transactional analytical processing (HTAP) relies on a single system to process the mixed workloads of transactions and analytical queries simultaneously. It not only eliminates the extract-transform-load (ETL) process, but also enables real-time data analysis. Nevertheless, in order to process the mixed workloads of OLTP and OLAP, such systems must balance the trade-off between workload isolation and data freshness. This is mainly because of the interference of highly-concurrent short-lived OLTP workloads and bandwidth-intensive, long-running OLAP workloads. Most existing HTAP databases leverage the best of row store and column store to support HTAP. As there are different requirements for different HTAP applications, HTAP databases have disparate storage strategies and processing techniques. This study comprehensively surveys the HTAP databases. The taxonomy of state-of-the-art HTAP databases is introduced according to their storage strategies and architectures. Then, their pros and cons are summarized and compared. Different from previous works that focus on single-model and spark-based loosely-coupled HTAP systems, real-time HTAP databases with a row-column dual store are focused on. Moreover, a deep dive into their key techniques is accomplished regarding data organization, data synchronization, query optimization, and resource scheduling. The existing HTAP benchmarks are also introduced. Finally, the research challenges and open problems are discussed for HTAP.

Key words: HTAP databases; row and column; data organization; query optimization; data synchronization; resource scheduling

* 基金项目: 国家自然科学基金(61925205, 62072261, 62232009)

收稿时间: 2022-02-18; 修改时间: 2022-05-08; 采用时间: 2022-05-28; jos 在线出版时间: 2022-07-22

随着现代社会中各类大规模实时分析应用的出现，许多的业务场景需要我们既能处理高并发的事务请求，又能够对最新数据作实时分析. 传统的数据处理流程是从在线事务处理(on-line transaction processing, OLTP)到数据 ETL(extract-transform-load, 抽取-转换-加载)这样的过程，再到在线分析处理(on-line analytical processing, OLAP)的数据处理流程，但这样的流程耗时耗力，并且分析的数据往往已经过时，不能及时挖掘出有用信息[1]. 如图 1 所示：与传统的将 OLTP 与 OLAP 分离的系统架构不同，混合事务与分析处理(hybrid transactional analytical processing, HTAP)技术主要基于一站式架构混合处理 OLTP 和 OLAP 的负载，这省去了传统架构中的 ETL 过程. 而且该项技术使得组织机构和企业可以在数据密集型应用中分析最新事务数据，从而做出及时且准确的决策[1,2]. 例如，通过采用 HTAP 技术，在线电子商务企业可以实时地分析最新事务数据，识别出未来销售趋势后及时地采取对应的在线广告投放活动. 大型银行和金融系统可以在处理高并发交易事务的同时，检测出异常的欺诈交易. 这既提供了高质量交易服务，又保障了交易安全.

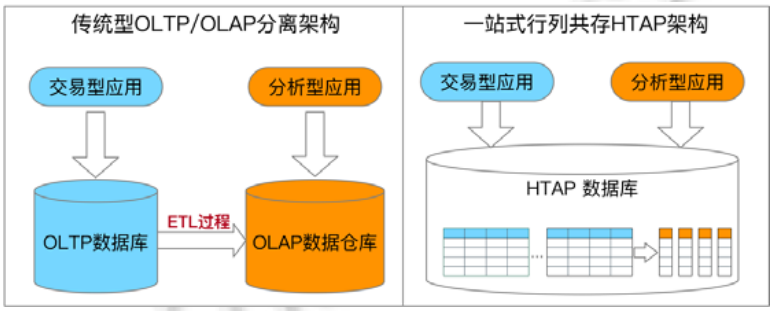


图 1　传统 OLTP/OLAP 分离架构与一站式 HTAP 架构对比

HTAP 数据库技术是数据库领域的一个研究热点，工业界和学术界都在研究相关系统和技术. 2014 年，著名的咨询公司 Gartner 针对该类场景在报告中首次提出了 HTAP 的概念[1]，其认为, HTAP 技术将对未来商业应用产生很多新的机会，并预测：到 2024 年, HTAP 技术将在数据实时分析领域得到广泛采用. 在 HTAP 这一概念被正式提出之后，许多 HTAP 数据库应运而生，其中有传统数据库厂商通过扩展行存引擎或列存引擎实现，如 Oracle[3]、SQL Server[4]、DB2[5]、SAP HANA[6]等；也有各种专门面向 HTAP 的数据库系统被提出，如 TiDB[7]、*F*1 Lightning[8]、MySQL Heatwave[9]等. 这些 HTAP 数据库的主要共同点是，都以行列存储共存的方式来高效地处理混合负载. 这主要是因为行存引擎更适用于高并发的行级别事务处理，列存引擎更有利于分析查询处理，特别是对于查询只需要读取数据表的部分列的情况. 然而，由于不同 HTAP 数据库面向不同的应用场景，所以这些系统的存储架构和处理方式各有不同. 例如：对于银行、金融类应用，由于它们对 OLTP 性能与数据分析实时性要求都很高，用户可采用基于内存的单机 HTAP 数据库，以同时保证高性能的事务处理与实时数据分析；针对大型电子商务等应用，用户可采用基于磁盘的分布式 HTAP 数据库，以保证系统能够弹性地水平扩展. 然而，由于高并发、短时延的 OLTP 负载与带宽密集型、高时延的 OLAP 负载的访问模式不同，且它们互相干扰，HTAP 数据库必须在负载隔离性与数据分析新鲜度之间做出取舍：一方面，在同一台机器上处理混合负载会造成 OLTP 与 OLAP 的隔离性很低，但分析引擎能够快速读取最新事务数据，因此数据分析的新鲜度很高；另一方面，在不同机器上分别处理 OLTP 与 OLAP 有很高的负载隔离性，但是由于数据同步的延迟会造成分析的数据可能不是最新的. 因此，需要根据具体的应用场景选择 HTAP 数据库的存储架构.

当前, HTAP 数据库技术主要面临 4 个方面的挑战，包括: (1) 数据组织问题; (2) 数据同步问题; (3) 查询优化问题; (4) HTAP 资源调度问题.

(1) 数据组织问题是关于如何根据以往的 HTAP 负载适应性地组织数据，以优化系统的处理性能及降低存储代价. 针对数据组织，目前主要分为基于主行存储的内存列选择[9−11]与基于负载驱动的行列混合存储[12−15]这两大类技术路线：前者基于人工与简单规则[9,10]，或者基于独立性假设的整型规划算法[11]进行列选择，然而这些选择方案没有考虑多列的访问与查询性能之间的关系，因此未来需要研

究考虑多列效用关系的列选择方法; 后者通过分析历史负载的执行情况建立代价函数, 然后动态地调整行列混合存储. 关系表可被横向或纵向地划分, 以减小查询访问的数据量. 例如, Peloton[14]采用灵活模式在逻辑上将关系表划分为不同的列组, 其基于负载驱动的方法将经常访问的多个属性组合成列组以加速查询. 然而, 该类技术不仅增加了系统在事务处理与查询处理层面的复杂度, 还使得许多以往的查询优化规则失效, 目前还未被任何商用系统采用. 该类技术未来需要解决如何在不修改原有查询处理层与优化层的基础上自适应地组织数据.

(2) 数据同步问题需要将最新的事务数据更新到列存储中, 以保证数据分析的高新鲜度. 但该问题的一个主要难点在于: 如何在保证高新鲜度的同时, 尽量减小数据同步对系统造成的性能影响. 目前的数据同步方法主要基于增量存储进行批量定时更新, 包括基于阈值的同步[3,5,9]、基于两阶段迁移的同步[4]、基于字典合并的同步[6]以及基于增量日志的同步[7,8]. 然而, 目前的同步方法只采用定时更新的策略, 未来需要研究基于负载趋势预测的自适应更新时间窗口选择, 以兼顾数据分析新鲜度与系统整体的处理性能.

(3) 查询优化问题涉及到: (i) 如何为查询选择存储访问路径(行存或列存); (ii) 如何利用新硬件如异构 CPU/GPU 处理器进行负载加速; (iii) 如何建立索引高效地支持 HTAP. 第(i)个问题的挑战主要在于平衡计划搜索的代价与查询的执行性能, 目前主要有基于规则[3,10]和基于代价函数[4,7]两种方法: 基于规则的方法(如先做列式扫描, 缺省的列再做行式扫描)搜索的计划空间有限, 有可能选择次优的访问路径; 基于代价函数的方法依赖于独立性假设与均匀分布假设, 无法应对关联数据与倾斜分布的情况. 因此, 未来需要研究能够动态搜索计划空间并考虑复杂数据分布的方法. 第(ii)个问题的方法[16,17]目前主要使用 CPU 处理事务以及利用 GPU 加速查询, 其主要挑战在于如何很好地隔离事务处理, 并使得 GPU 分析最新数据. 第(iii)个问题需要解决的挑战主要是: 如何针对并发事务高效地更新索引, 并且能够同时利用索引加速查询. 目前方法主要分为基于路径复制[18]的索引与多版本索引技术[19].

(4) HTAP 资源调度问题是指合理地为系统中执行 OLTP 负载的实例和执行 OLAP 负载的实例分配 CPU、内存等资源, 使得系统性能最大化. 目前, 资源调度方法可分为基于负载执行的调度算法[20,21]和基于数据新鲜度驱动的调度算法[22]. 基于负载执行的技术通过监控当前混合负载的执行情况动态地调度系统资源给执行负载. 例如:

- Psaroudakis 等人[20]提出, 通过启动新线程为阻塞队列重新执行任务, 以充分利用 CPU 资源;
- Sirin 等人[21]提出, 通过在第 3 级共享缓存, 划分出更多的资源给 OLAP 隔离执行, 以提高其吞吐量;
- Raza 等人[22]提出了基于数据新鲜度驱动的算法, 该算法以给定新鲜度阈值为基准, 在不同执行方式中进行切换: 首先, 隔离执行混合负载, 当数据新鲜度小于给定阈值时, 则切换到共享最新数据的执行方式.

目前, 针对 HTAP 的资源调度算法没有同时考虑负载特点和系统行为趋势的特点, 所以无法提前按趋势进行资源调度, 系统性能可能出现抖动. 因此, 未来可考虑基于历史统计信息建模, 通过机器学习技术来进行资源调度, 以提高系统的稳定性并保证数据分析的新鲜度.

本文将系统地对 HTAP 数据库系统的现有工作进行调研与综述.

- 首先, 根据存储架构的不同, 将主流 HTAP 数据库的存储架构分为四大类, 包括: (1) 主行存与内存型列存; (2) 分布式行存与列存副本; (3) 单机磁盘型行存与分布式列存; (4) 主列存与增量型行存; 同时, 介绍了每一类的代表性系统, 比较了不同的存储架构在处理 OLTP 与 OLAP 负载时的优缺点(见第 1 节).
- 其次, 详细介绍了 HTAP 数据库的关键技术, 包括数据组织技术、数据同步技术、查询优化技术、资源调度技术这 4 个部分(见第 2 节).

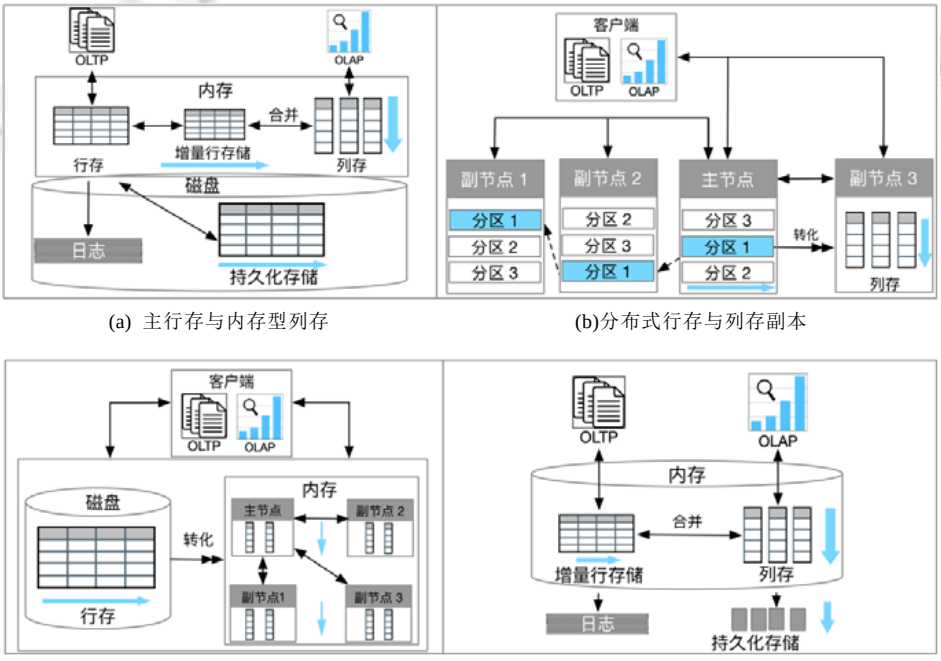
- 同时，还介绍了面向 HTAP 数据库的构建技术(见第 3 节)与评测基准(见第 4 节)，包括数据生成、执行规则以及评价指标.
- 最后，本文对 HTAP 技术的未来研究方向进行展望，包括自动化内存列选择、面向 HTAP 的学习型查询优化器、自适应的 HTAP 资源调度以及面向 HTAP 的评测基准套件这 4 个方面(见第 5 节).

现有综述工作[1,2]侧重于行存型 HTAP 数据库[23,24]、列存型 HTAP 数据库[16,17,38]以及基于 Spark[25]的松耦合 HTAP 系统[26,27](详见 1.6 节). 本文主要关注于支持完整 ACID 事务特性和行列共存的 HTAP 数据库系统. 本文从存储架构的角度，对近 10 年的 HTAP 数据库系统进行了分类、总结与对比，并从数据组织、查询优化、数据同步、资源调度这 4 个方面详细介绍了 HTAP 数据库的关键技术. 本文还介绍了面向 HTAP 数据库的构建技术与基准评测系统，讨论了当前 HTAP 技术的挑战，并对未来研究方向进行了展望.

1 HTAP 数据库的分类与特点

从广义上来说，只要同时支持 OLTP 和 OLAP 的数据库系统，都应属于 HTAP 数据库系统的范畴. 例如，大数据平台 Splice Machine (https://splicemachine.com/)分别包含了针对事务处理的 HBase[28]数据库引擎和针对数据分析的 Spark[25]引擎，然而这些面向大数据处理的 HTAP 系统往往涉及到数据的 ETL 过程，不能满足实时事务处理和分析处理的要求，因此它们不能算严格意义上的 HTAP 数据库. 另外，基于单模型存储(行存或列存)的 HTAP 数据库往往不能同时兼顾 OLTP 和 OLAP 的性能. 众所周知：行存可以利用索引技术面向写优化，列存可以扫描少量列以加速查询. 近年来，主流 HTAP 数据库都已支持行列共存. 因此，本文的研究范围是支持行列共存的 HTAP 数据库. 该类系统不仅能够并发地、高效地支持事务处理，而且能够在线分析最新的事务数据.

本文主要按照存储策略对 HTAP 数据库进行分类，并针对每一类对它们的主要方法进行介绍. 如图 2 所示，对于主流 HTAP 数据库的存储架构，HTAP 数据库可分为四大类：(a) 主行存与内存型列存；(b) 分布式行存与列存副本；(c) 单机磁盘型行存与分布式列存；(d) 主列存与增量型行存.



(a) 主行存与内存型列存　(b)分布式行存与列存副本

(c) 单机磁盘型行存与分布式列存　(d) 主列存与增量型行存

图 2　HTAP 数据库的 4 类存储架构图

针对每一类系统的存储架构, 表 1 列出了对应的代表性系统, 总结并比较了它们针对 OLTP 和 OLAP 处理的性能分级、扩展性分级以及负载隔离性和数据新鲜度的分级. 其中, 每个分级由高、中、低这 3 个等级衡量, 数据新鲜度被定义为当前时刻 OLAP 已经更新的元组数目与 OLTP 的所有更新的元组数目的比率[29].

表 1 面向 HTAP 数据库的存储架构分类与性能特点比较

存储架构	代表性系统	OLTP		OLAP		隔离性	新鲜度
		性能	扩展性	性能	扩展性		
主行存与内存型列存	Oracle[3], SQL Server[4], DB2 BLU[5]	高	中	高	低	低	高
分布式行存与列存副本	TiDB[7], *F*1 Lightning[8]	中	高	中	高	高	低
单机磁盘型行存与分布式列存	MySQL Heatwave[9]	中	中	高	高	高	中
主列存与增量型行存	SAP HANA[6]	中	低	高	中	低	高

第(a)类架构利用内存优化的主行存与内存型列存支持 HTAP, 因此 OLTP 和 OLAP 的处理性能都很高, 而且 OLAP 能够快速从增量行存储访问最新数据. 但该类架构基于单机处理混合负载, 因此扩展性与隔离性都不高, 尤其是内存型列存往往不支持持久化, 因此 OLAP 的扩展性很低. 第(b)类架构基于磁盘型分布式行存储与列存储支持 HTAP, 因此扩展性与隔离性都很高, 但是由于分布式系统需要保证数据一致性, 其性能比第(a)类架构处理性能要低. 另外, 由于需要从增量日志中合并读取最新数据, 其新鲜度也相对较低. 第(c)类架构基于分布式内存型列存支持查询分析, 因此 OLAP 的性能、扩展性、隔离性都很高, 但是由于需要读取事务处理节点的最新缓存数据, 所以新鲜度偏低. 其基于单机磁盘型行存处理事务, 因此性能比第(a)类要低, 且扩展性比第(b)类要低. 第(d)类架构基于内存型列存处理 OLAP, 其从增量行存中读取最新数据, 因此分析性能与新鲜度很高, 其列存可以持久化, 因此扩展性比第(a)类要高. 其利用缓存的增量行存储处理事务, 因此 OLTP 的扩展性与隔离性偏低.

1.1 基于主行存与内存型列存的HTAP数据库

如图 2(a)中架构图所示, 该类架构下的系统以行存为主存储, 列存为内存型存储. 行存引擎主要针对 OLTP 进行优化, 负责高并发事务处理与写日志等操作, 并持久化数据在磁盘中. 同时, 其在数据更新时会将更新的记录同步到基于内存的增量行存储中. 列存引擎主要面向 OLAP 进行优化, 其数据按列组织, 因此只需扫描对应的列数据以响应查询. 该类系统针对 OLTP 和 OLAP 都有很高的吞吐量, 这是因为事务与分析操作基于内存进行快速处理. 另外, 数据新鲜度也很高, 因为列存引擎能够同时访问最新数据. 然而受限于内存容量, 该类系统针对 OLTP 与 OLAP 的扩展性不高. 另外, 由于事务处理与分析操作都在同一台机器上进行, 系统处理混合负载的隔离性不高. 因此, 其在并发处理 OLTP 和 OLAP 负载时会产生对 CPU、共享缓存等资源的争抢而影响整体性能. 目前, 该类系统需要解决的问题是在保持高性能、高新鲜度的同时也要提高系统的扩展性与隔离性. 一种可能的解决方案是, 利用分布式内存技术将 OLTP 和 OLAP 负载分布在多台机器的内存中进行处理, 但这种方案面临的主要难点是如何根据负载划分行存的数据与列存的数据, 以保持高性能的混合负载处理与高数据分析新鲜度. 另外, 如何划分和置放增量行存储(在行存节点或是列存节点)也是一个需要考虑的问题. 基于主行存与内存型列存的 HTAP 数据库包括 DB2[5]、Oracle[3]、SQL Server[4]. 这些系统的主要差异在于, 内存列存储是否支持持久化与更新. 比如: DB2 和 SQL Server 都能将内存中的列数据持久化到磁盘并加以更新, 并且 SQL Server 还能够对压缩后的列存数据通过冷热数据划分后进行更新; 而 Oracle 不支持列存储的持久化与更新.

DB2 数据库在原有行存关系数据库的基础上, 结合 BLU 内存加速器针对 HTAP 负载进行优化. 其基于行存的关系数据库支持完整的 ACID 特性, 并可面向内存优化写操作. BLU 是 DB2 的列存加速器名称, 它通过基于频率的字典压缩、内存优化、SIMD (single-instruction, multiple data, 单指令、多数据)等技术提升分析性能[5]. Oracle 系统[3]在自己原生的行存上集成了基于内存的列存储. 行存储通过缓存进行数据的更新、点查询和短范围查询. 列存储可通过压缩编码和 SIMD 等技术进行查询加速. 同时, 最新的更新数据会被记录到增量行存储中, 以便列存引擎访问最新数据. SQL Server 系统通过在内存表上建立列存索引的方式来加速分析型

查询. 本质上, 在内存中的列存索引也是另一种形式的列存, 因为它包含了该列的所有数据[30]. SQL Server 在原有磁盘型行存引擎的基础上集成了基于内存的 Hekaton 行存引擎[31]和基于内存的 Apollo 列存引擎[4], 其基于内存的行存引擎不仅可以加速事务处理, 而且会将最新的更新保留在内存表的尾部作为增量行存储. 这样, 列存引擎也能够同时访问增量最新数据. SQL Server 的列索引分为聚集列索引和非聚集列索引: 聚集列索引包含了所指向内存表的所有列数据, 非聚集列索引只包含部分列数据. 列索引将数据以 100 万行为一个单位分组后, 分段进行压缩索引. 所有列分段和压缩字典的元数据被写入磁盘中进行保存, 元数据可以帮助所有分段组成一个完整列. 被频繁使用的列索引会被放入内存中以减少磁盘 I/O, 不频繁使用的列索引则放到外存中.

1.2 基于分布式行存与列存副本的HTAP数据库

如图 2(b)中的架构图所示, 该类 HTAP 数据库以分布式架构支持混合事务与分析处理. 其分布式行存为主存储, 列存为只读副本, 主节点在处理事务时写入日志, 并异步式地向其他节点服务器发送最新日志. 其通过分布式协议进行事务处理. 其中, 有部分节点会被选为列存节点, 负责加速复杂查询. 特别地, 只有行存节点参与事务协议, 列存节点不参与分布式事务投票. 当事务成功提交后, 主节点将最新日志复制到列存节点, 然后, 列存节点将行数据转换成列数据后写入持久化存储. 由于事务和查询分别在行存节点和列存节点上处理, 该类系统具有高度的负载处理隔离性. 另外, 由于完全基于分布式架构, 该类系统面向 OLTP 和 OLAP 的扩展性都很高. 然而, 相比于单机的 HTAP 架构, 基于分布式的架构需要不断通信以保持不同节点间的数据一致性, 因此会产生较大延迟, 所以, 该类架构针对 OLTP 与 OLAP 的处理性能不高. 另外, 由于需要通过日志回放的方式同步行存节点与列存节点之间的数据, 其在列存节点上进行数据分析时, 可能有最新数据还未被转换合并到持久化的列存储, 因而其数据分析的新鲜度偏低. 该类系统需要解决的问题是, 在保持高扩展性、高隔离性的同时提高系统的新鲜度与处理性能. 解决该问题的难点在于: 如何高效地合并增量数据, 并使得列存引擎能够快速访问最新数据. 基于分布式行存与列存副本的 HTAP 数据库包括 TiDB[7]、*F*1 Lightning[8]等分布式数据库. 该类系统的主要差异在于分布式事务处理技术与节点间的数据同步方式. 例如: TiDB 与 *F*1 Lightning 采用两阶段提交协议进行分布式事务处理, 同时结合 Paxos[32]一致性协议进行异步式的数据副本之间同步, 并将行存副本转换为列存副本. 该方法能够保证数据的强一致性与系统的高可用性, 但事务处理时延较长. SingleStore (https://www.singlestore.com/)采用主备式且基于内存的分布式事务处理, 这种方法能够快速处理单机事务, 但数据同步的时延较长, 且其存在单点故障问题, 系统的扩展性较低且恢复时间较长.

TiDB 基于两阶段提交协议与扩展的 Raft 协议[33]进行分布式事务处理与数据同步. Raft 协议定义节点有 3 种角色: 领导者节点、跟随者节点以及候选者节点. 其中, 领导者节点负责接收读写操作, 并在本地进行写操作时, 异步地将数据同步给跟随者, 如果领导者出现故障, 将由跟随者在候选者中投票选出新的领导者. TiDB 在原有 Raft 协议的基础上增加了学习者节点[8], 其作为列存节点, 且不参与投票. 领导者节点在向跟随者节点同步数据时, 也向学习者节点异步地存储一个列存副本. 通过建立一个增量列式树, 列存节点可定期地将更新合并到列存副本中, 以保持列存数据的高新鲜度. TiDB 的 SQL 引擎针对分布式行存储, 通过两阶段提交的方式实现了可重复读的事务隔离级别. 除此之外, TiDB 实现了一个基于代价模型的查询优化器. 根据代价估计结果, 系统可决定查询计划在行存或列存中执行, 或是行列混合执行. *F*1 Lightning 基于 Spanner[34]数据库支持分布式事务, 结合 Paxos 协议[32]保证分布式行存的数据一致性; 其 Lightning 服务器管理增量存储, 通过一个名为变化泵(changepump)[8]的模块监测最新的数据变化; 如果有事务更新, 则将更新写入内存型增量日志文件, 然后定期合并多个增量日志文件并将结果持久化到磁盘, 且持久化的数据会被转换为列式格式. 其查询引擎支持算子下推到列存储进行加速, 如果查询的数据超过了其列存数据的时间窗口(最近的 10 个小时), 查询引擎将在 Spanner 中读取历史数据. SingleStore 是一个基于 MemSQL 开发的内存型分布式数据库, 其基于行存储的分布式架构, 在部分节点持久化时写入列存副本. 列存副本可用于加速复杂查询. 其采用主备式的分布式事务处理, 在主节点处理完事务后, 将数据更新再同步到其他副节点.

1.3 基于单机磁盘型行存与分布式列存的HTAP数据库

如图2(c)中的架构图所示, 该类系统以磁盘型行存数据库为基础, 并集成了分布式内存型列存引擎. 主行存负责事务处理, 分布式列存引擎则负责加速分析型查询. 最新的事务数据将被实时地更新到分布式列存中, 以实现行存与列存之间的数据同步. 该类系统通过行存型增量缓存实现列存数据更新, 所以相较于日志型的数据同步方式, 有更高的新鲜度. 但是由于需要同时更新在不同机器上的列存数据, 相较于在同一机器上的内存更新, 其具有更低的新鲜度和更高的隔离性. 因为其基于分布式列存储, 所以该类系统有更高的 OLAP 吞吐量和更好的扩展性. 针对 OLTP 负载, 其基于单机的行存数据库可提供适中的吞吐量和扩展性. 该类系统需要解决的主要问题是: 当面临大规模更新时, 由于 OLTP 的更新数据频繁地同步到分布式列存中, 会极大地影响 OLAP 的分析性能, 因此需要设计延迟更新机制以保证性能. 然而, 延迟地更新列存数据会影响到系统的数据分析新鲜度, 因此, 如何在分析性能与新鲜度之间做出合理的取舍是一个挑战.

以 MySQL Heatwave[9]为例, 其基于 MySQL 的系统结合云服务 Heatwave 提供了一个 HTAP 的整体解决方案. 原有的 MySQL 数据库负责事务处理, 紧密集成的 Heatwave 云服务通过基于内存的分布式列存引擎来加速复杂查询. 此外, 分析型查询由系统查询引擎基于代价估计后决定是否下推到内存列引擎中执行, 常被访问的热数据将会留在内存中, 不常使用的冷数据将会被压缩后持久化到磁盘中. MySQL 的事务更新定期(默认为 200 ms 的间隔)向内存中的分布式列存同步, 以保证数据分析的新鲜度. 该系统的一个新特性是面向 HTAP 的自动驾驶舱服务, 它利用机器学习技术自动地优化底层存储, 包括自动压缩列存、自动内存列数据管理以及自动行存储与列存储的数据同步等.

1.4 基于主列存与增量型行存的HTAP数据库

如图 2(d)中的架构图所示, 该类系统以内存型主列存为基础, 并结合了增量行存支持 HTAP. 主列存引擎主要处理 OLAP 查询负载, 增量行存引擎负责 OLTP 事务处理, 并直接将更新数据定期合并到主列存中. 该类系统因为以列存储为主, 所以 OLAP 的处理性能很高, 且增量行存直接与列存连接, 数据分析的新鲜度也很高. 但其缺点也很明显, 就是面向增量行存的 OLTP 处理性能中等, 扩展性与负载隔离性很低. 该类系统需要解决的主要问题是: 在保证 OLAP 分析的高性能与高新鲜度需要的同时, 提高 OLTP 处理的性能与扩展性. 然而, 解决该问题的难点是, 在同一台机器上处理 OLTP 负载与 OLAP 负载会互相干扰. 因此, 同时保证系统的 OLAP 与 OLTP 的高性能具有很大的挑战性.

以 SAP HANA[6,35,36]为例, 其列存储引擎可通过压缩、向量执行以及 OLAP 多维分析等技术进行查询加速. 另外, 其利用了 CPU 的多级缓存机制来优化事务处理. 其中, *L*1 缓存被用于存储写操作数据, 在 *L*1 缓存中, 数据以行存形式保存, 数据也会同时被写到重做日志里进行持久化; *L*2 缓存被用于合并 *L*1 缓存的数据以及批量导入的数据, 在 *L*2 缓存中, 数据以列存形式保存; 当 *L*2 缓存的数据到达一定数量时, 会被合并到列式主存中; 最后, 列主存用于查询加速与数据的持久化.

1.5 不同HTAP数据库架构适用场景

上述 4 类 HTAP 数据库架构各有优缺点, 那么如何选择它们, 取决于用户面对的具体应用场景. 总的来说, 用户可遵循以下原则.

(1) 对于面向高性能、高新鲜度且扩展性要求不高的 HTAP 应用, 如银行交易与分析系统, 可选择第(a)类架构的系统(同步延迟约为 1 ms–100 ms)[3–5]. 一方面, 该类银行应用需要高效地处理客户的事务请求并需要及时地产生分析报表, 因此, 其对事务处理性能和实时数据分析性能的要求都很高; 另一方面, 该类应用主要面向固定银行客户群体, 对扩展性要求不高;

(2) 对于面向高扩展性、可容忍一定的数据同步延迟的 HTAP 应用, 如大型在线电商交易与分析平台, 可选择第(b)类架构的系统(同步延迟约为 100 ms–10 s)[7,8]. 该类 HTAP 应用面向大规模用户的并发事务请求, 对扩展性要求很高. 另外, 该类应用对新鲜度要求没有第(1)类高, 可接受一定时间内数据不一致的情况;

(3) 对于面向高性能与高扩展性的 OLAP、高隔离性的 HTAP 应用, 如大型物联网系统, 可选择第(c)类架构的系统(同步延迟约为 200 ms)[9]. 该类应用主要侧重于 OLAP 数据分析. 由于需要分析大规模的实时物联网信息, 其对数据分析的性能和扩展性要求很高, 而且还需要系统有实时采集与分析新数据的能力;

(4) 对于面向高性能的 OLAP、高新鲜度且扩展性要求不高的 HTAP 应用, 如金融反欺诈系统, 可选择第(d)类架构的系统(同步延迟约为 1 ms–10 ms)[6,37]. 该类应用需要在处理事务的同时进行反欺诈检测, 其侧重于高新鲜度且高性能的 OLAP 分析, 对事务的性能要求没有第(1)类高, 对扩展性的要求也没有其他 3 类高.

1.6 其他类型的HTAP系统

由于行存更适用于事务处理而列存更适用于分析处理, 本文侧重于总结与比较行列共存型 HTAP 数据库, 还有一些其他类型的 HTAP 系统没有在表 1 中进行比较, 包括: (1) 行存型 HTAP 系统; (2) 列存型 HTAP 系统; (3) 基于 Spark 的 HTAP 系统. 行存型 HTAP 系统的主要问题是没有基于列存加速 OLAP 分析, 所以 OLAP 的分析性能相比于行列共存的 HTAP 系统会有数量级的差距. 列存型 HTAP 系统基于单机的内存型列存处理事务, 因此事务的性能和扩展性都不高. 基于 Spark 的 HTAP 系统主要缺点是其 OLTP 系统只能支持简单的事务更新, 并且基于 Spark 的 OLAP 系统需要读取外部文件来进行数据分析, 同时需要经过特定的 ETL 过程准备数据, 所以数据的新鲜度很低, 无法满足实时事务处理和分析处理的要求.

(1) 行存型 HTAP 系统. 该类系统[23,24]基于主备架构的内存型行存储支持 HTAP. 比如: 早期版本的 Hyper[23]基于操作系统的写复制(copy-on-write)技术来同步一份行快照用于支持 OLAP 的分析, 主线程负责处理 OLTP 请求, 这样, 系统能够使用不同的线程分别处理 OLAP 和 OLTP 负载. 但是由于系统基于单机单线程处理事务, 可扩展性不高, 且其基于行存的分析性能也不高, 后期版本的 Hyper[38]已经支持主列存与增量型行存的架构. 除此之外, BatchDB[24]也采用一个基于内存的主备行存储来支持 HTAP, 其主要方法是通过主行存处理 OLTP, 副本行存处理 OLAP, 然后定期地将主行存的数据更新批量同步到副行存.

(2) 列存型 HTAP 系统[16,17,38]. 该类系统基于主备的内存型列存储的方式支持 HTAP. 比如: Hyrise[39]系统利用一种可伸缩的列存储方式支持 HTAP, 其基本方法是: 划分比较窄的数据分区(即少量数据列)处理 OLAP 负载, 因为这有利于顺序扫描; 然后划分比较宽的数据分区(即大多数列)来处理 OLTP 负载. RateupDB[17]与 Caldera[16]采用 CPU/GPU 的异构处理器架构来支持 HTAP. 其中, RateupDB 基于主备列存结合增量同步的方式支持 HTAP, 这与 BatchDB 的技术类似, 但 RateupDB 是基于列存储的; 而 Caldera 采取写复制的技术基于主备列存支持 HTAP, 这与 Hyper 采用的技术类似, 但 Caldera 是基于列存储的.

(3) 基于 Spark 的 HTAP 系统. 该类系统[26,27]通过结合独立的 OLTP 系统和 OLAP 系统支持 HTAP, 其中, OLTP 系统以列式数据库为主(如 HBase[28]), OLAP 系统主要采用的是 Spark[25]系统, 并且该类系统的数据都统一存储在分布式文件系统里(如 HDFS[40]). 例如: Splice Machine 基于 HBase 数据库实现数据的更新和事务处理, 并结合 Spark 进行数据分析[26]. SnappyData[27]系统集成了事务型系统 Apache Gemfire 和 Spark 系统用于同时处理流式、事务型以及交互式分析的应用. Wildfire[26]系统底层基于 Parquet[41]的统一列式存储, 上层统一使用 Spark 与大数据应用进行交互, 中间层增加了事务引擎, 结合 Spark 执行器支持 HTAP.

2 HTAP 数据库关键技术

总体来说, HTAP 数据库的关键技术, 包括数据组织技术、查询优化技术、数据同步技术、资源调度技术这 4 个部分. 表 2 列出了具体每一类技术的关键技术, 并针对 HTAP 负载处理总结了它们主要的优缺点. 其中, 数据组织技术是 HTAP 数据库基于负载组织数据存储形式(行存/列存/混合存储)的技术(见第 2.1 节), 数据同步

技术是 HTAP 数据库同步行存储数据与列存储数据的技术(见第 2.2 节), 查询优化技术是 HTAP 数据库基于行列共存的引擎优化查询分析的技术(见第 2.3 节), 资源调度技术是关于如何基于负载驱动以及基于新鲜度驱动进行资源调度的技术(见第 2.4 节).

表 2 HTAP 关键技术总览与优缺点比较

HTAP 技术类别	关键技术	代表性工作	主要优点	主要缺点
数据组织技术	基于主行存的内存列选择	MySQLHeatwave[9], Oracle[10], Ref.[11]	事务性能高	分析性能低
	基于负载驱动的行列混合存储	Refs.[12−15]	存储代价低	系统复杂度高
数据同步技术	基于内存增量表与内存型列存的数据同步	Oracle[3], SQL Server[4], SAP HANA[6]	性能高	扩展性低
	基于增量日志与持久化列存的数据同步	TiDB[7], *F*1 Lightning[8]	扩展性高	合并代价高
查询优化技术	混合行/列存储扫描	TiDB[7], Oracle[3] SQL Server[4]	分析性能高	搜索空间大
	异构 CPU/GPU 硬件加速	RateupDB[17], Caldera[16]	分析性能高	事务性能低
	面向 HTAP 负载的索引技术	Refs.[18,19]	事务性能高	内存空间大
资源调度技术	基于负载驱动的资源调度	Refs.[20,21]	性能高	新鲜度低
	基于新鲜度驱动的资源调度	Ref.[22]	新鲜度高	性能不高

由于在 HTAP 数据库中维护全量同步的行存储与列存储代价太大, 因此需要采取适应性的数据组织技术. 该类技术根据负载适应性地组织行列存储, 以使用相对少的存储代价高效地处理混合负载. 已有的关键技术主要分为两大类: (1) 基于主行存的内存列选择[9−11]; (2) 基于负载驱动的行列混合存储[12−15]. 第(1)类技术维护行存储的全量数据, 其关键技术是基于负载自动化地选择列数据到内存中进行查询加速, 主行存数据库能够高效地处理事务, 但当内存列未被查询命中时, 分析性能会有所下降. 第(2)类技术基于混合负载的访问模式动态地横向或纵向地划分各个关系表, 以生成行列混合存储. 其主要优点是数据冗余低、存储代价低. 但是由于底层的混合存储需要在查询处理层与查询优化层作相应改动, 系统的复杂度很高.

在 HTAP 数据库中, 数据同步技术指的是将增量数据高效地转换、合并到列存储的技术. 根据增量存储的类型(内存增量表/增量日志)以及列存储的类型(内存型/磁盘型), 本文将其分为两大类关键技术: (1) 基于内存的增量数据合并[3,4,6,9]; (2) 基于日志的增量数据合并[7,8]. 第(1)类技术包括基于阈值的同步[3,5,9]、基于两阶段迁移的同步[4]、基于字典合并的同步[6]. 该类技术基于内存进行数据同步, 所以性能高, 但可容纳的增量数据有限, 扩展性低. 第(2)类技术[7,8]主要结合 B+树与增量树进行多阶段的增量数据合并, 然后将数据转换为持久化列存, 该类技术扩展性很高, 但合并代价高.

HTAP 数据库的查询优化技术主要涉及到如何为查询选择优化的数据访问路径(行存或列存)、如何利用新硬件进行查询加速以及如何建立索引使得查询高效地访问最新的事务数据. 因此, 本文将其分为 3 类技术: (1) 混合行/列存储扫描[3,4,7]; (2) 异构 CPU/GPU 硬件加速[16,17]; (3) 面向 HTAP 负载的索引技术[18,19]. 第(1)类技术通过为查询算子选择适合的数据访问路径以加速查询处理, 该项技术的优点是可以利用行列存储各自的特点获得更好的分析性能, 缺点是搜索空间很大. 第(2)类技术利用异构 CPU/GPU 处理器进行查询加速, 其利用 CPU 任务并行和 GPU 数据并行的特点, 分别处理 OLTP 事务和 OLAP 分析, 但该类技术的主要优点是分析性能高, 而事务处理的性能由于资源的抢占会下降. 第(3)类技术基于索引技术支持高效的 HTAP 处理性能. 该类技术的主要优点是可以支持高效的事务处理, 主要缺点是需要大量内存空间存储索引.

HTAP 资源调度需要考虑系统整体性能与数据分析新鲜度这两个重要指标, 根据目前技术所考虑的系统指标, 本文将其分为两类技术: (1) 基于负载驱动的资源调度[20,21]; (2) 基于新鲜度驱动的资源调度[22]. 第(1)类技术通过监控当前混合负载的执行情况动态地调度系统资源, 如 CPU、共享缓存、内存带宽等. 该类技术的优点是分析性能高, 缺点是没有考虑到数据分析的新鲜度. 第(2)类技术以给定新鲜度阈值为基准, 动态地调整执行方式, 以更好地调度资源. 其优点是数据分析新鲜度有保证, 但会牺牲一部分系统性能.

2.1 HTAP数据库的数据组织技术

HTAP 数据库数据组织的关键技术主要分为两大类: (1) 基于主行存的内存列选择[9−11]; (2) 基于负载驱动

的行列混合存储[12−15]. 接下来分别介绍这两类技术.

2.1.1 基于主行存的内存列选择

内存列选择技术以全量行存数据为基础, 通过选择部分频繁被访问的列数据到内存中, 使得相应查询算子可以通过内存列扫描而被加速处理. 一种直接的选择方法是, 由数据库系统管理员根据具体的应用需求和数据模式手动地指定内存列集合[9]. 其根据选择的列集合和设定的内存大小判断是否满足条件: 如果满足, 则执行内存列的选择. 但是由人工指定的方法对查询负载的适应性不够强, 随着负载的变化, 由于大量查询都没有命中, 选定的内存列集合可能无法满足加速查询的需求. 针对这类问题, 需要根据查询负载自动地选择内存列. 目前, 主要的自动化方法是以热力图的方法与整型规划算法的方式来选择内存列.

基于热力图的方法[10]根据查询负载来记录内存列被访问的频度, 然后动态调整在内存中的列存数据, 访问频度高的列被保留或抽取在内存中, 访问频繁低的列需要被调出内存. 如图3所示, 基于热力图的内存选择技术将内存列存储分为 3 种状态: 热列数据、备选列数据以及冷列数据. 首先, 初始状态用户选择的所有列数据为备选列数据, 根据记录查询访问每个列的频度, 将频繁被访问的列标记为热列数据. 热列数据会自动地从全量行存储进行数据填充, 从而能够对将来同样类型的查询进行加速. 如果一部分列在指定时间窗口内没有被访问过, 则会被标记为冷列数据. 冷列数据中已被填充的数据需要被压缩然后调出内存, 并被持久化到磁盘中. 基于热力图的内存选择技术根据当前已出现的查询负载进行列选择, 对重复出现的查询有加速效果, 但当有新的查询出现后, 会出现查询无法命中的情况, 因而影响分析性能. 另外一个缺点是: 目前的自动选择方法暂时未考虑到数据更新的频繁程度, 可能会出现选择的列数据在一段时间后, 由于频繁的数据更新已经变旧.

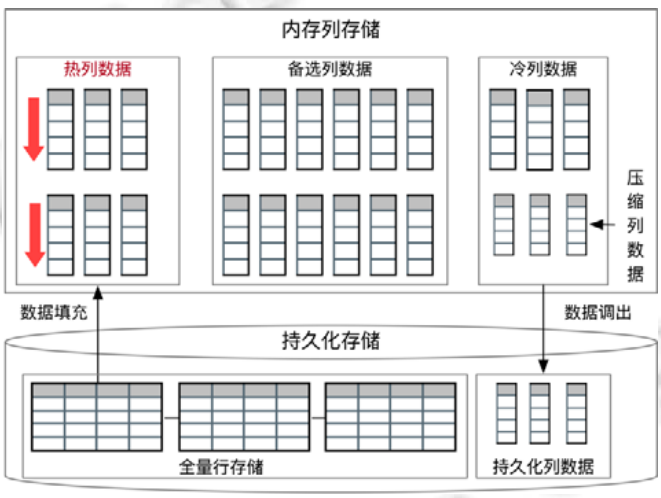


图 3 基于热力图的自动化内存列选择技术

基于整型规划的算法[11]旨在将效用高的属性列放入内存, 将效用低的属性列持久化到磁盘. 其将数据组织问题形式化为 0/1 背包问题(即每一属性列的大小对应为物品的重量, 列的效用对应为物品的价值, 然后选择价值最大的物品集合到有限容量的背包中). 给定一个查询负载 $W=\{q_1,\ldots,q_n\}$与可用内存大小 M, 其中每个查询 q_i 可涉及单列 c_i 或多列$\{c_1,\ldots,c_n\}$. 主行存储数据库中有 N 个列集合 C, 其中生成第 i 列的代价为 f_i. 假设 c_i 针对查询负载 W 能带来的查询收益为 g_i, 列 c_i 的大小为 m_i, 因此列 c_i 的效用为$(g_i-f_i)/m_i$. 该问题优化的目标为: 在选择列的总大小不超过 M 的条件下, 最大化选择列集合的总效用. 该问题是一个优化问题, 且是一个经典的 NP 难问题, 可以使用整型规划算法进行求解. 但已有方法仅基于独立性假设估计候选列集合的效用, 忽略了多列之间的关联关系, 因此可能选择到次优的方案. 其次, 当选择列的数量很大时, 整型规划算法往往不够高效. 最后, 已有方法并未考虑选择列的更新代价. 在 HTAP 场景下, 如果选择的列被频繁更新, 将会导致系统整体性能下降.

2.1.2 基于负载驱动的行列混合存储

该类技术[12−15]采取基于负载驱动的行列混合存储方案. 例如: H2O[12]支持 3 种存储形式, 即纯列存、纯行存、列组(关系表的不同纵向划分). 给定一批查询负载, 其通过评估不同存储方案的代价(包括处理代价与转换代价), 然后在线地选择一个最优的存储方案进行切换. Casper[13]基于负载驱动选择列数据的组织方式, 包括列的划分个数, 每一个划分的大小、范围、排序方式、更新方式以及缓存大小. 其通过代价函数评估每一个方案的代价, 然后针对给定负载, 求解最小化其代价的方案. Peloton[14]采用灵活模式在逻辑上将关系表划分为不同的列组, 然后在物理层面以砖块的形式将不同的列组物化在磁盘里. 其基于负载驱动的方法将经常访问的属性组合成列组, 以加速查询. 与之前仅考虑单机的情况不同, Proteus[15]基于分布式 HTAP 数据库选择最优的行列混合存储方案. 因此, 其考虑的方案空间更大, 包括数据的划分方式(纵向或横向)、同一份数据的副本格式(行存或列存)、数据存储位置(内存或磁盘). 其通过将备选方案特征化, 然后通过学习型代价函数选择最优方案.

目前, 基于混合组织的存储方式主要有两大缺点: 首先, 混合组织数据的方式增加了系统在查询处理与优化层面的复杂度, 需要实现许多新的执行规则, 并且以往的传统代价模型也由于混合存储而失效; 其次, 混合存储的事务处理会访问多个不同的列组, 造成多次磁盘 I/O.

2.2 HTAP数据库的数据同步技术

由于行列共存型 HTAP 数据库采用行存与列存双存储的方式来并发处理混合负载, 那么如何同步行存与列存之间的事务数据以保持数据分析的新鲜度同时又能保证系统运行的性能是一个关键的挑战. 在 HTAP 数据库中, 查询分析需要同时扫描列存储与增量存储, 但如果增量存储的数据不断增大, 基于增量存储与列存储扫描的方式其代价较大. 因此, 需要定期地将增量存储数据合并到列存中. 根据增量存储的类型(增量内存表或增量日志)以及列存储的位置(内存或磁盘), 本小节将介绍两类数据同步技术, 包括基于内存增量表与内存型列存的数据同步与基于增量日志与持久化列存的数据同步.

2.2.1 基于内存增量表与内存型列存的数据同步

本节介绍 3 种关键技术: 两阶段迁移的数据同步技术、基于字典排序合并的数据同步技术以及基于阈值驱动的数据同步技术.

1) 两阶段迁移的数据同步技术

两阶段迁移技术[4]基于增量存储和删除表来实现, 其中,

- 增量存储以附加的形式将更新数据插入到内存表的尾部, 其对于每一条记录分配一个到列主存的 RID 映射, 该映射能够直接定位其在列存储中的分组与偏移位置, 但还未被合并到列存的尾部数据的 RID 都是暂时无效的, 它们会被建立 B+树索引, 以便在合并时被快速访问;
- 删除表同样基于内存表, 其记录了在列存中被(逻辑上)删除数据的 RID. 当列存做扫描时, 也会扫描删除表, 同时, 在列存与删除表中的数据会被跳过.

为了减少扫描增量存储与删除表的代价, 两阶段迁移技术作为后台进程定期地将增量存储的数据迁移到主行存. 如图 4 所示, 两阶段迁移技术分为以下两个阶段.

(1) 第 1 迁移阶段通过扫描增量存储的尾部来获取最新的事务数据, 对于近期(通常 1 小时内)频繁被更新的数据, 将暂时不会被加入, 因为它们很有可能再次被更新, 加入它们会导致迁移性能的下降. 当被选定元组确定后,它们被压缩成列组放到内部表中, 所有的新增 RID 映射被加入到删除表中, 整个第 1 阶段作为一个事务提交;

(2) 在第 2 迁移阶段的开始, 行存表的映射还在删除表中, 删除表的数据也还未被迁移, 所以它们对新查询都还是不可见的. 因此, 第 2 阶段将针对每一行在删除表中的数据进行删除, 并更新行存表的映射, 这样的操作被当作一次短事务. 当这些事务处理完之后, 增量存储的尾部数据被清除, 新数据即对后续查询可见.

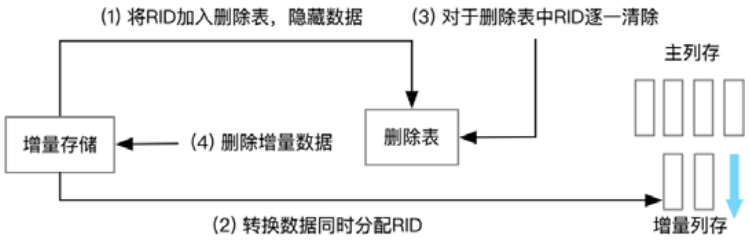


图 4　基于两阶段迁移的数据同步技术

2) 基于阈值驱动的数据同步技术

该类技术[3]使用事务表(transaction journal)统一记录变更数据的 RowID，当变更记录数目达到阈值后，系统会根据事务表的 RowID 到行存中取得最新数据，然后合并到列存储中. 如图 5 所示：当数据更新时，对应的元组 RowID 将被插入到增量存储中. 其基于阈值向列存合并增量数据，并有两种传播形式：第 1 种形式是通过设定内存更新的阈值，当数据量超过这个阈值后，内存中的主列存与增量存储中最新的事务数据合并，但是如果阈值设置得过大，可能导致列存数据的新鲜度较低. 因此，第 2 种形式是定期、定量地将增量存储中的数据合并到主列存中.

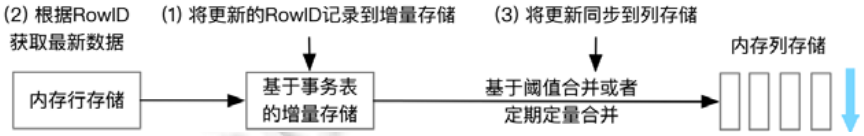


图 5　基于阈值驱动的数据同步技术

3) 字典排序合并的数据同步技术

该项技术[6]利用字典对插入的增量数据进行编码，然后利用排序技术合并，其主要流程分为两个阶段.

(1) 行存更新与增量列存储合并. 在第 1 阶段，根据新增的行数据逐列转换到增量列存储中. 根据新增的列数据，首先查询在增量列存储中的字典：如果数据在字典中存在，则直接将数据添加到数据列；如果不存在，则先更新字典，分配字典索引，然后添加新数据. 如图 9 所示：在第 1 阶段，新增的姓氏数据中，“李”“张”“赵”都在字典中. 因此，直接将它们对应的索引加入到数据列尾部；

(2) 增量列存储与全量列存储合并. 在第 2 阶段，首先将增量列存储中的字典与全量列存储中的字典合并. 通过插入排序，获得合并后的全量字典，然后将增量列存储中的数据对应地添加到全量数据中，全量数据中的数据索引也一起更新. 如图 6 所示：在第 2 阶段中，将增的姓氏字典插入合并到全量姓氏字典后，它们的索引也发生了相应的变化，因此需要在姓氏数据列也重新更改映射，这样，新增数据中的姓氏列即完成了数据同步.

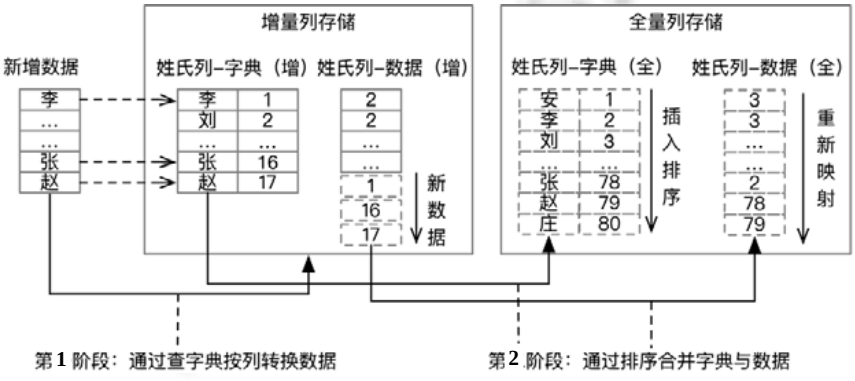


图 6　基于字典排序合并的数据同步技术

总体来说, 基于内存式增量合并的数据同步技术能够高效、快速地将增量数据合并到主列存储中, 这主要是因为所有同步操作都在内存中进行. 该类技术的缺点是由于内存容量有限, 扩展性不够. 例如, SAP HANA 的新增数据只能容纳 10 万级别的数据.

2.2.2 基于增量日志合并的数据同步技术

本小节介绍基于增量日志合并的数据同步技术[7,8], 该类技术基于磁盘型增量日志合并进行数据同步. 当增量存储被存放在多个增量日志文件中时, HTAP 数据库会在做查询处理的同时访问这些增量日志文件. 然而, 当需要扫描的增量日志太多时, 会造成读放大, 即针对单个查询读取了很多个页面, 从而极大地影响到查询处理的效率. 因此, 需要将增量日志定期合并到持久化列存储中.

如图 7 所示: 该类技术将数据的增加、删除、更新操作以附加的方式写到内存的同时, 也会将更新操作写入 B+树, 为数据建立有序索引. 内存中的增量数据会被合并成更大的数据块, 然后被转移到磁盘的增量缓冲区里. 这些数据都是按照写入的顺序被合并到一起的, 所以它们本身是无序的, 但在合并的过程中会将事务中重复的元组去掉, 同一行的多版本数据需要合并, 对于提交未成功且被回滚的日志操作, 需要在日志中删除. 最后, 这些增量数据会被继续合并到持久化存储中. 持久化存储将数据块转换成列存格式后有序地组织数据, 且每块数据涵盖数据的一部分范围. 由于增量空间的数据键值是无序的, 合并的操作会产生较大开销, 因此在合并的过程中会访问最开始建立的 B+树, 从而快速找到数据的准确位置, 合并完成之后, 旧的数据块会被移除, 从而保证了持久化存储中的数据也是最新的.

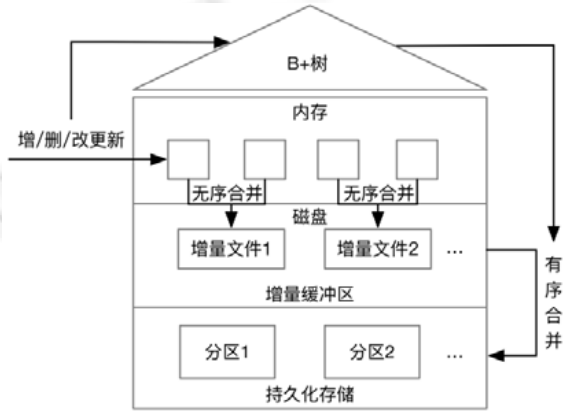


图 7 基于字典排序合并的数据同步技术

2.3 HTAP数据库的查询优化技术

HTAP 数据库查询优化的关键技术主要有 3 类: 混合行/列存储扫描[3,4,7]、CPU/GPU 加速[16,17]以及面向 HTAP 负载的索引技术[18,19]. 本节接下来将分别介绍各类查询优化技术.

2.3.1 混合行/列扫描技术

HTAP 数据库查询优化的另一种新技术是混合行/列扫描技术, 即对于一个查询的算子集合, 一部分在行存引擎里执行, 另一部分在列存引擎里执行, 从而达到比纯行存执行或纯列存执行更好的性能效果. 例如: 对于短范围或点查询, 可以利用基于 B+树索引的行存引擎执行; 对于多列扫描、复杂聚集, 可以利用列存储通过压缩和 SIMD 向量执行等技术进行执行, 最后, 查询引擎结合行存与列存的执行结果并返回给前端. 混合行/列扫描技术通常是流水线执行, 比如先行存执行索引查找, 结果输入到列存后进行顺序扫描, 最后输出最终结果. 目前, 许多 HTAP 数据库已经支持跨行列引擎混合执行查询的操作, 如 TiDB[7]、Oracle[3]、SQL Server[4]等. 如图 8 所示: SQL 查询查找在北京注册的汽车的证书和颜色, 其首先针对注册表 Register 的地点属性作选择谓词, 然后针对满足条件的注册表 Register 与汽车表 Vehicle 做连接操作, 最后针对 Vehicle 表进行投影操作. 由于 Register 有 B+树索引, 可以将在 Register 表的选择算子在行存中执行, 然后将连接算子和投影算

子在列存中的 Vehicle 表执行, 其中, VID 集合是上一步选择算子的输出数据, 然后查询引擎根据对应的 ID 在列存上进行快速扫描, 最后直接投影对应的 license 和 color 列.

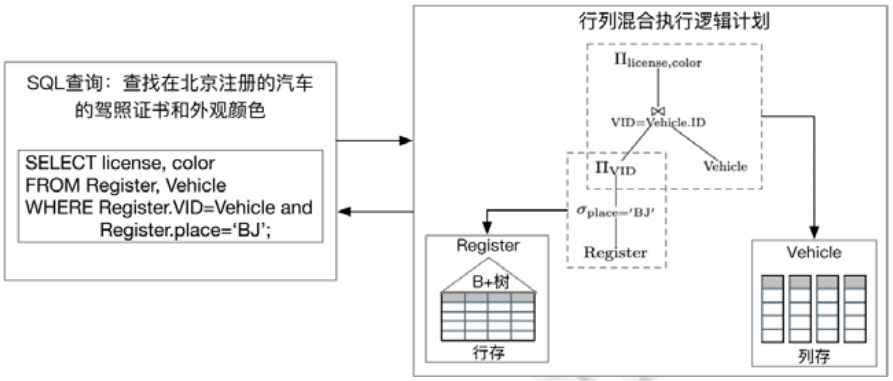


图 8 基于混合行/列扫描的查询优化技术

混合行/列扫描的关键是, 判断一个查询或算子应该针对行存或是针对列存执行. 一种启发式的方法是优先选择内存列执行, 如果有缺省的列在行存中再去行存中继续执行. 这类方法判断流程简单, 且对很多情况都是有效的. 目前, Oracle、SQL Server 都支持这种方法. 然而, 该类方法导致先在行存里执行的计划被忽略, 因此获得的执行计划可能不是最优的. 还有一种主要方法是基于代价函数的判断执行, 即分别比较算子在行存执行和列存执行中的代价, 然后决定执行顺序. 代表性系统是 TiDB, 其分别基于行存扫描、索引扫描、列存扫描建立代价函数, 然后选择代价最小的执行方式.

基于代价函数的方法[7]通过估计访问路径的代价来选择行列混合执行方式. 公式(1)是最优化代价函 $C_{optimal}$, 其在列存储扫描代价 C_{column}、行存储扫描代价 C_{row}、索引扫描代价 C_{index} 这 3 项中选择最小代价的方式. 公式(2)是扫描列存储的代价函数, 其中, $\tilde{S}_i$ 是第 i 列的扫描估计大小, $C_{scan}(i)$是扫描第 i 列数据的单位代价, $C_{seek}(i)$是寻找第 i 列的代价. 公式(3)是扫描行存储的代价公式, 其中, $\tilde{S}_j$ 是扫描第 j 块数据的估计行数, $C_{scan}(j)$是扫描单位行数据的代价, $C_{seek}(j)$是寻找第 j 块数据块的代价. 公式(4)是索引扫描的代价函数, 其中, $\tilde{S}$ 是扫描索引的估计大小, C_{scan} 是扫描索引的单位代价, C_{seek} 是寻找索引文件的代价:

$$C_{optimal}=\min(C_{column},C_{row},C_{row}) \tag{1}$$

$$C_{column}=\sum_{i=1}^{n}(\tilde{S}_i\cdot C_{scan}(i)+C_{seek}(i)) \tag{2}$$

$$C_{row}=\sum_{j=1}^{n}(\tilde{S}_j\cdot C_{scan}(j)+C_{seek}(j)) \tag{3}$$

$$C_{index}=\tilde{S}_i\cdot C_{scan}+C_{seek} \tag{4}$$

目前, 数据库里常用的估计方法主要有基于直方图与基于简单抽样的方法[30].

综上所述, 针对行列混合执行计划优化, 目前的方法主要基于启发式规则, 如优先考虑列扫描, 然后考虑行扫描[10]. 但这样的计划可能不是最优, 例如, 有索引的行扫描有可能比列扫描效率更高. 基于代价函数针对行列执行计划选择[4,7], 但这些代价函数都基于独立性和均匀分布假设, 当假设不成立时, 往往结果不准确.

2.3.2 异构 CPU/GPU 的查询加速技术

CPU/GPU 的异构处理器加速技术也是 HTAP 数据库查询优化的一项重要技术[16,17], 其利用 CPU 任务并行的特点处理 OLTP 事务, 利用 GPU 数据并行的特点处理 OLAP 分析. 目前, 主流的 CPU/GPU 的加速技术都是基于双存储来隔离 OLTP 与 OLAP 的负载执行, 即在执行过程中不共享内存与缓存, 以减少混合负载执行过程中的资源争抢. 如图 9 所示: HTAP 负载被分类为 OLAP 与 OLTP 负载, OLAP 通过 GPU 在分析存储上执行, OLTP 通过 CPU 在事务存储上执行, 然后事务存储更新的数据可定期批量同步到分析存储中. 通常来说, 分析

存储会采用列式存储, 事务存储会采用行式存储, 但出于工程复杂度的考虑, 也有都选择行存或都选择列存的系统. 值得一提的是: 针对 OLAP 部分, 还可以部署一些 CPU 核用于加速一些任务并行的短查询.

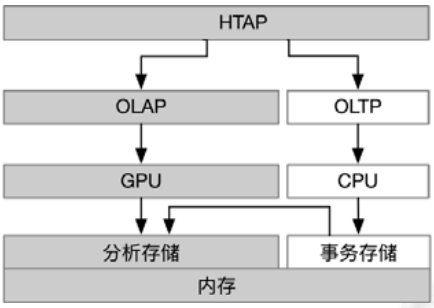


图 9 基于 CPU/GPU 查询优化技术的架构图

CPU 针对事务存储的更新基于多版本并发控制, 对于插入操作, 直接附在存储的尾部, 并初始化事务创建元组的时间戳与删除元组的时间戳. 对于删除操作, 则设置被删除的元组的时间戳, 然后将 ID 加入到删除表中, 用于后续分析存储同步数据. 对于更新操作, 则变换为一次插入和一次删除旧元组的操作.

随着高性能通用图形计算处理器 GPGPU 与 CUDA、OpenCL 等编程框架的出现, 以 GPU 为核心的查询处理技术迎来了蓬勃发展[42,43]. 针对选择、投影、扫描等基本关系算子, GPU 查询引擎通过将它们转换为多个并行的原语(primitives)进行执行. 例如: 选择算子可以用过滤(filter)原语实现, 而过滤原语又由底层的映射(map)原语、前缀求和(prefix sum)原语以及分散(scatter)原语组成. 针对复杂关系算子如连接和聚集算子, 它们属于计算密集型操作, 更加适合利用 GPU 进行并行加速, 主要方法也是利用 GPU 底层并发原语如扫描(scan)、分散(scatter)、划分(split)来实现排序归并连接、哈希连接等常用连接算法.

2.3.3 面向 HTAP 负载的索引技术

该类技术旨在通过索引技术同时加速事务处理与查询分析, 目前的相关工作不是很多, 主要包括基于路径复制[18]的索引技术与基于多版本[19]的索引技术. 前者以并发二叉树(parallel binary tree, P-Tree)[18]为代表, 其主要思想是: 通过在平衡二叉树上对最新事务涉及的数据路径进行复制, 从而使得数据能够被多核处理器并发地更新. 此外, P-Tree 实现了快照隔离级别, 所以查询也能同时访问索引上可见的快照数据. 事务读取操作获取当前可见版本的根节点指针, 然后沿着索引查找路径上相应的数据; 更新操作将创建新版本的根节点, 然后复制所有相关路径并更新目标节点; 事务读取与更新操作都可以在 O(log$n$)时间内完成. P-Tree 还支持跨多表的嵌套模式, 使得查询可以不用做连接即可通过索引访问多表数据. 多版本索引技术以多版本分区 B-Tree (multi-version partitioned B-tree, MV-PBT)[19]为代表, 其主要解决的问题是如何高效地对同一数据的多版本数据进行建立索引并支持快速地查询最新的可见版本数据. MV-PBT 基于分区 B-Tree 实现, 分区根据指定的键进行划分, 每一个分区有相同的搜索键. 事务的所有更新将写入内存缓存区, 当缓冲区被填满后, 会将数据持久化到对应的分区中. 由于同一数据的不同版本信息已被索引到分区 B-Tree 中, 因此其支持快速地搜索数据的可见版本. 已有索引技术的主要优点是提升了事务读写的性能, 但这些技术的主要缺点是在事务处理时需要消耗大量的内存空间用于保存数据的中间结果以及版本信息. 此外, 目前的索引技术主要面向事务优化, 虽然能够支持查询访问, 但并没有做特定的优化, 未来可结合面向列存的索引[3], 如区域图(zone map)与布隆过滤器(Bloom filter)增加对查询分析的优化.

2.4 HTAP数据库的资源调度技术

HTAP 数据库需要同时支持执行 OLTP 事务与 OLAP 查询, 但这两种负载之间会存在数据的同步与 CPU 资源的共享. 已有研究结果显示: 当 OLTP 事务与 OLAP 查询同时执行时, 由于数据的同步与它们之间共享资源造成的互相干扰, HTAP 数据库的整体性能会下降 5 倍[44]. 因此, 如何高效地调度 OLTP 与 OLAP 负载之间资源从而最大化 OLAP 的数据分析新鲜度和系统整体运行性能, 是一个关键的问题. 本节介绍两种资源调度

技术, 即基于负载驱动的调度和基于新鲜度驱动的调度. 总体来说, 前者以提高系统性能为目标, 但是不考虑数据分析的新鲜度; 后者以数据新鲜度为目标, 但可能以牺牲系统性能为代价. 下面具体介绍这两种技术.

2.4.1 基于负载驱动的资源调度技术

该项技术通过监控当前混合负载的执行情况, 动态地调度系统资源如 CPU、共享缓存、内存带宽等[20,21]. 由于 OLTP 的访问方式为短时延、高并发事务型处理, OLAP 的访问方式为长时延、数据密集型分析处理, 因此, 根据它们各自的特点, 需要分别进行资源调度.

首先, 对于 CPU 资源, 可分别针对事务并发度和分析查询并发度来进行动态调度[20]. 例如: 对于事务的并发度, 初始化负载执行的并发线程为系统的 CPU 核数, 随着负载的执行, 当监控进程(watchdog)发现有被阻塞的线程就加入队列, 调度器另外启动线程为阻塞队列执行任务, 达到充分利用 CPU 资源的效果. 对于查询的并发度, 主要是从数据的并发粒度来进行考虑. 例如: 对于一个列的聚集操作, 可将列数据划分为多份进行并发处理, 每一个线程处理一份数据的聚集操作. 然而, 数据划分的粒度也并非越高越好, 如果过度地划分线程, 会使得线程之间通信、数据同步、调度等流程造成过多额外开销. 因此, 可统计过去一段时间内该列的执行情况, 然后取并发线程数量的平均值为查询的并发度.

其次, 对于共享缓存和内存带宽资源, 也可通过观测负载执行情况动态地进行调整. 例如: 在执行混合负载时, 如果发现 OLAP 的吞吐量比单独执行时下降过多, 说明 OLTP 的执行影响到了 OLAP 的执行, 因此可以通过在 *L*3 共享缓存划分更多的资源给 OLAP 隔离执行(如通过英特尔芯片的缓存分配技术)[21]. 此外, 当发现 OLAP 的吞吐量下降过多时, 往往是由于增量存储的生成速度比合并到主分析存储的速度要快, 导致 OLAP 读取增量存储的开销很大, 因而需要多分配资源给增量存储的合并任务.

2.4.2 基于新鲜度驱动的资源调度技术

该类技术以给定的新鲜度为阈值, 在不同执行方式中切换[22]. 其首先采取隔离执行, 当数据新鲜度小于给定阈值后, 需要同步数据后再进行分析, 或者切换共享数据的执行方式. 总体来说, 执行 HTAP 混合负载有两种方法: (1) 隔离执行 OLTP 和 OLAP 负载, 然后定期地把 OLTP 的最新数据增量式地同步到 OLAP 的实例; (2) OLTP 和 OLAP 同时执行, 并且都共享同一实例. 对于第 1 种方式, 更有利于系统高性能运行, 缺点是 OLAP 分析的数据新鲜度低, 因为 OLTP 也同时产生了新数据; 对于第 2 种方式, 能够保证 OLAP 分析最新 OLTP 数据, 但缺点是共同执行 OLTP 与 OLAP 会互相干扰系统执行负载. 因此, 如何在两者之间找到平衡点是一个关键问题.

如图 10 所示, 该架构采取混合执行资源调度. 左侧的 3 个 CPU 用于对 OLTP 实例 1 执行负载, 中间的 3 个 CPU 用于对 OLTP 实例 2 执行 OLAP 负载, 右侧的 5 个 CPU 用于对 OLAP 实例执行查询. OLTP 的数据可通过数据交换器传输到 OLAP 实例, OLAP 实例还可以通过互连器访问 OLTP 实例 2 的数据, 并共享中间的 3 个 CPU.

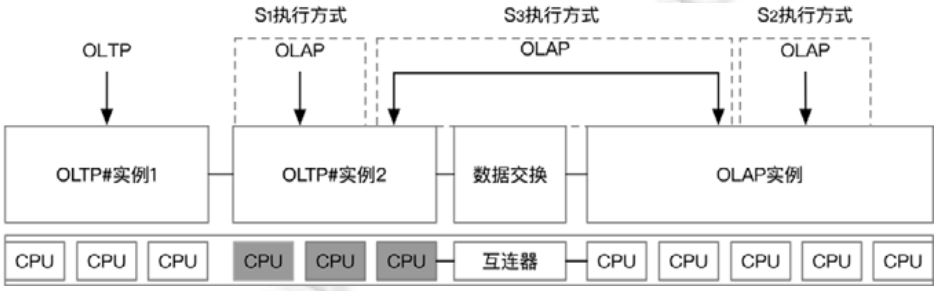


图 10 基于新鲜度驱动的资源调度框架

系统可支持 3 种执行方式, 如图 10 所示. 其中, 执行方式 S1 为同地式(co-location)执行, 即通过写复制 OLAP 访问最新的 OLTP 实例, 如 SAP HANA、Hyper 属于这一类; 执行方式 S2 为分离式(isolated) OLTP 与 OLAP 执行, 即 OLTP 实例和 OLAP 实例各自利用自己的 CPU 执行, 然后定期地合并数据到 OLAP 实例, 如

TiDB、BatchDB 属于这一类; 执行方式 S3 为混合式(hybrid)地执行, 即当有查询到来时, OLTP 实例切换到另一个 OLTP 实例, OLAP 查询可同时访问 OLAP 实例与 OLTP 的最新增量数据, 如 Oracle 与 SQL Server 都属于这一类.

对于具体资源调度来说, 系统默认选择 S2 分离式执行. 当数据新鲜度小于给定阈值后, 根据具体的服务级别协议(service-level agreement, SLA)[35], 可以跳转到 S3 的混合状态执行或是 S1 的同地式执行. S1 可以马上分析最新数据, 但分析的数据容量有限. S3 有一定时间延迟, 但支持的数据容量和 CPU 资源更多. 在 S3 中, 也可以先传输数据再分析, 或是直接混合分析 OLAP 实例和 OLTP 实例的数据, 这也取决于具体的服务级别协议. 如果数据新鲜度又达到阈值以上时, 可重新跳转到 S2 分离式执行.

目前, 针对 HTAP 的资源调度主要以固定规则为主, 基本方法都是通过观测系统执行混合负载后的性能指标或基于数据分析的新鲜度驱动, 根据预先给定的规则进行资源调度. 例如: 如果系统执行混合负载时, 针对 OLTP 或者 OLAP 的吞吐量下降幅度很大, 那么调度算法将分配更多的 CPU 线程和缓存资源用于执行相应的负载. 然而, 这些调度方法没有对已知负载的特性和系统行为进行建模, 因此无法及时地进行调整, 稳定性很差.

3 面向 HTAP 数据库的应用程序开发

HTAP 数据库面向上层应用提供统一的服务接口, 所以与传统数据库的接口类似, 只需提供数据库的服务地址给目标应用程序, 即可对应用的数据进行管理, 包括数据字典的定义、数据的增/删/改/查、混合事务处理与查询处理. 与传统数据库不同, HTAP 数据库需要指定主行存数据库中哪些数据需要被抽取并转换为列存储, 以加速分析型查询. 当初始化完成后, 系统便能够结合第 2 节的关键技术进行自动化的数据组织、查询优化、数据同步以及资源调度. 目前主要有 3 种列存构建方式: (1) 基于内存列的构建[3,9,10]; (2) 基于列存索引的构建[4,30]; (3) 基于分布式列存副本的构建[7,8].

3.1 基于内存列的构建

该类方法默认在内存中的数据为列式存储, 因此用 INMEMORY 关键字指定目标数据为列存, 其可支持不同粒度的列存储, 包括表级别与列级别. 此外, 其可根据应用场景需求指定数据的优先级, 例如: 可在(非常重要/高/中/低/无)这 5 个级别中选择一个优先级, 然后被设定为“非常重要”级别的数据会被马上填充, 而“无”级别的数据只有在被扫描的时候才被填充. 它还支持对数据压缩模式的指定, 例如: 如果数据可能被频繁更新, 可选择无压缩模式; 数据的查询性能如果很重要, 可以选择偏向查询的压缩方式(即兼顾存储空间与压缩率); 数据的存储空间如果有限制, 还可以选择偏向存储空间的压缩(即偏向低压缩率的算法). 具体例子如下(oracle 21c[10]):

```
ALTER TABLE Vehicle INMEMORY MEMCOMPRESS FOR CAPACITY HIGH PRIORITY NONE;
```

该语句选定 Vehicle 表为列存(关键字 INMEMORY), 选择偏向存储空间的压缩(关键字 MEMCOMPRESS FOR CAPACITY HIGH)以及选择无重要级别的优先级(PRIORITY NONE).

3.2 基于列存索引的构建

该类方法通过建立名为列存索引(columnstore index)的方式指定列存的构建. 实际上, 列存索引也包含了所有的列数据, 与内存列的构建方式不同, 它还支持持久化的存储方式, 具体例子如下(SQL server 2016[4]):

```
CREATE TABLE Vehicle (INDEX Vehicle_col COLUMNSTORE) WITH (MEMORY_OPTIMIZED=OFF);
```

该语句为行存表 Vehicle 建立了名为 Vehicle_col 的列存索引(关键字 COLUMNSTORE), 并且选择了存储模式为非内存(关键字 MEMORY_OPTIMIZED 设定为 OFF).

3.3 基于分布式列存副本的构建

该类方法为选定的行存表建立分布式列存副本, 以提升系统的分析性能与可用性, 具体例子为(TiDB[7]):

```
ALTER TABLE Vehicle SET TIFLASH REPLICA 3;
```

该语句为行存表 Vehicle 建立了同名的 3 份列副本(关键字 TIFLASH REPLICA 3), 每个副本可被放置在不同的机器或存储区域. 此外, 它还支持在查询时利用手动 HINT 指定访问的数据格式, 具体例子如下:

SELECT/*+read_from_storage (TIFLASH [Vehicle])*/from Vehicle;

该查询语句在 SELECT 后插入手工 HINT (关键字 read_from_storage (TIFLASH [Vehicle])), 因此, 该查询将使用列存副本进行扫描, 还可以将 TIFLASH 替换为 TIKV 后使用行存副本进行扫描.

4 HTAP 数据库评测基准

数据库评测基准(database benchmark)是数据库管理系统必不可少的重要组成部分, 其不仅是测试与衡量数据库系统性能的关键工具, 也是推动数据库新技术不断发展的助推器. 数据库发展近 60 年来, 工业界和学术界提出并开发出了一大批优秀的数据库评测基准工具. 评测基准对 HTAP 数据库技术的发展也至关重要, 但目前可用的标准和工具并不多, 现有评测基准主要以结合 TPC 以往的面向事务处理的基准测试和面向决策支持的基准测试为主, 下面我们具体介绍 3 个具有代表性的评测基准.

4.1 评测基准CH-benCHmark

CH-benCHmark[45]是一个目前被广泛采用的针对 HTAP 数据库的性能评测基准, 由于其简单易用, 并结合了两个经典的数据库评测基准, 即面向事务处理的基准 TPC-C (http://tpc.org/tpc_documents_current_versions/pdf/tpc-c_v5.11.0.pdf)和面向决策分析的基准 TPC-H (http://tpc.org/tpc_documents_current_versions/pdf/tpc-h_v3.0.0.pdf). 已有许多 HTAP 系统都使用 CH-benCHmark 进行了性能评测. CH-benCHmark 的主要贡献在于, 融合了 TPC-C 和 TPC-H 的数据模型(如图 11 所示)、数据生成以及工作负载. 而且, 其制定了一套测试 OLTP 和 OLAP 的评测基准执行参数, 用于控制 HTAP 的 4 个方面的执行规则, 包括数据库大小、负载混合类型、隔离级别以及混合执行时用于分析的数据新鲜度. 最后, 提出了一套基于参照的评价指标, 可用于细粒度分析系统性能.

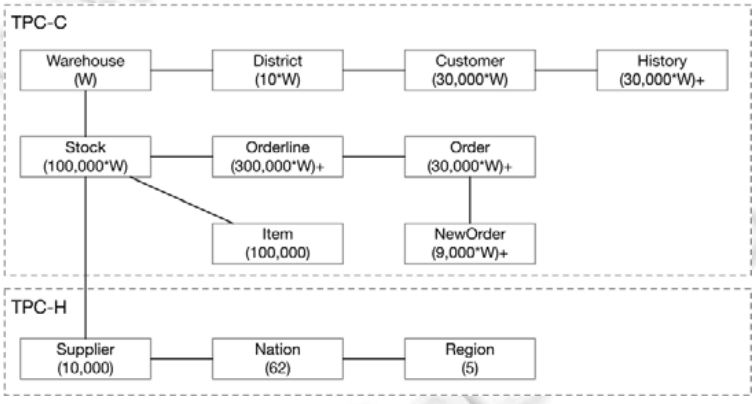


图 11 CH-benCHmark 数据模型

下面, 本小节从数据模型与数据生成、查询负载与执行规则、评测指标这 3 个方面来介绍评测基准 CH-benCHmark.

(1) 数据模型与数据生成

因为 TPC-C 和 TPC-H 的数据模型都基于商品零售场景, 且它们的数据表有重合部分, CH-benCHmark 的数据模型保留了 TPC-C 的原有的包括地区表、仓库表、库存表、新订单表等所有 9 个表, 然后将 TPC-H 数据模型中的 8 个表融入进来. 其合并了它们的用户表、订单表、订单线表与产品线表、商品表与零件表, 去掉了 TPC-H 的产品供应商表(partsupp), 最终, CH-benCHmark 的数据模型包含了融合后的 12 个数据表, 可同时支持 OLTP 和 OLAP 的操作. 对于数据生成, TPC-C 与 TPC-H 使用不同的数据生成扩大模型(scaling model),

TPC-C 基于连续性扩大模型, 即随着系统的运行, 数据不断增加. TPC-H 基于固定式扩大模型, 即给定扩大因子(scale factor), 生成的数据库大小是固定的. 为统一数据生成, CH-benCHmark 将 TPC-H 也修改为与 TPC-C 一致的连续性扩大模型.

(2) 查询负载与执行规则

CH-benCHmark 的查询负载也基本沿用了原有的负载操作. 因为 TPC-C 的部分完全被保留, 所以 CH-benCHmark 支持原有的 5 个事务操作, 即针对写的新订单事务、支付事务操作以及针对读的订单状态事务、订单交货事务、库存状态事务. 它们执行时所占的比例分别为 44%、44%、4%、4%、4%. 针对分析查询, CH-benCHmark 根据数据模型的变动相应地修改了 TPC-H 的原有 22 个查询, 除简化了一些算术操作和适应性的主外键连接调整, 基本保留了原有查询的商业语义性和语法结构. 在执行 CH-benCHmark 时, 有 4 个参数可以配置. 首先, 因为数据的生成是以数据仓库的个数来进行扩展生成, 可设定数据仓库的个数决定测试数据库的大小. 其次, 可决定执行时工作负载的类型. 可以设定只有 OLTP 的事务负载, 只有 OLAP 的查询负载, 或者混合两种负载执行. 对于混合执行的方式, 可以多个流同时执行 OLTP 和 OLAP 负载, 每一个流可包含执行所有事务操作, 或者包含所有 22 个查询. CH-benCHmark 还可设置隔离的级别, 比如可设置低隔离级别为读提交, 从而支持快速数据处理. 还可设置更高隔离级别如可重复读. 最后, CH-benCHmark 还可指定混合执行时数据的新鲜度. 通过设定时间阈值或者事务的数目, 系统可决定何时将最近数据用于查询处理中.

(3) 评测指标

TPC-C 的评测指标主要以事务完成的吞吐量来衡量, 用 *tpmC* 表示, 即每分钟完成的平均事务数量. TPC-H 的主要评测指标是分析查询的延时衡量, 用 *QphH* 表示, 即每小时所完成的查询数量. 因为 TPC-C 和 TPC-H 评测的侧重点有所不同, 所以一个简单的方法是直接给它们的指标结果标准化然后加权结合, 然而这样操作无法更直观地比较系统针对混合型 OLTP 和 OLAP 工作负载的处理性能. 因此, CH-benCHmark 提出了一种参照率指标的方法, 比如符号*@tpmC* 代表参照 OLTP 吞吐量. 参照 OLTP 和 OLAP 的评测指标公式如下:

$$M(OLTP)=\frac{tpmC}{QphH}@tpmC \tag{5}$$

公式(5)中, 评价指标 *M* 被表示为 OLTP 与 OLAP 的指标比率, 并且参照了 OLTP 的吞吐量. 如果主要参照对象是 OLAP, 则可用下面公式(6)来表示:

$$M(OLTP)=\frac{tpmC}{QphH}@QphH \tag{6}$$

此外, 为了对比在隔离处理 HTAP 和混合处理 HTAP 时系统的性能, 可分别执行负载, 然后再来对比系统在处理负载间抢占资源时的性能. 举例来说: 当 OLTP 为主要参照对象时, 顺序执行混合负载的指标为 5.7@5084 *tpmC*, 然后并行执行混合负载的指标为 6.5@5188 *tpmC*, 那么可以认为系统在并行执行混合负载后性能没有下降, 且有略微的提升.

4.2 评测基准HTAPBench

HTAPBench[46]是一个在 2017 年被提出来的专门针对 HTAP 数据库的评测基准, 它同样是基于评测基准 TPC-C 和 TPC-H 而设计的. 其主要贡献在于: 提出了一种统一指标, 用于衡量系统在处理 OLTP 和 OLAP 负载的性能. 另外, 还开发了一个协调执行 OLTP 和 OLAP 负载的客户端均衡器, 用于控制吞吐量和分析延时的关系. 其测试的主要目标是考察 HTAP 系统在保持给定的 OLTP 吞吐量的同时, 是否拥有能够处理一定 OLAP 分析的能力.

(1) 数据模型与数据生成.

HTAPBench 采用与 CH-benCHmark 一样的数据模型, 即保留所有 TPC-C 的表和实体关系, 然后融合 TPC-H 的 8 张表到 TPC-C 的数据模型中. HTAPBench 的测试数据生成也直接使用原有的数据生成器, 但加入了时间戳用于控制混合负载的执行.

(2) 查询负载与执行规则.

HTAPBench 的查询负载分为 TPC-C 的事务处理负载和 TPC-H 的报告分析负载两部分. 首先, 给定一个目标吞吐量 *T*, 由于 TPC-C 规定每个数据仓库不超过 12.86 *tpmC*, 可通过公式(7)和公式(8)来计算客户端 *C* 和数据仓库 *W* 的数目:

$$C=\frac{T}{1.286} \tag{7}$$

$$W=\frac{C}{10} \tag{8}$$

然后, 即可开始 OLTP 的事务数据的导入与执行. 对于 OLAP 查询, 首先测试需要保证每次的查询动态访问数据的区域, 而不是每次都固定地访问整个数据区域, 这样会造成每次查询之间的性能差异不大. 因此, 其在生成分析查询时, 通过数据密度分析当前数据增加最多的数据区间, 然后通过滑动窗口直接定位到这一区间为查询的目标. 当查询都准备好后, HTAPBench 通过客户端均衡器确定需要启动多少工作数目来执行 OLAP 查询. 客户端均衡器的主要作用是监控当前的 OLTP 执行的吞吐量与目标吞吐量, 然后决定是否继续增加工作数目*#OLAPworker* 来执行 OLAP.

(3) 评测指标.

HTAPBench 提出了一种统一指标来衡量系统在执行 HTAP 混合负载的性能, 这主要是为了避免分别衡量 OLTP 与 OLAP 的处理性能时忽略了 OLTP 对 OLAP 的影响. 通过固定 OLTP 的吞吐量, 提出了指标 *QpHpW*, 即每个工作每小时所完成响应的查询数目, 如公式(9)所示:

$$QpHpW=\frac{QphH}{\#OLAPworker}@tpmC \tag{9}$$

其中, *QphH* 是每小时所完成的查询数量, *#OLAPworker* 是当前工作数目, *QpHpW* 由它们的比率与 OLTP 的吞吐量 *tpmC* 组成, 其中, *@tpmC* 代表参照系为 OLTP 吞吐量. *QpHpW* 越大, 表明系统在满足一定 OLTP 吞吐量的情况下, 每个工作每个小时响应的查询越多.

4.3 评测基准CBTR

CBTR[47,48]是最早提出测试系统在处理混合 OLTP 和 OLAP 负载时分析性能的基准测试. 与前两个合成模拟数据的基准测试系统不同, CBTR 基于一个订单兑现的真实数据集构建, 该数据集包含 18 个数据表, 数据呈倾斜分布, 其中最宽的表有 326 列, 但它们仅满足第一范式. CBTR 的工作负载包含 9 个事务处理和 4 个分析查询, 其中, 事务处理分为读写事务(如创建新订单、新快递)和只读事务(如查询未完成或已完成订单). 分析查询包含销售订单的分析(如比较不同种类商品的销售情况、查询订单平均完成时间等). CBTR 的查询负载执行主要分为两种类型: 纯 OLTP 事务处理、混合 OLTP/OLAP 执行. 通过交互式地增加 OLTP 或 OLAP 客户端, 可在执行期间内观测系统的性能动态变化. 然而, 由于 CBTR 的系统和数据集都没有公开, 所以并没有得到广泛的使用和研究.

5 HTAP 数据库关键技术的未来研究方向与挑战

当前, 如何针对行列共存型 HTAP 数据库进行列存数据管理、查询优化、资源调度以及系统评测, 仍然非常具有挑战性: 首先, 基于内存的列存储虽然能够通过列扫描加速查询, 并且利用压缩技术缩小存储空间, 但内存的容量毕竟有限, 当数据集很大且不断增长时, 还是无法把所有列存数据放入内存; 其次, 由于一个查询计划的算子既可以下推到行存执行, 又可以下推到列存执行, 因此, 针对不同数据访问路径, 如何自动地为查询优化器选择最优执行计划也非常关键; 同时, 为了协同执行 OLTP 和 OLAP 负载并保持数据一致性, 如何针对 HTAP 混合负载进行系统资源调度, 从而既考虑数据的一致性, 又提高系统性能也是一个难点; 最后, 虽然目前存在一些面向 HTAP 的基准测试, 但它们都存在许多不足, 有很大的改进空间. 总体来讲, HTAP 数据库技术包含内存列选择、行列混合查询优化、HTAP 资源调度、HTAP 评测基准这 4 个挑战性问题.

5.1 基于深度强化学习的内存列选择

深度强化学习将学习问题转化为马尔可夫决策过程, 代理通过试错的方式不断地与环境交互学习一个深度神经网络, 通过最大化与环境交互过程中获得的累计奖励, 从而找到解决问题的最优策略的方法[49,50]. 目前, 深度强化学习已被成功应用到数据库中的许多传统任务当中, 如参数调优[51,52]、连接顺序优化[53]、索引和视图自动选择[54]等. 针对内存列选择问题, 基于深度强化学习的方法很适合被用于动态搜索列组合, 从而帮助数据库系统自动选择所需要的列. 将深度强化学习应用到内存列选择问题主要有 3 个挑战: 首先, 如何尽量少地执行负载来搜索列组合是一个具有挑战性的问题, 这需要我们能找到更有用的候选集; 其次, 由于列与列之间的效用是有关联的, 针对一个列组合, 如何准确地估计它们所能产生的效用也非常具有挑战性; 最后, 由于列在内存中可以被压缩, 如果在列选择中考虑列的压缩算法选择[55], 如游程编码(run length encoding)、字典编码(dictionary encoding)、比特编码(bit-vector encoding), 则内存列选择问题将变得更加复杂.

5.2 面向HTAP的学习型查询优化器

针对 HTAP 的查询优化, 未来可考虑设计学习型算法[56,57]同时考虑行算子和列算子, 从而高效地搜索计划空间并生成高质量的执行计划. 特别地, 如果能够提前判断哪些算子更能帮助执行计划, 然后可以利用提示集(hint set)的方式生成对应查询计划. 例如: 面向行的提示集有取消循环嵌套连接、强制使用哈希索引等, 面向列的提示集有强制使用向量化连接与向量化聚集. 目前已有研究提出学习型的查询优化器[58−62]针对行算子建立不相交的划分集合, 然后利用强化学习的探索-利用(exploration-exploitation)机制, 学习到最优的提示集优化策略. 针对 HTAP 的查询优化器, 如何设计一个行列感知的学习型查询优化器非常具有挑战性.

- 首先, 选择最优化的提示集可以被形式化为一个基于上下文的多臂老虎机问题, 其中, 每一个臂是一个候选提示集, 上下文是当前提示集生成的计划集合, 目标是选择最优化的臂来最小化遗憾(regret), 即最优化提示集的计划收益与当前选择提示集的计划收益之差. 因此, 如何设计高效的算法以解决多臂老虎机问题是一个难点.
- 其次, 如何将行列混合算子划分为不相交的集合也是一个具有挑战性的问题, 因为这比之前只考虑行算子的情况更为复杂, 搜索空间更大.
- 最后, 在不实际执行查询的情况下, 如何准确评估查询计划使用提示集后的收益也是一个难点.

5.3 自适应的HTAP资源调度

给定一个 HTAP 负载, 在多种资源调度执行方式中选择一种执行方式可以被形式化为一个分类问题, 问题的目标是在一定资源的调度下, 最小化处理混合负载的代价以及数据分析新鲜度小于阈值的时间. 针对这一问题, 需要我们设计轻量级的、自适应负载的算法. 这里有 3 个关键性挑战: 首先, 负载会持续进入到系统, 并且低并发、高延时的 OLAP 负载与高并发、低延时的 OLTP 负载的访问特点各不相同; 其次, 系统在持续调度资源的同时, 还要保证数据分析新鲜度的要求; 最后, 如何考虑更细粒度的资源调度方式也非常具有挑战性. 例如, 图 10 的 3 种执行方式可以通过 CPU 线程、缓存等资源的进一步细粒度而划分为多种执行方式.

5.4 面向HTAP的评测基准

未来可以从 4 个方向改进现有 HTAP 评测基准: 首先, 未来可基于真实分布的数据集和工作负载建立不同 HTAP 场景(包括银行、金融、保险、电子商务等)的基准测试, 并考虑新鲜度指标; 其次, 由于 TPC-H 的数据模型和规模都很小, 查询负载不够丰富, 数据分布也主要以均匀和独立分布为主, 未来可加入更多的倾斜分布和数据关联到 TPC-H 中[63,64]; 此外, 目前的评测基准缺乏对 OLTP 执行的动态控制, 未来可设计更加多样的执行控制方式, 如考虑 OLTP 和 OLAP 的交叉执行或 OLAP 的连续执行伴随 OLTP 的间断执行等情况; 最后, 未来可设计微型评测基准针对特定的 HTAP 任务, 包括数据组织、数据同步、查询优化以及资源调度. 综上所述, 未来需要设计一个能够涵盖真实场景、可控制数据生成、多样性负载执行规则、全面评测 HTAP 数据库的评测基准套件.

6 总 结

本综述对近 10 年的具有代表性的 HTAP 数据库进行了梳理、总结与分类. 本文重点对行列共存型 HTAP 数据库的关键技术进行了详细介绍, 同时也介绍了面向 HTAP 数据库的构建技术与评测基准, 并对 HTAP 技术的未来研究方向进行了展望. 本文首先对 HTAP 数据库进行了总体梳理、分类以及特点比较. 我们将 HTAP 数据库按照存储架构分为四大类, 即: (1) 主行存与内存型列存; (2) 分布式行存与列存副本; (3) 单机磁盘型行存与分布式列存; (4) 主列存与增量型行存. 针对每一类系统的存储架构, 介绍了它们的主要存储策略与处理方法, 并列出了它们的代表性系统, 比较了它们各自处理 HTAP 负载的优缺点. 本文重点对 HTAP 数据库的关键技术进行了总结, 包括数据组织技术、查询优化技术、数据同步技术、资源调度技术这 4 个部分. 对每一类技术, 从处理性能、扩展性、数据新鲜度等方面比较了它们的优缺点. 本文凝炼总结了每一类技术类型下的关键技术, 并对每一种技术进行了详细介绍. 本文也介绍了面向 HTAP 数据库的构建技术, 包括基于内存列、基于列存索引以及基于分布式列存副本的构建. 本文还回顾了 HTAP 数据库的评测基准, 包括 CH-benCHmark、HTAPBench 以及 CBTR, 并详细介绍了它们的数据生成、执行规则以及评价指标. 最后, 本文讨论了当前 HTAP 关键技术的不足, 并展望了未来的研究方向, 包括内存列选择、行列混合查询优化、HTAP 资源调度、HTAP 评测基准这 4 个方面.

现代大规模事务处理与实时分析应用场景催生出了 HTAP 技术. 由于分布式处理、内存技术、行列共存等技术的发展, 使得实现高吞吐量与低延时的 HTAP 数据库成为了可能. 随着多核处理器、协处理器、GPU 等新硬件以及 AI 算法、云计算的不断发展, HTAP 数据库未来有着广阔的发展与应用前景.

致谢 本文由国家自然科学基金(61925205, 62072261, 62232009)、华为公司、好未来教育公司给予大力支持.

References:

[1] Özcan F, Tian YY, Tözün P. Hybrid transactional/analytical processing: A survey. In: Salihoglu S, Zhou WC, Chirkova R, Yang J, Suciu D, eds. Proc. of the Int'l Conf. on Management of Data. ACM, 2017. 1771−1775.

[2] Hieber D, Grambow G. Hybrid transactional and analytical processing databases: A systematic literature review. In: Popescu M, Agosto A, Giudici P, Sztandera L, eds. Proc. of the Data Analytics. Nice: IARIA XPS, 2020. 90−98.

[3] Lahiri T, Chavan S, Colgan M, Das D, Ganesh A, Gleeson M, Hase S, Holloway A, Kamp J, Lee TH. Oracle database in-memory: A dual format in-memory database. In: Gehrke J, Lehner W, Shim K, Cha SK, Lohman GM, eds. Proc. of the Int'l Conf. on Data Engineering. Seoul: IEEE, 2015. 1253−1258.

[4] Larson PÅ, Birka A, Hanson EN, Huang W, Nowakiewicz M, Papadimos V. Real-time analytical processing with SQL server. Proc. of the VLDB Endowment, 2015, 8(12): 1740−1751.

[5] Raman V, Attaluri G, Barber R, Chainani N, Kalmuk D, KulandaiSamy V, Leenstra J, Lightstone S, Liu S, Lohman GM. DB2 with BLU acceleration: So much more than just a column store. Proc. of the VLDB Endowment, 2013, 6(11): 1080−1091.

[6] 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: Candan KS, Chen Y, Snodgrass RT, Gravano L, Fuxman A, eds. Proc. of the Int'l Conf. on Management of Data. Scottsdale: ACM, 2012. 731−742.

[7] Huang DX, Liu Q, Cui Q, *et al*. TiDB: A raft-based HTAP database. Proc. of the VLDB Endowment, 2020, 13(12): 3072−3084.

[8] Yang J, Rae I, Xu J, *et al*. *F*1 lightning: HTAP as a service. Proc. of the VLDB Endowment, 2020, 13(12): 3313−3325.

[9] MySQL Heatwave. Real-time Analytics for MySQL Database Service. Public Documentation Version 3.0. 2021. 1−20.

[10] Oracle. Database In-Memory Guide. Public Documentation 21c. 2021. 71−88.

[11] Boissier M, Schlosser R, Uflacker M. Hybrid data layouts for tiered HTAP databases with pareto-optimal data placements. In: Proc. of the Int'l Conf. on Data Engineering. Paris: IEEE, 2018. 209−220.

[12] Alagiannis I, Idreos S, Ailamaki A. H2O: A hands-free adaptive store. In: Dyreson CE, Li FF, Ozsu MT, eds. Proc. of the Int'l Conf. on Management of Data. Snowbird: ACM, 2014. 1103−1114.

[13] Athanassoulis M, Bøgh KS, Idreos S. Optimal column layout for hybrid workloads. Proc. of the VLDB Endowment, 2019, 12(13): 2393−2407.

[14] Arulraj J, Pavlo A, Menon P. Bridging the archipelago between row-stores and column-stores for hybrid workloads. In: Ozscan F, Koutrika G, Madden S, eds. Proc. of the Int'l Conf. on Management of Data. San Francisco: ACM, 2016. 583−598.

[15] Abebe M, Lazu H, Daudjee K. Proteus: Autonomous adaptive storage for mixed workloads. In: Bonifati A, Abbadi AE, Barcelo P, eds. Proc. of the Int'l Conf. on Management of Data. ACM, 2022. 1−15.

[16] 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.

[17] Lee R, Zhou MH, Li C, Hu SG, Teng JP, Li DY, Zhang XD. The art of balance: A RateupDB experience of building a CPU/GPU hybrid database product. Proc. of the VLDB Endowment, 2021, 14(12): 2999−3013.

[18] Sun YH, Blelloch GE, Lim WS, Pavlo A. On supporting efficient snapshot isolation for hybrid workloads with multi-versioned indexes. Proc. of the VLDB Endowment, 2019, 13(2): 211−225.

[19] Riegger C, Vinçon T, Gottstein R, Petrov I. MV-PBT: Multi-version indexing for large datasets and HTAP workloads. In: Bonifati, Zhou YL, Salles MAV, Bohm A, Olteanu D, eds. Proc. of the Int'l Conf. on Extending Database Technology. Copenhagen: OpenProceedings.org, 2020. 217−228.

[20] Psaroudakis I, Scheuer T, May N, Ailamaki A. Task scheduling for highly concurrent analytical and transactional main-memory workloads. In: Bordawekar R, Lang CA, Gedik B, eds. Proc. of the Int'l Workshop on Accelerating Data Management Systems Using Modern Processor and Storage Architectures. Trento: VLDB, 2013. 36−45.

[21] Sirin U, Dwarkadas S, Ailamaki A. Performance characterization of HTAP workloads. In: Proc. of the Int'l Conf. on Data Engineering. Chania: IEEE, 2021. 1829−1834.

[22] Raza A, Chrysogelos P, Anadiotis AC, Ailamaki A. Adaptive HTAP through elastic resource scheduling. In: Maier D, Pottinger R, Doan AH, Tan WC, Alawini A, Ngo HQ, eds. Proc. of the Int'l Conf. on Management of Data. Portland: ACM, 2020. 2043−2054.

[23] Kemper A, Neumann T. HyPer: A hybrid OLTP&OLAP main memory database system based on virtual memory snapshots. In: Abiteboul S, Bohm K, Koch C, Tan KL, eds. Proc. of the Int'l Conf. on Data Engineering. Hannover: IEEE, 2011. 195−206.

[24] Makreshanski D, Giceva J, Barthels C, Alonso G. BatchDB: Efficient isolated execution of hybrid OLTP+OLAP workloads for interactive applications. In: Salihoglu S, Zhou WC, Chirkova R, Yang J, Suciu D, eds. Proc. of the Int'l Conf. on Management of Data. Chicago: ACM, 2017. 37−50.

[25] Zaharia M, Chowdhury M, Franklin MJ, Shenker S, Stoica I. Spark: Cluster computing with working sets. In: Naham EM, Xu DY, eds. Proc. of the 2nd USENIX Workshop on Hot Topics in Cloud Computing. Boston: USENIX, 2010. 1−7.

[26] Barber R, Garcia-Arellano C, Grosman R, *et al*. Evolving databases for new-gen big data applications. In: Proc. of the 8th Biennial Conf. on Innovative Data Systems Research. Chaminade: CIDR, 2017. 1−8.

[27] Mozafari B, Ramnarayan J, Menon S, Mahajan Y, Chakraborty S, Bhanawat H, Bachhav K. SnappyData: A unified cluster for streaming, transactions and interactice analytics. In: Proc. of the 8th Biennial Conf. on Innovative Data Systems Research. Chaminade: CIDR, 2017. 1−8.

[28] Apache HBase. Reference Guide. Pulic Documentation Version 3.0. 2021. 444−455.

[29] Bouzeghoub M, Peralta V. A framework for analysis of data freshness. In: Nauman F, Scannapieco M, eds. Proc. of the Int'l Workshop on Information Quality in Information Systems. Paris: ACM, 2004. 59−67.

[30] Dziedzic A, Wang JJ, Das S, Ding BL, Narasayya VR, Syamala M. Columnstore and B+ tree—Are hybrid physical designs important? In: Das G, Jermaine CM, Bernstein PA, eds. Proc. of the Int'l Conf. on Management of Data. Houston: ACM, 2018. 177−190.

[31] 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: Ross KA, Srivastava D, Papadias D, eds. Proc. of the Int'l Conf. on Management of Data. New York: ACM, 2013. 1243−1254.

[32] Lamport L. Paxos made simple. ACM SIGACT News (Distributed Computing Column), 2001, 32(4): 51−58.

[33] Diego O, Ousterhout JK. In search of an understandable consensus algorithm. In: Gibson G, Zeldovich N, eds. Proc. of the USENIX Annual Technical Conf. Philadelphia: USENIX, 2014. 305−319.

[34] Corbett JC, Dean J, Epstein M, *et al*. Spanner: Google's globally distributed database. ACM Trans. on Computer System, 2013, 31(3): 1−22.

[35] Goel AK, Pound J, Auch N, Bumbulis P, MacLean S. Towards scalable real-time analytics: An architecture for scale-out of OLxP workloads. Proc. of the VLDB Endowment, 2015, 8(12): 1716−1727.3.

[36] Färber F, May N, Lehner W, Große P, Müller I, Rauhe H, Dees J. The SAP HANA database—An architecture overview. IEEE Database Engineering Bulletin, 2012, 35(1): 28−33.

[37] Shen SJ, Chen R, Chen HB, Zang BY. Retrofitting high availability mechanism to tame hybrid transaction/analytical processing. In: Brown AD, Lorch JR, eds. Proc. of the 15th USENIX Symp. on Operating Systems Design and Implementation. Virtual Event: USENIX, 2021. 219−238.

[38] Neumann T, Mühlbauer T, Kemper A. Fast serializable multi-version concurrency control for main-memory database systems. In: Sellis TK, Davison SB, I, eds. Proc. of the Int'l Conf. on Management of Data. Melbourne: ACM, 2015. 677−689.

[39] Grund M, Krüger J, Plattner H, Zeier A, Cudre-Mauroux P, Madden S. Hyrise: A main memory hybrid storage engine. Proc. of the VLDB Endowment, 2010, 4(2): 105−116.

[40] Borthakur D. HDFS architecture guide. Hadoop Apache Project, 2008. 1−14.

[41] Vohra D. Apache Parquet. Practical Hadoop Ecosystem. Berkeley: Springer, 2016. 325−335.

[42] Pei W, Li ZH, Pan W. Survey of key technologies in GPU database system. Ruan Jian Xue Bao/Journal of Software, 2021, 32(3): 859−885 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6175.htm [doi: 10.13328/j.cnki.jos.006175]

[43] Gallet B, Gowanlock M. Heterogeneous CPU-GPU epsilon grid joins: Static and dynamic work partitioning strategies. Data Science and Engineering, 2021, 6(1): 39−62.

[44] 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: Nambiar R, Posess M, eds. Proc. of the Technology Conf. on Performance Evaluation and Benchmarking. Hangzhou: Sprinder, 2014. 97−112.

[45] Cole R, Funke F, Giakoumakis L, Guy W, Kemper A, Krompass S, Kuno H, Nambiar R, Neumann T, Poess M. The mixed workload CH-benCHmark. In: Graefe. G, Salem K, eds. Proc. of the 4th Int'l Workshop on Testing Database Systems. Athen: ACM, 2011. 1−6.

[46] Coelho F, Paulo J, Vilaça R, Pereira J, Oliveira R. HTAPBench: Hybrid transactional and analytical processing benchmark. In: Binder W, Cortellessa V, Koziolek A, Smirni E, Poess M, eds. Proc. of the 8th ACM/SPEC on Int'l Conf. on Performance Engineering. ACM, 2017. 293−304.

[47] Bog A, Plattner H, Zeier A. A mixed transaction processing and operational reporting benchmark. Information Systems Frontiers, 2011, 13(3): 321−335.

[48] Bog A, Sachs K, Plattner H. Interactive performance monitoring of a composite OLTP and OLAP workload. In: Candan KS, Chen Y, Snodgrass RT, Gravano L, Fuxman A, eds. Proc. of the 2012 ACM SIGMOD Int'l Conf. on Management of Data. Scottsdale: ACM, 2012. 645−648.

[49] Chai MK, Fan J, Du XY. Learnable database systems: Challenges and opportunities. Ruan Jian Xue Bao/Journal of Software, 2020, 31(3): 806−830 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5908.htm [doi: 10.13328/j.cnki.jos.005908]

[50] Li GL, Zhou XH. Survey of data management techniques for supporting artificial intelligence. Ruan Jian Xue Bao/Journal of Software, 2021, 32(1): 21−40 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6121.htm [doi: 10.13328/j.cnki.jos.006121]

[51] Li GL, Zhou XH, Li SF, Gao B. Qtune: A query-aware database tuning system with deep reinforcement learning. Proc. of the VLDB Endowment, 2019, 12(12): 2118−2130.

[52] Zhang J, Liu Y, Zhou K, Li GL, Xiao ZL, Cheng B, Xing JS, Wang YT, Cheng TH, Liu L, Ran MW, Li ZK. An end-to-end automatic cloud database tuning system using deep reinforcement learning. In: Bonce PA, Manegold S, Ailamaki A, Deshpande A, Kraska T, eds. Proc. of the Int'l Conf. on Management of Data. Amasterdam: ACM, 2019. 415−432.

[53] Yu X, Li GL, Chai CL, Tang N. Reinforcement learning with tree-LSTM for join order selection. In: Proc. of the Int'l Conf. on Data Engineering. Dallas: IEEE, 2020. 1297−1308.

[54] Yuan HT, Li GL, Feng L, Sun J, Han Y. Automatic view generation with deep learning and reinforcement learning. In: Proc. of the Int'l Conf. on Data Engineering. Dallas: IEEE, 2020. 1501−1512.

[55] Boissier M. Robust and budget-constrained encoding configurations for in-memory database systems. Proc. of the VLDB Endowment, 2021, 15(4): 780−793.

[56] Li GL, Zhou XH, Cao L. AI meets database: AI4DB and DB4AI. In: Li GL, Li ZH, Idreos S, Srivastava, eds. Proc. of the Int'l Conf. on Management of Data. Virtual Event: ACM, 2021. 2859−2866.

[57] Li GL, Zhou XH. Machine learning for data management: A system view. In: Proc. of the Int'l Conf. on Data Engineering. Virtual Event: IEEE, 2022. 1−4.

[58] Zhou XH, Chai CL, Li GL, Sun J. Database meets artificial intelligence: A survey. IEEE Trans. on Knowledge Data Engineering, 2022, 34(3): 1096−1116.

[59] Li GL, Zhou XH, Sun J, Yu X, Han Y, Jin LY, Li WB, Wang TQ, Li SF. openGauss: An autonomous database system. Proc. of the VLDB Endowment, 2021, 14(12): 3028−3041.

[60] Marcus R, Negi P, Mao HZ, Tatbul N, Alizadeh M, Kraska T. Bao: Making learned query optimization practical. In: Li GL, Li ZH, Idreos S, Srivastava, eds. Proc. of the Int'l Conf. on Management of Data. Virtual Event: ACM, 2021. 1275−1288.

[61] Marcus R, Negi P, Mao HZ, Zhang C, Alizadeh M, Kraska T, Papaemmanouil O, Tatbul N. Neo: A learned query optimizer. Proc. of the VLDB Endowment, 2019, 12(11): 1705−1718.

[62] Lan H, Bao ZF, Peng YW. A survey on advancing the DBMS query optimizer: Cardinality estimation, cost model, and plan enumeration. Data Science and Engineering, 2021, 6(1): 86−101.

[63] Boncz P, Anatiotis AC, Kläbe S. JCC-H: Adding join crossing correlations with skew to TPC-H. In: Nambiar R, Poess M, eds. Proc. of the Technology Conf. on Performance Evaluation and Benchmarking. Munich: Springer, 2017. 103−119.

[64] Leis V, Gubichev A, Mirchev A, Boncz P, Kemper A, Neumann T. How good are query optimizers, really? Proc. of the VLDB Endowment, 2015, 9(3): 204−215.

附中文参考文献:

[42] 裴威, 李战怀, 潘巍. GPU 数据库核心技术综述. 软件学报, 2021, 32(3): 859−885. http://www.jos.org.cn/1000-9825/6175.htm [doi: 10.13328/j.cnki.jos.006175]

[49] 柴茗珂, 范举, 杜小勇. 学习式数据库系统: 挑战与机遇. 软件学报, 2020, 31(3): 806−830. http://www.jos.org.cn/1000-9825/5908.htm [doi: 10.13328/j.cnki.jos.005908]

[50] 李国良, 周煊赫. 面向 AI 的数据管理技术综述. 软件学报, 2021, 32(1): 21−40. http://www.jos.org.cn/1000-9825/6121.htm [doi: 10.13328/j.cnki.jos.006121]

张超(1990－), 男, 博士, 助理研究员, CCF 专业会员, 主要研究领域为数据库与大数据管理技术.

冯建华(1967－), 男, 博士, 教授, 博士生导师, CCF 杰出会员, 主要研究领域为数据库, 数据安全与隐私保护, 信息检索.

李国良(1980－), 男, 博士, 教授, 博士生导师, CCF 杰出会员, 主要研究领域为数据库, 大数据分析和挖掘, 群体计算.

张金涛(2000－), 男, 硕士生, 主要研究领域为数据库与机器学习的交叉技术.