软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2024,35(2):899−926 [doi: 10.13328/j.cnki.jos.006952]

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

云原生数据库综述*

董昊文, 张 超, 李国良, 冯建华

(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: 云基础设施的虚拟化、高可用、可弹性调度等特点, 为云数据库提供了开箱即用、可靠可用、按需计费等优势. 云数据库按照架构可以划分为云托管数据库 (cloud-hosted database) 以及云原生数据库 (cloud-native database). 云托管数据库将数据库系统直接部署到云上虚拟机环境中, 具备低成本、易运维、高可靠的优势. 在此基础上, 云原生数据库充分利用云基础设施弹性伸缩的特点, 采用计算存储分离的架构, 实现了计算资源和存储资源的独立伸缩, 进一步提升数据库性价比. 然而计算存储分离的架构为数据库系统设计带来了新的挑战. 深入分析云原生数据库系统的架构和技术. 首先将云原生 OLTP 和云原生 OLAP 的数据库架构按照资源分离模式的差异分别进行归类分析, 并对比各类架构的优势与局限. 其次, 基于计算存储分离的架构, 按照各个功能模块深入探讨云原生数据库的关键技术: 主要包括云原生 OLTP 关键技术 (数据组织、副本一致性、主备同步、故障恢复以及混合负载处理) 和云原生 OLAP 关键技术 (存储管理、查询处理、无服务器感知计算、数据保护以及机器学习优化). 最后, 总结现有云原生数据库的技术挑战并展望未来研究方向.

关键词: 云原生数据库; 计算存储分离; 数据库即服务

中图法分类号: TP311

中文引用格式: 董昊文, 张超, 李国良, 冯建华. 云原生数据库综述. 软件学报, 2024, 35(2): 899–926. http://www.jos.org.cn/1000-9825/6952.htm

英文引用格式: Dong HW, Zhang C, Li GL, Feng JH. Survey on Cloud-native Databases. Ruan Jian Xue Bao/Journal of Software, 2024, 35(2): 899–926 (in Chinese). http://www.jos.org.cn/1000-9825/6952.htm

Survey on Cloud-native Databases

DONG Hao-Wen, ZHANG Chao, LI Guo-Liang, FENG Jian-Hua

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

Abstract: The virtualization, high availability, high scheduling elasticity, and other characteristics of cloud infrastructure provide cloud databases with many advantages, such as the out-of-the-box feature, high reliability and availability, and pay-as-you-go model. Cloud databases can be divided into two categories according to the architecture design: cloud-hosted databases and cloud-native databases. Cloud-hosted databases, deploying the database system in the virtual machine environment on the cloud, offer the advantages of low cost, easy operation and maintenance, and high reliability. Besides, cloud-native databases take full advantage of the characteristic elastic scaling of the cloud infrastructure. The disaggregated compute and storage architecture is adopted to achieve the independent scaling of computing and storage resources and further increase the cost-performance ratio of the databases. However, the disaggregated compute and storage architecture poses new challenges to the design of database systems. This survey is an in-depth analysis of the architecture and technology of the cloud-native database system. Specifically, the architectures of cloud-native online transaction processing (OLTP) and online analytical processing (OLAP) databases are classified and analyzed, respectively, according to the difference in the resource disaggregation mode, and the advantages and limitations of each architecture are compared. Then, on the basis of the disaggregated compute and storage architectures, this study explores the key technologies of cloud-native databases in depth by functional modules. The technologies under

* 基金项目: 国家自然科学基金 (62232009, 61925205, 62072261)
收稿时间: 2022-10-25; 修改时间: 2022-12-11; 采用时间: 2023-04-06; jos 在线出版时间: 2023-08-23
CNKI 网络首发时间: 2023-08-28

discussion include those of cloud-native OLTP (data organization, replica consistency, main/standby synchronization, failure recovery, and mixed workload processing) and those of cloud-native OLAP (storage management, query processing, serverless-aware compute, data protection, and machine learning optimization). At last, the study summarizes the technical challenges for existing cloud-native databases and suggests the directions for future research.

Key words: cloud-native database; compute-storage disaggregation; database-as-a-service (DBaaS)

近年来云服务技术的普及和发展推动了计算机应用云端化的进程. 云服务开箱即用、按需计费、弹性伸缩等特性为数据库系统领域也带来了全新的发展机遇, 越来越多的本地部署的数据库系统已经向云端迁移[1]. 从市场规模来看, 2022 年数据库系统全球整体的市场规模约为 653 亿美元[1], 其中数据库即服务 (database-as-a-service) 的全球市场规模约为 135 亿美元[2], 占比约为 20.7%. 按照目前的发展趋势来看, 未来几年间云数据库市场规模的增长速度 (年平均增长率预期为 15.7%) 将高于总体数据库市场规模增长速度 (年平均增长率预期为 10.8%). 这将进一步扩大云数据库系统在整体数据库领域中的市场占比, 得益于云环境的发展, 亚马逊、微软、谷歌等大型云数据库公司取得了明显的市场扩张, 在市场份额和市场利润等方面已经逐步追赶甚至超越 Oracle、IBM 等传统数据库系统巨头公司[3]. 因此, 云数据库系统将在下一代数据库管理系统的发展中扮演重要角色.

云数据库系统的快速发展得益于其所具备的多方面优势[4]. 这些优势可以从云数据库系统的使用者和提供商两个角度进行分析.

● 对于云数据库使用者而言, 云数据库系统相较于传统数据库系统具备 4 个方面的优势: (1) 弹性: 云数据库可以基于工作负载实现自动扩缩容, 使实际资源使用量实时匹配于工作负载, 避免因预先部署资源不足导致的业务容量瓶颈. (2) 可用性: 云数据库同时维护多个计算和存储副本以保证系统的高可用性, 同时底层云基础设施跨地域的部署方式能够有效应对单一数据中心的极端灾难. (3) 灵活性: 开箱即用的特性让用户免于复杂的数据库部署过程; 系统支持自动化的运维调优, 降低用户的管理压力. (4) 低成本: 按需计费模型让用户仅需为实际资源使用量付费, 而非传统的预置模式. 结合实际工作负载的波动性, 云数据库能大幅降低用户的使用成本.

● 对于云数据库提供商而言, 云数据库系统相较于传统数据库系统具备 3 个方面的优势: (1) 市场份额扩大: 随着大数据时代的到来, 数据处理需求扩张迅速, 云数据库能吸引缺乏专业数据库运维团队的小规模企业, 从而拓展新的客户群体. (2) 降低平均成本: 规模效应下, 大规模的数据中心能降低单位硬件资源的建设成本以及采购成本. 此外, 整体性的集群设计和管理团队能够降低平均运维成本. (3) 提高资源利用效率: 云数据库系统中资源不会与用户绑定, 而是按照工作负载动态分配给有需求的用户. 因此, 提供商可以将一份硬件资源分时段地分配给多位用户, 实现资源超卖以避免资源的闲置

纵观近十年的发展历程, 云数据库系统可分为两个发展阶段: (1) 计算存储耦合架构的云托管数据库系统 (Amazon RDS for MySQL[5]、Azure SQL Database[6]、Google Cloud SQL[7]等), 以及 (2) 计算存储分离架构的云原生数据库系统 (Amazon Aurora[8]、Snowflake[9]、Google BigQuery[10]等). 云托管数据库系统直接将传统数据库迁移到云上虚拟机环境中, 最大程度复用已有传统数据库的架构设计和代码实现. 系统根据用户需求动态调整租用的虚拟机数量和配置, 具备虚拟机级别的弹性资源调度能力. 其最大优势在于其极低的额外开发成本, 即不需要大规模重构传统数据库系统的代码实现就能够完成数据库从本地到云端的迁移. 但是虚拟机级别的资源调度将计算和存储资源绑定, 调度过程中将受制于资源耦合, 不能够充分发挥不同类型云服务的弹性调度能力. 此外, 虚拟机的启动和停止存在较长时间的延时, 这对于调度的管理提出了严峻的考验[11−13]. 为了解决上述的问题, 云原生数据库系统旨在利用不同类型云基础服务独立伸缩的特点, 以达到优化弹性调度能力、提高资源利用效率的目的. 如图 1 所示, 云原生数据库系统采用计算存储分离的架构设计[8]: 计算模块负责查询处理、事务管理、并发控制、查询优化等计算密集型工作, 由云计算服务支持; 存储模块负责处理日志存储、存储管理、备份以及恢复等存储密集型工作, 由云存储服务支持. 计算存储分离的设计不适用于传统数据库系统的数据管理模式, 包括主备节点之间的数据传输方式、数据一致性保证、数据恢复等. 因此, 云原生数据库系统提出了一系列关键技术, 如日志即数据、数据异步回放、基于共享缓存的数据恢复以及基于云存储层的计算下推等.

数据库系统的上层应用主要包含两类工作负载, 包括 (1) 面向事务处理场景的 OTLP (online transactional

processing) 工作负载和 (2) 面向数据分析场景的 OLAP (online analytical processing) 工作负载. 二者具备完全不同的特性, 前者重视事务的 ACID (atomicity, consistency, isolation, and durability) 特性和事务的并发处理能力, 而后者重视大规模批量数据分析的效率. 相应地, 云原生数据库可被划分为两种类型: (1) 云原生 OLTP 数据库系统, 如 Amazon Aurora[8]、Azure HyperScale[14]、阿里 PolarDB Serverless[15]等; 以及 (2) 云原生 OLAP 数据库系统, 如 Amazon Redshift[16, 17]、Snowflake[9]、Google BigQuery[10]等. 虽然两类系统都采用计算存储分离的架构, 但由于它们面对不同的工作场景, 因此二者在查询处理流程和核心问题上存在明显差异. 云原生 OLTP 数据库需要基于日志保证事务的 ACID 特性, 即在事务处理过程中要求保证事务的原子性、持久性、副本一致性、故障恢复能力等关键特性. 此外, 云原生 OLTP 数据库的核心问题在于如何既能利用计算存储分离架构所带来的优势, 又能同时降低 I/O 开销、加速主备节点数据同步以及加速故障恢复. 云原生 OLAP 数据库旨在基于云端对象存储提供高扩展、高弹性、高可用的数据分析能力, 因此, 如何基于云存储进行元数据管理、数据组织、查询优化以及数据保护是其核心问题. 此外, 随着无服务器感知与机器学习技术的普及, 云原生 OLAP 数据库也可进一步优化其查询服务的成本与效率.

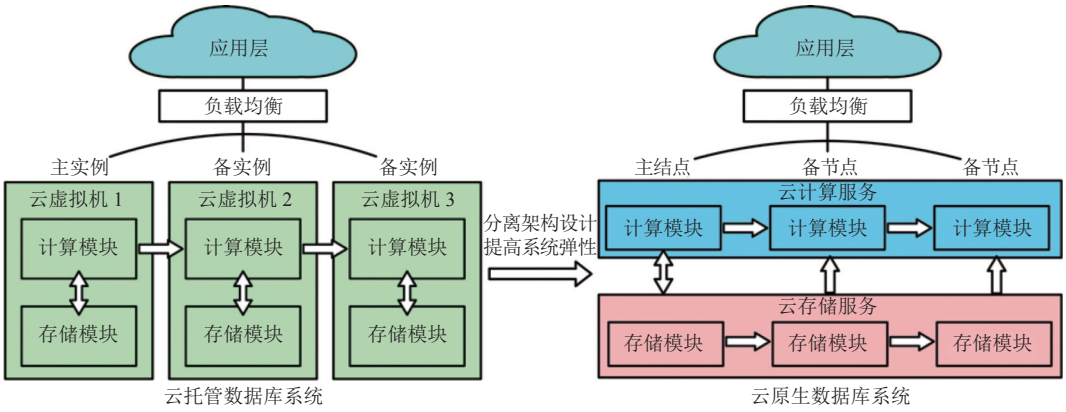


图 1　云托管与云原生数据库架构差异

本文的主要贡献在于对于云原生关系型数据库系统现有的工作进行了全面的调研和综述. 以云原生数据库的计算存储分离架构为出发点, 从架构设计和关键技术两个重要维度分析现有的云原生 OLTP 和 OLAP 数据库系统. 本文将云原生 OLTP 数据库划分为 3 类不同架构并进行横向对比 (见第 1 节), 之后从数据组织、副本一致性、主备同步、故障恢复以及 HTAP (hybrid transaction and analytical processing) 这 5 个方面深入剖析其技术细节 (见第 2 节). 同时将云原生 OLAP 系统划分为两类不同架构 (见第 3 节), 之后从存储管理、查询处理、无服务器感知、数据保护以及机器学习优化这 5 个方面展开讨论其技术细节 (见第 4 节). 最后, 本文将结合现有工作面临的主要技术挑战展望云原生数据库系统技术的未来研究方向, 提出云原生多写技术、无服务感知的资源调度、HTAP 负载优化以及面向多云的数据管理技术这 4 个未来关键的研究方向 (见第 5 节). 整体来看, 本文从架构和技术两个维度展开探讨现有云原生 OLTP 和 OLAP 数据库系统, 以计算存储分离的设计为根本展示了现有关系型云原生数据库研究工作的全景, 并对于未来研究方向进行展望.

1　云原生 OLTP 数据库架构

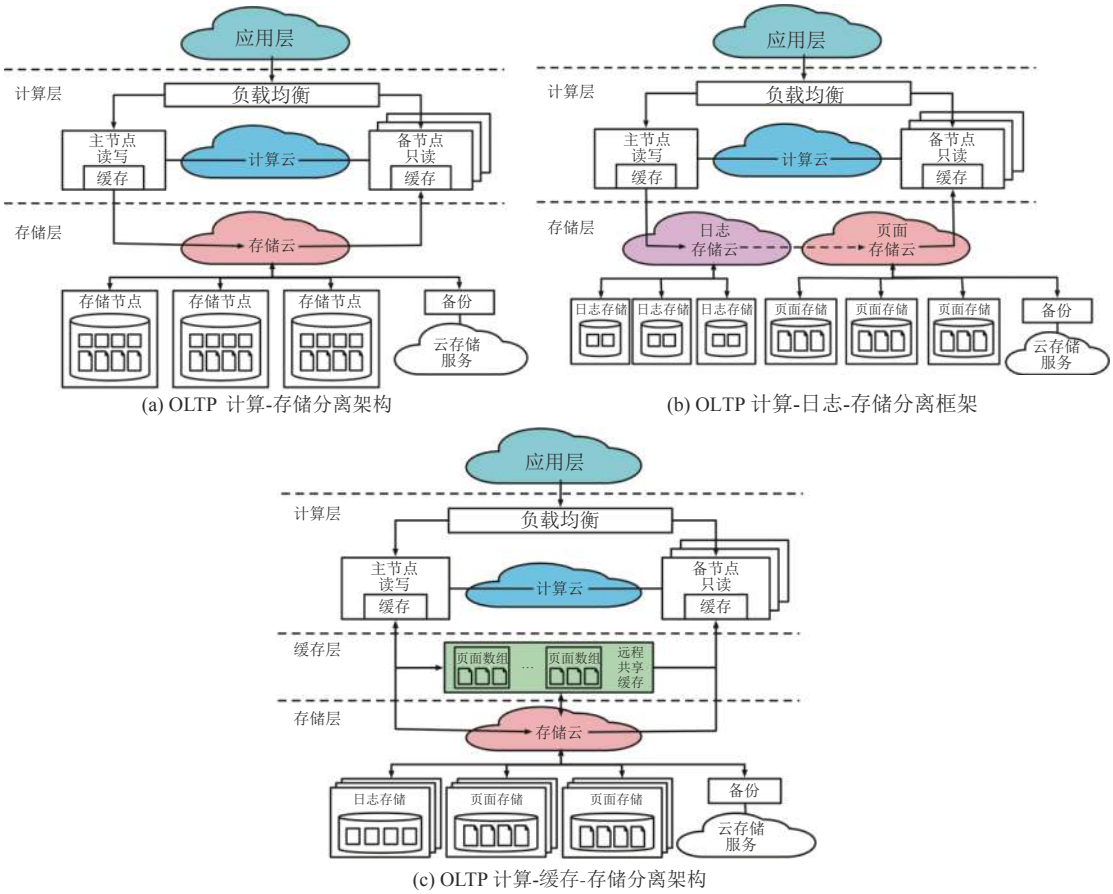
OLTP 数据库系统适用于事务处理场景, 系统需要在保证事务处理最基本 ACID 准则的基本前提下提高事务处理的吞吐能力. 首先, 传统数据库 (以及云托管数据库) 为了提升性能, 采用计算存储耦合的设计方式, 不能够支持单一类型资源的独立调度, 因此易于造成因资源绑定而产生的资源浪费. 云原生数据库采用计算存储分离的方式, 实现不同类型资源独立伸缩, 降低了成本. 其次, 传统数据库 (以及云托管数据库) 为了提升性能, 需要将重做日志 (redo log) 和记录页面刷新到存储介质中, 而基于脏页刷盘进行数据更新的方式面临严重的写放大问题[18−20],

在以网络 I/O 为瓶颈的云环境中将严重限制系统整体性能[20]. 为了解决写放大问题, 云原生 OLTP 系统提出了日志即数据的技术, 即只刷新重做日志, 脏页面不写回到持久存储, 且存储层内不同节点通过异步回放重做日志的方式来更新记录页面.

按照对于不同类型云服务分离管理模式的差异, 现有云原生 OLTP 数据库系统可以划分为如下 3 类架构设计: (1) 计算-存储分离架构; (2) 计算-日志-存储分离架构; (3) 计算-缓存-存储分离架构. 后续的 3 个小节将从架构设计的动机、数据读写路径、系统优势与限制、典型系统这 4 个维度横向对比这 3 类不同的架构, 详细介绍各类架构之间的异同之处.

1.1 OLTP 计算-存储分离架构

第 1 类云原生 OLTP 数据库系统采用计算-存储分离的架构, 整体架构如图 2(a) 所示.



(a) OLTP 计算-存储分离架构

(b) OLTP 计算-日志-存储分离框架

(c) OLTP 计算-缓存-存储分离架构

图 2 云原生 OLTP 数据库 3 类架构设计

(1) 设计动机. 从设计的动机来看, 第 1 类架构具备如下 3 个特点: (i) 从弹性方面考虑, 计算资源和存储资源可以实现独立调度, 避免虚拟机内部计算存储耦合产生的资源浪费; (ii) 从效率方面考虑, 持久化存储的更新依赖于日志回放, 减少计算存储层间的脏页面传输, 缓解写放大问题; (iii) 从可用性方面考虑, 系统采用了多层次的故障恢复策略, 支持单个计算或存储节点级别的故障恢复, 具备更低的平均故障恢复延时. 按照功能模块划分为计算层和存储层: 其中计算层负责查询优化、事务管理、并发控制以及本地数据缓存等计算密集型工作, 底层由云计算服务支持; 存储层负责日志管理、存储管理以及备份和恢复等存储密集型工作, 底层由云存储服务支持.

(2) 数据读写路径. 数据写入路径方面, 主节点接收数据更新查询并生成对应的重做日志, 将重做日志写回到

存储层中多数派的存储节点, 即视为写入操作完成. 然后异步完成备计算节点同步以及存储节点间的同步过程. 在这类架构下, 存储节点内同时存储重做日志以及记录页面, 并异步完成日志数据的回放. 数据读取路径方面, 计算层节点优先通过本地缓存获取数据, 当缓存失效时向存储层请求最新数据. 存储节点内部异步将日志回放到页面, 同时节点支持直接解析未完成回放的日志数据, 避免阻塞计算层的读取请求.

(3) 优势与局限. 相较于云托管数据库, 计算-存储分离架构的云原生 OLTP 数据库具备如下 3 个优势: (i) 降低数据写延时: 日志写入与记录页面同步更新分离 (异步), 副本间的记录页面同步不会阻塞写入过程; (ii) 减少写放大问题: 系统采用重做日志回放实现存储层记录页面的数据更新, 避免计算存储层间的脏页面传输从而减少网络开销; (iii) 提高弹性调度能力: 计算和存储采用不同的云服务, 资源调度过程完全分离. 系统的主要限制在于: 计算层备节点缓存失效时读取延时较高, 需要通过存储层请求数据. 存储节点在重做日志未完成回放时, 无法直接从记录页面读取数据, 需要面临在线解析日志获取最新数据的额外延时.

(4) 典型系统. 基于计算-存储分离架构设计的云原生数据库系统包含 Amazon Aurora[8]、Google AlloyDB[21]. 二者采用了类似的架构设计, 主要区别在于功能模块的实现上采用了不同的技术. Aurora 数据库是首个提出日志即数据的云原生数据库, 其同时针对底层存储优化了现有的 Quorum 机制, 即通过增加副本数量提高了存储层的可用性和容错能力; 同时设计了 Quorum 集合机制来维护副本运行状态, 在此基础上利用备用副本提高了故障的恢复速度. AlloyDB 数据库增加了系统对于 HTAP 工作负载的支持: 其通过计算层内自动行存转列存的方式来提升数据查询的效率.

1.2 OLTP 计算-日志-存储分离架构

第 2 类云原生 OLTP 数据库系统采用了计算-日志-存储分离的架构, 整体架构如图 2(b) 所示.

(1) 设计动机. 从设计的动机来看, 第 2 类架构相较于第 1 类架构而言, 存储层被进一步划分为日志存储云以及页面存储云, 以提升系统弹性. 其中日志存储云和页面存储云分别处理日志和页面的存储管理、备份以及故障恢复等工作. 以微软 Azure 提供的云存储服务为例[22], 在存储容量为 32 TB 的情况下, 相较于标准 HDD (hard disk drive) 存储服务, 优质 SSD (solid state drive) 存储服务需要支付约 3 倍的存储成本, 从而提供约 10 倍的吞吐以及 I/O 速度. 因为日志是顺序写入存储, 如采用高速云存储服务 (例如 SSD 存储服务), 可提升一个数量级的写入速度, 同时页面存储可采用相对廉价的云存储服务 (例如 HDD 存储服务) 来降低成本. 第 2 类架构具备如下两个特点: (i) 从效率方面考虑, 日志存储采用高速存储可以降低日志写入延时, 进而降低系统写延时. (ii) 从弹性方面考虑, 日志和页面采用不同类型的存储服务, 两种存储资源独立分配, 进一步提高了系统的弹性调度能力.

(2) 数据读写路径. 数据写入路径方面, 第 2 类架构的写入过程与第一类架构类似, 同样采用持久化重做日志的方法完成事务的数据更新. 但是不同于第 1 类架构之处在于, 第 2 类架构将日志存储和页面存储分离, 重做日志的持久化由高速的日志存储云服务完成, 因此系统的写入延时更低. 异步处理由日志存储云向页面存储云传输重做日志, 由页面存储节点完成重做日志的回放过程. 数据读取路径方面, 计算层优先通过本地缓存获取数据, 当缓存失效时向页面存储云请求最新数据. 值得注意的是, 数据读取过程中计算层不会直接访问日志存储的任何数据. 当日志数据没有同步到页面存储时, 计算层会等待存储层完成日志回放.

(3) 优势与局限. 相较于第一类架构而言, 计算-日志-存储分离架构的云原生 OLTP 数据库具备两大优势: (i) 更低的数据写延时. 这主要得益于日志存储采用的高速云存储服务, 因此系统的写入延时能够进一步降低; (ii) 更高的弹性调度能力. 因为日志存储和页面存储采用不同类型的存储服务, 且资源调度互相独立, 所以进一步强化弹性调度能力. 然而, 当缓存失效时, 第 2 类架构相比于第 1 类架构读取延时更高, 这是因为计算层需要等待页面存储回放重做日志. 此外, 日志和页面的分离管理面临更复杂的故障恢复情景. 例如, 所有页面存储节点均丢失中间日志, 仅能通过日志存储恢复的场景, 这提高了故障恢复算法的复杂性.

(4) 典型系统. 基于计算-日志-存储分离架构设计的云原生数据库系统包含 Azure HyperScale[14]、华为 Taurus[23]. 二者采用了类似的架构设计, 主要区别在于读写请求处理模块的实现, 由于日志存储与日志存储分别管理, 该模块需要管理数据在存储层的状态, 尤其是数据从日志存储到页面存储的同步状态. 二者在这个功能模块的

实现上采用了不用的方案. HyperScale 在计算层和存储层之间额外添加 XLOG 服务层负责这部分功能. 而 Taurus 数据库在计算层的每个节点上添加额外的存储抽象层来负责该功能. 前者独立的功能模块设计有利于弹性伸缩, 而后者则分散模块的处理压力, 避免中心式的管理成为系统瓶颈.

1.3 OLTP 计算-缓存-存储分离架构

第 3 类云原生 OLTP 数据库系统采用了计算-缓存-存储分离的架构, 整体架构如图 2(c) 所示.

(1) 设计动机. 第 3 类架构相较于前两类架构而言, 借助于远程内存服务额外增加了所有计算节点共享的远程缓存, 实现了对于本地缓存的扩充, 提高了整体的缓存命中率. 此外, 由于缓存区由所有计算节点共享, 同样兼具了计算节点间更新同步加速的功能. 第 3 类架构具备如下 3 个特点: (i) 从效率方面考虑, 系统实现远程共享缓存. 共享缓存内的记录页面相较于存储层内的页面而言具备更低的数据读取延时; (ii) 从吞吐方面考虑, 从存储层读取的数据可以保留在共享缓存中, 避免了相同数据被不同计算节点多次读取时需要在存储层重复读取的问题, 从而提高系统读取吞吐能力; (iii) 从弹性调度方面考虑, 共享缓存所用的远程内存可以基于工作负载自动扩缩容, 系统增加了对于内存资源弹性调度的支持.

(2) 读写路径. 数据写入路径方面, 第 2 类架构的写入过程与前两类架构类似, 即采用持久化重做日志的方法完成事务的数据更新. 不同之处在于, 当更新数据存在于远程缓存时, 需要同时将更新数据写入到远程共享缓存. 更新方式类似于第 2 类架构, 日志存储异步将重做日志更新到页面存储并于页面存储节点内部完成数据回放过程. 数据读取路径方面, 远程缓存作为本地缓存的补充, 当本地缓存失效时可以通过远程缓存获取数据. 当本地与远程缓存均失效时, 才会继续向存储层获取最新数据.

(3) 优势与局限. 相较于前两类架构而言, 计算-缓存-存储分离架构的云原生 OLTP 数据库系统具备如下 3 个优势: (i) 更低的数据读延时: 共享缓存底层由远程内存支持, 数据访问速度高于持久化存储服务, 因此系统具备更低的数据读取延时; (ii) 更高的吞吐能力, 计算节点共享远程缓存数据, 避免重复的存储层数据访问, 进而提高系统的吞吐能力; (iii) 更高的弹性调度能力: 系统增加对于远程内存资源扩缩容的支持, 弹性调度能力加强. 然而, 此类系统的缺点是更大的共享缓存网络压力, 这是因为所有共享缓存均访问同一片远程内存区域, 容易成为系统瓶颈. 此外, 共享缓存需要具备极低的网络延时, 因此必须搭建高速 RDMA (remote direct memory access) 网络作为硬件支持, 因此网络硬件部署的成本更高.

(4) 典型系统. 基于计算-日志-存储分离架构设计的云原生数据库系统包含阿里 PolarDB Serverless[15]. 该系统借助可弹性调度的远程共享缓存服务, 在云原生数据库中加入了多个计算节点共享的远程缓存. 数据更新可以通过共享缓存从主节点传输到备节点, 从而提高备节点的数据读取效率. 在这类架构中一个重要的技术难点在于保证共享缓存与主节点的数据一致性.

1.4 云原生 OLTP 数据库架构总结

表 1 总结了云原生 OLTP 数据库系统不同架构设计的特性, 分别从读写吞吐能力、系统可用性、弹性扩容能力以及部署的成本角度进行分析.

表 1 云原生 OLTP 数据库架构

云原生OLTP系统架构	代表性系统	读性能	写性能	可用性	弹性	成本
计算-存储分离架构	Aurora[8], AlloyDB[21]	中	中	高	高	低
计算-日志-存储分离架构	HyperScale[14], Taurus[23]	高	中	极高	极高	中等
计算-缓存-存储分离架构	PolarDB Serverless[15]	极高	高	极高	极高	高

第 1 类系统结合计算云服务与存储云服务, 相对成本较低. 事务更新的持久化仅依赖于重做日志的持久化, 因此系统仅需要完成重做日志到存储层的写入过程, 就可以通知上层用户事务更新操作成功, 具备良好的写吞吐能力. 存储层中日志数据和页面数据同时存储在相同的存储节点, 并且节点异步处理重做日志到记录页面的回放操作. 当重做日志没有完成到页面的更新回放时, 节点需要进行日志解析过程才能读取到最新数据. 由于日志解析过

程将增加数据读取的延时, 从而一定程度上限制了系统的读取吞吐能力.

第 2 类系统同样结合计算云服务以及存储云服务. 这类架构与第 1 类架构的核心区别在于存储云服务进一步细分成日志存储云以及页面存储云, 日志数据和页面数据独立存储于对应的云存储服务的节点中, 实现了日志和页面数据的分离存储. 日志数据异步地更新到页面存储云, 并基于页面存储所反馈的状态信息异步清理已经完成同步的日志. 日志存储云采用高速云存储服务, 有效降低数据写入延时. 同时, 日志存储云的清理机制保证日志存储节点不需要庞大的存储容量, 能控制相对昂贵的高速存储服务的使用成本. 但是同样受限于日志异步更新的机制, 未完成同步的记录要求计算节点在读取查询过程中需要向多个页面存储节点的请求数据. 且最坏情况下, 当出现已完成回放的页面存储节点的异常时, 读取请求必须等待其余页面存储节点完成重做日志的回放, 这限制了此类系统的读取吞吐能力.

第 3 类系统结合计算云服务、存储云服务以及共享内存服务. 考虑到共享内存要求所有计算节点共同访问相同的内存区域, 因此对于网络带宽有严格要求. 此外, 共享内存同时需要具备低访问延时的特性, 才能利用共享内存实现计算节点间高效数据同步. 因此共享内存服务依赖于高速低延时的 RDMA 网络, 具备较高的网络部署成本. 而高速的共享缓存降低了数据更新在主节点到备节点之间的更新同步延时, 同时缓解多个计算节点重复向存储层请求相同数据的问题, 系统的读取性能得到显著提高. 此外, 计算、缓存、存储服务三者均可独立调度, 因此系统可以支持更细粒度的资源管理.

2 云原生 OLTP 数据库关键技术

总体来说, 云原生 OLTP 数据库的关键技术, 包括数据组织技术、存储层副本一致性技术、主备更新同步技术、故障恢复技术、HTAP 技术 5 个部分, 如图 3 所示. 表 2 列出了具体每一部分的关键技术, 并横向对比同类技术的主要优缺点. 其中, 数据组织技术是云原生数据库存储层管理不同类型数据 (日志数据/页面数据) 的技术 (见第 2.1 节). 存储层副本一致性技术是云原生数据库存储层维护多个数据副本一致性的技术 (见第 2.2 节). 主备更新同步技术是云原生数据库计算层主计算节点将事务更新同步到备节点的技术 (见第 2.3 节). 故障恢复技术是云原生数据库应对不同类型异常而设计的多层次故障恢复的技术 (见第 2.4 节). HTAP 技术是云原生数据库在原有 OLTP 系统之上添加对于 HTAP 负载优化的技术 (见第 2.5 节).

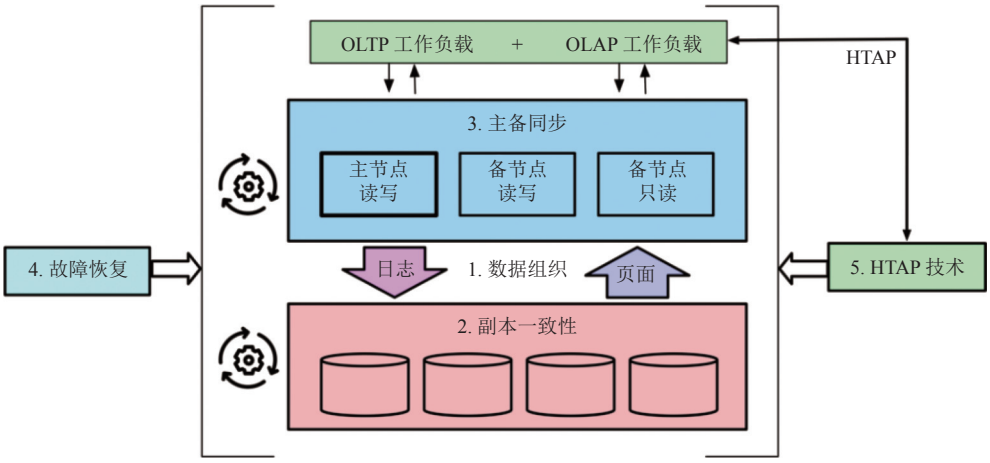


图 3 云原生 OLTP 数据库关键技术分类

在云原生 OLTP 数据库中, 数据组织技术是指存储层组织重做日志和记录页面两类不同数据的技术. 这类技术根据架构设计的差异适应性地组织数据, 以求同时兼顾存储层数据更新和读取的效率. 已有的关键技术主要分为两大类: (1) 日志-页面耦合的存储策略; 以及 (2) 日志-页面分离的存储策略. 第 1 类技术一般应用于日志-页面耦合架构, 这类系统中存储层使用统一的存储服务, 单一存储节点内同时存储日志和页面数据. 这类架构下不存在跨

存储服务的数据传输, 其关键技术在于以日志为中心的数据组织模式, 存储节点支持直接通过重做日志读取记录, 但是需要面临额外的日志解析开销. 第 2 类技术一般应用于日志-页面分离架构, 这类系统中存储层采用不同的存储服务管理两类数据, 单一存储节点仅保留其中之一. 这类架构下事务更新的同步需要进行跨存储服务的数据传输, 其关键技术在于降低更新同步的延时以及数据传输量, 以达到降低存储层的读取延时. 由于该结构下读取过程仅通过页面存储读取数据, 因此能够避免前者的日志解析的时间开销.

表 2　云原生 OLTP 数据库关键技术总览与优缺点比较

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	日志-页面耦合存储策略	Aurora[8]	写入效率高	额外的日志处理
	日志-页面分离存储策略	HyperScale[14] Taurus[23]	弹性调度能力强	阻塞的更新同步
存储层 副本一致性技术	基于Quorum算法的协议	Aurora[8,24]	并发能力强	额外的同步过程
	基于Paxos类算法的协议	PolarFS[25]	可线性化	实现复杂
计算层 主备同步技术	基于持久化存储同步	Aurora[8]	可靠性高	同步延时高
	基于本地缓存同步	HyperScale[14]	同步延时低	不保证可靠性
	基于远程共享缓存同步	PolarDB Serverless[15]	读取延时低	缓存一致性问题
故障恢复技术	无重做的计算节点恢复	Aurora[8]	计算节点省略 Redo过程	存储节点 承担Redo压力
	两层ARIES容错协议	LegoBase[26]	计算节点通过缓存层加速恢复	仅为优化手段 不保证可靠性
HTAP技术	计算层动态行存转列存	AlloyDB[27]	节约存储空间	列存选择难度大
	存储层行列混存数据副本	TiDB[28]	隔离性好	数据新鲜度低
	内存型行存加持久化列存	SinglestoreDB[29]	读写延时低	内存开销大

在云原生 OLTP 数据库中, 存储一致性技术是指存储层用于维护多个数据副本一致性的相关技术. 这类技术一般基于传统分布式系统的相关协议, 在原有策略基础添加对于云环境的特定优化. 本文按照原始算法将这类技术划分为两类: (1) 基于 Quorum 算法的协议; (2) 基于 Paxos 类算法的协议. 第 1 类技术基于原始 Quorum 算法设计. Quorum 算法[30]具有简单的流程以及优秀的并发能力, 但是不能满足可线性化特征. 部分存储节点缺失更新时仍会继续运行, 这造成计算层必须向多个存储节点读取数据. 为了解决这一问题, 这类技术需要结合 Gossip 协议等方式来提供可线性化特征. 第 2 类技术基于 Paxos 类算法设计, 包括 Paxos[31]以及 Raft[32]算法及其变种[33,34]. 这类算法具备可线性化的特征, 但其严格的线性化流程限制了原始算法的并发能力. 因此这类技术需要结合云环境高并发特性加强原始算法的并发能力.

在云原生 OLTP 数据库中, 更新同步技术是指计算层主节点将事务的更新从主节点同步到只读备节点的技术. 这类技术要求系统在保证数据一致性的前提下, 降低备节点的同步延时. 这类技术可以按照同步路径可以划分为 3 类: (1) 基于持久化存储同步; (2) 基于计算节点本地缓存同步; (3) 基于远程共享缓存同步. 第 1 类技术是最基本的同步方式, 在现有云数据库一写多读的架构下能够保证数据一致性. 但是主要的限制在于通过持久化存储读取数据需要经过网络 I/O, 同步延时相对延时较高. 第 2 类技术则基于本地缓存进行更新同步, 主节点直接向备节点传输重做日志来更新本地数据. 这类技术的难点在于限制网络传输量以避免网络瓶颈. 第 3 类技术基于所有计算节点共享的远程缓存进行同步, 主节点将更新写入到共享存储后备节点可以直接从中读取. 这类技术的难点在于保证主节点与远程缓存的一致性, 避免备节点读取到已经变更的数据.

在云原生 OLTP 数据库中, 故障恢复技术是指系统为应对多种异常情况而设计的多层次故障恢复策略. 分离的架构设计让数据库系统不同层级间数据传输的网络成本高于传统的耦合结构, 因此在故障恢复过程中, 通过在层级内进行节点恢复来避免跨服务的网络传输. 此外, 分离的架构也同样提供了容错隔离性, 促成云原生环境下特有的故障恢复技术. 这一类技术包括如下两类: (1) 计算层无重做的节点恢复; (2) 计算缓存层的双层容错协议. 第

1 类技术基于计算-存储分离的架构, 不同于传统数据库系统, 云数据库中数据持久性完全由存储层保证. 因此计算层不需要经过重做阶段来恢复故障时的缓存状态, 而是将这一阶段下推到存储层异步完成, 降低计算节点故障恢复延时. 第 2 类技术基于计算-缓存分离的架构, 利用缓存层作为计算层的额外检查点. 利用二者容错独立特性, 将共享缓存作为计算节点的可靠存储来加速故障恢复, 同时持久化存储作为计算层和缓存层的可靠存储, 保证二者同时异常时的故障恢复能力. 这种两层容错的方式可以降低计算节点的平均故障恢复时间.

在云原生 OLTP 数据库中, HTAP 技术是指在原有 OLTP 系统基础上添加对于 OLAP 工作负载的优化. 本文重点关注于分离架构基础上的 HTAP 相关技术, 可以划分为 3 类: (1) 计算层动态行存转列存; (2) 存储层行列混存数据副本; (3) 内存型行存加持久化列存. 第 1 类技术关注于计算层, 数据从存储层导入到缓存后可以自动从行存转化为列存格式, 从而加速 OLAP 负载的处理速度. 技术难点在于优化器需要合理选择转化的数据列以及查询所使用的数据格式. 第 2 类技术关注于存储层, 系统同时维护行存和列存数据副本, 行存数据异步更新到列存数据, 利用列存数据优化分析型查询. 技术难点在于提高列存数据的新鲜度. 第 3 类技术发展自内存型数据库 MemSQL[35]. 系统在内存中按照行存格式进行事务处理, 持久化存储中则按照列存格式进行组织数据. 这种模式下, 内存中的行存格式数据能高效处理事务型查询, 而持久化存储中的列存格式数据能在分析型查询中加速大规模数据扫描、聚合、分组等运算. 这类技术的主要限制在于内存中的行存格式数据到持久化存储的列存格式数据的转化开销高. 因此系统在实际运行中要求热数据尽可能常驻于内存中来内存中数据到磁盘的转化频率, 而这将产生较高的内存开销.

2.1 云原生 OLTP 数据库的数据组织技术

云原生 OLTP 数据库的数据组织的关键技术主要分为两大类: (1) 日志-页面耦合的存储策略; (2) 日志-页面分离的存储策略. 接下来分别介绍这两类技术.

2.1.1 日志-页面耦合的存储策略

在 OLTP 数据库中, 日志数据记录事务产生的数据变化过程, 而页面记录特定时刻数据库的实际存储状态. 日志-页面耦合的存储策略使用统一的存储服务管理这两类数据.

Aurora[8]数据库存储层采用了计算-存储耦合的存储策略, 核心的特性可以概括为“日志即数据”. 这类策略的数据读取和写入均以日志为中心. 不同于传统数据库直接通过页面读取记录的方式, 系统将日志数据组织成链表形式, 形成日志链. 日志链保存记录创建后的所有变化过程, 可以从中解析出任意数据库版本下的记录数据. 数据写入过程中, 存储节点仅需要获取并持久化计算层传输的重做日志就可以通知计算层更新成功. 存储节点内部异步处理重做日志的回放. 这种方式避免了传统数据库脏页刷盘而带来的页面数据传输, 有效减少计算层和存储层之间的网络带宽开销. 同时更新的回放过程不会阻塞进程, 降低数据写入延时. 数据读取过程中, 存储节点需要根据事务的版本号在日志链上进行下界查找, 确定最邻近的重做日志. 由于查找时间开销与日志链的长度相关, 因此节点需要压缩日志链长度以提高查询效率. 页面在这类策略中被定义为日志链的压缩: 存储节点将舍弃已经提交事务更新对应的重做日志, 保留特定版本的数据快照形成记录页面. 通过这种方式, 数据被组织成记录页面加部分日志链的形式: 运行中事务产生的数据更新均保留在日志链中, 因此运行中事务的读取不受影响; 同时页面保存回放完成的数据记录, 保证良好的数据读取速度.

图 4 展示了这类数据组织模式下的数据读写实例. 当前存储节点已经完成日志编号 L_{9009} 的持久化过程, 此时记录 X 存储于页面 k (版本 1000), 同时相关日志编号 L_{1001} 到 L_{9009} 的日志没有回放到记录页面. 对于数据写入过程, 节点接收新的写入请求并将接收到编号 L_{9010} 的重做日志进行持久化, 完成日志持久化任务后通知上层写入成功. 存储节点后台异步完成日志链的压缩任务. 当检测到日志 L_{1001} 到 L_{20x0} 的长度达到压缩阈值且对应事务已经提交, 则将合并这部分日志并回放到页面 k , 此时页面 k 的数据将更新到 20x0 版本的状态. 对于数据读取请求, 当节点接收到对于 X 的读取请求时, 会检测读取请求来源事务的版本编号 T . 当 T 不超过已经完成持久化的页面版本 1000 时, 节点将直接向上层返回页面 k 中存储的 X 记录值 32001. 而如果 T 超过了已经完成持久化的页面版本 1000, 则需要通过日志链推断版本 T 时刻记录 X 的状态 (推断过程中需要忽略未提交事务的影响), 并向上层返回版本 T 的 X 值 X_T.

这类技术的主要挑战在于日志链的组织. 数据库基于日志链的长度, 后台将重做日志合并到记录页面. 由于日

志链的合并过程在效果上等同于批量日志回放, 符合云原生数据库数据更新基于存储层日志回放的特性. 总体上, 这类技术的优势在于: (1) 避免脏页回放, 降低计算层网络带宽压力; (2) 避免数据写入锁定页面, 提高事务的写吞吐能力. 其主要的限制在于: 读取过程中引入了额外的日志解析过程, 增加数据读取延时.

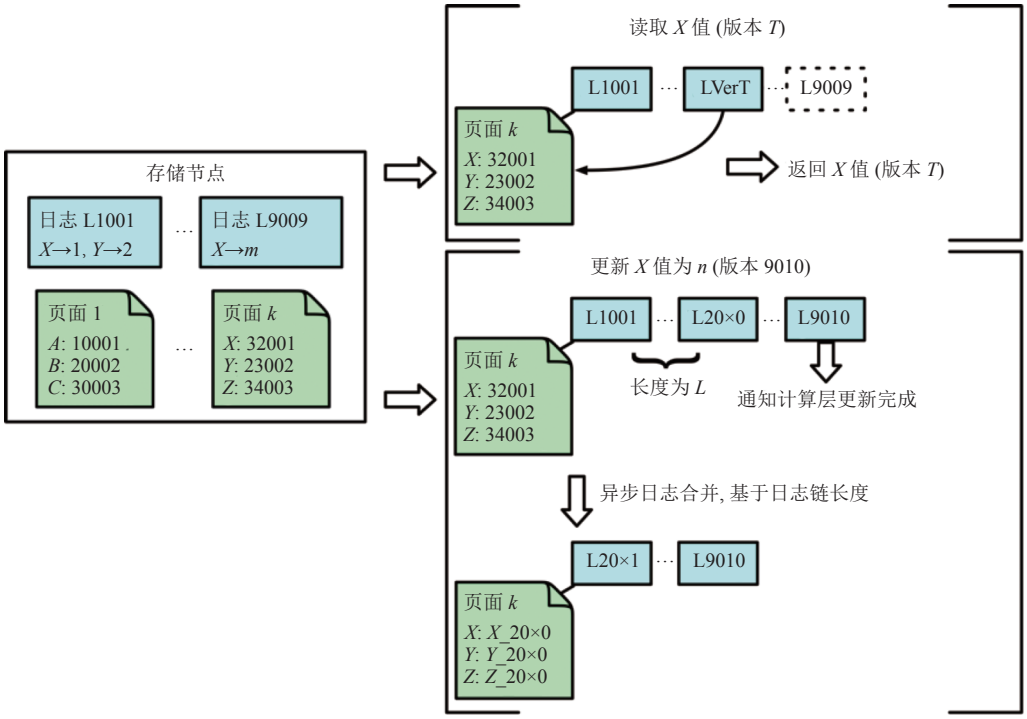


图 4 日志-页面耦合存储策略中的数据读写路径

2.1.2 日志-页面分离的存储策略

日志-页面分离的存储策略分别管理重做日志和记录页面两类数据, 一般会区分二者的特性分别使用不同的存储服务. 日志数据使用高速云存储 (如 SSD 云存储服务) 来提高写吞吐能力的同时, 页面数据使用一般云存储 (如 HDD 云存储服务) 来控制存储成本. 各类存储服务独立调度, 更好地发挥底层云存储服务的弹性调度能力.

HyperScale[14]、Taurus[23]数据库存储层采用了计算-存储分离的存储策略, 核心特性可以概括为“可用性与持久性分离”. 数据持久性由日志存储保证, 可用性则由页面存储保证. 计算层更新数据将会向存储层中的日志存储传输重做日志, 而读取数据则从存储层的页面存储中解析日志记录. 这类方法同样避免了脏页刷盘产生的写放大问题. 同时日志存储利用高速云存储服务提高日志持久化效率, 系统具备更低的数据写延时. 同时页面数据采用一般云存储服务, 降低数据库系统整体使用成本. 这种数据组织模式下, 数据读写路径如图 5 所示. 数据写入过程中, 计算层仅需要向日志存储传输重做日志和元信息, 高速的日志存储服务能降低写入日志的时间开销. 数据读取过程中, 计算层需要从页面存储读取记录. 读取和写入的存储区域不同, 因此日志存储需要不断向页面存储传输重做日志以更新页面数据. 由于页面存储不需要保证数据持久性, 故接收重做日志数据时, 页面存储节点不要求全部成功接收. 计算节点采用宽松策略接收这些日志, 允许数据部分丢失, 以避免部分节点低效率或异常而导致系统整体阻塞. 页面存储节点内部基于 Gossip 协议及时通过其余节点补全自身缺失的重做日志. 最后, 当所有页面存储节点均完成日志回放后, 该日志的持久性可以由页面存储保证, 从而对应日志可以从高速的日志存储云中去除, 减少日志存储的空间使用量来降低高速存储服务的使用成本.

这类技术的主要挑战在于存储层内重做日志的回放. 日志存储到页面存储依赖于网络 I/O, 同时需要考虑到页面存储部分节点的临时故障. 因此日志传输采用宽松的传输协议并结合页面节点间 Gossip 协议来避免日志回放

过程阻塞系统运行. 总体上, 这类技术的优势在于: (1) 利用高速云存储负责日志的写入, 降低整体数据写入延时; (2) 存储层内无阻塞的日志回放过程, 加速数据更新的同步. 其主要的限制在于: 由于部分页面节点可能出现故障导致临时的数据缺失, 读取操作可能需要由多个存储节点完成, 一定程度上限制了数据读取的吞吐量.

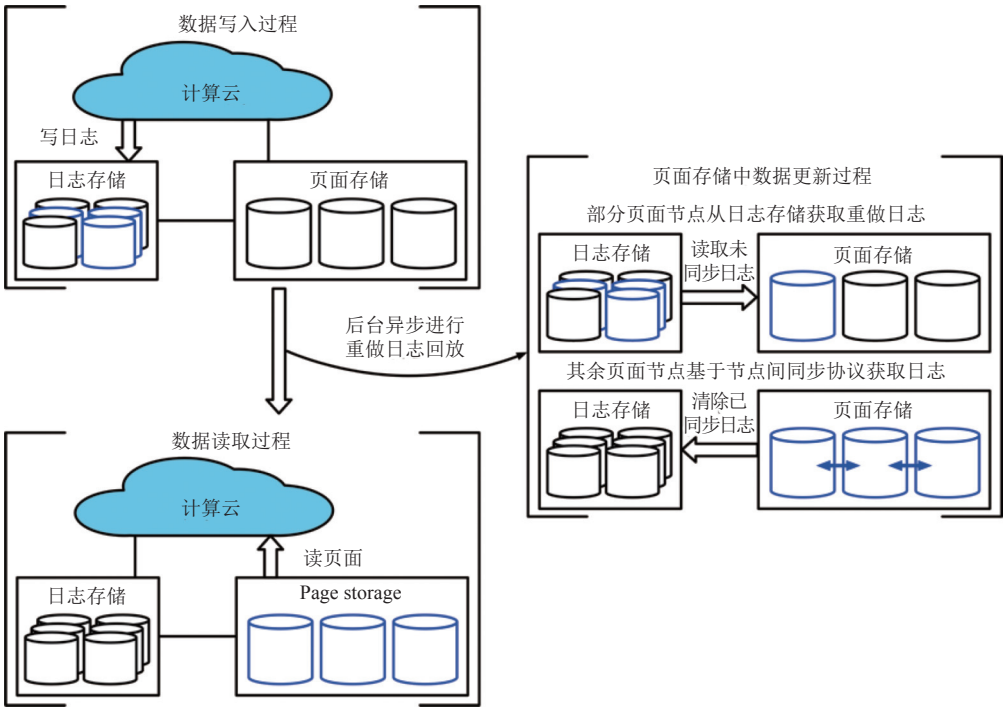


图 5　日志-页面分离存储策略中的数据读写路径

2.2　云原生 OLTP 数据库的副本一致性技术

云原生 OLTP 数据库的数据一致性的关键技术主要分为两大类: (1) 基于 Quorum 算法的协议; (2) 基于 Paxos 类算法的协议. 接下来将分别介绍这两类技术.

2.2.1　基于 Quorum 算法的一致性协议

Quorum 算法是分布式系统用于保证数据一致性基本算法. 通过读取投票和写入投票的机制限制分布式系统运行过程中的读写串行化以及写写串行化, 有效解决了读写冲突以及写写冲突的问题. 而云数据库系统底层为了保证底层数据的高可用性和强持久性, 存储层需要采用分布式多副本存储的策略, 而 Quorum 算法可以保证在底层多副本环境中的读写一致性.

Aurora[8,24]在存储层采用了 Quorum 算法并针对于云环境进行了优化, 主要体现在可用性和数据恢复两方面. 一方面, 从可用性考虑, 云服务管理中对于实际物理服务器划分了容错域的隔离级别. 通过地域隔离的手段, 不同容错域之间发生故障的可能性完全独立, 以便于应对断电、地震、火山爆发等极端灾难时保证数据持久性. 但应用于传统系统的 3 副本 Quorum 算法的容错能力受限, 在应对容错域级别的长期故障与单节点短期异常时就会导致系统处于不可用状态. Aurora[8]数据库则添加了容错域内的副本冗余, 通过 6 副本的 Quorum 算法强化容错能力, 达到“容错域+1”的容错能力. 另一方面, 从数据恢复方面考虑, Aurora[24]提出了 Quorum 关系集合的机制: 同属相同 Quorum 分组的存储副本组成一个 Quorum 关系集合. 存储副本出现异常时会拓展新的副本并添加新的关系集合. 系统利用关系集合同时管理多个 Quorum 分组, 其中任意分组达成基本的读写投票即可在对应集合的存储副本执行对应操作. 这种机制保证存储节点故障时可以无阻塞的切换到新的存储副本, 降低存储层的故障恢复延时.

这一类算法的主要问题在于协议不满足可线性化特征. 在部分节点缺失部分更新的情况下, 最小投票仍可以

被满足, 从而维持系统继续运行. 但在这将造成系统内部分副本都处于不一致性状态, 要求计算层需要严格按照读取投票的要求向多个副本请求数据, 对于系统的读取吞吐能力产生负面影响. 因此, 存储层内部需要需要额外添加存储副本间的 Gossip 协议来填补这部分可能的更新缺失, 加强副本间的一致性.

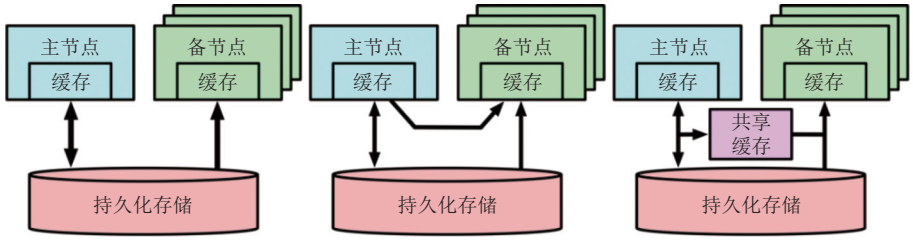
2.2.2 基于 Paxos 类算法的一致性协议

Paxos 类算法包含了 Paxos、Raft 以及相关变种, 属于一类在不可靠环境中保证数据一致性的算法. 原始算法严格按照阶段进展并依次提交, 具备可线性化特征. 云数据库底层存储层同样可以采用 Paxos 类算法来保证数据副本间的一致性. 这一类算法的主要问题在于其原始算法严格的线性化流程限制了系统的并发能力. 因此在真实云数据库系统的实现上, 需要结合实际场景放宽一定限制条件来提高算法并发能力.

PolarFS[25]是 PolarDB 存储层引擎, 其利用底层高并发 I/O 的物理服务器环境优化传统 Raft 算法. 特别地, 其提出了 ParallelRaft 算法来解决传统 Raft 算法并发能力不足的问题. 该算法在传统 Raft 算法的基础上支持多个阶段并发提交, 允许存储节点非顺序通知并接收更新日志, 因此支持存储节点内接收的更新日志序列中存在一定空隙. 这种机制保证部分日志传输的阻塞不会影响系统整体运行, 保证了良好的并发处理能力. 同时, 为了加速落后节点的同步进程, 系统通过批量化的日志接收策略来降低同步延时.

2.3 云原生 OLTP 数据库的主备同步技术

云原生 OLTP 数据库的数据一致性的关键技术主要分为 3 大类: (1) 基于持久化存储的更新同步; (2) 基于本地缓存的更新同步; (3) 基于远程共享缓存的更新同步. 图 6 展示了不同类型更新方式的横向对比, 接下来将分别介绍这 3 类不同的同步方式.



(a) 基于持久化存储的更新同步 (b) 基于本地缓存的更新同步 (c) 基于远程共享缓存的更新同步

图 6 主备节点更新同步的 3 种方式

2.3.1 基于持久化存储的更新同步

云原生 OLTP 数据库计算层中主节点到备节点的第 1 类同步方式依赖于持久化存储, 如图 6(a) 所示. 由于系统仅有主节点支持读写查询, 其余备节点均仅支持只读查询. 单一的更新源头保证了持久化存储不存在写写冲突的可能性, 只需要忽略本地缓存, 备节点所有查询请求均等待主节点写入成功后, 从持久化存储中读取数据. 这种方式必然能够读取到主节点的全部数据更新. 因此, 备节点通过持久化存储获取数据更新是计算层最基本且最稳定的更新同步方式, 在存储层数据持久性和可用性不受影响的情况下保证更新同步机制的可靠性.

虽然基于持久化存储实现更新同步的方式具备最好的可用性, 但是基于这一途径的更新同步执行效率较低. 备节点在读取持久化存储需要面对计算存储层之间网络传输的 I/O 代价, 存储层持久化存储读取的 I/O 代价, 以及等待主节点数据写入确认的延时. 仅依赖于这种同步方式, 从数据传输量以及数据访问延时的角度考虑均存在瓶颈, 因此该方案一般作为最坏情况下的数据一致性保障.

2.3.2 基于本地缓存的更新同步

云原生 OLTP 数据库计算层中主节点到备节点的第 2 类同步方式依赖于计算节点本地缓存的数据状态同步, 如图 6(b) 所示. 基于持久化存储的更新同步过程忽略了缓存对于查询的优化, 当备节点缓存数据与主节点一致时, 可以直接利用本地缓存而避免从存储层读取数据, 相较于基于持久化存储的方式而言可以显著降低数据读取延时以及计算层与存储层之间的数据传输量. 这种同步方式的主要问题在于保证主节点和备节点缓存内数据状态的一致性, 必须保证主节点完成更新后备节点不能读取到自身缓存的过期数据. 这一问题可以通过主节点向备节点直

接传输重做日志来解决. 备节点直接回放接收的重做日志即可完成本地缓存内的数据更新.

这类同步方式具备 3 个关键特性: (1) 低数据访问延时, 通过本地缓存读取数据的延时远低于通过存储层获取数据; (2) 宽松的传输协议, 主节点向备节点传输重做日志采用宽松的传输协议 (不需要备节点确认过程、不需要备节点持久化), 因此不增加主节点数据写延时; 由持久化存储保证传输失败情况下的更新同步; (3) 同步过程的数据传输任务下放到存储层, 为了避免主节点成为系统网络传输瓶颈, 主节点先将重做日志传输到存储层后, 由存储层分布式的存储节点将日志传输到备节点.

2.3.3 基于远程共享缓存的更新同步

云原生 OLTP 数据库计算层中主节点到备节点的第 3 类同步方式依赖于计算节点共享的远程缓存, 如图 6(c) 所示. 这一类同步方式需要依赖于计算节点的共享缓存, 在实际系统中需要在硬件层面搭建 RDMA 网络实现跨物理服务器的远程内存访问功能, 相对成本较高. 对比于基于本地缓存的同步机制来看, 在这类方式下, 备节点不需要接收主节点传输的重做日志并更新自身缓存; 仅需要将更新后的记录页面直接传输到共享缓存区域即可, 备节点可以直接通过共享缓存获取更新记录. 因此这种同步方式具备更低的更新同步延时.

这类同步方式具备 3 个关键特性: (1) 提高内存资源使用效率, 真实云数据库系统中, 由于资源使用量的不均衡性, 经常会面临部分服务器内存资源闲置的额情况. 通过远程共享缓存可以有效利用这一部分闲置资源. (2) 低更新同步延时, 同步过程的数据传输任务下放到共享缓存层, 相较于存储层具备更快的传输速度; (3) 共享缓存不会向存储层写回数据, 主节点直接向存储层写入重做日志的来保证数据持久性, 共享缓存仅用于提高数据查询和同步的效率. 以此来避免脏页写回所造成写放大问题, 减轻共享缓存的网络压力.

2.4 云原生 OLTP 数据库的故障恢复技术

云原生 OLTP 数据库的故障恢复的关键技术主要分为两大类: (1) 无重做的计算节点恢复; (2) 计算-缓存双层容错协议. 接下来将分别介绍这两类技术.

2.4.1 无重做过程的计算节点恢复

传统数据库采用 ARIES (algorithms for recovery and isolation exploiting semantics) 算法进行故障恢复[36], 整体可以分为分析、重做、撤销 3 个阶段. 其中, 重做过程是根据未写回持久化存储的脏页表, 重新执行产生这些脏页的事务. 由于在传统数据库系统中, 持久化存储的页面更新依赖于脏页刷盘. 因此事务更新的持久性与本地缓存状态相关, 必须恢复这些没有完成持久化的脏页数据才能避免事务更新丢失. 同时传统数据库系统中持久化存储不具备计算能力, 不能够独立解析日志并更新数据, 因此计算层的重做过程在传统数据库系统不可省略.

图 7 对比了传统数据库系统与云原生数据库数据更新方式的差异. 云原生数据库事务更新向存储层传输重做日志, 任何情况下均不会传输页面数据. 云原生系统的存储节点具备一定的计算能力, 可以处理日志数据并异步回放到记录页面中. 页面的持久性与本地缓存信息完全独立, 因此计算节点不需要恢复本地缓存的脏页状态, 可以跳过传统 ARIES 算法的重做过程. 而这一部分将下推到存储层由存储节点内部异步处理.

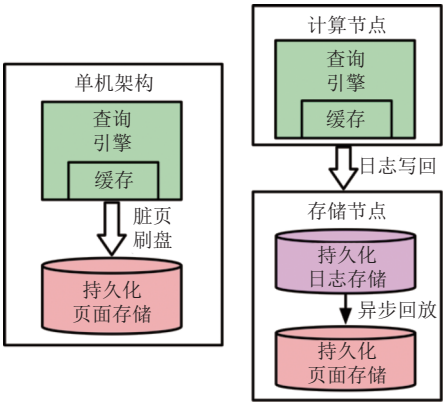


图 7 传统数据库与云原生数据库数据更新差异

这一技术的主要限制在于: 由于恢复过程不需要获取本地缓存状态, 故计算节点恢复后面临冷缓存问题. 但是考虑到云原生系统一般采用多层缓存的设计, 且多层缓存之间容错独立. 因此, 主节点外的缓存 (包括共享缓存、存储层缓存) 不会随计算节点故障而同时故障. 因此这些缓存可以缓解计算节点恢复后的冷缓存问题.

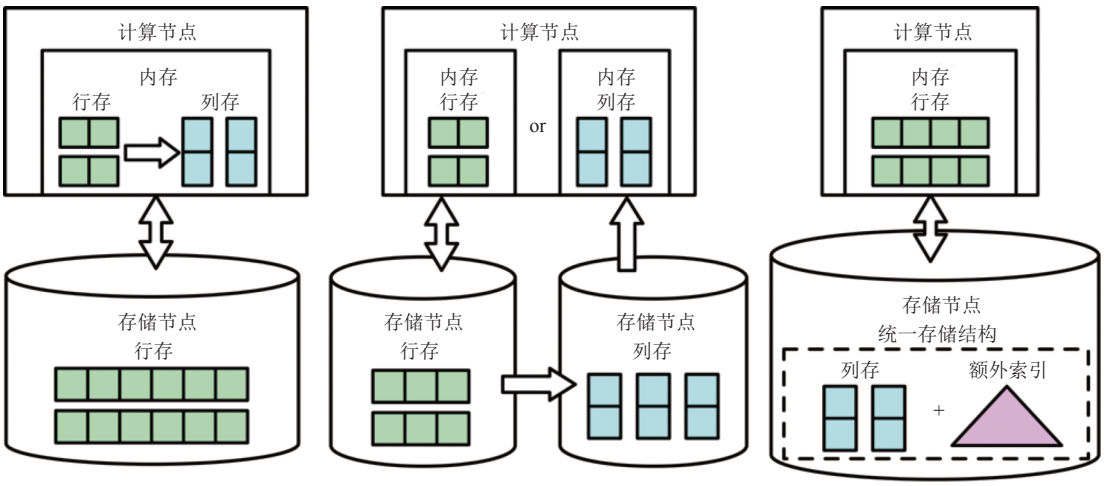
2.4.2 计算-缓存双层容错协议

对于计算-缓存-存储分离架构的云原生数据库系统, 不同于操作系统层级的远程内存服务[37,38], 数据库层级的远程内存服务可以基于数据库工作负载的特性定制化内存管理, 因此避免了操作系统层级内存服务的内核调用开销. 由于云上的计算服务和远程内存服务一般由不同的实际物理服务器提供, 因此同时发生故障的概率较小. 基于上述条件, LegoBase 提出了两层容错协议: 第 1 层将远程内存作为计算节点的可靠存储; 第 2 层将持久化存储作为计算节点和远程内存的可靠存储. 借助远程内存远低于持久化的数据访问延时, 在仅计算节点发生故障时, 计算节点可以通过第 1 层容错协议, 直接从远程内存中快速读取检查点信息, 因此大幅降低了故障恢复时间. 而即使面对计算节点和远程内存同时异常的情况, 通过持久存储的检查点信息同样可以快速恢复计算节点状态.

两层容错协议具备如下两个特性: (1) 降低计算节点平均故障恢复延时: 由于计算节点和共享缓存的物理隔离性, 得益于共享缓存远低于持久化存储的数据访问延时, 可以实现短时间的轻量级故障恢复; (2) 保证最坏情况下数据持久性: 持久化存储中的预写日志和检查点作为数据持久性的基本保证. 即使面对计算节点和共享缓存同时出现异常的情况, 依旧能够通过第 2 层容错机制保证系统基本的数据持久性不受影响.

2.5 云原生 OLTP 数据库的 HTAP 技术

云原生 OLTP 数据库的 HTAP 关键技术主要分为 3 大类: (1) 计算层动态行存转列存; (2) 存储层行列混存数据副本; (3) 内存型行存加持久化列存. 图 8 展示了不同技术的示意图, 接下来将分别介绍这 3 类 HTAP 技术.



(a) 计算层动态行存转列存　(b) 存储层行列混存数据副本　(c) 内存行存加持久化列存

图 8 云原生数据库计算存储分离架构下的 HTAP 技术

2.5.1 计算层动态行存转列存

云原生数据库中第 1 类 HTAP 技术的实现方式基于计算层动态数据格式转化, 如图 8(a) 所示. 存储层采用按照行存格式保存记录数据. 当计算层从存储层读取行存格式的数据后, 会基于规则或机器学习方法将部分数据自动转化为列存储格式, 以加速数据分析中各类聚集查询算子的执行. 同时为了避免转化后 OLTP 工作负载受到影响, 系统支持同时在内存内同时保留行存和列存两种数据格式并基于代价模型为执行计划的查询算子选择更合适的数据格式. 这一类技术的主要挑战在于计算层的优化器模块的设计: 优化器需要基于工作负载决定哪些行存数据需要转化为列存, 由于存储格式转化会占用计算和内存资源, 因此选择高效用的数据非常关键; 同时优化器需要基于不同类型的工作负载特征决定基于行存数据或列存数据执行特定算子 (如点查询可基于行存, 聚集查询可基

于列存), 然而当查询的执行计划空间很大时, 选取一个最优的执行计划非常困难.

2.5.2 存储层行列混存数据副本

云原生数据库中第 2 类 HTAP 技术的实现方式基于存储层同时存储异构的数据副本, 如图 8(b) 所示. 不同于第 1 类方法, 存储层同时存储行存格式和列存格式的数据副本. 上层计算节点基于不同任务类型访问相应数据, 负责处理事务型查询的节点仅访问行存格式副本, 而负责分析型查询的节点仅访问列存格式副本. 因此 OLTP 负载的处理过程和 OLAP 负载的处理过程完全隔离, 互不干扰. OLTP 数据更新会实时更新缓存内页面数据并同步到行存格式的数据副本. 存储层内部处理数据更新在副本间的同步, 包括行存格式副本之间以及行存格式到列存格式之间的数据同步. 具体来说, 行存储副本之间维持原有的一致性协议, 并将列存格式的副本作为学习者, 批量化异步地接收行存副本的数据更新. 这一类技术的主要挑战在于 OLAP 负载的数据新鲜度问题: 由于行存副本与列存副本之间的同步延时, 导致系统在处理分析型查询时面临数据新鲜度低的问题.

2.5.3 内存型行存加持久化列存

云原生数据库中第 3 类 HTAP 技术的实现方式为内存中行存储加持久化列存储, 如图 8(c) 所示. 这一类技术由内存型数据库发展而来. 系统在内存中按照行存格式管理数据, 并组织成跳表 (skip list) 的数据格式加速内存中的数据读取和写入效率. 系统通过向持久化存储中追加写入更新日志的方式以持久化数据. 而针对于 OLAP 负载的大规模扫描、聚合、分组等分析型查询, 数据库系统在存储层内直接维护列存格式的记录数据, 并按照数据段 (segments) 进行组织, 以处理分析型查询. 此外, 存储层内在列存格式的基础上添加二级索引, 用来加速点查询、范围查询以及连接运算. 内存中的行存格式数据保证了事务型查询的效率, 而外存的列存格式数据保证了分析型查询的效率, 因此兼顾 HTAP 负载处理性能. 这一类的主要挑战在于内存开销: 由于内存和外存使用的数据格式不同, 因此在写入和读取过程中均需要经过额外的格式转化过程, 占用计算资源. 因此系统必须尽可能减少二者之间的数据交换. 在实际运行场景中, 系统必须占用足够的内存空间保证大部分热数据被缓存才能维持系统的高效运行.

3 云原生 OLAP 数据库架构

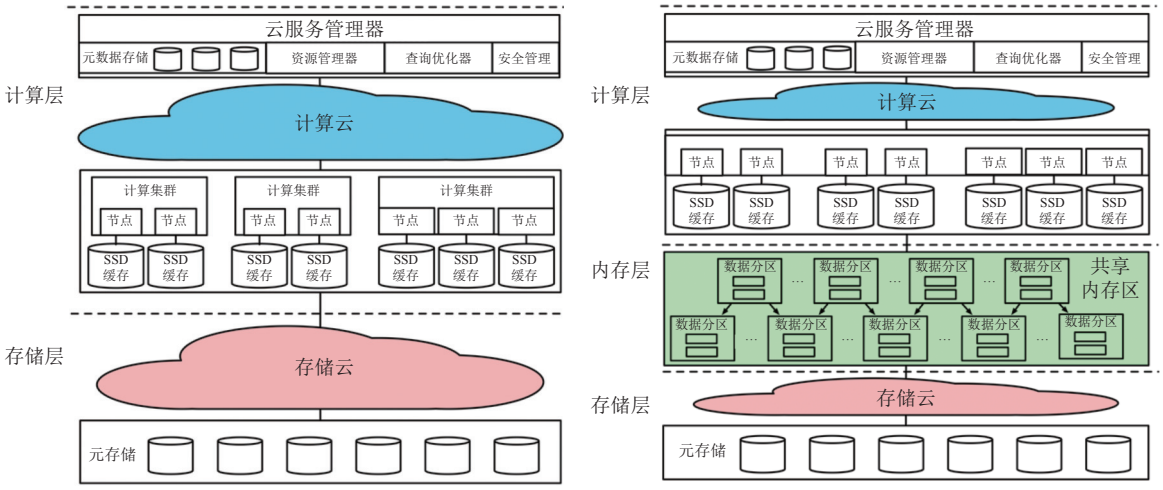
云原生 OLAP 数据库系统[4]主要面向大规模数据分析型场景, 这类场景以分析型负载为主, 同时也支持以事务型日志的方式批量更新数据. 与传统的 MPP (massively parallel processor) 型数据仓库架构[39]相比, 云原生 OLAP 数据库系统通过采用存储-分离架构不仅提高了系统的弹性, 还基于云平台的异构计算节点和统一云存储实现了系统的高可用. 举例来说, 传统的 MPP 型数据仓库采用相同配置的节点并发地处理查询, 但当有节点故障或者在新的节点加入后, 系统会对数据重新划分, 且对所有节点执行数据洗牌 (shuffle) 操作, 这造成了很大的系统开销. 而云原生 OLAP 数据库基于云服务对系统的节点采用不同的计算资源配置 (如 CPU 核数与内存容量) 以适应不同的负载类型 (I/O 密集型任务或计算密集型任务). 特别地, 其还基于云存储避免了数据重划分, 即工作节点的本地存储只用于缓存计算, 如缓存没有命中则从云存储中加载数据; 云存储服务则通过跨区域的多副本部署保证了存储的高可用, 并实现了对用户的透明管理.

目前的云原生 OLAP 数据库架构包含两大类: (1) 计算-存储分离的两层架构; 与 (2) 计算-内存-存储分离的 3 层架构. 下面分别介绍它们的特性, 包含主要设计动机, 查询执行流程, 优势与局限, 以及典型系统.

3.1 云原生 OLAP 计算-存储分离的两层架构

如图 9(a) 所示, 第 1 类云原生 OLAP 数据库系统采用了计算-存储分离的两层架构, 计算层与存储层之间采用高速网络进行连接, 采用这类架构的主要代表系统为 Amazon Redshift[40]和 Snowflake[9]. 具体来说, 这类架构的计算层包含了云服务管理器和计算云. 前者是整个系统的“大脑”, 包含了元数据存储、资源管理器、查询优化器、和安全管理等主要模块; 后者是计算层的“肌肉”, 主要负责利用多个计算节点并行地执行具体的查询任务. 这些计算节点不区分主备节点, 而是由上层云服务管理层统一调度. 它们具有一定的存储能力, 每个计算节点有独立的 SSD 存储用于缓存热数据, 并按照数据分片分配任务, 单个计算节点在分析过程中只处理自身的数据分片并借助

本地存储资源保存中间数据, 仅有在合并过程中会和其他计算节点通信并进行数据合并; 存储以数据分片为中心, 数据传递以及底层存储均不涉及日志数据, 而是以数据分片作为数据传输和底层存储的内容. 特别地, 存储层完全与计算层解耦, 内部的持久化、高可用、与容灾备份管理对用户透明, 并以云存储的形式由大型云服务提供商提供服务.



(a) 云原生 OLAP 的计算-存储分离架构　　(b) 云原生 OLAP 的计算-内存-存储分离架构

图 9　云原生 OLAP 数据库架构

(1) 主要设计动机. 从设计的目的来看, 此架构具备如下 3 个特点: (i) 高度弹性服务. 即通过计算层与存储层的完全解耦, 实现了计算节点与存储节点的独立扩缩容机制, 使得用户可根据自身的计算与存储需求弹性地使用数据库服务, 从而极大地节约成本; (ii) 高可用服务. 系统可以基于云提供商的云存储服务实现高可用. 一方面, 云存储实现了跨区域的多数据副本部署, 当某区域的副本节点出现故障, 服务提供商还可以利用其他节点的数据副本提供服务. 另一方面, 计算节点的本地存储仅用于缓存作用, 当有计算节点发生故障或者有新的节点发生故障时, 系统无需进行数据重分配操作, 只需对相应节点进行新的缓存调用即可. (iii) 异构计算服务. 系统可在计算层启动多个计算集群, 每个集群可实例化多个不同配置的计算节点, 用于执行用户的不同的计算任务, 且计算集群之间可做到隔离执行, 以保证对多个租户同时提供高质量服务.

(2) 查询执行流程. 基于这类云架构的查询处理分为以下 3 个步骤: (i) 查询优化. 云服务管理器接受查询请求后, 查询优化器结合元数据存储的统计信息进行查询重写与优化, 并生成并行查询计划. (ii) 基于缓存的查询处理. 查询子计划被发送到计算集群的各个计算节点, 如果缓存命中, 则直接将查询在本地处理后返回结果. 反之, 则向存储云请求数据. (iii) 基于下推的查询处理. 存储云收到请求后, 则访问对应的列式文件, 并通过下推计算到存储文件, 最后返回过滤后的结果. (iv) 结果合并与返回. 计算集群合并各个计算节点的中间结果 (存在 shuffle 过程), 最后合并结果, 并由云服务管理器返回给用户.

(3) 优势与局限. 相比于传统的 MPP 型数据仓库, 这类架构能够提供更高的可用性, 更好的弹性, 以及更低的成本. 其主要面临的挑战为如何减少计算层与网络层之间的网络传输. 一方面, 即使计算层与网络层之间有高速网络, 但相比于本地缓存的带宽速度还是有数量级的差距. 另一方面, 云服务的计费与扫描的数据量和传输的数据量都有关, 所以需要尽量减少数据的网络传输.

(4) 典型系统. 基于云原生 OLAP 计算存储分离架构的系统主要以 Redshift[40]和 Snowflake[9]为代表. 二者都采用了类似的架构, 主要区别在于对计算层与存储层分别做了不同的优化. Snowflake 在计算层增加了统一的元数据管理, 使得查询能够被统一调度与优化. 其计算集群高度隔离, 每一个集群服务于一个租户, 且可采用不同配置的计算实例. 其在云存储之上建立了本地存储集群, 并通过懒加载的方式进行按需在线动态扩容[39]. 其查询处理基

于列式的向量化执行, 支持面向云存储的计算下推. Redshift 由最初的 MPP 型数据仓库转化为云原生数据库[20], 其基于 AWS (Amazon Web service) 生态在计算层与存储层之间增加了加速层, 包含了针对半结构化数据查询加速的 Specturm 集群, 结合 FPGA 进行计算下推的 AQUA (advanced query accelerator) 集群, 以及面向编译代码缓存的 CaaS (compilation as a service) 服务. 其同时增加了本地统一存储服务 RMS (redshift managed storage), 可支持 16 PB 级别的本地 SSD 缓存. 其查询处理基于 C++的代码生成, 并通过同地式连接进行分布式查询优化.

3.2 云原生 OLAP 计算-内存-存储分离的 3 层架构

如图 9(b) 所示, 第 2 类云原生 OLAP 数据库系统采用了计算-内存-存储分离的 3 层架构. 计算层、内存层、存储层之间同样采用高速网络进行连接, 采用这类架构的主要代表系统为谷歌的 BigQuery[10]. 与第 1 类架构类似, 这类架构的计算层包含了云服务管理器和计算云, 计算节点可使用本地的 SSD 存储作为缓存. 不同的是, 其计算集群采用一种统一调度的方式管理所有的工作节点. 另外, 其增加了一层分布式共享内存层, 用于处理复杂查询的数据洗牌 (shuffle) 操作, 避免中间结果写入磁盘造成 I/O 代价. 最底层的存储云同样提供高可用与弹性的数据存储服务.

(1) 主要设计动机. 与第 1 类架构不同, 这类架构在计算节点到存储节点之间添加了共享内存层, 用于多个计算节点之间进行数据合并操作, 例如不同数据表之间的连接与聚集运算等. 通过这种共享内存节点的方式进行合并运算, 能够有效降低计算节点之间进行合并操作的数据传输量与磁盘 I/O, 提高查询的执行效率. 另外, 通过统一的集群调度, 其能获得更高的资源利用率. 由于内存层可以进一步独立调度, 其系统弹性也很高.

(2) 查询执行流程. 这类云架构的查询处理分为以下 4 个步骤: (i) 查询优化. 云服务管理器接受查询请求后, 查询优化器结合元数据存储的统计信息进行查询重写与优化, 并生成并行查询计划. (ii) 基于缓存的查询处理. 查询子计划被发送到计算集群的各个计算节点, 如果缓存命中, 则直接将查询在本地处理后发送给共享内存层. 反之, 则向存储云请求数据. (iii) 基于内存的数据洗牌操作. 计算节点从存储云加载到数据后, 则访问对应的列式文件, 并通过下推计算到存储文件, 最后将过滤后的结果发送给共享内存层. 共享内存层通过多阶段的数据洗牌操作后将相同键值的数据发送给同一节点. (iv) 结果合并与返回. 计算集群合并各个计算节点的中间结果 (不存在 shuffle 过程), 最后合并结果将由云服务管理器返回给用户.

(3) 优势与局限. 相比于传统的 MPP 型数据仓库, 这类架构同样能够提供更高的系统吞吐量, 更好的弹性, 以及更高的资源利用率. 其主要面临的挑战为如何提升系统吞吐的同时能够降低查询执行的费用. 一方面, 分配的共享内存空间过大会造成服务的费用激增. 另一方面, 分配的共享内存空间太小会造成数据溢出, 中间结果会被写入到本地磁盘中, 造成大量磁盘 I/O 代价和网络传输代价.

(4) 典型系统. 基于云原生 OLAP 计算-内存-存储分离架构的系统主要以 BigQuery[7]为代表. 其源自于 Google 的交互式查询系统 Dremel[41,42], 通过将其改造为云原生系统后增加了共享内存层, 以加速复杂查询. 其云存储实现在 Google 的分布式文件系统 Colossus 上, 实现了数据的高可用存储. 文件的格式采用了自研的列存格式 Capacitor, 支持查询在压缩数据上进行直接执行. 针对分布式查询处理, 其采用异步的生产者-消费者执行模式, 生产者不断生成新的数据分区并分发给不同的节点进行数据处理, 其中 shuffle 过程通过在共享内存层进行加速, 消费者异步地读取不同的节点的输出结果, 将它们合并后在本地执行聚集操作. 如果数据超过内存层的容量, 则进行溢出操作.

3.3 不同架构 OLAP 系统特性总结

表 3 总结了云原生 OLAP 数据库系统不同架构设计的特性, 分别从分析处理的吞吐能力、数据读取的效率、资源的隔离性、弹性扩容能力以及部署的成本角度进行分析.

表 3 云原生 OLAP 数据库架构

云原生OLAP系统架构	代表性系统	吞吐	效率	隔离性	弹性	成本
计算-存储分离架构	Redshift[16]、Snowflake[9]	高	中	高	高	低
计算-内存-存储分离	BigQuery[10]	极高	高	中	极高	中等

第 1 类系统采用了计算-存储分离架构, 需要计算云服务以及存储云服务的支持, 类似于 OLTP 的计算-存储分离架构的系统. 但是不同之处在于, OLAP 系统面向大规模数据分析, 其存储云节点按照数据分片进行管理, 不需要分析日志数据保证了数据分析过程的高吞吐率. 此外, 上层计算云节点不区分主节点和备节点进行统一的管理, 计算集群中的工作节点独立地处理各个数据分片的分析任务, 且计算集群之间具有良好的隔离性. 系统支持计算和存储的单独扩展, 具有较强的弹性资源调度能力和相对低的服务成本.

第 2 类系统采用了计算-内存-存储分离架构, 需要计算云服务、内存云服务以及存储云服务的支持, 类似于 OLTP 系统的计算-缓存-存储分离结构. 不同之处在于, 内存云服务不是用于缓存底层存储层数据, 而是用于不同计算节点间进行数据合并操作, 通过共享内存降低数据合并时大量的节点间网络通信, 进一步提高了数据分析的吞吐能力. 但是, 节点间的共享内存降低了不同计算任务间的隔离性. 额外的内存云服务的引入要求具备大规模内存的服务器设备, 相较于第一类系统架构提高了服务成本. 但系统支持在计算、内存和存储资源上的单独扩展, 因此具备更强的资源弹性调度能力.

4 云原生 OLAP 数据库关键技术

云原生 OLAP 数据库关键技术主要包含云存储管理技术、查询处理技术、无服务器感知计算 (serverless computing) 技术、数据保护技术、机器学习技术 5 个方面. 如图 10 所示, 云存储管理作为云数据库的底座, 支撑着上层的云服务; 无服务器感知计算与查询处理模块接受来自上层多租户的 SQL 请求, 并通过与云存储交互以提供数据的查询服务; 数据保护模块对整体云服务进行端到端的保护; 机器学习技术一方面对云数据库服务进行优化, 另一方面基于弹性的云数据库服务进行自动化机器学习管理.

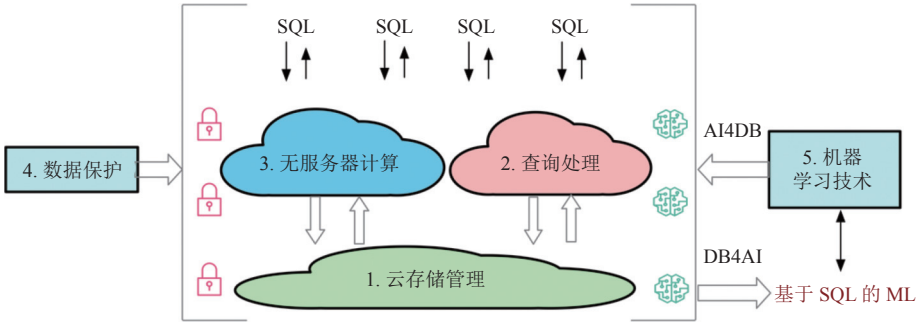


图 10 云原生 OLAP 数据库关键技术总览

后文表 4 列出了每一个技术类别的关键技术以及它们的优缺点. 首先, 存储管理技术涉及元数据管理、关系数据划分以及半结构数据表示 3 个部分; 查询处理技术分为结合了下推计算、Shuffle 内存层以及缓存的列式扫描的 3 类技术; 无服务器感知计算主要分为基于数据库实例的 Serverless 以及基于函数服务的 Serverless; 数据保护分为基于密钥管理算法的数据保护与基于可信计算环境“飞地 (Enclave)”的数据保护; 机器学习技术分为两个部分: 基于机器学习的云原生数据库技术和面向云原生数据库的自动机器学习.

4.1 存储管理技术: 元数据管理、关系表数据组织与半结构化数据管理

云原生 OLAP 数据库系统的存储管理包含 (1) 元数据管理; (2) 关系表数据组织; (3) 半结构化数据管理这 3 个部分. 元数据管理对云原生数据库系统的存储管理与查询优化都非常重要, 由于云原生 OLAP 数据库基本都不支持索引操作, 因此元数据可用于支持数据剪枝, 数据克隆, 以及数据的多版本控制 (multiversion concurrency control, MVCC) 与查询访问, 其主要的优点是可以优化查询, 但目前的主要的问题是其更新代价较高. 云存储服务可为计算层提供高可用、弹性的存储服务, 但云原生数据库也可对本地集群的缓存数据进行管理, 这主要涉及数据的组织与划分技术, 我们介绍基于连接图的关系数据划分, 其能基于启发式方法快速选择较优方案, 目前主要问题是只考虑了连接的频率, 未考虑查询代价. 最后, 由于云平台上的用户也拥有许多半结

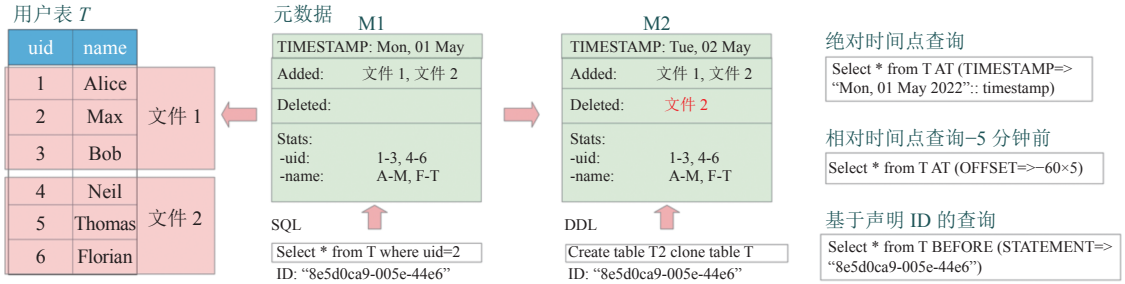
构化的数据, 因此, 如何在云平台上对半结构化数据进行管理也是一个很重要的问题, 我们将介绍基于列存的半结构化表示.

表 4 云原生 OLAP 数据库关键技术总览与优缺点比较

技术类别	关键技术	代表性工作	主要优点	主要缺点
存储管理技术	基于元数据的剪枝、拷贝与MVCC	Snowflake[9]	高吞吐	数据更新代价高
	基于连接图的关系数据划分	Redshift[40]	高效率	未考虑查询代价
	基于列存的半结构化数据表示	BigQuery[10]	存储代价低	查询延迟高
查询处理技术	列式扫描结合下推计算	PushdownDB[43]	低代价	没有结合缓存
	列式扫描结合Shuffle内存层	BigQuery[10]	高吞吐	高代价
	列式扫描结合下推计算和缓存	FlexpushdownDB[44]	高吞吐	扩展性低
无服务器感知计算	基于数据库实例的Serverless	Athena[45]	高吞吐	高代价
	基于函数服务的Serverless	Starling[46]	高度弹性	函数之间无法通信与交换状态
数据保护技术	基于密钥管理算法的数据保护	Snowflake[9]	高度扩展性	明文数据查询
	基于Enclave的数据保护	Azure SQL Database[47]	高度安全性	查询效率低
机器学习技术	基于机器学习的云原生数据库技术	Redshift[40]	高准确度与高性能	模型的适应性低
	面向云原生数据库的自动机器学习	Sagemaker[48]	高度扩展性与弹性	模型训练代价高

4.1.1 基于元数据的剪枝、拷贝与多版本并发控制 (MVCC)

元数据常被称为数据字典或“数据的数据”, 其主要存储数据的统计信息与版本信息. 针对云原生 OLAP 数据库, 此模块位于系统计算层的云服务管理器内, 可支持存储管理与查询优化, 其具体实现一般采用分布式的键值存储, 以保证高扩展性与高可用性, 如 Snowflake 采用了分布式的键值数据库实现其元数据存储. 下面介绍元数据管理的 3 项关键技术: (a) 基于元数据的剪枝; (b) 零克隆技术; (c) 时间旅行操作; 如图 11(a) 所示, 对于用户表 T, 元数据 M1 存储了其时间戳 (timestamp), 对应分区的文件名 (文件 1 和文件 2), 以及各列的统计信息 (值域的范围). 对于 SQL 查询“Select * from T where uid=2”, 云服务管理器可首先访问其元数据 M1, 通过查找 uid 的值域的范围可将文件 2 进行剪枝, 只扫描文件 1. 利用元数据还可以实现零克隆技术, 如图 11(b) 所示, 对于复制表 T 到 T2 的 DDL 操作“Create table T2 clone table T”, 云服务管理器并不直接复制表 T 的数据到 T2, 而是通过新建一个元数据 M2 来记录源数据表的增量改变. 如图 11(c) 所示, 通过元数据记录的版本信息, 云服务还可支持通过 MVCC 机制进行历史数据版本查询, 即对不同版本的数据进行查询访问.



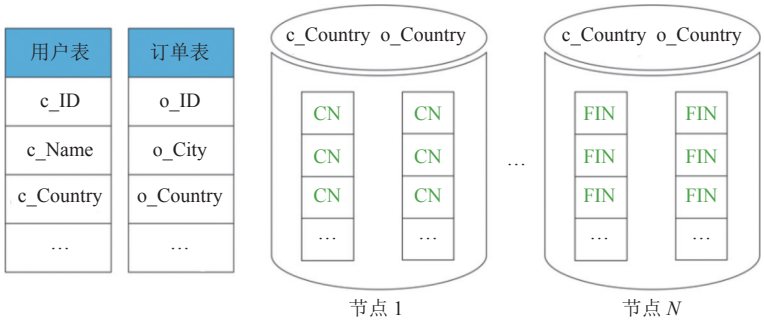
(a) 基于元数据的剪枝 (b) 零克隆技术 (c) 历史数据版本查询

图 11 元数据管理的 3 项关键技术

4.1.2 基于连接图的关系数据划分

针对云原生 OLAP 数据库, 关系表将被水平式地划分为多个子文件, 每个子文件类似于传统关系数据库的数据块或页面的概念, 但划分的文件一般不支持修改, 只能通过复制一份新文件的方式将更新操作附加在新文件中. 每个文件内部主要采用列式存储或混合行/列存储的格式, 文件头包含一些元数据以及每一列的存储偏移. 因此,

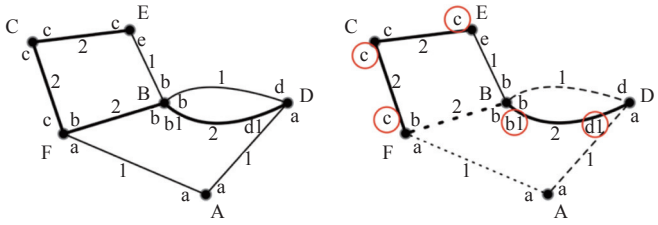
如果查询只访问文件中的部分列, 通过读取文件头的信息可以读取到相应列的位置, 而无需全表扫描. 文件组织好后可以通过云服务直接保存在云存储中 (如 AWS S3 对象存储云服务). 虽然云存储提供商能保证数据的跨域部署与高可用, 云原生 OLAP 数据库在本地集群也可将热数据分区后置放在不同的计算节点以加速查询的分布式执行. 划分数据的方法一般通过指定关系表的划分键, 然后根据划分键的哈希值产生对应的元组集合, 并将它们分配到不同的计算节点中. 这种方式使得相同哈希值的元组集合都只在本地做连接操作, 而无需节点之间进行数据传输. 如图 12(a) 所示, 假设数据库模式有两个关系表: 用户表 (c_ID, c_Name, c_Country,…) 和订单表 (o_ID, o_City, o_Country,…), 且它们能够通过 Country 列做连接运算. 如图 12(b) 所示, 如果按照 Country 列作为数据库的划分键, 通过哈希运算后, 拥有相同哈希值的元组集合会被分配到同一节点 (CN 的元组到节点 1, FIN 的元组到节点 N). 这样两表在 Country 列的连接运算都只在本地执行.



(a) 数据模式 (b) 数据划分

图 12 基于连接图的数据划分示例

然而, 划分键的选择问题是一个 NP-hard 问题, 即如何在候选列中选择一个最优的划分键方案非常困难, 尤其是针对表有很多连接键的情况下, 很难在庞大搜索空间下快速得到好的方案. 下面介绍一种基于连接图的划分键选择方法[49], 其基本思想是通过数据模式建立连接图, 其中点为数据表, 边为两个列之间的连接, 边上的权重为查询当中涉及对应连接的次数. 问题的目标是尽可能让权重大的连接键都划分到同一节点. 该方法将划分问题形式化为一个最优化问题, 然后基于贪心算法进行问题的求解. 如图 13 所示, 数据模式中有{A, B, C, D, E, F}这 6 个表顶点, 顶点之间的连接用边表示, 并且顶点附近的小写字母表示了具体的连接键, 边中间的数字表示了边的权重. 注意到顶点之间可能有多条边 (B、D 两顶点之间有 b-d, b1-d1 两条边). 算法分为两个阶段, 第 1 阶段通过最大匹配算法生成两个不相交的候选集 R1={C.c, F.c}和 R2={F.b, B.b}. 其中, R1 和 R2 的权重分别为 4 和 2. 第 2 阶段通过贪心算法扩展 R1 的集合, 最后 R1={C.c, F.c, B.b1, D.d1}, R1 的总权重为 6, 可得到一个较好的划分.



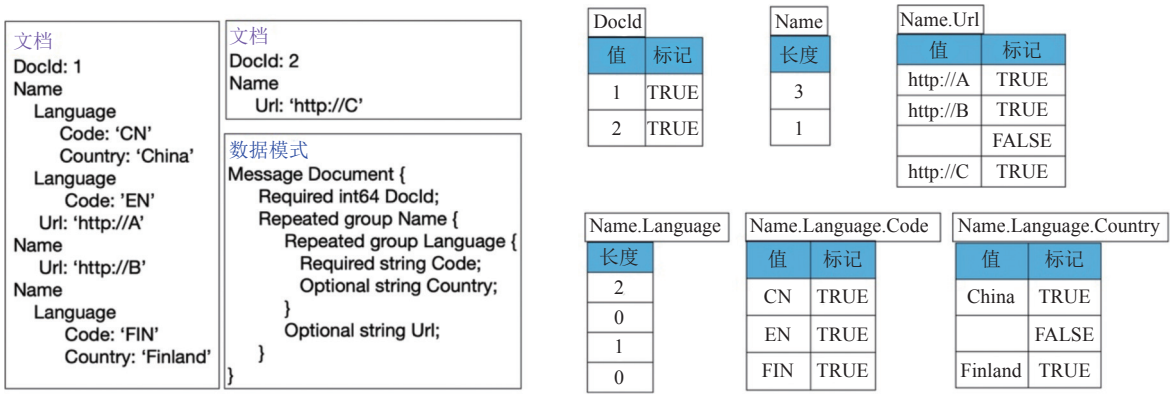
(a) 连接图 (b) 权重为 6 的划分

图 13 数据划分示例[49]

4.1.3 基于列存的半结构化数据表示

传统关系数据库面向的主要业务场景为银行事务系统与企业 ERP (enterprise resource planning) 系统, 这些场景的数据模式比较固定. 而云原生 OLAP 数据库也面向许多互联网应用, 所以需要管理大量的半结构化文档数据如 HTML 网页, XML 与 JSON 文档, 它们的数据模式很灵活, 常常有嵌套结构, 且不采用统一标准化的数据模式.

目前云原生数据库主要有两种方法管理半结构化数据: 一是根据文档模式提前转换文档为列存数据[37]. 二是采用变长字段存储文档数据, 在查询数据时采用自动类型推断[9]. 第 1 种方法为根据给定文件模式将多个文档转化为列存表示. 其长度列记录当前字段出现的次数, 标记列记录当前字段是否为空. 图 14(a) 给出了两个嵌套文档与它们的模式, 其中模式规定了每个字段的类型以及是否是可重复或者可选字段. 例如 Name 与 Language 为可重复字段, 即可嵌套其他字段. Country 与 Url 为可选字段. 图 14(b) 根据其数据模式将半结构文档表示成了列存形式. 例如, 对于文档 1, Name 字段重复了 3 次, 其嵌套的 Url 字段前 2 个对象有值, 第 3 个对象为空. 对于 Language 字段, 其在文档 1 的第 1 个位置重复了 2 次, 第 3 个位置重复了 1 次, 嵌套的字段包括 Code 和 Country.



(a) 文档与模式 (b) 基于列存的半结构数据表示

图 14 基于列存的半结构化数据表示[42]

4.2 查询处理: 结合计算下推与数据缓存的查询处理

云原生 OLAP 数据库系统面向分析性负载采取计算与存储分离的架构, 以实现计算节点与存储节点的细粒度弹性伸缩. 但由于计算层与存储层的分离, 中间的网络传输成为了瓶颈, 因此如何减少从存储层到计算层的数据传输量是云原生 OLAP 数据库要解决的一个重要问题. 目前的数据库主要采取算子下推技术结合本地缓存减少网络传输量, 下面具体介绍.

4.2.1 基于计算下推的查询处理

基于计算下推的查询处理技术[9,40]将计算下推到存储层, 以减少数据的网络传输量. 如亚马逊的 S3 的 Select 服务提供了基本操作的接口, 包括选择, 过滤, 与简单聚集操作. 用户可指定云存储 S3 对象所在的桶 (bucket) 与键 (key), 然后利用 SQL 语句对 S3 对象进行过滤, 只返回需要的数据. 由于 S3 Select 服务需要计费, 因此其主要挑战为如何既能利用 S3 Select 的算子下推减少数据传输量, 又能尽量减少调用 S3 Select 的次数.

基于 S3 Select 提供的原子操作, 还可实现索引、连接、聚集等复杂操作[42]. 对于 S3 Select 的索引实现, 首先在 S3 中存储一个基于偏移的索引表, 然后基于索引表进行 S3 对象的数据搜索. 其过程主要分为两个阶段: (1) 对于给定谓词, 利用 S3 Select 查找索引表中所有的相关记录, 返回给计算节点; (2) 根据返回的每条索引记录, 计算节点根据偏移值利用 HTTP 请求向 S3 存储获取数据. 基于索引表的 S3 Select 适合于选择率高的查询, 可大幅减少存储端的数据传输量. 对于 S3 Select 的两表连接实现, 首先针对小表建立布隆过滤器, 然后将得到的二进制数组利用 S3 Select 发送给第 2 个大表在存储端进行连接操作. 对于聚集实现也可通过两个阶段实现, 首先对目标对象表扫描获得对应列的所有唯一值, 然后将每一个唯一值发送给 S3 Select 进行聚集.

4.2.2 基于 Shuffle 内存层的查询处理

基于 Shuffle 内存层的查询处理技术[41]基于 Shuffle 内存层进行查询处理. 其从存储层读取列式数据, 将简单的谓词做下推计算后, 结合 Shuffle 内存层进行查询并发处理. 具体来说, 其将本地节点下推后的数据划分后分发到不同的工作节点进行三阶段 (map-shuffle-reduce) 的计算. 其中, 中间的 Shuffle 过程在统一内存层进行, 这避免了数据的重分发以及中间结果的磁盘写入, 因此其吞吐量很高. 然而, 针对中间结果集很大的查询, 该方法会产生

较高的代价, 因此需要限定内存层的大小, 如果结果集超过内存大小, 需要将数据溢出到磁盘.

4.2.3 结合本地缓存与计算下推的查询处理

结合本地缓存与计算下推的查询处理技术[44]结合本地缓存与云存储的混合计算下推. 对于查询计划生成, 一般采取两个经验法则: (i) 本地缓存下推比云存储下推更高效; (ii) 云存储下推比计算节点加载数据后计算更高效. 如图 15 所示, 在云存储上, 表 R(A, B) 和表 S(C, D) 都将每个属性划分为两个分段后保存. 同时缓存中有 4 个列分段 (即 A1, B1, C1, D2). 当前 SQL 查询涉及关系表 R 与 S 在属性 A 和 C 上的连接, 以及属性 B 与 D 上的基于范围谓词的过滤. 查询计划同时利用本地缓存和云存储进行计算下推, 其中, A1, B1, D2 都在本地缓存进行了计算下推, 其余分段都在云存储进行了计算下推. 注意到 C1 分段并没有从本地缓存获取, 这是由于 D1 不在缓存, 所以没有进行计算下推; 另外, C2 分段在云存储端也通过 D2 进行了计算下推, 但根据经验法则 (i), 最后 D2 的数据从缓存中加载. 由于本地缓存中不包含所有的属性 C 和属性 B 的数据, 本例中的哈希连接和聚集操作也没有下推, 而是通过合并数据后在计算端进行了连接与聚集.

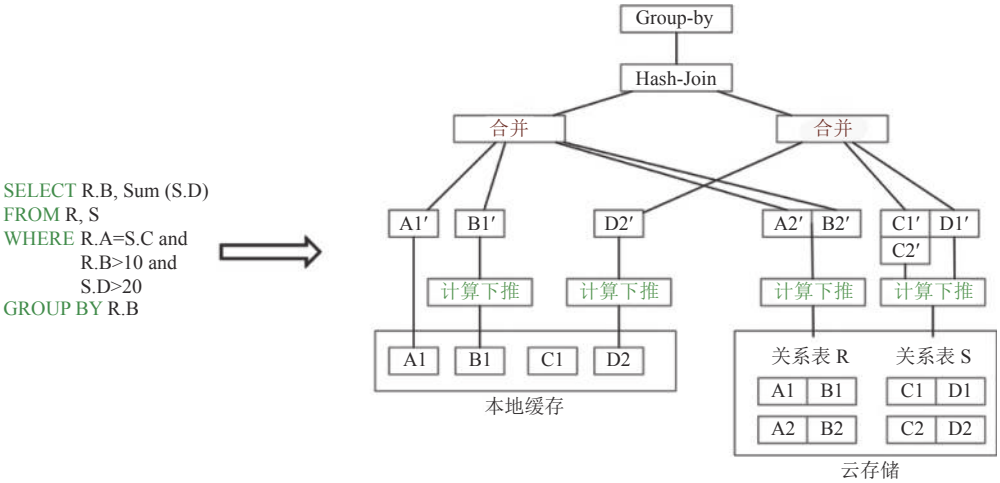


图 15 结合数据缓存与计算下推的查询处理[44]

4.3 无服务器感知计算: 基于数据库实例或基于函数的弹性计算

无服务器感知计算 (serverless computing) 被认为是下一代云计算的主要形式, 其主要目标为从第 1 代云计算的简化基础设施与系统管理变换到新一代云计算的简化用户编程[50]. 针对数据库管理系统的无服务器感知计算, DBA 只需要编写查询而无需关心数据库的资源配置, 云提供商即可按需为提交的查询调度资源. 目前主要有两种方式: (1) 基于数据库实例进行弹性调度资源; (2) 基于 Lambda 函数的弹性计算.

4.3.1 基于数据库实例的无服务器感知计算

基于数据库实例的无服务器感知计算技术[15,40,41,47,51]基于数据库实例提供无服务器化的查询引擎计算服务, 并按照整个实例的启动和暂停进行整体的调度, 即实例启动运行时按照计算量计费, 实例暂停时进行资源回收, 不对用户进行收费. 然而这类方法的主要难点在于数据库系统的启动和停止具备较长的延时, 因此, 如何降低系统启停产生的额外延时是这一研究方法的关键. 目前最先进的方法是基于时间序列的建模方法[47]. 具体来说, 其基于历史工作负载的时间序列建模, 然后动态调整数据库系统运转状态. 其在用户的工作负载到来前提前启动云数据库服务, 使得用户的等待时延大幅降低. 当短时间无工作负载的情境下该技术也不会停止系统运行, 因为用户可能还会随时登入执行负载. 当长时间没有用户请求, 才会暂停实例. 这类技术主要的问题是系统的资源调度机制对用户不透明, 用户可能会被强制超额使用服务, 因此需要公平的审计机制来保证其合理性.

4.3.2 基于函数服务的无服务器感知计算

函数即服务 (function as a service, FaaS) 是近年来云数据服务领域发展的另一个关键技术[46,52,53]. 其通过云服

务的框架, 来支持用户的敏捷开发. 其核心特点为无服务器感知计算: 即将一些常见的计算功能包装成由云服务平台统一调度的函数模组, 用户直接基于这些函数模组进行程序设计, 因此可以实现毫秒级别的程序启动, 并实现高速弹性调度的能力 (可同时启动上百的并发函数). 目前大型云厂商都提供了相应的基于 Lambda 函数的无服务计算, 如 AWS Lambda、Azure Functions、Google Cloud Functions. 但针对数据库的查询计算, 基于 Lambda 函数的无服务器感知计算有两大难点. 首先, 每个函数是有生命周期的 (约 5 min), 因此需要有一定的机制来保存每个函数计算后的结果状态, 以达到成功执行查询计划的目的. 其次, 当并发执行多个函数会出现某些函数由于资源匮乏而滞后, 从而导致整个查询计划执行缓慢, 即木桶效应. 对于第 1 个难点, 可通过云存储来保存中间状态. 对于第 2 个难点, 目前主要通过启动并发复制任务的方式来解决. 如图 16 所示, 云上用户提交向协调器提交 SQL 查询, 协调器对查询进行解析并生成查询计划, 然后将查询计划的代码生成后上传到云端函数服务 (如 AWS Lambda). 同时, 协调器向云端函数服务请求调用多个并发函数以执行查询. 云端函数服务以 Lambda 函数为单位, 从云存储中读取数据后进行计算, 然后将中间结果写入云存储中. 后续的函数通过读取中间状态数据继续执行查询的计算. 查询执行成功后结果最后返回给用户.

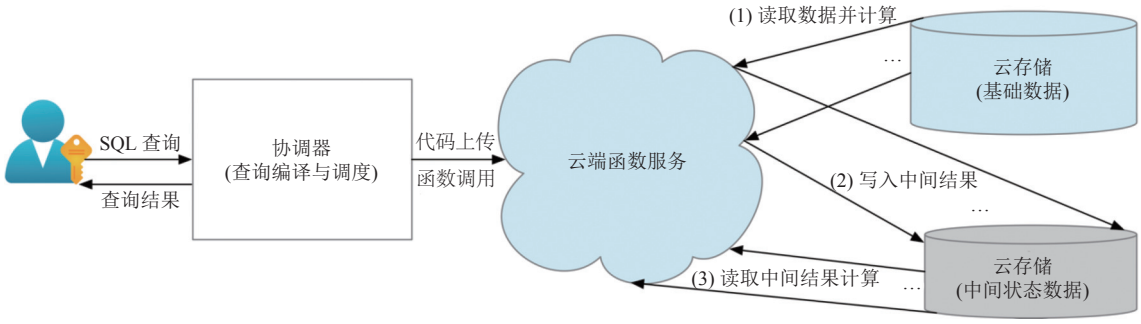


图 16 基于函数服务的无服务查询执行

4.4 数据保护技术: 基于密钥管理算法或基于 Enclave 的数据保护

目前云数据库的数据保护技术主要包含两大类: (1) 基于密钥管理算法的数据保护和 (2) 基于可信计算环境 Enclave 的数据保护.

4.4.1 基于密钥管理算法的数据保护技术

基于密钥管理算法的数据保护技术采用加密算法和密钥管理策略以保护数据. 例如, Snowflake[9]采用 AES 256 位的对称加密算法结合层次模型的密钥管理, 其提出了基于层次结构的密钥管理架构, 即自顶向下从根密钥、账户密钥、表密钥、文件密钥 4 个层次对密钥进行管理, 其还通过密钥的轮转机制定期对密钥进行更换. 这类技术易于实现, 并基于公有云平台 (如 AWS) 提供安全保护; 数据在导入、传输、存储、导出的时间点都处于加密状态, 但在查询处理时需要解密后再进行处理, 未来可研究在密文上进行查询的算法.

4.4.2 基于 Enclave 的数据保护技术

基于 Enclave 的数据保护技术基于硬件技术进行数据保护, 且用户数据仅在硬件区域内被解密处理, 数据具有更高的安全性. 例如, 微软的 Azure SQL 基于可信计算环境 Enclave 进行数据保护[54]. 其假设数据库系统与云服务提供商均为不可信环境, 仅客户端与 Enclave 区域为可信环境. 针对查询处理, 用户通过自身的密钥通过验证服务后, 向加密数据库提交查询, 然后数据以密文的方式被传输到 Enclave 后才进行解密后的查询处理. 这类技术当前的主要挑战为查询的效率与扩展性问题, 这主要是受限于 Enclave 的内存大小与查询处理能力.

4.5 融合云原生数据库与机器学习的相关技术

结合云原生数据库技术与机器学习技术 (artificial intelligence, AI) 是当前云上数据管理的一个重要发展趋势. 一方面, 机器学习技术可以被用于优化许多云原生数据库的关键任务, 包括负载管理、数据划分键选择、索引选择、参数调优等[55]. 另一方面, 机器学习技术可以基于云原生数据库为用户提供智能化的机器学习服务, 比如, 数

据库前端可基于 SQL 进行模型建立与推断, 后端基于可扩展的弹性计算与存储进行自动化机器学习管理.

4.5.1 基于机器学习的云原生数据库技术

基于机器学习的云原生数据库技术采用高级的机器学习技术优化云数据库的服务. 例如, Redshift 的自动化负载管理工具 AutoWLM 通过训练 XGBoost 模型预测查询的内存消耗量与执行时间, 从而能够自动地为用户的负载提前调整并发度[40]. 文献 [56]提出了利用深度强化学习来解决云数据库的数据划分键的选择问题, 其将划分键相关的特征编码为 [表, 查询频率, 外键] 的一个向量, 然后利用强化学习的探索与利用机制进行不同划分键组合的搜索. 文献 [57] 利用强化学习进行云数据库上的索引选择, 其基于蒙特卡洛搜索树进行不同索引组合的选择探索, 然后选择当前的最优方案. 总的来说, 基于机器学习的云原生数据库技术能够利用历史数据训练出准确度很高的模型, 从而优化服务性能. 当前这类技术面临的挑战是模型的适应性比较低, 为了解决负载异构性对模型准确度的影响, 这类技术往往需要针对用户的每一个集群或每一类负载单独地训练一个模型. 因此, 未来需要研究面向云原生数据库的机器学习模型迁移技术.

4.5.2 基于云原生数据库的机器学习技术

面向云原生数据库进行机器学习主要有 3 个优势. 首先, 它提供了基于 SQL 的机器学习途径, 使得用户可以更方便快捷地进行机器学习. 例如, Redshift 支持用户通过“Create Model”的 SQL 语法建立模型, 并通过指定目标列进行训练后用于后续预测任务. 其次, 基于云原生数据库进行机器学习可以做到“数据不动模型动”的优化, 使得训练好的模型可以在本地进行推断, 减少了数据传输的开销. 最后, 基于云原生数据库的机器学习可以利用云原生数据库的弹性计算和存储能力进行更好的模型自动化管理. 例如, 自动化机器学习系统 Sagemaker[48]依托 Redshift 可启动多个不同的实例, 并开启 250 个不同的任务, 针对不同的数据样本进行模型选择与自动调参. 面向云原生数据库的机器学习技术主要的挑战是模型的训练成本高, 相对于本地进行机器学习, 用户要额外负担在云上的数据存储、模型训练以及模型推断的成本. 因此, 未来需要解决如何结合用户的预算提供服务的问题.

5 云原生数据库关键技术的未来研究方向与挑战

云原生数据库技术方兴未艾, 带来了许多新的机会, 同时也面临着许多新的挑战. 总的来说, 主要有 4 个方面的问题需要解决, 包含: (1) 云原生多写技术; (2) 基于无服务器感知计算的资源调度; (3) 云原生 HTAP 数据库技术; (4) 面向多云的数据管理技术.

5.1 云原生多写技术

当前云原生 OLTP 数据库基本都只支持一写多读, 不能实现多节点写, 从而造成写扩展性受限, 特别是不能支持写需求大的应用. 另外, 当写节点故障时, 需要切换到备节点, 切换过程将导致秒级或分钟级业务恢复时间 (recovery time objective, RTO). 针对计算存储分离架构的多写技术, 目前有两种架构有待进一步探索. 第 1 种为基于 RDMA 的多写架构, 其通过 RDMA 来同步多个写节点对于远端共享存储的操作. 第 2 种为基于共享缓存的多写架构. 其多写节点可通过特定协议更新本地缓存, 以减少网络通信. 当前, 两种多写架构都面临写偏斜的问题, 即有大量的写操作集中在部分数据分片时, 系统的性能会大幅下降. 未来需要有特定技术解决该挑战.

5.2 基于无服务器感知计算的资源调度

针对云原生数据库的无服务器感知计算[50,58], 目前还存在诸多挑战. 首先, 针对基于函数服务的无服务计算, 如何对负载进行有状态的无服务器感知计算是一个主要挑战. 由于函数之间无法直接通信与共享状态, 需要一种新的基于缓存的机制进行状态共享, 当前基于云存储的状态交换机制延迟较高, 无法满足实时计算的需求. 另外, 如何针对函数服务进行调度与配置也非常具有挑战性, 这主要需要权衡并发度与代价之间的关系. 最后, 针对基于数据库实例的无服务器感知计算, 如何渐进式地进行平滑式的弹性扩容以避免震荡也非常具有挑战性. 此外, 结合数据库与函数服务进行无服务器感知计算也是一个重要方向.

5.3 云原生 HTAP 数据库技术

基于计算与存储的分离架构, 云原生数据库支持 HTAP[59,60]主要面临两大挑战. 首先, 尽管在云环境下, 计算

可以无限扩容, 但如何在计算层动态调度资源分别给面向 OLTP 与面向 OLAP 的计算节点使得云服务满足 SLA (service-level agreement) 协议的要求具有很大的挑战性. 一方面, OLTP 负载强度过高时会导致 OLAP 负载所读取的数据新鲜度不高. 另一方面, 分析型查询所资源过多会导致服务成本激增. 因此需要根据 SLA 协议通过资源调度在服务性能与数据新鲜度之间找到一个平衡点. 其次, 如何设计新的存储方式从而同时加速事务读取与查询分析操作也非常具有挑战性. 目前云原生针对事务型查询的存储使用行式的日志, 针对分析型查询的存储使用列式文件, 但由于从日志读取记录需要额外解析, 而列式文件需要合并增量日志, 这造成事务读取与查询分析操作性能都不高. 因此需要设计一种基于内存的统一存储格式来解决这个问题.

5.4 面向多云的数据管理技术

随着多云 (multi-cloud) 时代的到来, 企业会同时使用多家不同云计算厂商的服务[61], 这对数据管理技术提出了许多新的挑战. 首先, 如何在多云环境下提供高可用服务具有很大挑战, 即当部分节点停机或整个区域发生灾害, 如何利用多云环境进行快速恢复. 其次, 如何在多云环境下进行存储管理, 以便进行跨云的快速数据迁移也很有挑战性, 这主要由于不同云厂商下的数据组织形式多样化, 且云存储互相隔离, 所以在线迁移延迟很高. 最后, 如何针对多云环境设计高效且代价感知的查询优化算法也面临着巨大挑战[62].

6 总 结

本综述全面调研云原生数据库, 总结了它们的发展背景和核心优势, 并根据 OLTP 和 OLAP 两类不同的工作场景分别探讨其架构设计和关键技术. 本文凝炼了主流云原生数据库的架构选择和关键技术, 包括云原生 OLTP 数据库的数据组织技术、副本一致性技术、主备同步技术、故障恢复技术以及 HTAP 技术, 和云原生 OLAP 数据库的存储管理技术、查询处理技术、无服务器感知计算、数据保护技术以及基于机器学习的数据库优化. 最后, 本综述讨论了云原生数据库技术未来的研究方向和挑战.

致谢 本文由华为公司、好未来教育公司给予大力支持.

References:

[1] Global database management system (DBMS) market outlook. 2023. https://www.expertmarketresearch.com/reports/database-management-system-market

[2] Database-as-a-service (DBaaS): Global strategic business report. 2023. https://www.researchandmarkets.com/reports/4804281/database-as-a-service-dbaas-global-strategic

[3] China database industry market depth research and investment outlook report on 2023–2028. 2022 (in Chinese). https://m.huaon.com/detail/840559.html#reportmethod

[4] Li GL, Dong HW, Zhang C. Cloud databases: New techniques, challenges, and opportunities. Proc. of the VLDB Endowment, 2022, 15(12): 3758–3761. [doi: 10.14778/3554821.3554893]

[5] Mathew S, Varia J. Overview of Amazon Web services. Amazon Whitepapers, 2014, 105: 1–22.

[6] Mazumdar P, Agarwal S, Banerjee A, Mazumdar P, Agarwal S, Banerjee A. Azure SQL database. Pro SQL Server on Microsoft Azure. 2016. 129–156.

[7] Krishnan SP, Gonzalez JL, Krishnan SP, Gonzalez JL. Google Cloud SQL. Building Your Next Big Thing with Google Cloud Platform: A Guide for Developers and Enterprise Architects. New York: Apress, 2015.

[8] Verbitski A, Gupta A, Saha D, Brahmadesam M, Gupta K, Mittal R, Krishnamurthy S, Maurice S, Kharatishvili T, Bao XF. Amazon aurora: Design considerations for high throughput cloud-native relational databases. In: Proc. of the 2017 ACM Int'l Conf. on Management of Data. Chicago: ACM, 2017. 1041–1052. [doi: 10.1145/3035918.3056101]

[9] Dageville B, Cruanes T, Zukowski M, Antonov V, Avanes A, Bock J, Claybaugh J, Engovatov D, Hentschel M, Huang JS, Lee AW, Motivala A, Munir AQ, Pelley S, Povinec P, Rahn G, Triantafyllis S, Unterbrunner P. The snowflake elastic data warehouse. In: Proc. of the 2016 Int'l Conf. on Management of Data. San Francisco: ACM, 2016. 215–226. [doi: 10.1145/2882903.2903741]

[10] Bisong E. Google BigQuery. In: Building Machine Learning and Deep Learning Models on Google Cloud Platform: A Comprehensive Guide for Beginners. Berkeley: Apress, 2019. 485–517. [doi: 10.1007/978-1-4842-4470-8_38]

[11] Cecchet E, Singh R, Sharma U, Shenoy P. Dolly: Virtualization-driven database provisioning for the cloud. ACM SIGPLAN Notices, 2011, 46(7): 51–62. [doi: 10.1145/2007477.1952691]

[12] Pritchett D. BASE: An acid alternative: In partitioned databases, trading some consistency for availability can lead to dramatic improvements in scalability. Queue, 2008, 6(3): 48–55. [doi: 10.1145/1394127.1394128]

[13] Soror AA, Minhas UF, Aboulnaga A, Salem K, Kokosielis P, Kamath S. Automatic virtual machine configuration for database workloads. ACM Trans. on Database Systems, 2008, 35(1): 7. [doi: 10.1145/1670243.1670250]

[14] Antonopoulos P, Budovski A, Diaconu C, Hernandez Saenz A, Hu J, Kodavalla H, Kossmann D, Lingam S, Minhas UF, Prakash N, Purohit V, Qu H, Ravella CS, Reisteter K, Shrotri S, Tang DX, Wakade V. Socrates: The new SQL server in the cloud. In: In: Proc. of the 2019 Int'l Conf. on Management of Data. Amsterdam: ACM, 2019. 1743–1756. [doi: 10.1145/3299869.3314047]

[15] Cao W, Zhang YQ, Yang XJ, Li FF, Wang S, Hu QD, Cheng XT, Chen ZZ, Liu ZJ, Fang J, Wang B, Wang YH, Sun HQ, Yang Z, Cheng ZS, Chen S, Wu J, Hu W, Zhao JW, Gao YS, Cai SL, Zhang YY, Tong JW. PolarDB serverless: A cloud native database for disaggregated data centers. In: Proc. of the 2021 Int'l Conf. on Management of Data. ACM, 2021. 2477–2489. [doi: 10.1145/3448016.3457560]

[16] Gupta A, Agarwal D, Tan D, Kulesza J, Pathak R, Stefani S, Srinivasan V. Amazon redshift and the case for simpler data warehouses. In: Proc. of the 2015 ACM SIGMOD Int'l Conf. on Management of Data. Melbourne: ACM, 2015. 1917–1923. [doi: 10.1145/2723372.2742795]

[17] Pandis I. The evolution of Amazon redshift. Proc. of the VLDB Endowment, 2021, 14(12): 3162–3174. [doi: 10.14778/3476311.3476391]

[18] Levandoski J, Lomet D, Sengupta S. LLAMA: A cache/storage subsystem for modern hardware. Proc. of the VLDB Endowment, 2013, 6(10): 877–888. [doi: 10.14778/2536206.2536215]

[19] Levandoski JJ, Lomet DB, Sengupta S. The Bw-Tree: A B-tree for new hardware platforms. In: Proc. of the 29th IEEE Int'l Conf. on Data Engineering (ICDE). Brisbane: IEEE, 2013. 302–313. [doi: 10.1109/ICDE.2013.6544834]

[20] Levandoski J, Lomet D, Sengupta S, Stutsman R, Wang R. High performance transactions in deuteronomy. In: Proc. of the 2015 Conf. on Innovative Data Systems Research (CIDR). 2015.

[21] Gutmans A. Cloud blog databases. Introducing AlloyDB for PostgreSQL: Free yourself from expensive, legacy databases. 2022. https://cloud.google.com/blog/products/databases/introducing-alloydb-for-postgresql

[22] Azure Managed Disks pricing. 2022. https://azure.microsoft.com/en-us/pricing/details/managed-disks/

[23] Depoutovitch A, Chen C, Chen J, Larson P, Lin S, Ng J, Cui WL, Liu Q, Huang W, Xiao Y, He YJ. Taurus database: How to be fast, available, and frugal in the cloud. In: Proc. of the 2020 ACM SIGMOD Int'l Conf. on Management of Data. Portland: ACM, 2020. 1463–1478. [doi: 10.1145/3318464.3386129]

[24] Verbitski A, Gupta A, Saha D, Corey J, Gupta K, Brahmadesam M, Mittal R, Krishnamurthy S, Maurice S, Kharatishvilli T, Bao XF. Amazon aurora: On avoiding distributed consensus for I/Os, commits, and membership changes. In: Proc. of the 2018 Int'l Conf. on Management of Data. Houston: ACM, 2018. 789–796. [doi: 10.1145/3183713.3196937]

[25] Cao W, Liu ZJ, Wang P, Chen S, Zhu CF, Zheng S, Wang YH, Ma GQ. PolarFS: An ultra-low latency and failure resilient distributed file system for shared storage cloud database. Proc. of the VLDB Endowment, 2018, 11(12): 1849–1862. [doi: 10.14778/3229863.3229872]

[26] Zhang YQ, Ruan CY, Li C, Yang XJ, Cao W, Li FF, Wang B, Fang J, Wang YH, Huo JZ, Bi C. Towards cost-effective and elastic cloud database deployment via memory disaggregation. Proc. of the VLDB Endowment, 2021, 14(10): 1900–1912. [doi: 10.14778/3467861.3467877]

[27] Sheshadri Ranganath, Ravi Murthy. Cloud Blog Databases. AlloyDB for PostgreSQL under the hood: Columnar engine. 2022. https://cloud.google.com/blog/products/databases/alloydb-for-postgresql-columnar-engine

[28] 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]

[29] Prout A, Wang SP, Victor J, Sun Z, Li YZ, Chen J, Bergeron E, Hanson E, Walzer R, Gomes R, Shamgunov N. Cloud-native transactions and analytics in SingleStore. In: Proc. of the 2022 Int'l Conf. on Management of Data. Philadelphia: ACM, 2022. 2340–2352. [doi: 10.1145/3514221.3526055]

[30] Skeen D. A Quorum-based Commit Protocol. Ithaca: Cornell University, 1982.

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

[32] Lamport L. Paxos made simple. ACM SIGACT News (Distributed Computing Column), 2001, 12: 51–58.

[33] Lamport L. Fast paxos. Distributed Computing, 2006, 19(2): 79–103. [doi: 10.1007/s00446-006-0005-x]

[34] Chandra TD, Griesemer R, Redstone J. Paxos made live: An engineering perspective. In: Proc. of the 26th Annual ACM Symp. on Principles of Distributed Computing. Portland: ACM, 2007. 398–407. [doi: 10.1145/1281100.1281103]

[35] 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]

[36] Mohan C, Haderle D, Lindsay B, Pirahesh H, Schwarz P. ARIES: A transaction recovery method supporting fine-granularity locking and partial rollbacks using write-ahead logging. ACM Trans. on Database Systems, 1992, 17(1): 94–162. [doi: 10.1145/128765.128770]

[37] Gu JC, Lee Y, Zhang YW, Chowdhury M, Shin KG. Efficient memory disaggregation with INFINISWAP. In: Proc. of the 14th USENIX Conf. on Networked Systems Design and Implementation. Boston: USENIX Association, 2017. 649–667.

[38] Shan YZ, Huang YT, Chen YL, Zhang YY. LegoOS: A disseminated, distributed OS for hardware resource disaggregation. In: Proc. of the 13th USENIX Conf. on Operating Systems Design and Implementation. Carlsbad: USENIX Association, 2018. 69–87.

[39] Vuppalapati M, Miron J, Agarwal R, Truong D, Motivala A, Cruanes T. Building an elastic query engine on disaggregated storage. In: Proc. of the 17th USENIX Conf. on Networked Systems Design and Implementation. Santa Clara: USENIX Association, 2020. 449–462.

[40] Armenatzoglou N, Basu S, Bhanoori N, Cai MC, Chainani N, Chinta K, Govindaraju V, Green TJ, Gupta M, Hillig S, Hotinger E, Leshinksy Y, Liang JT, McCreedy M, Nagel F, Pandis I, Parchas P, Pathak R, Polychroniou O, Rahman F, Saxena G, Soundararajan G, Subramanian S, Terry D. Amazon redshift re-invented. In: Proc. of the 2022 Int'l Conf. on Management of Data. Philadelphia: ACM, 2022. 2205–2217. [doi: 10.1145/3514221.3526045]

[41] Melnik S, Gubarev A, Long JJ, Romer G, Shivakumar S, Tolton M, Vassilakis T. Dremel: Interactive analysis of Web-scale datasets. Proc. of the VLDB Endowment, 2010, 3(1–2): 330–339. [doi: 10.14778/1920841.1920886]

[42] Melnik S, Gubarev A, Long JJ, Romer G, Shivakumar S, Tolton M, Vassilakis T, Ahmadi H, Delorey D, Min S, Pasumansky M, Shute J. Dremel: A decade of interactive SQL analysis at Web scale. Proc. of the VLDB Endowment, 2020, 13(12): 3461–3472. [doi: 10.14778/3415478.3415568]

[43] Yu XY, Youill M, Woicik M, Ghanem A, Serafini M, Aboulnaga A, Stonebraker M. PushdownDB: Accelerating a DBMS using S3 computation. In: Proc. of the 36th IEEE Int'l Conf. on Data Engineering (ICDE). Dallas: IEEE, 2020. 1802–1805. [doi: 10.1109/ICDE48307.2020.00174]

[44] Yang YF, Youill M, Woicik M, Liu YZ, Yu XY, Serafini M, Aboulnaga A, Stonebraker M. FlexPushdownDB: Hybrid pushdown and caching in a cloud DBMS. Proc. of the VLDB Endowment, 2021, 14(11): 2101–2113. [doi: 10.14778/3476249.3476265]

[45] Amazon Athena. 2022. https://aws.amazon.com/cn/athena/

[46] Perron M, Fernandez RC, DeWitt D, Madden S. Starling: A scalable query engine on cloud functions. In: Proc. of the 2020 ACM SIGMOD Int'l Conf. on Management of Data. Portland: ACM, 2020. 131–141. [doi: 10.1145/3318464.3380609]

[47] Poppe O, Guo Q, Lang W, Arora P, Oslake M, Xu SZ, Kalhan A. Moneyball: Proactive auto-scaling in Microsoft Azure SQL database serverless. Proc. of the VLDB Endowment, 2022, 15(6): 1279–1287. [doi: 10.14778/3514061.3514073]

[48] Das P, Ivkin N, Bansal T, Rouesnel L, Gautier P, Karnin Z, Dirac L, Ramakrishnan L, Perunicic A, Shcherbatyi I, Wu W, Zolic A, Shen HB, Ahmed A, Winkelmolen F, Miladinovic M, Archembeau C, Tang A, Dutt B, Grao P, Venkateswar K. Amazon SageMaker Autopilot: A white box AutoML solution at scale. In: Proc. of the 4th Int'l Workshop on Data Management for End-to-end Machine Learning. Portland: ACM, 2020. 2. [doi: 10.1145/3399579.3399870]

[49] Parchas P, Naamad Y, Van Bouwel P, Faloutsos C, Petropoulos M. Fast and effective distribution-key recommendation for amazon redshift. Proc. of the VLDB Endowment, 2020, 13(12): 2411–2423. [doi: 10.14778/3407790.3407834]

[50] Schleier-Smith J, Sreekanti V, Khandelwal A, Carreira J, Yadwadkar NJ, Popa RA, Gonzalez JE, Stoica I, Patterson DA. What serverless computing is and should become: The next phase of cloud computing. Communications of the ACM, 2021, 64(5): 76–84. [doi: 10.1145/3406011]

[51] Zhan CQ, Su MM, Wei CX, Peng XQ, Lin L, Wang S, Chen Z, Li FF, Pan Y, Zheng F, Chai CL. AnalyticDB: Real-time OLAP database system at Alibaba cloud. Proc. of the VLDB Endowment, 2019, 12(12): 2059–2070. [doi: 10.14778/3352063.3352124]

[52] Sreekanti V, Wu CG, Lin XC, Schleier-Smith J, Gonzalez JE, Hellerstein JM, Tumanov A. Cloudburst: Stateful functions-as-a-service. Proc. of the VLDB Endowment, 2020, 13(12): 2438–2452. [doi: 10.14778/3407790.3407836]

[53] Kraft P, Li Q, Kaffes K, Skiadopoulos A, Kumar D, Cho D, Li J, Redmond R, Weckwerth N, Xia B, Bailis P, Cafarella M, Graefe G, Kepner J, Kozyrakis C, Stonebraker M, Suresh L, Yu XY, Zaharia M. Apiary: A DBMS-backed transactional function-as-a-service framework. arXiv:2208.13068, 2022.

[54] Antonopoulos P, Arasu A, Singh KD, Eguro K, Gupta N, Jain R, Kaushik R, Kodavalla H, Kossmann D, Ogg N, Ramamurthy R, Szymaszek J, Trimmer J, Vaswani K, Venkatesan R, Zwilling M. Azure SQL database always encrypted. In: Proc. of the 2020 ACM

SIGMOD Int'l Conf. on Management of Data. Portland: ACM, 2020. 1511–1525. [doi: 10.1145/3318464.3386141]

[55] Li GL, Zhou XH. Survey of data management techniques for 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]

[56] Hilprecht B, Binnig C, Röhm U. Learning a partitioning advisor for cloud databases. In: Proc. of the 2020 ACM SIGMOD Int'l Conf. on Management of Data. Portland: ACM, 2020. 143–157. [doi: 10.1145/3318464.3389704]

[57] Wu WT, Wang C, Siddiqui T, Wang JX, Narasayya V, Chaudhuri S, Bernstein PA. Budget-aware index tuning with reinforcement learning. In: Proc. of the 2022 Int'l Conf. on Management of Data. Philadelphia: ACM, 2022. 1528–1541. [doi: 10.1145/3514221.3526128]

[58] Jonas E, Schleier-Smith J, Sreekanti V, Tsai CC, Khandelwal A, Pu QF, Shankar V, Carreira J, Krauth K, Yadwadkar N, Gonzalez JE, Ada Popa R, Stoica I, Patterson DA. Cloud programming simplified: A Berkeley view on serverless computing. arXiv:1902.03383, 2019.

[59] Zhang C, Li GL, Feng JH, Zhang JT. A survey of key techniques of HTAP databases. Ruan Jian Xue Bao/Journal of Software, 2023, 34(2): 761–785 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6713.htm [doi: 10.13328/j.cnki.jos.006713]

[60] 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]

[61] Narasayya V, Chaudhuri S. Multi-tenant cloud data services: State-of-the-art, challenges and opportunities. In: Proc. of the 2022 Int'l Conf. on Management of Data. 2022. 2465–2473. [doi: 10.1145/3514221.3522566]

[62] Stoica I, Shenker S. From cloud computing to sky computing. In: Proc. of the 2021 Workshop on Hot Topics in Operating Systems. Ann Arbor: ACM, 2021. 26–32. [doi: 10.1145/3458336.3465301]

附中文参考文献:

[3] 2023–2028年中国数据库行业市场深度研究及投资前景展望报告. 2022. https://m.huaon.com/detail/840559.html#reportmethod

[55] 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]

[59] 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 [doi: 10.13328/j.cnki.jos.006713]

董昊文(1999－), 男, 博士生, CCF 学生会员, 主要研究领域为云数据库系统.

李国良(1981－), 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, 大数据分析和处理.

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, 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.