浙江大学
ZHEJIANG UNIVERSITY

《高级数据库技术》
关于云原生数据库系统的架构与技术研究的文献综述

姓 名 刘京宗

学 号 22451040

学 院 软件学院

实验日期 2024 年 10 月 17 日

授课教师 唐秀

关于云原生数据库系统的架构与技术研究的文献综述

刘京宗 22451040

摘 要 本报告围绕云原生数据库技术,分析了其在企业中的应用背景和发展趋势。以 PolarDB-X、PolarDB-IMCI、PolarDB-MP 以及 Galaxybase 等为代表的分布式云原生数据库系统为例，探讨了其核心架构设计、关键技术以及性能评估。报告涵盖了数据库弹性扩展、多主架构、HTAP 工作负载支持、跨数据中心事务一致性等技术难题的解决方案。通过测试与对比，展示了这些技术在提升数据库性能、资源利用率和数据一致性方面的重要作用。

关键词 云原生数据库，多主架构，分布式弹性，混合事务分析处理，HTAP，跨数据中心事务，一致性

Literature Review on the Architecture and Technology of Cloud-Native Database Systems

Abstract This report focuses on cloud-native database technologies, analyzing their application background and development trends in enterprises. Using distributed cloud-native database systems such as PolarDB-X, PolarDB-IMCI, PolarDB-MP, and Galaxybase as examples, the report explores key architectural designs, core technologies, and performance evaluations. It covers solutions for technical challenges such as elastic scaling, multi-primary architecture, HTAP workload support, and cross-data center transaction consistency. Through testing and comparisons, the report demonstrates the importance of these technologies in improving database performance, resource utilization, and data consistency.

Keywords Cloud-native database, multi-primary architecture, distributed elasticity, hybrid transactional and analytical processing, HTAP, cross-data center transaction, consistency

一 引言

随着云计算技术的普及，数据库系统的云端迁移已成为企业数字化转型的关键步骤。传统数据库系统面临的挑战在于无法适应快速变化的业务需求和大规模的数据处理压力，尤其是在高并发场景下，数据一致性、系统弹性和扩展性成为主要瓶颈。在此背景下，云原生数据库应运而生，旨在通过分布式架构、计算与存储分离、跨数据中心事务支持等技术手段，实现更高的性能、更强的弹性和更好的资源利用。

云原生数据库系统以“云原生架构”为核心，强调无缝支持云计算的弹性扩展、故障恢复和高可用性。这类数据库不同于传统的单体式架构，它们通过去中心化的协调机制和分布式存储技术，解决了数据一致性、分布式事务处理和混合事务分析处理（HTAP）等复杂场景中的技术难题。特别是在多数据中心、高可用性和混合工作负载处理方面，云原生数据库提供了极具优势的解决方案。

本文通过对多种云原生数据库系统（如 PolarDB-X、PolarDB-IMCI、PolarDB-MP 和 Galaxybase）的深入分析，探讨其架构设计、关键技术以及系统性能。特别是这些系统如何解决多主架构、事务一致性、数据弹性扩展、混合事务和分析处理等问题，以满足现代企业的复杂业务需求。通过具体的性能测试和应用场景分析，本报告为未来的数据库设计与优化提供了有力参考。

二 PolarDB-X：面向云原生应用的弹性分布式关系数据库[1]

1 背景

随着云计算的广泛应用，企业数据库系统逐步迁移到云端，促使新一代云原生数据库系统的诞生。这些系统面临的三大核心挑战包括：跨数据中心的高可用性和容灾能力：需要支持多数据中心部署，确保在任意单点失效时能够维持系统的运行。资源弹性和可扩展性：云应用的需求可能在短时间内急剧变化，系统需要能够迅速扩展计算和存储资源以适应突发流量。HTAP（Hybrid Transactional and Analytical Processing）工作负载的支持：需要在同一系统中同时高效地处理事务性和分析性工作负载，避免传统分离 OLTP 与 OLAP 系统所带来的数据冗余和复杂性。

PolarDB-X 是在这种需求下提出的一个分布式关系型数据库系统，旨在满足云原生应用对弹性、扩展性和混合工作负载支持的需求。PolarDB-X 基于 PolarDB 的云原生架构，提供了跨数据中心事务支持、快速弹性伸缩以及高效的 HTAP 处理能力。

2 系统架构设计

PolarDB-X 采用了云原生分离计算和存储的架构设计，将系统划分为三层：计算节点（CN）、数据库节点（DN）、存储节点（SN）。计算节点（CN）负责处理分布式事务和查询，包括事务协调、查询优化和执行。数据库节点（DN）负责处理单分片事务和跨数据中心的复制，并与存储节点进行交互。存储节点（SN）通过 PolarFS（PolarDB 的文件系统）持久化存储数据，提供数据存储和复制功能。这种设计允许每一层独立扩展，从而使系统能够应对大规模的读写请求并且保证系统的可扩展性和弹性。

图 1 展示了 PolarDB-X 的系统架构，这种架构设计极大增强了系统的弹性和可扩展性。例如，当负载增加时，计算节点和数据库节点可以分别扩展，而无需移动数据，大幅减少了系统扩展的时间和成本。

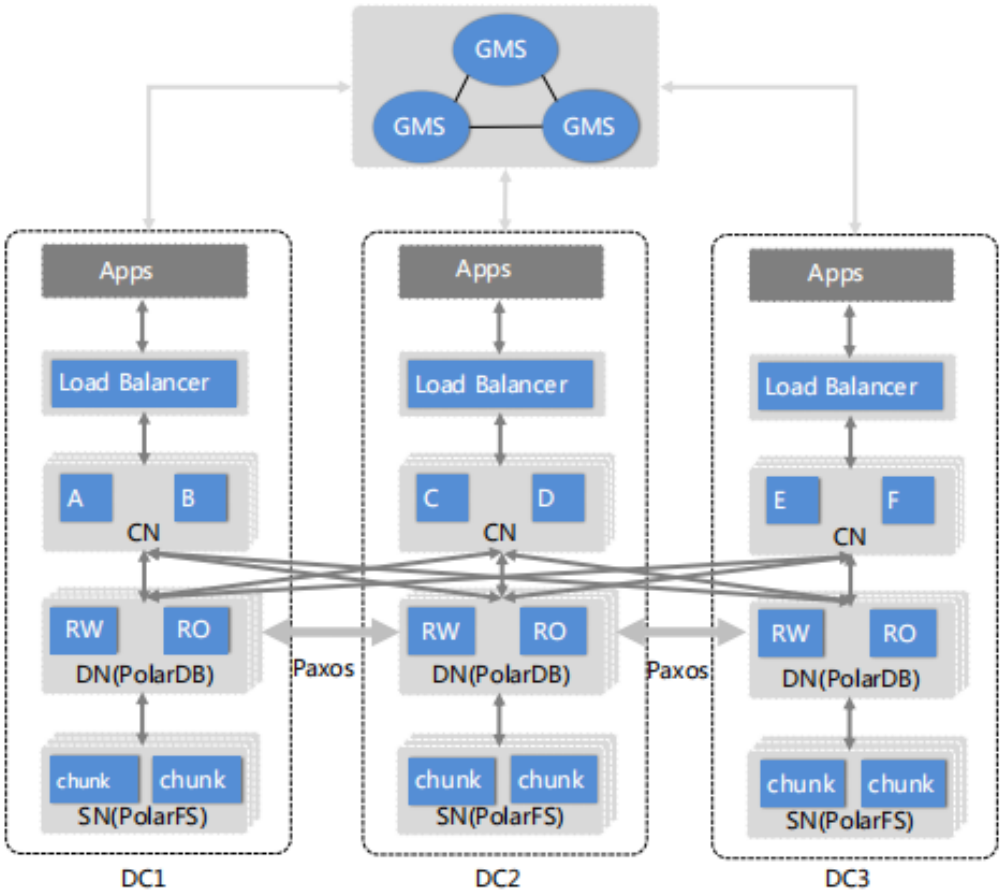


Fig. 1 PolarDB-X 的系统架构示意图

3 关键技术

3.1 HLC-SI 两阶段提交事务机制

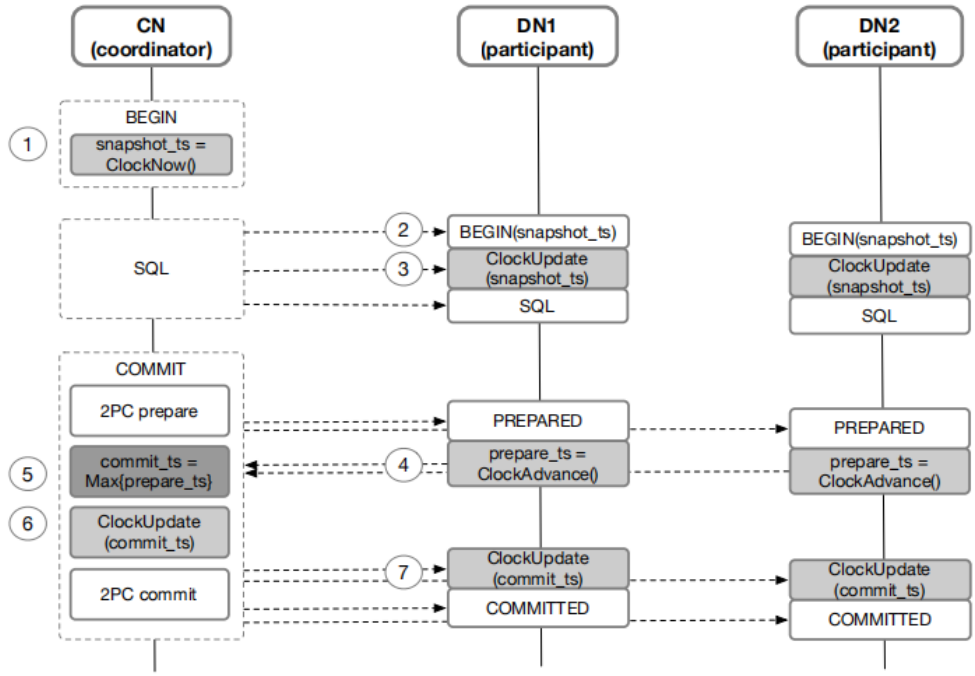


Fig. 2 HLC-SI 和两阶段提交

图 2展示了 HLC-SI（混合逻辑时钟快照隔离）在分布式数据库中的两阶段提交事务机制。HLC-SI 结合物理时钟和逻辑时钟，确保事务的顺序性和一致性，避免了传统时间戳服务的性能瓶颈。

事务开始：当事务开始时，协调者（CN）调用 ClockNow() 获取事务的快照时间戳（snapshot_ts），确定读取数据的版本，并将该时间戳发送给参与的数据库节点（DN）。

时钟同步：参与节点在接收 snapshot_ts 后，调用 ClockUpdate(snapshot_ts) 同步本地时钟，确保所有节点看到一致的数据版本。

预提交阶段：在两阶段提交的第一阶段，参与节点对事务的写集进行验证后，调用 ClockAdvance() 获取预提交时间戳（prepare_ts），并返回给协调者。

生成提交时间戳：协调者根据参与节点返回的 prepare_ts，选择最大值作为全局的提交时间戳（commit_ts），并将该时间戳发送给所有参与节点。

完成提交：参与节点收到 commit_ts 后，调用 ClockUpdate(commit_ts) 更新本地时钟，完成事务提交。

HLC-SI 具有如下优势：去中心化：通过分布式的时钟管理，避免了集中式时间戳服务的瓶颈和单点故障。减少延迟：无需频繁访问中心化的时钟服务，降低了跨数据中心事务的延迟。高可扩展性：每个节点独立管理时钟，同时通过轻量级同步确保全局一致性。

3.2 多租户架构

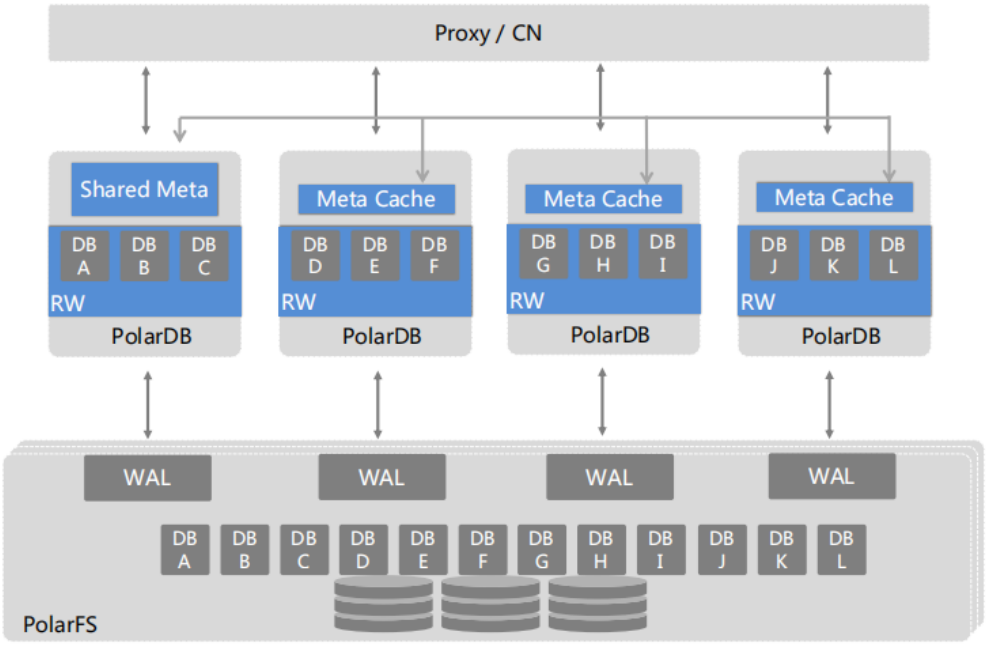


Fig. 3 PolarDB-X 多租户架构

PolarDB-X 将每个租户视为一个独立的逻辑实体，通常对应于一组数据库或表。为了提高资源利用效率，多个租户可以共存在一个数据库实例中，但这些租户在逻辑上是隔离的，避免了跨租户的事务冲突。

图 3展示了在 PolarDB-MT（多租户架构的 PolarDB）中，可以为不同的租户分配多个读写节点 (RW nodes)，以扩展系统的写入能力。这种设计打破了传统数据库单点读写的限制，使得租户之间的写操作不会相互干扰。每个读写节点 (RW node) 可以处理不同租户的数据，且各节点的写操作相互独立。例如，在图 3中，多个 RW 节点分别绑定不同的数据库（DB A-F、DB G-L），因此每个节点都专注于特定租户的数据写入。所有 RW 节点共享底层的存储系统（如 PolarFS），使得不同节点之间的数据保持一致。这种设计提高了存储的弹性和可扩展性。

每个 RW 节点有其私有的重做日志（Redo Log），记录该节点的写操作。这些重做日志是相互独立的，且日志中的修改仅对应各自的租户数据。这一设计避免了写入日志时的冲突，并允许不同 RW 节点并行处理事务。在 RW 节点发生故障时，其他 RW 节点可以通过重做日志来恢复故障节点的事务状态。重做日志可以并行回放，因此即使一个节点失败，系统中的其他节点也能快速接管其事务处理，确保系统的高可用性。

所有 RW 节点共享一个全局数据字典（Global Data Dictionary），这意味着所有节点对于数据库的元数据（如表结构、索引信息等）是一致的。数据字典由一个特定的 RW 节点（主节点）管理，负责对数据字典的修改操作，而其他 RW 节点只保持只读缓存。当需要进行 DDL（数据定义语言）操作时，相关 RW 节点需要获取独占的元数据锁（MDL），阻止其他节点对该表进行修改，确保数据一致性。

为了实现系统的弹性扩展，租户可以在不同的 RW 节点之间迁移。当某个租户的资源需求激增时，可以将该租户迁移到另一个较空闲的 RW 节点，以缓解资源瓶颈。这一过程被称为租户迁移 (Tenant Transfer)。

迁移过程具体为首先，代理服务或计算节点（Proxy/CN）会暂停新事务请求，并等待当前 RW 节点完成所有正在进行的事务。然后，源 RW 节点会将租户的脏页刷新到共享存储，并更新系统表中的租户绑定信息。最后，目标 RW 节点接管租户的数据处理。整个迁移过程对应用程序透明，用户不会察觉到数据库实例的改变，仅会经历短暂的阻塞。

多租户架构使系统具有如下优势：

1. 高弹性：通过支持多个 RW 节点和租户动态迁移，PolarDB-X 可以快速扩展写入能力，确保系统能够应对突发的流量增长。例如，当某个租户的访问量突然增加时，可以将其迁移到一个新的 RW 节点来平衡负载。

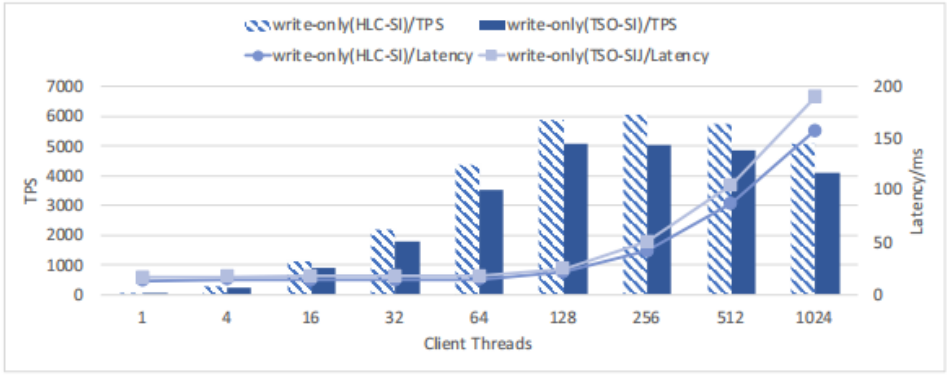
2. 写操作并行处理：由于每个租户都绑定到独立的 RW 节点，多个 RW 节点可以并行处理不同租户的写操作，从而提高系统的整体吞吐量。

3. 快速恢复和高可用性：通过共享的重做日志和存储系统，任何 RW 节点的故障都能被其他节点快速恢复，从而保持系统的高可用性和数据一致性。

4. 租户隔离：每个租户的数据完全隔离，不会发生跨租户的事务冲突，确保数据的安全性和一致性。

4 系统评估

4.1 跨数据中心事务性能评估



(a) Sysbench Write-Only Transactions

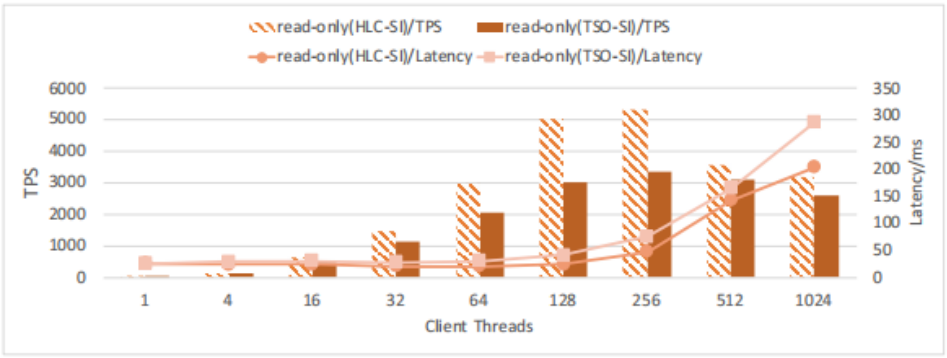


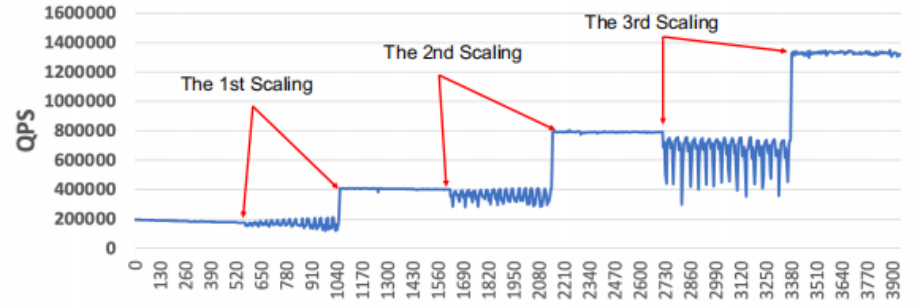
Fig. 4 跨数据中心事务性能评估

该实验比较了 HLC-SI 和传统 TSO-SI 在跨数据中心（DC）环境中的性能表现。实验使用 Sysbench 的读写事务负载，通过部署在三个数据中心的 PolarDB-X 集群，测量读写事务的吞吐量和延迟。

图 4(a) 展示了写事务中 HLC-SI 和 TSO-SI 的吞吐量和延迟对比。HLC-SI 的事务延迟更低，且在高并发情况下，吞吐量提升了 19%。图 4(b) 展示了读事务的对比，HLC-SI 同样表现出更好的事务延迟，随着并发线程的增加，吞吐量表现明显优于 TSO-SI。HLC-SI 在跨数据中心的分布式环境下，通过减少时钟同步带来的延迟，显著提高了系统的事务性能。

4.2 弹性扩展能力测试

(a) Scale Using Fast Tenant Migration Mechanism Of PolarDB-MT



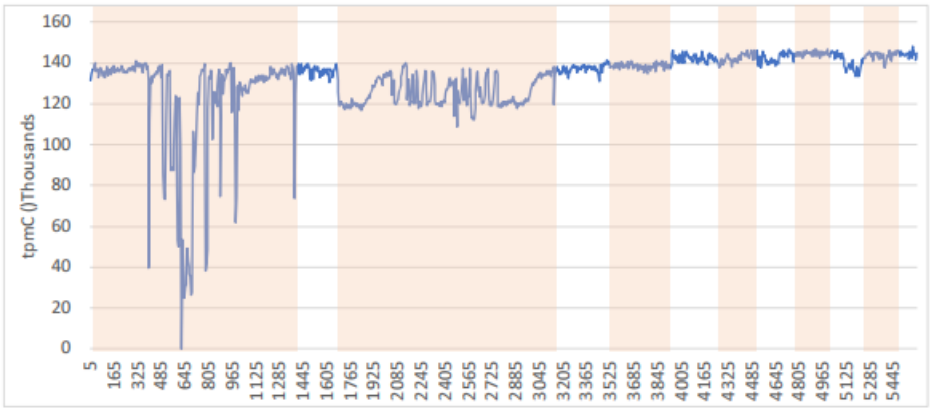
(b) Scale By Copying Data Between Source And Target

Fig. 5 弹性扩展能力测试

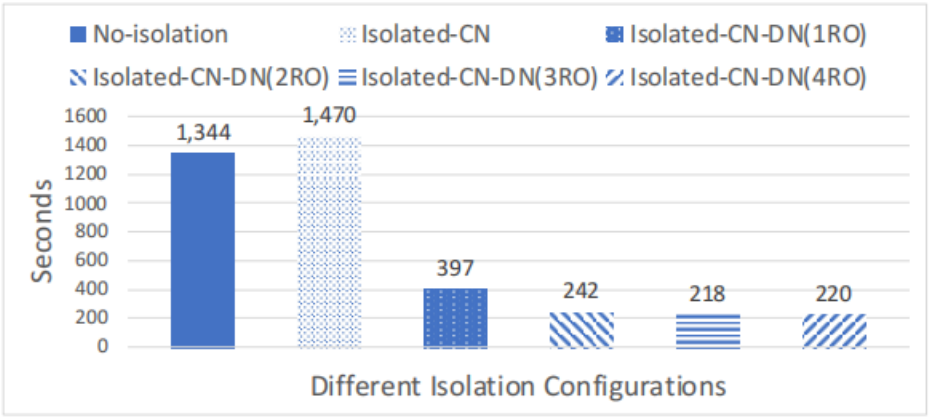
通过对 PolarDB-X 集群的动态扩展能力进行测试，实验比较了两种扩展方法的性能：快速租户迁移和传统数据传输方法。扩展期间运行 Sysbench 的读写事务负载，并测量扩展过程的耗时和吞吐量提升。

图 5(a) 显示，使用快速租户迁移机制时，三次扩展的时间分别为 4.2 秒、4.5 秒和 4.6 秒，吞吐量提升了 113%、94% 和 68%。图 5(b) 表明，使用传统的数据传输方法扩展集群耗时更长，每次扩展耗时分别为 489 秒、527 秒和 660 秒，比快速租户迁移慢了 116-143 倍。快速租户迁移机制大幅缩短了扩展时间，在应对流量激增的场景下具有明显优势。

4.3 HTAP 混合负载测试



(a) Performance variation of TPC-C while TPC-H runs six times



(b) Latency of each run of TPC-H

Fig. 6 HTAP 混合负载测试

混合工作负载（OLTP 和 OLAP），同时运行 TPC-C（事务处理）和 TPC-H（分析查询）负载，评估 PolarDB-X 在 HTAP 模式下的表现。重点测试了资源隔离、只读节点（RO Node）的使用对 OLTP 和 OLAP 的影响。

图 6(a) 显示，在没有资源隔离的情况下，TPC-C 的事务处理性能有显著抖动（最低跌至 57K tpmC），OLTP 任务受到 OLAP 的强烈干扰。当开启资源隔离后，OLTP 性能明显稳定，抖动减少。图 6(b) 显示，通过增加 RO 节点处理 OLAP 查询，TPC-H 的查询延迟逐渐下降。例如，使用两个 RO 节点时，OLAP 查询延迟下降了 39%，使用三个 RO 节点时进一步下降了 10%。资源隔离和通过增加 RO 节点处理 OLAP 查询，有效减轻了 OLTP 工作负载的干扰，提高了混合负载环境下系统的整体性能。

4.4 MPP 和内存列存索引测试

Fig. 7 MPP 和内存列存索引测试

图 7展示了 PolarDB-X 在运行 TPC-H 查询时，不同存储和执行引擎（行存储、列存储、并行处理引擎）对查询性能的影响。图中横轴为 TPC-H 查询的编号，纵轴为查询的执行时间，使用了对数刻度来展示性能的提升。列存储在处理大规模数据扫描和复杂查询（如聚合、过滤操作）时效率更高，因为列存储优化了 IO 和计算成本。使用列存储的内存列存索引显著提升了部分查询的性能，尤其是需要大量扫描和聚合的查询，如 Q6、Q14，这些查询的性能提升最高可达 1828%。

并行执行引擎（MPP）通过在多个节点之间并行处理查询任务，进一步提升了查询性能。对比使用单节点的行存储，MPP 能将查询分解为多个子任务并在多个计算节点上同时运行，因此可以显著加速查询。对于 Q9 等涉及多个表的复杂查询，使用 MPP 的性能提升超过了 263%，展现出良好的可扩展性。

5 小结

PolarDB-X 在架构设计上延续了共享存储数据库系统（如 Amazon Aurora）的思路，但通过引入三层架构和多租户设计，解决了 Aurora 等系统在跨数据中心事务和弹性扩展方面的局限性。

相比 TiDB 等分布式数据库系统，PolarDB-X 采用去中心化的时间戳服务（HLC-SI），有效避免了 TSO 的瓶颈问题。同时，PolarDB-X 在 HTAP 支持上通过引入 MPP 执行引擎和内存列存储索引，显著提升了混合工作负载的性能，而其他系统（如 Spanner）则未充分优化这一点。

未来，PolarDB-X 可以进一步优化租户迁移的技术，特别是进一步减少事务暂停时间。此外，可以借鉴其他数据库系统（如 Zephyr 和 Albatross）的在线迁移技术，提升数据库在高负载下的可用性和性能表现。

三 PolarDB-IMCI：阿里巴巴的云原生 HTAP 数据库[2]

1 背景

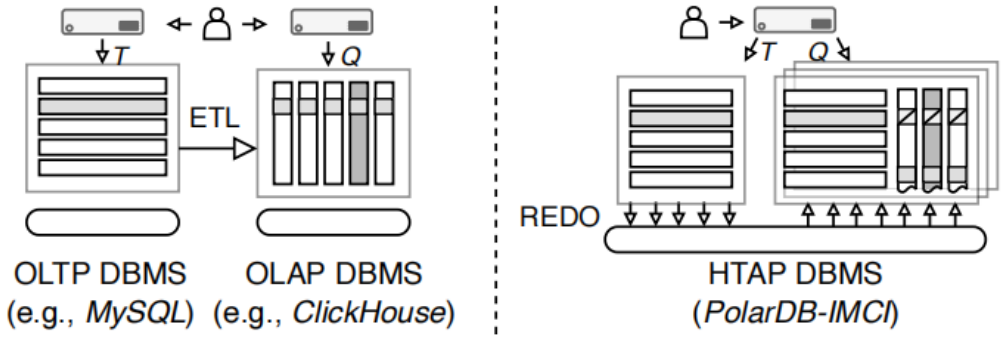


Fig. 8 ETL 与 PolarDB-IMCI 的比较

随着云原生数据库的兴起，企业逐渐向云迁移以获取更高的资源弹性和成本效益。然而，传统的 OLTP（在线事务处理）和 OLAP（在线分析处理）数据库通常被独立部署，各自专注于事务处理或分析处理。这样做的一个显著问题是需要额外的数据同步（如 ETL 流程，图 8所示），不仅导致数据新鲜度降低，还增加了复杂性和成本。文章提到，在阿里巴巴的实际客户中，近 30% 的 PolarDB 用户需要将数据同步到独立的数据仓库进行分析。这一现象反映了市场上对 HTAP（混合事务和分析处理）数据库的强烈需求。

图 8中显示了传统方案如何通过 ETL 将 OLTP 数据同步至 OLAP 数据库，以及 PolarDB-IMCI 如何通过集成的 HTAP 架构实现数据同步的消除。这解决了许多企业面临的复杂同步问题，为构建统一的数据处理平台提供了强大支持。

2 系统架构

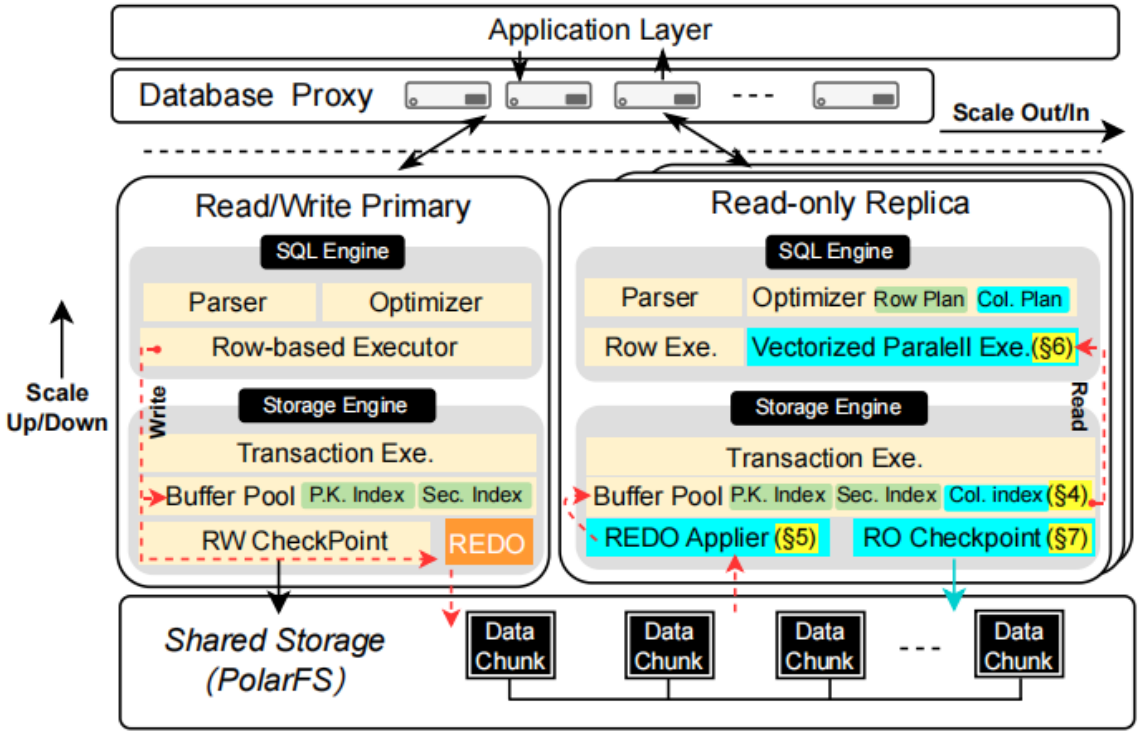


Fig. 9 PolarDB-IMCI 的云原生架构

PolarDB-IMCI 的系统架构。该系统采用了典型的计算与存储分离设计，这种设计提供了高度的资源弹性，允许计算节点根据负载变化快速扩展或缩减，而无需移动数据。图 9展示了 PolarDB-IMCI 的云原生架构，包括计算层的主节点（处理读写请求）和多个只读副本节点（处理分析查询），以及底层的 PolarFS 共享存储系统。

在这个架构下，读写节点（RW）处理 OLTP 事务，而只读节点（RO）主要负责处理 OLAP 查询。这种设计通过将分析查询负载转移至 RO 节点，确保了 OLTP 查询的响应速度不会受到影响。同时，RO 节点上的列存储设计进一步提高了 OLAP 查询性能，存储和查询效率得到显著提升。

3 关键技术

3.1 列索引存储与优化

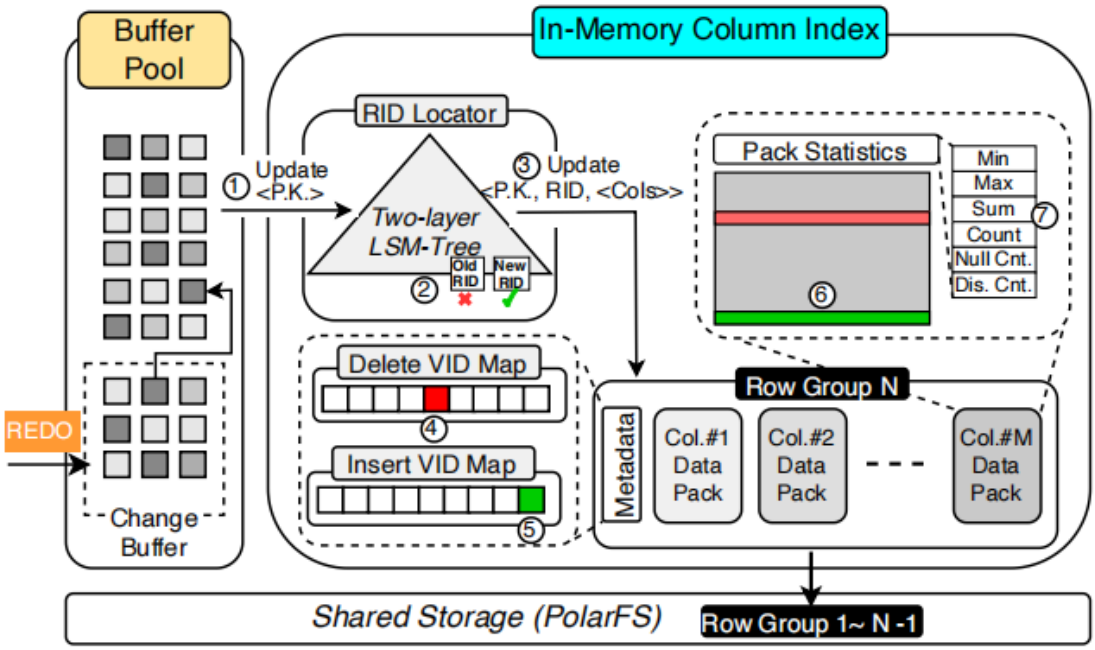


Fig. 10 IMCI 存储的数据更新流程

图 10展示了 PolarDB-IMCI 如何在 IMCI（内存列索引）存储中进行数据更新。该图通过编号 1 到 7 的步骤，说明了数据更新的全过程：

插入数据：首先，系统会为插入的行分配一个空的 RID（行 ID）。更新定位器：接着，利用主键将新分配的 RID 更新到定位器中（即两层 LSM 树中）。写入数据：系统将数据写入列数据包（Data Pack）中的空槽。记录版本 ID（VID）：插入的 VID 记录了该事务提交的时间戳。删除操作：删除时通过定位器找到记录的物理位置，并设置相应的删除 VID。更新操作：更新是通过先删除再插入实现的，即新版本的记录会被追加到部分包中，而旧版本将被逻辑删除。压缩与重排：数据包在达到最大容量后会被压缩，而删除后的“空洞”会被系统定期重排以提升扫描性能。

3.2 更新传播机制与性能优化

更新传播是 HTAP 系统中非常重要的功能，它影响到 OLAP 查询的数据新鲜度。PolarDB-IMCI 通过提前提交日志发送（CALS）和两阶段无冲突并行重放（2P-COFFER）机制，解决了传统 HTAP 系统中更新延迟高和对 OLTP 负载影响大的问题。

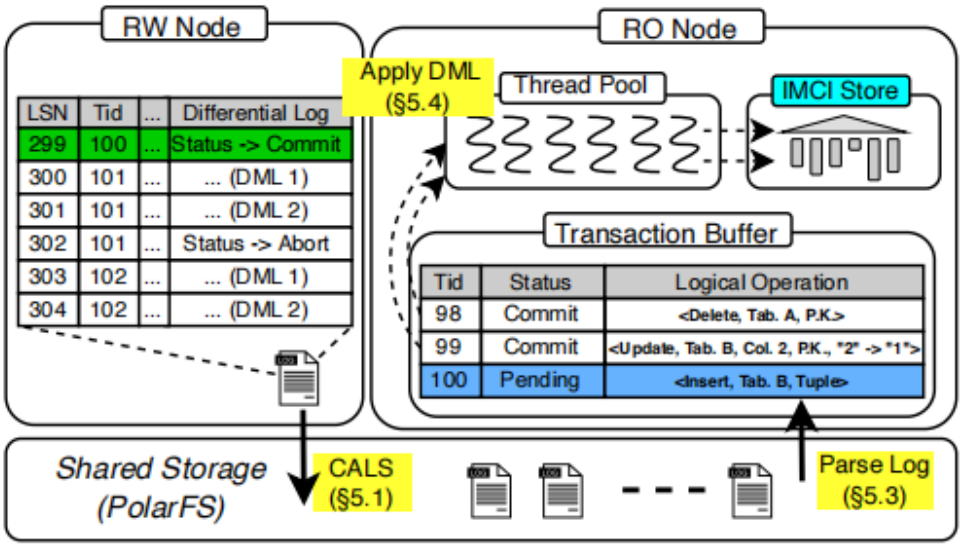


Fig. 11 REDO 日志传输的概览

图 11展示了 PolarDB-IMCI 如何通过 REDO 日志进行更新传播。其关键步骤如下：

写入日志：在主节点（RW 节点）执行事务时，每个日志条目都会被分配一个日志序列号（LSN），这些日志条目包括 DML 操作（如插入、删除、更新）和事务的提交或回滚信息。广播 LSN：主节点将最新的 LSN 广播给只读节点（RO 节点）。RO 节点通过读取这些日志来执行相应的操作。CALS 机制：PolarDB-IMCI 采用了提前日志传输机制（CALS），可以在事务提交之前就将日志传输给 RO 节点，从而减少数据更新的延迟。事务缓冲区：DML 操作被解析为逻辑操作后，存储在事务缓冲区中，并等待 RO 节点进一步处理。

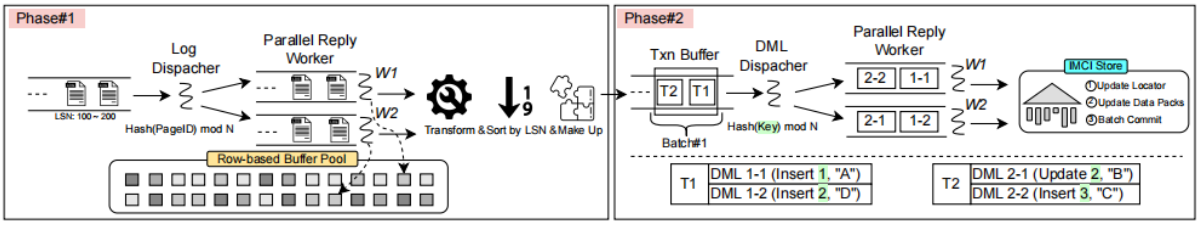


Fig. 12 PolarDB-IMCI 的两阶段无冲突并行重放（2P-COFFER）机制

2P-COFFER 机制（图 12）将日志解析和 DML 应用分为两个阶段：首先将 REDO 日志解析为物理操作，再将其转换为逻辑 DML 语句。这种机制允许多线程并行处理不同的数据页和行，实现了无冲突的并行重放，显著提高了数据同步效率。

3.3 分析查询处理与优化

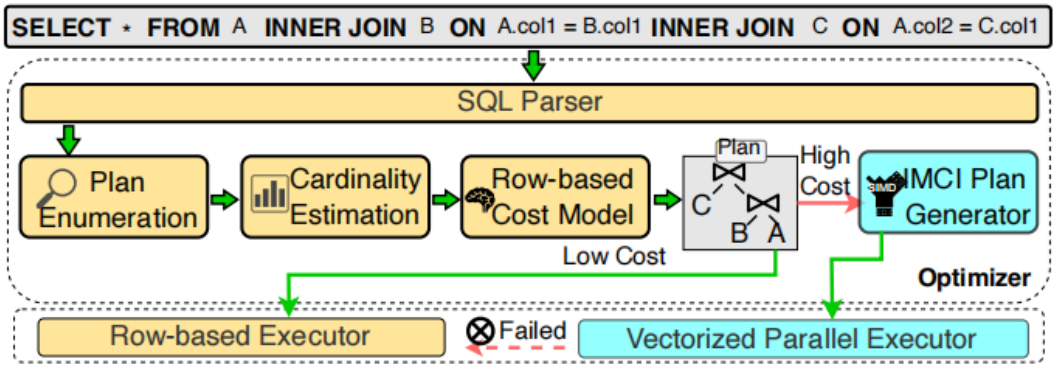


Fig. 13 PolarDB-IMCI 优化器的工作流程

图 13展示了 PolarDB-IMCI 系统中优化器的工作流程，具体描述了查询如何在系统中进行路由和执行引擎的选择。PolarDB-IMCI 的优化器采用了双层策略，分别是在节点之间和节点内部进行路由。

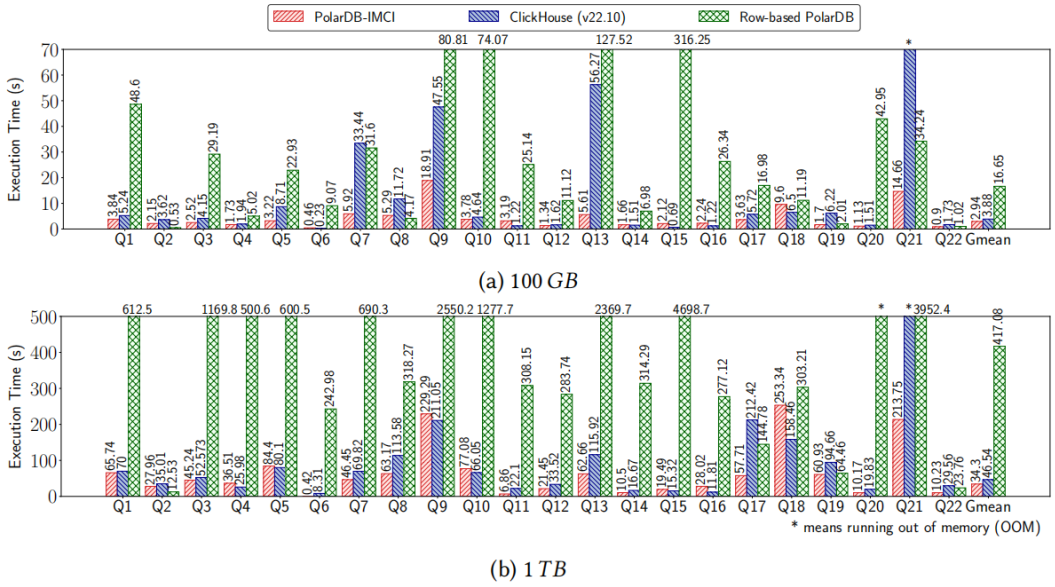
PolarDB-IMCI 的代理层提供了统一的 SQL 接口，所有应用请求（包括事务性和分析性查询）都可以通过该接口提交。代理层根据请求的类型，将读写请求（如事务处理）路由到读写（RW）节点，而只读查询（如分析查询）则路由到只读（RO）节点。对于多个 RO 节点，代理层会根据当前会话数量平衡流量。

如图 13所示，PolarDB-IMCI 在每个 RO 节点内部实现了两个执行引擎：一个用于点查询的行存储执行引擎，另一个用于分析查询的列存储执行引擎。PolarDB-IMCI 的优化器通过基于行存储的成本估计来选择合适的执行引擎。优化器首先生成一个行存储执行计划（通常成本较低）。如果行存储计划的成本超出某个阈值（即高成本），则生成列存储计划，并在列存储引擎上执行。这种优化策略可以充分利用 PolarDB-IMCI 的行列混合存储架构，确保每个查询都选择最合适的执行引擎，从而在性能和资源利用之间取得平衡。

IMCI 计划生成：与直接构建列存储执行计划不同，PolarDB-IMCI 通过从行存储执行计划转换生成列存储执行计划。这一转换过程如图 8 所示。通过这种方式，列存储计划可以保留原有查询的所有行为特性，并确保在表达式计算时能充分利用 SIMD（单指令多数据）优化。此外，列存储计划还能够重用行存储计划中的错误检测机制，确保在不同执行引擎间的一致性。图 8 中的整个流程展示了 PolarDB-IMCI 系统中查询优化器如何根据查询的类型和成本动态选择合适的执行引擎，实现高效的查询执行。

4 系统评估

4.1 TPC-H 基准测试评估 OLAP 性能



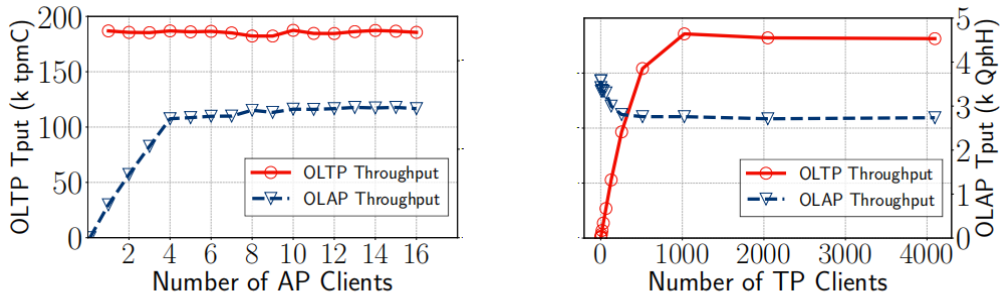
(a) 100 *GB*

(b) 1 *TB*

Fig. 14 TPC-H 基准测试评估 OLAP 性能

通过 TPC-H 基准测试，评估了 PolarDB-IMCI 在 OLAP（在线分析处理）查询中的性能。图 14展示了 PolarDB-IMCI 与行存储 PolarDB 以及 ClickHouse 的对比。在 100GB 数据集下，PolarDB-IMCI 在几何平均值上比行存储 PolarDB 的性能提升了 5.56 倍，而在 1TB 数据集下，性能提升达到了 12.15 倍。其中，针对扫描密集型查询的加速效果最为显著（例如 Q10 和 Q15）。与 ClickHouse 相比，PolarDB-IMCI 在大多数查询中也表现出色，证明了其在处理大规模 OLAP 工作负载时的优越性。

4.2 混合负载测试



(a) Perf. isolation of OLTP

(b) Perf. isolation of OLAP

Fig. 15 混合负载测试 2

图 15评估了 PolarDB-IMCI 在混合负载（HTAP）场景下的表现。通过 CH-benCHmark 进行的实验显示，PolarDB-IMCI 可以同时处理高并发的 OLTP（在线事务处理）和 OLAP 查询。在 512 个 OLTP 客户端饱和系统的条件下，OLTP 吞吐量达到 186,890 tpmC（新订单事务/分钟），同时 OLAP 查询也能以

2916 QphH（每小时 TPC-H 查询）运行。此实验验证了 PolarDB-IMCI 在 OLTP 和 OLAP 之间的资源隔离效果，即便在 OLTP 负载增加的情况下，OLAP 性能仅下降了不到 20%。

4.3 更新传播和 OLTP 性能影响

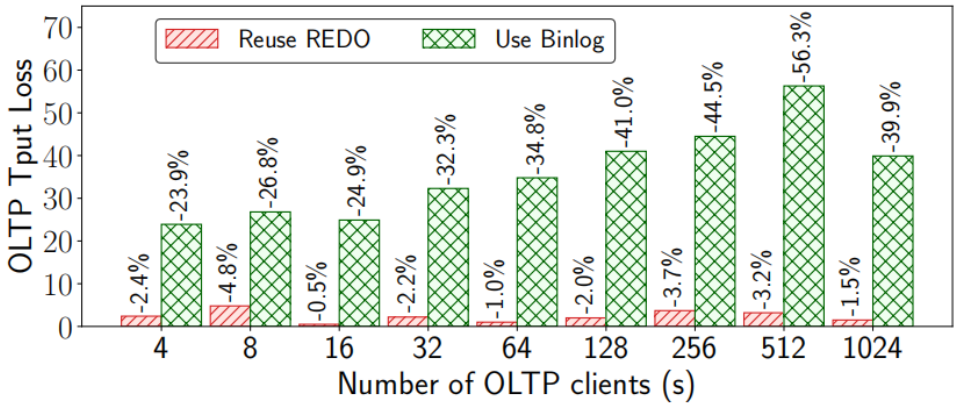


Fig. 16 更新传播和 OLTP 性能影响

图 16展示了 PolarDB-IMCI 在处理 OLTP 负载时对系统性能的影响。实验采用 sysbench 写操作负载进行，结果显示，与使用 Binlog 相比，IMCI 通过重用 REDO 日志进行更新传播，对 OLTP 性能的影响极小。启用 IMCI 的情况下，系统吞吐量仅下降了不到 5%，而使用 Binlog 的系统性能下降则更为明显。

4.4 资源弹性评估

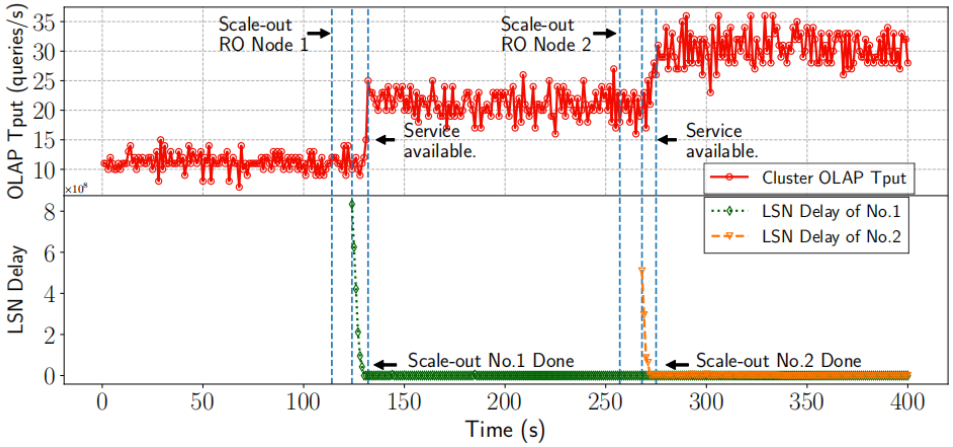


Fig. 17 资源弹性评估

为了测试 PolarDB-IMCI 的资源弹性，实验通过 sysbench 插入负载和 TPC-H Q6 查询来评估系统的横向扩展能力。图 17显示，在加入新的只读（RO）节点后，系统在数秒内能够完成节点扩展，并迅速恢复 OLAP 服务。新节点在加入后 9 秒内就能赶上最新的数据更新，展示了 PolarDB-IMCI 快速扩展的能力和弹性。

4.5 生产环境中的性能评估

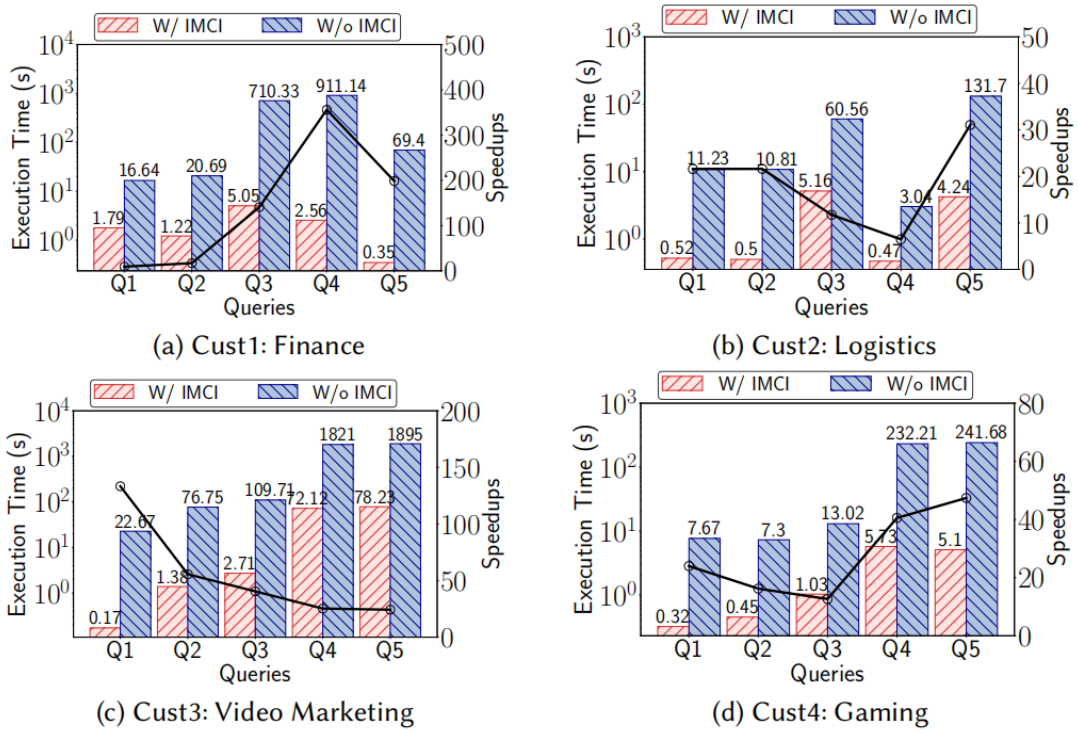


Fig. 18 生产环境中的性能评估

在图 18中，实验展示了 PolarDB-IMCI 在金融、物流、视频营销和在线游戏等实际生产环境中的表现。针对复杂的查询，尤其是涉及多表连接的查询，PolarDB-IMCI 显示了显著的加速效果。通过对比行存储 PolarDB，实验结果表明，PolarDB-IMCI 可以为这些复杂查询带来数量级的性能提升。例如，对于物流和游戏行业的查询，IMCI 在某些 SQL 查询上的加速比高达 100 倍以上。

5 小结

PolarDB-IMCI 通过创新的设计，成功实现了云原生 HTAP 系统的多个关键目标。其内存列索引和日志重放机制为 OLAP 查询提供了显著的性能提升，同时对 OLTP 负载影响最小。该系统的资源弹性和数据新鲜度进一步验证了其在实际生产环境中的可行性和优势。

四 PolarDB-MP：基于分离共享内存的多主云原生数据库[3]

1 背景

随着云计算技术的飞速发展，云原生数据库的需求大幅增加。传统的主从架构云数据库，如 AWS Aurora、Azure Hyperscale、以及阿里巴巴的 PolarDB，通常依赖单个主节点处理写请求。这种架构在面对写密集型工作负载时，存在扩展性不足、单点故障风险和主节点故障恢复过程中的短暂停机等问题。为了克服这些限制，近年来提出了多主架构的云原生数据库。这些多主架构主要分为两类：共享无关架构和共享存储架构。共享无关架构（如 Spanner、DynamoDB、TiDB 等）尽管在可扩展性上具有优势，但在处理跨分区事务时需要依赖复杂的分布式事务机制，导致较高的性能开销。另一方面，共享存储架构（如 IBM pureScale 和

Oracle RAC）虽然避免了跨节点分布式事务的部分开销，但在高冲突场景下依然面临性能瓶颈，尤其是昂贵的分布式锁管理和高网络开销问题。因此，设计一个高效的多主云原生数据库成为一个关键挑战。本文提出的 PolarDB-MP 通过结合分离的共享内存和共享存储，旨在解决这些传统架构的性能瓶颈问题。

2 架构设计

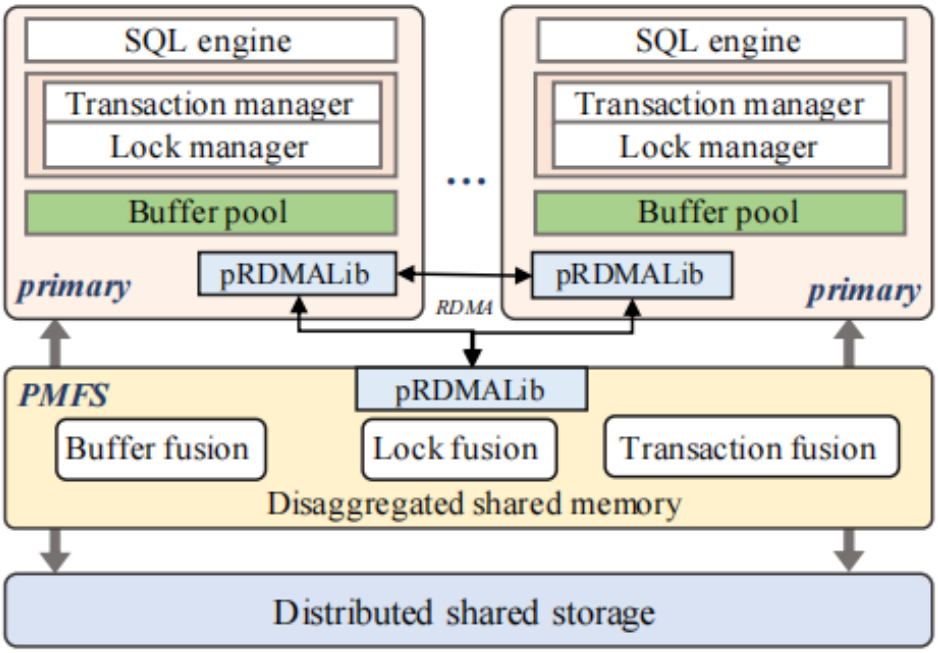


Fig. 19 PolarDB-MP 的架构

PolarDB-MP 的设计基于分离共享内存架构，允许多个主节点平等访问共享的存储和内存资源，从而减少跨节点分布式事务的开销。图 19展示了 PolarDB-MP 的总体架构。与传统的共享存储系统不同，PolarDB-MP 通过引入 Polar 多主融合服务器（PMFS）来管理全局事务、缓存一致性以及跨节点的锁控制。PMFS 由三个主要部分组成：

事务融合：管理多节点之间的事务可见性和顺序，确保事务的一致性。PolarDB-MP 通过时间戳 Oracle（TSO）来对事务进行全局排序，并利用共享内存跨节点访问本地事务数据。该设计减少了跨节点事务协调的延迟，提高了系统整体性能。

缓存融合：实现分布式缓存池（DBP），允许不同节点快速访问和更新数据页。节点通过 RDMA 将修改后的数据推送到 DBP，其他节点可以通过共享内存访问这些更新的数据页，从而实现缓存一致性。

锁融合：管理跨节点的数据页和行级锁，确保多节点同时访问数据时的物理数据一致性和事务一致性。PolarDB-MP 的设计重点在于利用 RDMA 技术来加速节点间的通信，并通过分离的共享内存实现低延迟、高吞吐量的事务处理。与现有的多主架构相比，PolarDB-MP 在系统设计中更强调全局资源的高效管理和协调，减少了数据同步和锁管理带来的性能损失。

3 系统评估

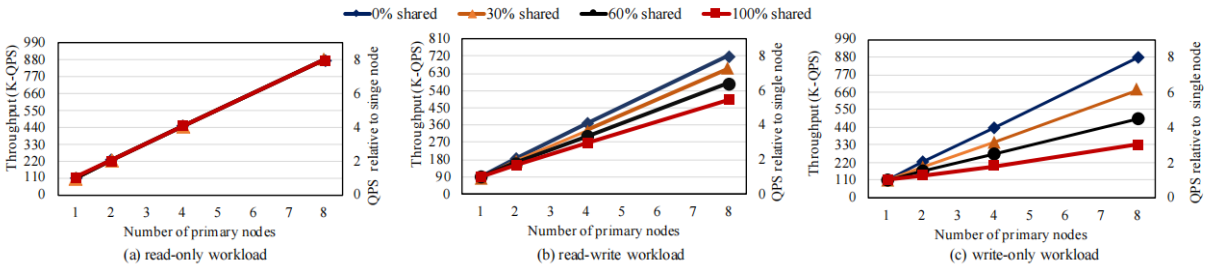


Fig. 20 SysBench 测试

在性能评估方面，PolarDB-MP 展现了出色的性能和扩展能力。文章中的实验结果表明，PolarDB-MP 在不同工作负载下的性能均优于现有的云原生多主数据库架构。图 20中显示了 PolarDB-MP 在 SysBench 读写操作下的性能，系统在 8 节点的集群配置下表现出接近线性的扩展性。当数据完全共享时，8 节点集群在读写混合工作负载下的吞吐量提升了 5.4 倍，而在纯写操作的负载下吞吐量提升了 3 倍。

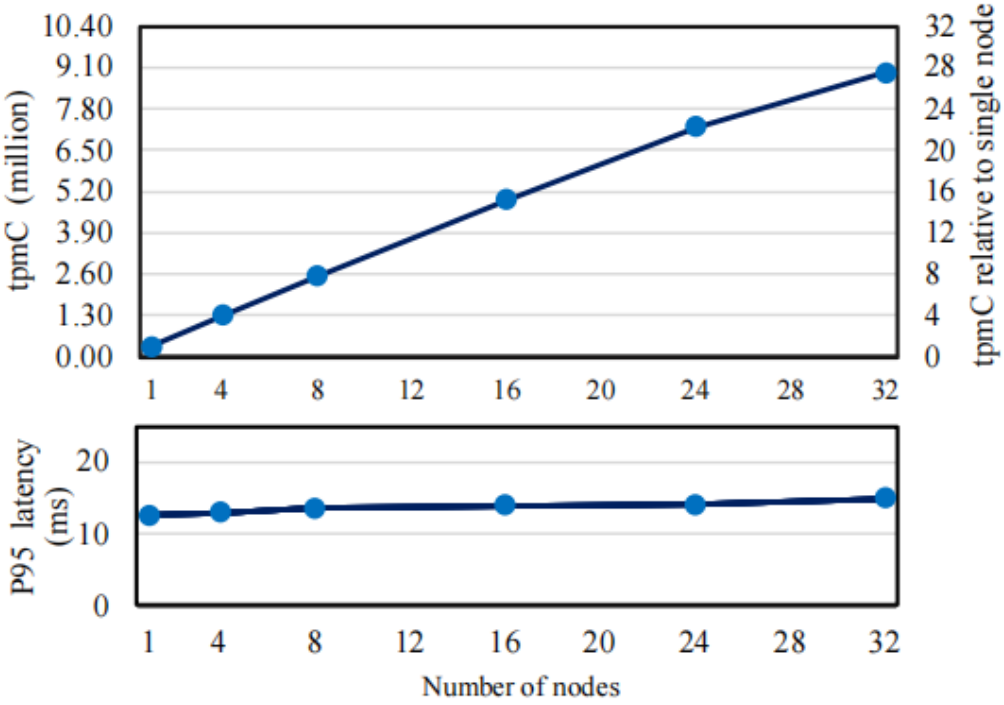


Fig. 21 大规模集群中的 TPC-C 性能

在 TPC-C 基准测试中，PolarDB-MP 的扩展性同样出色。图 21显示，系统在 32 节点集群下达到了 9.1 百万 tpmC 的吞吐量，是单节点性能的 28 倍。即使在数据共享较少的场景下，PolarDB-MP 依然能够维持良好的性能和低延迟。

4 小结

PolarDB-MP 是一种基于分离共享内存的多主云原生数据库，能够有效提升写操作的吞吐量，并在高冲突场景下保持优异的性能。通过分离共享存储和内存架构，PolarDB-MP 解决了传统多主数据库在分布式事务上的性能瓶颈问题。同时，PolarDB-MP 的 PMFS 通过优化全局事务协调、缓存一致性和跨节点锁控制，显著提高了系统的扩展性和容错性。实验结果表明，PolarDB-MP 在各种负载下的性能均优于现有的主流多主云原生数据库，如 Aurora-MM 和 Taurus-MM，具备广阔的应用前景。

五 Galaxybase: 高性能原生分布式 HTAP 图数据库[4]

1 背景

近年来，随着金融、制造、政府等行业对处理大规模图数据需求的不断增长，图数据库在图数据管理中的作用日益重要。传统的关系型数据库由于数据存储结构和查询模式的限制，在处理图形数据时往往表现出效率低下，尤其在进行复杂的图遍历和多跳查询时。为了应对这些挑战，图数据库利用顶点和边的模型对数据进行表示，提供了更灵活的方式来处理数据之间的复杂关系。这使得图数据库在社交网络分析、金融欺诈检测和知识图谱等领域具有独特的优势。然而，当前大部分图数据库在处理图查询和事务时依旧面临性能瓶颈，特别是在面对混合事务/分析处理（HTAP）场景时。为了解决这些问题，本文提出了 Galaxybase，一种高性能、原生的分布式图数据库，旨在为 HTAP 工作负载提供高效的解决方案。

Galaxybase 是为解决图数据库在处理大规模、高并发和复杂查询任务中的性能挑战而设计的。其核心理念是通过优化存储结构和分布式事务处理机制，兼顾 OLTP（在线事务处理）和 OLAP（在线分析处理）工作负载，实现对大规模图数据的高效管理。该系统不仅能够在保持数据一致性的同时处理并发事务，还能在分布式环境中通过高效的查询优化机制实现图遍历和复杂分析的快速响应。

2 系统架构

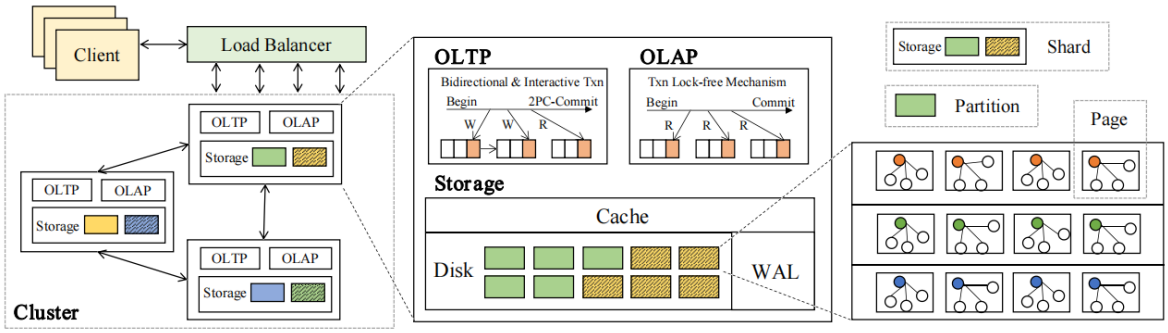


Fig. 22 Galaxybase 的系统架构

Galaxybase 的系统架构如图 22 所示，由 HTAP 计算层和存储层组成，并采用分布式集群的方式进行部署。系统中的每个服务器负责处理来自客户端的请求，通过负载均衡机制将请求分配到不同的服务器进行处理。计算层主要分为 OLTP 模块和 OLAP 模块，分别处理事务性查询和分析性查询；存储层则由缓存模块和磁盘存储模块组成，用于快速的数据检索和持久化存储。

1. 存储模型

Galaxybase 的存储模型采用三层架构，包括分片（Shard）、分区（Partition）和页面（Page）。在每个分区中，系统通过日志结构化邻接列表（Log-Structured Adjacency List）来存储顶点和边的数据，确保图遍历操作的高效性。为了进一步提升查询效率，Galaxybase 为每个顶点设计了边页面（Edge Page），将相邻的边按类型和方向分组存储，这样在进行图遍历时能够避免频繁的磁盘查找，减少 I/O 操作。

2. 分布式事务处理

Galaxybase 的事务处理模块为 OLTP 和 OLAP 工作负载分别设计了不同的事务处理机制。在 OLTP 场景下，系统采用双向事务处理模式，确保跨分区的边操作（如插入、删除和更新）在分布式环境中能够保持数据一致性。而在 OLAP 场景中，系统则采用无锁机制来处理只读事务，确保在执行复杂图分析任务时不会阻塞事务处理，保证高并发性能。

3 关键技术

3.1 日志结构化邻接列表（Log-Structured Adjacency List）

日志结构化邻接列表是 Galaxybase 用于存储顶点和边的主要结构。它通过将顶点及其相邻的边以邻接列表的形式存储，并利用日志结构化方式减少了随机写放大的问题。这种结构在进行批量数据读写时非常高效，特别适用于图遍历操作，因为它能够顺序地访问磁盘上的数据，避免了频繁的随机读写。图 **??** 展示了日志结构化邻接列表的存储模型。

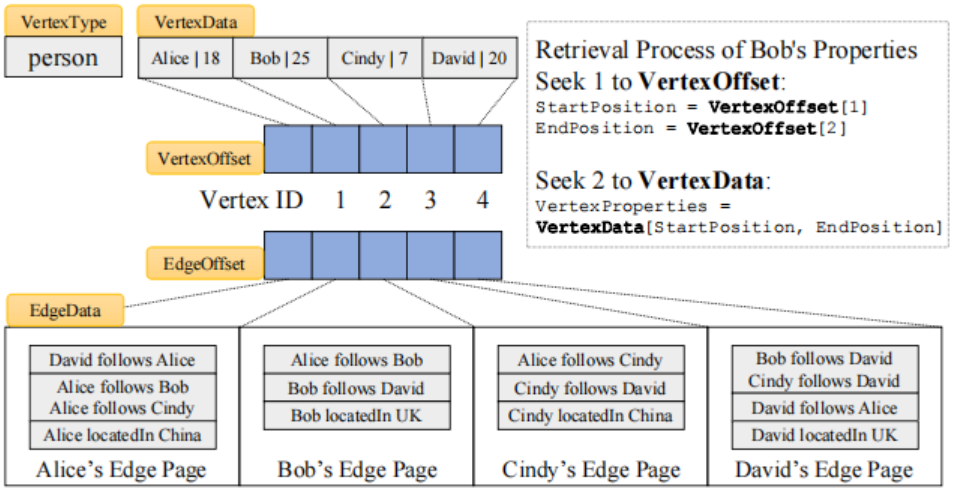


Fig. 23 日志结构化邻接列表

3.2 HTAP 事务处理机制

Galaxybase 针对 HTAP 工作负载设计了专门的事务处理机制。OLTP 事务处理侧重于高并发下的数据一致性和事务完整性，使用双向事务模式来保证跨分区边操作的一致性。而对于 OLAP 工作负载，系统采用了多版本并发控制（MVCC）和无锁读写的机制，确保在数据分析过程中不会因为 OLTP 事务而造成阻塞，保证了分析任务的连续性和数据的隔离性。

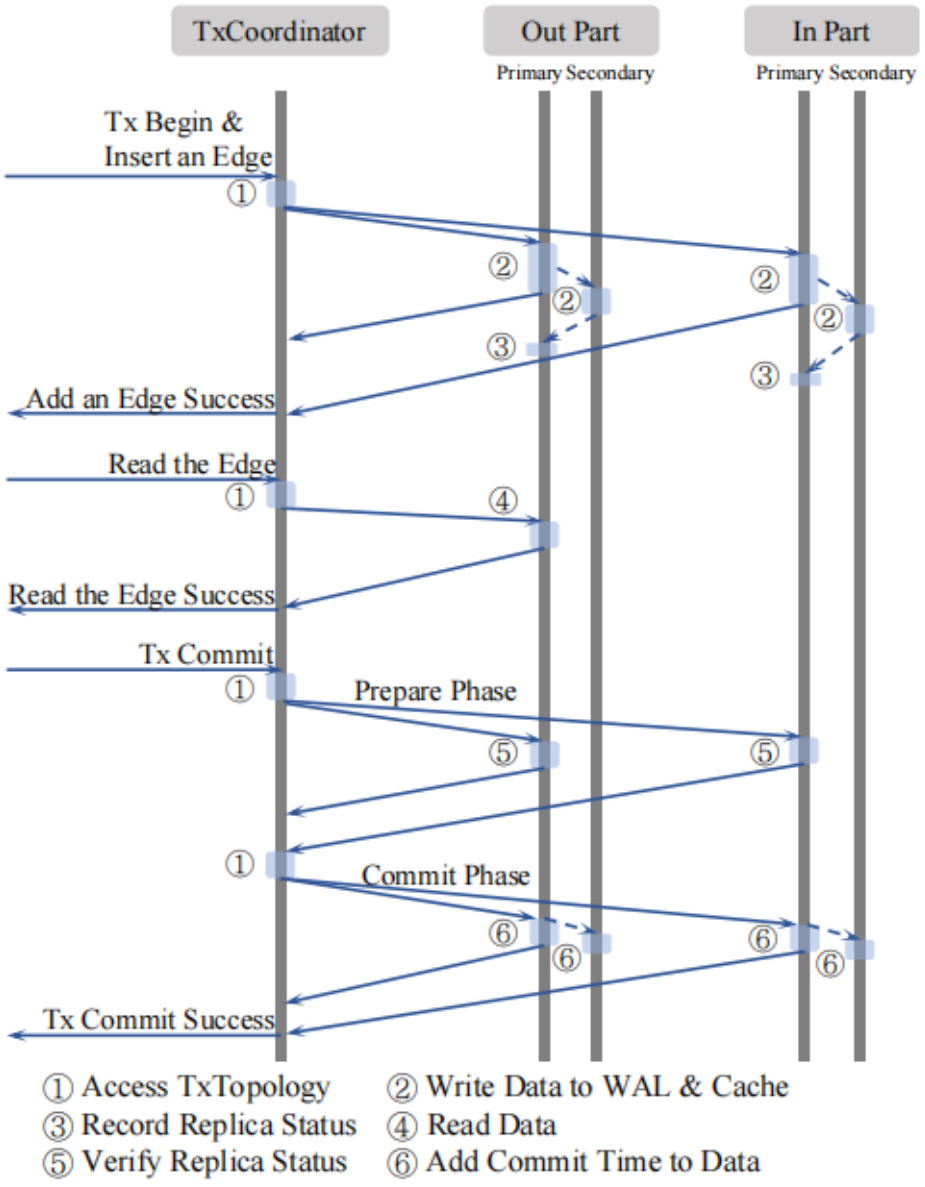


Fig. 24 HTAP 事务处理机制

4 小结

Galaxybase 作为一种高性能、原生的分布式图数据库，针对混合事务/分析处理（HTAP）场景的需求，展示了卓越的性能和灵活性。通过其独特的存储结构——日志结构化邻接列表（Log-Structured Adjacency List）和边页面（Edge Page），Galaxybase 实现了对大规模图数据的高效读写和遍历操作。此外，系统通过双向事务机制和无锁并发控制，有效地支持了 OLTP 和 OLAP 工作负载的并发执行，确保了数据一致性与查询性能。

总体而言，Galaxybase 凭借其创新的存储设计和分布式事务处理机制，成为处理大规模复杂图数据的理想解决方案，在金融、教育、能源等多个领域的实际应用中展现了强大的稳定性和高效性。

六 利用分布式 PMem 存储加速云原生数据库[5]

1 背景

随着云计算技术的普及，传统单体数据库架构已经无法满足现代大规模数据处理的需求，尤其是在面对云原生环境中数据存储和处理的挑战时。传统的关系型数据库（如 MySQL）在处理用户账户信息、订单管理、物流跟踪等业务应用时，由于数据量的快速增长，性能瓶颈逐渐显现。为了应对这些挑战，字节跳动开发了 Volcano Engine Database（veDB），采用了分布式存储架构，解决了数据存储容量不足的问题。

然而，veDB 的计算与存储分离架构也带来了新的问题，例如远程存储访问导致的高事务延迟波动和大数据读取时的高查询延迟。这些延迟问题严重影响了某些关键业务场景的性能，例如需要低延迟、高吞吐量的批量订单处理。本文提出了通过持久化内存（PMem）和远程直接内存访问（RDMA）技术来优化存储层的方案，显著降低系统的读写延迟，并提升系统的吞吐量。

2 系统架构

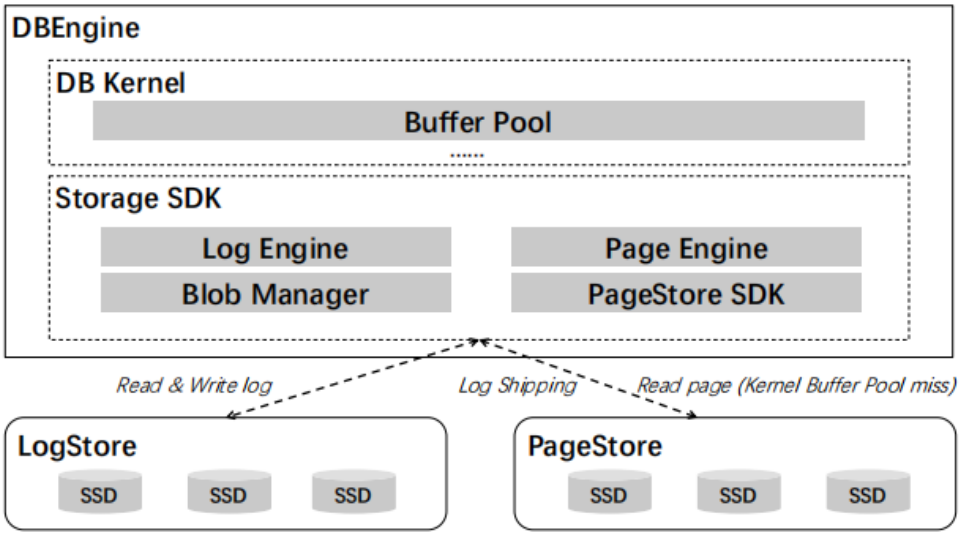


Fig. 25 veDB 的系统架构

veDB 的系统架构遵循计算与存储分离的设计模式，基于日志即数据库（Log-Is-Database）的原则。系统的主要组件包括计算层（DBEngine）和存储层（LogStore 和 PageStore）。

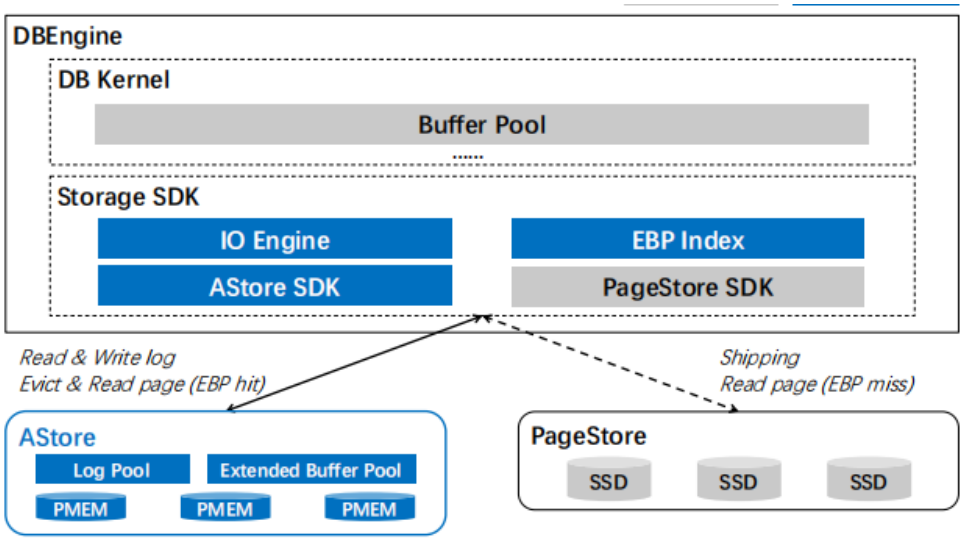


Fig. 26 引入持久化内存（PMem）并通过 RDMA 进行远程访问的新架构

DBEngine：作为系统的计算核心，负责事务处理、查询执行和数据修改。它通过写入日志（REDO Log）来记录数据库的变更。日志的持久性通过写前日志（Write-Ahead Logging, WAL）机制保证，即在事务提交前，先将日志写入 LogStore。

LogStore：负责 REDO 日志的持久化存储。日志数据以分布式方式存储，支持高可用性和故障恢复。LogStore 采用了分布式的 blob 存储系统，日志以 BlobGroup 的形式进行管理。每个 BlobGroup 由多个并发写入的 Blob 组成，能够处理大规模的并发 I/O 请求。

PageStore：用于存储数据页，并通过不断应用 REDO 日志来保持数据页的最新状态。PageStore 中的数据以段（Segment）的形式管理，段在不同可用区内复制以保证高可用性。

为了优化日志和数据读取的性能，本文提出了引入持久化内存（PMem）并通过 RDMA 进行远程访问的新架构（图 26）。新的存储系统 AStore 替代了原有的 LogStore，提供了低延迟、高吞吐的存储能力，同时通过扩展缓冲池（EBP）缓存热点数据页，进一步减少远程数据访问的延迟。

3 关键技术

1 持久化内存（PMem）

PMem 是一种介于 DRAM 和 SSD 之间的存储介质，具有接近 DRAM 的访问速度和 SSD 的持久化能力。相较于传统的 SSD 存储，PMem 在读写延迟上有显著的优势，能够在微秒级别完成存储操作。在本文的设计中，AStore 基于 PMem 构建，实现了比 SSD 更低的写入延迟和事务提交时间。此外，由于 PMem 的持久化特性，它还可以在系统故障后迅速恢复数据。

2 远程直接内存访问（RDMA）

RDMA 是一种网络技术，允许一台服务器直接访问另一台服务器的内存，而无需经过对方的操作系统和 CPU 处理。这种“单侧操作”能够绕过传统的网络栈，减少数据传输过程中的 CPU 开销，大幅提升 I/O 性能。在 AStore 中，RDMA 用于快速读写远程 PMem 存储，整个 I/O 操作路径不涉及远程存储服务器的 CPU 处理，从而实现极低的存储访问延迟（读延迟约 10 微秒，写延迟约 20 微秒）。

通过结合 PMem 和 RDMA，AStore 大大减少了事务日志的写入延迟，提高了事务处理的吞吐量，并

使得日志提交延迟的波动保持在一个狭窄的范围内，满足了高要求的业务场景需求。

3 查询下推（Query Push-Down）

为了优化大数据量的查询性能，veDB 引入了查询下推（Query Push-Down）框架，将部分计算操作（如过滤、聚合等）下推到存储层进行处理。这一设计可以有效减少计算层与存储层之间的数据传输开销，并充分利用存储层的空闲计算资源。例如，在处理大规模的分析查询时，查询下推框架能够将数据筛选、聚合等操作直接在存储层完成，减少计算层的数据处理负担。

结合 PMem 加速的 AStore，查询下推框架进一步提升了分析查询的性能，特别是当需要访问大量缓存数据页时，下推计算能够有效降低整体查询延迟。

4 扩展缓冲池（Extended Buffer Pool, EBP）

veDB 的计算与存储分离架构导致了大量的远程数据访问，增加了事务处理的延迟。为了降低远程存储的访问成本，AStore 引入了扩展缓冲池（EBP），用于缓存那些被计算层缓存驱逐的热点数据页。EBP 通过 RDMA 直接访问 PMem 中的缓存数据页，这种设计使得大规模数据的读取性能显著提升。在读取 16KB 数据页时，EBP 能够将延迟降低至约 20 微秒，这与本地 SSD 的访问延迟相当。

EBP 不仅提供了较大的缓存容量，还能够在事务处理过程中减少远程存储访问次数，从而提高系统的整体性能。此外，本文设计的 EBP 采用优先级策略，允许为关键业务数据保留更多的缓存空间，进一步优化了查询性能。

4 小结

本文介绍了一种通过持久化内存（PMem）和远程直接内存访问（RDMA）加速云原生数据库系统的方法。通过引入 AStore，veDB 的存储层能够提供更低的读写延迟和更高的事务吞吐量。实验结果表明，本文设计的系统在标准基准测试和实际业务场景中，系统吞吐量提高了 1.5 倍，延迟减少高达 20 倍。特别是在需要高并发和低延迟的业务场景下，AStore 展现出了卓越的性能提升。

此外，本文提出的扩展缓冲池（EBP）和查询下推框架能够进一步优化大数据查询的性能，特别是在分析查询场景中，显著减少了远程数据读取和传输的延迟。

七 总结与展望

文献题目	背景	核心贡献
PolarDB-X：面向云原生应用的弹性分布式关系数据库	PolarDB-X 设计于满足多数据中心部署的需求，同时通过分离计算和存储提升弹性，支持 OLTP 和 OLAP 的混合工作负载	采用三层架构（计算节点、数据库节点、存储节点），支持 HLC-SI（混合逻辑时钟快照隔离）实现跨数据中心事务一致性
PolarDB-IMCI:阿里巴巴的云原生 HTAP 数据库	PolarDB-IMCI 旨在提供同时支持 OLTP 和 OLAP 的高效解决方案，减少数据同步开销，提升数据新鲜度和资源弹性	通过列存储和透明查询路由机制实现高效的分析性能和低延迟的事务处理。设计了基于存储与计算分离的双格式存储架构，实现了 OLTP 和 OLAP 请求的资源隔离。
PolarDB-MP: 基于分离共享内存的多主云原生数据库	主要为了解决单主架构在高并发写入场景中的性能瓶颈，提出了一种多主架构，旨在提升写入扩展性和高可用性	提出 Polar Multi-Primary Fusion Server (PMFS)，利用 RDMA 实现全局事务协调、缓冲区融合和跨节点的锁管理，设计 LLSN 机制维持日志顺序并优化故障恢复流程
利用分布式持久内存加速云原生数据库	为解决分离存储架构带来的高延迟问题，提出结合持久内存（PMem）和 RDMA 的分布式存储系统，以加速数据库性能	设计了基于 PMem 和 RDMA 的 AStore 存储引擎，通过单侧 RDMA 实现低延迟的日志写入和事务处理，并支持计算推送加速查询执行
Galaxybase: 高性能原生分布式图数据库用于 HTAP	针对现有图数据库在处理大规模图数据时的性能瓶颈，尤其在 OLTP 和 OLAP 混合负载中的挑战，提出了新型分布式图数据库	采用日志结构的邻接列表和边页结构，优化了图遍历和单边查询性能，并通过双向交互事务机制支持 OLTP 和 OLAP 负载的并行处理

Table 1 云原生数据库系统文献总结对比

本报告详细探讨了当今云原生数据库系统的架构、技术创新及其在实际应用中的优势。通过分析 PolarDB-X、PolarDB-IMCI、PolarDB-MP 以及 Galaxybase 等代表性系统，展示了它们在不同应用场景下的性能和弹性。其中，PolarDB-X 的三层架构设计、HLC-SI 的去中心化事务管理机制，极大地提升了系统在跨数据中心部署时的高可用性和数据一致性。PolarDB-IMCI 则通过计算与存储分离架构，实现了 OLTP 与 OLAP 工作负载的资源隔离与高效处理，显著减少了数据同步的开销。PolarDB-MP 凭借分离共享内存的多主架构，在高并发写入场景中表现出极强的扩展性和容错能力。Galaxybase 作为分布式图数据库，成功应对了大规模图数据处理中的复杂查询与事务并发挑战，为 HTAP 工作负载提供了强有力的支持。

通过性能评估和实验数据，本文揭示了这些云原生数据库系统在处理高并发、跨数据中心和混合工作负载时的卓越表现。与此同时，报告也指出了未来云原生数据库系统可能的优化方向，如进一步减少租户迁移过程中事务的暂停时间、优化数据同步机制以应对更复杂的业务需求等。

总的来说，云原生数据库技术为现代企业提供了一套全面且高效的数据管理解决方案。其通过创新的架构设计和先进的分布式事务处理技术，解决了传统数据库无法应对的扩展性和一致性问题，尤其在面向全球化、多数据中心部署的业务场景下，展示了广阔的应用前景。未来，随着云原生技术的不断发展和成熟，这些数据库系统将在更多行业中得到广泛应用，推动企业的数字化转型和业务创新。

参 考 文 献

[1] CAO W, LI F, HUANG G, et al. Polardb-x: An elastic distributed relational database for cloud-native applications [C]//2022 IEEE 38th International Conference on Data Engineering (ICDE). IEEE, 2022: 2859-2872.

[2] WANG J, LI T, SONG H, et al. Polardb-imci: A cloud-native htap database system at alibaba[J]. Proceedings of the ACM on Management of Data, 2023, 1(2): 1-25.

[3] YANG X, ZHANG Y, CHEN H, et al. Polardb-mp: A multi-primary cloud-native database via disaggregated shared memory[C]//Companion of the 2024 International Conference on Management of Data. 2024: 295-308.

[4] TONG B, ZHOU Y, ZHANG C, et al. Galaxybase: A high performance native distributed graph database for htap [J]. Proc. VLDB Endow, 2024, 17(12): 3893-3905.

[5] SUN J, MA H, ZHANG L, et al. Accelerating cloud-native databases with distributed pmem stores[C]//2023 IEEE 39th International Conference on Data Engineering (ICDE). IEEE, 2023: 3043-3057.