Rethink the Scan in MVCC Databases

摘要

当今的许多业务场景下，混合型事务分析（Hybrid Transactional/Analytical Processing，HTAP）的工作负载越来越常见。在 HTAP 工作负载下，频繁的事务处理可能导致多版本并发控制系统（Multi-Version Concurrency Control，MVCC）中维护的版本链长度增加，严重增加分析事务的查询延迟，从而影响分析事务的吞吐量。针对版本链问题，我们复现了 vWeaver——一种基于跳表技术的版本可见性查找加速算法，将版本链查找时间控制在对数级别。虽然 vWeaver 优化了基于内存的 MVCC 系统吞吐量，但在基于磁盘的 MVCC 系统上，对旧版本的查询仍会导致冷数据的传输，从而加重磁盘的 I/O 负担。

存储内计算（In-Storage-Computing，ISC）可以将需要访问存储介质的计算逻辑卸载到存储介质中运行，从而减少主机与存储介质间的数据传输，大大减轻基于磁盘的 MVCC 系统的 I/O 负担。因此，结合版本链跳转技术，我们将版本链查找逻辑转移至可计算存储（Computational Storage Device，CSD）介质中，在减少版本链查找次数的同时，降低冷数据的传输。基于开源数据库 Bustub 框架，我们实现了原型系统 CSD_Bustub 并对其进行了简单的性能测试。

关键词： HTAP；版本链跳转；可计算存储；存储内计算

1 引言

随着现代社会大型实时分析应用的逐渐流行，关系型数据库已经难以处理高并发的事务请求。在商业层面上，当全球进入数字化时代，数字化技术渗透到各行各业，各种应用产生的海量数据是企业决策的重要依据之一，业务需要实时根据事务处理的落地数据进行客户端快速反馈，比如实时风控、交易历史明细查询、欺诈监测等等。在技术层面上，由于传统的数据仓库查询链路长，延迟大，很难满足业务快速多变的诉求，引起了混合型的事务处理和分析处理系统（Hybrid Transactional/Analytical Processing，HTAP）的快速发展。传统的事务处理和分析处理是分开进行的，事务处理主要依赖于在线事务处理系统（OnLine Transaction Processing，OLTP），分析处理则主要依赖于在线分析处理系统（Online Analytical Processing，OLAP）。HTAP 系统则是结合了这两种系统的特点，同时支持高并发、高吞吐、低延迟的事务处理和强大的分析处理。然而两种系统的融合也带来了一些挑战，HTAP 系统需要在系统性能与数据新鲜度之间做出取舍。

近年来，更新密集型的数据库应用给 HTAP 系统带来了一定挑战。由于更新密集型的数据库应用会产生大量的事务处理（Transaction Processing，TP）请求，在 MVCC 系统中，频繁的 TP 事务会不断向版本链添加记录版本，这导致了在执行分析处理（Analytical Processing，

AP）事务时搜索版本链的时间过长，增加了 AP 事务的延迟，降低系统效率。在 MySQL 上使用 TPC-CH [2] 测试的结果如图 1所示，该测试使用一个长时间运行的事务，该事务在内部连续执行 TPC-H 查询。由于 TP 事务导致链长度的增加，会严重影响长时间运行的 AP 事务，使得 AP 查询的延迟增加。

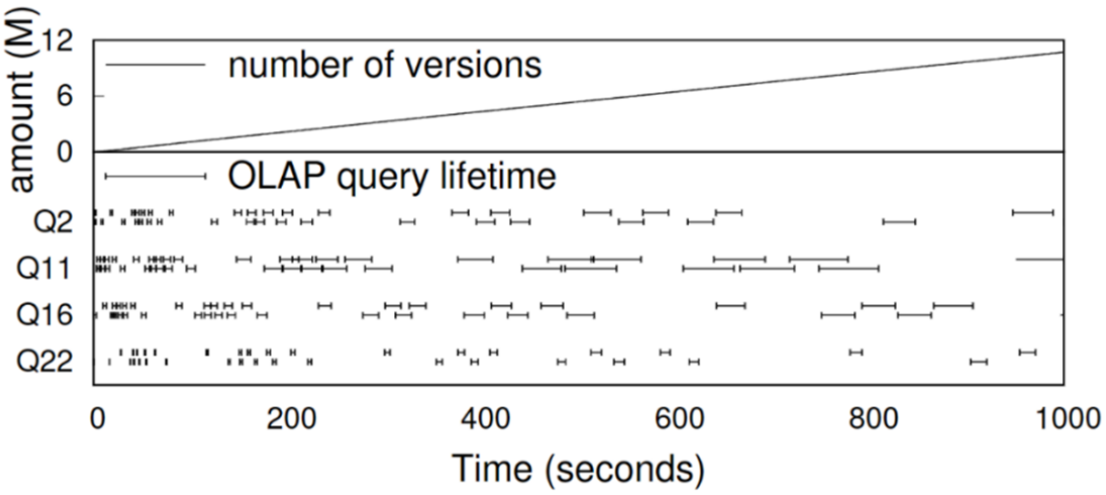

图 1. MySQL 中长时间运行的 OLAP 查询（图片源于 vWeaver [7]）

由于数据库社区将面临越来越多支持 HTAP 工作负载的需求，因此近年来学术界对研究这一问题的关注日益迫切。针对这些挑战，学术界中的一些研究探索了 HTAP 与近年热点（如可计算存储 [13]）的结合，将查询计划执行的全部或部分卸载到存储介质中进行存储内计算。因此，近数据处理（Near-Data Processing，NDP）可以有效解决冷数据传输的问题，而研究版本链跳转加速技术可以进一步加快执行效率，均对 HTAP 数据库的高效执行具有重要意义。

2 相关工作

目前，学术界存在许多加速版本链搜索的工作，主要分为三个方向。细粒度垃圾回收方法 [1,6,8] 通过采用细粒度的垃圾版本回收策略，控制版本链的长度，加快 AP 事务数据读取的效率。数据推送方法 [11] 将数据推送给 AP 事务，避免存储不必要的版本，节省版本存储空间，并且可以基于快照访问最新版本。版本链跳转技术 vWeaver [7] 则是利用类似跳表 [12] 的方法，加速点查询与范围查询，极大程度减少了版本跳转的次数。

尽管在多版本系统的版本链搜索上已经存在较多研究，但在基于磁盘的 HTAP 数据库系统中，仍无法避免大量冷数据传输的问题。随着可计算存储 [7]（Compute-Storage-Device，CSD）近年来的发展，一些研究 [9,14] 开始探索利用存储内计算（In-Storage-Computing，ISC）来提高 HTAP 系统的性能。CSD 是一种新型存储技术，它将传统的存储和计算功能融合在一起，使存储设备不仅可以存储数据，还可以进行计算操作。ISC 指在存储设备内部进行计算处理，比起将数据传输到外部处理器进行处理，其可以大大减少数据传输的开销，并且可以在存储设备内部直接对数据进行操作和分析，从而提高数据处理的效率和速度。Tobias 等人 [14] 提出了一种更新感知型的近数据处理架构，它可以在 HTAP 数据库中累积更新增量并定期同步至 CSD 中，保证了事务一致性与数据新鲜度。Kitaek Lee 等人 [9] 提出了分析卸

载引擎，提供了规范接口以支持各种不同的数据库供应商的数据格式，便于 CSD 中的加速运算。

2.1 MVCC 存储系统中的版本链加速

2.1.1 细粒度垃圾回收

Steam [1] 建立在 HyPer [5] 的 MVCC 实现基础上，在遍历版本链时及时清理不再被活动事务所需要的版本，避免了版本链过长的问题。具体而言，Steam 使用两个链表来跟踪正在运行和已提交的事务，新事务被追加到活动事务链表，当活动事务提交时，移动到已提交事务链表以保留其创建的版本，事务链表按时间戳隐式排序。在版本修剪方面，Steam 采用了一种基于区间的算法，类似于 HANA 数据库 [3]。但 Steam 不是在后台进行修剪，而是在插入事务时主动进行前台修剪，将版本链长度限制为活动事务的数量，避免了版本链过长的问题。但是，Steam 在插入新版本时立即修剪版本链的行为，可能会造成一定程度的延迟，影响查询的响应时间。

vDriver [6] 提出了一种新型版本存储形式，将行内存储与行外存储相结合。它将第一个版本与记录一起存储，同时将其他版本移至单独的版本存储空间，实现了快速恢复和无损索引结构。vDriver 提出了一个新颖的版本清理方法，精确的给出了识别过期版本的充要条件。同时，vDriver 通过版本分类将具有不同生命周期的版本分开存储，避免了长周期事务对版本回收的影响。虽然 vDriver 可以通过其版本管理架构和垃圾回收机制有效避免长期事务对 MVCC 系统中版本清理的影响，但也存在许多不足：将 vDriver 整合到现有系统中可能需要对物理页面布局进行更改，这可能需要大量的工程工作量；vDriver 可能会对现有系统的 undo 日志空间管理造成影响，这可能需要额外的工作来管理现有系统的冗余 undo 空间；vDriver 的版本清理所选择的是段时间戳而不是单个版本时间戳，可能会影响版本清理的完整性。

Diva [8] 引入了临时版本索引的概念，类似于 UNIX 的 inode 结构，将版本索引与版本数据分开存储，实现了高效的版本搜索。同时，Diva 实施了基于时间间隔的版本垃圾回收策略，通过设置预设的时间间隔，匹配间隔内的版本和事务，实现了高效识别与清理过时记录版本。其避免了昂贵的引用跟踪，且能够批量清理过期版本，提高了系统性能。虽然 Diva 在解决 MVCC 系统在 HTAP 环境下的挑战上取得了一定成功，但存在复杂性增加、性能开销增大、适用范围限制等不足之处，仍需进一步研究和改进以确保系统的有效性和适用性。

2.1.2 数据推送

Kvell+ [11] 通过引入在线交换处理（Online Commutative Processing，OLCP）模型来避免传统快照隔离中的空间放大和垃圾回收问题。OLCP 利用数据项的可交换性进行处理，使得在进行大规模数据处理时，不会出现空间放大和垃圾回收成本的问题。具体来说，当数据项更新时，旧版本的数据会被传播给可能需要的 OLCP 查询，而不是保留在存储中。这种方式有效地减少了存储空间的占用和垃圾回收的开销，提高了系统的性能和效率，但仍存在实现复杂性、性能影响、可扩展性和兼容性等方面的不足，需要进一步优化和改进。

2.1.3 版本链跳转

先前的方法主要集中在清理过期的记录版本来削减版本空间，但没有解决传统线性版本搜索算法对版本链遍历的性能限制。vWeaver [7] 通过构建高效的版本搜索结构来加速版本空间上的扫描操作。它首先构建了一个概率搜索结构，加速链内版本查找（点查询）。然后利用链间跳转结构将相邻链进行连接，实现对给定快照相邻键值版本的高效移动，加速链间版本跳转（范围查询）。然而，尽管 vWeaver 使用节俭跳表等技术大幅度减短了版本搜索路径，但在基于磁盘的 MVCC 系统上，仍面临冷数据的 I/O 传输问题。

2.2 可计算存储

当前学术界已有许多利用 CSD 加速的工作，如加速神经网络、推荐系统、OLAP 系统，本节将介绍一些近年来的代表性工作。

RecSSD [15] 是一种基于固态硬盘（SSD）的近数据处理解决方案，通过将关键的嵌入表操作的计算任务从主机 CPU 卸载到 SSD 上，减少数据通信的往返时间，并提高内部 SSD 带宽利用率。该设计针对数据中心规模的 SSD 的主要性能瓶颈进行优化，实现了高效的嵌入表操作。RecSSD 通过在实际系统中使用服务器级 CPU 和 Micron UNVMe 的驱动库来实现，减少了端到端神经网络推理的延迟，比现成的 SSD 系统提高了 4 倍的性能，并与 DRAM 内存的性能相当。因此，RecSSD 实现了高效和可扩展的数据中心神经推荐推理。

RM-SSD [13] 是一种基于 FPGA 的存储系统，旨在优化推荐系统的推理性能。它包括 RM-SSD 驱动程序和用户库，采用 C++ 运行时库，可与 Python-based 深度学习框架（如 PyTorch、Caffe2）轻松集成。RM-SSD 通过内存中的计算和存储中的计算引擎来实现推理性能的优化。实验结果显示，RM-SSD 相比基准 SSD 和最新工作，推理吞吐量提高了 20-100 倍。此外，RM-SSD 还能有效减少读取放大现象，大大提高了计算效率。具体而言，内存中的计算引擎通过元素级池化操作和特征交互将嵌入表返回的向量聚合成单个向量，然后与密集特征进行连接，最终由顶部 MLP 层处理。存储中的计算引擎则包括嵌入查找引擎和 MLP 加速引擎，通过内层分解、层间组合和内核搜索等技术来最小化 FPGA 资源消耗并最大化吞吐量。

SmartSAGE 系统 [10] 将延迟优化的软件系统与 ISP 加速的硬件架构综合起来，实现了端到端 GNN 训练时间的显著加速。它相较于基准 SSD 中心系统，平均提升了 3.5 倍（最大提升 5.0 倍）。通过智能地将数据密集的前端数据准备阶段的步骤转移到 ISP 单元，SmartSAGE 系统成功地解决了基准 SSD 中心训练系统的瓶颈，实现了显著的性能改进。

YourSQL [4] 充分利用现代固态存储设备的计算能力，实现在存储设备内部进行数据过滤，从而减少数据在存储网络和主机系统之间传输的量。通过在存储设备内部进行数据过滤，YourSQL 实现了非常早期的数据过滤，避免了将数据从磁盘移动到主机系统的不必要传输，从而提高了查询效率。同时，允许查询在主机系统和固态硬盘之间进行分布式处理，充分利用固态硬盘的计算能力来处理非复杂的查询操作。但由于 YourSQL 的设计侧重于利用存储设备的计算能力，可能在处理复杂查询或需要大规模扩展性的情况下存在一定限制。

3 本文方法

本节将详细叙述版本链跳转技术 vWeaver 的基本原理。首先在 3.1 节描述基于跳表的链内搜索结构，确保高概率的对数搜索时间复杂度；然后在 3.2 节描述如何插入相邻链间的跳转结构，实现有效的跨链移动。

3.1 链内搜索结构

版本链搜索问题指的是，找到在给定时间戳（即快照）之前创建的版本。由于版本链本质上就是一种链表，那么版本链搜索问题的本质就是在给定链表中查找指定元素。基于跳表加速技术 [12]，vWeaver 提出的链内搜索结构是一种低成本的跳表（节俭跳表）。每个节点只需要多维护一个指针 v_ridgy，就能维持搜索时间处于对数级别。该节俭跳表是自包含的，它只在插入和搜索版本时作用于这个指针，而不需要在原始数据库的版本存储中进行任何修改。本质上，vWeaver 使用现有的版本链，在它之上动态构建快捷指针，形成了节俭跳表。

3.1.1 基本结构

节俭跳表在空间开销方面实现了实用性，它不需要维护索引节点，而是在概念上将一个新节点放置在两个可能的层级：当前层级的顶部（即堆叠）或地面级别（即重开栈），并使用硬币翻转来决定目标层级（1/2 的概率堆叠，1/2 的概率重开栈）。缩短版本链查找中不必要线性遍历的关键是：当前版本节点的 v_ridgy 指针（类似索引跳转的指针）指向与当前节点层级相同或比当前节点层级更高的版本节点，作为跳转的快捷链接。图 2说明了原始跳表和节俭跳表之间的相似与不同之处。那么 v_ridgy 指针的定义是：给定一个节点 N_i，其在节俭跳表中的层级为 L_i，节点 N_i 的 v_ridgy 指针指向节点 N_p，满足 L_p 大于等于 L_i 且 N_p 是距离 N_i 最近的版本栈栈顶节点。如果 L_p 等于 L_i，那么 N_i 和 N_p 就像跳表中的索引节点一样，可以跳过一些底层节点；如果 L_p 大于 L_i，那么在 N_p 之后将能跳过更多的节点，而且数量指数增长。因此，通过遍历 v_ridgy 指针，可以实现对数级别的搜索，即控制搜索时间复杂度在 $O(lgN)$。

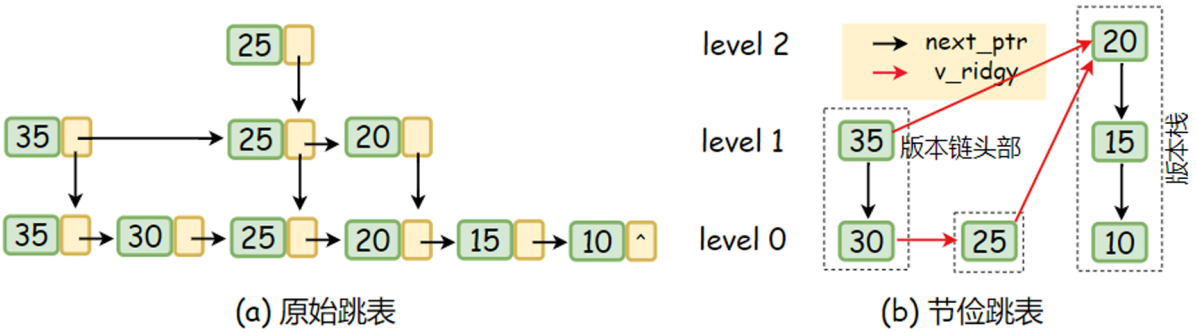

图 2. 原始跳表与节俭跳表的区别

3.1.2 插入新版本

当一个事务在运行时生成一个新版本时，会将版本插入节俭链表中（必定插入头部）。包括两个步骤：(1) 首先通过抛硬币决定版本的目标层级，(2) 然后找到一个 N_p 来设置新版本

节点的 v_ridgy 指针。通过抛硬币的形式，新版本堆叠在当前版本栈顶部或新开一个版本栈的概率各为 1/2。确定新版本节点的层级后，通过搜索找到一个 N_p 节点，并将新版本节点的 v_ridgy 指针指向该节点。整个过程如图 3所示，通过这个过程，可以满足每个节点的 v_ridgy 指针均指向先前版本栈的顶部节点，实现对数级别的搜索效率。

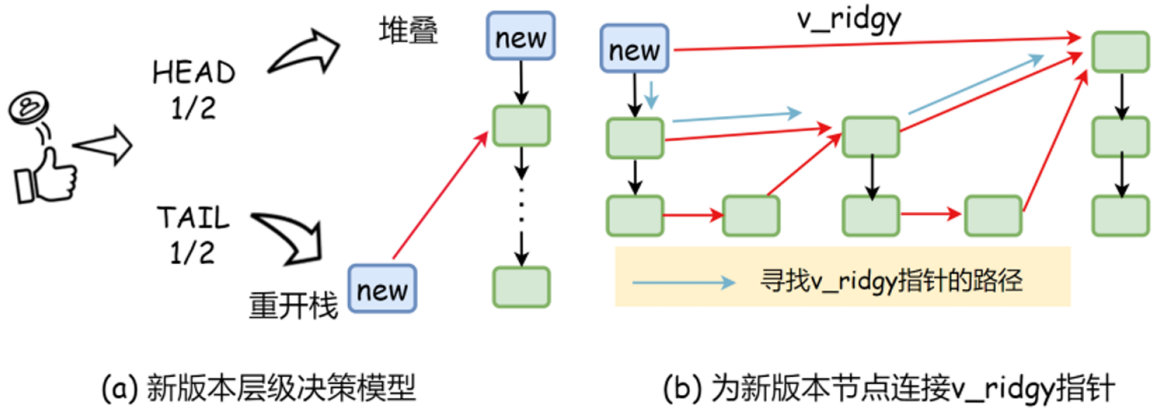

图 3. 插入新版本流程图

3.1.3 版本搜索

节俭跳表实现了从任何一个较新的版本到一个较旧的版本的对数时间搜索，且只需多维护一个指针 v_ridgy。在搜索一个版本时，使用 v_ridgy 指针（类似跳表中的索引），有效地将搜索引导到目标版本。给定节俭跳表和快照，通过检查 v_ridgy 指针所指版本节点 N_p 的时间戳来决定是否跳转。如果给定的快照时间戳比 N_p 小，将跳转到 N_p，跳过当前版本节点和 N_p 之间的所有版本节点；否则跳转到 next 指针所指节点（即原始版本链中的前一个版本）。图 4给出了两个版本搜索的例子。

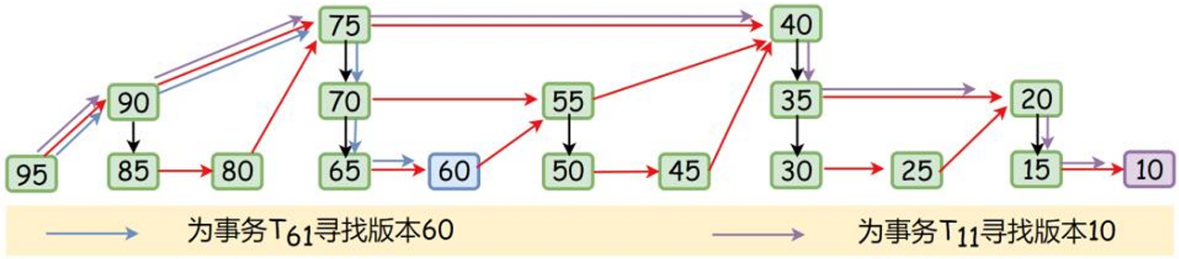

图 4. 版本搜索流程图

3.2 链间跳转结构

3.1 节叙述的链内搜索结构主要用于加速事务版本空间的点查询（特定记录单个版本链中的版本搜索）。而在实际应用中，常常会出现相邻记录（在索引上键值相邻）的范围查询事务，本节叙述的链间跳转结构用于加速版本空间上的范围查询（多个相邻记录对应的多个版本链中的范围版本搜索）。vWeaver 为每个记录版本多维护一个指针 k_ridgy，用于跳转到相邻版

本链中缩短下一个版本链的跳转次数，结合 3.2.1 节所叙述的指针 v_ridgy，在相邻版本链中快速搜索，就像在索引上扫描一样。

令 v_a^k 表示记录 r_a 在时间戳 k 被创建的记录版本。链间跳转需要解决的主要问题是，设置 v_a^i 的一个跳转指针 k_ridgy 指向相邻键记录 r_b 的版本 v_b^j，满足 v_b^j 是 v_a^i 在时间轴上的最近邻。如图 5所示的一个例子展现了 k_ridgy 的设置过程，当新版本 v_a^{85} 创建时（提交时），需要设置前一个版本 v_a^{80} 的 k_ridgy 指针指向 v_a^{80} 相邻键记录的最近邻版本 v_b^{82}。如果存在一个事务 T_i 其快照时间戳为 t_{s1} 落于 82 到 85 之间，需要查询记录 r_a 和 r_b，此时由于 v_b^{82} 是 v_a^{85} 的最近邻，对于 T_i 而言是最新的可见版本。T_i 访问 v_a^{80} 之后，可以顺着 k_ridgy 指针访问到 v_b^{82}。而如果 T_i 的时间戳 t_{s2} 落于 80 到 82 之间，v_b^{82} 对于 T_i 不可见，在 T_i 访问 v_a^{80} 之后，仍然可以顺着 k_ridgy 指针走到 v_b^{82}，然后使用 3.1 所述的链内搜索算法从 v_b^{82} 继续往后查询（这比从 r_b 版本链的头部开始查询要快得多）。

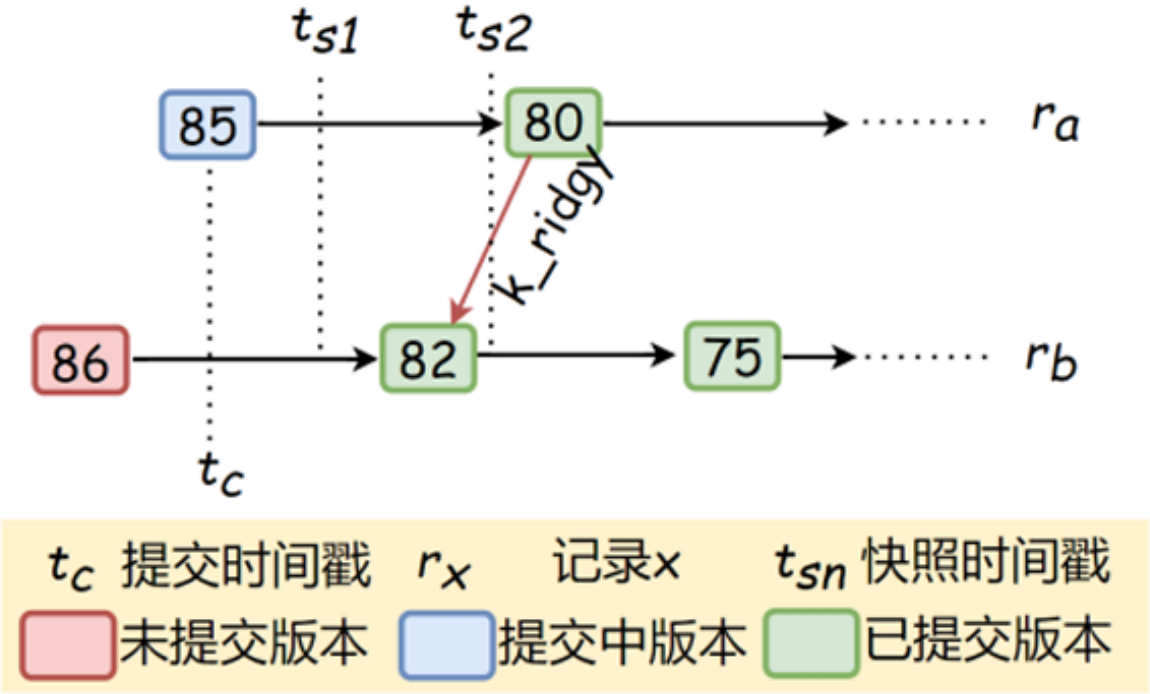

图 5. 链间连接过程

4 复现细节

4.1 创新点

基于 vWeaver 与可计算存储，我们提出了版本链卸载模型，如图 6所示。其中，主机的 NDP 数据库管理系统在普通的数据库管理系统上增加了版本链卸载模型，它会在查询涉及版本链查找逻辑时，将版本链查找逻辑提取出来并卸载到 CSD 中。CSD 中的 A53 核心负责接收读写请求与版本链查找请求，并与 DRAM 进行数据交互。

当系统接收到 OLTP 事务时，经过查询处理器与执行引擎，修改处于缓存区中的记录（若记录对应的页未命中则进一步进入 CSD 读取物理页）；缓存区驱逐页时，会进入 CSD 中，CSD 中 PS 区域的 A53 核心接管读取记录的逻辑，与存储介质进行交互，写入物理页。

当系统接收到 OLAP 事务时，经过查询处理器与执行引擎，当执行引擎涉及版本链查找逻辑时，版本链卸载模型提取出当前事务的读取时间戳 t_s 与记录对应的版本链首地址 r_p，进行 NDP 调用。NDP 调用首先需要将参数（r_p，t_s）传输到 CSD 中，然后发出指令让 CSD 中的 A53 根据所传输的参数进行盘内的版本链查找，并将查找到的可见版本返回给主机，实现版本链查找过程的卸载。

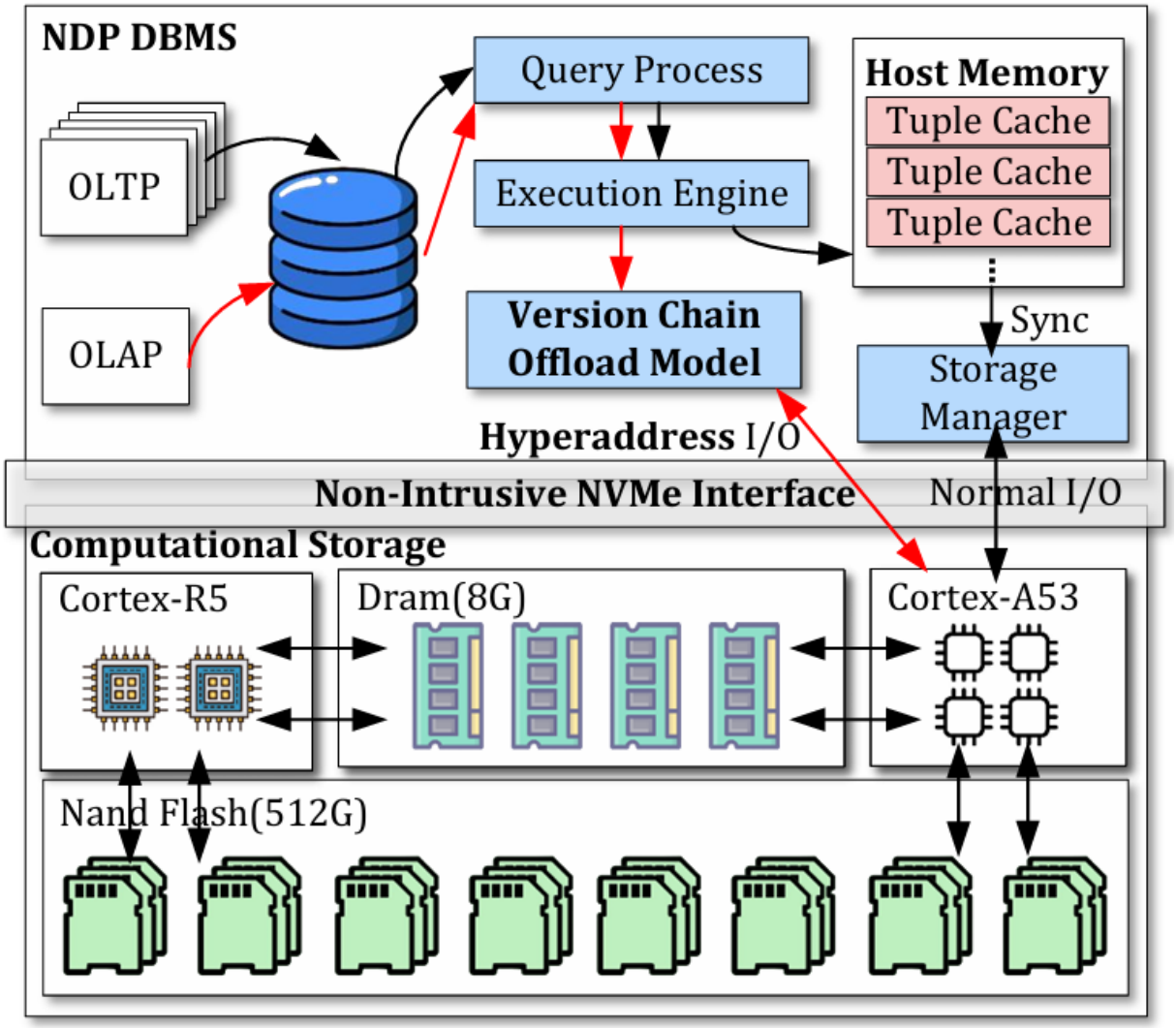

图 6. 版本链卸载模型架构图

4.2 CSD_BUSTUB

基于开源数据库框架 BUSTUB，我们开发了一个基于 CSD 的原型系统 CSD_BUSTUB。具体的实现过程如图 7所示。当一个事务发起 NDP 调用时，版本链卸载模型将其分解为写请求和读请求。首先，根据超地址，它将 NDP 调用的参数写入 CSD，其中（页面 ID，槽位 ID）表示查询记录对应的版本链的头地址，t_s 表示当前事务的读取时间戳。然后，它读取超地址范围内的一个地址，表示 NDP 调用的启动。CSD 中的 Cortex A53 核心识别该地址，并将常规读取操作转换为版本可见性检查逻辑。接着，它从 CSD 的 DRAM 中读取之前写入的参数，并根据（页面 ID，槽位 ID）定位闪存中的版本链头。然后，它执行版本可见性检查逻辑，以找到可见版本，并将其返回给数据库管理系统（DBMS），从而允许主机上的程序恢复执行。

与 BUSTUB 相比，CSD_BUSTUB 将版本可见性检查逻辑卸载到计算存储中，从而减轻了主机与存储之间的 I/O 负担。CSD 中的高带宽总线使得数据访问更加高效，促进了更快速的数据传输。此外，我们将版本跳跃加速技术（v_ridigy）集成到 CSD BUSTUB 中，进一步减少了存储中的随机访问，并提高了版本可见性检查的效率。

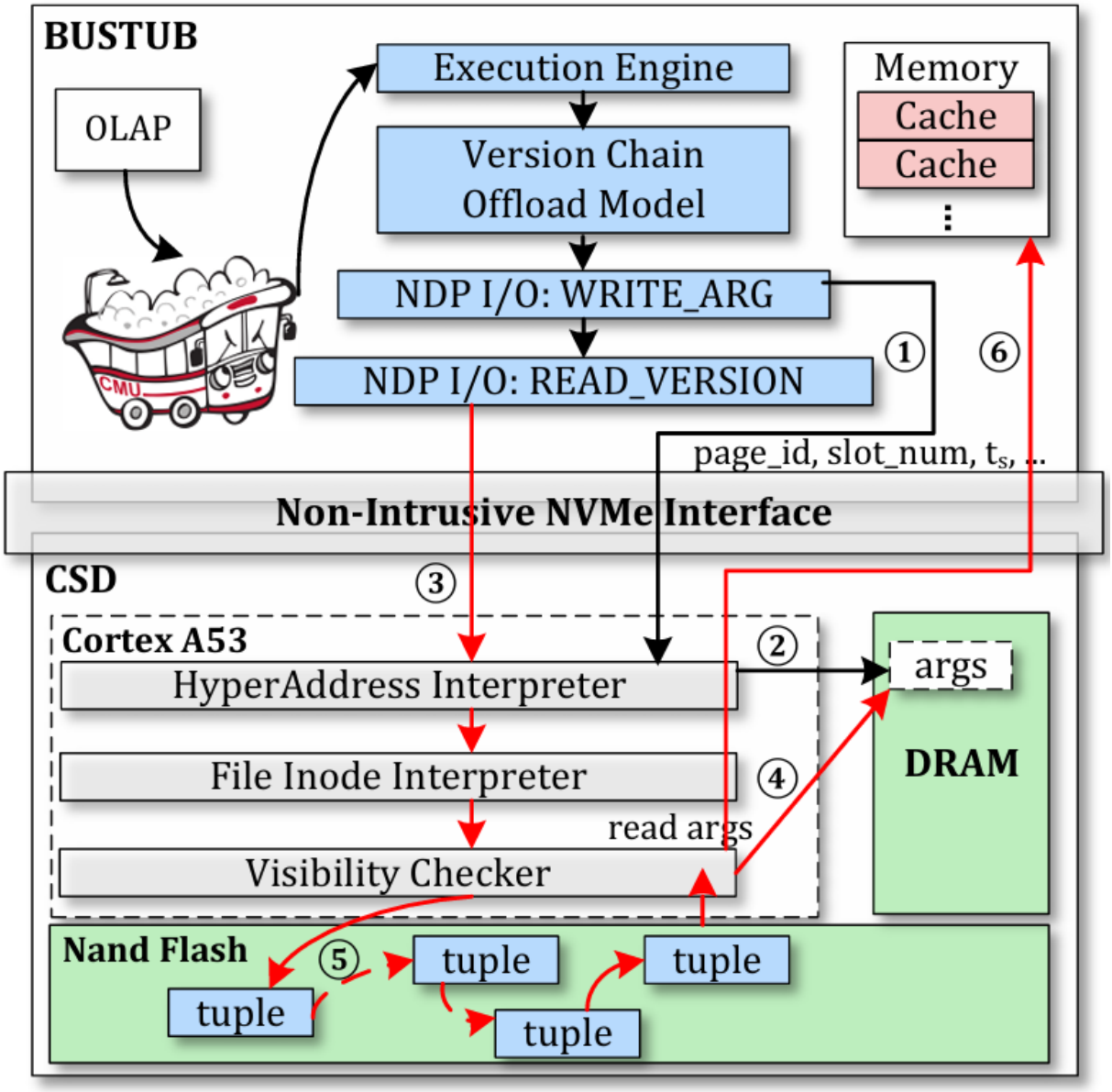

图 7. CSD_BUSTUB 架构图

5 实验结果分析

我们的实验设置如图 8所示。四种实验配置均基于真实的 CSD 硬件。实验涉及两个变量：版本可见性逻辑的执行位置（是在主机内执行还是在 CSD 内执行）以及是否应用版本链跳跃技术（如节俭跳跃列表）来加速。通过在这四种不同的实验设置中，我们评估了版本可见性检查操作对 MVCC_BUSTUB 和 CSD_BUSTUB 多版本原型系统性能的影响，具体通过变化 AP 事务的读取时间戳进行评估。

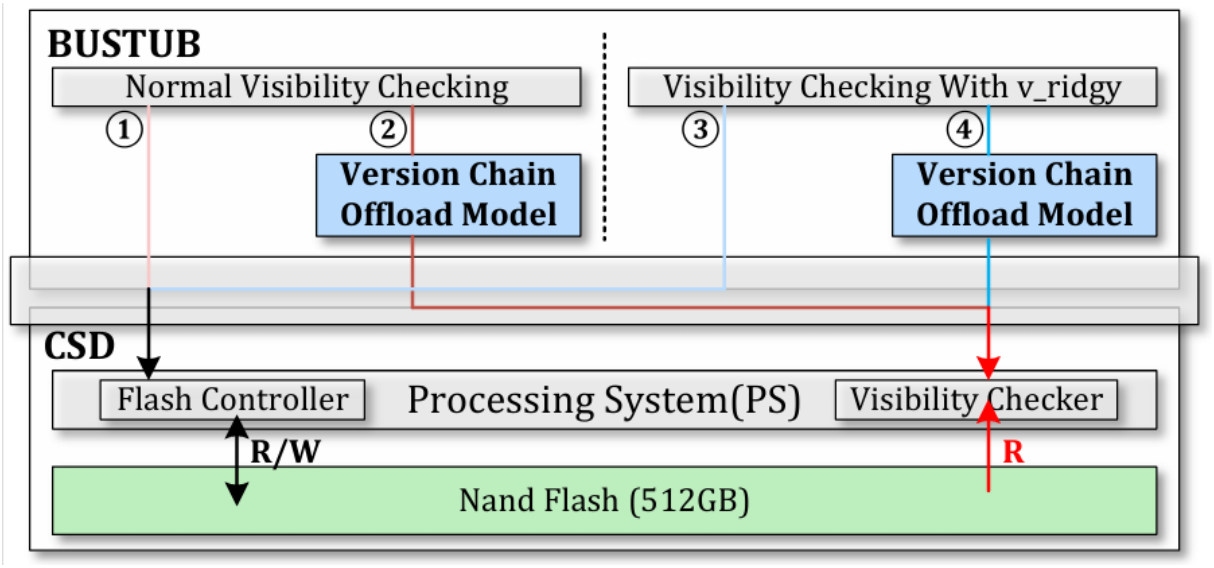

图 8. 对比实验设置

对于扫描评估，我们创建了一个包含多个记录的表，并持续执行 TP 事务以均匀更新记录，将版本链的平均长度增加到 10,000。随后，我们生成了具有读取时间戳为 1、50、100、500、1000、5000、8000、9000 和 10,000 的 AP 事务。在所有 TP 事务提交后，AP 事务执行了表查询。AP 事务在 MVCC_BUSTUB 和带有 v_ridgy 优化的 MVCC_BUSTUB 上的执行时间（以毫秒为单位）如图 9所示。

原始的 MVCC_BUSTUB 采用传统的版本链搜索方法（即从最新版本开始遍历，直到找到可见版本）。访问旧版本加剧了冷数据的传输，显著增加了 I/O 工作负载，导致随着读取时间戳增加，时间开销呈线性下降。通过图 9中的数据可以看到，带有 v_ridgy 优化的 MVCC_BUSTUB 的时间开销与原版本相比并不在同一数量级（因此采用了对数坐标）。这与理论预期一致，即节俭跳跃列表优化成功地将版本链搜索的次数控制在对数级别。

图 9还展示了 AP 事务在 CSD_BUSTUB 和 MVCC_BUSTUB 原型系统中的性能。在 MVCC_BUSTUB 中的版本链搜索过程中，频繁的冷数据传输（从 CSD 到主机）发生，导致 I/O 负载增加和性能下降。通过将版本可见性检查逻辑卸载到 CSD 内运行，减少了这一数据传输瓶颈，带来了显著的性能提升。这个性能与带有 v_ridgy 优化的 MVCC_BUSTUB 相当，甚至在版本链搜索次数较少时（例如 AP 读取时间戳为 8000 或 9000 时）还超越了它。

带有 v_ridgy 加速的两种原型系统在测试数据上的性能比较也显示在图 9中。尽管版本链跳跃技术在主机上运行时（MVCC_BUSTUB W.A.）已经提供了显著的加速，但仍然存在改进空间，因为跳转到较旧版本仍然可能导致冷数据传输。从图 9中可以看出，将版本链查询逻辑卸载到 CSD 上进一步提高了执行速度。然而，当 AP 事务的读取时间戳相对较新时，将逻辑卸载到 CSD 上可能会降低效率，因为较新的记录通常已经在缓存中，而卸载则会带来额外的数据传输开销（例如传递参数）。

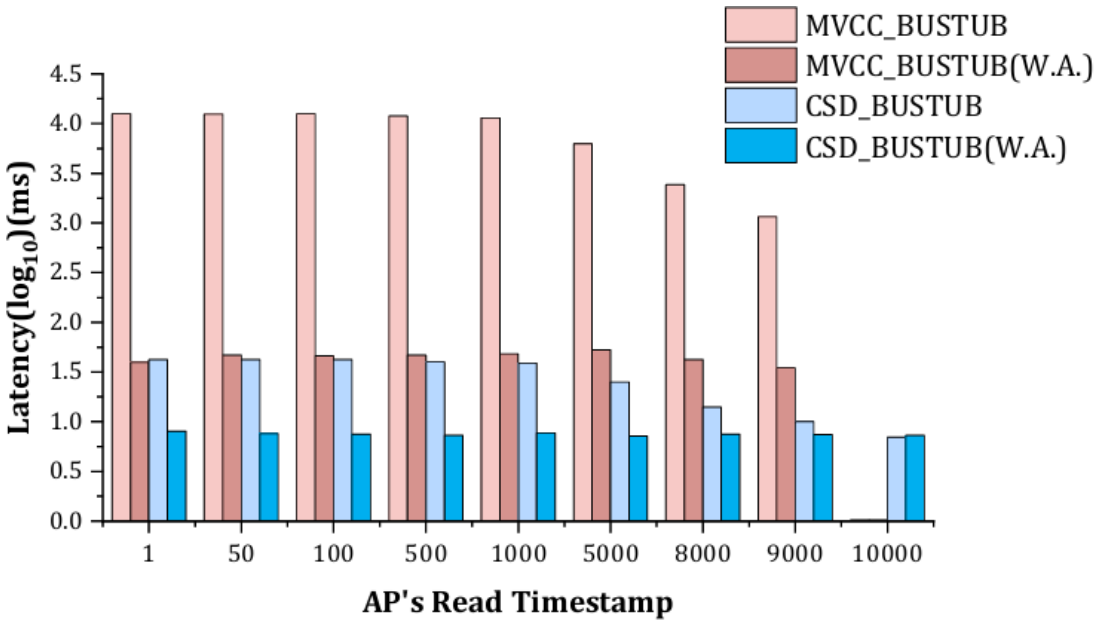

图 9. 不同时间戳快照下查询时间的变化情况

6 总结与展望

本文结合版本链跳转技术 vWeaver 与 CSD，对基于磁盘的 MVCC 系统进行查询加速。本文提出了版本链卸载模型，并基于版本链查找实现了 CSD 加速原型系统 CSD_Bustub，在不同测试基准上分别进行了有效性测试与性能测试。实验表明，携带版本链跳转技术的 MVCC_Bustub 原型系统相比于不携带版本链跳转技术的 MVCC_Bustub 系统，平均情况下速度提高了约 120 倍,这证实了版本链跳转技术的有效性。CSD_Bustub 相比于 MVCC_Bustub (均不携带版本链跳转技术)，平均情况下速度提高了约 200 倍，在携带版本链加速的情况下进行版本链卸载，也有约 6 倍的性能提升，这证实了版本链卸载模型的正确性与加速有效性。

虽然本文所提出的版本链卸载模型结合版本链跳转技术，已在版本链查找加速方面取得成绩。但由于本文实现的原型系统基于 CSD 当中的 DRAM，仅 4G 的容量无法承担较大的工作负载，未来可探索基于 CSD 当中的 Flash 进行卸载模型的设计。同时，当前的原型工具基于简单的 Bustub 框架，无法应对复杂的查询语句，未来可探索基于成熟的数据库系统（如 MySQL、PostgreSQL 等）进行 CSD 加速。

参考文献

[1] Jan Böttcher, Viktor Leis, Thomas Neumann, and Alfons Kemper. Scalable garbage collection for in-memory mvcc systems. *Proceedings of the VLDB Endowment*, 13(2):128–141, 2019.

[2] Richard Cole, Florian Funke, Leo Giakoumakis, Wey Guy, Alfons Kemper, Stefan Krompass, Harumi Kuno, Raghunath Nambiar, Thomas Neumann, Meikel Poess, et al. The mixed workload ch-benchmark. In *Proceedings of the Fourth International Workshop on Testing Database Systems*, pages 1–6, 2011.

[3] Franz Färber, Sang Kyun Cha, Jürgen Primsch, Christof Bornhövd, Stefan Sigg, and Wolfgang Lehner. Sap hana database: data management for modern business applications. *ACM Sigmod Record*, 40(4):45–51, 2012.

[4] Insoon Jo, Duck-Ho Bae, Andre S Yoon, Jeong-Uk Kang, Sangyeun Cho, Daniel DG Lee, and Jaeheon Jeong. Yoursql: a high-performance database system leveraging in-storage computing. *Proceedings of the VLDB Endowment*, 9(12):924–935, 2016.

[5] Alfons Kemper and Thomas Neumann. Hyper: A hybrid oltp&olap main memory database system based on virtual memory snapshots. In *2011 IEEE 27th International Conference on Data Engineering*, pages 195–206. IEEE, 2011.

[6] Jongbin Kim, Hyunsoo Cho, Kihwang Kim, Jaeseon Yu, Sooyong Kang, and Hyungsoo Jung. Long-lived transactions made less harmful. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 495–510, 2020.

[7] Jongbin Kim, Kihwang Kim, Hyunsoo Cho, Jaeseon Yu, Sooyong Kang, and Hyungsoo Jung. Rethink the scan in mvcc databases. In *Proceedings of the 2021 International Conference on Management of Data*, pages 938–950, 2021.

[8] Jongbin Kim, Jaeseon Yu, Jaechan Ahn, Sooyong Kang, and Hyungsoo Jung. Diva: Making mvcc systems htap-friendly. In *Proceedings of the 2022 International Conference on Management of Data*, pages 49–64, 2022.

[9] Kitaek Lee, Insoon Jo, Jaechan Ahn, Hyuk Lee, Hwang Lee, Woong Sul, and Hyungsoo Jung. Deploying computational storage for htap dbmss takes more than just computation offloading. *Proceedings of the VLDB Endowment*, 16(6):1480–1493, 2023.

[10] Yunjae Lee, Jinha Chung, and Minsoo Rhu. Smartsage: training large-scale graph neural networks using in-storage processing architectures. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*, pages 932–945, 2022.

[11] Baptiste Lepers, Oana Balmau, Karan Gupta, and Willy Zwaenepoel. Kvell+: Snapshot isolation without snapshots. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 425–441, 2020.

[12] William Pugh. Skip lists: a probabilistic alternative to balanced trees. *Communications of the ACM*, 33(6):668–676, 1990.

[13] Xuan Sun, Hu Wan, Qiao Li, Chia-Lin Yang, Tei-Wei Kuo, and Chun Jason Xue. Rm-ssd: In-storage computing for large-scale recommendation inference. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 1056–1070. IEEE, 2022.

[14] Tobias Vinçon, Christian Knödler, Leonardo Solis-Vasquez, Arthur Bernhardt, Sajjad Tamimi, Lukas Weber, Florian Stock, Andreas Koch, and Ilia Petrov. Near-data processing in database systems on native computational storage under htap workloads. *Proceedings of the VLDB Endowment*, 15(10):1991–2004, 2022.

[15] Mark Wilkening, Udit Gupta, Samuel Hsia, Caroline Trippel, Carole-Jean Wu, David Brooks, and Gu-Yeon Wei. Recssd: near data processing for solid state drive based recommendation inference. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 717–729, 2021.