关系型数据库向量检索的存储与执行路径优化

赵淳[1]，盛凌志[2]，任蕾福[1]，金仁哲[2]，崔凯峰[1]，王羽婷[2]
1. 中国移动通信有限公司数智事业部，北京 102200；
2. 中国移动通信集团北京有限公司，北京 100027

摘要

针对关系型数据库向量检索中存储访问与执行调度开销大的问题，在openGauss内核中实现了内存亲和索引布局与旁路执行路径。前者通过内存映射直接访问只读向量索引页面，减少缓冲区管理开销；后者基于OpFusion框架简化查询执行流程，将调用层级由约15层压缩至约5层。在50万条1024维向量、前10项近邻检索负载下，内存亲和布局使缓冲区管理器页面访问次数减少93.1%，索引扫描耗时降低51.2%；两项优化叠加后，低并发下吞吐量最高提升69.2%，平均延迟降低43.0%，第99百分位延迟降低51.5%，查询结果与基线一致。结果表明，优化存储访问与执行路径可在保持检索语义的前提下提升只读向量检索性能。

关键词

向量检索；内存亲和布局；数据库；层次化可导航小世界图

中图分类号：TP391　　文献标志码：A

Optimization of storage and execution paths for vector retrieval in relational databases

ZHAO Chun[1], SHENG Lingzhi[2], REN Leifu[1], JIN Renzhe[2], CUI Kaifeng[1], WANG Yuting[2]
1. Digital Intelligence Business Division, China Mobile Communications Co. , Ltd. , Beijing 102200, China；
2. China Mobile Communications Group Beijing Co. , Ltd, Beijing 100027 China

Abstract

To reduce storage-access and execution-scheduling overhead in vector retrieval within relational databases, a memory-affine index layout and a bypass execution path were implemented in the openGauss kernel. The former enabled direct access to read-only vector index pages through memory mapping, reducing buffer-management overhead. The latter simplified query execution through the OpFusion framework, reducing the call depth from approximately 15 layers to approximately 5 layers. The optimizations were evaluated using 500, 000 vectors of 1, 024 dimensions under top-10

nearest-neighbor retrieval. The memory-affine layout reduced page accesses through the buffer manager by 93.1% and index-scan time by 51.2%. With both optimizations enabled, throughput was increased by up to 69.2% under low concurrency, while average latency and 99th-percentile latency were reduced by 43.0% and 51.5%, respectively. Query results remained identical to the baseline. The results indicate that optimizing storage access and execution paths improves read-only vector retrieval performance while preserving retrieval semantics.

Key words
vector search, memory-centric layout, database, hierarchical navigable small world

0 引言

随着大模型训练与推理一体化架构逐步成为主流，高维向量数据的统一存储、高效读写与低延迟检索，已成为支撑大模型语义理解、检索增强生成（retrieval-augmented generation，RAG）、推荐系统等应用的核心技术[1-2]。大模型训推场景对向量数据的访问呈现高频、随机、只读等特点，同时要求系统具备事务一致性、结构化查询语言（structured query language，SQL）兼容与统一存储管理能力，传统架构难以同时满足。

当前，向量检索系统主要分为两类技术路线：一类是以Milvus[3]为代表的专用向量数据库，它们针对纯向量检索场景深度优化，检索效率高，但缺乏事务处理、SQL查询、权限管理等关系型数据库核心能力，仅面向推理侧设计，无法支撑大模型训练数据管理与训推一体全流程；另一类是在传统关系型数据库中集成向量检索能力，例如面向PostgreSQL的近似最近邻检索扩展PASE[4]。这类方案可完整复用SQL生态，但存储引擎与执行器面向通用联机事务处理（online transaction processing，OLTP）与联机分析处理（online analytical processing，OLAP）负载设计，未针对向量检索的访问模式优化，在训推一体高并发场景下存在明显性能瓶颈。

针对上述问题，基于openGauss数据库的向量引擎[5-6]，面向大模型训推一体场景的向量检索需求，提出了两项核心优化——内存亲和索引布局与向量检索SQL旁路执行（SQL bypass）快速执行路径，旨在从存储访问和执行器调度两个层面消除关系型数据库在向量检索场景下的性能瓶颈。基于内存映射系统调用mmap的内存亲和布局：将层次化可导航小世界图（hierarchical navigable small world graph，HNSW）索引文件直接映射至进程地址空间，配合页面指针数组与按页组批量映射实现零拷贝访问，绕过缓冲池（buffer pool）开销，有效提升检索性能；基于OpFusion的近似最近邻（approximate nearest neighbor，ANN）查询SQL bypass快速执行路径：将执行器调用层级由约15层压缩至约5层，有效抑制第99百分位延迟（P99延迟）。

以上创新点在openGauss内核中完成两项正交优化的工程实现，并通过系统性实验验证：相较优化前基线，向量检索的吞吐与尾延迟均有明显改善，且召回率保持稳定、无精度损失。

其余各部分结构安排如下：第一章阐述向量数据库与向量检索的研究现状，第二章分析openGauss数据库推理阶段的存

储瓶颈，第三章重点介绍本文针对向量检索提出的性能优化创新方案，第四章对实验结果进行分析，第五章为全文总结。

1 向量数据库与向量检索研究现状

1.1 近似最近邻搜索算法

近似最近邻搜索是向量检索的核心技术，其目标是在高维空间中快速找到与查询向量距离最近的K个数据点。随着数据规模从百万级增长到十亿级，精确搜索的计算代价已不可接受，ANN方法通过牺牲少量精度换取数量级的速度提升，成为工业界的主流选择[1,7]。

现有ANN算法主要分为四类[7]：(1)基于哈希的方法，如局部敏感哈希(locality sensitive hashing，LSH)，通过哈希函数将相似向量映射到相同桶中。此类方法检索速度极快，但为达到高精度通常需要构建多张哈希表，导致巨大的内存开销和精度不稳定。(2)基于树的方法，如KD-Tree和Ball-Tree，通过空间划分加速搜索。在低维空间中，这类方法能实现对数级别的搜索效率，但随着维度升高，其性能会因“维度灾难”而急剧下降，甚至退化至接近线性扫描。(3)基于量化的方法，通过向量压缩降低存储和计算开销。最具代表性的乘积量化(product quantization，PQ)将高维向量划分为多个子空间并分别进行量化编码，将高维距离计算转化为高效的查表求和操作[8]。其一系列改进工作，如优化乘积量化(optimized product quantization，OPQ)通过最小化量化误差提升精度，以及近期提出的RaBitQ通过理论保证的误差界和比特级运算实现更优的精度-效率权衡[9]，均推动了该方向的发展。量化方法本质上是计算密集型的，其瓶颈在于查表计算，对内存带宽相对友好，并可利用图形处理器(graphics processing unit，GPU)等硬件进行批处理加速。(4)基于图的方法，通过构建近邻图实现高效的贪心搜索。Li等人[7]的大规模基准测试表明，基于图的方法在召回率和查询延迟的综合表现上持续领先于其他类别。

在基于图的方法中，HNSW[10]因其卓越的性能已成为工业界事实上的标准索引结构。HNSW的核心在于构建一个多层的图结构：上层由少量节点和“长边”构成，实现快速、粗粒度的区域定位；下层则包含全部节点和“短边”，用于进行精细搜索。其搜索过程从顶层入口点开始，在每一层执行贪心算法找到局部最近邻，然后逐层下降，最终在底层通过扩展搜索获得精确结果，理想复杂度可达$O(\log N)$。HNSW的优越性能源于其能以极高的概率将搜索路径长度维持在极低的水平。

然而，HNSW的性能优势伴随着特定的、对底层系统不友好的数据访问模式。其搜索过程本质上是高度随机的指针追逐。为了找到最近邻，算法需要在图的各层间跳跃，频繁访问内存中不连续、不可预测的节点。这种访问模式对中央处理器(central processing unit，CPU)缓存极不友好，导致高缓存缺失率，并将查询延迟的瓶颈从CPU计算转移到了内存访问延迟。

1.2 向量数据库系统

随着向量检索需求的爆发式增长，向量数据库系统已形成多元化的技术生态[1]。根据系统架构的不同，可将现有方案分为两类：专用向量数据库、现有数据库加上向量扩展。

专用向量数据库从零设计，以向量为核心数据模型。国外代表性系统包括：Milvus[3]是由 Zilliz 开发的开源向量数据库，支持 HNSW、基于倒排文件（inverted file，IVF）的精确距离扫描索引（IVF-Flat）和倒排文件与乘积量化结合的索引（IVF-PQ）等多种索引，提供分布式架构和云原生部署能力，并于2021年发表了系统论文；Pinecone 是全托管的商业向量数据库服务，以免运维和百毫秒以内的查询延迟为主要特点；Weaviate 支持向量与关键词的混合检索；Qdrant 以 Rust 实现，强调高性能和类型安全。这些专用系统在纯向量检索场景下表现优异，但普遍缺乏完整的事务处理、复杂SQL查询和细粒度权限管理能力。

数据库向量扩展在现有数据库中集成向量检索能力。PostgreSQL 的 pgvector 是该方向最知名的开源项目，通过扩展机制添加了 vector 数据类型和 HNSW/IVFFlat 索引。这种方案的优势在于可以复用 SQL 生态的全部能力——事务、JOIN、权限、备份恢复等——但受限于 PostgreSQL 执行器和存储引擎的通用设计，在向量检索性能上与专用系统存在差距；针对这一差距，已有工作分别从执行模型[11]、索引框架[4]、带过滤条件的近似最近邻搜索图算法[12]以及面向图索引的存储布局及缓冲策略[13]等角度展开探索；例如，openGauss 在 6.0 版本集成了向量检索能力；阿里云 AnalyticDB-V 在其 OLAP 引擎中同样集成了向量检索能力。

1.3 向量检索系统优化技术研究进展

关系型数据库的向量扩展需在既有存储引擎、缓冲管理与执行框架内承载高频、随机、只读的 ANN 检索负载。现有研究多集中于算法层面，如更好的索引结构与量化方法，对系统层面的关注相对不足。已有的系统层工作可归为执行与算子、缓冲管理与访存路径、索引与存储布局、混合查询与属性过滤四个方向，本文关注的两个瓶颈——执行器调度开销与存储访问开销——分别落在前两个方向上。

执行与算子层面。pgvector 经索引访问方法接口注册 HNSW 索引，ANN 检索交由通用执行器完成，一条仅含 LIMIT 与索引扫描两个节点的查询，仍需付出与多表连接相同的 Portal 创建与递归初始化开销；这一开销源于 PostgreSQL 系列数据库的 Portal-Executor 递归执行模型，一条简单查询需经过十余层函数调用[6]。VBASE[11]利用向量索引遍历的"松弛单调性"统一了 Top-K 与 Range 算子的终止判据，但停留在算子语义层，未削减执行框架自身的固定开销。openGauss 已有的 OpFusion 框架为简单的 INSERT/UPDATE/DELETE/SELECT 查询提供了 bypass 执行路径，跳过 Portal-Executor 的递归初始化与调度，但尚未覆盖 ANN 向量检索查询。

缓冲管理与访存路径层面。Crotty 等人[14]在 CIDR 2022 上系统分析了 mmap 与 buffer pool 的权衡，指出 mmap 可以避免 buffer pool 的哈希查找、引用计数与轻量级锁（lightweight lock，LWLock）开销，但在通用数据库管理系统（database management system，DBMS）场景下存在脏页回写不可控、内存驻留策略无法精确管理等正确性风险，这也是关系型数据库迟迟未对向量索引启用 mmap 的主要原因。ScaleCache[15]转而提升 buffer pool 自身的众核可扩展性，以分组引用计数与单指令多数据（single instruction, multiple data，SIMD）加速的页映射哈希表降低

Pin/Unpin与哈希查找的竞争，但其面向通用负载，并未为只读向量索引消除整条页面获取路径。Oracle 23ai把HNSW索引移出缓冲区缓存、常驻系统全局区（system global area，SGA）中专设的向量内存池，方向与本文一致，但以索引全量驻留内存为前提，规模受内存池上限约束，且无法在资源不足时退回通用路径。Milvus在其2.x版本中引入mmap以扩展存储容量，允许将索引与原始数据映射到磁盘以降低内存占用，但该方案实现于专用向量数据库的语境下，不需要考虑与关系型数据库buffer pool的兼容性。DiskANN[16]采用基于SSD的图索引与搜索机制，在有限内存条件下支持十亿级向量检索，其重点在于大规模近似最近邻搜索的存储与访问效率。本文则关注关系型数据库内只读向量索引的页面访问与执行调度开销。

索引与存储布局层面。GaussDB-Vector[13]为图索引设计了面向输入/输出（input/output，I/O）优化的存储架构与缓冲策略，pgvectorscale以StreamingDiskANN和统计式二值量化支持超出内存容量的向量集合；面向专用检索引擎，学术界亦提出了I/O高效的近邻查询缓存策略[17]与更新友好的磁盘常驻索引框架[18]。这类工作以更少的字节换取更大的容量，或优化专用引擎自身的存储与缓存层，但均建立在既有页面获取接口之上，单次页面访问的固定开销并未减少。

混合查询与属性过滤层面。pgvector自0.8.0起引入迭代式索引扫描，以缓解属性过滤后结果不足的问题；AnalyticDB-V[19]则以精度感知的代价模型在多种混合查询计划间择优。二者均面向带过滤的复杂检索，而大模型推理侧最高频的、不带过滤的纯向量Top-K查询反而未被专门优化。

综上，现有工作的不足集中于两点：访存路径的改造缺少与通用缓冲管理并存的兼容设计；纯向量Top-K查询的执行框架开销被忽视。针对第一点，本文的内存亲和布局仅对只读HNSW索引启用mmap，并以实例级、会话级、索引级三层开关与映射失败自动回退规避文献[14]指出的风险，相比全内存驻留方案不受索引规模约束；针对第二点，本文的SQL bypass将纯向量Top-K查询从Portal-Executor递归框架中摘出。两项优化分别作用于存储访问层与执行调度层，正交且可叠加。

2 openGauss数据库推理阶段存储瓶颈分析

通过对openGauss向量检索查询的火焰图分析，本文识别出两个关键的系统级性能瓶颈。

如图1所示，火焰图清晰地展示了向量检索查询的CPU时间分布。其中，距离计算函数占据了最大的CPU时间比例，这是HNSW算法本身的计算开销，属于算法层面的固有成本。除此之外，两个系统级开销尤为突出：

瓶颈一：buffer pool存储访问开销。buffer pool相关函数（ReadBuffer、LockBuffer、UnlockReleaseBuffer等）在向量检索查询的CPU时间中占比达到5%~10%，成为继距离计算之后的第二大性能瓶颈。这是因为HNSW图搜索具有高频、随机、只读的访问模式——一次Top-K查询需要访问数百甚至数千个图节点，每个节点的邻居列表和向量数据分别存储在不同的页面中，页面局部性差会直接放大页面访问次数[20]，每次页面访问都需要

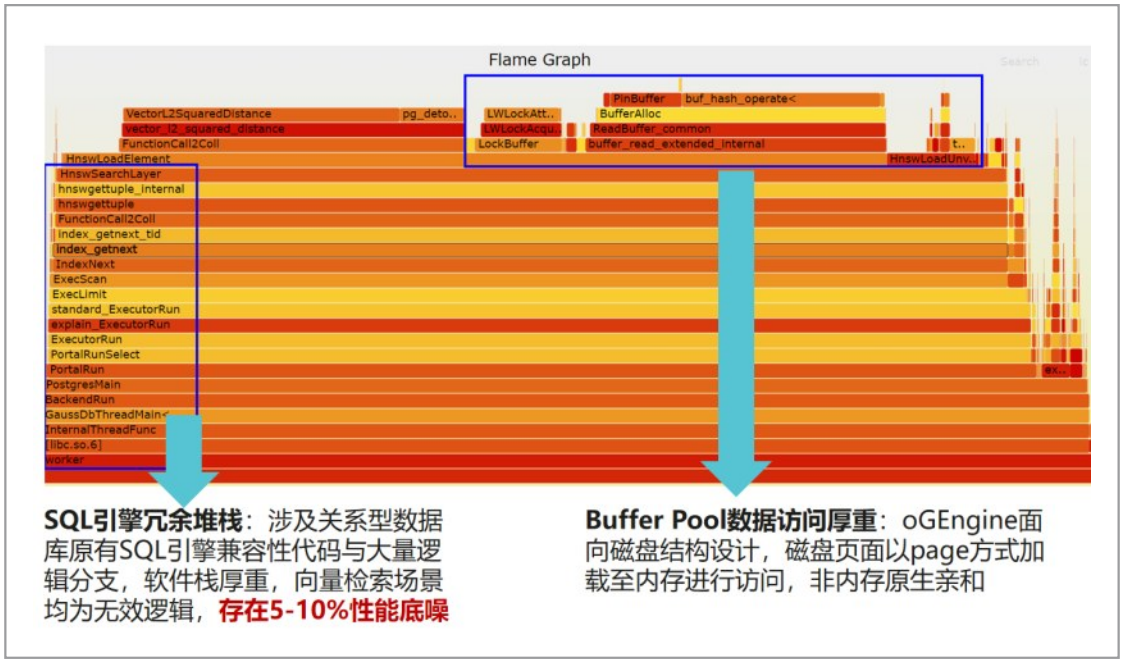

图1 SQL检索火焰图

执行哈希查找、原子操作（Pin/Unpin）和锁获取/释放。传统关系型数据库采用buffer pool架构管理磁盘数据页面的内存缓存，当执行器需要访问某个数据页面时，需要经过以下步骤：（1）根据页面标识构造BufferTag；（2）在共享哈希表中查找对应的缓冲区描述符；（3）对缓冲区执行Pin操作（增加引用计数）；（4）获取缓冲区的轻量级锁（LWLock）；（5）读取页面内容；（6）释放LWLock并执行Unpin操作[14]。上述流程对于传统OLTP工作负载是合理的，但对于向量索引的只读检索场景，buffer pool的脏页管理、预写日志（write-ahead logging，WAL）等写入相关机制完全是冗余开销。

瓶颈二：执行器递归调度开销。传统数据库的查询执行采用Portal-Executor递归模型[6]，一条向量检索查询在传统路径下共涉及约15层函数调用，其中Portal创建、执行器启动、递归初始化、结果返回、递归清理等阶段对于向量检索查询而言是冗余开销。这些阶段是为支持复杂查询（如多表JOIN、子查询、游标等）而设计的通用框架代码，对于向量检索这类计划树结构简单（仅LIMIT + ANN索引扫描两个节点）、执行路径固定的查询而言，是不必要的。

3 向量检索性能优化方案

针对以上问题，本章分别提出基于内存亲和布局的向量索引存储优化和基于SQL bypass的向量检索执行路径优化方案。两项优化设计正交，可独立或协同使用。

需要明确的是，本文提出的两项优化主要面向只读、无属性过滤的ANN-TopK查询。典型查询形式为：在已建立的HNSW索引上，根据外部输入向量执行近邻搜索，并按照距离返回前K条结果。内存亲和布局针对只读HNSW索引设计，

通过mmap直接访问索引页面；SQL bypass路径则面向计划结构简单的ANN索引扫描查询，通常由ANN索引扫描和LIMIT构成，不包含WHERE过滤、JOIN、子查询和复杂投影。因此，带属性过滤的混合检索、JOIN检索、复杂查询计划以及写入、更新和高频增量维护等场景不属于本文优化的主要适用范围。对于不满足上述条件的查询，系统保留传统buffer pool页面访问路径和Portal-Executor执行路径；当mmap不可用或映射失败时，系统自动回退到传统访问路径，以保证查询功能和结果正确性。

3.1 内存亲和布局

3.1.1 基于mmap的内存亲和访问机制设计

针对buffer pool在向量检索场景下的性能瓶颈，本文提出了一种基于mmap系统调用的内存亲和索引布局方案，其核心思想是：对于HNSW向量索引这类只读、高频访问的数据，绕过buffer pool，通过mmap将索引文件直接映射到进程地址空间，使得页面访问退化为简单的指针解引用操作。

整体架构如图2所示，采用双通道设计：在HNSW图搜索过程中，通过CanUseMmap()函数判断当前索引是否启用了mmap优化。若启用，则走mmap路径，通过直接指针访问读取页面数据；若未启用或mmap失败，则自动回退到传统buffer pool路径，保证功能的正确性和可用性。mmap优化提供了三层控制粒度（实例级、会话级、索引级），允许管理员根据实际场景灵活配置。

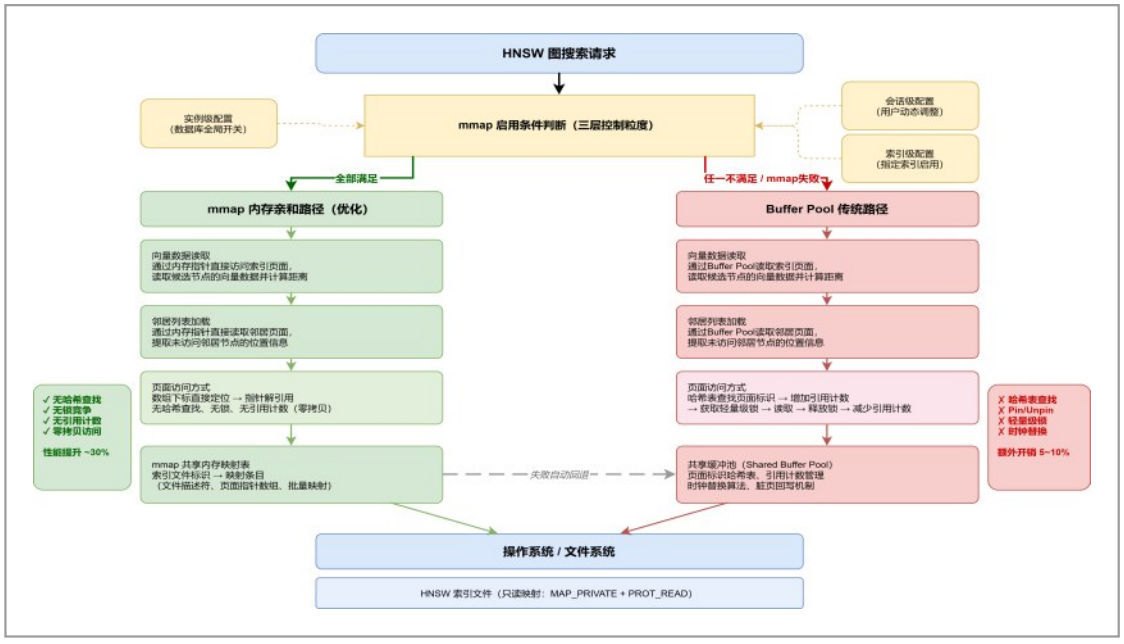

图2 内存亲和访问机制架构图

在共享内存管理方面，本文设计了一种专用的内存映射管理条目，用于在多个数据库后端进程之间协调 HNSW 索引文件的映射。每个 HNSW 索引文件对应一个映射条目，并通过共享哈希表进行查找。需要区分的是，跨进程共享的主要是索引文件标识、文件大小、映射页组大小、最大块号等映射元数据；页面指针则表示具体进程地址空间中的映射结果，不能简单理解为可在不同进程之间直接解引用的统一虚拟地址。当某个后端进程首次访问索引页面时，系统根据映射条目中的文件标识、块号和页组大小执行 mmap，并记录该页组的映射地址和页面定位信息。后续访问根据页号查找已经建立的映射，从而减少重复的文件定位和 mmap 调用。由于不同后端进程具有独立的虚拟地址空间，同一索引文件在不同进程中建立映射后返回的地址可能不同，因此页面指针的有效性应限定在建立该映射的进程地址空间内。换言之，共享哈希表用于协调索引文件的映射管理和页面定位，实际页面地址的使用必须由当前进程保证其有效性。映射条目的核心信息如表 1 所示。

表1 内存映射管理条目的核心信息

信息类别	说明	作用
索引文件标识	唯一标识一个索引文件段	作为共享哈希表的查找键
文件描述符	操作系统层面的文件句柄	用于后续的 mmap 系统调用
文件总大小	索引文件的字节数	确定可映射的最大范围
页组大小	每次批量映射的页面数量	控制 mmap 调用的粒度
最大块号	文件中的最大页面编号	防止越界访问
元数据页指针	指向索引元数据页的内存地址	快速获取索引参数信息
页面指针数组	存储所有已映射页面的内存地址	在当前进程的有效地址空间内，通过页面编号快速定位目标页面

其中，页面指针数组是实现零拷贝访问的关键数据结构。当某个页面首次被访问时，系统通过 mmap 将其映射到内存，并将返回的内存地址存入数组的对应位置。此后，同一进程访问该页面时，只需通过页面编号作为数组下标即可直接获取内存地址，无需再次调用 mmap 或经过 buffer pool 的哈希查找。

为进一步减少 mmap 系统调用的次数，本文采用了按页组批量映射策略。如图 3 所示，当访问某个目标页面时，系统不仅映射该页面，还同时映射其所在页组内的所有连续页面。页组大小可通过系统参数配置（默认为多个连续页面）。这样，后续对同一页组内其他页面的访问可以直接通过指针数组获取地址，避免了频繁的系统调用开销。

3.1.2 HNSW 图搜索的 mmap 优化路径

在 HNSW 图搜索过程中，mmap 优化主要体现在两个关键操作上：向量数据读取（MmapLoadElement）和邻居列表加载（HnswLoadUnvisitedFromMmap）。

MmapLoadElement 函数用于在图搜索过程中读取候选节点的向量数据并计算距离。其核心逻辑是：通过 GetMMapPage 获取目标页面的 mmap 指针，然后直接通过 PageGetItem 读取元组

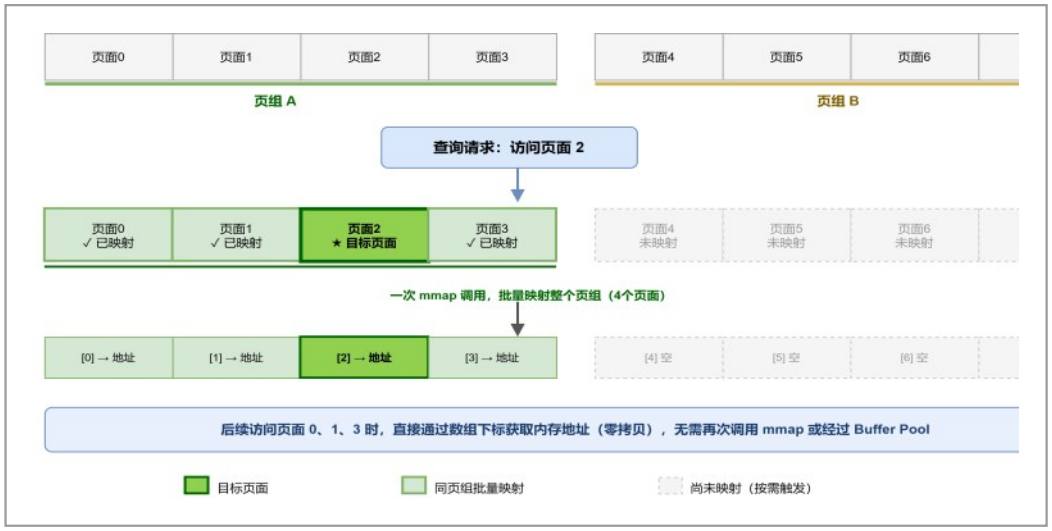

图3　按页组批量映射策略示意图

数据，最后调用距离函数计算查询向量与候选向量的距离。与传统的HnswLoadElement相比，省去了ReadBuffer（哈希查找 + 可能的磁盘I/O）、LockBuffer（获取LWLock）和UnlockReleaseBuffer（释放LWLock + Unpin）三个步骤。当mmap页面不可用时，函数自动回退到HnswLoadElement。

MmapLoadElement算法与传统HnswLoadElement对比如图4所示：

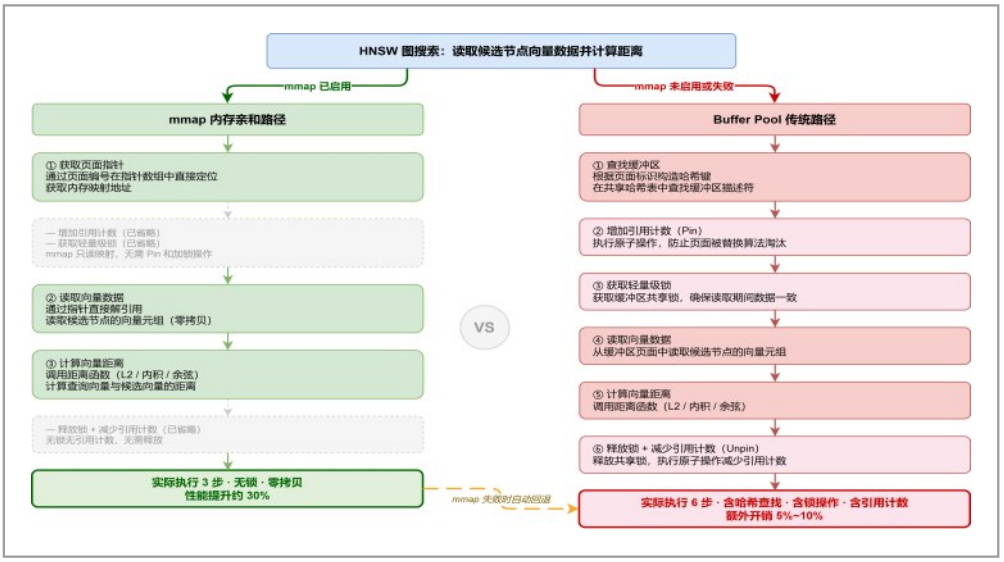

图4　MmapLoadElement的算法与传统HnswLoadElement对比

3.2 向量检索执行路径优化

3.2.1 基于OpFusion框架的ANN查询识别

针对执行器递归调度开销，本文提出了基于OpFusion框架的SQL bypass快速执行路径。如第2章所述，openGauss的OpFusion框架是一个轻量化查询执行机制，为简单查询提供bypass执行路径，跳过Portal-Executor的递归初始化和调度开销。OpFusion已支持INSERT、UPDATE、DELETE和简单SELECT等操作类型。本文在此基础上新增了面向ANN向量检索的执行类型，将向量检索查询纳入bypass优化范围。

ANN查询的识别分为两个阶段。第一阶段是前置条件检查：系统验证当前节点为数据节点、计划未被共享、非自定义计划等基本条件，确保查询具备进入bypass路径的前提。

当前置条件全部满足后，进入第二阶段——查询类型识别。系统检查计划树结构，判断是否满足以下条件：（1）根节点为LIMIT节点（可选），LIMIT值为常量或参数绑定；（2）扫描节点为ANN索引扫描节点；（3）无WHERE过滤条件；（4）目标列均为简单列引用；（5）距离函数参数为外部绑定参数。当所有条件满足时，查询被标记为可bypass的ANN查询类型。

本文为ANN查询设计了专门的计划节点，其关键属性包括：基础扫描信息（目标表、分区信息等）、索引标识（指定使用的HNSW/IVFFlat/DiskANN索引）、过滤条件（ANN查询中通常为空）、排序表达式（ORDER BY中的距离函数表达式）、扫描方向（按距离从近到远）、以及搜索数量与LIMIT值（控制HNSW搜索的ef_search参数和返回行数）。与普通的索引扫描节点相比，ANN索引扫描节点增加了搜索数量和LIMIT值两个字段，用于在索引扫描层面直接控制HNSW搜索的参数，避免了在LIMIT节点层面进行额外的行数截断。

3.2.2 向量检索轻量化执行路径

当查询被识别为可bypass的ANN查询后，OpFusion框架创建专用的向量检索执行对象，其执行流程分为准备、扫描和完成三个阶段。

在准备阶段，系统从查询参数中解析LIMIT和OFFSET值，并根据LIMIT值动态计算ef_search参数。这是一个关键设计：当LIMIT值大于默认ef_search时，自动将ef_search提升到LIMIT值，确保搜索结果的完整性；当LIMIT值较小时，使用默认ef_search以保证召回率。这种动态调整使用户无需手动调优搜索参数。

在扫描阶段，系统直接打开目标表和HNSW索引，构建扫描键并初始化索引扫描。随后循环调用索引扫描接口获取最近邻结果，进行属性映射后发送给客户端。若存在OFFSET，先跳过前N行结果。

在完成阶段，系统关闭索引扫描，释放表和索引的访问锁，设置查询完成状态。

传统执行路径与SQL bypass路径的对比如图5所示。通过上述轻量化执行路径，向量检索查询的执行器调用层级从传统路径的约15层压缩到约5层，实现了端到端延迟优化。

4 实验分析

本章通过对比实验验证第3章提出的两项优化的有效性。实验以优化前的openGauss 6.0.0作为基线，分别评估内存

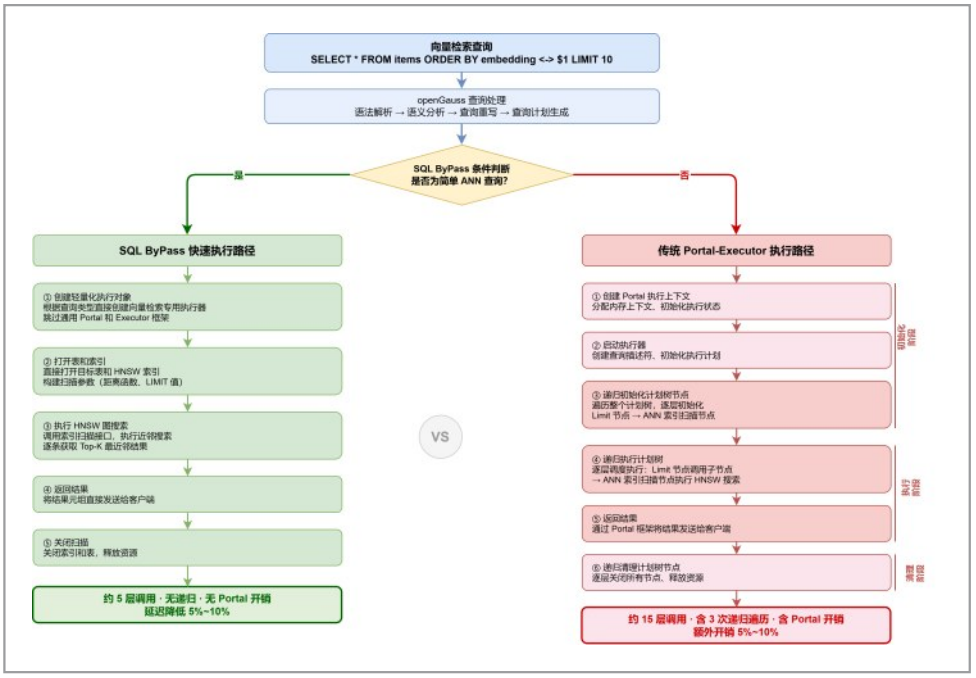

图5　传统执行路径与SQL bypass路径的对比

亲和布局优化（mmap）、SQL bypass执行路径优化以及两者叠加后的性能提升效果，从每秒查询数（queries per second，QPS）、平均延迟和P99延迟（第99百分位延迟）三个维度，在不同并发度下进行系统性评估。

4.1 实验环境与配置

实验在相同的硬件环境与数据集下进行，表2列出了实验环境的详细配置。

实验采用控制变量方法，重点考察mmap和SQL bypass在不同并发度下的独立贡献及协同效果。实验采用HNSW索引，构建参数*M=16*（每个节点的最大出边数）、*ef_construction=64*（构建时的候选集大小），查询参数*ef_search=100*（查询时的候选集大小），L2欧氏距离度量，执行Top-10近邻检索。实验在并发=1、4、8、16四种场景下分别测试。为保证结果的可靠性，独立运行三次，轮数分别为50、100、300轮，合计450轮，每轮每

表2　实验环境配置

配置项	详细信息
操作系统	openEuler 22.03
处理器	鲲鹏920(192核,2.6GHz,8 NUMA节点)
内存	724GB
数据库版本	openGauss 6.0.0
数据集	Cohere wikipedia-2023-11-embed-multilingual-v3 (simple子集)、50万条、1024维

个线程执行100条查询。每轮平均延迟为该轮所有请求延迟的算术平均值；每轮P99延迟采用NumPy默认线性插值方法计算。实验汇总时，先对每次运行中的轮级指标取平均，再按50、100、300轮的轮数加权合并。因此，本文报告的P99延迟为轮级P99的加权均值，并非将所有请求延迟合并后计算的总体P99，下文简称P99延迟。QPS按每轮完成的查询数除以各线程累计查询耗时的最大值估算，再按轮数加权汇总。实验设置了以下四组对比配置：

（1）基线：openGauss 6.0.0，采用传统buffer pool存储访问路径和Portal-Executor递归执行路径；

（2）仅bypass：在openGauss 6.0.0基础上启用SQL bypass；

（3）仅mmap：在openGauss 6.0.0基础上启用内存亲和布局优化，执行路径保持传统模式；

（4）完整优化：在openGauss 6.0.0基础上同时启用两项优化。

本文按轮数加权汇总测试结果，报告各配置在给定环境与负载下的性能变化。由于独立运行次数有限，结果采用描述性分析，不进行统计显著性推断。

4.2 实验结果

实验结果如下图6、图7以及表3所示:

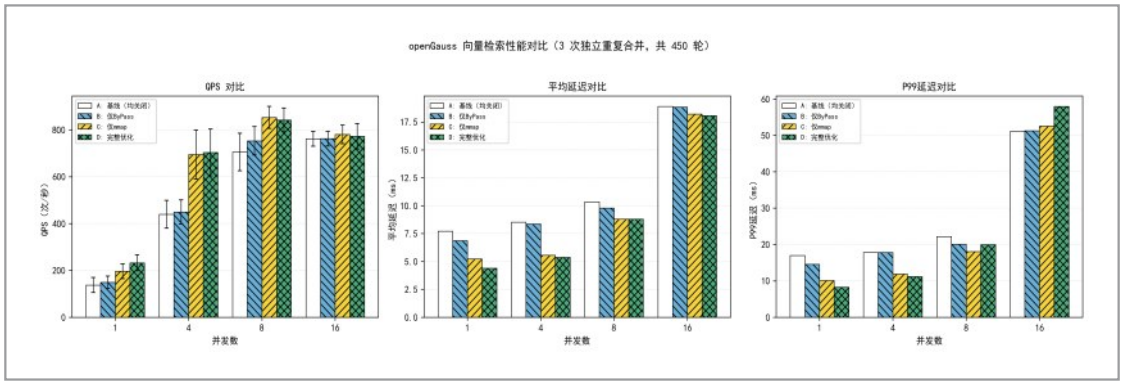

图6 四种配置在不同并发度下的性能对比(QPS、平均延迟、P99延迟)

QPS误差棒表示轮级QPS的加权总体标准差;平均延迟和P99延迟按第4.1节所述方法汇总。

对于数据规模、向量维度、Top-K参数、HNSW构建参数以及不同过滤选择率下的多维度、多规模组合实验，本文尚未完成系统实测。因此，文中关于页面访问次数、索引扫描耗时和单次页面获取代码路径开销的结论，仅适用于上述固定负载条件，不应直接外推到其他数据规模或检索参数。后续工作将针对数据规模、向量维度、Top-K和过滤条件开展分层控制变量实验，以建立更完整的开销模型。

4.2.1 内存亲和布局的独立贡献

对比"基线"（A组）与"仅mmap"（C组）两组数据可以量化内存亲和布局的独立贡献。本次实验结果表明，仅启用mmap即可在中低并发级别下带来显著的性能提升：并发1时QPS提升43.1%，平

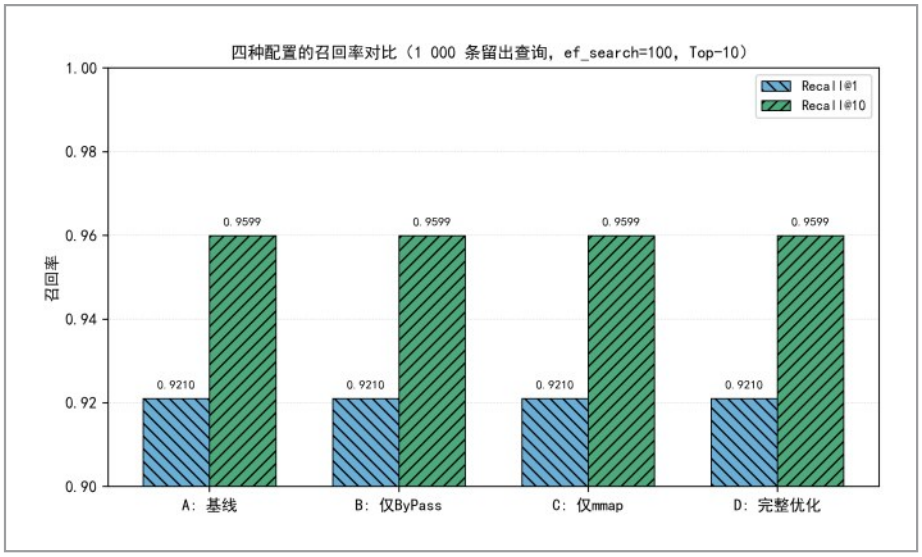

图7 四种配置召回率对比

表3 内存亲和布局对页面访问路径的影响

配置	经缓冲区管理器的页面访问数	其中磁盘读	索引扫描耗时(ms)	单次查询总耗时(ms)
关闭mmap	1399	0	3.959	4.217
开启mmap	97	0	1.933	2.141
变化幅度	−93.1%	—	−51.2%	−49.2%

同一查询在两种配置下分别执行5次，剔除首次冷启动后取中位数；ef_search=100，Top−10。

均延迟降低32.1%，轮级P99延迟均值降低40.5%；并发4时提升幅度最大，QPS提升58.1%，平均延迟降低34.7%，P99降低33.2%；并发8时QPS提升20.9%，平均延迟降低14.9%，P99降低18.5%。为定位性能提升的来源，本文对同一查询在两种配置下执行EXPLAIN (ANALYZE, BUFFERS)对照测量，结果如表3所示。关闭mmap时，单次Top-10查询的索引扫描需经缓冲区管理器的页面访问计数为1399次，且该次测量未记录到缓冲区管理器路径上的磁盘读取。启用mmap后该计数降至97次，降幅93.1%，仅保留元数据页与结果元组的获取，索引扫描耗时由3.959 ms降至1.933 ms，降幅51.2%。由于两种配置下均无磁盘I/O，上述结果与减少缓冲区管理路径开销的预期一致；据此折算，该对照条件下，每次减少的缓冲区页面访问所对应的耗时差约为1.56μs，仅为平均摊销估计。

4.2.2 SQL bypass的独立效果

对比“基线”（A组）与“仅bypass”（B组）可以评估SQL bypass的独立贡献。实验结果显示，SQL bypass在各并发级别下均带来小幅但可测的改善：并发1时QPS提升8.8%，平均延迟降低10.9%，P99降低14.4%；并发4时QPS提升2.3%，平均延迟降低1.9%；并发8时QPS提升6.8%，P99降低9.6%；并发16时QPS提升约0.1%，平均延迟降低0.3%。观测到

的QPS提升幅度总体小于仅mmap配置，原因在于向量ANN查询的执行热点集中在HNSW图遍历和高维距离计算上，执行器递归调用的开销占比较小，bypass跳过的Portal创建与Executor递归初始化步骤在总查询时间中贡献有限。

4.2.3 两项优化的协同增益

对比“仅mmap”（C组）与“完整优化”（D组）可以量化两项优化的协同效应。在mmap基础上叠加SQL bypass，并发1时QPS由196.5提升至232.4，增幅18.3%，平均延迟降低16.0%，P99降低18.4%；并发4及以上时两种配置的QPS差异在±2%以内。这揭示了两项优化之间的层次关系：当mmap消除了存储层的buffer pool瓶颈后，执行器递归调度的相对开销占比上升，此时SQL bypass才能有效发挥作用。这与4.2.2节中bypass单独效果有限的观察形成对照——只有先解决存储层瓶颈，执行层优化才能显现价值。

综合而言，采用mmap内存亲和布局结合SQL bypass旁路优化的方案，相对基线在并发1与4时QPS分别提升69.2%与60.0%，平均延迟分别降低43.0%与36.9%，轮级P99延迟均值分别降低51.5%与36.9%；并发8时QPS提升19.4%，P99降低10.0%。并发升至16时，四种配置的QPS均收敛至762至780、平均延迟均落在18.1至18.9ms，整体吞吐与平均响应时间已经趋于饱和。然而，完整优化配置的P99延迟为57.91 ms，高于基线配置的51.10 ms，增幅为13.3%。这表明在高并发饱和阶段，完整优化配置的QPS和平均延迟与基线接近，但少数查询出现了更明显的尾延迟。

这一现象与mmap访问路径的运行机制相符。buffer pool路径中的页面访问由数据库缓冲区管理器统一维护，而mmap路径需要由各数据库后端进程在自身虚拟地址空间内访问映射页面。对于尚未建立本进程页表映射的页面，首次访问可能触发软缺页处理和页表填充；在多个后端进程并发访问不同索引页面时，这类操作还可能与线程调度、内存分配以及非一致性内存访问（non-uniform memory access，NUMA）远端访问产生叠加影响。大多数查询不会同时触发较多此类事件，因此平均延迟变化有限；少数恰好访问较多未建立映射页面或受到调度等待影响的查询，则可能形成较长的响应时间，从而抬高P99指标。由于表3显示两种路径均未产生磁盘读取，因此这里所讨论的缺页更可能属于不涉及磁盘I/O的软缺页或页表建立开销，而不是磁盘加载导致的主要缺页。

需要说明的是，本文现有实验未对每个数据库后端进程的缺页次数、上下文切换次数和硬件缓存事件进行独立采集，因此上述分析是基于观测到的尾延迟变化、mmap实现机制以及已有页面访问结果作出的机制解释，不能将P99升高完全归因于单一的缺页事件。高并发下的CPU调度、NUMA访存和系统资源竞争也可能共同影响尾延迟。后续工作将通过进程级缺页统计和硬件性能计数器进一步验证不同访问路径的尾延迟来源。

5 总结

本文针对大模型训推一体场景下，关系型数据库面向高维向量检索时存在的两大系统级性能瓶颈——buffer pool的冗余存储访问开销与执行器的递归调度开销，在openGauss内核中提出并实现了两项相

互正交的优化方案。其一是基于mmap的内存亲和索引布局，将HNSW索引文件直接映射到进程地址空间，把页面访问退化为指针解引用，消除了哈希查找、引用计数与轻量级锁开销；其二是基于OpFusion框架的SQL bypass快速执行路径，通过识别简单ANN查询跳过Portal创建与Executor递归初始化，将执行器调用层级由约15层压缩到约5层。

在50万条1024维Cohere文本嵌入向量、Top-10检索场景下开展实验验证，实验独立重复三次共450轮。执行计划对照测量表明，内存亲和布局使经缓冲区管理器的页面访问由1399次降至97次，索引扫描耗时降低51.2%；按索引扫描耗时差与页面访问计数差折算，每次减少的缓冲区页面访问对应约1.56 μs的平均摊销耗时差。性能测试表明，两项优化叠加后，并发1与4时检索吞吐分别提升69.2%与60.0%，平均延迟分别降低43.0%与36.9%，P99尾延迟分别降低51.5%与36.9%；并发8时吞吐提升19.4%。并发升至16时四种配置的QPS与平均延迟差距缩小，推测系统已达饱和、瓶颈由页面访问转移至与存储路径无关的资源，优化收益因而被抹平。在全部配置下，1000条留出查询返回结果与基线逐条完全一致，Recall@10均为0.9599，验证了两项优化不改变检索语义、无精度损失。

未来工作将从三个方向继续展开：一是结合NUMA感知的内存分配策略，进一步优化多核高并发下的访存局部性，并针对饱和阶段的资源瓶颈开展定位；二是将SQL bypass路径扩展到带WHERE过滤与JOIN的混合查询场景，支持更复杂的检索需求；三是优化按页组映射的粒度选择并研究向量索引增量更新与mmap映射的协同机制，以覆盖高频写入场景下的内存亲和优化。

参考文献：

[1] Pan J J, Wang J, Li G. Survey of vector database management systems[J]. The VLDB Journal, 2024, 33(5): 1591-1615.

[2] 宋子文,王斌,张喜瑞,等. 向量数据库中近似最近邻搜索关键技术综述[J]. 软件学报, 2026,37(3):971-1005.
SONG Z W, WANG B, ZHANG X R, et al. Survey on key techniques of approximate nearest neighbor search in vector databases[J]. Journal of Software, 2026, 37(3): 971-1005.

[3] Wang J, Yi X, Guo R, et al. Milvus: a purpose-built vector data management system[C]//Proceedings of the 2021 International Conference on Management of Data, June 20-25, 2021, Xi'an, China. New York: ACM, 2021: 2614-2627.

[4] Yang W, Li T, Fang G, et al. PASE: PostgreSQL Ultra-High-Dimensional Approximate Nearest Neighbor Search Extension[C]//Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data, 2020. DOI: 10.1145/3318464.3386131.

[5] Huang A, Guan J, Cai J, et al. Industrial-strength OLTP using main memory and many cores[J]. Proceedings of the VLDB Endowment, 2020, 13(12): 3099-3111.

[6] Huawei Technologies Co., Ltd. Database principles and technologies: based on Huawei GaussDB[M]. Singapore: Springer, 2023.

[7] Li W, Zhang Y, Sun Y, et al. Approximate nearest neighbor search on high dimensional data: experiments, analyses, and improvement[J]. IEEE Transactions on Knowledge and Data Engineering, 2020, 32(8): 1475-1488.

[8] Jegou H, Douze M, Schmid C. Product quantization for nearest neighbor search [J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2011, 33(1): 117-128.

[9] Gao J, Long C. RaBitQ: quantizing high-dimensional vectors with a theoretical error bound for approximate nearest neighbor search[J]. Proceedings of the ACM on Management of Data, 2024, 2(3): Article 167.

[10] Malkov Y A, Yashunin D A. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs[J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2020, 42(4): 824-836.

[11] Zhang Q, Xu S, Chen Q, et al. VBASE: Unifying Online Vector Similarity Search and Relational Queries via Relaxed Monotonicity [C/OL]//17th USENIX Symposium on Operating Systems Design and Implementation. Boston: USENIX Association, 2023:377-395.

[12] Gollapudi S , Karia N , Sivashankar V ,et al. Filtered-DiskANN: Graph Algorithms for Approximate Nearest Neighbor Search with Filters[C]//Proceedings of the ACM Web Conference 2023, 2023. DOI:10.1145/3543507.3583552.

[13] Sun J, Li G, Pan J, et al. GaussDB-Vector: A large-scale persistent real-time vector database for LLM applications[J]. Proceedings of the VLDB Endowment, 2025, 18(12): 4951-4963.

[14] Crotty A, Leis V, Pavlo A. Are You Sure You Want to Use MMAP in Your Database Management System? [C]//CIDR 2022, Conference on Innovative Data Systems Research. 2022.

[15] Liu M, Kang J, Wang K, et al. ScaleCache: Scalable and Production-Grade Buffer Management for Disk-Based Database Systems[J]. Proceedings of the VLDB Endowment, 2025, 18(12): 5073-5085.

[16] Subramanya S J , Devvrit F , Simhadri H V ,et al.DiskANN: fast accurate billion-point nearest neighbor search on a single node[C]//Advances in Neural Information Processing Systems. Vol. 32. Curran Associates, Inc., 2019.

[17] 周依杰,林圣原,巩树凤,等. GoVector:I/O-高效的高维向量近邻查询缓存策略[J]. 软件学报,2026,37(3):1021-1036.
Zhou Y J, Lin S Y, Gong S F, et al. GoVector: I/O-efficient caching strategy for high-dimensional vector nearest neighbor search[J]. Journal of Software, 2026, 37(3): 1021-1036.

[18] 邱海浪,彭煜玮,彭智勇. LSMDiskANN:更新友好型磁盘向量索引框架[J]. 软件学报,2026,37(3):1058-1083.
Qiu H L, Peng Y W, Peng Z Y. LSMDiskANN: update-friendly disk-resident vector index framework[J]. Journal of Software, 2026, 37(3): 1058-1083.

[19] Wei C, Wu B, Wang S, et al. AnalyticDB-V: A Hybrid Analytical Engine Towards Query Fusion for Structured and Unstructured Data[J]. Proc. VLDB Endow., 2020, 13(12): 3152-3165.

[20] Li L , Gong S , Yang Y ,et al.I/O Optimizations for Graph-Based Disk-Resident Approximate Nearest Neighbor Search: A Design Space Exploration[J].Proceedings of the VLDB Endowment, 2026, 19(7): 1484-1498. DOI: 10.14778/3801059.3801064.

作者简介



盛凌志（1976– ），男，博士，工程技术系列国务院特殊津贴专家，正高级工程师，中国移动通信集团北京有限公司重要客户中心副总经理，主要研究方向为移动通信、大数据、行业数字化创新。



赵淳（1981– ），男，硕士，中国移动通信有限公司数智事业部基础平台研发中心副总经理，主要研究方向为云计算、数据库以及AI领域的产品能力研发、推广和落地应用。

录用日期: XXXX-XX-XX
通信作者:
基金项目:
Foundation Items: