Computer Science and Application 计算机科学与应用, 2024, 14(1), 68-77
Published Online January 2024 in Hans. https://www.hanspub.org/journal/csa
https://doi.org/10.12677/csa.2024.141008

Hans汉斯

一种基于异步I/O的磁盘向量检索算法

吴松林[1]，田春岐[1]，毕枫林[2]

[1]同济大学电子与信息工程学院，上海
[2]华东师范大学数据科学与工程学院，上海

收稿日期：2023年12月22日；录用日期：2024年1月19日；发布日期：2024年1月30日

摘　要

在机器学习与深度学习等领域中，近似最近邻搜索(ANNS)扮演着至关重要的角色，在近十多年来受到越来越多研究者的关注。传统的ANNS算法需要将向量原始数据和索引数据全部存储在内存中，这限制了其处理大数据的能力。本文提出了一种创新的基于异步I/O的磁盘向量近似最近邻搜索算法(AIO-ANN)，该算法通过生成非阻塞I/O请求并即刻处理完成的请求，有效提升了搜索效率并降低了高延迟I/O请求的负面影响。在搜索过程中，AIO-ANN生成一批非阻塞I/O请求，并立即处理每个完成的I/O请求，而不是等待整批请求完成。同时为了最大化I/O等待时间的利用，算法将大部分计算任务转移到了I/O等待期间。此外，算法还整合了其他优化措施，如数据缓存与结果初始化。在大规模数据集上的实验，证明了AIO-ANN在搜索速度上超越了主流的ANNS算法DiskANN。

关键词

向量检索，异步I/O

An Asynchronous I/O-Based Disk Vector Retrieval Algorithm

Songlin Wu[1], Chunqi Tian[1], Fenglin Bi[2]

[1]College of Electronic and Information Engineering, Tongji University, Shanghai
[2]School of Data Science & Engineering, East China Normal University, Shanghai

Received: Dec. 22nd, 2023; accepted: Jan. 19th, 2024; published: Jan. 30th, 2024

Abstract

In the fields of machine learning and deep learning, Approximate Nearest Neighbor Search (ANNS) plays a crucial role and has garnered increasing attention from researchers over the past decade.

文章引用: 吴松林, 田春岐, 毕枫林. 一种基于异步I/O的磁盘向量检索算法[J]. 计算机科学与应用, 2024, 14(1): 68-77.
DOI: 10.12677/csa.2024.141008

Traditional ANNS algorithms require storing all vector raw data and index data in memory, limiting their ability to process large datasets. This paper introduces an innovative Asynchronous I/O-based Disk Vector Approximate Nearest Neighbor Search algorithm (AIO-ANN), which enhances search efficiency and reduces the negative impact of high-latency I/O requests by generating non-blocking I/O requests and immediately processing completed requests. During the search process, AIO-ANN generates a batch of non-blocking I/O requests and processes each completed request immediately, rather than waiting for the entire batch to complete. To maximize the utilization of I/O waiting time, the algorithm shifts the majority of computational tasks to these waiting periods. Additionally, the algorithm incorporates other optimization measures, such as data caching and result initialization. Experiments on large-scale datasets have proven that AIO-ANN surpasses mainstream ANNS algorithms like DiskANN in search speed.

Keywords

Vector Search, Asynchronous I/O

Copyright © 2024 by author(s) and Hans Publishers Inc.
This work is licensed under the Creative Commons Attribution International License (CC BY 4.0).
http://creativecommons.org/licenses/by/4.0/
Open Access

1. 引言

最近邻搜索(Nearest Neighbor Search, NNS)是信息检索、模式识别、数据挖掘和推荐系统等领域的基础算法[1]。然而随着数据规模的持续增长和以及维度增长带来的维度灾难，现有的 NNS 算法已经不适用于大规模数据集与低延迟的场景。因此，大多数研究者提出了近似最近邻搜索(ANNS)以平衡查询精度和效率。现有的 ANNS 方法可以分为两个过程：离线索引构建和在线查询。在大部分算法的实现中，为了高效的索引构建和极低的查询延迟(通常在 5 ms 以内)，索引构建和查询过程都需要在内存中进行。

随着向量数据的增多，数据规模可以达到数亿甚至十亿，例如 BigANN 基准测试中使用的数据集(https://big-ann-benchmarks.com/)，向量数据的规模接近十亿级别。对于 128 维的 uint8 数据，仅将所有向量数据加载到内存中就需要 119 GB 的物理内存，这使得纯内存方案变得十分昂贵，尤其是当下流行的 CPU 也仅仅支持 128 G。

为了解决这个问题，一种思路是将数据分组划分为许多切片。每个切片可以部署在一个单独的物理机上，从而构建一个分布式系统。当查询请求到来时，相同的查询应用于每台机器上，最后汇总获得全局的结果。这种方法在解决大规模数据问题很受欢迎。然而，它有一个致命缺点，每一个查询必须应用于集群中所有机器，这意味着增加机器数量不会提高查询速度。并且，这种方案也很难做到负载均衡，因为每一个查询必须应用到所有机器上。而另一个处理大规模向量数据检索的方案，则是使用高性能的固态硬盘(SSD)代替内存(DRAM)降低成本。SSD 的价格比 DRAM 低许多，而且 SSD 可以存储 TB 级别的数据，这比 DRAM 的容量大数千倍。更重要的是，随着 PCIe 技术的进步，SSD 在过去的十年中已经变得越来越强大。最广泛使用的 PCIe 3.0 SSD 甚至可以实现 3000 MB/s 的带宽，这意味着使用 SSD 代替 DRAM 已经成为了潜在的可能。已经有 DiskANN [2]、SPANN [3]、Zoom [4]、GRIP [5]等多个过往的研究表明，使用 SSD 代替 DRAM 是完全可行的。

在本文中，我们介绍了一种新型基于固态硬盘(SSD)的近似最近邻搜索(ANNS)算法。该算法利用异步 I/O 框架 io_uring 来优化搜索延迟。通常，ANNS 算法包括两个阶段：索引构建和在线搜索。对于索引

构建阶段，我们的算法 AIO-ANNS 主要采用 DiskANN 的索引构建方法。在 DiskANN 的论文中指出，其图索引构建方有助于查询算法减少 SSD 的访问次数。在搜索阶段，我们提出了一个基于异步 I/O 的优化方法来降低搜索延迟。因为算法依赖的索引数据全部存放在 SSD 上，在搜索过程时，当需要访问一个索引结点时，必须要访问 SSD。访问一个向量索引节点对应着一个单独的 I/O 请求。虽然 SSD 具有内在并行性，但显然等待一批 I/O 操作同时完成的时间比等待一个单独的 IO 操作要更长。以往提出的基于 SSD 的算法，通常在搜索迭代阶段提交一批 I/O 请求，然后等待所有请求完成。相反地，我们使用 io_uring 将 I/O 请求拆分为两个阶段：提交阶段和完成阶段。通过异步 IO 方式，算法消除了等待所有 I/O 请求完成的不必要时间，从而极大缓解了高延迟 I/O 请求带来的不利影响。

2. 相关工作

随着数据规模的持续增长，向量数据集扩展到数十亿甚至上千亿的规模，向量维度达到数百甚至上千，传统的基于内存的算法常常不能满足索引构建和查询处理的要求。为了应对这一挑战，出现了两种主流解决方案：对于一个向量数据集合 $\mathbf{X} \in \mathbb{R}^{n\times m}$，其包含 n 个维度为 m 的向量数据。给定一个查询向量 $q \in \mathbb{R}^m$，搜索的目标是需要从 X 中找到与查询向量最近的向量 p^*，即 $p^* = \arg\min_{p\in X} Dist(p,q)$。这样的向量 p^*被称作最近邻。类似的，可以定义 k-最近邻问题。受制于暴力搜索的计算开销大和高查询延迟的缺陷，近似最近邻搜索被提出用于从底层数据库从检索出 k 个近似最近邻的向量，以满足低延迟和高吞吐的要求。对于给定查询 q，ANNS 算法会返回 k 个向量构成的集合 S，也被称为候选集。我们可以定义：

$$k\text{-}recall@k = \frac{|X \cap S|}{k}$$

ANNS 算法的目标是在尽可能低的延迟下取得最大的召回率。并且，算法的索引构建时间和索引大小也是必须要考虑的因素，因此算法不仅需要在 online 的查询上取得很好的效果，也需要满足 offline 快速构建的需求。

过去的几十年里，研究者提出过诸多数据结构和算法用于解决近似最近邻搜索问题。这些算法可以划分为四类：基于树的算法；基于哈希的算法；基于量化的算法；基于临近图的算法。近些年来，基于量化的算法和图的算法成为了 ANNS 算法的主流。特别是各种临近图的算法，在大部分数据集上都取了了最优的性能，但却没有带来更多的索引构建时间。

k-d 树是最早提出用于向量检索的算法，其思想类似于二叉搜索树。通过比较数据，把大于当前节点的数据放在树的右侧，小于节点的数据放于左侧，但是与二叉树不同的是，k-d 树大小的比较是在向量的维度上进行的。ball-tree 修改了 kd-tree 的划分方式，将划分的子空间从超长方体修改为球体。vp-tree 的划分不发生在向量的具体维度上，其使用距离度量对于整个向量空间进行考虑。Muja 等人[6]整合了一系列基于树的查询算法，推出了 flann 库。

基于哈希的算法目标是将高维向量编码为二进制向量，并且尽可能保留原本向量空间下向量之间的距离关系，主要可以分为局部敏感哈希[7]和哈希学习[8]。局部敏感哈希采用与数据无关的哈希函数，整个学习处理过程不依赖任何的数据内容。LSH 通过一个局部敏感的哈希函数将相似的数据点以更高的概率映射到相同的哈希编码上。查询时，只需要在遍历查询向量落入的哈希桶中所有的向量，就可以找到最近邻。哈希学习通过机器学习机制将数据映射为二进制串的形式，能够显著减少数据的存储和通信开销，从而提高学习系统的效率。

向量量化(Vector Quantization, VQ)是信息论中进行数据压缩的传统算法[9]，其目的是降低数据维度但尽可能保持向量数据原始分布。乘积量化(Product Quantization, PQ)在向量量化的基础上发展而来，jegou

等人[10]首次将 PQ 用于解决 ANN 问题。PQ 算法的主要思想是对原始向量进行维度切分，对切分后得到的子向量进行向量量化后，再使用笛卡尔积进行组合。通常使用 PQ 算法有很多优点：1) PQ 对原始向量进行压缩，使得向量数据可以全部存放在内存中；2) 查询向量和压缩向量之间的距离可以通过查表方式快速计算得到。3) PQ 算法的实现和数据节点都很简单，这使得其可以与其他算法组成混合算法。因为 PQ 对子空间的切分没有考虑到向量数据的分布，划分完全与数据无关，但是现实中大部分数据集不同的维度分布是不均匀的，Ge 等人提出了 Optimized Product Quantization 也就是 OPQ 来保留向量的原始分布，其做法是简单描述是通过一个正定矩阵 R 对于原始的向量空间进行了旋转[11]。Ge 等人同时还提出了 Locally Optimized Product Quantization (LOPQ)来进一步优化 OPQ 算法[12]。量化通常和倒排索引结合使用，例如 IVFADC [13]、FAISS [14]、IVFOADC + G + P [15]以及 SCANN [16]。通常这些算法会使用 kmeans 或者其他聚类算法，将整个向量数据集切分为多个分区，然后对于每个分区内的向量进行 PQ 操作。在查询时，仅仅需要对少量最近的分区进行访问，变量分区内的所有向量，通常每次查询访问的分区数远小于总的划分分区数。通常只需要少于 64 GB 的内存就可以存放十亿的压缩向量，并且 fassi 可以做到小于 5 ms 查询延迟。但是根据 ANN benchmark 的测试结果，这些算法在 1-recall@1 取得相当糟糕的性能，并且在高 Recall (通常指 99%以上)其查询速度急剧衰减。

最近十年内，取得了最优性能的 ANNS 算法通常都是基于临近图这一数据结构[17]。基于图的 ANNS 算法会使用原本的数据集(a)构建一个图索引(b)，图上的节点对应原始数据集中的向量。其查询算法通常使用贪心算法(Best-First-Search)，每一步选择最优节点从入口点逐渐逼近查询点。Malkov 最先提出了基于插入增长式的构建索引算法(Navigable Small World Graphs, NSW) [18]，基于 NSW 算法，Malkov 进一步提出了更高效的算法——Hierarchical Navigable Small World Graphs (HNSW) [19]。HNSW 在选择节点邻居时，采用 relative neighbour graph 中的启发式算法，并且构建分层的图索引，使得算法可以高效地进行贪心查询。直到今天，HNSW 仍然是大部分 ANNS 场景下的第一选择，算法健壮性得到广泛认可。由浙大和阿里团队提出的 NSG (Navigating Spreading-Out Graph)算法同样取得了优异的性能表现[20]，不同于 HNSW 增长插入的构建方式，其使用 KGraph 构建初始图，并在初始图上进行优化。Wang 等人[1]对基于图的 ANNS 算法进行了深入且全面的综述，涵盖了大部分主流图算法。

这些算法在该领域取得了显著的进展，为处理越来越大和复杂的数据集提供了新的策略和解决方案。但是随着数据规模的不断增长，向量数据集的尺度可以扩展到亿级甚至十亿级别，并且向量的维度也达到上百维甚至上千维。传统的纯内存算法已经不能满足构建和查询需求，甚至算法无法工作。为了解决这一问题，产生了两种主流的处理方案。1) NSG 论文中提出的分布式解决方案。类似于分布式数据库的处理办法，将数据进行分片，在一台服务器上只对一个分片进行索引构建和查询执行。2) 以 DiskANN 算法为代表的一系列使用内存和磁盘混合的算法。

通常，向量数据分片的主流处理办法是随机划分，包括当下最成熟的分布式向量数据库 milvus 也是采样类似的处理。对于单个 query 而言，需要在集群中所有的机器都执行一遍查询。这无疑与分布式的思想相违背——对集群的扩展并没有带来速度的提升。因此使用磁盘和内存的混合算法得到了更多企业和研究人员的关注。

这里的提到的磁盘通常是数据库中广义的非易失性存储介质，不仅仅代表 HDD，还包括 SSD 或者傲腾非易失性内存。微软也许在业务场景中对于混合算法有着迫切的需求，提出了一系列基于磁盘的 ANNS 算法。在 2018 年，微软提出了 Zoom 算法[4]，其思想很简单，将数据划分为需要很小的分区，每个分区的中心点组成的集合，使用 HNSW 进行索引构建，并全部放入内存中，而每个分区中的向量经过 PQ 量化后存放到 SSD 上。这样查询就变成了一个两阶段过程。次年，微软再一次发布了改进后的算法 GRIP [5]，和 Zoom 算法类似，但是 GRIP 将 PQ 量化后的压缩向量也存放到内存中，仅仅在 SSD 上存放

全精度向量。在查询时，先通过内存导航图找到对应的 partition，然后使用 PQ 向量获得 PQ 距离，再使用 PQ 距离对候选向量进行排序，从 SSD 上拉取 PQ 距离最近的全精度向量后，再使用全精度向量与查询向量的距离进行重排。这类算法往往在 recall@1 取得不错的效果，但是在大 topK 的情况下，算法几乎丧失优势。

2020 年，微软提出了 DiskANN 算法，与 Zoom 和 GRIP 算法完全不同，DiskANN 使用数据集中全部向量构建 Vanama 索引(Vanama 和 NSG 只有轻微区别，它的构建步骤与 NSG 类似，不同于 NSG 以 K 最近图为初始图，在其论文中直接使用了随机图作为初始图，并使用一些启发式算法构建最终图索引)，并且构建好的图索引邻接表和全精度向量共同存放到 SSD 上。并且 DiskANN 还会同时训练 PQ，来对所有向量进行量化，并将压缩后的向量全部存放在内存中。搜索时，其先用 PQ 压缩向量计算，索引节点向量与 query 向量的 PQ 距离。根据 PQ 距离大小从小到大排序，取前 beam 个离 query 向量最近的节点。从 SSD 上读取其对应的 entry，这样我们就获得了该节点对应的全精度向量数据，以及其邻接表。我们使用全精度向量重新计算距离，作为最终查询结果的候选。而邻接表中记录的邻居，则计算这些邻居的 PQ 向量与 query 向量的距离，并重新排序，从 SSD 继续拉取新的 entry。直到满足终止条件。

Chen 等人在 2021 年提出了一种基于 kmeans-tree 的新型算法 SPANN，在内存中存放导航图，而在 SSD 上存放经过 kmeans 划分后的 partition，每个 partition 中含有多个全精度向量。Zilliz 的 Search 团队同样研发了一种混合算法 BBAnn [21]，其与 SPANN 类似也是使用 kmeans 进行聚类，然后在内存中构建导航图，全精度向量放在 SSD 上。

3. 研究动机

基于图的 ANNS 算法可以分为构建索引阶段和搜索阶段，并且这些算法大部分在搜索阶段都使用了一种 greedy search 的搜索策略，即贪心搜索。greedy search 会维护一个固定长度的侯选池来存储需要遍历的向量，并且根据查询向量和这些向量之前的距离进行排序。算法迭代式地从侯选池中检索获得最近的未访问结点，并且将其标记为已访问。接着，算法会检查此结点在图索引上的所有邻居节点，计算邻居结点所代表的向量与查询向量之间的距离。并使用(邻居向量，距离)对来更新侯选池。如果侯选池所有的结点都得到了访问，算法就终止。这种搜索模式会产生大量访问 SSD 的往返操作(round-trip)，其中一个往返操作涉及 OS 提交一个 I/O 请求到 SSD，然后 SSD 将数据返回到 OS。因此，我们使用 beam search 来尽可能地访问 SSD 的 round-trip 次数。如图 1 所示，beam search 与 greedy search 的不同点在于，beam search 每一次从侯选池中获取多个需要访问的结点，将原本多次小的 IO 请求合并一个大的 IO 请求，从而减少了 round-trip 的数量。虽然 beam search 最终需要读取的数据比 greedy search 要多一些，但是由于 SSD 本身具有内在并行性，在 SSD 上执行时，beam search 的速度会显著超过 greedy search。

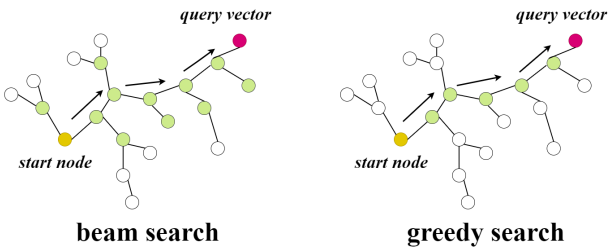

Figure 1. Beam search compares with greedy search
图 1. Beam search 与 greedy search 的对比

在许多场景中，beam search 已经可以取得非常优秀的性能，但是在图 1 遍历过程中，揭示了 beam search 一个显著的缺点。即 beam search 每一次需要提交一批读请求，并且等待这一批请求全部完成。这

使得一次 round-trip 的延迟会由最慢的那个 IO 请求所决定。如果延迟出现不一致情况，则会显著伤害系统的性能。遗憾的是，这种情况在固态硬盘(SSD)、硬盘驱动器(HDD)和网络环境中经常出现。大多数 IO 操作都执行得很快，但少数几个可能需要异常长的时间，导致所谓的拖尾现象。

我们设计了一个针对 SSD 实验，在实验中我们制造了大量的随机读请求，并且记录了每个读请求完成的时间。结果完全符合我们的猜测：虽然 98.5%的读请求的完成延迟低于 100 微秒，但最慢的请求却需要长达 10,000 微秒，结果如图 2 所示。因此，为了提高基于 SSD 的近似最近邻(ANNs)算法的性能，减少这些高延迟请求的影响变得至关重要。受到流水线概念的启发，我们提出了一种策略来改善这种效应。算法不再等待一个整批的 IO 请求完成，而是在每个请求完成时处理它——计算距离、更新候选池，然后继续处理下一个请求，直到批中的所有请求都完成。

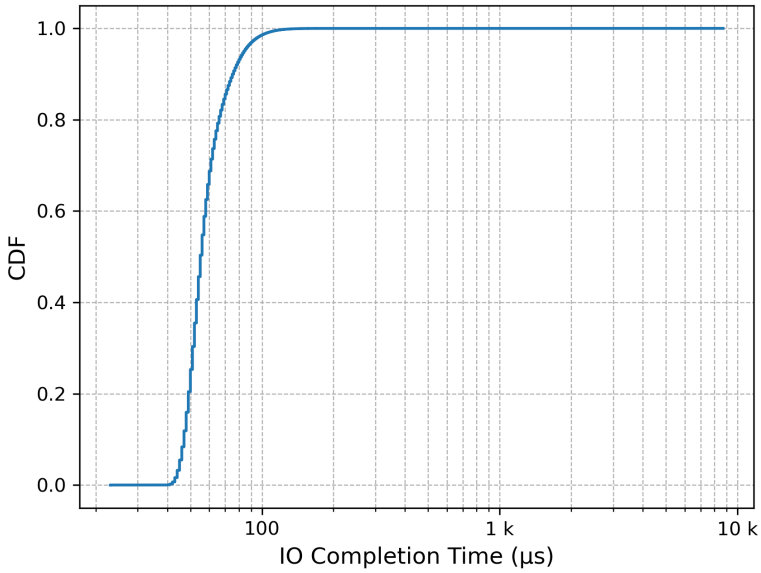

Figure 2. CDF of I/O completion time
图 2. I/O 完成时间的分布曲线图

此外，大多数 ANNs 算法通常涉及两个池：一个用于导航的候选池，按照 Product Quantization (PQ)压缩距离排序，以及一个按照全精度距离排序并作为最终结果呈现给用户的结果池。对于导航至关重要的候选池必须在下一次 beam search 迭代之前更新。另一方面，结果池不对搜索路径做出贡献，因此可以在搜索的任何时候进行更新。通过解耦这两个组件，允许在 IO 请求等待时间内更新结果池，可以提高算法的整体效率。

4. 算法细节

在之前的部分已经提到，AIO-ANN 算法很大程度收到了 DiskANN 算法的启发，因此在索引结构上 AIO-ANN 与 DiskANN 算法保持一致。我们使用两轮构建的 Vanama 图作为我们的图索引，并且整个图索引数据全部放置在 SSD 上，而不是 DRAM 中。在 beam search 的过程中，我们在图上进行游走，迭代式地从 SSD 中读取图索引数据以及向量原始数据。这个过程也是整个算法主要耗时部分。此外，我们还采用了 BBAnn 中提到的分层 kmeans 技术来优化我们的索引在 SSD 上的数据布局。BBAnn 是一种主要设计用于范围检索的算法，它使用分层均衡 kmeans 来将数据切分为多个簇。我们将这一技术与 DiskANN 相结合，有助于优化数据布局。在内存中，我们使用 PQ 量化后的压缩向量来代替原始向量，这可以为我们的算法节省 4 到 16 倍的内存成本。算法具体的架构如图 3 所示。

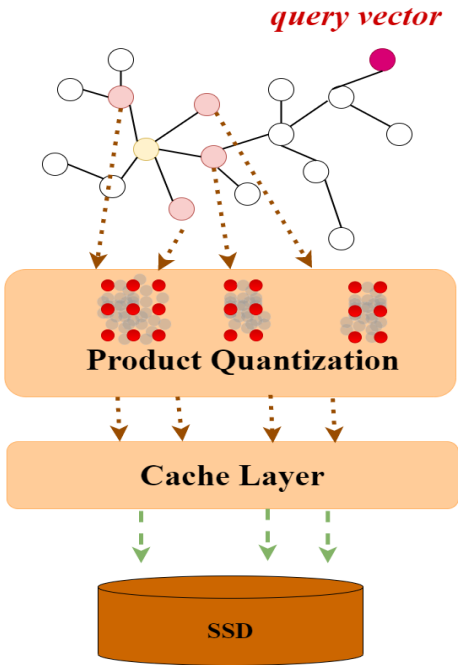

Figure 3. The architecture of AIO-ANN
图 3. 算法具体架构

在研究部分的延迟实验表明，SSD 的读请求延迟并不稳定，并且受到拖尾效应的影响，这对算法的性能是有害的。此外，实现还暗示了一个有前景的策略——通过解耦候选池和结果池的更新，可以在 IO 请求完成期间隐藏更新结果池的时间。在图 4 的算法 1 中详细描了这一过程。

AIO-ANN 将每一个 IO 请求过程分解为两个部分，分别为提交过程与完成过程。在提交过程，算法会一次性提交多个读请求，并且不会阻塞住，而是立刻返回提交的结果。在完成过程中则恰恰相反，算法会阻塞住，直到有 IO 请求完成并返回。然而，在 linux 的传统的文件操作中是不存在这样的系统调用的，例如 read 或者 write 都必须阻塞直到完成。因此我们使用 io_uring 这一工具来实现我们的目的。io_uring 是 linux kernel 提供的最新文件操作 API，它可以支持各种异步操作。Io_uring 使用了一个共享的环形缓冲队作为内核与用户空间通信的方式，极大地降低了系统调用的开销。

在算法的实现代码中使用了 IORING_SETUP_SQPOLL 标志用于启用 iouring 的内核线程模式。在这种模式下，提交阶段与完成阶段都不涉及任何系统调用。在提交一批 IO 请求与等待任意一个 IO 请求完成之间，我们执行结果池的更新。这一过程包括了计算向量全精度距离，并根据这个结果重排最终的 topk 结果。通过这一优化策略，可以显著消除由长尾效应带来的负面影响。

在数据库领域还有一种常用的减少 IO 的策略，那就是内存缓存。在 ANNs 领域中同样可以实现类似的缓存机制，与数据库的缓存策略不同的是，数据库通常缓存的最小单位是内存页，而 ANNs 的缓存通常是缓存一个索引结点，包括这个索引结点的邻接表与原始向量。在实际搜索过程中，如果缓存命中，那么就可以减少 IO 请求。考虑到每个查询的 IO 请求数量通常不会超过 100，如果缓存次数命中足够多，会带来巨大的性能提升。

我们设计的算法具体架构如图 3 所示，在最上层我们使用了 DiskANN 提出的 Vanama 索引结构图，在 Vanama 图上从起始点出发上使用 beam search，迭代方式寻找最终结果。在每一步迭代过程，算法使用压缩 PQ 向量来计算近似距离，用于指导生成访问 SSD 的请求。在 SSD 上层，算法添加了 Cache 层，用于降低 I/O。如果 cache 不命中，才会真正去访问 SSD。

最后算法还采用了 result-based 技术来优化 ANNS 中初始选点的步骤。在实际的应用场景中，向量查询通常表现出局部性，这意味着一批查询向量在短时间内可能具有相似的特征。因此，我们使用了前一个查询的结果来初始化当前的侯选池，可以加速算法路由过程。

Algorithm 1 AIO-ANN 算法

Require: 内存中的 PQ 压缩向量, SSD 上索引 G, query q, 侯选池 C, 结果池 R, IO 请求数 N

Ensure: q 的 top-k 最近邻

使用上一次查询结果初始化 C
while C 有未被访问的结点 **do**
  $U \leftarrow C$ 中 top-N 未被访问结点
  提交 N IO 请求到 SSD
  使用全精度向量距离更新 R
  while 有 IO 请求没有完成 **do**
    $u \leftarrow$ 等待一个 IO 请求完成并返回
    for u 中未被检查的邻居 e **do**
      使用 $(\mathrm{id}(e), \mathrm{dist}_{PQ}(e, q))$ 更新 C
    end for
  end while
end while
return R 中的 top-k 结果

Figure 4. The AIO-ANN algorithm 1
图 4. AIO-ANN 算法 1

5. 实验结果

在本节中，我们将 AIO-ANN 算法与目前主流算法 DiskANN 进行对比基准测试。所有实验都在相同规格的主机上完成，配置如下：

操作系统：Ubuntu 22.04

处理器：AMD Ryzen 7 5700G CPU

内存：32GB 3200MHzDDR4 DRAM

存储：三星 980 Pro 1TB

实验使用到了三个不同的数据集：

BigANN 10M：该数据集由大型图像数据集中提取的 SIFT 描述符组成。

SpaceV 10M：这个与网络搜索相关的数据集是微软 Bing 为竞赛发布的。它由 Microsoft SpaceV Superior 模型编码的文档和查询向量组成。

deep 10M：该数据集由 Imagenet 分类任务上预训练的 GoogLeNet 模型的最后一个全连接层的投影和归一化输出组成。

使用“Recall-with-QPS”度量来评估所提算法的性能，该度量包括两个组成部分：

Recall：在像 ANNS 这样的信息检索系统的上下文中，召回率量化了系统检索所有相关文档的能力。从数学上定义，它是检索到的相关结果数与现有总数的比率。越高的召回率表示系统检索能力越强。

QPS：表示每秒查询数，QPS 是信息检索系统的常见性能度量。它测量系统每秒可以处理的查询数。

高 QPS 表示一个能够在短时间内处理大量查询请求的高效系统。

我们在三个数据集上进行了实验，搜索线程数设置为 4。结果如图 5 所示。在 recall@1 和 recall@10 的所有三个数据集上，我们提出的 AIO-ANN 算法的性能对比 DiskANN 提高了 20%至 30%。这一优势在从较低的 90%到较高的 95%的召回范围内始终可观察到。此外，我们的算法显示出更强大的稳定性，更少的性能波动。

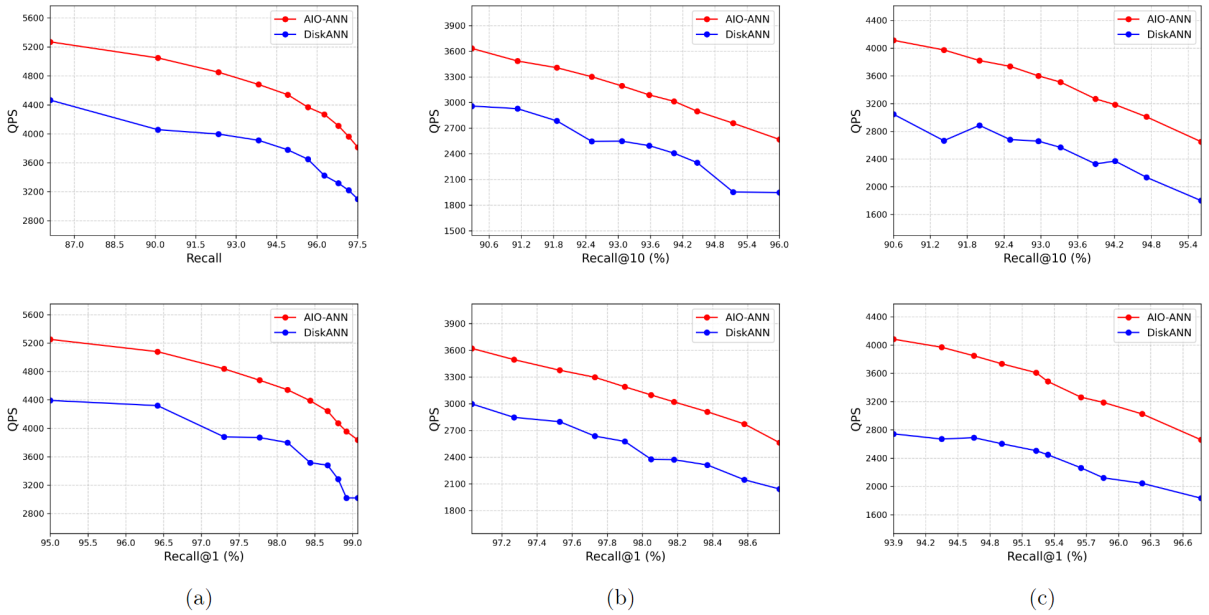

Figure 5. AIO-ANN vs. DiskANN on (a) BigANN 10M, (b) deep 10M, and (c) SpaceV 10M.
图 5. AIO-ANN 与 DiskANN 对比：(a) BigANN 10M；(b) deep 10M；(c) SpaceV 10M

6. 结论

在本文，我们提出了一种新型的磁盘向量检索算法 AIO-ANN。该算法主要使用异步 I/O 技术，尽可能降低了少量高延迟读请求对整体搜索性能的不利影响。在每一步迭代过程，算法并不等待所有 I/O 请求同时完成，而是在处理已有数据的同时等待其他 I/O 请求的完成。同时我们还讨论了两个可能优化，查询缓存与结果初始化，并应用在了算法的实现中。实验的结果表明，AIO-ANN 在性能上超越了主流的 DiskANN 算法，并且具有更强的稳定性。AIO-ANN 有很大潜力能提升已有的 ANNS 系统的性能，特别是在高 I/O 请求延迟的情况下。未来的工作将寻求探索额外的优化，并测试该算法在更大规模数据集上的性能。

参考文献

[1] Wang, M., Xu, X., Yue, Q. and Wang, Y. (2022) A Comprehensive Survey and Experimental Comparison of Graph-Based Approximate Nearest Neighbor Search. *Proceedings of the VLDB Endowment*, **14**, 1964-1978.

[2] Jayaram Subramanya, S., Devvrit, F., Simhadri, H.V., Krishnawamy, R. and Kadekodi, R. (2019) DiskANN: Fast Accurate Billion-Point Nearest Neighbor Search on a Single Node. *Proceedings of the* 33*rd International Conference on Neural Information Processing Systems*, Vancouver, 8-14 December 2019, 13766-137760

[3] Chen, Q., *et al*. (2021) SPANN: Highly-Efficient Billion-Scale Approximate Nearest Neighborhood Search. *Advances in Neural Information Processing Systems*, New York, USA, 21 December 2021, 5199-5212.

[4] Zhang, M. and He, Y. (2018) Zoom: SSD-Based Vector Search for Optimizing Accuracy, Latency and Memory. https://arxiv.org/abs/1809.04067

[5] Zhang, M. and He, Y. (2019) GRIP: Multi-Store Capacity-Optimized High-Performance Nearest Neighbor Search for Vector Search Engine. *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, Beijing, 3-7 November 2019, 1673-1682.

[6] Muja, M. and Lowe, D. (2009) Flann-Fast Library for Approximate Nearest Neighbors User Manual. Computer Science Department, University of British Columbia, Vancouver.

[7] Wang, J., Shen, H.T., Song, J., *et al*. (2014) Hashing for Similarity Search: A Survey. https://arxiv.org/abs/1408.2927

[8] Wang, J., Liu, W., Kumar, S. and Chang, S.F. (2015) Learning to Hash for Indexing Big Data—A Survey. *Proceedings of the IEEE*, **104**, 34-57. https://doi.org/10.1109/JPROC.2015.2487976

[9] Gersho, A. and Gray, R.M. (2012) Vector Quantization and Signal Compression. Springer, New York.

[10] Jegou, H., Douze, M. and Schmid, C. (2010) Product Quantization for Nearest Neighbor Search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **33**, 117-128. https://doi.org/10.1109/TPAMI.2010.57

[11] Ge, T., He, K., Ke, Q. and Sun, J. (2013) Optimized Product Quantization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **36**, 744-755. https://doi.org/10.1109/TPAMI.2013.240

[12] Kalantidis, Y. and Avrithis, Y. (2014) Locally Optimized Product Quantization for Approximate Nearest Neighbor Search. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, Columbus, 23-28 June 2014, 2329-2336. https://doi.org/10.1109/CVPR.2014.298

[13] Klein, B. and Wolf, L. (2019) End-to-End Supervised Product Quantization for Image Search and Retrieval. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Long Beach, 15-20 June 2019, 5036-5045. https://doi.org/10.1109/CVPR.2019.00518

[14] Johnson, J., Douze, M. and Jégou, H. (2019) Billion-Scale Similarity Search with GPUs. *IEEE Transactions on Big Data*, **7**, 535-547. https://doi.org/10.1109/TBDATA.2019.2921572

[15] Baranchuk, D., Babenko, A. and Malkov, Y. (2018) Revisiting the Inverted Indices for Billion-Scale Approximate Nearest Neighbors. In: Ferrari, V., Hebert, M., Sminchisescu, C. and Weiss, Y., Eds., *Computer Vision—ECCV* 2018, Springer, Cham, 202-216. https://doi.org/10.1007/978-3-030-01258-8_13

[16] Guo, R., Sun, P., Lindgren, E., et al. (2020) Accelerating Large-Scale Inference with Anisotropic Vector Quantization. *Proceedings of the 37th International Conference on Machine Learning*, 13-18 July 2020, 3887-3896.

[17] Toussaint, G.T. (1980) The Relative Neighbourhood Graph of a Finite Planar Set. *Pattern Recognition*, **12**, 261-268. https://doi.org/10.1016/0031-3203(80)90066-7

[18] Malkov, Y., Ponomarenko, A., Logvinov, A. and Krylov, V. (2014) Approximate Nearest Neighbor Algorithm Based on Navigable Small World Graphs. *Information Systems*, **45**, 61-68. https://doi.org/10.1016/j.is.2013.10.006

[19] Malkov, Y.A. and Yashunin, D.A. (2018) Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **42**, 824-836. https://doi.org/10.1109/TPAMI.2018.2889473

[20] Fu, C., Xiang, C., Wang, C. and Cai, D. (2019) Fast Approximate Nearest Neighbor Search with the Navigating Spreading-Out Graph. *Proceedings of the VLDB Endowment*, **12**, 461-474. https://doi.org/10.14778/3303753.3303754

[21] Simhadri, H.V., *et al*. (2022) Results of the NeurIPS'21 Challenge on Billion-Scale Approximate Nearest Neighbor Search. *Proceedings of the NeurIPS* 2021 *Competitions and Demonstrations Track*, New York, USA, 6-14 December 2021, 177-189.