

AKNN-Qalsh: PostgreSQL 系统高维 空间近似最近邻检索插件^{*}

张楚涵, 张家侨, 冯剑琳
(中山大学数据科学与计算机学院, 广东 广州 510006)

摘要: 复杂数据对象(如图片、文本)通常被表示成高维特征向量。PostgreSQL 系统现有的最近邻检索方法 KNN-Gist 基于树状索引实现, 无法高效支持高维数据的最近邻检索。引入的 PostgreSQL 系统高维空间近似最近邻检索插件: AKNN-Qalsh, 基于位置敏感哈希机制实现, 支持大规模、高维数据对象的近似最近邻检索。通过在五个真实数据集上的密集实验, 验证了该插件的有效性。

关键词: 高维数据; 特征向量; 最近邻检索; 位置敏感哈希; PostgreSQL 插件

中图分类号: TP311.13 **文献标志码:** A **文章编号:** 0529-6579 (2019) 03-0079-07

AKNN-Qalsh: an approximate KNN search extension for high-dimensional data in PostgreSQL

ZHANG Chuhan, ZHANG Jiaqiao, FENG Jianlin

(School of Data and Computer Science, Sun Yat-sen University, Guangzhou 510006, China)

Abstract: Complex data objects (such as pictures, text) are usually represented as high-dimensional feature vectors. The existing nearest neighbor search method KNN-Gist in PostgreSQL is based on the tree-structured index and cannot efficiently support the nearest neighbor search of high-dimensional data. The PostgreSQL system high-dimensional approximate nearest neighbor search extension: AKNN-Qalsh is introduced, which is based on the Locality-Sensitive Hashing scheme and supports approximate nearest neighbor search of large-scale, high-dimensional data objects. The effectiveness of the extension via extensive experiments on five real data sets is demonstrated.

Key words: high-dimensional data; feature vector; nearest neighbor search; Locality-Sensitive Hashing; PostgreSQL extension

最近邻检索是数据库研究的重要问题之一, 即查找在数据库 D 中与查询对象距离最近的数据对象。在数据库中进行图片相似性搜索、文本相似性搜索、地理空间位置比较等查询, 本质上都是要解决最近邻检索问题。

PostgreSQL 作为业界内第一个吸纳 KNN (K-nearest neighbor, K 最近邻) 技术的数据库系

统^[1], 为用户提供了 KNN-Gist 索引来解决最近邻检索问题, 但 KNN-Gist 索引仅适用于如地理信息等低维数据。例如, 基于 KNN-Gist 索引实现的图片相似性检索的插件 ImgSmlr^[2], 也只能利用低维的图片签名, 通过图片的全局特征进行相似性检索, 无法直接利用高维特征向量进行相似性检索, 当需要避免图片尺度变化对图片相似性检索的影响

* 收稿日期: 2018-10-11

基金项目: 国家自然科学基金 (61772563)

作者简介: 张楚涵 (1995 年生), 女; 研究方向: 数据库、数据挖掘; E-mail: zhangchh7@mail2.sysu.edu.cn

通信作者: 冯剑琳 (1970 年生), 男; 研究方向: 数据库、数据挖掘; E-mail: fengjlin@mail.sysu.edu.cn

时, `ImgSmlr` 无法胜任基于尺度不变特征变换 (SIFT) 的检索^[3]。此外, PostgreSQL 还拥有开源的字符串相似性检索插件 `pg_trgm`^[4], 它通过测定字母数字文本基于三元模型匹配的相似性进行检索, 适用于较小规模文本、字符串等数据。一般情况下, 图片、视频、文本等信息的特征向量的维度较高, 目前, PostgreSQL 中还没有针对高维数据的最近邻检索的扩展。因此, 我们希望为 PostgreSQL 数据库系统设计并实现一个高效的、支持大规模、高维度数据的最近邻检索插件。该插件可以作为一个最近邻检索模块, 与图片、文本或视频等复杂数据对象的特征提取模块组合, 在数据库中实现基于高维特征向量的图片、文本相似性检索。

现有的可以精确找到查询对象的最近邻的数据结构, 如 KNN-Gist 所使用的树结构的检索性能会随着数据对象维度的升高而急剧下降。在向量维度大于 10 之后, 其性能甚至会低于暴力地线性检索^[5]。而在很多实际应用场景中, 往往不需要检索查询对象绝对精确的最近邻。这使得我们可以一定程度上通过降低检索精度来提高检索速度, 将最近邻检索问题转化为近似最近邻检索问题再求解: 若查询对象 q 与其精确最近邻 o^* 的距离为 r^* , 那么, c -近似最近邻检索就是要找出 q 的近似最近邻 o , 使得 q 与 o 的距离小于 r^* 的 c 倍, 其中 c 为近似比例, 且 $c > 1$ 。

位置敏感哈希 (Locality-Sensitive Hashing, 简称 LSH) 机制^[6]及其变体^[7-10]是目前解决高维空间中近似最近邻检索问题最有效的手段。位置敏感哈希的基本思想是: 如果两个高维特征向量的距离较小, 那么, 它们就会以较大的概率被哈希到同一个桶中。位置敏感哈希的一个新变体, QALSH 机制^[11-12], 是目前业界领先的、有理论保证的高维空间近似最近邻检索方法。

与 KNN-Gist 索引基于树结构的最近邻检索机制不同, QALSH 机制通过解决一系列给定查询半径的近似最近邻检索问题来求解查询对象的近似最近邻: 可以发现, c -近似最近邻问题的检索半径依赖于 r^* , 而 r^* 无法事先确定, 因此, 近似最近邻检索问题无法直接求解。需要将 c -近似最近邻问题转化为一组检索半径 r 确定的 (依次取值 $1, c^1, c^2, c^3, \dots$) 近似最近邻检索问题, (r, c) -近似最近邻检索: 对于给定的检索半径 r 和查询对象 q 返回一个数据对象使得 q 到该数据对象的距离小于或等于 cr 。

QALSH 机制针对外存设计, 能够更好地与数据库系统相结合。并且, QALSH 机制查询引导的分桶策略和虚拟再哈希技术, 使得我们可以在大数据集上进行任意检索半径 r 的 (r, c) -近似最近邻检索。

本文所描述的近似最近邻检索插件 AKNN-Qalsh 就是基于 QALSH 机制设计并实现的。QALSH 机制的检索性能基本不受数据维度的影响, 这一性质使得 AKNN-Qalsh 为 PostgreSQL 提供了高效的针对高维空间的最近邻检索功能具有如下特点:

- 1) 支持大规模、高维数据对象的最近邻检索
- 2) 通用性: 可以与图像、文本、视频等各种多媒体数据的特征提取模块相结合, 解决多种数据对象的最近邻检索问题;
- 3) 检索速度快: 检索速度远低于线性检索;
- 4) 检索精度高: 检索精度在用户可接受范围内, 且接近线性检索结果, 有理论保证;
- 5) 检索效率稳定: 其检索效率受数据维度影响极小;
- 6) 使用方便: 用户通过直接调用相应的函数方法即可在数据库中实现数据预处理或近似最近邻检索。

1 AKNN-Qalsh 的设计与实现

图 1 给出了利用 AKNN-Qalsh 插件进行近似最近邻检索的流程, 包含预处理和查询两个阶段: 在预处理阶段需要在 PostgreSQL 中构建关系表; 在查询阶段, 根据关系表中的信息检索查询对象的近似最近邻。

1.1 数据表

在 PostgreSQL 数据库中, 数据集中的数据对象和查询对象分别存放在图 2 所示的 `data` 表和 `query` 表中, 表中每一行由对象的 `id` 和其向量坐标 (`coordinate`) 构成。另外, 需要在 `data` 表的 `id` 列上建立 Hash 索引, 使得检索时能够在 `data` 表中能够快速根据 `id` 值取得对应的向量坐标。

1.2 预处理阶段

1.2.1 参数表 QALSH 机制需要预先根据数据集中数据对象的个数 n 、用户规定的近似比例 c 等参数, 确定最近邻检索所需的哈希表个数 m 以及碰撞阈值 l 等参数, 这些参数均需保存在 `param` 表中以记录该数据集的信息。

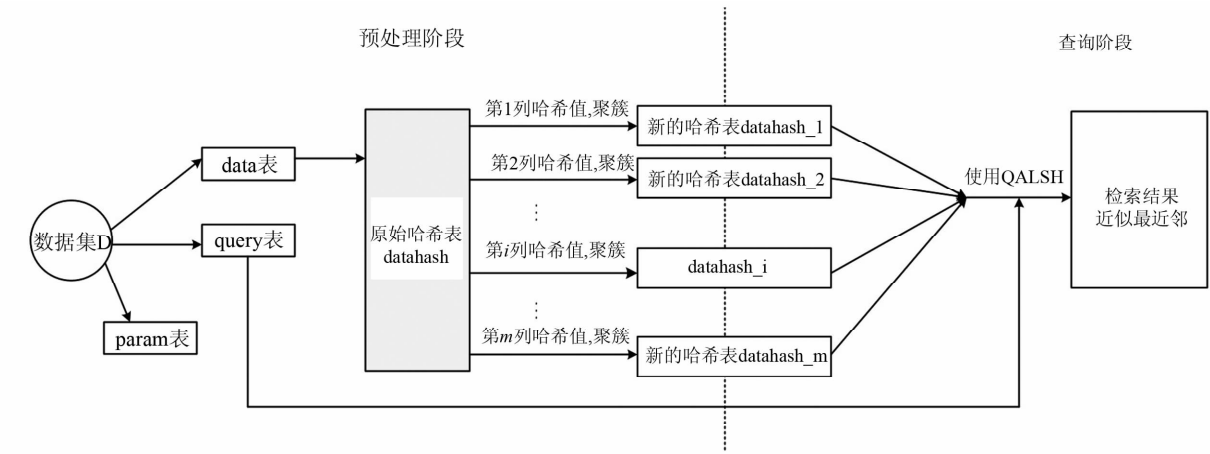


图 1 AKNN-Qalsh 插件整体概述
Fig. 1 Overview of AKNN-Qalsh

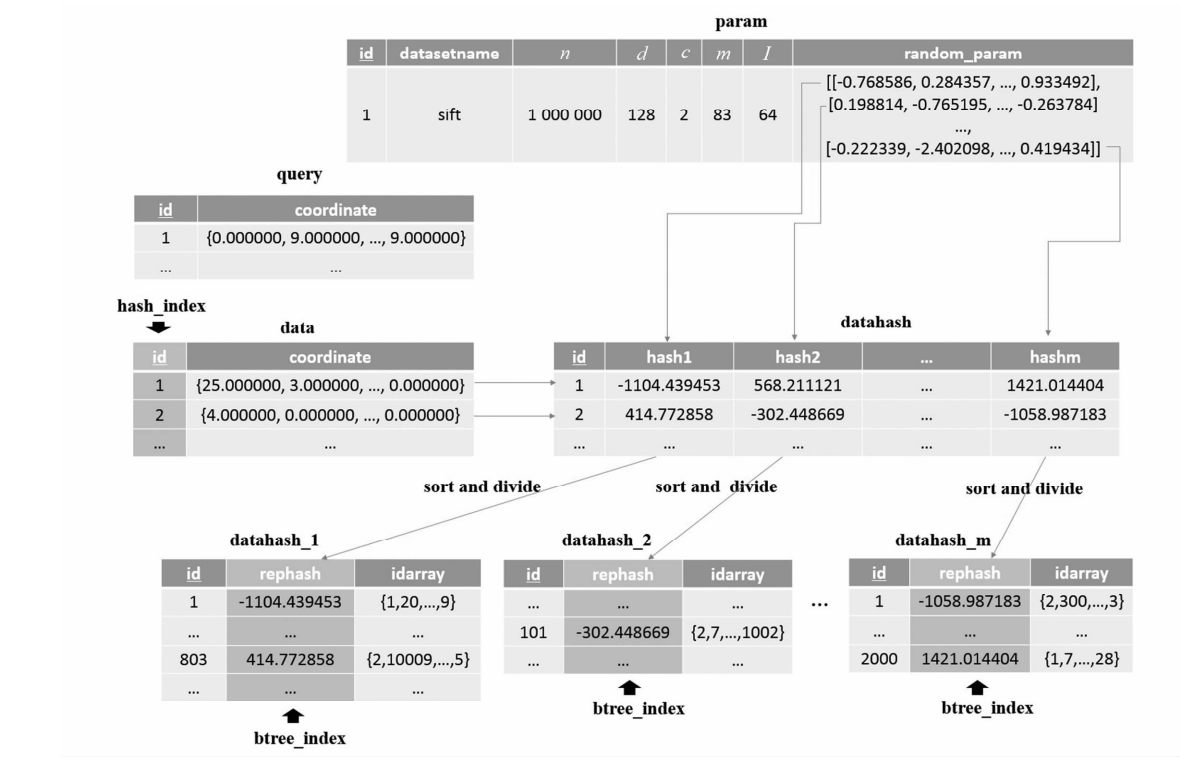


图 2 AKNN-Qalsh 插件中的关系表
Fig. 2 Tables of AKNN-Qalsh

另外，用来映射高维数据的 m 个位置敏感哈希函数信息将存放在 `param` 表的 `random_param` 字段下。对于一个数据对象维度为 d 的数据集，QALSH 机制对应的位置敏感哈希函数形式为： $h = \vec{a} \cdot \vec{o}$ ，其中 $\vec{o}$ 是数据对象的向量坐标， $\vec{a}$ 为每一个分量都独立产生于标准正态分布的 d 维向量，每一

个随机向量 $\vec{a}$ 定义了一条随机直线，也界定了一个哈希函数。因此，该数据集的 `param` 表的 `random_param` 字段存放的数组大小为 $m \times d$ 。在该数组中，从第一个数开始的每 d 个数表示一个随机向量，一共 m 个随机向量。分别构成了 m 个 QALSH 机制中的位置敏感哈希函数。`param` 表中的每个字段类型以及对应的参数含义详见表 1。

表 1 param 表字段
Table 1 Fields of param table

字段	类型	参数含义
id	integer	ID
datasetname	varchar[]	数据集名称
<i>n</i>	integer	数据对象个数
<i>d</i>	integer	数据对象维度
<i>c</i>	float	近似比例
<i>m</i>	integer	位置敏感哈希函数个数
<i>l</i>	integer	碰撞阈值
random_param	float[]	位置敏感哈希函数

1.2.2 哈希表 对应于 param 表中 random_param 字段下的 *m* 个哈希函数，需要分别为该数据集的数据对象生成 *m* 个哈希表。直观地，一个哈希表中的每一行由数据对象的 id、数据对象映射到随机向量得到的哈希值构成。也就是说，对应一个哈希函数，将得到一个 *n* 行、2 字段的哈希表。如图 2 所示，对应 param 表中的哈希函数 h_2 ，可以得到哈希表 datahash_2；datahash_2 哈希表中 id = 2 的行存放了 id 为 2 的数据对象 o_2 经过映射得到的哈希值， $h_2(o_2) = \vec{a}_2 \cdot \vec{o}_2$ 。

为了保证 QALSH 机制在检索阶段对哈希表的范围查询可以高效地完成，需要为每一个哈希表的哈希值的列利用 B⁺ 树进行索引。PostgreSQL 系统提供的 B-Tree 索引结构是根据 B⁺ 树的一种变体，B^{link}-树来实现的^[13]，恰好可以满足这一需求。因此，我们直接利用 PostgreSQL 提供的 B-Tree 索引为哈希表建立索引，使得 AKNN-Qalsh 与 PostgreSQL 系统更好地耦合。

值得注意的是，这种直观的哈希表结构在索引过程中将会暴露出缺陷。在实际应用场景下，数据集中数据对象的个数 *n* 值可能很大，直接对哈希表中的哈希值列建立 B-Tree 索引时得到的索引结构比较庞大，而庞大的索引结构将会导致非常可观的空间开销。并且，由于事先并未对哈希表中的数据对象根据其哈希值进行排序便建立索引，在数据库中，若对该列进行范围查询，则其索引指向的表元组是分散在堆文件不同页面中的，连续性很差。也就是说，通过范围查询获取满足条件的元组时，会发生堆文件页面的跳跃，引起随机 I/O，也就增加了索引查询的时间开销。另外，由于 PostgreSQL 中 B-Tree 索引结构的一个索引元组指向一个索引页面或表元组。因此，一个叶子节点索引的数据对象非常有限。当利用 B-Tree 索引进行范围查询时，如果满足条件的数据对象较多时，需要读取的叶子节

点也比较多，也会引起读取索引时的 I/O 增加，增加索引扫描的时间开销。为了解决上述的几个问题，可以设计一种新的哈希表结构。

对于索引时页面跳跃引起随机 I/O 的问题，通过事先对哈希表中的数据对象按照其哈希值进行排序来解决。而针对 B-Tree 索引结构庞大的问题，在利用 QALSH 机制进行最近邻检索时，并不需要粒度（即每个叶子节点中的一个索引元组索引的数据对象个数）为 1 的 B-Tree 索引结构。因此，可以通过调整 B-Tree 索引的粒度来控制 B-Tree 索引的大小。考虑预先对哈希表中的数据对象进行聚簇，设定每个簇中含有的数据对象个数为 *h*，我们称其为簇的容量。每一个簇的代表哈希值为该簇中所有数据对象哈希值的最小值，再以簇为最小单位建立索引，索引的键即为簇的代表哈希值，索引的值即为簇中数据对象的 id。

上述的聚簇策略，在降低树结构的深度的同时，也解决了满足范围查询条件的数据对象较多时，需要读取较多个叶子节点的问题：如果某次范围检索中满足条件的数据对象有 2 000 个，假设 B-Tree 索引的一个页面中可以存放 100 个索引元组，新的哈希索引结构只需要读取 1 个索引叶子页面即可取到所有满足条件的数据对象，而没有经过聚簇的哈希表结构则需要读取 20 个索引叶子页面。由此可见，新的哈希表结构可以减少索引结构的空间开销，更重要的是，可以显著提升索引效率。

1.3 查询阶段

1.3.1 KNN 检索 AKNN-Qalsh 插件的近似最近邻检索部分按照针对 (*r*, *c*)-NN 问题的 QALSH 机制设计并实现：在数据库中，对于每一个 datahash_*i*, *i* = 1, 2, ..., *m* 表，检索哈希值满足查询对象哈希值到查询对象哈希值加 $\frac{1}{2}wr$ 范围或查询对象哈希值减 $\frac{1}{2}wr$ 到查询对象哈希值范围的对象并计数，即完成了一轮给定半径 *r* 的扫描（如图 3 所示）。

新一轮检索，更新半径 *r* 为原来的 *c* 倍，并依据 QALSH 机制中的虚拟再哈希技术，对于每一个 datahash_*i*, *i* = 1, 2, ..., *m* 表分别检索满足上一轮检索下界减 $\frac{1}{2c}wr$ 到上一轮检索下界范围或上一轮检索上界到上一轮检索上界加 $\frac{1}{2c}wr$ 范围的对象并计数。当数据对象的计数大于阈值 *l* 时，计算其与查询对象的真实距离并更新容量为 *k* 的结果集。在

检索过程中, 若满足 QALSH 机制的检索算法的终止条件, 则算法终止并返回检索结果, 否则, 需要继续更新检索半径进行新一轮的检索。

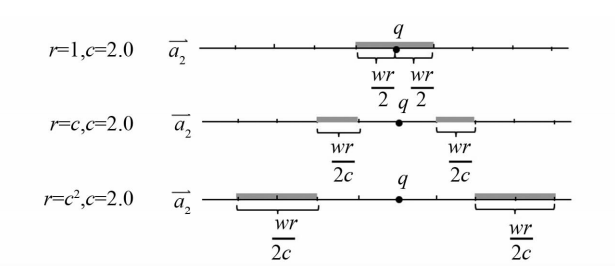


图 3 KNN 检索中的半径更新与虚拟再哈希
Fig. 3 Radius update and Virtual Rehashing

图 3 为检索过程中, 以某一哈希表为例的半径更新与虚拟再哈希的示例, 其中阴影部分代表每轮检索的查询范围。

AKNN-Qalsh 插件在实现基于 B-Tree 索引的范围查询时, 利用了 PostgreSQL 系统提供的一系列索引操作方法, 如 `index_beginscan` 等, 而不是简单

地通过其提供的 SPI (server programming interface, 即服务器编程接口), 直接利用 SQL 语句进行查询。做到了与 PostgreSQL 系统的紧密结合, 从而进一步提升了检索效率。

1.4 增、删、改功能

为了更加贴合数据库扩展应用的需求, AKNN-Qalsh 插件还支持增、删、改三个数据库基本操作。如实现向数据集和或查询集中增加对象的操作, 需要将新的对象添加到相应数据表、计算其哈希值并更新哈希表。还需要注意 param 表中的参数的变化, 如数据对象个数 n 增加等。同样地, 删除、修改操作与插入操作类似, 都需要更新 PostgreSQL 中相应的关系表。

1.5 用户接口

前文描述了 AKNN-Qalsh 插件在运行时生成的数据库表的结构、预处理部分和查询部分的实现、以及增删改三个基本操作的实现。我们希望用户在使用 AKNN-Qalsh 插件时, 不需要理解或记忆这些实现细节。因此, 我们为用户提供了方便的用户接口如表 2 所示。

表 2 用户接口
Table 2 The interface functions of AKNN-Qalsh

接口函数	参数	返回值	功能
aknnqalsh_index	数据集文件路径、查询集文件路径、数据集名称、 n 、 d 、 c 、 h	void	预处理接口
aknnqalsh_knn	查询对象的 id、 k	int[]	检索接口, 返回 Top- k 近似最近邻的 id 值
aknnqalsh_data/query_insert	数据/查询对象特征向量	void	增添数据/查询对象
aknnqalsh_data/query_update	数据/查询对象特征向量	void	修改数据/查询对象
aknnqalsh_data/query_delete	数据/查询对象 id	void	删除数据/查询对象

2 实验

2.1 数据集

本文将利用 Mnist^①、Sift 和 Gist^②、LabelMe^③、P53^④ 五个数据集进行实验, 数据集的信息以及其对应的查询集信息如表 3 所示, n 记录了数据集中数据对象的个数, d 表示数据对象的维度, 查询集中查询对象的个数为 qn 。

2.2 性能评测指标

本文以运行时间和平均总体比率作为对比实验的评估度量, 来评估该近似最近邻检索扩展程序的性能。两种度量方式的定义如下。

2.2.1 运行时间 (running time) 运行时间记为检索一个查询对象 q 的 k 个近似最近邻所用的时间, 对数据集和查询集的预处理时间不计入在内。本实验中统计了检索 100 个查询对象的 k 近似最近邻所用时间的平均值, 并以此作为度量。

① <http://yann.lecun.com/exdb/mnist>
② <http://corpus-texmex.irisa.fr/>
③ <http://labelme.csail.mit.edu/Release3.0/>
④ [http://archive.ics.uci.edu/ml/datasets/P53 + Mutants](http://archive.ics.uci.edu/ml/datasets/P53+Mutants)

表 3 数据集信息
Table 3 The information of dataset

dataset	<i>n</i>	<i>d</i>	<i>qn</i>
Mnist	60 000	50	100
Sift	1 000 000	128	100
Gist	1 000 000	960	100
LabelMe	180 993	512	100
P53	31 059	5 408	100

2.2.2 平均总体比率（average overall ratio）即比率（ratio），用于度量 KNN 检索方法的精确度。对于查询对象 q ，其最近邻检索结果的 $\text{Ratio} = \frac{1}{k} \sum_{i=1}^k \frac{\|o_i, q\|}{\|o_i^*, q\|}$ ， $i = 1, 2, \dots, k$ ，其中， o_i 是方法对于查询对象 q 返回的第 i 个数据对象， o_i^* 是查询对象 q 的第 i 个真实最近邻。

2.3 对比实验

由于目前 PostgreSQL 中还没有针对高维空间的

近似最近邻检索方法，我们将本文所述的近似最近邻检索扩展程序与最基本的线性检索进行比较。线性检索计算查询对象与每个数据对象之间的距离，从而找到精确的最近邻，在实验中简记为 Linear。对于不同数据集，我们均将参数 c 设置为 2.0，参数 h 统一设置为 200。AKNN-Qalsh 插件的检索时间开销与线性检索的时间开销、检索精度比较如图 4、5 所示。可以发现，AKNN-Qalsh 插件的检索速度远高于线性检索方法，且其近似检索精度的损失在可接受范围内。面对大规模、高维度的数据集时，线性检索的时间开销非常大，利用线性检索在 Gist 数据集中获取一个查询对象的 Top-100 个最近邻需要近 60 s 的时间，而利用 AKNN-Qalsh 插件仅需要 1 s。且其查询结果的平均总体比率大约为 1.07。说明 AKNN-Qalsh 插件可以在短时间内获取查询对象的近似最近邻结果，还保证了较高的精确度。

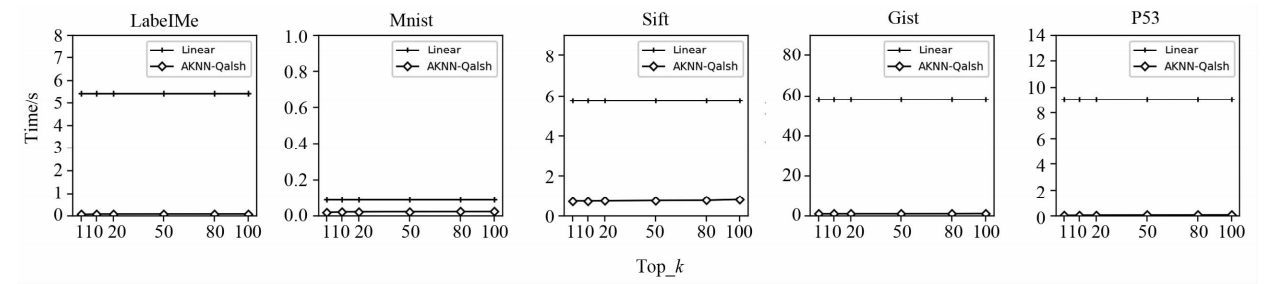


图 4 AKNN-Qalsh 插件与线性检索的检索时间比较
Fig. 4 The search time of Linear and AKNN-Qalsh

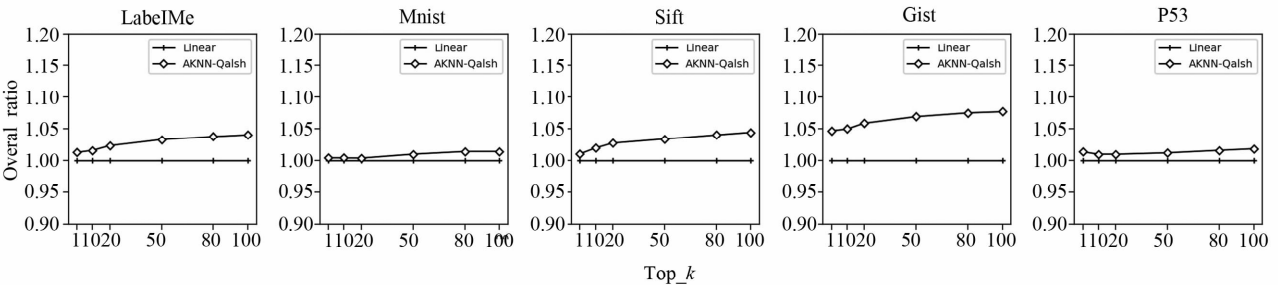


图 5 AKNN-Qalsh 插件与线性检索的检索精度比较
Fig. 5 The overall ratio of Linear and AKNN-Qalsh

其实，不同的 h 值会影响 AKNN-Qalsh 插件的检索性能。当 h 值较小时，B-Tree 索引的粒度相对较小时，B-Tree 索引结构较庞大，最近邻检索的时间开销将会很高； h 值过大时，B-Tree 的索引粒度

过大，起不到索引效果，检索精确度将降低。因此，我们需要为 AKNN-Qalsh 插件选择恰当的 h 值。上述对比实验中，针对不同的数据集，统一设定了簇的容量为 200，是比较好的折中。

3 结 论

本文详述了 PostgreSQL 系统近似最近邻检索插件 AKNN-Qalsh 的设计与实现。该插件借助 Post-

greSQL 数据库的 B-Tree 索引与 Hash 索引、以及统一的索引扫描接口, 可以高效地支持近似最近邻检索。通过真实数据集上的密集实验, 我们展示了插件检索准确率高、检索速度快的特点。

参考文献:

[1] The PostgreSQL Global Development Group. PostgreSQL 9.1 Press Kit [EB/OL]. (2011-09-12) [2018-5-14]. <https://www.postgresql.org/about/press/press-kit91/>.

[2] KOROTKOV A. ImgSmlr [CP/OL]. (2018-02-24) [2018-5-14]. <https://github.com/postgrespro/imgsmlr>.

[3] LOWE D G. Distinctive image features from scale-invariant keypoints [J]. International journal of computer vision, 2004, 60(2): 91-110.

[4] PostgreSQL 9.6.9 Documentation. Appendix F. Additional Supplied Modules [EB/OL]. [2018-5-14]. <https://www.postgresql.org/docs/9.6/static/pgtrgm.html>.

[5] WEBER R, SCHEK H J, BLOTT S. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces [C]//VLDB. 1998, 98: 194-205.

[6] INDYK P, MOTWANI R. Approximate nearest neighbors: towards removing the curse of dimensionality [C]//Proceedings of the thirtieth annual ACM symposium on Theory of computing. ACM, 1998: 604-613.

[7] PANIGRAHY R. Entropy based nearest neighbor search in high dimensions [C]//Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm. Society for Industrial and Applied Mathematics, 2006: 1186-1195.

[8] LV Q, JOSEPHSON W, WANG Z, et al. Multi-probe LSH: efficient indexing for high-dimensional similarity search [C]//Proceedings of the 33rd international conference on Very large data bases. VLDB Endowment, 2007: 950-961.

[9] TAO Y, YI K, SHENG C, et al. Efficient and accurate nearest neighbor and closest pair search in high-dimensional space [J]. ACM TODS, 2010, 35(3): 20.

[10] GAN J, FENG J, FANG Q, et al. Locality-sensitive hashing scheme based on dynamic collision counting [C]//Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data. ACM, 2012: 541-552.

[11] HUANG Q, FENG J, ZHANG Y, et al. Query-aware locality-sensitive hashing for approximate nearest neighbor search [J]. Proceedings of the VLDB Endowment, 2015, 9(1): 1-12.

[12] HUANG Q, FENG J, FANG Q, et al. Query-aware locality-sensitive hashing scheme for lp norm [J]. The VLDB Journal, 2017, 26(5): 683-708.

[13] LEHMAN P L. Efficient locking for concurrent operations on B-trees [J]. ACM TODS, 1981, 6(4): 650-670.

(责任编辑 冯兆永)