基于 LSH 的高维大数据 k 近邻搜索算法

王忠伟,陈叶芳,钱江波,陈华辉
(宁波大学信息科学与工程学院,浙江宁波 315211)

摘 要: 局部敏感哈希(LSH)及其变体是解决高维数据 k 近邻(kNN)搜索的有效算法. 但是,随着数据规模的日趋庞大,传统的集中式 LSH 算法结构已经不能够满足大数据时代的需求. 本文分析传统 LSH 方案的不足之处,拓展 AND-OR 结构,提出通过索引而不比较原始数据直接实现高维大数据 k 近邻搜索算法 C2SLSH. 理论分析和实验证明,C2SLSH 在分布式平台下具有稳定的可扩展性,在保证同等精确率的情况下,处理速度大约是现有方法的 3 倍.

关键词: 高维数据 k 近邻; 局部敏感哈希; MapReduce; 冲突计数排序
中图分类号: TP311.13 **文献标识码:** A **文章编号:** 0372-2112 (2016)04-0906-07
电子学报 URL: http://www.ejournal.org.cn **DOI:** 10.3969/j.issn.0372-2112.2016.04.022

LSH-Based Algorithm for k Nearest Neighbor Search on Big Data

WANG Zhong-wei, CHEN Ye-fang, QIAN Jiang-bo, CHEN Hua-hui
(*Faculty of Electrical Engineering and Computer Science, Ningbo University, Ningbo, Zhejiang* 315211, *China*)

Abstract: The locality sensitive hashing (LSH) and its variants are efficient algorithms to solve the k nearest neighbor (kNN) search problems on high-dimensional data. However, with the increase of large data size, the traditional centralized LSH algorithms cannot meet the challenge of the big data era. Based on a new AND-OR construction, this paper proposes an algorithm (called C2SLSH) for the k nearest neighbor search on big data. Different to the traditional algorithms, the C2SLSH can directly get the results from an index without having to compare the original data. The theoretical analysis and experimental results show that the algorithm has stable scalability on a distributed platform. Furthermore, it is faster than the conventional methods for about three times with the same accuracy rate.

Key words: kNN; LSH; MapReduce; collision counting sorting

1 引言

随着网络规模的扩大,数据,尤其是高维数据呈爆炸式增长[1~3]. 从这些海量数据中搜索近似对象是很多应用的关键,如近似检索、推荐系统、k 近邻问题等. 通过构造索引,如 R-tree[4]、K-D tree[5] 等,可快速找到查询对象的近邻或近似对象. 然而,随着数据维度的增加,这些算法的效率大幅度降低. 当维度大于 10 的时候,这些方法甚至比线性扫描方法还慢,这种现象被称为"维度灾难"[6]. 局部敏感哈希(LSH, Locality Sensitive Hashing)是由 Indyk 和 Motwani 提出的一种高维索引结构[7],它的基本思想是针对数据对象集,利用一组特定的哈希函数来建立哈希表. 使得在一定的相似度量条件下,相似对象映射到相同的区域的概率大于非相似对象. LSH 及其变体[8~11] 可以将相似的对象以高概率映射到同一个桶中,并通过解决近似最近邻的方法来替代解决精确近邻问题,是处理高维数据近似问题的有效算法.

然而,随着大数据时代的到来,"维度灾难"和数据规模两个问题叠加,使得 k 近邻查询问题变得非常困难,传统的集中式 LSH 算法结构已经不能够满足需求. 目前 Hadoop 平台[12] 深受业界关注,它具有可扩展、易使用、节点容错以及高可靠性等优秀特性,适合处理大规模数据. 但是如果直接将局部敏感哈希及其变体应用到该平台,将不能充分发挥分布式平台的优势. 本文在 Hadoop 分布式平台下研究基于 LSH 的高维大数据 kNN 搜索算法,保证精确率和快速性,并减少了空间复杂度.

收稿日期:2014-12-24;修回日期:2015-09-20;责任编辑:孙瑶
基金项目:国家自然科学基金(No. 61472194, No. 61572266);浙江省自然科学基金(No. LY13F020040);宁波市自然科学基金(No. 2014A610023);"信息与通信工程"浙江省重中之重学科开放基金

本文的贡献如下:(1)提出适合分布式实现的 AND-OR 构造方法,通过理论分析证明经索引而不用比较原始数据可直接实现高维大数据 k 近邻搜索;(2)充分考虑分布式计算的因素,紧密结合 MapReduce 计算过程,在 Hadoop 分布式平台上实现了所提出的算法;(3)真实数据集实验表明,所提出算法在分布式平台上具有有效性和稳定性,实验结果和理论分析的结论相一致.

2　相关工作

大数据处理面临的一大挑战是如何设计合适的索引结构,从而能高效地完成近邻搜索. 虽然 LSH 是一种非常流行的高维数据处理算法,但是由于其对空间的依赖性较强,因此并不适合直接用来处理大数据. 有研究者改进了 LSH 算法并在分布式平台上应用,但是相关的研究还比较少.

MIXED-LSH[13] 把 LSH 应用到大数据集,在由多个节点组成的 P2P 分布式环境中实现了 LSH 算法. 在分布式环境中实现 LSH 的一个简便方法是每个节点都保持相同的哈希表的数量. 但是这会增加远程访问,因为许多节点要存取访问所有的哈希表. 如果通信延迟是瓶颈的话,这种方法响应时间会比较长. MIXED-LSH 巧妙地使用为节点分配哈希桶的方法来减少远程访问. 特别地,MIXED-LSH 把相关的哈希桶存储在一起,分配到同一节点的哈希桶来自于不同的哈希表. 由于这一策略的使用,在处理一个查询访问时,对于那些应该被访问的对象,执行 MIXED-LSH 索引方法可以一次访问多个哈希桶,从而减少了远程访问的频率. MIXED-LSH 重点在于解决分布式环境下所产生的网络开销,其使用的距离度量是 L1 距离,很难有效的扩展到其他距离度量的应用. 另外 MapReduce 和 P2P 是不同的分布式环境,不能迁移应用.

LayeredLSH 算法[14] 是对 EntroryLSH[15] 的分布式改进,可使用 MapReduce 框架实现计算过程. 通过二次哈希把数据对象 $\boldsymbol{q}$ 存在某台机器 x 上. 查询时,先给对象 $\boldsymbol{q}$ 一个小的偏移量,生成 $\boldsymbol{q}+\boldsymbol{\delta}_i$,把 $\boldsymbol{q}$ 和 $\boldsymbol{q}+\boldsymbol{\delta}_i$ 进行二次哈希,根据它们对应的哈希值,找到对应的机器 x,然后在机器 x 上执行相似性搜索. 通过采用二次哈希的策略,第一次哈希数据对象能哈希到同一个桶,那么第二次哈希会分布相同机器上,在 Reduce 阶段给查询对象一个偏移量,根据哈希值首先找到相应桶对应的机器,然后再进行搜索. LayeredLSH 保证了良好的负载平衡,但是这种方法是以更多的时间开销为代价的. 另外,二次哈希也使得空间复杂度比传统 LSH 算法更大.

虽然一些改进的 LSH 方法在较小的数据集上能提供良好的过滤效果并且表现优异,但是随着数据集的增大,算法效果明显降低. 上述相关 LSH 的分布式算法没有具体考虑假阳性和假阴性的处理,会导致候选集的冗余度太大. 另外,在得到候选集之后,从候选集获取结果集的过程中,需要使用相应的距离度量计算检验候选集中每一个数据对象,此计算过程仍占据了查询时间的主要部分,所以对于较大的数据集来说,LSH 算法的时间效率依然存在改进的空间. 本文提出的以冲突次数作为返回结果的测度,充分考虑 AND-OR 对假阳性和假阳性的控制,在保证结果正确率的情况下,有效的减少了查询响应的时间.

3　LSH 及 AND-OR 构造

定义 1(LSH)　设 S 是距离度量 dist 的(数据)对象集合,d_1、d_2 为距离度量 dist 的两个距离. 如果哈希函数族 $\boldsymbol{H}$ 中的每一个函数 h 满足如下两个条件,则哈希函数族 $\boldsymbol{H}$ 对于 S 中任意对象 $\boldsymbol{x}$ 和 $\boldsymbol{y}$ 称为$(\mathrm{dist}_1,\mathrm{dist}_2,p_1,p_2)$-sensitive:

(1)如果 $d(\boldsymbol{x},\boldsymbol{y})\leqslant \mathrm{dist}_1$,那么 $\Pr[h(\boldsymbol{x})=h(\boldsymbol{y})]\geqslant p_1$;

(2)如果 $d(\boldsymbol{x},\boldsymbol{y})\geqslant \mathrm{dist}_2$,那么 $\Pr[h(\boldsymbol{x})=h(\boldsymbol{y})]\leqslant p_2$.

其中 $\mathrm{dist}_1<\mathrm{dist}_2$,$p_1>p_2\in[0,1]$,$\Pr[\,]$是概率.

LSH 函数适合许多距离度量,其中 l_p 范式是基于 p-stable 分布的 LSH 函数族[16]. 在这种情况下,对每一个高维数据对象 $\boldsymbol{v}$,哈希函数形式为

$$\boldsymbol{h}_{\alpha,b}(\boldsymbol{v})=\left\lfloor\frac{\boldsymbol{a}\cdot\boldsymbol{v}+b}{w}\right\rfloor$$

其中 $\boldsymbol{a}$ 是一个与 $\boldsymbol{v}$ 同维度的随机向量集合,它的元素从 p-stable 分布中独立选取,w 是窗口长度参数,b 是从 $[0,w]$ 中随机选取的实数.

从 LSH 的定义可以推出,使用一个哈希函数会产生假阳性(False Positive)或假阴性(False Negative)错误. 假阳性就是非相似的数据对象被哈希到相同的桶中,而假阴性就是真正相似的对象未能被哈希到相同的桶. 为了提高 LSH 查询的正确性,传统 LSH 函数哈希族引入 AND-OR 构造技术来改善[17]. AND 构造使得 $\boldsymbol{x}$ 和 $\boldsymbol{y}$ 成为彼此的相似候选对象,当且仅当在所选同一组哈希函数计算结果同时相等. OR 构造使得 $\boldsymbol{x}$ 和 $\boldsymbol{y}$ 成为彼此的相似候选对象,当且仅当所选同一组哈希函数计算结果至少有一个相等.

传统 AND-OR 构造可以提高精确度,但是时空效率太低,主要是因为构造过程中使用了太多的哈希函数,计算量大且耗费时间,存储太多的对象副本也耗费空间. 另外,在获得候选结果集之后,从候选集获得结果集,需要计算真实相似度的集合往往也较大,使得计算时间延长.

4　冲突计数排序 LSH

定理 1　在相同次数 LSH 运算中,如果两个对象的

距离越近,那么哈希到同一个桶的次数越多.

证明 根据定义1,对于两个对象 $\boldsymbol{x},\boldsymbol{y}$,$f(t)$代表服从 p-stable 分布的概率密度函数. 设 $\sigma=\|\boldsymbol{x}-\boldsymbol{y}\|$,有 $p(\sigma)=\Pr[h(\boldsymbol{x})=h(\boldsymbol{y})]=\int_0^w \frac{1}{\sigma}f\left(\frac{t}{\sigma}\right)\left(1-\frac{t}{w}\right)\mathrm{d}t$[7]. 该公式表明两个对象 $\boldsymbol{x}$ 和 $\boldsymbol{y}$ 映射到同一个随机向量区间的概率 $p(\sigma)$ 与两个对象的距离 σ 成反比,也就是说,两个对象距离越大获得相同哈希值的概率越小. 对于固定个数的哈希函数 n,设其中冲突的个数是 $\alpha(\alpha<n)$,那么如 α 的值越接近 n,即两个对象映射到同一个桶中的次数越多,则它们的相似度就越高. 相反两个对象相似度越低. 证毕

根据定理1,kNN 计算过程可分为两个步骤:第一步,使用选定的多个哈希函数计算每个对象的哈希值,并映射到相应的桶中;第二步,计算 LSH 冲突次数. 由于对象在 LSH 哈希空间下和原空间的距离是高概率等价的,所以使用 LSH 冲突次数来计算 kNN 可以得到正确的结果集. 基于此,本节提出 C2SLSH(Collision Counting Sorting LSH)算法结构. 通过将对象之间产生的冲突次数进行统计,并对结果正确性的影响进行理论分析.

如前所述,给定一个$(\mathrm{dist}_1,\mathrm{dist}_2,p_1,p_2)$-sensitive 的哈希函数族,可通过组合获得一新的哈希函数族$(\mathrm{dist}_1,\mathrm{dist}_2,p_1',p_2')$-sensitive. 即给定一个$(\mathrm{dist}_1,\mathrm{dist}_2,p_1,p_2)$-sensitive 的哈希函数族 $\boldsymbol{G}$,通过组合 β 个 α-AND 构造(即一个 β-OR 构造),可获得一个新的$(\mathrm{dist}_1,\mathrm{dist}_2,1-(1-p_1^{\alpha})^{\beta},1-(1-p_2^{\alpha})^{\beta})$-sensitive 哈希函数族 $\boldsymbol{G}'$,如图1所示.

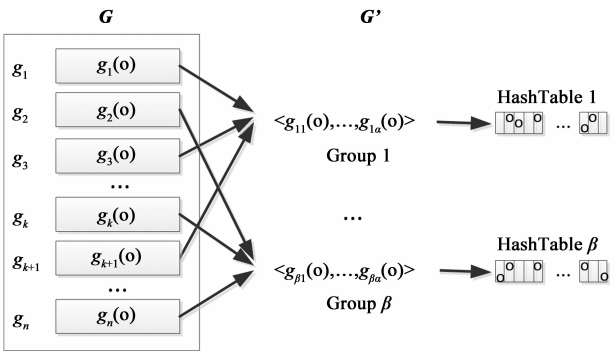

图1 C2SLSH算法的AND-OR构造过程

C2SLSH 对固定个数的哈希函数的结果进行冲突次数统计并排序,间接实现了更高效的 α-AND 构造和 β-OR 构造. 也就是说,C2SLSH 算法实现的 α-AND 构造是从 n 个 LSH 函数 $g_1,g_2,\cdots,g_n$ 中任意选取 α 个组合而成的,这里的 $\alpha(1\leqslant\alpha\leqslant n)$ 即是统计数;这种组合的个数共有 $\beta=C_n^{\alpha}=\frac{n!}{\alpha!\ (n-\alpha)!}$种,即是 β-OR 构造. 在查找 k 近邻结果的时候,首先在冲突次数最多的集合中寻找结果. 如果结果不足 k 个,再改变 α 的取值,每次减小1,一直找到 k 个结果并返回. 根据定理1,对于 C2SLSH 算法的 AND-OR 方案,令 $u=p(\sigma)$代表相似对象在一个哈希表中冲突的概率,假设这个过程中 α 的值改变了 γ 次,根据 α 和 γ 的取值,可以得出相似对象冲突的概率 $\varepsilon(u)=\sum_{i=n-\gamma}^{n}\binom{n}{i}u^i(1-u)^{n-i}$.

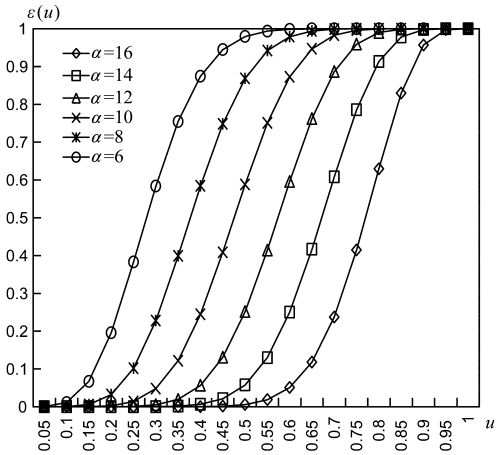

图2 当α变化时$\varepsilon(u)$曲线的变化趋势

通过计算,由图2可知,当 $n=20$ 并给定一个较大的 α 值时,存在一个区间$(0,t_1)$使得 $\varepsilon(u)$ 接近0并且缓慢递增;也存在一个相对应的区间$(t_2,1)$使得 $\varepsilon(u)$ 接近1并且缓慢递增. 最终可以得到一个结论:$\varepsilon(u)$在区间$[t_1,t_2]$内快速上升,并且 $\varepsilon(u)$的快速上升区间会随着 α 的递减而缓慢向左移动,也就是说,我们可以通过改变 α 的值来改变 $\varepsilon(u)$的快速上升区间,从而影响 kNN 查询结果.

从上述理论分析过程也可以看出,C2SLSH 算法之所以高效,有以下两方面的原因:第一,由于数据对象经过每个哈希函数计算之后就会对应一个数据副本,我们在设计 AND-OR 结构时,使用了较少的 LSH 函数,因此 C2SLSH 算法的空间复杂度相对于传统的 AND-OR 构造大幅度降低;第二,C2SLSH 算法方案根据 LSH 相似度即冲突计数排序的方式返回最终结果集,相对于传统 AND-OR 及其相关算法而言,能够保证快速的查询时间.

5 C2SLSH 的分布式实现

为满足对大数据的处理需求,本节在 MapReduce 框架上实现 C2SLSH 算法来处理 kNN 问题. 首先选取合适的哈希函数,然后计算数据对象的哈希值,得到相应的哈希索引值,索引文件由分布式系统集群共同维持,索引构造的目标是快速准确地处理数据对象的近邻查询. 查询时首先计算查询对象的哈希值,然后在索引中

查找与查询对象具有相同哈希值的对象,确定每个查询对象匹配的候选对象集合,并对集合中的对象计数并排序,将冲突次数最多的对象作为结果依次输出.

图 3 展示了 LSH 函数和 MapReduce 框架紧密结合构建 LSH 索引的过程. 建立索引的过程是一个 MapReduceJob 过程. 为减少分布式计算过程中的数据传输量,在数据预处理阶段我们为所有的数据对象设置唯一的 id 号. 分布式 LSH 索引构造以及查询过程如下.

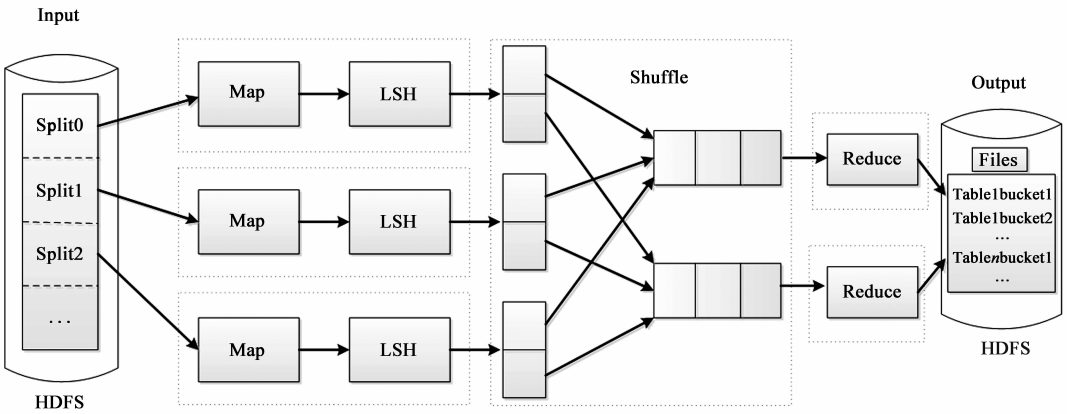

图3　MapReduce建立C2SLSH算法索引过程

(1)计算哈希值:在图 3 中 Map 阶段,预先初始化所选择的函数,Map 阶段使用一致的函数参数,Map 阶段计算对象的哈希值,由哈希函数及其哈希桶形成组合对 Intpair(Intpair 是自定义类,可以保存两个整型数值)作为 MapReduce 过程中的 key,而对象的 id 号作为 value,最终 Map 阶段输出键值对为((Table,bucket)id).

(2)索引文件分配:在图 3 中 Reduce 阶段,对相同桶内的数据进行归并,生成索引文件,由分布式文件系统 HDFS 维持,每个哈希桶对应 HDFS 中的一个文件. 由于将数据对象设置了 id 标识,也使得索引文件占用的存储空间更小,便于调到内存中处理. 本阶段将具有相同哈希值的 ids 保存到同一个二进制文件中,索引文件内容为键值对((Table,bucket),$id_1 id_2 \cdots$). 至此,构建索引的过程完成.

kNN 查询过程,首先使用索引构造过程中所选定的 LSH 函数计算查询对象的哈希值,接着根据哈希值选择将被统计计数的候选对象集,冲突计数排序的 kNN 查询处理过程使用两个 MapReduce Job 来完成,最终将为每个查询对象输出 k 近邻结果集. 处理流程如图 4 所示.

(3)相关候选者统计:在图 4 中 Map1 阶段,输入文件会被分割成多个输入分片,InputFormat 类可以直接读取二进制存储的文件,获取相应的 key 和 value 作为输入. Map1 阶段为每个查询对象输出一个组合对,当输入分片中的所有候选者被处理完,这时 key 是查询对象,value 是 id 及它们和查询对象的冲突统计,此阶段的输出为(query,(id,1)).

(4)候选者计数:在图 4 中 Reduce1 阶段,将 Map1 阶段生成的二元键值对分发到 Reduce1 阶段,对产生的结果进行聚合. 处理中将(query,(id,1))转变为((query,id),1),此时 key 值为(query,id),聚合具有相同 key 值的对象集并统计计数,生成(query,id),count))键值对,这里 count 是查询对象和候选对象的冲突次数,最终结果输出到 HDFS 上.

(5)候选集结果排序:接下来对第一个 MapReduce Job 的输出结果按计数降序排序. 在这个过程中,不仅对 key 进行排序,也要对相同 key 的 values 进行排序,所以接下来我们使用自定义的二次排序方法. 在图 4 中 Map2 阶段对每一个读入的(query,(id,count))键值对变换为(query,count) id),然后 Shuffle 阶段进行二次排序,排序结果发送到 Reduce2 阶段.

(6)返回最后结果:在图 4 中 Reduce2 阶段,对于 kNN 查询,我们为每个查询对象返回前 k 个结果,输出形式为(query,id) count)). 这种处理方式,可以在一个 MapReduce 任务中同时处理多个查询,当查询集较大时也能够使用该方法解决 kNN 问题.

6　实验

6.1　实验设置

实验在 16 台 PC 机组成的集群上进行,其中 1 台作为 Master 节点,其余 15 台都可以作为 Slave 节点. 每台 PC 配置相同:CPU 为 Intel Core i3 3.4GHz,8GB 内存,操作系统为 CentOS6.5,Hadoop 版本为 1.2.1,JDK 版本为 1.6. 采用两个真实数据集来评估本文的方法,一个是 LabelMe 图像数据集[18],每幅图像被表示为 512 维的 GIST 特征. 另一个是 Tiny 图像数据集[19],每幅图像被表示为 384 维的 GIST 特征. 这些数据集已被用于评价其他 LSH 算法的性能[20].

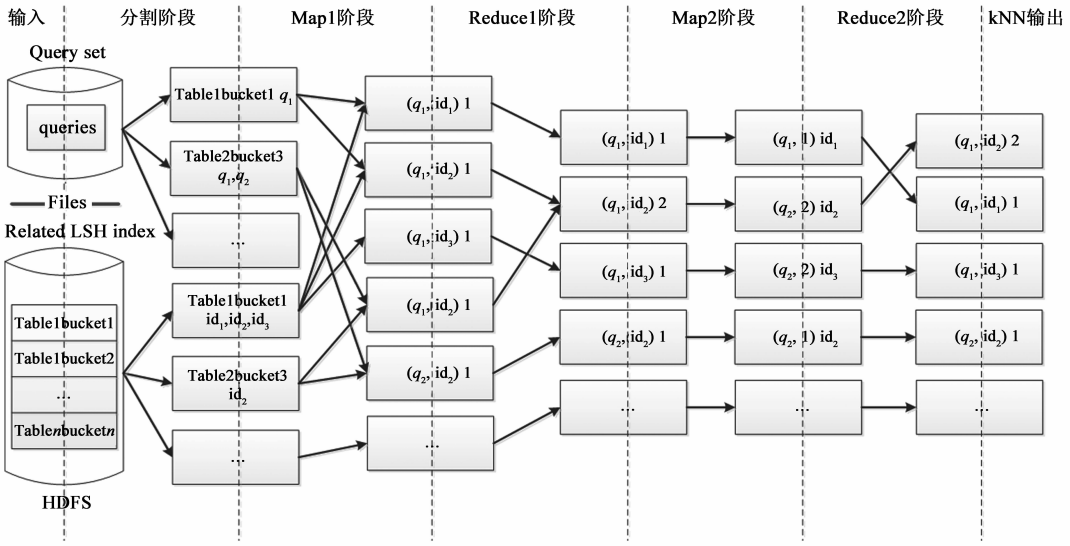

图4 kNN查询处理过程

6.2 实验结果与分析

实验对比算法包括:基本的线性扫描、RankReduce[21]和 LSHZ(LSHZ-order).

在 Hadoop 平台上利用 LSH 高效处理高维数据的 kNN 查询,据我们所知,目前只有 RankReduce 算法. 它采用"过滤 + 验证"过程,首先从所有数据集获得查询结果的候选集,然后在候选集中,计算与查询对象之间的距离并排序,从而获得最终的结果集. RankReduce 的计算量较大,尤其是验证过程. 另外,对于不同的哈希表,不同候选集中可能包含较多相同的对象,存在重复计算,同样需要较大的计算量.

另外,我们认为,将 LSB[22]和 H-zkNNJ[23]结合,也可处理高维数据的 kNN 查询. 对于高维数据集,LSB 分两个阶段处理:首先用 LSH 降维,然后将降维后数据转换成 Z-order 值,并用 B-tree 来索引. 但 LSB 算法是集中式平台下的研究成果,不能直接迁移到分布式平台上使用. H-zkNNJ 是用 Z-order 解决多维大数据 kNN 连接的方法,但当数据维度达到30 维以上时,算法效率会明显降低[23],显然无法直接用于高维数据处理. 我们把结合算法称为 LSHZ 算法,并实验比较.

6.2.1 准确率与精确性

评估准确性的方法,是将其与线性扫描所有数据做对比,并将线性扫描过程作为一个 MapReduce Job 来实现. 对于给定 k 值的 kNN 搜索算法的性能度量,我们使用 $C(v)$来代表正确率,线性扫描 kNN 返回结果集使用 $N(v)$表示,$C(v)$就是 $N(v)$在所提算法得到的结果集 $I(v)$中所占的比例,可以使用下面这个公式来表示:

$$C(v)=\frac{|N(v)\cap I(v)|}{|N(v)|}$$

实验使用数量为 50000 的 LabelMe 数据集和 Tiny 数据集,改变哈希函数的个数 m,同时改变桶宽 w 的大小,来测定算法的准确性. 在这两个数据集上使用相同 kNN 查询且令 $k=10$ 结果如图 5 所示,为提高正确率,在不改变桶宽的情况下,需增加哈希表个数. 固定哈希表的数目并改变桶宽时,也可调整算法的正确率. 因此可以选择适当数目的哈希表和合适的桶宽保证较高的正确率. 由于每个哈希表中都会创建所有数据的副本,因此实验中需要权衡存储成本与执行时间.

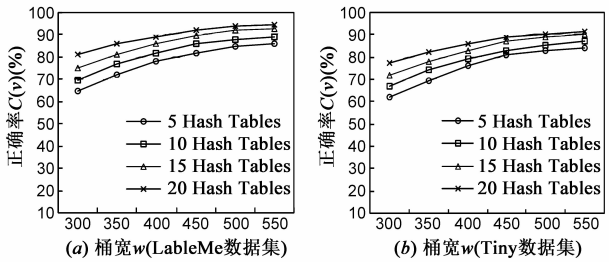

图5 桶宽大小与哈希表个数对正确率产生的影响

LSH 是一种随机的算法,它并不能保证返回精确最近邻,但会高概率得到近似最近邻,我们采用和文献[11]相同的测量方法来评估查询结果的精确性. 特别地,对于一个查询对象 $\boldsymbol{q}$ 通过相应算法所得到的 kNN 查询结果为 $\boldsymbol{o}_1,\cdots,\boldsymbol{o}_k$,并根据他们与查询对象 $\boldsymbol{q}$ 距离的不减顺序排序,使用 $\boldsymbol{o}_1^*,\cdots,\boldsymbol{o}_k^*$ 代表线性扫描所得到的到查询对象 $\boldsymbol{q}$ 的 kNN 结果,同样以距离的不减顺序进行排序. 那么,Rank-i 近似指标可以定义为 $R_i(\boldsymbol{q})=\frac{\|\boldsymbol{o}_i,\boldsymbol{q}\|}{\|\boldsymbol{o}_i^*,\boldsymbol{q}\|}$,其中 $i=1,\cdots,k$ 并且 $\|\cdot\|$代表欧氏距离,平均精确指标被定义为 $\frac{1}{k}\sum_{i=1}^{k}R_i(\boldsymbol{q})$,平均精确指标是一个大于 1 的数值,其值越接近 1,结果越精确. 给定一个

查询集合 Q 时,其中所有查询对象的平均精确指标作为查询精确性的最终测度,这个平均值被称为总体平均精确指标,如图 6 所示.

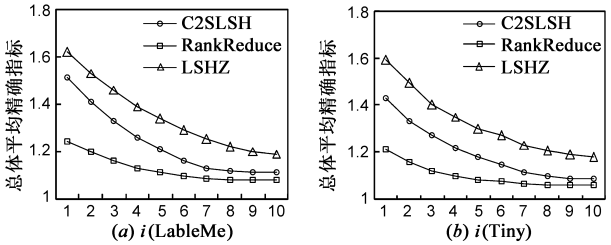

图6　查询结果的平均精确指标

从图 6 中可以看出,在 LableMe 和 Tiny 数据集上搜索的 Rank-i 总体平均精确指标. 这里实验设置哈希函数为 20 个,桶宽 w 为 400,根据 H-zkNNJ 实验结果,两个偏移量副本就可以达到较好的效果,我们在这里设置 Z-order 偏移量副本数目为 2. 结果表明,与线性扫描方法相比,C2SLSH 方法、RankReduce 方法以及 LSHZ 方法所返回的结果对象的先后顺序都有误差,随着 k 值的增大,三种方法的结果越来越接近. 尽管 C2SLSH 算法不能返回查询对象 k 个结果的精确顺序,但能保证查询结果的错误率较小,从而满足近似最近邻的查询需求.

6.2.2　运行时间

本实验评估 C2SLSH 算法时间效率是与两种分布式下 LSH 相关算法做对比. C2SLSH 对所有查询对象的候选结果集统计计数并排序,根据冲突次数获得最终结果集 $I(v)$,因此有较好的时间效率. 实验使用四个节点的集群,数量为 20000、40000、60000、80000、100000 的 LableMe 及 Tiny 数据作为实验数据集,对比测量同等实验条件下三种算法的查询时间. 实验结果如图 7 所示,结果表明相同规模的数据要达到相同精确率时,C2SLSH 方法更快,约是 RankReduce 方法的运行时间的三分之一甚至更少.

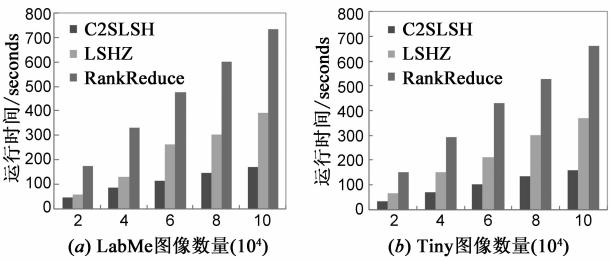

图7　不同数量真实图像数据集上运行时间

另外,为了测定桶宽 w 的变化对 C2SLSH 算法运行时间的影响,同样采用四个节点集群作为实验平台,固定哈希表数目为 20,使用数量为 50000 的 LableMe 和 Tiny 图像数据集. 使得桶宽 w 是一个连续变量,对桶宽大小提供了较小的控制,如图 8 所示,在固定哈希表个数的情况下,随着桶宽的变大,虽然可以提高查询结果的精确率,但是所需运行时间也会变长.

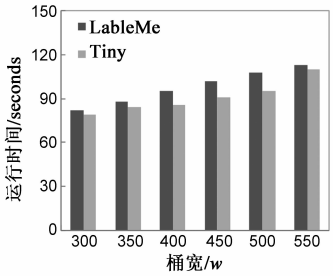

图8　桶宽w变化对C2SLSH算法运行时间产生的影响

6.2.3　可扩展性

为了测试 C2SLSH 算法的可扩展性,我们在不同节点个数的分布式集群上,采用 6 组不同数量的图像数据进行测试. 实验集群的节点个数采用递增的形式(4 个、8 个、16 个);对于不同数据集 LableMe 和 Tiny 的 LSH 参数设置,我们根据之前的实验结果,选定 20 个哈希函数,固定桶宽为 400,使用 20000、40000、60000、80000、100000 的 LableMe 图像数据集以及 Tiny 图像数据集进行实验对比. 随着数据量的增长,查询运行时间增长越缓慢则表明分布式索引策略越好. 从图 9 中可以看到随着数据量的增加,查询时间增加相对缓慢,另外,增加集群节点个数时,查询时间也会随之减小,所以 C2SLSH 适合分布式平台,具有较好的稳定性和可扩展性.

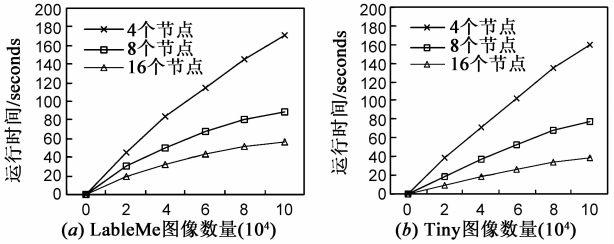

图9　C2SLSH算法在多台集群上的可扩展性

7　总结

本文为解决高维大数据的 kNN 查询问题提出了一个基于 LSH 适合分布式平台的高效算法,首先对这个算法进行了理论分析,然后在 Hadoop 分布式平台上实现了该算法并测试了该算法的可行性. 通过实验,验证了该算法在分布式平台上的有效性. C2SLSH 算法不仅可以处理高维大规模数据的 kNN 问题,也可以把它扩展到其他感兴趣的应用上,如近似检测,文档分类或者推荐系统等.

参考文献

[1] Shi J R, et al. Metric learning for high-dimensional tensor Data[J]. Chinese Journal of Electronics, 2011, 20(3): 495-498.

[2] 巩敦卫,王更星,孙晓燕. 高维多目标优化问题融入决策者偏好的集合进化优化方法[J]. 电子学报,2014,42(5):933 -939.
GONG Dun-wei, WANG Geng-xing, SUN Xiao-yan. Set-based evolutionary optimization algorithms integrating decision-maker's preferences for many-objective optimization problems[J]. Acta Electronica Sinica,2014,42(5):933 -939. (in Chinese)

[3] 赵永威,郭志刚,李弼程,等. 基于随机化视觉词典组和上下文语义信息的目标检索方法[J]. 电子学报,2012,40(12):2472 -2480.
ZHAO Yong-wei, GUO Zhi-gang, LI Bi-cheng, et al. Object retrieval method based on randomized visual dictionaries and contextual semantic information[J]. Acta Electronica Sinica,2012,40(12):2472-2480. (in Chinese)

[4] Guttman A. R-trees:A Dynamic Index Structure for Spatial Searching[M]. USA:ACM Press,1984.

[5] Bentley J L. K-d trees for semidynamic point sets[A]. Proceedings of the Sixth Annual Symposium on Computational Geometry[C]. USA:ACM Press,1990. 187 -197.

[6] Gionis A, Indyk P, Motwani R. Similarity search in high dimensions via hashing[A]. Proceedings of the 25th International Conference on Very Large Data Bases[C]. USA: VLDB Press,1999, Vol 99. 518 -529.

[7] Indyk P, Motwani R. Approximate nearest neighbors: towards removing the curse of dimensionality[A]. Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing[C]. USA:ACM Press,1998. 604 -613.

[8] Qian Jiangbo, Zhu Qiang, Chen Huahui. Multi-granularity locality-sensitive bloom filter[J]. IEEE Transactions on Computers,2015,64(12):3500 -3514.

[9] Lv Q, Josephson W, Wang Z, et al. Multi-probe LSH: efficient indexing for high-dimensional similarity search[A]. Proceedings of the 33rd International Conference on Very Large Data Bases[C]. USA: VLDB Endowment Press, 2007. 950 -961.

[10] 龙柏,孙广中,熊焰,陈国良. 一种基于多核机群架构的混合索引结构[J]. 电子学报,2011,39(2):275 -279.
LONG Bai, SUN Guang-zhong, XIONG Yan, CHEN Guo-liang. A hybrid index structure based on multi-core cluster[J]. Acta Electronica Sinica,2011,39(2):275 -279. (in Chinese)

[11] Gan Junhao, et al. Locality-sensitive hashing scheme based on dynamic collision counting[A]. Proceedings of the ACM SIGMOD International Conference on Management of Data[C]. USA:ACM Press,2012. 1 -12.

[12] Hadoop:Open-source Implementation of mapreduce[OL]. http://lucene.apache.org/hadoop/. 2012.

[13] Koga H, Oguri M, Watanabe T. MIXED-LSH: Reduction of remote accesses in distributed locality-sensitive hashing based on L1-distance[A]. Proceedings of IEEE 26th International Conference on Advanced Information Networking and Applications (AINA)[C]. USA: IEEE Press, 2012. 175 -182.

[14] Bahmani B, Goel A, Shinde R. Efficient distributed locality sensitive hashing[A]. Proceedings of the 21st ACM International Conference on Information and Knowledge Management[C]. USA:ACM Press,2012. 2174 -2178.

[15] Panigrahy R. Entropy based nearest neighbor search in high dimensions[A]. Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm [C]. USA:ACM Press,2006. 1186 -1195.

[16] Datar M, Immorlica N, Indyk P, et al. Locality-sensitive hashing scheme based on p-stable distributions[A]. Proceedings of the Twentieth Annual Symposium on Computational Geometry[C]. USA:ACM Press,2004. 253 -262.

[17] Rajaraman A, Ullman J D. Mining of Massive Datasets [M]. England:Cambridge University Press,2011.

[18] LabelMe, the Open Annotation Tool[OL]. http://labelme.csail.mit.edu.

[19] Python Version of Matlab Code for Accessing the Tiny Image Dataset[OL]. http://horatio.cs.nyu.edu/mit/tiny/data/index.html.

[20] Pan J, Manocha D. Bi-level locality sensitive hashing for k-nearest neighbor computation[A]. Proceedings of IEEE 28th International Conference on Data Engineering (ICDE)[C]. USA:IEEE Press,2012. 378 -389.

[21] Stupar A, Michel S, et al. Rankreduce processing k-nearest neighbor queries on top of mapreduce[A]. Proceedings of the 8th Workshop on Large-Scale Distributed Systems for Information Retrieval[C]. USA:ACM Press,2010. 13 -18.

[22] Tao Y, Yi K, et al. Efficient and accurate nearest neighbor and closest pair search in high-dimensional space[J]. ACM Transactions on Database Systems (TODS),2010, 35(3):20.

[23] Zhang C, Li F, Jestes J. Efficient parallel kNN joins for large data in mapreduce[A]. Proceedings of the 15th International Conference on Extending Database Technology [C]. USA:ACM Press,2012. 38 -49.

作者简介

王忠伟 男,1988 年出生于河南商丘. 宁波大学信息科学与工程学院硕士研究生,主要研究方向为大数据、数据挖掘.
E-mail:huanfengwei@163.com

陈叶芳(通信作者) 女,1973 年出生于浙江省安吉县. 宁波大学讲师,主要研究方向为数据处理和挖掘.
E-mail:chenyefang@nbu.edu.cn