



一种高维大数据全 k 近邻查询算法

王忠伟, 陈叶芳, 肖四友, 钱江波

(宁波大学信息科学与工程学院 宁波 315211)

摘要: 全 k 近邻 (all k -nearest neighbor, $AkNN$) 查询, 是 k 近邻查询的一个变型, 旨在在一个查询过程中为给定数据集的每个对象确定 k 个最近邻。提出了一种在 Hadoop 分布式平台下处理高维大数据的 $AkNN$ 查询算法。首先使用行条化思想结合 p -stable LSH 算法将高维数据对象降维, 然后结合空间填充曲线 Z-order 的优良特性, 把降维后的数据嵌入一维空间中, 接着进行范围查询。整个过程使用 MapReduce 框架分布式并行处理。实验结果表明, 所提出的算法可以高效处理高维大数据的 $AkNN$ 查询。

关键词: 高维; $AkNN$; MapReduce; 行条化; 局部敏感散列; Z-order

doi: 10.11959/j.issn.1000-0801.2015171

An $AkNN$ Algorithm for High-Dimensional Big Data

Wang Zhongwei, Chen Yefang, Xiao Siyou, Qian Jiangbo

(Faculty of Electrical and Computer Science, Ningbo University, Ningbo 315211, China)

Abstract: A new variant of k nearest neighbor queries, which called as all k -nearest neighbor queries ($AkNN$), is a process to search the k nearest neighbors of each object in a data set. An $AkNN$ query algorithm for high-dimensional big data on the Hadoop system was proposed. Using the banding technique and the p -stable LSH algorithm, dimensionality reduction was performed, then the data was embedded in a Z-order curve. The preprocessed data were continued to be treated on a MapReduce framework in a distributed parallel manner. Experimental results show that the proposed algorithm can efficiently handle $AkNN$ queries for large-scale high-dimensional data.

Key words: high-dimensional, $AkNN$, MapReduce, banding, locality sensitive hashing, Z-order

1 引言

在高维空间的 kNN 查询中, 数据对象的每一维度表示一个给定的属性或特征, 这些对象通过如欧氏距离等距

离度量函数进行比较。这样, 两个对象之间的相似性就是相应的属性值间的距离。对于这种 kNN 查询, 也就是检索数据中到给定的查询对象 q 距离最小的 k 个对象, 已经有很多方法。然而, 在数据挖掘和相似性搜索的很多应用中,

收稿日期: 2015-03-20; 修回日期: 2015-07-09

基金项目: 国家自然科学基金资助项目 (No.61472194), 浙江省自然科学基金资助项目 (No.LY13F020040), 宁波市自然科学基金资助项目 (No.2014A610023, No.2015A610119), “信息与通信工程”浙江省重中之重学科开放基金资助项目 (No.xkx1423)

Foundation Items: The National Natural Science Foundation of China (No.61472194), Zhejiang Provincial Natural Science Foundation (No.LY13F020040), Ningbo Natural Science Foundation (No.2014A610023, No.2015A610119), Zhejiang Key Discipline Fund of Information and Communication Engineering (No. xkx1423)

论文引用格式: 王忠伟, 陈叶芳, 肖四友等. 一种高维大数据全 k 近邻查询算法. 电信科学, 2015171

Wang Z W, Chen Y F, Xiao S Y, et al. An $AkNN$ algorithm for high-dimensional big data. Telecommunications Science, 2015171

在处理大规模数据时有必要生成一个预先知道的全局的 k NN 查询结果。更准确地说,在内集(或数据库) S 上为外集(或查询集合) R 中的每一个对象执行一次 k NN 查询,称为 Ak NN(all k -nearest neighbor,全 k 近邻)查询。 Ak NN 查询是 k NN 查询的一个变型,旨在一次查询中为给定的数据集中的每个对象确定 k NN。调查研究发现,许多数据挖掘算法可以通过 Ak NN 计算来加速,如并行 k NN 分类和 k -means 聚类^[1-5]。

高维数据的处理面临的最大问题是“维度灾难”,当数据对象的维度超过特定的大小时,许多研究者已经确认现存的低维空间索引方法,如 R 树或者空间填充曲线等不能有效地处理高维空间的数据^[6-8]。同时,所有基于索引技术的空间划分来解决近邻查询问题的方法,当处理高维数据的时候,其效果还不如线性扫描方法。另外,研究人员还指出,通过使用近似近邻查询,既可以避免维度灾难问题的发生,还可以达到和精确查询一样的效果^[9,10]。基于上述事实,随着所需处理的数据量日趋庞大,促使设计出一种支持分布式并行处理的 Ak NN 算法。MapReduce 是分布式并行环境下处理大规模数据的一个基础框架,可以实现可扩展数据处理。本文使用基于 MapReduce 数据处理的一个十分流行的软件 Hadoop^[11]。目前还没有基于 MapReduce 框架高维大数据 Ak NN 的研究,但是在 MapReduce 框架上的连接查询处理是一个热门话题^[12-14],该问题与 Ak NN 问题有些类似。对于 Ak NN 查询,现有的算法仅限制使用于集中式环境的低维数据,H-zkNNJ^[15]是一个解决多维大规模数据 k NN 连接的方法,当它的维度达到 30 维以上的时候,算法效率就会降低,显然对于高维数据的处理并不合适。对于在 Hadoop 上如何高效处理高维数据的 Ak NN 查询,还没有文献对此进行研究。

本文提出一种在 Hadoop 平台上高效处理高维大数据的 Ak NN 查询算法。算法思想是:首先将高维数据进行分布式并行降维处理,即利用行条化思想^[16],把处理行条数据的散列函数选定为 p -stable LSH^[17],使用的 LSH(locality sensitive hashing,局部敏感散列)函数可以根据经验选定参数,只要保证近似的对象仍然被映射到近似的空间中;然后在多维空间中确定数据对象的近似映射空间;最后通过考虑局部范围内可能的 k NN 对象,即为数据集中的每个数据对象确定 k NN 对象,从而获得一个全局的 Ak NN 结果。本文贡献如下。

· 结合行条化思想与 LSH 算法以及空间填充曲线

Z-order 技术,提出一种高维大数据 Ak NN 查询算法,称为 ZbH((Z-order banding LSH)算法。

- 将所提出的算法在 Hadoop 分布式平台下实现,并解决了实现过程中所遇到的各种问题,如保证降维后数据的邻近性,查询块边界范围定义等。
- 使用两个真实高维大数据集进行实验,并采用与 LSH 直接结合 Z-order 的方法(LSHZ 算法)进行对比,验证了算法的有效性和实用性。

2 相关知识

2.1 Ak NN 定义

如第 1 节中所提到的, Ak NN 查询旨在为给定的外集 R 中的每一个对象在内集 S 中检索 k 近邻,集合 R 和集合 S 可以拥有相同的数据。集合 Q 中的对象 q 的 k 近邻集合定义如下。

定义 1 (k NN)令 $q \in R^n$ 代表一个查询对象, $Q \subset R^n$ 代表一个数据对象集合,令 d 代表在 R^n 上的距离度量,那么对于任何正整数 k ,集合 $kNN(q, Q)$ 代表集合 Q 中对象 q 的 k 近邻,令 p 和 $\hat{p}$ 代表 Q 中的两个对象,那么有如下定义。

$$|kNN(q, Q)| = k \quad (1)$$

$$\begin{aligned} \forall p \in kNN(q, Q), \\ \forall \hat{p} \in \{Q - kNN(q, Q)\} \Rightarrow d(q, p) \leq d(q, \hat{p}) \end{aligned} \quad (2)$$

定义 2 (Ak NN 查询)令 $R, S \subset R^n$ 代表两个相同的数据集, d 代表在 R^n 上的距离度量,那么对于任何正整数 k ,一次 Ak NN 查询结果集 $AkNN(R, S)$,满足:对于 $\forall r \in R, \forall s \in S$,如果 $(r, s) \in AkNN(R, S)$,那么 $(r, s) \in kNN(r, S)$ 。

Ak NN 查询遵循如下原则:给定两个集合 R 和 S ,想要为 $r \in R$ 的所有元素寻找 k NN,最直接的方法是将集合 S 直接分成多个分组,但这明显不够高效,因为每次查询需要将集合 S 中的所有对象遍历一次。一个高效的算法需要使用多个元素将 S 中的对象分成多个子组,使得空间上邻近的对象落入相同的分组中。接着,为了最小化搜索空间,重要的步骤是确定查找 R 的子分组中每一个元素的 k 近邻时, S 中的哪些子分组应该被搜索。因此,查找空间,也就是包含 R 的子分组所有元素 k 近邻的区域应该被界定。

2.2 LSH 定义及行条化思想

对于高维空间数据的处理,寻找相似对象的一般过程是将所有对象相互之间进行比较并且从它们之间识别出相似对象。很明显,查询全部的对象组合相当浪费时间。为



了解决这个问题,应该把注意力集中到那些最可能相似的对象上,才能避免计算全部的对象组合。这个问题的解决途径叫做 LSH^[10]。LSH 是由 Indyk 和 Motwani 提出的一种高维索引结构,它的基本思想是针对数据对象集,利用一组特定的散列函数来建立散列表,从而使得在一定的相似度量条件下,相似对象映射到相同区域的概率大于非相似对象。LSH 算法具有处理高维数据的优良特性,其最大优势是可以保证高维数据的空间邻近性,可用来保证空间相近的对象高概率地映射到同一个桶中,而空间距离远的对象以很低的概率被散列到同一个桶中。

定义 3 (LSH) 假设有一个使用距离度量 d 的对象空间, $dist_1, dist_2$ 为距离度量 d 的两个距离。如果散列函数族 H 中的每一个函数 h 均满足如下两个条件,则散列函数族 H 对于空间中任意的 x 和 y 称为 $(dist_1, dist_2, p_1, p_2)$ -sensitive。

- 如果 $d(x, y) \leq dist_1$, 那么 $Pr[h(x)=h(y)] \geq p_1$;
- 如果 $d(x, y) \geq dist_2$, 那么 $Pr[h(x)=h(y)] \leq p_2$ 。

其中, $dist_1 < dist_2, p_1 > p_2 \in [0, 1]$ 。

LSH 函数适合许多距离度量, 最适合 l_p 范式的是基于 p -stable 分布的 LSH 函数族^[17]。在这种情况下, 对每一个高维数据对象 v , 散列方案考虑 k 个独立的散列函数形式, 表示如下:

$$h_{ab}(v) = \left\lfloor \frac{a \cdot v + b}{w} \right\rfloor \quad (3)$$

其中, a 是一个与 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 是桶宽参数; b 是从 $[0, w]$ 中随机选取的实数。本文关注 $p=2$ 欧氏距离的情况, 是正态分布 $N(0, 1)$ 的 2-stable 分布。

LSH 的一般理论是首先多次散列每一个对象, 使得相似的对象被散列到同一个桶中。然后, 同一个桶中的对象被认为是彼此的候选相似对象, 也就是仅检查候选对象来确定相似对象。但是在实际实验中, 为了保证正确率, 这种方法需要搜索的对象仍然有很多。在这种情况下一种常规的方法是行条化技术^[16], 它能起到作用。一种适当的实现 LSH 的方法是把对象维度分割成 b 个行条, 这样每个行条中包含 r 行。然后对于每个行条, 为其选择一个散列函数并且这个散列函数将会与其他行条的散列函数一起, 使用 b 个整数为每一个对象生成大量的散列桶。图 1 展示了部分对象集合矩阵包含 12 行, 被分成 4 个行条并且每个行条有 3 行。通过这种方式可以实现 LSH 算法的 AND-OR 构造过程, 从而减少结果可能存在的假阳性和假阴性。

band1	...	1 0 0 0 2 3 2 1 2 2 0 1 3 1 1	...
band2			
band3			
band4			

图 1 将对象分割成 4 个行条, 并且每个行条包含 3 行

从图 1 可以看到, 在第 1 个行条的第 2 列和第 4 列具有相同的向量 $[0, 2, 1]$, 并且在对第 1 个行条散列的时候第 2 列和第 4 列明显会被散列到同一个桶中。因此, 不管这些列在其他 3 个行条中是否相似, 这个对象组合可以作为一个候选相似组合。更进一步地说, 在第 1 个行条中这两列没有被确认为候选相似组合, 但在其他 3 个行条中, 还有其他 3 次机会成为候选相似组合。在一定比率上, 两个非常相似的对象会在多个行条中产生冲突。因此, 相对于那些非相似的对象组合, 这种方法直接高概率地提取出相似对象, 使之成为候选相似组合。

3 ZbH 算法思想

高维数据的 k -NN 查询问题的相关研究比较少, 这就启发寻找高维数据处理的方法。在保证空间局部性的情况下, LSH 算法可以将高维数据降到多维空间, 这样就可以将高维空间的近邻查询问题转化到多维空间。为了最小化查询空间, 需要寻找可能的线性计算开销, 利用空间填充曲线 Z-order 近似处理, 将多维数据映射到一维空间, 作为替代精确近邻查询的方法。

3.1 LSHZ 算法思想

对于高维大数据的 MapReduce 分布式处理, 将 LSH 与 Z-order 方法相结合, 首先使用 LSH 算法对高维数据降维, 然后使用 Z-order 方法完成对多维数据的近似处理, 称为 LSHZ 算法。LSHZ 算法的提出受 LSB 算法^[18]的研究启发, 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, 但可以借鉴 LSB 的算法思想: 对于一个高维数据集 D , 首先把数据集中的对象 v 转换成 m 维的对象 $G(v)$, 然后再转换成一个 Z-order 值 $z(v)$ 。 $z(v)$ 是个标量, 简称 z 值, 能够通过一个 B-tree 来索引所有的 Z-order 值。使用 p -stable LSH 散列函数族: $H_{ab}(v) = \left\lfloor \frac{a \cdot v + b}{w} \right\rfloor$, a 是随机的, 生成不同散列函数: $H_1(v), H_2(v), \dots, H_m(v)$; 将对象 v 转换成 $G(v)$, $G(v)$ 的维度 m , 得到 $G(v) = \langle H_1(v), H_2(v), \dots, H_m(v) \rangle$ 。再通过将它的二进制坐标值进行交叉计算得到一

个对象的 z 值,将 $G(v)$ 转换成 $z(v)$ 。例如,给定一个在二维空间的对象 (2,6), 它的二进制坐标对应的是 (010,110), 因此,它的 z 值是 011100=28。对于一个对象集合 D 的 z 值曲线,通过连接集合 D 中对象 z 值的数值顺序获得并且递归生成 z 型曲线。一个关键的计算 z 值的操作仅需要简单的位偏移操作,比较容易实现, z 值生成过程的具体例子如图 2 所示。

若直接使用 LSHZ 算法处理高维大数据,可能会存在以下两个方面的不足。

- 由于 LSH 函数本身的性质,需要多个 LSH 函数才能保证高维数据对象之间的差异性能较好地体现,这也是传统 LSH 算法结构需要进行多次 AND-OR 构造的原因,时间效率较低。
- 为了保证一定的正确率,LSHZ 算法每次使用 LSH 计算对象的全维度,在分布式平台下,这意味着建立多个原始高维数据对象的副本,对于 Hadoop 分布式计算平台来说开销太大,致使空间效率很低。

3.2 ZbH 算法思想

为解决 LSHZ 算法存在的不足,本节提出一种新的思路,称之为 ZbH 算法,算法的主要思想是在 LSHZ 算法基础之上加入行条化思想。利用行条化思想对高维数据对象分割处理,并使用 LSH 函数对行条内的数据进行处理,那么行条内数据的处理类似于传统 LSH 算法的 AND 操作,而行条之间对候选对象组合的选择,类似于传统 LSH 算法的 OR 操作。另外,Z-order 可以高效地保证低维空间近邻对象的邻近性。基于此,对行条化思想的 OR 操作方式使用 Z-order 曲线进行改善,结合 Z-order 对高维大数据进行高效的处理。

使用 ZbH 算法对高维数据的 k NN 查询处理过程包含两个阶段:数据对象降维阶段;近似处理阶段。在数据对

象降维阶段,结合行条化思想和 LSH 算法,为高维数据对象生成桶向量,将集合 R 和集合 S 对象生成相应的桶向量。近似处理阶段可以细分为生成候选集阶段和候选集筛选阶段。在生成候选集阶段,把 R 和 S 相应的桶向量集合使用空间填充曲线 Z-order 映射到一维空间中,然后将一维空间分段,使得每个分段中包含那些彼此空间距离相近的对象。因此,对于集合 S 中的每个对象 p ,它可能的连接候选集可以在 p 映射的桶中找到。在候选集筛选阶段,对于每一个来自集合 R 中的对象 q ,所有 k NN 候选集的对象通过计算它们到 q 对象的距离,算法返回到对象 q 距离最小的 k 个候选者作为 k NN,将这个处理过程称为 ZbH 算法。接下来,将呈现本算法更多的细节。

定义 4 (对象的桶向量)对于输入维度为 n 的对象 v ,按维度分割成 $m(m < n)$ 个段(也称行条),再使用随机散列函数族生成 m 个独立随机 p -stable LSH 函数 $h_1, h_2, h_3, \dots, h_m$,每个行条被对应的函数映射为一个散列值,输入对象 v 可表示为: $\{h_1(v), \dots, h_m(v)\}$,称为对象 v 的桶向量。

因此,生成对象的桶向量可以用来对高维数据对象降维,当数据对象维度很大的时候,对象的维度分布以及对象之间的差异性也可以被精确地捕捉到。对于高维大数据处理,可以针对行条内数据并行创建对象的桶向量。

接着,利用 z 值再将对象的桶向量所代表的数据对象映射到一维空间,将查询对象 q 的 k NN 查询转化成一维空间中的范围查询。如图 2 所示,在大多数情况下, z 值可以保证空间局部性,因此可以根据 q 对应的 z 值找到 q 的 k NN(也就是在 Z_D 中 z_q 位置的前后方向搜索,其中 Z_D 是数据集 D 在 Z 值上的映射)。然而,这样得到的结果有时候是不准确的,因此对输入的数据集 D 生成随机独立的 α 个偏移量副本,并且为每一个 D 的随机偏移量副本重复上述操

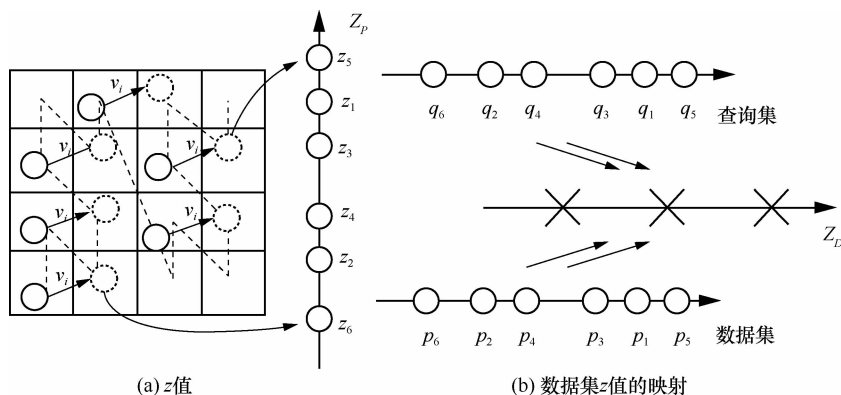


图 2 z 值生成过程



作。在实际实验中,只需要很少数量的随机偏移量就可以达到较高的准确率,实验结果也表明, $\alpha=2$ 就足够保证数据对象高概率的近似性。

从上述 z 值生成过程可以看到,对象的桶向量的每一个值就是每次散列处理后数据对象所映射的桶号,两个相似度较高的对象,会被多次散列到同一个桶中,它们之间的差异性会通过对象的桶向量的多次 Z-order 曲线操作很好地表现出来。

在 z 值曲线上进行范围查询时,其中每一次的 Z-order 曲线操作过程可以看成桶合并的过程,也就是说,根据对象的桶向量的 z 值将相近桶合并到了 Z-order 曲线上相近的分段中,使得可以应用 ZbH 算法对高维大数据的处理进行高效的查询空间界定。在 MapReduce 框架上实现上述思想不是一件简单的事情:需要巧妙地对给定的数据集合进行分区,才能达到一个好的负载平衡。接下来,将讨论如何结合 MapReduce 框架实现 ZbH 算法。

4 ZbH 算法分布式实现

本节在整体上介绍 ZbH 算法的分布式实现过程,包含两个主要阶段:降维和近似处理。使用行条化思想结合 p -stable LSH 函数的方法将原始数据对象降维;然后在降维处理后的多维数据对象上,使用空间填充曲线 Z-order 进行近似处理。这样就为高维大数据的 $AkNN$ 查询问题提出了一个可行的解决途径,使得所提出的方法可以高效地进行 $AkNN$ 查询的处理,下面是 ZbH 算法两个阶段的具体实现过程。

4.1 降维阶段分布式实现

本节使用一个 MapReduce 过程来实现对高维数据对象的降维。

首先,对于输入的原始高维数据对象,将其维度分成多个行条;然后,对于数据集中所有对象相应的每个行条,使用同样的 LSH 函数进行处理,每个数据对象的每个行条数据会被映射到一个桶中,将这个桶号作为这个行条对应的桶向量的一个具体值。这样设计有以下几个好处:将高维数据对象分成多个行条的做法,使得每次 LSH 计算的数据量变小,对于高维数据来说,这种做法也显得很灵活,所提出的算法可以适应任何维度的数据对象,可以根据原始数据对象的维度,具体指定降维后数据的维度;正如之前分析,LSH 函数能够保证相近的对象映射到同一个桶中,这样的降维处理方式,就保证原始空间相近的对象在经过降维处理后还能被映射到相近的桶中,保证了原始数据空间位置的局部性;另外,在实验的过程中也得到启示,由于降维之后的处理使用空间填充曲线 Z-order,所以降维之后的维度数目最好在 30 以内。

降维阶段分布式实现的概览如图 3 所示。在 map 阶段将为每一个原始数据对象添加唯一的 ID 标识,方便之后对数据对象的处理,在 reduce 阶段将每一个数据对象分成多个行条,并对每个对象相应行条内的数据使用 LSH 进行处理,最终得到原始高维数据降维后的桶向量,将这个桶向量作为原始数据对象的代表和后续步骤处理的输入数据。在实际实验中,需要根据集群的节点个数,指定具体的 mapper 和 reducer 个数,合理的 mapper 和 reducer 个数可以加速数据的处理。

本节实现了对高维数据对象的降维处理,并且保证了降维后原始数据对象的邻近性。将输出到 DFS 上的桶向量数据作为下一个处理过程的输入数据集,进行近似 $AkNN$ 查询处理。接下来介绍 Z-order 近似处理阶段的分布式实现的具体过程。

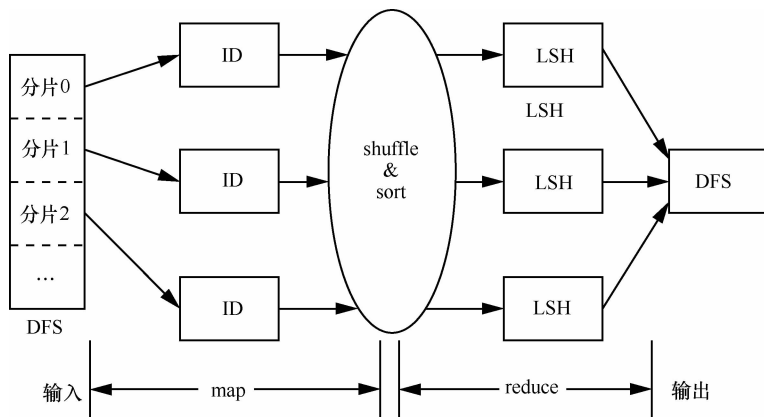


图 3 降维处理过程

4.2 Z-order 阶段分布式实现

本节使用 3 个 MapReduce 循环来实现 Z-order 阶段的近似处理,整个过程如图 4 所示。任务提交时,主节点首先生成 α 个偏移量 $\{U_1, U_2, \dots, U_\alpha\}$,并将它们保存到 DFS 的一个文件中,接着把这个文件添加到分布式缓存中,使得在初始化阶段和所有的 mapper 通信,这个过程如图 4 中的(1)所示。ZbH 算法的第一个 MapReduce 阶段,目标是通过偏移量 $U_i (1 \leq i \leq \alpha)$ 构造 R 和 S 的偏移的副本 R_i 和 S_i ,同时确定 R_i 和 S_i 的分区值。这个过程包括图 4 中的(2)~(5),也就是使用分布式缓存中所生成的偏移量计算 z 值,并从总体 z 值域中取样本,用来确定 R_i 和 S_i 的分区信息。

在 MapReduce 分布式平台上,对于各个从节点来说,最优的负载均衡是没有网络传输开销的,令集合 R 和集合

S 的每个块数量均等,使得时间计算上能保持一致。

将偏移的副本 R_i 和 S_i 分成若干块,其中 R_{ij} 和 S_{ij} 分别代表相应副本集合的第 j 个块,对于一个对象 r 的 z 值,如果 S_{ij} 没能包含 k 个向前(向后)的对象,有 $z_r \in R_{ij}$ 。很明显,可以考虑直接搜索 S_{ij} 的前后两个方向,也就是 $S_{ij-1}(S_{ij+1})$,并且重复相同的步骤,直到找到 k 个向前(向后)的对象。为避免在这个过程中检查多个块并保证搜索一个 S_{ij} 块是有效的,复制 S_{ij} 最近的块向前(向后)的 k 个对象(如果有必要,当 S_{ij-1} 和 S_{ij+1} 不足 k 个对象的时候,会进一步到达更远的块)。这样就保证了 S_{ij} 块中包含任意的 $r \in R_{ij}$ 的 z 值向前(向后)的 k 个对象,使用符号 $C_i(r)$ 表示数据对象 r 在第 i 次偏移副本中的候选集,复制 3 个相邻分组数据对象到 S_i 的实例如图 5 所示。

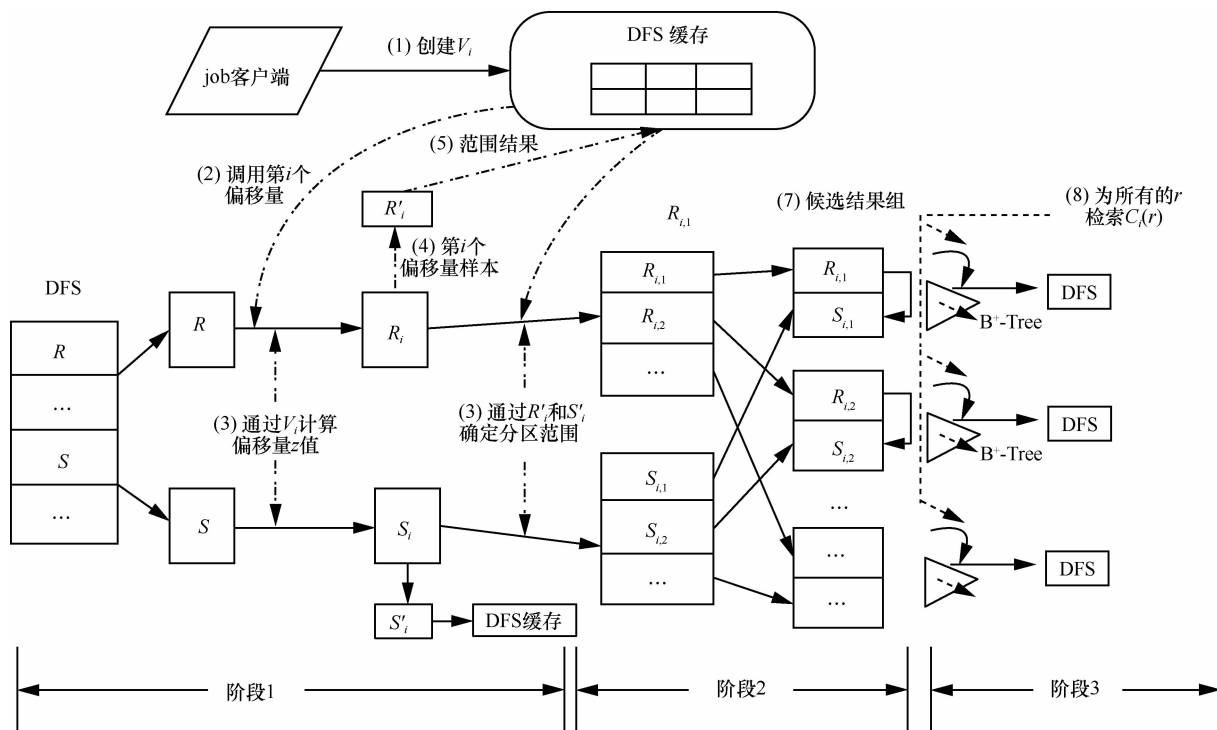


图 4 Z-order 阶段近似处理过程

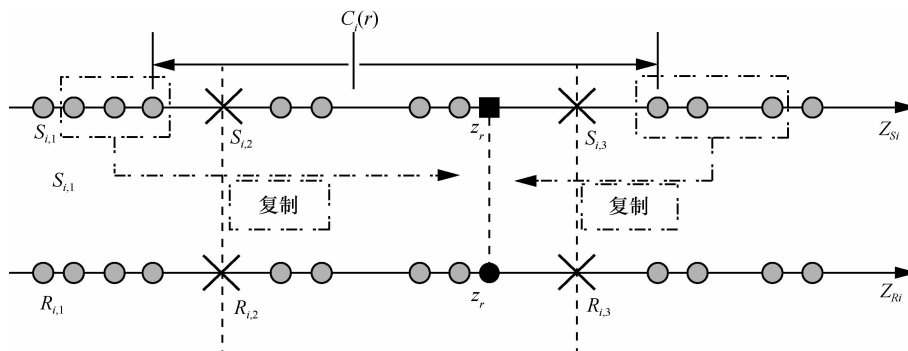


图 5 复制 3 个相邻分组数据对象到 S_i 实例



第二个 MapReduce 阶段的目标是把 R_i 和 S_i 分割成近似的块,并为任意的 $r \in R$ 找到 $kNN(r, S)$ 的候选集合,如图4中的过程(7)和(8)。首先根据第一个 MapReduce 阶段生成的分区信息数据,将 R 在 S 内分成高概率等大小的块。然后,再找出 R_i 和 S_i 的边界,并使得 R_i 在 S_i 中都能找到 k 近邻,此时 S_i 的区域将相应的 R_i 分组包含在内,就可以在一个 S_i 块内为所有的 R_i 找到 kNN 。

最后一个 MapReduce 阶段为来自于第二个阶段的 reducer 所生成的 $kNN(r, C_i(r))$ 的任意 $r \in R$ 决定 k 近邻,这在 MapReduce 中可以轻松地实现。

本节在 MapReduce 分布式平台上,使用 Z-order 编码的方式,实现了对多维对象桶向量的近似处理。本节和行条化的 LSH 对高维数据的降维阶段联合起来,为高维大数据的 AkNN 查询问题提供了一个可行的解决方法。

5 实验仿真

5.1 实验设置

实验在 16 台 PC 机组成的集群上进行,其中 1 台作为主节点,其余 15 台作为从节点。每台 PC 配置相同:CPU 为 Intel Core i3 3.4 GHz, 8 GB 内存,操作系统为 CentOS 6.5, 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^[19],每幅图像被表示为 512 维的 GIST 特征。另一个是 Tiny 图像数据集^[20],其中有 8 000 万张图片,每个图像被表示为 384 维的 GIST 特征。这些数据集已经被用于评价其他 LSH 算法的性能^[21]。

5.2 实验结果

5.2.1 精确指标

本节研究 ZbH 算法的精确指标(近似质量)。LSH 是一种随机的算法,它并不能保证返回精确的结果,但会以一定的高概率使得相近的对象映射到相近的桶中,因此需要测定算法的精确性,采用和参考文献[20]相同的测量方法来评估。特别地,对于一个数据对象 q ,通过使用的算法所返回的 kNN 查询结果为 $o_1, \dots, o_k$,并根据它们到数据对象 q 的距离的不减顺序排序,使用 $o_1^*, \dots, o_k^*$ 代表线性扫描所得到的到数据对象 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 $R_i(q) = \frac{\|o_i, q\|}{\|o_i^*, q\|}$,其中 $i=1, \dots, k$, 并且 $\|\cdot\|$ 代表欧氏距离,平均精确指

标定义为 $\frac{1}{k} \sum_{i=1}^k R_i(q)$, 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。给定一个数据集 Q 时,其中所有数据对象的平均精确指标作为 AkNN 精确性的最终测度,这个平均值被称为总体平均精确指标。

在高维大数据上测定 AkNN 结果的精确开销很大,因此,随机选择大数据集的一部分作为实验数据。对于每一个数据对象,使用本文的方法计算它们到精确方法计算的第 k 个真实近邻的距离,把平均精确指标作为衡量近似质量的标准。使用 15 个从节点的集群大小作为所有近似质量测定相关实验的平台,为了测定数据集大小对近似质量的影响,固定参数 $k=10$, 散列函数个数 $m=16$, 桶宽 $w=400$, 偏移副本数 $\alpha=2$, 使用 Tiny 数据集并以数据量从 50 000 到 300 000 逐渐增大的方式做对比实验,算法 LSHZ 和 ZbH 在 GIST 数据集上的近似指标如图 6 所示。图 6(a)展示的测定结果表明, LSHZ 算法和 ZbH 算法都能得到较好的近似率,在所有情况下保持平均近似率在 1.3 并且在最坏的情况下也没有超过 1.6。实验结果表明,对于 LSHZ 和 ZbH 算法,数据集的大小对近似质量的影响不大。

接下来,固定数据集的大小,测定参数 k, m, w 以及 α 对 LSHZ 和 ZbH 算法近似质量的影响。从图 6(b)中可以看出,在 Tiny 数据集上逐渐增大 k 值时, LSHZ 和 ZbH 算法所返回的结果对象的先后顺序都有误差,但是在一般情况下,也都能达到期望的近似率,随着 k 值的增大,可以看到 ZbH 算法的结果更趋向稳定。实际上 ZbH 算法中 z 值曲线的作用就是保证每次得到 k 近邻结果的正确性更高,所以在一般情况下,此方法可以保证 k 近似最近邻结果的可靠性。图 6(c)展示了 LSHZ 和 ZbH 算法随着散列函数个数 m 的增大,近似率有所改善,也就是说使用越多的 LSH 散列函数,结果越好。但是需要注意的是,由于 Z-order 空间填充曲线处理多维数据的时候,在数据维度超过 30 以上的情况下,运行时间会大大增加,效果还不如线性扫描,这就是所说的“维度灾难”,所以要权衡 LSH 的个数和 Z-order 曲线的处理效果。图 6(d)展示了不同的 w 对于 ZbH 算法的近似质量有一定的影响,对于 LSHZ 算法几乎没有多少影响,由图 6(d)可知,对于 Tiny 数据集,将桶宽设置为 400, 能使得 ZbH 算法的效果近似质量更好。如图 6(e)所示,较大的 α 可以保持较好的数据局部性,使得平均精确指标较低,实验效果更好,但是这也是以更多的时间开销为代价

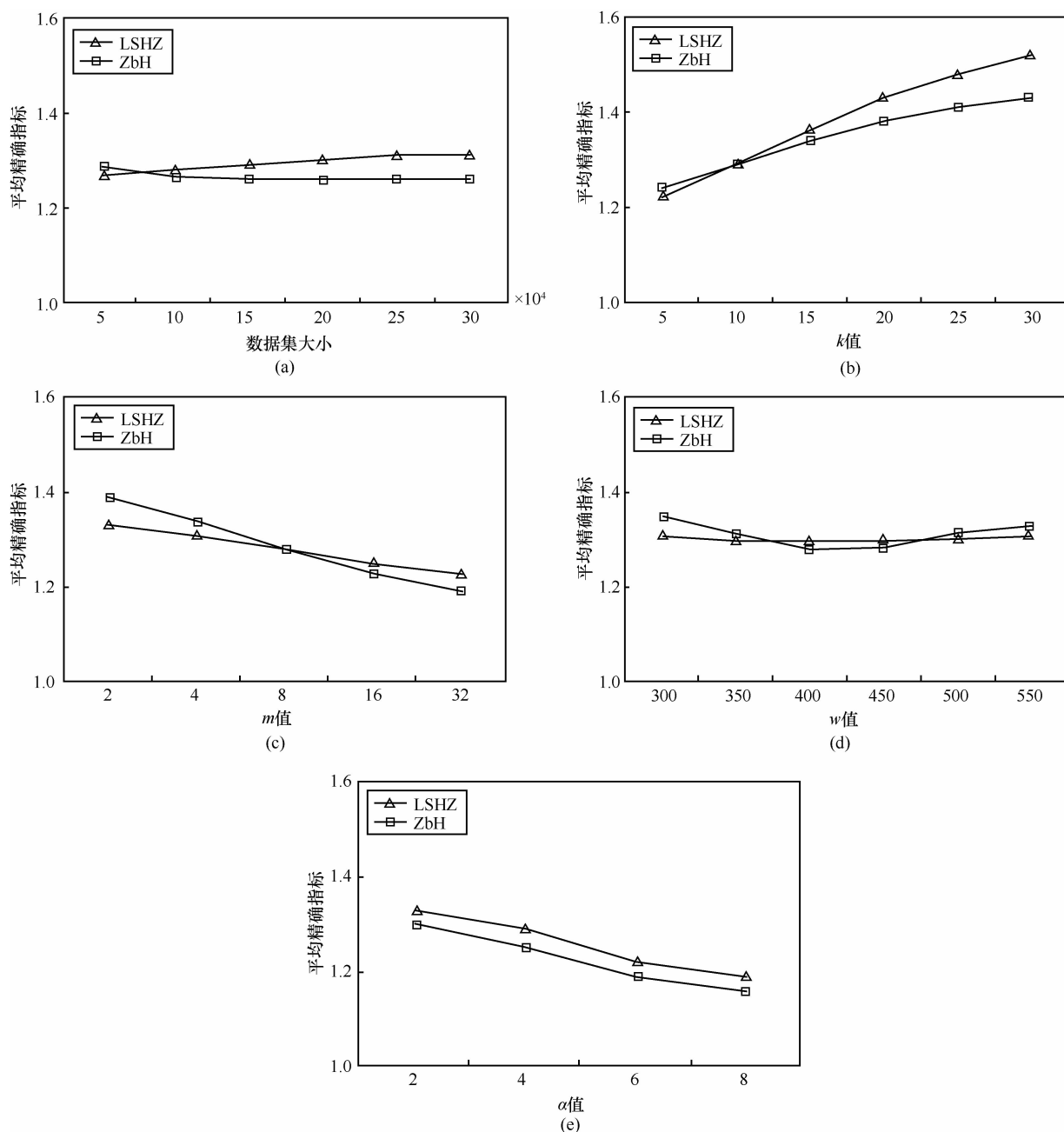


图6 算法 LSHZ 和 ZbH 在 GIST 数据集上的近似指标

的,不同的 α 对于 LSHZ 和 ZbH 算法的近似质量有一定的影响,但是对于同样的 α 值,也就是说,构造同样的偏移量副本的话,ZbH 算法的近似率效果更好。

5.2.2 运行时间

不同实验参数的选取,也会使得 LSHZ 和 ZbH 算法的运行时间不同,本节对实验的运行时间进行测定。为了展示不同参数对实验不同阶段的影响,更进一步地,分别关注降维阶段和使用 Z-order 空间填充曲线处理阶段,LSHZ

算法的两个阶段是 LSHZ 阶段 1 和 LSHZ 阶段 2, ZbH 算法的两个阶段是 ZbH 阶段 1 和 ZbH 阶段 2。

同样使用 15 个从节点的集群大小,作为运行时间测定相关实验的平台,为了测定数据集大小对运行时间的影响,固定参数 $k=10, m=16, w=400, \alpha=2$,使用 Tiny 数据集并以数据量从 50 000 到 300 000 逐渐增大的方式做对比实验,不同参数对运行时间的影响如图 7 所示。图 7(a)展示的测定结果表明,LSHZ 和 ZbH 算法运行时间的差别较大,

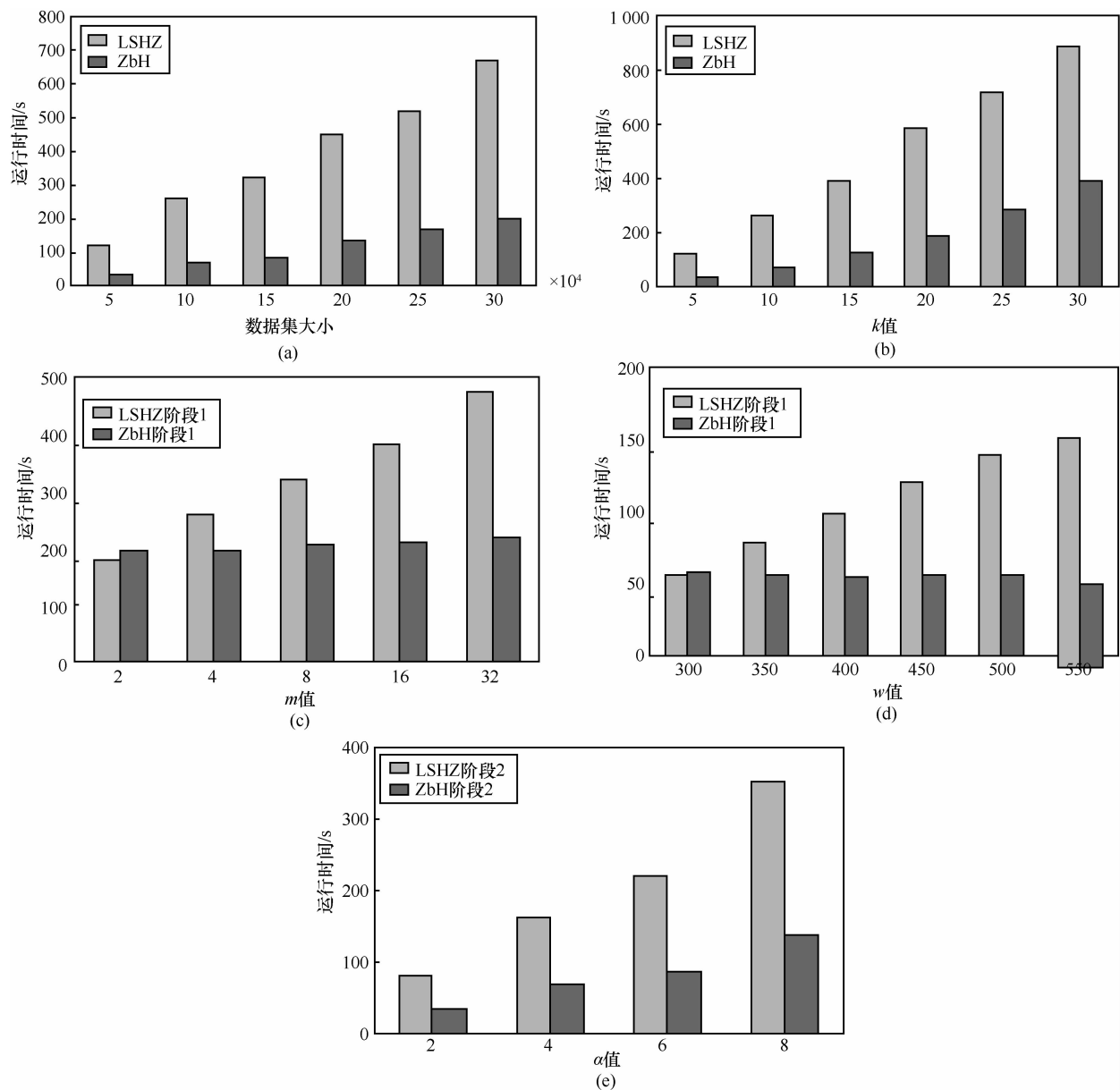


图7 不同参数对运行时间的影响

总体来说,ZbH 算法比 LSHZ 算法在运行时间上明显更快,这个数值差异主要是在数据处理的阶段1产生的,主要原因是 ZbH 算法每次计算的数据维度较少,需要计算的数据量小,而 LSHZ 算法是对原始数据对象的所有维度进行散列计算,所以 ZbH 算法运行时间效率更高。

接下来,固定数据集的大小,测定参数 k 、 m 、 w 以及 α 对 LSHZ 和 ZbH 算法运行时间的影响,当测定某一参数时,其他参数保持不变。图 7(b)展示了不同的 k 值对于 LSHZ 和 ZbH 算法的运行时间效果,当 k 值增大时,两种算法的运行时间都会相应地增加,但是 LSHZ 算法需要的

时间更长,原因可能是进行全维度计算时,散列到同一个桶中的数据对象较多,而 ZbH 算法采用行条化处理之后,减少了非相似对象映射到同一个桶中的概率。图 7(c)展示了不同的 m 值对于 LSHZ 和 ZbH 算法的运行时间影响,当 m 值增大时,LSHZ 算法运行时间也随之快速增加,而 ZbH 算法的运行时间几乎不受影响,很显然,这是因为 ZbH 算法随着 m 的增大,处理的数据量相对于 LSHZ 算法大大减少。图 7(d)展示了不同的桶宽 w 值对 LSHZ 和 ZbH 算法的运行时间效果,随着 w 值的增大,LSHZ 算法运行时间明显增加,而 ZbH 算法运行时间没有受太大的影响,这和 m

值变化对算法的运行时间影响趋势类似。图 7(e)展示了不同随机偏移量 α 对算法结果性能的影响。对于 LSHZ 和 ZbH 算法的运行时间效果,当 α 值增大时,运行时间随之增加,但是这样可以得到较高的近似质量,因此实验中需要全局考虑 α 值的选取。另外,可以推出 ZbH 算法的运行时间相对增加较慢,这说明 ZbH 算法对假阳性和假阳性的过滤效果更好。

5.2.3 可扩展性

为了测试 ZbH 算法的可扩展性,在不同节点个数的分布式集群上,采用 6 组不同数量的数据集进行测试,使用 50 000、100 000、150 000、200 000、250 000、300 000 的 Tiny 和 LableMe 图像数据集进行对比实验,实验结果如图 8 所示。实验集群的节点个数采用递增的形式(4 个、8 个、16 个);对于不同数据集的 LSH 参数设置,根据之前的实验结果,选定使用参数 m ,也就是散列函数的个数为 16,固定 k 大小为 10,桶宽 w 为 400 的情况。随着数据量的增长,观察运行时间效果,以此来测定算法可扩展性以及结果的稳定性。从图 8 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,所以该算法对于分布式平台是适合的,具有较好的稳定

性。因此,可以将该算法应用到更大规模的数据集中。

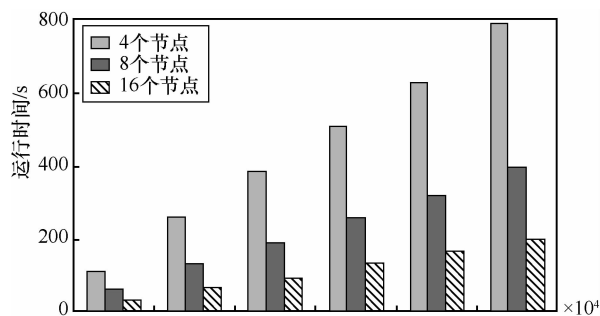
6 结束语

本文为解决大规模高维数据的 k NN 问题提出了一个有效的解决方法,首先结合使用行条化结合 LSH 算法的思想对原始的高维数据在分布式平台下进行降维处理,然后在降维后的低维数据集上使用 Z-order 曲线的优良特性实现 k NN 查询过程,从而从整体上实现了高维大规模数据的 k NN 查询问题。实验结果表明,本文提出的算法在分布式系统上具有有效性,不仅可以对大规模高维图像数据进行处理,还可以用来加速一些数据挖掘算法,也可以把它扩展到其他感兴趣的应用上,比如网页近似检测、文档分类或者视频推荐等。

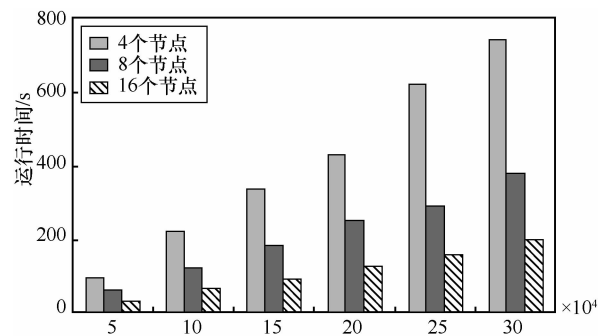
研究工作是在流行的分布式并行框架 MapReduce 上进行的,还有其他的分布式并行计算框架,比如 Spark^[22](这是最近所提出的在商用机器集群上基于内存的计算模型)。将所提出的算法扩展到 Spark 并行框架上也是一个很有挑战性的工作。MapReduce 框架很适合对成批的,离线的一次写入多次读取的数据集进行处理。然而,对于迭代工作的支持 MapReduce 框架还是有它的局限性。使用 MapReduce 框架实现迭代算法,需要启动多个 MapReduce job,由于在 map 和 reduce 阶段结束时,数据都需要重新写回硬盘,下一阶段开始时再次从硬盘读取数据,这就导致较多的开销。Spark 为集群内存计算和应用提供基础,可使得查询和处理在内存中被加载,并且可以使用一个被称为 RDD(resilient distributed dataset,弹性分布式数据集)的特殊结构重复处理,这些优势使得 Spark 比基于硬盘的 Hadoop 平台更快。因此,在 Spark 平台上设计一个迭代的,可并行的方法来更高效地解决 k NN 问题,是以后的研究工作。

参考文献

- 1 Böhm C, Krebs F. The k -nearest neighbour join: Turbo charging the KDD process. *Knowledge and Information Systems*, 2004, 6(6): 728~749
- 2 Xia C, Lu H, Ooi B C, *et al.* Gorder: an efficient method for k NN join processing. *Proceedings of the 30th International Conference on Very Large Data Bases*, Toronto, Canada, 2004: 756~767
- 3 Yu C, Cui B, Wang S, *et al.* Efficient index-based k NN join processing for high-dimensional data. *Information and Software Technology*, 2007, 49(4): 332~344



(a) Tiny 数据集大小



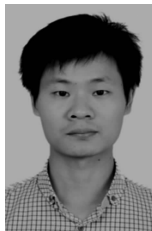
(b) LableMe 数据集大小

图 8 ZbH 算法在多台集群上的可扩展性



- 4 Chen Y, Patel J M. Efficient evaluation of all-nearest-neighbor queries. Proceedings of the 23rd International Conference on Data Engineering, Istanbul, Turkey, 2007: 1056~1065
- 5 Emrich T, Graf F, Kriegel H P, *et al.* Optimizing all-nearest-neighbor queries with trigonometric pruning. Lecture Notes in Computer Science, 2010(6187): 501~518
- 6 Zhang J, Mamoulis N, Papadias D, *et al.* All-nearest-neighbors queries in spatial databases. Proceedings of the 16th International Conference on Scientific and Statistical Database Management, Santorini Island, Greece, 2004: 297~306
- 7 Kourioukidis N, Evangelidis G. The effects of dimensionality curse in high dimensional k NN search. Proceedings of 15th Panhellenic Conference on Informatics (PCI), Gastonia, USA, 2011: 41~45
- 8 Weber R, Schek H J, Blott S. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. Proceedings of the 24rd International Conference on Very Large Data Bases, New York, USA, 1998: 194~205
- 9 Arya S, Mount D M, Netanyahu N S, *et al.* An optimal algorithm for approximate nearest neighbor searching fixed dimensions. Journal of the ACM (JACM), 1998, 45(6): 891~923
- 10 Indyk P, Motwani R. Approximate nearest neighbors: towards removing the curse of dimensionality. Proceedings of the 30th Annual ACM Symposium on Theory of Computing, Dallas, Texas, USA, 1998: 604~613
- 11 Hadoop. <http://hadoop.apache.org/>, 2015
- 12 Afrati F N, Ullman J D. Optimizing joins in a MapReduce environment. Proceedings of the 13th International Conference on Extending Database Technology, Palais de Beaulieu, Lausanne, Switzerland, 2010: 99~110
- 13 Jiang D, Tung A, Chen G. Map-join-reduce: toward scalable and efficient data analysis on large clusters. IEEE Transactions on Knowledge and Data Engineering, 2011, 23(9): 1299~1311
- 14 Vernica R, Carey M J, Li C. Efficient parallel set-similarity joins using MapReduce. Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data, Indianapolis, Indiana, USA, 2010: 495~506
- 15 Zhang C, Li F, Jests J. Efficient parallel k NN joins for large data in MapReduce. Proceedings of the 15th International Conference on Extending Database Technology, Berlin, Germany, 2012: 38~49
- 16 Rajaraman A, Ullman J D. Mining of Massive Datasets. Cambridge: Cambridge University Press, 2011
- 17 Datar M, Immorlica N, Indyk P, *et al.* Locality-sensitive hashing scheme based on p-stable distributions. Proceedings of the 20th Annual Symposium on Computational Geometry, New York, USA, 2004: 253~262
- 18 Tao Y, Yi K, Sheng C, *et al.* Quality and efficiency in high dimensional nearest neighbor search. Proceedings of the 35th SIGMOD International Conference on Management of Data, Providence, Rhode Island, USA, 2009: 563~576
- 19 Labelme. <http://labelme.csail.mit.edu>, 2015
- 20 Fergus R, Torralba A, Freeman W T. Tiny Images Dataset. <http://horatio.cs.nyu.edu/mit/tiny/data/index.html>, 2015
- 21 Pan J, Manocha D. Bi-level locality sensitive hashing for k -nearest neighbor computation. Proceedings of the 28th International Conference on Data Engineering (ICDE), Washington DC, USA, 2012: 378~389
- 22 Spark. <http://spark.apache.org/>, 2015

[作者简介]



王忠伟,男,宁波大学硕士生,主要研究方向为数据挖掘。



陈叶芳,女,宁波大学讲师,主要研究方向为数据处理和挖掘。



肖四友[通信作者],男,博士,宁波大学副教授,主要研究方向为数据处理和挖掘。



钱江波,男,博士,宁波大学教授,主要研究方向为数据处理和挖掘、逻辑电路设计。