海量高维向量的并行 Top-*k* 连接查询

马友忠[1),2)], 慈祥[1)], 孟小峰[1)]

[1)](中国人民大学信息学院，北京 100872)

[2)](洛阳师范学院信息技术学院，洛阳河南 471022)

摘要 在很多应用领域中，向量的 Top-*k* 连接查询是一种很重要的操作，给定两个向量集合 R 和 S，Top-*k* 连接查询要求从 R 和 S 中返回距离最小的前 *k* 个向量对。由于数据的海量性和高维特性，传统的集中式算法已经无法在可接受的时间内完成连接查询任务。MapReduce 作为一个并行处理框架，能够有效地处理大规模数据。由于其高可扩展性、高可用性等特点，MapReduce 已经成为海量数据处理的首选实现方案，在很多领域都得到了广泛的应用。本文基于分段累积近似法对高维向量进行降维，然后利用符号累积近似法对高维向量进行分组；在此基础上，结合 MapReduce 框架，提出了基于 SAX 的并行 Top-*k* 连接查询算法，实验表明，本文所提方案具有良好的性能和扩展性。

关键词 高维向量；MapReduce 框架；Top-*k* 连接查询

中图法分类号 TP311.1

Parallel Top-*k* Join on Massive High-Dimensional Vectors

MA You-Zhong, CI Xiang, MENG Xiao-Feng

[1)](School of Information, Renmin University of China, Beijing 100872)

[2)](School of Information and Technology, Luoyang Normal University, LuoyangHenan 471022)

Abstract Top-*k* joins on high-dimensional vectors are very important operations in many applications. Given two vector sets R and S, a Top-*k* join returns *k* closest pairs of vectors. The traditional centralized algorithms cannot deal with the large scale high-dimensional vectors in an efficient way.MapReduce, as a parallel processing framework, can deal with large scale data set. It has been widely used in many applications because of its high availability and high scalability. In this paper we adopt Piecewise Aggregate Approximation technique (PAA) to reduce the dimensionality of the vectors; then group the vectors using Symbolic Aggregate Approximation technique (SAX); based on the MapReduce framework, we propose SAX-based parallel Top-*k* join query algorithms. The experiments results show that the proposed approaches have better performance and scalability.

Key words High-Dimensional vector; MapReduce framework; Top-*k* join query

1 引言

在很多应用中，为了便于数据分析，处理对象经过特征提取，都可以表示成向量形式，如时间序列、图形、视频、轨迹数据等。为了能够比较准确地表示原始对象，向量维度往往比较高，有数百维，甚至上万个维度。高维向量的相似性连接查询是一种很重要的操作，在很多任务中都有广泛应用，如

本文得到国家自然科学基金项目(课题号：61379050, 91224008), 国家863计划项目(课题号： 2013AA013204), 高等学校博士学科点专项科研基金资助课题(课题号：20130004130001)，中国人民大学科学研究基金(课题号: 11XNL010)支持.是计算机学会会员（E200018222M）. 马友忠，男，1981年生，博士，E-mail: ma_youzhong@163.com，主要研究领域为Web数据管理、云数据管理. 慈祥，男，1986年生，博士研究生，E-mail: cixiang31415926@126.com，主要研究领域为云计算、大数据管理、在线聚集. 孟小峰，男，1964 年生，教授，博士生导师，主要研究方向包括网络数据管理、云数据管理、移动数据管理、XML 数据管理、闪存数据库系统以及隐私保护，Email: xfmeng@ruc.edu.cn. 通讯作者：孟小峰, Email: xfmeng@ruc.edu.cn.

聚类、副本检测、去重等。高维向量的相似性连接查询有两大挑战：一是数据的规模在不断扩大，传统的集中式处理方式已不再适用；二是向量的维度比较高，可以达到成千上万维，由于维度灾难的存在，基于索引的算法已无法奏效。

相似性连接查询需要用户事先提供一个相似度阈值，然而在实际应用当中，用户往往很难知道准确的阈值。Top-*k* 连接查询无需用户提供相似度阈值，直接返回最相似的（距离最小的）前 *k* 个对。Top-*k* 连接查询在空间数据库中已经进行了深入的研究[1-2]，现有的算法大都是基于某种高效的空间索引（如 R 树），只适合于小规模的、低维数据，无法处理大规模的、高维数据。

MapReduce[3]是谷歌公司提出的一个并行处理框架，由于其高可扩展性、容错性和高可用性，得到了学术界和企业界的广泛关注，已经成为海量数据处理的首选方案。本文的主要工作就是基于 MapRedcue 框架，设计并实现了海量高维向量的并行 Top-*k* 连接查询算法，实验结果表明，本文所提方案具有较好的性能和扩展性。本文主要贡献如下：

（1）提出了一种基于符号累积近似法（Symbolic Aggregate Approximation technique）的高维向量阈值连接方法，该方法具有较好的过滤效果，

（2）提出了一种基于采样的 Top-*k* 阈值估计方法，并基于估计的阈值，提出了基于 SAX 的 Top-*k* 连接查询算法；

（3）针对真实数据集和模拟数据集做了丰富的实验，对基础方案和本文提出的方案进行了对比，实验结果表明，本文所提方案具有较好的性能和扩展性。

本文组织结构如下：第二节对相似性连接查询相关工作进行了综述；第三节给出了 Top-*k* 连接查询的定义，并介绍了相关基础知识；第四节给出了基于 SAX 的向量阈值连接查询算法；第五节介绍了基于阈值的 Top-*k* 连接查询算法；第六节介绍了基于 SAX 的 Top-*k* 连接查询算法；第七节进行了实验分析；最后对本文进行了总结。

2 相关工作

相似性连接查询是一种很重要的操作，目前已经有大量的研究工作。庞俊[4]等人对相似性连接查询技术进行了综述，按照不同的分类标准，相似性连接可以分为不同的类别。按照数据类型来分，可以分为字符串相似性连接、集合相似性连接、向量相似性连接等；按照返回结果的不同，可以分为所有对相似性连接、阈值相似性连接、Top-*k* 相似性连接和 KNN 相似性连接等。

Epsilon Grid Order[5]、KDB-tree、Quickjoin[6]等主要研究了在欧氏空间中，给定距离阈值的情况下，如何快速地计算出满足距离要求的所有相似对。然而，在很多应用中，无法事先知道距离阈值，此时，如何快速找到距离最近的前 *k* 个对就具有重要的意义。[7]充分利用 R-tree 索引结构的过滤效果，从而快速找到最近的前 *k* 个对。为了降低时间复杂度，[8，9]在空间填充曲线技术的基础上，针对高维向量集合，提出了相应的近似算法。上述算法大都是单机环境下的算法，针对低维、小规模数据集，其性能还可以接受。但是其时间复杂度随着维度的增加往往呈现指数级增长，无法在可接受的时间范围内处理大规模、高维数据集合的相似性连接查询问题。

大规模数据的连接操作是一个计算代价很高的操作，目前有一些研究工作尝试利用 MapReduce 框架来解决该问题，主要包括集合相似性连接[10-14]、向量相似性连接[15-16]、Top-*k* 连接[17-18]等。

从采用的主要技术来分，海量集合相似性连接可以分为三类，前缀过滤[10-11]、Word-Count-Like[12-13]和混合方案[14]。Vernica 等人[10]提出了一种基于前缀过滤技术的海量集合数据相似性连接方案，该方案主要由三个阶段完成，每个阶段由一个 MapRedue job 负责实现。其核心思想是：如果两个集合的前缀没有公共元素，那么这两个集合肯定不可能相似。基于该规则，就可以大大减少候选集合的规模。为了进一步减少候选集合的数量，荣垂田[11]等人提出了一种基于多前缀的集合相似性连接方案。通过多个前缀进行过滤，可以减少为正例的个数，从而进一步减少候选对数量。但是基于前缀过滤技术的方案也存在一些固有的不足之处。一是网络传输代价较大：每个集合都要复制多次，复制的次数等于前缀的长度，因此对于较长的集合来讲，数据复制率太高，会导致网络传输代价比较大；二是重复比较次数比较多：对于任意两个集合，如果它们的前缀中有 *k* 个公共元素，则会重复比较 *k* 次。为了解决前缀过滤技术存在的问题，[12-13] 提出了 Word-Count-Like 的解决方案，该方案充分利用了 MapReduce 框架的基本特性，基本思路类似于 Word-Count 程序的思路。该方案的优点是避免了重复比较，任何一个集合对只需要比较一次。但是存

在一些局限性：任何两个集合只要包含一个公共元素，都需要比较一次，因此候选集合的数量特别大。通过对上述两种方案的分析，[14]提出了一种混合解决方案，将前缀过滤技术和 Word-Count-Like 方案进行了结合，从而进一步提高了集合相似性连接的效率。

基于集合的相似性连接技术不能直接应用于欧氏空间下的相似性连接问题。MR-DSJ[15]等人提出了一种基于网格划分的大规模向量相似性连接方法。该方法具有很好的并行特性，很容易在 MapReduce 框架下实现。其不足之处是该方法只适合于维度较低的情况，一旦维度比较高时，其性能急剧下降。为了解决更高维度向量的相似性连接问题，Sergei Fries 等人[16]在 MR-DSJ 的基础上提出了一种新的基于 MapReduce 的连接方案 PHiDJ，PHiDJ 方案主要通过对维度分组和变长的网格划分来提高连接速度。卢卫[17]等人提出了一种基于泰森多边形的 KNN 连接解决方案。其基本思想是：给定两个向量集合 R、S，首先基于泰森多边形将集合 R 划分成 k 个子集，R_1，R_2，…，R_k，然后按照一定的规则，将集合 S 也划分成 k 个子集，S_1，S_2，…，S_k。但是对 S 的划分要满足一个要求：$\forall r \in R_1, KNN(r,S) \subseteq S_1$。在这种情况下，集合 R 中的每一个子集只需要与对应的 S 的子集进行比较就可以了，从而节省大量的网络传输代价和计算代价。张弛[18]等人基于线性化技术，在 MapReduce 框架下提出了一种近似的 KNN 连接查询方案。

刘义[19]等人针对海量空间数据提出了一种新颖的、基于 MapReduce 框架的并行 Top-k 连接查询方案，该工作分别在空间连接阶段和 Top-k 结果获取阶段进行了优化。雷斌[20]等人提出了一种大规模概率数据上基于 EMD 距离的并行 Top-k 相似性连接算法。文献[21]在 MapReduce 框架下，针对直方图概率数据提出了一种基于 EMD 距离的阈值相似性连接方案。文献[22]是最早尝试在 MapReduce 框架下解决海量向量 Top-k 连接查询问题的研究工作，首先提出了两种串行化算法：divide-and-conquer 和 branch-and-bound；然后又提出了两种基于 MapReduce 框架的并行化算法，分别是所有对划分算法（TopK-P-MR: All pair Partitioning Algorithm）和关键对划分算法（TopK-F-MR: Essential Pair PartitioningAlgorithm）。但是，文献[22]中的算法一般比较适合于中低维度的向量，无法高效处理超高维度向量的 Top-k 连接查询问题。

3 问题定义与基础知识

3.1 问题定义

定义 1. 向量阈值连接查询. 给定两个向量集合 R 和 S，距离函数 *dist*，距离阈值 ε，集合 R 和 S 的连接查询$R \bowtie S$的结果就是返回所有距离小于 ε 的向量对，即：$R \bowtie S = \{(r,s) | r \in R, s \in S,\ dist(r,s) \leq \varepsilon\}$。其中 $R = \{r_1, r_2, \ldots, r_{|R|}\}$，$S = \{s_1, s_2, \ldots, s_{|S|}\}$，$r_i$ (s_j)是 d 维向量：$r_i = (r_{i1}, r_{i2}, \ldots, r_{id})$，距离函数 *dist* 采用欧几里得距离。

定义 2. Top-k 向量连接查询. 给定两个向量集合 R 和 S，距离函数 *dist*，集合 R 和 S 的 Top-k 连接查询的结果就是返回距离最小的前 k 个向量对。即：$Top\_k(R \bowtie S) = \{(r_{i1}, s_{j1}), \ldots, (r_{ik}, s_{jk})\}$，并满足如下要求：

- $r_{ik} \in R, s_{jk} \in S$
- $dist(r_{i1}, s_{j1}) \leq dist(r_{i2}, s_{j2}) \leq \cdots dist(r_{ik}, s_{jk})$
- 对于任意一对 (r_{im}, s_{jm})，如果 $(r_{in}, s_{jn}) \in Top\_k(R \bowtie S)$，则 $dist(r_{ik}, s_{jk}) \leq dist(r_{in}, s_{jn})$

3.2 分段累积近似法

分段累积近似法（Piecewise Aggregate Approximation：PAA）是由 Keogh 等人[23]最先提的一种有效的时间序列降维方法，后来被广泛应用于时间序列相似性查询、模式发现、相似轨迹查询等多个领域。PAA 的核心思想是：将原有的时间序列从高维空间（n 维）降到低维空间（m 维），$m \ll n$，通过在低维空间的距离计算来实现高维空间数据的过滤，从而提高计算效率。具体如下：

给定一个长度为 n 的时间序列

$T=(t_1,t_2,\ldots,t_n)$，把 T 划分成 m 个等宽的块，得到长度为 m 的序列 $\bar{T}=(\bar{t}_1,\bar{t}_2,\ldots,\bar{t}_m)$，每一个元素 $\bar{t}_i$ 的值等于原始序列中第 i 块内所有元素的平均值，即：

$$\bar{t}_i=\frac{m}{n}\sum_{j=\frac{n}{m}(i-1)+1}^{\frac{n}{m}i}t_j$$

给定两个时间序列 T 和 S，它们的欧几里得距离是：

$$D(T,S)=\sqrt{\sum_{i=1}^{n}(t_i-s_i)^2}$$

降维之后的 $\bar{T}$ 和 $\bar{S}$ 距离是：

$$D(\bar{T},\bar{S})=\sqrt{\frac{n}{m}}\sqrt{\sum_{i=1}^{m}(\bar{t}_i-\bar{s}_i)^2}$$

其中 $D(\bar{T},\bar{S})$ 是 $D(T,S)$ 的下界，即 $D(\bar{T},\bar{S})\le D(T,S)$，在[19]中已经得到证明。

3.3 符号累积近似法

在 PAA 的基础上，文献[24]中提出了一种符号化方法：符号累积近似法(Symbolic Aggregate Approximation：SAX)。通过将 PAA 中的每一个值与某个符号集合中一个符号相对应，最终可以将 PAA 用一个字符串来表示。文献[24]中证明 SAX 的距离也是原始时间序列欧氏距离的下界，即：$D(\hat{T},\hat{S})\le D(T,S)$。

$$D(\hat{T},\hat{S})=\sqrt{\frac{n}{m}}\sqrt{\sum_{i=1}^{m}d(\hat{t}_i-\hat{s}_i)^2}$$

其中 $d(\hat{t}_i-\hat{s}_i)$ 是两个符号之间的距离，可以通过一个查找表直接获得，查找表一般比较小，可以直接放在内存中。关于 SAX 的详细信息可以参考文献[24]。

图 1 展示了一个原始向量到 PAA 和 SAX 的转换示意图：V 是一个长度为 20 维的向量，每 4 维分成一段，可以转换成一个 5 维的向量 PAA（图 1-a）；然后通过符号化的方法，可以将 PAA 再转换成一个长度为 5 的字符串：$SAX(V,4,4)=a_1a_2a_3a_4a_2$（图 1-b）。

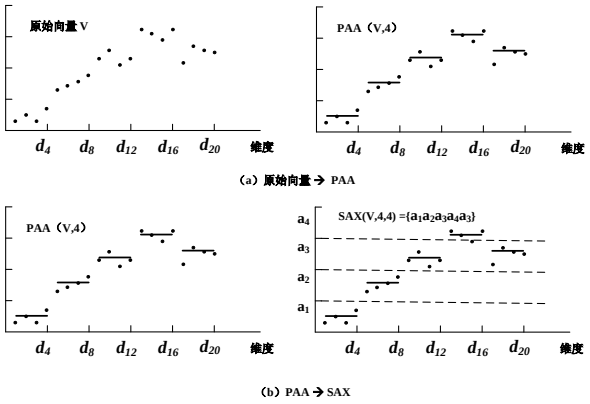


图 1. 原始向量、PAA、SAX 转换示意图

4 基于SAX的向量阈值连接查询

已有的向量阈值连接查询工作中，为了提高效率，大都采用了某种索引结构，如 KD-树、R-树等。这些方案大都是单机环境下的算法，无法有效地处理海量数据，并且这些方案只适合于低维向量，一旦维度较高时，会出现维度灾难，其性能会急剧下降，接近于穷举比较的代价。为了解决高维向量的阈值连接查询问题，基于 PAA 技术，WumanLuo[25]等人提出了一种新的维度聚集近似（Dimension Aggregate Approximation：DAA）方法。通过降维，把原有的高维向量用 DAA 来表示，在 DAA 的基础上，结合 MapReduce 框架提出了两种并行化方法：一阶段过滤-验证算法法和两阶段过滤-验证算法，从而大提高了阈值连接查询的效率。虽然通过 DAA 技术可以以较低的代价对原有高维向量进行过滤，但是该方案仍然没有减少比较的次数，每一对向量至少需要比较一次，比较的总次数仍然等于两个集合中向量数量的乘积，计算的代价仍然比较大。

实际上，有很多向量根本不需要进行比较。为了进一步减少比较的次数，本文在 DAA 的基础上，结合符号累积近似法，提出了一种基于 SAX 的阈值连接查询方法。其基本框架如图 2 所示，主要包括

四个阶段：数据划分、利用 SAX 进行分组、计算 SAX 相似对、结果验证。详细执行过程如算法 1 所示。

1)　数据划分

由于算法采取的是嵌套循环连接方法，所以在 Map 阶段，整个数据集合被近似均匀地划分成若干个块，然后对任意两个块进行比较。假设共分成 n 个块，则共有$(n*(n+1))/2$ 个块对需要进行比较，每一对由一个 reduce task 负责。同时对每一个向量 v，需要计算出向量的 PAA、SAX 值，然后把 PAA、SAX 信息和原始向量信息组合在一起，复制 n 份。PAA、SAX 的信息在 reduce task 中进行比较时会用到（line 1-4）。

2)　利用 SAX 进行分组

根据 PAA 和 SAX 的定义我们可以发现，不同的向量经过转换之后可能具有相同的 SAX 字符串，因此可以利用 SAX 对原始向量进行分组。在每一个 reduce task 中，对分配到该 task 上的所有向量，首先统计出所有不同的 SAX（line 7），然后再基于 SAX 对向量进行分组，即记录下每一个 SAX 所包含的向量集合（line 8）。

3)　计算 SAX 相似对

在每一个 reduce task 中，针对分配到其上面的所有 SAX，利用嵌套循环连接的方法，计算出任意两个 SAX 之间的距离，如果两个 SAX 之间的距离大于阈值 ε，则对应的原始向量之间的距离肯定大于 ε，该 SAX 对可以过滤掉，其所对应的所有原始向量都不需要再进行比较（line 9）。因为不同的向量可能具有相同的 SAX，所以 SAX 的数量一般会远小于原始向量的个数，因此，即使使用嵌套循环连接方法来计算相似的 SAX 对，其比较次数也将远少于原有的向量比较次数，因此可以大大提高计算效率。

4)　结果验证

计算出相似的 SAX 候选对以后，需要对其对应的原始向量进行进一步验证，计算出向量之间的实际距离值，如果其实际距离小于或等于距离阈值 ε，则可以作为结果输出，否则就过滤掉（line 10-13）。

4.1 代价分析

本小节主要对基于 SAX 的向量阈值连接查询算法（SAX-HDSJ）的代价进行分析，代价与文献[25]中的定义一样，主要包括通信代价 C_t 和计算代价 C_c。主要对 SAX-HDSJ 算法和文献[25]中的 B-DAA OSFR 算法的代价进行比较。表 6-1 显示了代价分析中用到的一些标识符。

表 1　代价分析中的标识符

标识符	描述
d	向量的原始维度
d'	降维后向量的维度
N	向量的数量
S	每一个 reduce 任务能够处理的向量数
M	总的 SAX 字符串数量
K	数据分区的数量
P_{PAA}	PAA 的过滤因子
P_{SAX}	SAX 的过滤因子
$c_t d$	传递一个向量的通信代价
C_t	总的通信代价
$c_c d$	一个向量对的计算代价
C_c	总的计算代价
$C_{B\text{-}DAA}$	B-DAA 方案的总代价
C_{SAX}	SAX-HDSJ 方案的总代价

SAX-HDSJ vs. B-DAA：因为在进行数据划分的时候，每个向量随机地分配到不同的块中，每一个块的大小基本上是一样的。块的数量一般可以根据每个 Reduce 任务的计算能力进行估算出来：$K=\frac{N}{\sqrt{S}}$。文献[25]中 B-DAA OSFR 的代价如下：

$$C_{B\text{-}DAA}=C_t+\frac{N(N+1)}{2}c_c d'+\frac{N(N+1)}{2}c_c d(1-P_{PAA})$$

其中，C_t 是总的通信代价，$\frac{N(N+1)}{2}c_c d'$ 是所有 DAA 对之间距离的总计算代价，$\frac{N(N+1)}{2}c_c d(1-P_{PAA})$ 经过 DAA 过滤之后，向量原始距离计算的总代价。

按照前面的分析方法，我们可以得到 SAX-HDSJ 算法的代价如下：

$$C_{SAX}=C_t'+\frac{M(M+1)}{2}c_c d'+\frac{N(N+1)}{2}c_c d'(1-P_{SAX})+\frac{N(N+1)}{2}c_c d(1-P_{PAA})$$

其中，C_t' 是 SAX-HSDSJ 算法的总通信代价，与 B-DAA 算法相比，除了需要传输 DAA 和原始向量信息之外，SAX-HSDSJ 算法还需要传输 SAX 字符串信息，然而由于 SAX 字符串的长度一般要比原始向量低很多，并且数量也比原始向量少很多，因此我们可以近似认为 C_t' 和 C_t 是一样的。$\frac{M(M+1)}{2}c_c d'$ 是 SAX 字符串距离的总计算代价；$\frac{N(N+1)}{2}c_c d'(1-P_{SAX})$ 是经过 SAX 过滤之后，PAA 距

离的总计算代价；$\frac{N(N+1)}{2}c_c d(1-P_{PAA})$是经过 SAX 和 PAA 过滤之后，原始向量距离的总计算代价。

SAX-HDSJ 算法的核心是通过 SAX 减少 PAA 距离的计算次数，所以 SAX 字符串的数量不能太多，否则会得不偿失。因此，我们有如下定理：

定理 1：如果$M < \sqrt{\frac{P_{SAX}}{2}} \cdot N$，那么$C_{SAX} < C_{B-DAA}$，即：SAX-HDSJ 算法的代价小于 B-DAA OSFR 算法的代价。

证明：

$$M < \sqrt{\frac{P_{SAX}}{2}} \cdot N$$

$$\Rightarrow M^2 < \frac{P_{SAX} \cdot N^2}{2}$$

$$\Rightarrow \frac{2M^2}{N^2} < P_{SAX}$$

$$\because \frac{2M^2}{N^2} = \frac{M(M+M)}{N \cdot N} > \frac{M(M+1)}{N(N+1)}$$

$$\therefore \frac{M(M+1)}{N(N+1)} < P_{SAX}$$

$$\Rightarrow \frac{M(M+1)}{2} + \frac{N(N+1)(1-P_{SAX})}{2} < \frac{N(N+1)}{2}$$

$$\Rightarrow \frac{M(M+1)}{2}c_c d' + \frac{N(N+1)}{2}c_c d'(1-P_{SAX}) < \frac{N(N+1)}{2}c_c d'$$

如前所述，假设C_t'和C_t大小相等，可以得到：

$$C_t' + \frac{M(M+1)}{2}c_c d' + \frac{N(N+1)}{2}c_c d'(1-P_{SAX}) + \frac{N(N+1)}{2}c_c d(1-P_{PAA})$$
$$< C_t + \frac{N(N+1)}{2}c_c d' + \frac{N(N+1)}{2}c_c d(1-P_{PAA})$$
$$\Rightarrow C_{SAX} < C_{B-DAA}$$

证明完毕。

算法 1. 基于 SAX 的相似性连接

输入: ε, *n*,*paaSize*//距离阈值，块编号，PAA 大小

输出:相似向量对和距离

```
1   map(k1,v1)
2   paa_v1←getPaa(v1);sax_v1←getSax(v1);
3   String newVal←paa_v1+ “,” + sax_v1 + “,” + v1.toString();
4   Replicate newVal n times;//n: the number of partitions;
5   reduce(k2,v2)
6   LinkedListsaxList1,saxList2,saxVecMap1,saxVecMap2, canSaxPairs;
7   saxList1,saxList2←getUniqueSax(v2);
8   saxVecMap1,saxVecMap2←getVectorGroup(saxList1, saxList2, v2);
9   canSaxPairs←getSaxPairs(ε , saxList1,saxList2);
10  For eachsaxPair : canSaxPairsdo
11    finalVecPairs←refineVecPairs(saxPair, saxVecMap1,saxVecMap2);
12  For eachvecPair: finalVecPairsdo
13  Output(vecPair.vectors, vecPair.dist);
```

5 基于阈值的Top-*k*连接查询

如果能事先能够知道第 k 个向量对的距离 ε，那么就可以以 ε 作为距离阈值，直接采用[25]提出的基于阈值的连接查询算法来进行计算，具体计算流程在第四节第一段已经进行了描述。然而事实上我们无法准确知道第 k 个向量对的距离。一种直观的方法就是可以预先选定一系列距离阈值，ε_1, ε_2, ε_3,...,其中：$\forall i < j,\ \varepsilon_i < \varepsilon_j$。针对每一个阈值 ε_i 都执行一次阈值连接查询算法，如果在 ε_i 情况下满足条件的结果数量小于 k，则以 ε_{i+1} 为新的距离阈值再重新执行一次阈值连接查询，直到满足距离阈值的结果对的数量大于等于 k，则结束。阈值的选择一般根据向量集合中距离的分布情况人为确定，如可以以某一个区间递增：$v_i = 0.05 * i, i = 1,2,...$。

该算法实现起来比较简单，但是因为每次都需要重新执行阈值连接查询，所以会带来大量的冗余、重复性的计算工作，效率较低。为了避免冗余计算，后面提出了基于 SAX 的 Top-k 连接查询算法。

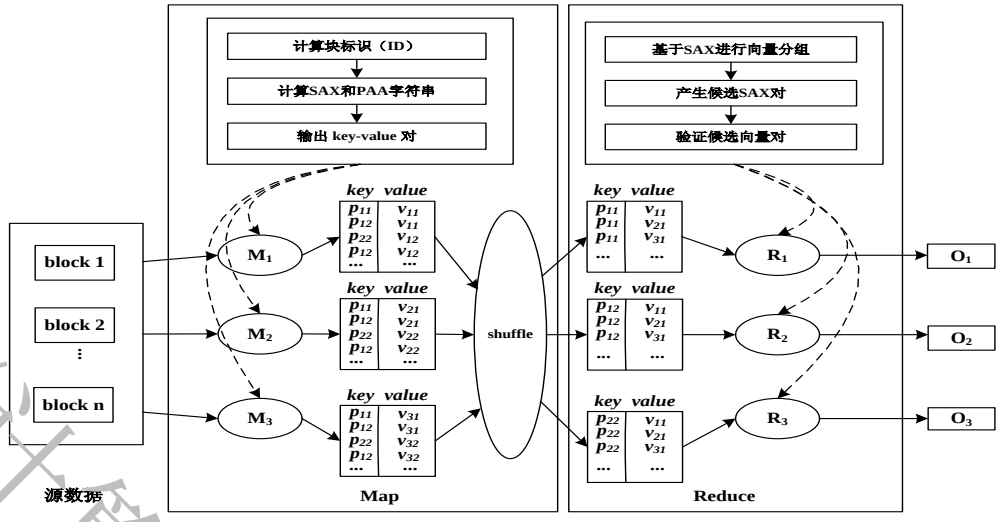


图 2. 基于 SAX 的向量阈值连接查询框架

6 基于SAX的Top-*k*连接查询

第五部分针对事先选择的不同距离阈值分别执行阈值连接查询算法，虽然比较直观，但是会出现大量的冗余计算。如果我们能够准确确定第 k 个对的距离 ε，则执行一次阈值连接查询即可得到需要的结果。由于距离阈值 ε 的大小直接影响到算法的执行时间，ε 越小，算法的执行时间越短，ε 越大，算法的执行时间越长。因此，如何估计出第 k 对的距离，并使其尽可能接近真实的第 k 个对的距离，就成了一个核心的问题。

在进行阈值估计的时候，最简单的方法是从原始数据集中随机采样一部分数据（如 10%），然后对采集到的样本进行两两比较，求出距离最小的前 k 个对，用第 k 个对的距离作为估计的距离阈值，该阈值肯定是 Top-k 结果的距离下界。此方法虽然简单，但是得到的阈值往往会比较大，导致后面的计算开销也会比较大。

为了使估计出的 ε 更加接近真实的第 k 个对的距离阈值，在采样之前，我们先对所有的向量基于 SAX 进行分组，然后针对每一个 SAX 分组中对应的向量进行采样，这样得到的采样数据之间的距离一般会比较小，因为距离越小的向量越倾向于分配到同一个其 SAX 分组中。

估计出第 k 个向量对的距离阈值 ε 以后，接下来就可以利用我们在第 4 节中提出的“基于 SAX 的向量阈值连接查询”方法来进行计算，因为只需要求出前 k 个向量对，所以需要对第 4 节中的算法进行稍微修改。

基于上述思想，我们提出了基于 SAX 的 Top-k 连接查询方法，图 3 给出了该方案的执行框架。该方案主要由四个 MapReduce Jobs 组成，其中 Job 1 和 Job 2 主要是用于阈值估计，Job 3 是执行基于阈值的 Top-k 连接查询，从而得到局部的 Top-k 结果，Job 4 主要是负责从所有局部的 Top-k 结果中计算出全局的 Top-k。

算法 1. 数据采样和距离计算（Job 1）

输入: ε,*paa_size*//距离阈值和 PAA 大小

输出: 相似向量对和距离

```
1   map(k1,v1)
2   paa_v1←getPaa(v1);
3   sax_v1←getSax(v1);
4   emit(sax_v1, v1);
5   reduce(k2,v2)
6   LinkedListsamples, vectorsPairs
7   Foreachvector: v2 do
8     j←Math.abs(random.nextInt()) % rate;//sample rate: 1/rate
9     If (j == 1)
10      samples.add(vec);
11    End if
12  vectorsPairs←loopJoin(samples, ε);
13  For eachvecPair: finalVecPairsdo
14  emit(vecPair.vectors, vecPair.dist);
```

算法 2 描述了 Job 1 的执行过程，在 map 阶段，主要负责计算出每个向量的 SAX，然后以 SAX 作为 key 值，以原始向量作为 value 传送到 reduce 端（line 1-4）；在 reduce 端，首先对每一个 SAX 分组里面的所有向量进行随机采样，得到样本集合（line 7-11），然后在样本集合中，计算出两两之间的距离（line 12）；最后把每个距离输出。（line 13-14）。

算法 3 描述了 Job 2 的执行流程，其主要目的是从 Job 1 的计算结果中找出距离最小的前 k 个对，并以此作为估计的距离阈值。

算法 4 描述了 Job 3 的执行流程：基于阈值的 Top-k 连接查询，其基本思想是基于第四部分的“基于 SAX 的向量阈值连接查询”。不同之处在于，由于只需要得到距离最小的前 k 个对，所以满足 ε 要求的并不一定都是需要的结果。因此在执行阈值连接的时候，可以维护一个长度为 k 的队列（line 8），把目前为止、距离最小的 k 个对及其距离存放其中。同时在计算的过程中，可以根据队列中结果的情况对阈值 ε 进行及时更新，如果第 k 个向量对的距离小于 ε，则把 ε 更新为第 k 个向量对的距离（line 35-37）以更新后的 ε 值进行后面的计算，从而加快执行速。同时，对于候选的 SAX 相似对，也可以利用 ε 进行过滤，如果某一个 SAX 对的距离大于 ε，则其对应的所有向量之间的距离肯定大于 ε，所以可以直接过滤掉（line 17）。随着计算的进行，ε 的值会越来越小，其过滤效果也会越来越好，从而可以减少很多不必要的计算，提高计算的效率。

算法 3．Top-k 距离阈值估计(Job 2)

输入: 相似向量对和距离

输出: ε//估计的 Top-k 距离阈值

```
1   map(k1,v1)
2   newKey← “one”;
3   newValue← v1;
4   emit(newKey. newValue);
5   reduce(k2,v2)
6   PriorityQueuepq = new PriorityQueue(k);//k: the top-k number
7   intcountTotal = 0;doubledist= 0.0;
8   For (Text val : values)
9       countTotal++;
10      dist←Double.parseDouble(val.toString());
11      if(countTotal<= k)
12        pq.add(dist);
13      elseif(dist<Double.parseDouble(pq.peek().toString()))
14        pq.poll();
15        pq.add(dist);
16      End if
17  End for
18  context.write(new Text("thre:   "), new Text(pq.peek().toString()));
```

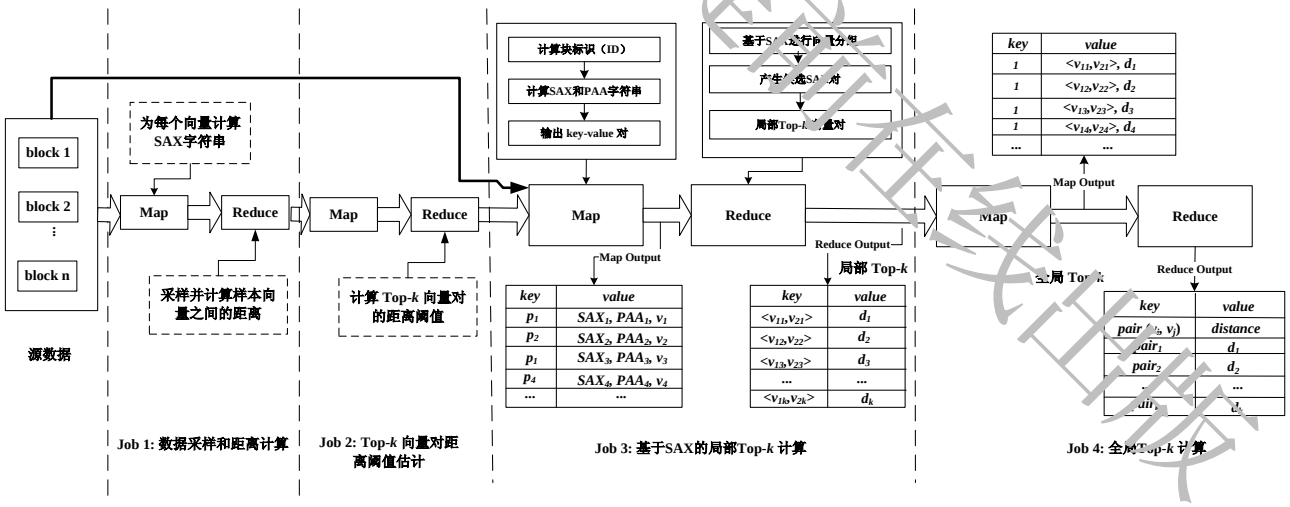


图 3 基于 SAX 的 Top-k 连接查询框架

算法 4．基于 SAX 的局部 Top-k 计算(Job 3)

输入: ε, n,$paaSize$//距离阈值，块编号，PAA 大小

输出: Top-k 向量对

```
1   map(k1,v1)
2   paa_v1←getPaa(v1);
3   sax_v1←getSax(v1);
4   String newVal←paa_v1+ “,” + sax_v1 + “,” + v1.toString();
5   Replicate newVal n times;//n: the number of partitions;
6   reduce(k2,v2)
7   LinkedListsaxList1,saxList2,saxVecMap1,saxVecMap2, canSaxPairs;
```

```
8   PriorityQueuepq← new PriorityQueue(k);//k: the top-k number
9   String[] vecList1, vecList2;
10  String sax1, sax2;
11  Double distTemp← 0.0;
12  IntcountTotal← 0;
13  saxList1,saxList2←getUniqueSax(v2);
14  saxVecMap1,saxVecMap2←getVectorGroup(saxList1, saxList2, v2);
15  canSaxPairs←getSaxPairs(ε , saxList1,saxList2);
16  For eachsaxPair: canSaxPairsdo
17   Ifsaxpair.dist>ε
18       continue;
19  End if
20  sax1←saxPair[0], sax2←saxPair[1];
21  vecList1 ←saxVecMap1.get(sax1);
22     vecList2 ←saxVecMap2.get(sax2);
23  For each v1: vecList1 do
24  For each v2: vecList2 do
25  distTemp←getDist(v1,v2)
26  If distTemp>ε
27            continue;
28  End if
29        countTotal++;
30        If(countTotal<= k)
31          pq.add(dist);
32        Elseif(dist<Double.parseDouble(pq.peek().toString()))
33          pq.poll();
34          pq.add(dist);
35          if(ε <Double.parseDouble(pq.peek().toString()))
36            ε ←thre = Double.parseDouble(pq.peek().toString());
37          End if
38        End if
39  Emit all the pairs in pq;
```

7 实验评估

本节我们对所提出的基于 SAX 的 Top-*k* 连接查询方案（SAX-Top-k）进行实验验证，并与基准方案：基于阈值的 Top-*k* 连接查询（Baseline-Top-*k*）进行性能对比，Baseline-Top-*k* 采用的基本算法是基于论文[25]提出的一种基于 MapReduce 框架的算法。同时测试了两种算法在单机环境下的性能对比（SAX-Top-*k*-Serial vs. Baseline-Top-*k*-Serial）。主要测试不同 *k* 值对执行时间的影响、不同数据集大小对执行时间的影响、不同维度对执行时间的影响以及阈值估计所消耗的时间。

7.1 实验环境

实验是在一个由 19 个节点组成的集群上进行的，其中 1 个作为 master 节点，16 个作为 slave 节点。节点配置如下：CPU：Q9650 3.00GHz, Memory：8GB, Disk：500GB，OS: 64bit Ubuntu9.10 server。其他参数如表 2 所示：

表 2：实验参数设置

参数	值
k	5000,10000,15000,20000
集群节点	19
map 任务个数/节点	4
reduce 任务个数/节点	4
数据量（万）	20，30，40，50

实验中所采用的数据集来自于[25]中所用到的数据集，可以从网站下载 1，具体参数如表 3 所示：

表 3：数据集

数据集	类型	数量	维度	大小
image-128	真实	500,000	128	423M
image-128-2w	真实	20,000	128	18M
image-960	真实	500,000	960	3.92G
Image-128-syn-1	模拟	200,000	128	175.4M
Image-128-syn-2	模拟	300,000	128	262.7M
Image-128-syn-3	模拟	400,000	128	347.9M
Image-960-syn-1	模拟	200,000	960	1.64G
Image-960-syn-2	模拟	300,000	960	2.46G
Image-960-syn-3	模拟	400,000	960	3.28G

7.2 实验结果与分析

7.2.1 不同*k*对执行时间的影响

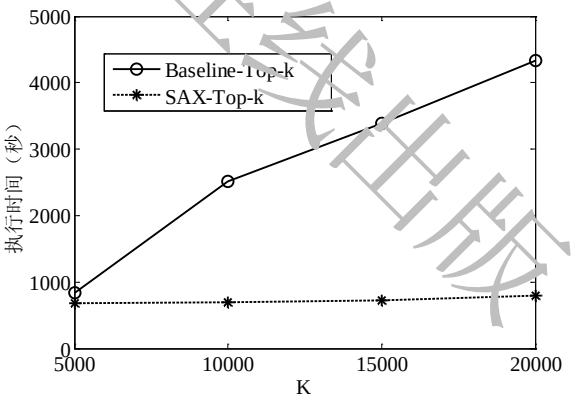


图 4. 不同 *k* 对执行时间的影响（image-128）

图 4 展示了针对 image-128 数据集，不同的 *k* 对执行时间的影响。在执行 Baseline-Top-*k* 时，阈值

⎵ http://corpus-texmex.irisa.fr

依次选择为：0.05，0.1，0.15，0.2，0.25，0.3，0.35，0.45，0.5。在 *k* 较小的情况下，SAX-Top-*k* 与 Baseline-Top-*k* 的执行时间差不多，因为在 k 比较小的情况下，Baseline-Top-*k* 只需要执行一次就可以达到要求；但是随着 *k* 的增加，Baseline-Top-*k* 往往需要执行多次才可以返回所需要的 *k* 个相似对，因此执行时间增长很快。SAX-Top-*k* 的执行时间虽然也有所增加，但是针对所有的 *k* 只需要执行一次，所以 SAX-Top-*k* 的性能比 Baseline-Top-*k* 好很多。并且 *k* 越大，SAX-Top-*k* 相对性能越好。

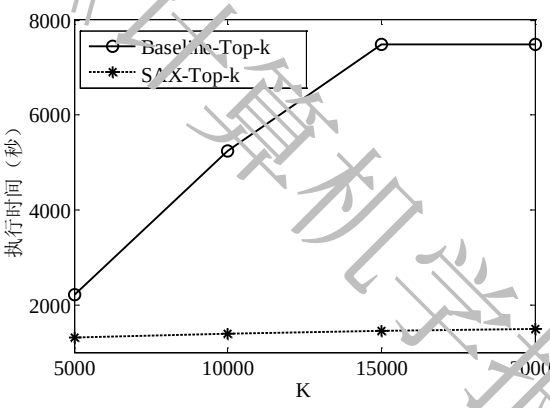


图 5. 不同 k 对执行时间的影响（image-960）

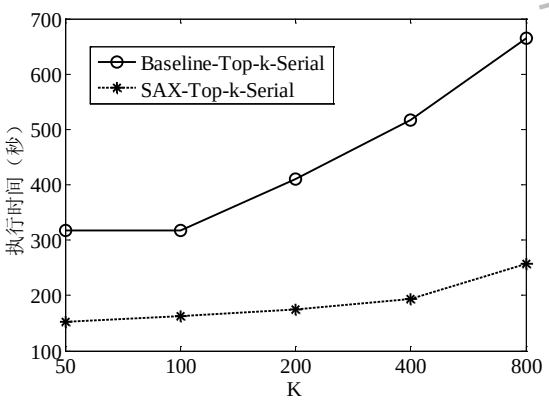


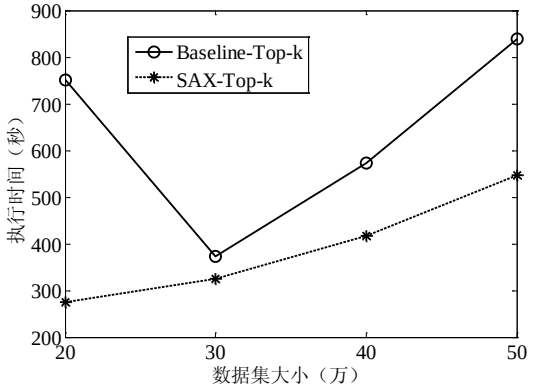
图 6. 单机环境下的执行时间对比（image-128-2w）

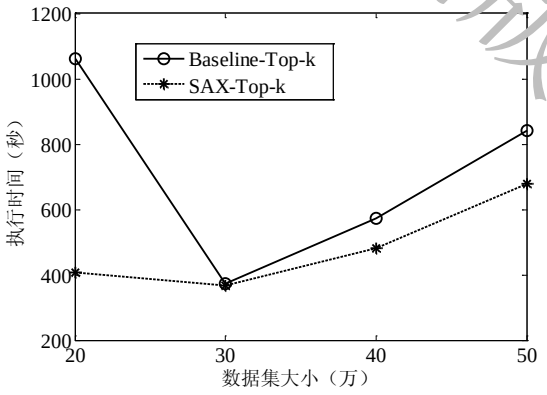
图 5 展示了针对 image-960 数据集，不同的 *k* 对执行时间的影响。其变化规律与 image-128 数据集基本一致，不过在 image-960 数据集中，当 *k* 为 15000 和 20000 时，Baseline-Top-*k* 的执行时间是一样的，这是因为在阈值为 0.1 时，返回结果的对数是 10566，阈值为 0.15 时，返回结果的对数是 30486，15000 和 20000 都在两者之间，所以在阈值为 0.15 时可以同时满足 15000 和 20000 的要求，故两种情况下的时间相同。

图 6 展示了在单机环境下两种算法的性能对比情况。因为是在单机下执行，所以选择的是较小的数据集（image-128-2w），*k* 的取值分别为 50、100、200、400、800。在执行 Baseline-Top-*k*-Serial 时，阈值依次选择为：0.05，0.1，0.15，0.2，0.25，0.3，0.35，0.45，0.5。从图 7 可以看出，SAX-Top-*k*-Serial 的性能优于 Baseline-Top-*k*-Serial 的性能，并且随着 *k* 的增大，SAX-Top-*k*-Serial 的相对性能越好。

7.2.2 不同数据集大小对执行时间的影响

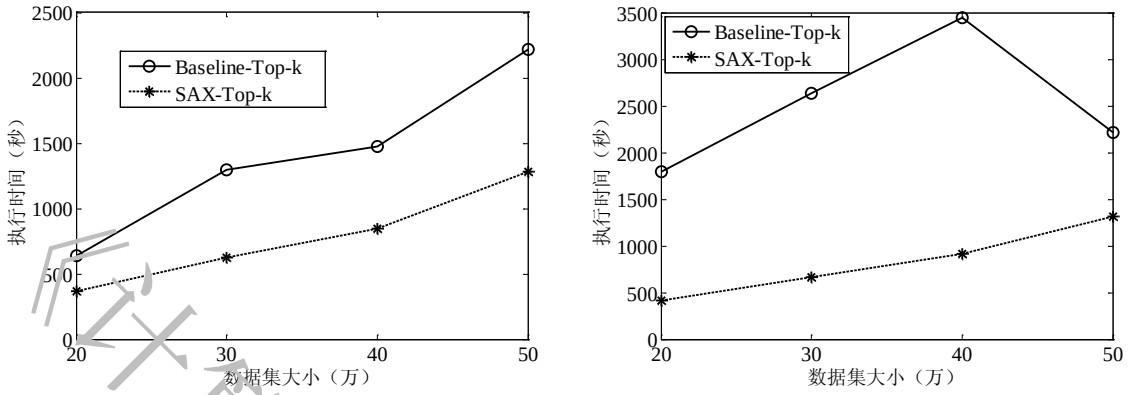
图 7 展示了数据集 image-128 在不同规模下的执行时间，图 7-a 和 7-b 分别表示 *k*=1000 和 *k*=5000 情况下的执行时间对比。从图 7-a 中可以看出，在所有数据规模情况下，SAX-Top-*k* 的执行时间都要小于 Baseline-Top-*k* 方法。当数据集超过 30 万的时候，随着数据集规模的增加，SAX-Top-*k* 的性能越来越好。从图 7-a 中我们还可以看到一个特殊变化：Baseline-Top-*k* 在数据集为 30 万时的执行时间小于 20 万时的执行时间，主要原因在于当数据集为 20 万的时候，要想得到预期数量的相似对，Baseline-Top-*k* 就需要执行 4 次计算，而在 30 万的时候，只需要执行一次计算就可以了，所以前者消耗的时间相对较长。图 7-b 显示了 *k*=5000 时的执行时间对比，其变化规律与 *k*=1000 时基本一致，不过总的执行时间要大于 *k*=1000 时的执行时间。





（a）k = 1000　　（b）k = 5000

图 7. 不同数据集大小对执行时间的影响（image-128）



（a）k = 1000　　（b）k = 5000

图 8. 不同数据集大小对执行时间的影响（image-960）

图 8 展示了数据集 image-960 在不同数据规模下的执行时间，总体来看，随着数据集规模的增大，两种算法的执行时间都有所增加，但是 SAX-Top-*k* 的执行时间一直都小于 Baseline-Top-*k* 的执行时间，并且增加速度也较小。Baseline-Top-*k* 的执行时间有时会出现波动（图 8-b：30 万 vs. 40 万），其原因在于 Baseline-Top-*k* 的执行时间既与每次的执行时间相关，又与执行的次数相关，而执行的次数则与具体的数据集的分布有关，所以就可能会出现在数据集比较大的情况下，执行次数比较少，相应的总的执行时间就比较短。

7.2.3 不同维度对执行时间的影响

图 9 展示了在不同维度下两种算法的执行时间对比，其中 k 为 5000，向量规模为 50 万。从图中可以看出，随着维度的增大，两种算法的执行时间都不断增加，但是 SAX-Top-k 算法的时间增长速度比较平缓，并且，维度越高，SAX-Top-k 算法的性能优势越明显。

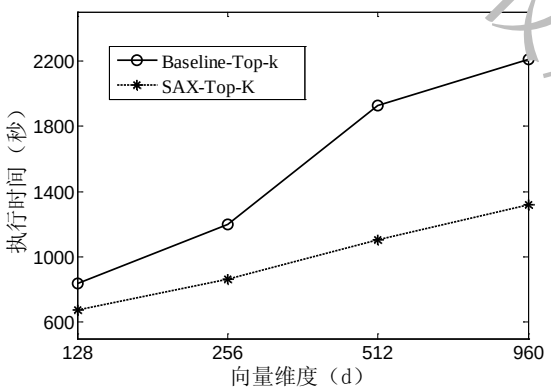


图 9. 不同维度对执行时间的影响（k=5000, n=50 万）

7.2.4 阈值估计时间及效果

图 10 展示了不同的采样规模下阈值估计所需要的时间，从图中可以看出，随着采样数量的增加，阈值估计所需要的时间也会逐渐增加；image-960 所需要的时间比 image-128 所需要的时间长，这是因为在同样的采样规模下，image-960 数据集的维度比 image-128 数据集的维度高，所以计算距离时所需要的时间也比较长，因此总的阈值估计时间也会比较长。

图 11 展示了在不同的采样规模下，所得到的估计阈值的大小。从图中可以看出，随着采样规模的增大，估计的阈值会逐渐减小，这说明采样规模越大，得到的估计阈值越接近真实的第 *k* 个对的距离阈值。而 image-128 的阈值比 image-960 的阈值要大一些，这主要是与数据集的数据分布有关，与维度没有直接的关系。

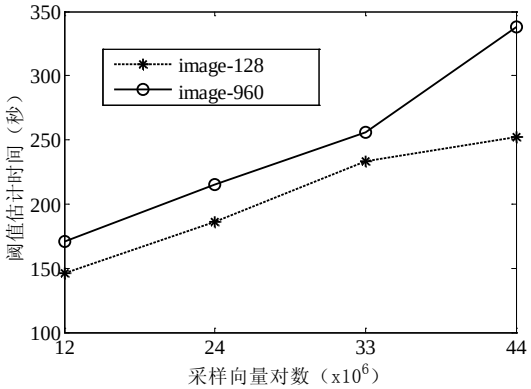


图 10. 不同采样规模所需的阈值估计时间

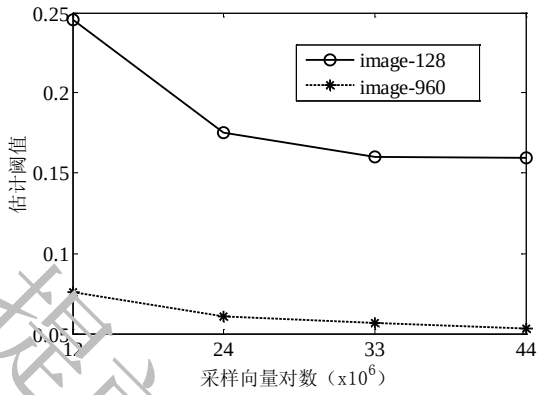


图 11 不同采样规模得到的估计阈值

8 结论

本文基于分段累积近似法（PAA）和符号累积近似法（SAX），对高维向量进行降维、分组，然后根据原始向量的 SAX 分组信息对原始向量进行抽样，从而估计出距离最小的第 *k* 个向量对的距离 ε,然后将 Top-*k* 连接查询转换成以 ε 为阈值的阈值连接查询。结合 MapReduce 框架，提出了基于 SAX Top-*k* 连接查询算法，实验表明，所提方案具有良好的性能。

未来将对现有的方案做进一步的优化：在 MapReduce job 执行计算的时候，在 reduce 端，每一个 reduce 任务结束之后，第 *k* 对的相似度阈值可能会发生改变，如果能够让后面的 reduce 任务在执行的过程利用更新后的阈值，则可以进一步提高过滤效果，加快计算速度。

参考文献

[1] Corral, A., Manolopoulos, Y., Theodoridis, Y., Vassilakopoulos, M. Closest pair queries in spatial databases//Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'00). Dallas, Texas, USA, 2000: 189-200

[2]Corral, A., Manolopoulos, Y., Theodoridis, Y., Vassilakopoulos, M. Algorithms for processing k-closest-pair queries in spatial databases. Data & Knowledge Engineering, 2004, 49(1): 67-104

[3]Dean, J., Ghemawat, S. MapReduce: simplified data processing on large clusters//Proceedings of the 6th USENIX Symposium on Operating Systems Design and Implementation (OSDI'04). San Francisco, California, USA , 2004: 137-150

[4] Pang Jun, Gu Yu, XuJia, Yu Ge. Research Advance on Similarity Join Queries. Frontiers of Computer Science and Technique, 2013, 7(1): 1-13(in Chinese)

（庞俊,谷峪,许嘉等. 相似性连接查询技术研究进展. 计算机科学与探索, 2013, 7(1): 1-13）

[5] Böhm, C., Braunmüller, B., Krebs, F., Kriegel, H. Epsilon grid order: An algorithm for the similarity join onmassive high-dimensional data//Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'01).Santa Barbara, CA, USA, 2001: 379–388

[6] Jacox, E.H., Samet, H..Metric space similarity joins.ACM Transactions on Database Systems, 2008, 33(2): 1-38

[7] Zhu, M., Papadias, D., Zhang, J., Lee, D.L.. Top-k Spatial Joins. IEEE Transactions on Knowledge and Data Engineering, 2005, 17(4): 567-579

[8] Katoh, N., Iwano, K.. Finding k farthest pairs and k closest/farthest bichromatic pairs for points in the plane//Proceedings ofSymposium on Computational Geometry (SOCG'92). Berlin, Germany，1992: 320-329

[9] Lopez, M.A., Liao, S. Finding k-closest-pairs efficiently for high dimensional data//Proceedings of the 12th Canadian Conference on Computational Geometry (CCCG'00). Fredericton, New Brunswick, Canada, 2000: 197-204

[10] Vernica, R., Carey, M.J., Li, C. Efficient parallel set-similarity joins using MapReduce//Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'10), Indiana, USA, 2010: 495-506

[11] Rong C, Lu W, Wang X, Du X, Chen Y, Tung A. Efficient and scalable processing of string similarity join. IEEE Transactions on Knowledge and Data Engineering, 2013, 25(10): 2217-2230

[12] Elsayed, T., Lin, J.J., Oard, D.W.. Pairwise document similarity in large collections with MapReduce//Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL'08). Columbus, Ohio, USA, 2008: 265-268

[13] Metwally, A., Faloutsos, C.. V-SMART-Join: A scalable MapReduceframework for all-pair similarity joins of multisets and Vectors//Proceedings of the VLDB Endowment, 2012, 5(8): 704-715

[14] Baraglia, R., Morales, G.D.F., Lucchese, C. Document similarity self-join with MapReduce//Proceedings of IEEE International Conference on Data Mining (ICDM'10) . Sydney, Australia, 2010: 731-736

[15] Seidl, T., Fries, S., Boden, B. MR-DSJ: distance-based self-join for large-scale vector data analysis with MapReduce//Proceedings of the 15th BTW conference on Database Systems for Business, Technology, and Web(BTW'13). Magdeburg, Germany, 2013: 37-56

[16] Fries S., Boden B., Stepien G., SeidlT..PHiDJ: parallel similarity self-join for high-dimensional vector data with MapReduce//Proceedings of the 30th IEEE International Conference on Data Engineering (ICDE' 14). Chicago, IL, USA, 2014: 796-807

[17] Lu W, Shen Y, Chen S, and B.C. Ooi. Efficient processing of k nearest neighbor joins using MapReduce//Proceedings of the VLDB Endowment, 2012, 5(10): 1016-1027

[18] Zhang, C., Li, F., Jestes, J.. Efficient parallel kNN joins for large data in MapReduce//Proceedings of the 15th International Conference on Extending Database Technology(EDBT'12). Berlin, Germany , 2012: 38-49

[19] Liu Yi, Chen Luo, Jing Ning, Liu Lu. Parallel top-k spatial join query processing on massive spatial data, Journal of Computer Research and Development, 2011, 48(Suppl.): 163-172(in Chinese)

（刘义,陈荦,景宁,刘露. 海量空间数据的并行 Top-k 连接查询. 计算机研究与发展, 2011, 48(Suppl.): 163-172）

[20] Lei Bin, Xu Jia, Gu Yu, Yu Ge. Parallel top-k similarity join algorithm on large probabilistic data based on earth mover's distance. Journal of Software, 2013, 24(S2): 188-199(in Chinese)

（雷斌,许嘉,谷峪,于戈. 概率数据上基于EMD距离的并行Top-k相似性连接算法. 软件学报, 2013,24(S2): 188-199）

[21] Huang J., Zhang R., Buyya R., Chen J. MELODY-JOIN: efficient earth mover's distance similarity joins using MapReduce//Proceedings of the 30th IEEE International Conference on Data Engineering (ICDE'14). Chicago, IL, USA, 2014:808-819

[22] Kim, Y., Shim, K.: Parallel top-ksimilarity join algorithms using MapReduce//Proceedings of the 30th IEEE International Conference on Data Engineering(ICDE'12). Chicago, USA, 2012: 510-521

[23] Keogh, E.J., Chakrabarti, K., Pazzani, M.J., Mehrotra, S. Dimensionality reduction for fast similarity search in large time series databases. Knowledge and Information Systems, 2001, 3(3): 263-286

[24] Lin, J., Keogh, E.J., Lonardi, S., Chiu, B.Y. Asymbolic representation of time series, with implications forstreaming algorithms//Proceedings of the 8th ACM SIGMOD Workshop on Research

Issues in Data Mining and Knowledge Discovery (DMKD'03). San Diego, California, USA, 2003: 2–11

[25] Luo, W., Tan, H., Mao, H., Ni, L.M. Efficient similarity joins on massive high-dimensional datasets using MapReduce//Proceedings of the 13th International Conference on Mobile Data Management (MDM'12). Bengaluru, India, 2012: 1-10

MA You-Zhong, born in 1981, Ph.D.. His research interests include Web data and Cloud data management.

CI Xiang, born in 1986, Ph.D. candidate. His research interests include Cloud computing and Big data management, Online aggregation.

MENG Xiao-Feng, born in 1964, professor and Ph. D. supervisor. His research interests include Web data management,Cloud data management, Mobile data management, XML datamanagement, Flash-aware DBMS.

Background

High-dimensional similarity join(HDSJ) plays an important role in many practical applications, such as near duplicated images or video detection, time series clustering and many other data mining algorithms. The aim of High-dimensional similarity join isto find out all the pairs of similar objects whose similarity is larger than a predefined threshold.

However in most cases, we can't get the desired threshold previously. If the chosen threshold is too small, we can't get enough result, otherwise, it will cost too much time. Top-k join can deal with the above problems in some extension. Given two vector sets R and S, a Top-k join returns k closest pairs of vectors. It doesn't need to know the threshold.

Top-k similarity join on high-dimensional data is facing two challenges: the first one is that the scale of the datasets is becoming more and more larger(millions or billions of objects); the second one is that the dimensionality of the objects is sufficiently high(thousands or tens of thousands). Because of the above two reasons, the traditional single machine-based centralized processing mode can't solve the Top-k similarity join on high-dimensional data problem in an acceptable time. Although there are many existing research works focusing on Top-k join problem, they can only deal with not very high dimensional vectors and most of them are running on single computer.

Recently, MapReducehas become the most popular platform that can deal with large scale datasets efficiently because of its high scalability, high availability and fault tolerance. Many research works have been done to deal with the similarity join problems using MapReduce, but most of them focus on the set or string data, the similarity measures are Jaccard similarity, cosine similarity et al., they can't be used to deal with the vector data directly with Euclidean distance.

In this paper we adopt Piecewise Aggregate Approximation technique (PAA) to reduce the dimensionality of the vectors; then group the vectors using Symbolic Aggregate approximation technique (SAX); based on the MapReduce framework, we propose SAX-based parallel Top-k join query algorithms. The experiments results show that the proposed approaches have better performance and scalability.

This research was partially supported by the grants from the Natural Science Foundation of China (No. 61379050, 91224008); the National 863 High-tech Program (No. 2013AA013204); Specialized Research Fund for the Doctoral Program of Higher Education(No. 20130004150001), and the Fundamental Research Funds for the Central Universities, and the Research Funds of Renmin University(No. 11XNL010).