



一种自适应子空间相似性搜索方法

任建新, 陈华辉

(宁波大学信息科学与工程学院 宁波 315211)

摘要:近年来,在多媒体信息检索、相似性连接和时间序列匹配等数据库领域的相似搜索研究备受关注。绝大部分工作都是在欧式空间条件下,使用度量距离函数计算最近邻(如 kNN 、 $kNNJ$)来解决搜索目标集合问题。但已有研究表明,此条件下的搜索结果准确性很容易受到高差异维度的影响,且对应的解决方案尚缺乏灵活性和顽健性。首先提出了单机环境下动态子空间(部分维度)下相似搜索问题及解决方案。随着数据规模的扩大,单机算法不能很好地扩展,随之又提出了 Hadoop 框架下的分布式算法。实验证实,在不影响准确率的情况下,分布式算法的性能要优于集中式算法。

关键词:自适应子空间;相似性搜索;非度量距离方法;MapReduce 分布式计算框架

doi: 10.11959/j.issn.1000-0801.2015190

An Adaptive Subspace Similarity Search Approach

Ren Jianxin, Chen Huahui

(College of Information Science and Engineering, Ningbo University, Ningbo 315211, China)

Abstract: In recent years, such database fields as multimedia information retrieval, similarity join and time series matching, where similarity search has attracted much attention. Existing researches mostly compute nearest neighbor to solve problems about search target set, such as kNN and $kNNJ$, by metric distance functions in the Euclidean space. But some studies showed that high dissimilarity dimensions had got great effect on the accuracy of answer and flexibility and robustness still were lacked in corresponding solutions. Thus centralized dynamic subspace or partial dimensions similarity search problem and algorithms were proposed at first. Furthermore, with the emerge of very large dataset, centralized algorithms cant extend very well. Finally, the distributed ones under hadoop framework were proposed. Experiments prove that distributed algorithms outperform centralized ones in the performance without accuracy loss.

Key words: adaptive subspace, similarity search, non-metric distance method, MapReduce distributed computing framework

1 引言

相似性搜索的任务是给定查询对象,从数据库中查找距离其最近的对象。一般采用具有度量属性的距离方法

(如欧氏距离),聚合两个对象在所有维度属性上的差异,从而根据得分衡量两个对象是否相似。然而,在实际应用中,有时由于误读或噪声而导致提取的数据属性集合中出现部分高差异属性值,进而对相似性度量产生较大的影

收稿日期:2015-03-02;修回日期:2015-07-13

论文引用格式:任建新,陈华辉.一种自适应子空间相似性搜索方法.电信科学,2015190

Ren J X, Chen H H. An adaptive subspace similarity search approach. Telecommunications Science, 2015190



响,所以简单的差异聚合处理势必大大降低搜索结果的准确率。样本数据库如图1所示,假如其中每一条记录均由从图像提取的特征向量构成,且前两个维度表示颜色,第3个维度表示文本描述,后3个维度表示形状。当给定查询对象 $Q=[1,1,1,1,1,1]$,那么基于欧式距离的Top-1相似性搜索结果是对象B。但是很容易看出,对象A和C在除 $d3$ 和 $d5$ 之外的其他维度上的属性值与查询对象更相近,也正是由于A. $d3$ 和C. $d5$ 高差异性维度的存在,所以才认为这两个与Q更相似的对象是更不相似的。

对象	$d1$	$d2$	$d3$	$d4$	$d5$	$d6$
A	1	2	50	1	1	2.2
B	5	5	5	5	5	5
C	2	1	1	2	48	1.5

图1 样本数据库

尽管后来有研究^[4]尝试着解决那些高差异维度的影响,但却局限于固定的或静态的子维度或部分维度空间中的对象间属性值差异的聚合,搜索结果的准确率有所改善但尚缺乏灵活性。再以图1为例,假定两个对象在子空间 $\{d1, d2\}$ 范围内计算相似度,Top-2集 $\{A, C\}$ 在每一维上与Q的差异最小且最相似,与实际相符;类似地,若在子空间 $\{d1, d2, d3\}$ 范围内计算,Top-2集 $\{C, B\}$ 与Q最相似,却与实际不符。由此而知,不能自适应地选择维度子空间,直接影响了相似性搜索结果的差异性,故本文提出了动态或自适应子空间下的相似性搜索问题。另外,随着互联网的快速发展,搜索处理的数据量也急剧增长,而MapReduce作为Hadoop平台下的分布式计算框架,能够满足大规模数据处理需求。总之,目前基于子空间的相似性搜索研究还不够成熟,且MapReduce框架下的研究工作更是没有。因此本文的主要工作就是针对现有的问题,提出高稳健、高效率且可扩展的子空间相似性搜索的解决方案。

本文主要贡献如下:

- 针对静态子空间下相似性搜索方案的局限性,提出了自适应子空间下朴素算法及其对应的分布式算法;
- 朴素算法的时间成本比较高,因此从减少访问对象属性数目的角度出发,设计并实现了高效且稳健的优化算法及其对应的分布式算法;
- 最后,进行了大量的实验论证,一方面表明了自适应子空间搜索算法的有效性;另一方面也验证了在

保证准确率的情况下,分布式搜索算法性能优于集中式算法,且解决了算法的可扩展性问题。

2 相关工作

在数据库领域,数据对象往往会被表示成特征点向量,那么相似性搜索^[2-5]就可以转化成空间中点对象的最近邻搜索问题,比如在基于位置的地理信息服务、数据挖掘和图像音频检索^[6]等应用中体现尤为突出。参考文献[1,7,8]提出R-tree及其变种、KD-tree、VA-file等数据或空间划分的方法,利用分支限界法,提前剪掉一些不必要的数据访问,从而提高精确最近邻搜索性能;之后,为了进一步加速相似性搜索效率,参考文献[9~12]提出了近似最近邻搜索方法,如散列-file、LSH和multi-LSH等,它们通过空间映射即降维将原始空间中的对象散列到新的空间,此时依然保证对象间的邻近性,从而利用这种特性在低维空间(如一维)执行快速搜索。比如LSH,它的基本思想是将空间中点对象散列到对应的桶中,且满足原始空间中相邻的对象能够以较大的概率落到相同的桶中,最后再根据欧式距离确定结果集。

但是,上述方法均采用具有度量属性,即满足正定性、对称性和三角不等式的距离函数(如欧式距离、海明距离和编辑距离等)来度量邻近关系,搜索结果存在一定的缺陷。随后,参考文献[13]提出了不完全满足度量属性的布雷格曼(Bregman)散度,作为NN和范围查询的距离度量,以适应概率分布^[6]、时间序列等特殊搜索应用。比如KLD,它是非对称的,但可以用来进行音频相似搜索,其中,音频对象被表示成高斯混合模型。参考文献[1,14~16]提出了在固定或静态维度子空间下的相似搜索,对于图1,如果Q与A在 $\{d1, d2, d4, d5\}$ 空间中聚合维度差异,那么现有的子空间相似搜索方法就要求Q与B和C也需在相同的子空间中聚合比较,尽管具备一定的合理性,但维度子空间的选择以及距离度量方法仍缺乏灵活性。

最后,随着大数据时代的到来,解决集中式算法的可扩展性问题的研究也越来越多。其中,参考文献[17]提出了RankReduce方法,它很好地将LSH和MapReduce相结合,解决了大规模数据上的 k NN问题。同样参考文献[18]也借助了MapReduce计算框架,但它解决的是大规模密集型应用中的 k NNJ问题。而且上述文献的实验结果也证实分布式算法不仅解决了集中式算法的可扩展性问题,还提高了算法的运行效率。鉴于此,本文也尝试着利用其解决动

态子空间下相似性搜索所面临的相同问题。

3 问题描述与主要内容

依据现有的文献资料证实,在现实应用中,静态子(或部分)空间下的相似性搜索问题也是有意义的。再比如案件侦察,根据受害人描述的特征构造出头像 E,并在数据库中查询符合条件的头像,虽然由于描述的各种不确定性导致返回的结果集(假设包含 E1、E2)与 E 不可能 100% 匹配,但依然有利于案件的侦破,这也体现出子空间相似性搜索的优越性。但有时,由于某方面特征的偏差,出现类似于 E1 与 E 在眼睛、眉毛特征上更匹配,而 E2 与 E 在眼睛、发型特征上更匹配的情形,从而认为 E1 和 E2 都与 E 相似也是合理的。针对这种支持自适应特征子空间的相似性搜索问题,本文先后提出了单机环境下解决方案及其对应的分布式方案。

3.1 相关定义

针对部分维度空间中的相似性搜索问题,如果要想实现 E1、E2 与 E 自适应地匹配最相似的特征,首先是找到一种距离方法来衡量对象间的相似性,因此本文引入了匹配差异的概念。

定义 1 (*N* 匹配差异) 已知数据对象 $P=\{p_1,\cdots,p_i,\cdots,p_d\}$, 查询对象 $Q=\{q_1,\cdots,q_i,\cdots,q_d\}$, $N\in[1,d]$ 。假设 $Diff_i=|q_i-p_i|$, 其中, $i=1,2,\cdots,d$, 那么把 $Diff_i$ 称为两个对象在维度 i 上的匹配差异, 把第 N 小的 $Diff_i$ 称为关于 Q 的 P 的 N 匹配差异, 记作 $N_MD[Q,P]$, 简称 N_MD 。

如图 2(b) 所示, $\{Diff1, Diff2\}$ 表示查询对象与数据对象在相应维度上匹配差异的集合, 注意这里并没有对差异进行聚合。举例说明, 1_MD 表示当 N 取 1 时, $\{Diff1, Diff2\}$ 中排名第一小的值即 0.5, 同理 2_MD 则是排名第二小的值即 2, 在高维条件下则以此类推。需要说明的是, 已知 A 和 Q 、 B 和 Q 、 A 和 B 的 1_MD 分别是 0.5、0.5 和 2.5, 那么不等式 $0.5+0.5>2.5$ 并不成立, 所以 N_MD 是一种非度量

距离方法, 且不同于参考文献[14,15]中的子空间距离度量方法。

由匹配差异的定义, 引出了下面的自适应或动态 N 维子空间相似性搜索问题。

定义 2 (*Top-K* 自适应 N 维子空间相似集合, 简称 *Top-K AN2S* 集合) 给定维数为 d 的数据库 DB , 查询对象 Q , 整数 K 和 N 。从 DB 中寻找一个包含 K 个对象的集合 R 且对于任意对象 $O1\in R, O2\in DB-R$ 时, 满足条件 $N_MD(Q, O1)\leq N_MD(Q, O2)$, 那么就称集合 R 是 Q 的 *Top-K AN2S* 集合, 其由关于查询对象 Q 的 N_MD 最小的前 K 个数据对象组成。

本文把计算 *Top-K AN2S* 集合的过程称为 *Top-K* 自适应 N 维子空间相似性搜索问题。如图 2(a) 所示, 解决该问题的朴素方法 (详见第 3.2 节) 的处理结果如下: 选取 1_MD 最小值 0.5 对应的数据对象 $\{A, B\}$ 作为 Q 的 *Top-1 A{1}2S* 集, 其中, $\{1\}$ 表示 N 取 1, 而选取 2_MD 最小值 2 对应的数据对象 $\{A\}$ 作为 Q 的 *Top-1 A{2}2S* 集; 同理, 选取 2_MD 值最小的两个 $\{2, 3\}$ 对应的数据对象 $\{A, C\}$ 作为 Q 的 *Top-2 A{2}2S* 集。由此可见, 实际操作中, 以每个数据对象与查询对象在全维度空间下的匹配差异集合为基础, 根据指定的参数值 K 和 N , 确定最终的结果。这里需要注意的是, 通过 1_MD 度量得到两个对象 A 和 B 分别来自不同子空间 $\{d1\}$ 和 $\{d2\}$ 的贡献, 也就说明实际在计算相似度得分的时候, 算法 1 能够很好地动态或自适应地选择当前数据对象与 Q 更相似 (匹配差异更小) 的子空间优先参与相似度计算, 而不是所有的数据对象在统一的子空间中与 Q 进行比较。

然而, 朴素算法只是一种解决动态子空间搜索问题的基本方法, 需要计算每个对象与查询对象在所有维度上的 N_MD , 时间复杂度比较大。为了进一步优化算法, 本文考虑从查询对象的每一维初始边界对象 (详情见定义 3) 入手, 以最小边界匹配差异为指标, 逐步朝大或小的方向迭

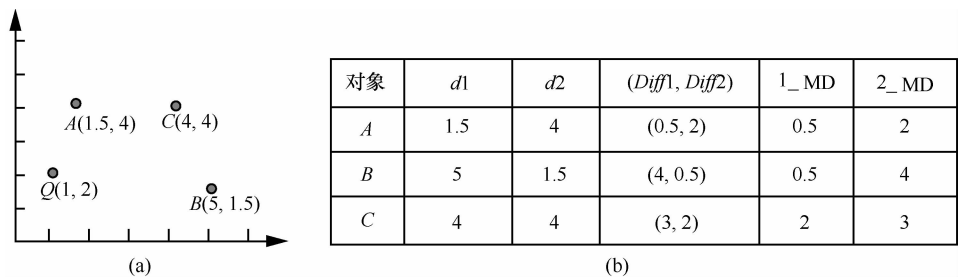


图 2 N_MD 概念



代扩展、更新边界,从而确定符合要求的 Top-K 自适应 N 维子空间下的搜索结果集。再如图 2(a)示例,假定数据集中还含有 D 、 E 和 F 等对象,且边界对象迭代更新序列是 $ABFEBA\cdots$,则优化算法会将序列中首次出现 2(即 N 取 2)次的对象 $\{B\}$ 作为 Q 的 Top-1 $A\{2\}2S$ 集,而将首次出现 3 次的对象 $\{A\}$ 作为 Top-1 $A\{3\}2S$ 集;类似地,将前 2 个依次出现 2 次的对象 $\{B,A\}$ 作为 Top-2 $A\{2\}2S$ 集,处理详情见第 3.3 节。

定义 3 (最小边界匹配差异) 定义如式 (1), 记作 MinBMD 。

$$\text{MinBMD} = \min_{0 \leq DD < 2d} \{BArray[DD]\} \quad (1)$$

其中, $BArray[DD]$ 表示边界数组,存放查询对象 Q 在各维度上的边界对象与 Diff_i 构成的对。假如 $a_i \leq q_i \leq b_i$ 且 a_i 和 b_i 分别表示对象 A 和 B 在第 i 维上的属性值,那么就称 A 和 B 当前是 Q 在第 i 维上的边界对象。

在边界对象迭代更新过程中,需要根据边界数组索引,确定当前欲更新对象的方向(朝大或小方向)以及所在维度,这里将两者合在一起抽象成为边界维度,详情见定义 4。

定义 4 (边界维度) 与边界数组关联,是其下标的抽象,包含数据维度 Dim 和边界更新方向 Direc 两个属性。定义如式(2),记作 BDIM 。

$$\text{BDIM} = \{\text{Dim}, \text{Direc}\} \quad (2)$$

如果边界数组 $BArray$ 的某一维下标是 DD_i , 则 $DD_i = \text{BDIM.Dim}_i + \text{BDIM.Direc}_i$, 其中 Dim_i 和 Direc_i 表示上述两个属性在维度 i 上的值。如图 3 所示,横向箭头右侧列即边界维度,通常已知 MinBMD 对应的 DD_i , 才计算得出 Dim_i 和 Direc_i , 从而确定将要更新的边界位置。例如,当 $\text{BDIM}=21$ 时,21 仅仅是上述两个属性值的拼接。

当 DB 中数据量很大且维数很高时,在一定程度上,顺序地更新 $BArray$ 中相同 MinBMD 值的边界时可能会影响

整个任务的运行效率,所以本文考虑采用并行的策略来改善优化后的算法性能,为此也引入了定理 1。

定理 1 如果存在 DD_m 和 DD_n , 且 $BArray[DD_m] = BArray[DD_n]$, 那么 $BArray[DD_m] \rightarrow BArray[DD_m+1]$ 及 $BArray[DD_n] \rightarrow BArray[DD_n+1]$ 可以并行执行,其中“ $\rightarrow$ ”表示边界数组的一次迭代更新过程。

证明 假定有两个 MinBMD 对应的 BDIM 分别是 $\{0,1\}$ 和 $\{1,0\}$ 且 $_{o}[0][1].\text{Diff}$ 等于 $_{o}[1][0].\text{Diff}$, $_{o}[][]$ 和 $_{n}[][]$ 分别表示更新前后 BDIM 处的边界值。当首先更新 $\{0,1\}$ 处的下界时, $_{n}[0][1].\text{Diff}$ 肯定大于 $_{o}[1][0].\text{Diff}$, 紧接着更新 $\{1,0\}$ 处的上界,相反亦成立。由此可知,在更新多个 DD_i 处的边界值时, $_{n}[][]$ 肯定大于 $_{o}[][]$, 即前面的更新不会影响 $_{o}[][]$ 的更新次序。既然如此,在大规模高维数据背景下就可以对两个或多个 BDIM 处的边界值同时进行处理即并行更新。

3.2 解决 Top-K AN2S 问题基本方案

当明确自适应子空间下的相似搜索问题之后,本文首先提出了基本解决方案 NA3S(naive adaptive subspace similarity search)算法。该算法是计算 DB 中每一个对象 O_i 关于 Q 的 N 匹配误差 $N_{MD}[Q, O_i]$, 其中, $i \in [1, \text{RecordNum}]$, $N \in [1, d]$ (i, N 都是自然数), RecordNum 是记录数,并且返回所有对象中 N_{MD} 值最小的 Top-K 集作为结果集。即使如此,朴素算法仅仅保证了搜索结果的准确性,但还是需要访问整个数据集中每个对象的所有属性值且计算其与 Q 对应维度属性的 Diff_i , 所以搜索过程可能比较耗时,具体算法实现略。为了便于朴素方案性能的实验对比,本文在实现了单机算法 NA3S 的同时,也分析研究实现了对应的基于 MapReduce 分布式计算框架的算法 NA3S_MR。算法核心伪代码如下。

过程 1 NA3S_MR(MR_job1 和 MR_job2)

MR_job1 的 UDF_Mapper 类的 $\text{map}(\text{key}, \text{value})$

$\text{key} :=$ 表示行偏移量; $\text{value}[] :=$ 存放当前记录被划分后的每一维属性值;

$\text{Queryobj}[] :=$ 存放用户指定的某个查询对象的各维属性值;

for $k0 = 1 : N$ //表示 $k0$ 维子空间

$\text{frontn}[] := \text{Diff}1 \sim k0$; //即存放 N 取 $k0$ 值时对应的 $k0$ 个匹配差异值

$\text{dimid} := 0$; //初始化 $k0$ 对应的增量;

while 依次取出 $\text{value}[]$ 和 $\text{Queryobj}[]$ 中每一维 i 的属

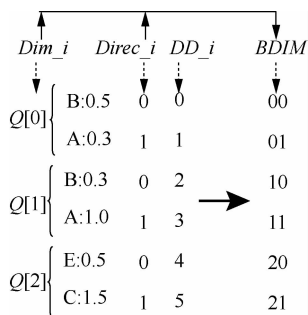


图 3 BDIM 概念表示

性值

```

Diffi := abs(value[i] - Querylist[i]);
if dimid 小于 k0
    frontn[dimid] := 当前的 Diffi;
    如果当前 Diffi 大于当前 frontn 中的最大值,更新当前最大值;
    dimid := dimid+1;
else if 当前 Diffi 小于当前 frontn 中的最大值
    更新 frontn 的最大值;
    for j = 1 : k0
        重新获取 frontn 最大值,即 frontn[k0];
    end while;
    output(k0,Rowcount+"-"+frontn[k0]); //优先输出不同维子空间下的匹配差异较小的对象集合
end for
MR_job1 的 UDF_Reducer 类的 reduce(key,iterator <value>)
    key := map 过程输出的 key; Re_set[K]:= 存放结果集;
    k:=0 //初始化参数 K 对应增量
    while 依次取出 iterator 引用的每一个 value
        value_k := frontn[k]; //取出当前 key 对应的 value 列表中的第 k 个元素
        if k 小于 K
            Re_set[k] := value_k;
            如果 value_k 大于当前 Re_set[] 中最大值,更新 Re_set 中的最大值及其对应的下标;
            i := i+1;
        else if value_k 大于当前 Re_set[] 中最大值
            更新当前 Re_set 中的最大值;
            for j = 0 : K-1
                if Re_set[j] 大于 Re_set[] 中当前最大值
                    将 Re_set[j] 和 j 分别作为当前最大值及其对应的下标;
                end for
            end while
        for m = 0:K-1
            output(key,Re_set[m]对应的下标+"-"+Re_set[m]); //取不同维数子空间中的 Top-K 个对象作为结果集
        MR_job1 是 NA3S_MR 算法核心作业,其中,map 过程的工作是计算每一个记录对象关于查询对象 Q 的 N_MD,

```

输出 key/value 格式为“k0,对象 ID:k0_MD”,其中,k0_MD 是 N_MD 的 N 取 k0 值时的匹配差异且 $k0 \in [1,d]$,k0 是自然数,d 是 DB 的维度。这里,单个 k0 仅仅对应数据对象的某一方面,可能还会倾向于在固定的特征空间中搜索目标集合,所以算法中的 k0 会从 [1,N] 中取多个不同连续值,最后再对多个 Top-K 子空间搜索结果集计算频繁项集,即 frequent Top-K AN2S 集,从而可以查询到与 Q 近似全维度相似对象,同时也避免了自适应选择维度的空间限于局部性。reduce 过程的工作是对相同 k0 的 value 列表进行约简分组,输出 key/value 格式为“k0,对象 ID:k0_MD”且 $k0 \in [1,N]$,而 MR_job2 的任务是将 MR_job1 的输出结果中相同 key 的 value 列表按照 Diff 升序输出 Top-K 个 k0_MD。最后,两个作业完成后就可以输出多个不同 N 下的 Top-K AN2S 集。

根据实验结果分析,朴素算法及其分布式方法的效率还不够理想,猜测原因可能在于实际参与计算的属性值的数量太大直接导致时间成本增加。因此本文又从减少访问属性数目的角度出发,设计实现了新的算法 NOA3S (naive optimizer adaptive subspace similarity search),验证此猜想。

3.3 NOA3S 算法基本思想

为了更好地引入优化算法 NOA3S,再以图 2 为例。当数据集中只含有 A、B、C 3 个对象且计算 Top-K 自适应 N 维子空间的对象集合(对于任意有效 K,N)时,按照算法 NA3S 的思路,首先要计算所有的匹配差异 Diff,接着提取每个对象的最小 Diff,构成 1_MD 集合 {0.5,0.5,2},然后再确定 Top-K 结果集。那么图 2 对应问题求解的时间复杂度就是 $O(VisitedAttriNum) + O(RecordNum \times \lg K)$,其中,VisitedAttriNum、RecordNum 分别表示访问的属性数目和记录数,且这里的 VisitedAttriNum 值是 $RecordNum \times d$ 。然而当数据集中的记录达到很大规模时,访问属性的数量也随之增加,并且复杂度也会有很大提升,所以本文就考虑是否可以在很大程度上通过减少属性的访问数量来优化朴素算法 NA3S,接着就提出了朴素优化动态子空间下的相似搜索方法。现假定数据库 DB(见表 1), $Q=\{3.0,5.5,4.5\}$, $K=N=2$,则 NOA3S 算法处理的大致过程如图 5 所示。

NOA3S 算法主要分两个阶段:数据预处理和边界数组更新阶段。以图 5 示例,对于表 1 的数据集 DB,预处理阶段的任务是对 DB 中每一维 a_i 进行排序(假定默认为升序);然后定位 $Q=(q_1, \dots, q_i, \dots, q_d)(i \in [1,d], i \text{ 是自然数})$ 的每一维在

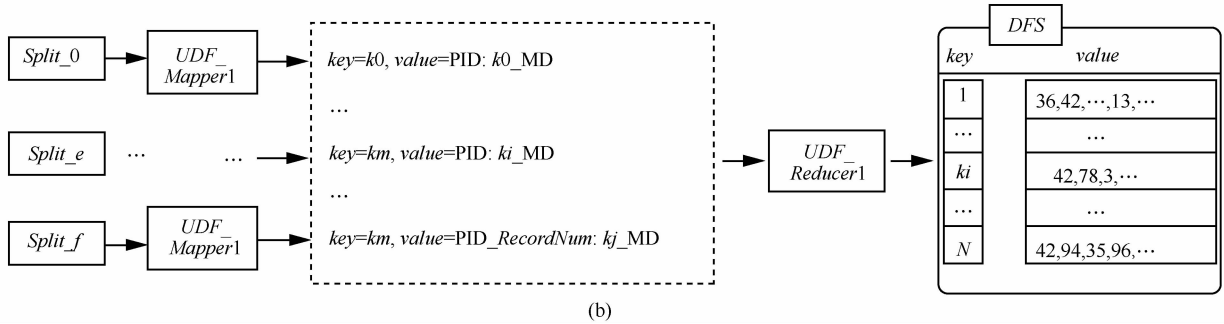
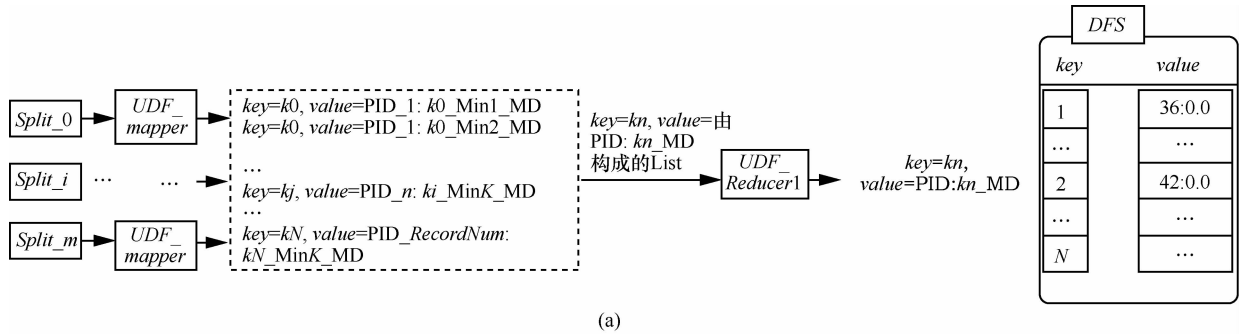


图4 NA3S 算法流程

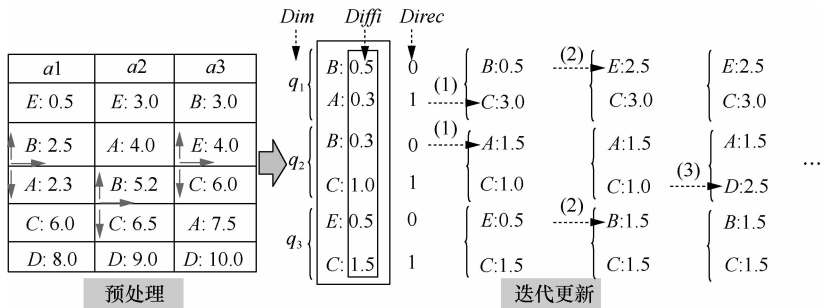


图5 NOA3S 算法流程

表1 样例数据集($d=3$, $RecordNum=5$)

对象	$d1$	$d2$	$d3$
A	3.3	4	7.5
B	2.5	5.2	3
C	6	6.5	6
D	8	9	10
E	0.5	3	4

当前维度 a_i 上的位置,如图5中左边的横向实细箭头指示,这里 q_i 的位置采用边界二元组标识,比如 $\langle B:0.5, A:0.3 \rangle$ 且规定前者是上界,后者是下界;最后将对象 ID: $Diffi$ 对顺序地保存到边界数组 $BArray[DD_i]$ 中,其中, $DD_i \in [0, d-1] \cup [0, 2d-1]$ 。对照表1 样例数据集,边界数组初始化如图5 虚线框部分。预处理工作完成之后,还需要设计其他的数据结构,如 $BDIM$,标志位数组和散列表,为后续边界更新做准备。算法中定位边界核心部分的实现如过程2 描述。

过程2 LocateBound($DB[\cdot][i], q_i, cflag$)

输入: $DB[\cdot][i]$: 数据集中已排好序的第 i 维列向量, q_i 为 Q 的第 i 维属性值, $cflag$ 为边界末端标志,取值-1,1。

输出: 当前维度所在边界或边界末端标志。

- (1) $vectoriter[]$ //存放边界迭代器的向量容器
- (2) $loop_iter$ //初始化循环迭代器,指向当前维度列向量的首元素
- (3) $_iter := NULL$; //中间变量
- (4) while $loop_iter++$ 未到达末尾
- (5) $_iter := loop_iter$;
- (6) 如果 $loop_iter$ 位置的值等于 q_i ,将边界位置的上界和下界的位置顺序存到 $vectoriter$ 再退出;
- (7) 否则,如果 $loop_iter$ 位置的值等于 q_i 且当前维度无上界,那么 $vector_iter$ 保存下界迭代器且 $cflag$ 置为-1再退出;

(8) 再否则, 如果 loop_iter 位置的值小于 q_i 且当前维度无下界, 那么 vector_iter 保存上界迭代器且 cflag 置为 1 再退出;

(9) end while;

(10) if(vector_iter 有两个元素) 将两个元素按照属性值升序排列, return $\text{vector_iter}[]$

对于整个数据集, d 次调用过程 2, 得到初始化边界数组。通常情况下, 1 次调用返回 q_i 维所在位置的 1 个上界(属性值小的方向)和 1 个下界(与前者相反), 如图 5 中大括号所指。特殊情况下, 当 q_i 到达维度首或末端时, 返回值仅有一个元素。这种情况下, 算法 NOA3S 并没有立即在过程 2 中将上/下界中的 Diff_i 设置为 Max(即无穷大值), 而是设定了 cflag 标志位和 flagmap 矩阵(d 个 2×2)对边界数组 $BArray$ 进行中转处理, 目的就是得到有效的初始 $BArray$, 从而使得算法更加顽健。如图 6 所示, 已知 q_1 对应的 $\text{cflag}=-1$, 说明其已经到达当前维度首端, 同时设置对应维度的 $\text{flagmap}[0][]$ 是 $\{0, -1\}$, 那么过程 2 返回后, $BDIM$ 的上/下界再指定为端对象 ID: Max; 同理, 已知 $\text{cflag}=1$, 说明其已经到达当前维度末端, 此时 $\text{flagmap}[1][]$ 设定为 $\{1, 1\}$, $BDIM$ 值的指定同上。另外, 在算法的第二阶段, 由于主程序要确定边界的更新方向, 故设计了 $BDIM.Direc$ 表示当前边界维度上的更新方向, 并且规定当 $Direc=0$ 时, 表示朝属性值小的方向更新, 而 $Direc=1$ 则相反(如图 5 中 $Direc$ 标识箭头所指)。

NOA3S 算法的主程序正式确定初始化的 $BArray$ 之后, 根据设定的参数 K 和 N 的值, 正式进入 Top-K 自适应 N 维子空间下的搜索过程。此过程大致又分为查找 MinBMD 阶段和对应下标 DD_i 处的上界或下界迭代更新阶段。前者比较简单, 可以采用二分查找的方式进行; 而后者是该算法的第二个核心点, 随着更新过程的推进(详情见过程 3), 在满足迭代终止条件下, 生成迭代更新序列, 最后通过最好优先选择策略确定结果集 S , 可参照定义 2

部分中的实例说明。另外, 实验证实第二阶段迭代次数 IterNum 是由 K 和 N 共同决定。一般情况下, 值越大, 迭代次数也越多, 且参与计算的属性数目也越多。由此, 可以推论优化算法的平均时间复杂度是 $O(2 \times d + \text{IterNum} \times \alpha)$ 且 $\alpha \in [1, 2d]$ (α 是自然数)。通常, K 和 N 的取值不会很大且数据具有一般性, 所以优化算法的整体时间效率相比朴素方法会有较大改善。图 5 列出了前 3 次迭代更新过程, 其中, 圆形标号和横向虚箭头分别表示当前迭代次数和当前更新的 $BDIM$ 值。

过程 3 BAUpdate($DB[]$, $BArray[]$, $DD_i[]$)

输入: DB 按列向量升序排列数据集; $BArray[]$ 是边界数组; $DD_i[]$ 是 MinBMD 对应下标的数组。

输出: null。

start_iter // 起始位置, 指向边界数组首元素

for start_iter 的位置没有到达末尾; $BDIM.Direc_i: 0$

if start_iter 当前位置的 Diff 被 2 整除

$BDIM.Direc_i := 1$; $BDIM.Dim_i := (DD_i - BDIM.Direc_i)/2$;

if $BArray[BDIM.Dim_i]$ 转换前包含上界和下界

$db_start_iter := DB[BDIM.Dim_i]$ 的起始位置

$db_end_iter := DB[BDIM.Dim_i]$ 的尾部位置

根据边界维度 $BDIM$ 保存的维度 Dim 和边界方向 $Direc$, 确定当前需要更新位置的下一个元素在 DB 中的位置; 其中, 如果 db_start_iter 和 db_end_iter 分别到达了当前维度的首部和尾部, 那么就置下一个位置处的边界值是当前对象 ID: Max;

else if $BArray[BDIM.Dim_i]$ 到达当前维首或末端

根据过程 2 返回值 cflag 得到的 $\text{flagmap}_{BDIM.Dim_i}[0][1]$, 可以确定在特殊情况下边界的更新方向;

end for ;

return ;

上述算法中, 由 MinBMD 对应的索引, 很容易就可以

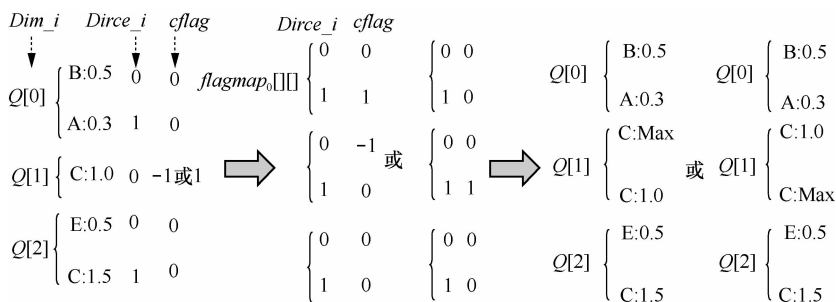


图 6 $BArray$ 变换处理



确定当前迭代更新目标。假设已知 DD_i , 若 DD_i 被 2 整除, 那么 $BDIM.Direc_i=0$, 否则 $BDIM.Direc_i=1$, 此外 $BDIM.Dim_i$ 的值等于 DD_i 减去 $BDIM.Direc_i$ 后再除以 2。需要注意的是, 某一次迭代可能会循环顺序更新一个或多个 DD_i 处的边界值, 这也为后面的分布式实现奠定了基础。类似于初始 $BArray$ 的转换, 过程(3)中的第(8)~(12)行、第(14)~(16)行都包含更新过程中对边界的判断处理。比如图 6 所示的情形, 当 $flagmap\ BDIM.Dim_i\ [0][1] = -1$ ($BDIM.Dim_i=1$) 时, $Q[1]$ 已到达当前维度的首端, 此时上界 $DiffBDIM.Dim_i$ 被设为 Max, 那么接下来迭代只能朝大的方向即下界进行更新, 对应过程 3 的第(14)~(16)行。当 $flagmap\ BDIM.Dim_i[0][1] == 0$ 时, 同理。

总之, 本文提出的优化算法 NOA3S 很好地处理了边界更新问题, 而且在改善原算法性能的条件下, 使得算法更为顽健。可是现实应用中, 随着数据规模的增长, 当前算法与 NA3S 存在同样的扩展性问题。针对这个问题, 之前考虑过采用单机下随机访问方法来应对数据增长, 但由于边界数组的更新的特殊性, 有时并不能很好地借助相对文件首部的偏移量来确定对应行的边界对象, 所以本文又在 Hadoop 框架下研究与实现了基于 MapReduce 分布式计算框架的 NOA3S_MR 算法。

3.4 NOA3S_MR 算法基本思想

由定理 1 可知, 维度对应的多个上界或下界在更新的时候可以并行执行, 因此本文从该角度出发设计并实现 Hadoop 框架下的 NOA3S_MR 算法。该并行算法主要由 3 个单一的 map/reduce 作业构成的一个链式 map/reduce 作业来执行。其中作业 1 的任务是将原始数据集按照各个维度进行排序, 并依照维度编号依次输入指定编号的文件当中, 这样做主要是为了实现简单的层次索引。再以图 5 为例, 假设将要更新的边界维度 $BDIM$ 是 $\{1,1\}$, 那么首先根据 $BDIM.Dim_i=1$, 可以快速地访问编号为 file_0001 的文件, 避免访问其他维无关的数据; 然后再根据 $BDIM.Dim_i=1$, 以 q_i 位置为基准, 朝大的方向提取下一条记录作为新的下界对象; 作业 1 的输出 key/value 对设计为(维度编号 d , 对象 ID: 对象当前维度值 $value_d$: 排序索引 ind), 其作为作业 2 的输入, 处理流程如过程 4。

继作业 1 之后, 作业 2 的任务是生成查询对象 Q 的初始化边界数组, 并将其存放到分布式文件系统(DFS)中, 其自定义 map 函数可以参照单机下的 LocateBound 方法实现, 输出 key/value 对格式同作业 1, 但此时的结果仅仅是

Q 的每一维在当前维度上的边界, 很大程度上降低了中间结果的数量, 处理流程如算法 5。另外, 对于 Q 的某一维 q_i , 边界的计算可能来自不同的 map 任务, 因此 map 端输出的上/下边界键值对数可能是 $2n(n \geq 1)$ 个。由于 q_i 只能有唯一的上界和下界, 所以在当前作业的 reduce 过程中对接收到的 value 值按照属性值的匹配差异排序并唯一地确定 $BDIM$ 对应的值。

作业 3 同单机算法一样, 也是该算法最为核心的部分, 即执行 $BArray$ 迭代更新过程直到解决 Top-K 自适应 N 维子空间的搜索问题时终止。对于作业 3, 在 map 任务开始前, 需要做一些准备工作, 如保存当前的迭代次数、从 DFS 读取上一次迭代的 $BArray$ 元素、MinBMD 以及迭代终止条件计算等, 接着 map 过程的工作是从 $BDIM.Dim_i$ 对应的编号文件中, 读取旧的边界在 $BDIM.Direc_i$ 方向的上(下)一个记录, 处理后作为新的上(下)界, 并输出到边界数组文件中。由于该作业只需要 map 任务就可以完成文件中 $BArray$ 数组的更新, 所以作业没有 reduce 过程(该作业算法伪代码略)。

过程 4 Job1.map (Mkey, Mvalue) while 某一条记录的每一维

output.write(维度编号 d , 对象 ID: 对象当前维度值 $value_d$: 排序索引 ind); // ind 初始化为 0

Job1.reduce(Mkey, MvalueIter)

MvalueSet:=Mvalue 集合; SortedNo:=1 // 数据有序时对应的索引编号

while MvalueIter.hasNext()

MvalueSet.add(MvalueIter.Next()); 更新当前 Mvalue 的 ind 为 SortedNo; SortedNo++;

foreach Mvalue in MvalueSet

output.write(维度编号 d , Mvalue);

过程 5 Job2.map(Mkey, Mvalue)

filesplitNo:=获取当前记录来自文件 ID, 即所在的维度 ID;

$q[filesplitNo]$:=从分布式缓存文件中获取 Q 的当前维度值;

借助单机的 LocateBound() 算法, 遍历编号为 filesplitNo 的文件寻找 $q[filesplitNo]$ 的维度区间

output.write(filesplitNo, 上界维度编号 upd : 维度值 $upvalue_d$: up_ind)

output.write(filesplitNo, 下界维度编号 $downd$: 维度值

```
downvalue_d:down_ind)
    Job2.reduce(Mkey,MvalueIter)
BoundvalueList:=边界值集合;
while MvalueIter.hasNext()
    BoundvalueList.add(MvalueIter.Next());
对 BoundvalueList 集合排序;
根据 BoundvalueList, 重新定位  $Q$  的当前维度值所在的边界;
output.write(当前维度编号  $d$ ,新的记录 ID:新的上/下界属性值:新的 ind );
分布式实现将单机循环顺序迭代更新变换为多个 map/reduce 任务并行完成,虽然设计复杂,但从一定程度上解决了扩展性以及效率问题。
```

4 实验分析与比较

本节重点就基本方法及 Hadoop 框架下的分布式算法做一些如有效性、时间效率和扩展性指标上的对比实验。

4.1 实验环境

本文部署和安装 4 台 PC 机作为 Hadoop 集群,其中,1 台作为主节点,剩余 3 台作为从节点。每台 PC 机配置一样,详细如下:Hadoop 版本是 1.1.2,操作系统是 RHEL6.5,JDK 为 1.7,CPU 为 Intel Core i5 3.1 GHz。实验主要采用两种数据集,分别是 COIL-100 和 CIFAR-100(大约 2.0 GB),其中前者包含 100 幅真实图像且每幅图像对应一个 54 维的特征向量,主要由颜色直方图、区域描述等特征表示;后者则包含 50 000 幅图像且每幅图像均由 3 072 维的特征向量表示。

4.2 自适应子空间相似性搜索算法有效性评估

与现有子空间下相似性搜索方案相比,本文针对新的非度量距离方法—— N 匹配差异,提出了自适应 N 维子空间下的相似性搜索问题,并给出了可行的解决方案。由于本文子空间下的相似搜索问题是相对欧氏距离下全维度空间中的相似性搜索问题存在的不足而提出的,这里有必要就实际搜索结果的准确率以及召回率证实新提出的子空间相似性搜索算法的有效性。以基本算法 NAN3S 为例,采用 COIL-100 真实图像数据集,假定查询对象 Q 是标号为 76 的图像且 K 取 5, N 在 [30,54] 范围取值,而这里 N 的选取源于大部分数据对象(包括查询对象)在较低维度范围内比较稀疏,且实际中主要考虑有意义的差异性的维度,即对应属性值不同为 0,最终算法返回的结果见表 2。相同

条件下,当 K 取 10 时,基于欧式距离度量的 k NN 搜索目标对象标号集合是 {76,23,19,38,100,47,46,56,69,75}。与数据集中真实图像对比,实际与查询对象相似的集合是 {76,69,6,8,15,19,23,27,91,100},那么采用上述两种不同度量方法下的搜索结果都有所差异,且经过统计计算比较,本文方法下的准确率和召回率相差不大,且综合得分要更高,因此就能够说明本文的匹配差异度量是合理的,而且采用其解决 Top- K 自适应 N 维子空间下的相似性搜索也是有效的。

表 2 Top-5 AN2S 目标集合 for 编号为 76 的查询图像

N 取特定值	Top-5 $A\{n^*\}2S$
$n^* \in \{30\}$	76,23,90,38,62
$n^* \in \{31,32,37\}$	76,23,38,84,69
$n^* \in \{33,34\}$	76,23,38,69,67
$n^* \in \{35,36\}$	76,23,38,69,27
$n^* \in \{38,39\}$	76,23,38,91,27
$n^* \in \{40\sim 44\}$	76,23,38,91,69
$n^* \in \{45\sim 53\}$	76,23,38,19,100
$n^* \in \{54\}$	76,23,19,47,100

4.3 朴素算法时间效率和扩展性比较

为了验证不同数据规模或维度条件下的时间效率问题,本文利用 MATLAB 工具首先剥离掉 CIFAR 数据集的类标签,接着将其转化成多个不同维数的数据集,如 50 000×32,50 000×54 等,然后实验设定 $K=4,k_0 \in [1,512]$,通过计算 Top- K 自适应 k_0 维子空间的搜索结果集,对 NA3S 和 NA3S_MR 做了实验对比分析。已知随着数据维度的变化,数据规模同样变化。对于表 3,由于起始时的分布式算法要完成作业配置、节点间资源分配调度等工作,所以在数据集较小时(比如小于或等于 54 维),性能还不如单机算法;随着维度的增加,分布式算法性能明显提升(如图 7 所示),尤其是当维数超过 128 时,两者运行效率的差距急剧增大。另外,本文也对朴素的分布式算法做了扩展性测试。如图 8 所示,随着从节点数的增多,集群就可以尽可能地调度多个节点并行处理整个作业,计算能力增强,从而可以很好地改善算法的运行效率。

表 3 数据维度与时间对比数据/min

算法	32	54	128	256	512
NA3S_MR	1.78	2.83	6.56	12.83	18.67
NA3S	2.67	4.51	10.61	20.06	43.60

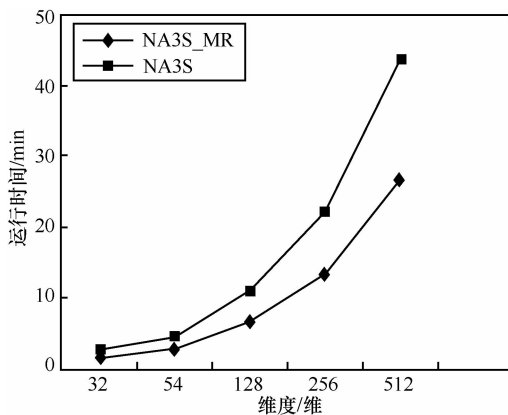


图7 数据规模(维度)与时间关系

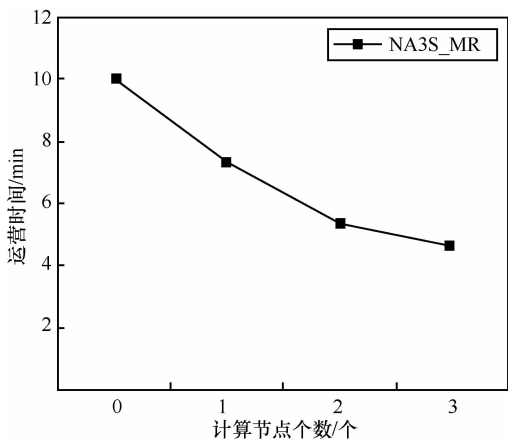


图8 计算节点数与时间关系

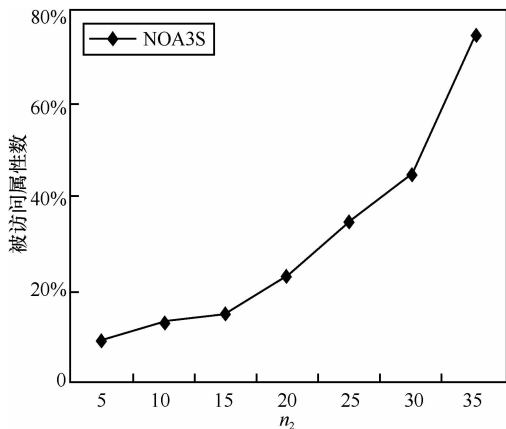
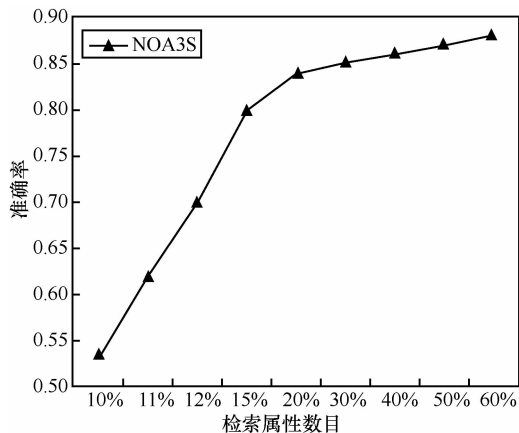
图9 被访问属性数与 n_2 的关系

图10 被访问属性数与准确率的关系

4.4 NOA3S 算法下参数影响

虽然 NA3S 算法分布式实现后,性能有较大改善,但随着维度的升高,时间效率依然不够理想,当然这也是朴素算法的缺陷。而改进后的算法 NOA3S,其关键是通过定位查询对象在排好序维度上的初始位置之后,再根据待求解问题的需要逐步向大或小的方向扩展,尽可能地减少了访问属性的数目,因而从理论上讲,其性能优于算法 NA3S,且适合处理更高维的数据。由于参数 K 和 $N \in [n_1, n_2]$ 的存在,本文在实验上使用 COIL-100 数据集验证了两者对算法性能的影响。如图 9 所示,曲线描述的是参数 N 的上界 n_2 与被检索属性数目之间的关系。实验中, K 取值为 4,且实验表明随着 n_2 的增长,检索属性的数目近似指数级增长。即使这样,往往 n_2 取值不是很大时就能满足 Top-K 自适应 N 维子空间的查询需求,所以与 NA3S 算法相比,从一定程度上减少了访问属性的数目。除此之外,本文还就被访问的属性数目与查询结果的准确率之间的关系进行了实验分析,如图 10 所示。由曲线可

知,检索属性数目的百分比在 10% 左右时,查询结果的准确率快速增长,之后准确率增长变化缓慢且有波动趋势。从本节实验可以得出,通过 n_2 的合理选择,搜索任务就可以在保证准确率的前提下,较大程度上提高算法的性能。

另外,从时间效率上,对比了单机环境下 NA3S 算法及其对应的优化算法 NOA3S。对于 NOA3S 算法,数据读取和列排序过程是基础,但也是最耗时的,此外迭代更新是本文设计优化的关键部分,但在固定条件下花费的时间平均在几秒之内,再加上被访问属性数目有所减少,因此优化后的算法在效率上优于朴素方法。

4.5 NOA3S 算法时间效率和扩展性比较

鉴于分布式下的朴素算法性能优于单机算法,本文还对 NOA3S 算法和对应的 NOA3S_MR 算法在相同条件下做了时间性能的比较。由于 NOA3S 算法在列有序条件下进行,那么根据理论和实验分析,平均情况下,由参数 K 、 N 决定的实际匹配搜索需求访问 K 个属性,且当 K 不

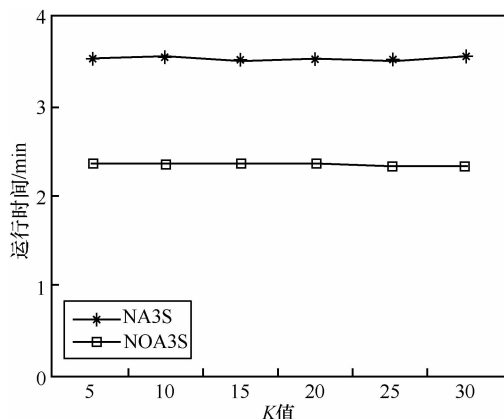


图 11 集中式下 K 值与时间的关系

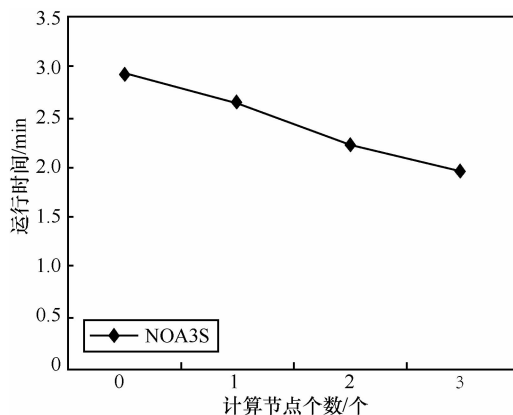


图 13 计算节点数与时间的关系

是很大时,参与计算的属性数目就更少,自然时间效率也有所改善,如图 12 所示。另外,图 12 中 NOA3S 在数据集 $50\,000 \times 128$ 测试下,耗时比较低,可能是因为数据维度上属性值的特殊性导致迭代次数减少所致。再者,根据定理 1 和 map/reduce 基础知识,实现的分布式方法不适合处理较小数据集,且随着数据规模(维度)的增大,迭代次数会增多,所以耗时越来越多,但在实验条件下,NOA3S_MR 的整体效果还是优于 NOA3S。

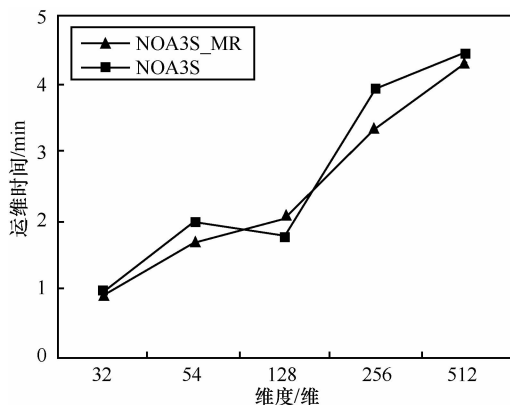


图 12 数据规模与时间的关系

最后,本文同样采用 $50\,000 \times 128$ 数据集对 NOA3S_MR 做了计算节点扩展性测试,图 13 依然能够表明随着计算节点数的增加,分布式算法在执行时,由 Hadoop 框架实现节点间多个任务调度,以并发完成大规模数据处理作业,因而如同流水线一般,能够很好地降低任务的运行成本。

5 结束语

自适应子空间相似性搜索作为一种特殊问题,在许多现实应用领域,尤其是图像检索领域,存在它的合理性,而且研究表明性能和效果往往比 k NN 查询好。一方面,本文也通过

理论分析和实验得到证实;另一方面,考虑到现实大数据环境下,单机算法不再适应需求,本文在完善和优化单机算法的基础上设计并实现了 Hadoop 框架下的并行算法。虽然目前并行算法实验效果已经比较理想,但考虑到 Hadoop 本身不是很适合处理迭代 map/reduce 作业,后期可能会考虑采用如 Spark 这样的内存计算框架来进一步改善算法性能。

参考文献

- 1 Lian X, Chen L. Subspace similarity search under Lp-Norm. *IEEE Transactions on Knowledge and Data Engineering*, 2012, 24(2):365~382
- 2 Hinneburg A, Aggarwal C, Keim D A. What is the nearest neighbor in high dimensional spaces. *Proceedings of the 26th VLDB Conference*, Cairo, Egypt, 2000: 506~515
- 3 张慧,郑吉平,韩秋廷. BTreeU-Topk:基于二叉树的不确定数据上的 Top-k 查询算法. *计算机研究与发展*, 2012(1): 2095~2105
Zhang H, Zheng J P, Han Q T. BTreeU-Topk: binary-tree based Top-k query algorithms on uncertain data. *Journal of Computer Research and Development*, 2012(1): 2095~2105
- 4 Shi Y, Graham B. Similarity search problem research on multi-dimensional data sets. *Proceedings of Tenth International Conference on Information Technology: New Generations (ITNG)*, Washington DC, USA, 2013:573~577
- 5 张彪,李川,徐洪宇等.基于特征子图的异构信息网络节点相似性度量. *电信科学*, 2014, 30(11):66~72
Zhang B, Li C, Xu H Y, et al. Heterogeneous information networks node similarity measurement based on feature sub-graph. *Telecommunications Science*, 2014, 30(11):66~72
- 6 Watanabe S, Sawada H, Minami Y, et al. Fast similarity search on a large speech data set with neighborhood graph indexing. *Proceedings of 2010 IEEE International Conference on Acoustics Speech and Signal Processing (ICASSP)*, Dallas, USA, 2010: 5358~5361



7 Marios,Yannis. R-trees: a dynamic index structure for spatial searching. Boston, MA, USA ,1984:993~1002

8 Kriegel H P, Kroger P, Schubert M, *et al.* Efficient query processing in arbitrary subspaces using vector approximations. Proceedings of the 18th International Conference on Scientific and Statistical Database Management, Washington DC, USA, 2006:184~190

9 Zhang D X, Agrawal D, Chen G, *et al.* HashFile : an efficient index structure for multimedia data. Proceedings of the IEEE 27th International Conference on Data Engineering (ICDE), Washington DC, USA, 2011:1103~1114

10 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

11 Lv Q, Josephson W, Wang Z, *et al.* Multi-probe LSH : efficient indexing for high-dimensional similarity search. Proceedings of the 33rd International Conference on Very Large Data Bases, Vienna, Austria, 2007:950~961

12 林悦. 基于散列算法的高维数据的最近邻检索 (硕士学位论文).浙江大学, 2013

Lin Y. Hashing based algorithms for nearest neighbor search in high dimensions (master dissertation). Zhejiang University, 2013

13 Zhang Z J, Ooi B C, Parthasarathy S, *et al.* Similarity search on bregman divergence: towards non-metric indexing. Proceedings of the VLDB Endowment, Springer, Geimany,2009:13~24

14 Thomas B, Subspace similarity search using the ideas of ranking and top- k retrieval. Proceedings of the IEEE 26th International Conference on Data Engineering Workshops (ICDEW), California, USA,2010:4~9

15 Thomas B. Subspace similarity search: efficient k -NN queries in arbitrary subspaces. Proceedings of Scientific and Statistical Database Management Springer Berlin Heidelberg,Germany, 2010:555~564

16 He X F. Incremental semi-supervised subspace learning for image retrieval. Proceedings of the 12th Annual ACM International Conference on Multimedia, New York, USA, 2004:2~8

17 Stupar A, Michel S, Schenkel R. Rankreduce-processing k -nearest neighbor queries on top of MapReduce. Proceedings of the 8th Workshop on Large-Scale Distributed Systems for Information Retrieval, Geneva, Switzerland, 2010:13~18

18 Zhang C, Li F F, Jestes J. Efficient parallel kNN joins for large data in MapReduce. Proceedings of the 15th International Conference on Extending Database Technology. New York, NY, USA, 2012:38~49

19 Kong W H, Wu J L, Guo M Y. Manhattan hashing for large-scale image retrieval. Proceedings of the 35th International ACMSIGIR Conference on Research and Development in Information Retrieval , Portland, Oregon, USA, 2012:45~54

[作者简介]



任建新,男,宁波大学信息科学与工程学院硕士生, 主要研究方向为大规模数据处理技术与信息检索。

陈华辉,男,博士,宁波大学信息科学与工程学院教授, 主要研究方向为数据库、数据流、数据挖掘、云计算。

2015190-12

2015 年 5 月通信业主要指标完成情况(一)

指标名称	单位	本年本月止累计到达	比上年同期累计(±%)	本月
营业收入	亿元	5 639.2	-1.8	1 168.0
其中:电信业务收入	亿元	4 737.5	2.8	991.3
固定资产投资完成额	亿元	910.5	2.4	242.0
固定本地电话通话时长合计	万分钟	9 541 848.2	-12.9	1 966 398.8
固定长途电话通话时长合计	万分钟	1 993 752.2	-9.6	418 758.8
移动电话去通话时长合计	万分钟	118 491 630.1	-2.4	24 751 745.7
其中:国内长途通话时长	万分钟	27 538 270.0	-3.8	5 885 683.0
国际长途通话时长	万分钟	47 934.9	-10.9	9 714.7
移动短信业务量	万条	30 168 457.6	-4.1	5 980 712.8
移动互联网接入流量	万吉字节	135 416.7	91.6	30 779.2

注:(1)电信业务收入增长率按可比口径计算。(2)固定长途电话通话时长和移动电话通话时长均包含相应的 IP 电话通话时长。(3)通话时长各项指标均为去话通话时长。

(来源:工业和信息化部运行监测协调局)