ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.20, No.7, July 2009, pp.1825−1838
doi: 10.3724/SP.J.1001.2009.03325

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

有效预处理 P2P 网络中的子空间 skyline 查询*

黄震华[1,2+], 王智慧[3], 郭建魁[3], 汪 卫[3], 施伯乐[3]

[1](同济大学 电子与信息工程学院,上海 201804)

[2](嵌入式系统与服务计算教育部重点实验室(同济大学),上海 200092)

[3](复旦大学 计算机与信息技术系,上海 200433)

Efficient Preprocessing of Subspace Skyline Queries in P2P Networks

HUANG Zhen-Hua[1,2+], WANG Zhi-Hui[3], GUO Jian-Kui[3], WANG Wei[3], SHI Bo-Le[3]

[1](School of Electronics and Information, Tongji University, Shanghai 201804, China)

[2](Key Laboratory of ESSC, Ministry of Education (Tongji University), Shanghai 200092, China)

[3](Department of Computing and Information Technology, Fudan University, Shanghai 200433, China)

+ Corresponding author: E-mail: zhh1980@fudan.edu.cn, http://www.fudan.edu.cn

Huang ZH, Wang ZH, Guo JK, Wang W, Shi BL. Efficient preprocessing of subspace skyline queries in P2P networks. *Journal of Software*, 2009,20(7):1825−1838. http://www.jos.org.cn/1000-9825/3325.htm

Abstract: Skyline query processing has recently received a lot of attention in database community. Lately, Akrivi Vlachou and D. Christos considered how to efficiently process subspace skyline queries in peer-to-peer networks, and proposed the concept of "extended skyline set" to reduce the volume of data transferred in the preprocessing phase for the first time. However, the experimental evaluation shows that this data structure is extremely limited in reducing the volume of data transferred in the preprocessing phase. Motivated by these facts, this paper proposes an efficient algorithm, i.e. TPAOSS (three-phase algorithm for optimizing skyline scalar), to reduce the volume of data transferred in the preprocessing phase. TPAOSS algorithm is based on the semantic relationship between full-space skylines and subspace skylines, and transfers the data through three phases. In the first phase, it only sends full-space skylines. In the second phase, it receives seed skylines. In the third phase, it exploits Bloom filter technology to obtain and send the replicated objects with seed skylines on the subspaces. Particularly, the paper presents two efficient strategies to reduce the volume of data transferred in the second phase. Furthermore, it presents detailed theoretical analyses and extensive experiments that demonstrate these algorithms are both efficient and effective.

Key words: subspace skyline query; Bloom filter; super-peer architecture; query optimization

摘 要: 多维空间的 skyline 查询处理是近年来数据库领域的一个研究重点和热点.Vlachou 等人首次考虑如何在 P2P 网络中有效进行子空间上的 skyline 查询,并提出"扩展 skyline 集合"的概念来减少预处理时的网络传输量.然而实验评估表明,扩展 skyline 集合只能有限地减少子空间 skyline 查询预处理的数据传输量.基于此,提出一种缩减预

* Supported by the National Natural Science Foundation of China under Grant No.60303008 (国家自然科学基金); the National Basic Research Program of China under Grant No.2005CB321905 (国家重点基础研究发展计划(973))

Received 2007-05-12; Accepted 2008-03-06

处理时数据传输量的有效方法 TPAOSS(three-phase algorithm for optimizing skyline scalar).TPAOSS 算法根据全空间 skyline 集合与子空间 skyline 集合间的语义关系分 3 个阶段来传输必要的数据,其中第 1 阶段发送全空间 skyline 对象;第 2 阶段接收种子 skyline 对象;而第 3 阶段基于 Bloom filter 技术发送种子 skyline 对象在子空间上的重复对象.为了降低第 2 阶段的数据传输量,给出两种接收种子 skyline 对象的有效策略.理论分析和实验评估结果表明,所给出的算法具有有效性和实用性.

关键词: 子空间 skyline 查询;Bloom filter;super-peer 体系结构;查询优化

中图法分类号: TP393 **文献标识码:** A

多维空间**的 skyline 查询处理技术是近年来数据库技术领域的一个研究重点和热点[1−8].这主要是因为 skyline 查询处理在许多领域有着广泛的应用,如多标准决策支持系统[1]、城市导航系统[2]、数据挖掘和可视化[3]以及用户偏好查询[4]等.给定有限规模的空间对象集合 $SO=\{O^1,O^2,\ldots,O^n\}$,其中每个空间对象 $O^i(i\in[0,n])$有 δ维属性,每维属性衡量它的一个子特征(比如距离、价格等).Skyline 查询就是在 SO 中找出一类空间对象集合 PSO,它满足如下条件:$PSO\subseteq SO$ 且 PSO 中每个元素对象不会在所有维上的取值均差于 SO 中的某一元素对象.显然,用户只需考虑属于 skyline 集合的那些对象,而不必关心那些被过滤掉的元素.这样,用户就可以在小规模的 skyline 结果集上对自己感兴趣的对象进行选择.

由于随着分布式网络的广泛使用,用户希望能够在分布式环境中有效处理不同子空间上的 skyline 查询.由于传统的 C/S(客户/服务器)模型的网络结构能够方便地升级为 SPA(super peer architecture)[9]架构的分布式网络,因此大多分布式网络均使用 SPA 架构.另一方面,随着 P2P(peer-to-peer)技术的深入应用,在 ICDE 2007 国际会议上,Vlachou 等人[10]开始考虑如何在 SPA 架构的 P2P 网络中有效地进行任一子空间上的 skyline 查询处理.在 SPA 结构的 P2P 网络中,核心是由几台处理能力较强的计算机(称为超点计算机)组成.Skyline 计算的工作量主要在这些计算机上完成.每台超点计算机关联若干台性能较差的计算机(称为简单计算机).大部分数据水平分割存储于这些简单计算机上.为了完成子空间上的 skyline 查询,我们要先把数据从各简单计算机传输到与之关联的超点计算机上,然后再进行子空间上的 skyline 计算.我们把将数据从简单计算机传输到与之关联的超点计算机的这个过程称为子空间 skyline 查询预处理.文献[10]的实验评估表明,在子空间 skyline 查询预处理阶段,如果将所有的数据从简单计算机传输到超点计算机,那么该阶段的网络传输时间将占子空间 skyline 查询总时间的 65%以上.因此,预处理阶段的数据传输开销是整个子空间 skyline 查询的性能瓶颈.基于此,文献[10]给出一个数据结构——“扩展 skyline 集合”来避免在预处理阶段传输所有的数据.同时,该数据结构又能够保证具有足够的信息来回答所有子空间上的 skyline 查询.然而我们发现,在子空间 skyline 查询预处理阶段,文献[10]仍存在如下两个不足:

- 扩展 skyline 集合虽然能够在一定程度上缩减预处理阶段的数据传输量,然而,它的剪枝能力有限.我们的实验评估表明,扩展 skyline 集合在大多数情况下只能缩减小于 15%的数据传输量;
- 当用户发出的子空间 skyline 查询个数较少时,扩展 skyline 集合中的大多数对象均是多余的,即它们不可能出现在任一用户查询的返回结果中,因此完全没有必要将这些对象从简单计算机传输到与之关联的超点计算机.

从上面的分析我们可以看出,虽然文献[10]注意到了预处理阶段是整个子空间 skyline 计算的瓶颈,然而,它仍然未能有效地解决该阶段数据传输量的问题.基于此,本文着重解决如下问题:如何在子空间 skyline 查询的预处理阶段降低简单计算机与超点计算机之间的数据传输量.为了有效解决该问题,本文完成如下工作:

- 基于全空间 skyline 集合与子空间 skyline 集合之间的语义关系,提出一种 3 阶段传输查询所需数据的有效方法 TPAOSS:在 TPAOSS 算法的第 1 阶段,我们只传送全空间 skyline 对象.在第 2 阶段,我们接收超点计算机回送的种子 skyline 对象信息;而在第 3 阶段,我们基于 Bloom filter 技术[11]来传送种子

** 我们把所有维度组成的空间称为全空间,把其中若干个维度组成的空间称为子空间,而把全空间和所有子空间统称为维空间.

skyline 对象[***]在子空间上的重复对象信息;

- 在 TPAOSS 算法的第 2 阶段,我们提出两种用来传输种子 skyline 对象的优化策略,即传送位置值策略及传送分组位置值策略.这两种策略均使用一维数值替代多维对象来进行传输,从而显著降低了网络传输量.特别地,传送分组位置值策略通过维空间的继承关系来进一步缩减网络的数据传输量;
- 从理论上分析和证明本文算法的正确性以及有效性.同时,在不同分布的数据集上实施大量的实验评估,并从多个不同的角度来综合考察本文算法的实用性和可扩展性.

本文第 1 节介绍与本文相关的研究工作.第 2 节具体描述本文的解决方法.第 3 节对本文提出的算法进行理论分析.第 4 节通过实验来评估本文工作的有效性和实用性.最后,第 5 节对本文的工作进行总结,并给出将来比较感兴趣的后续工作.

1 预备知识

在这一节中,我们简单介绍与本文工作相关的一些概念和技术.

1.1 子空间skyline查询计算

Pei 等人[4]利用 OLAP(on-line analytical processing)应用中的数据立方体格(data cube)来研究各维空间 skyline 集合之间的语义关系,并且给出一个可行的计算方法 Skyey 来同时产生全部 2^d–1(∅除外)个维空间上的 skyline 集合,其中,d 为全空间维度个数.显然,Skyey 算法不适合实际应用,因为通常情况下,用户只对部分维空间上的 skyline 集合感兴趣.

Tao 等人[3]着重研究不同维空间上的 skyline 查询,并首次给出一种计算任一维空间上 skyline 对象的方法 SUBSKY.SUBSKY 算法首先通过 k-mean 聚类算法将数据集划分为 m 个类,然后对于每个类 cl,以 cl 中的核心点为参照点进行计算任一维空间上的 skyline 对象.然而,SUBSKY 算法计算每一个维空间上的 skyline 对象时需要多次扫描数据集,而且对所有 m 个类中的对象均计算完成之后才能返回第 1 个 skyline 对象.因此,SUBSKY 算法的查询效率在多数情况下较低.

1.2 P2P网络中子空间skyline查询预处理

在 SPA 结构的 P2P 网络中,子空间 skyline 查询的基本过程如下[10]:用户在某一台超点计算机 SC(simple computer)上发出子空间 V 上的 skyline 查询 $skyQ(V)$;系统从 SC 开始沿着网络路径将该查询指令传输到各个超点计算机上;然后,每一台接收到查询指令的超点计算机从与之关联的各简单计算机上获取数据,接着计算出这些数据中的 skyline 对象,并将得到的所有 skyline 对象传输到 SC 上;最后,系统对 SC 得到的全部数据作进一步 skyline 计算,过滤掉那些具有假阳性(false positive)的对象,最终得到正确的子空间 V 上的 skyline 结果集.在 P2P 网络中,查询预处理阶段的数据传输量是整个子空间 skyline 查询的性能瓶颈.文献[10]的实验评估结果表明,在子空间 skyline 查询预处理阶段,如果将所有的数据从简单计算机传输到超点计算机,那么该阶段的网络传输时间将占子空间 skyline 查询总时间的 65%以上.基于此,文献[10]给出一个数据结构——“扩展 skyline 集合”来避免在预处理阶段传输所有的数据.同时,该数据结构又能够保证具有足够的信息来回答所有子空间上的 skyline 查询.

接下来,我们具体描述扩展 skyline 集合.不失一般性,我们假定维属性取值越小,那么在该维上的优越程度越高.

定义 1(支配关系)[1]. 对于 d 维空间的两个对象 $p(p.val_1,p.val_2,\ldots,p.val_d)$和 $r(r.val_1,r.val_2,\ldots,r.val_d)$,如果它们满足下列两个条件,那么我们称 p 支配 r:

① $\forall j\in[1,d],p.val_j\le r.val_j$;

② $\exists i\in[1,d],p.val_j<r.val_j$.

*** 子空间 V 上的种子 skyline 对象 p 具有如下特征:p 为全空间 skyline 对象,且 p 同时为子空间 V 上的 skyline 对象.

为了简单起见,我们把对象 p 支配对象 r,记为 $r \prec_d p$.

定义 2(skyline 集合)[1]. 设 AD 是 d 维对象全集,那么 d 维空间上 skyline 集合$\nabla^d(AD)$可表示为

$$\nabla^d(AD)=\{p|p\in AD\wedge\neg\exists r\in AD, p \prec_d r\}.$$

定义 3(扩展支配关系)[10]. 对于 d 维空间的两个对象 $p(p.val_1,p.val_2,\ldots,p.val_d)$和 $r(r.val_1,r.val_2,\ldots,r.val_d)$,如果它们满足条件$\forall j\in[1,d], p.val_j<r.val_j$,那么我们称 p 扩展支配 r,记为 $r \lhd_d p$.

比较定义 3 和定义 1 可知,如果 p 扩展支配 r,那么 p 必定支配 r;反之,不成立.

定义 4(扩展 skyline 集合)[10]. 设 AD 是 d 维对象全集,那么 d 维空间上扩展 skyline 集合 $\Delta^d(AD)$可表示为

$$\Delta^d(AD)=\{p|p\in AD\wedge\neg\exists r\in AD, p \lhd_d r\}.$$

比较定义 4 和定义 2 我们不难看出,d 维空间上 skyline 集合是 d 维空间上扩展 skyline 集合的子集,即

$$\nabla^d(AD)\subseteq \Delta^d(AD).$$

1.3 Bloom filter技术

Bloom filter 是 1970 年 Bloom 提出来的[11],广泛应用于 Web Cache 的共享.为了验证某一元素是否在特定的集合中,我们可以用 Bloom filter 来简单地表示这个集合.给定一个具有 n 个元素的集合 $S=\{s_1,s_2,\ldots,s_n\}$,我们用一个长度为 $m(m>n)$的位数组来描述 Bloom filter,初始化数组的每个元素的值为 0.值得注意的是,虽然 m 大于 n,但是集合 S 所占用的空间要远大于 m 个比特位.一个 Bloom filter 使用 t 个相互独立的哈希函数 $h_1,h_2,\ldots,h_t$,它们的值域为$\{0,1,\ldots,m-1\}$.对于每一个元素 $s\in S$,数组中对应于$h_1(s),h_2(s),\ldots,h_t(s)$的位被置成 1.图 1 显示了使用 4 个哈希函数的 Bloom filter.

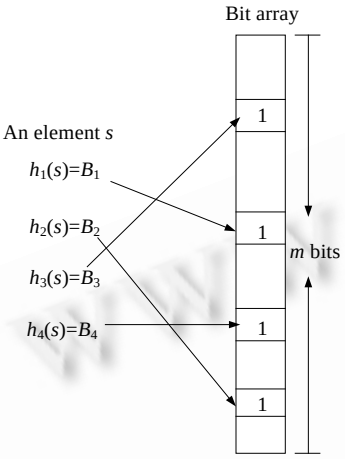


Fig.1 A Bloom Filter with 4 hash functions

图 1 使用 4 个哈希函数的 Bloom filter

当我们使用 Bloom filter 技术时,描述一个元素 $s\in S$ 就可以用它的哈希函数值 $h_1(s),h_2(s),\ldots,h_t(s)$在数组上对应位置的元素全为 1 来表示;如果某个存储单元为 0,则表示这个元素不在集合 S 上.不难看出,数组中的同一个位置可能被多次置为 1,因此,采用 Bloom filter 技术来查询一个元素是否在集合 S 上可能存在错误定位(false positive)****.然而在大多数现实应用中,这种错误定位的概率都比较小.例如,当 $m/n=10$ 并且 $k=8$ 时,错误定位的概率仅为 0.085%.特别地,文献[3]形式化给出错误定位率的大小为

$$P_{error}=(1-(1-1/m)^{k\cdot n})^k\approx(1-e^{k\cdot n/m})^k.$$

2 算法描述

预处理阶段的传输开销是影响子空间 skyline 查询效率的一个重要因素,文献[10]通过传输扩展 skyline 集合来降低预处理阶段的数据传输量.然而在多数情况下,扩展 skyline 集合仍然包含大量多余的数据对象,这些对象不可能出现在用户发出的子空间 skyline 查询的结果中.根据扩展 skyline 集合的定义可知,不在扩展 skyline 集合中的对象,必须在每个维度上均比扩展 skyline 集合中的某对象要差.显然,在实际应用中,对象在某些维度上会取得一定的优势.因此,扩展 skyline 集合的基数在多数情况下接近于原始数据集合.

由上面的分析我们可以看出,虽然文献[10]注意到了预处理阶段是整个子空间 skyline 查询的瓶颈,然而它仍然未能有效地解决该阶段数据传输量的问题.基于此,我们提出一种有效控制预处理阶段数据传输量的方法 TPAOSS.在给出 TPAOSS 算法之前,我们先给出全空间 skyline 集合与子空间 skyline 集合的关系,如定理 1 所示.

**** 错误定位指对于一个本不属于集合 S 的元素 w,它的 k 个哈希函数值却均为 1,即 $w\notin S\wedge\forall i\in[1,k], h_i(w)=1$.

定理 1(子空间 skyline 集合性质). 假定 AD 是 d 维对象全集,子空间 V 上的 skyline 查询涉及 $v(v\le d)$个维,那么以下关于全空间 skyline 集合$\nabla^d(AD)$与子空间 skyline 集合$\nabla^v(AD)$的等式成立:

$$\nabla^v(AD)\equiv\nabla^v(\nabla^d(AD))\cup\{p\mid p\in AD-\nabla^d(AD)\wedge\exists r\in\nabla^v(\nabla^d(AD)),\forall i\in V,r[i]=p[i]\}.$$

即子空间 V 上的 skyline 对象只由种子 skyline 对象以及在 V 上的取值与种子 skyline 对象均相同的非全空间 skyline 对象组成.

2.1 TPAOSS算法基本框架

假定全空间具有 d 个维,当用户发出 $w(w\le 2^d-1)$个子空间 skyline 查询命令 $skyQ(V_1),skyQ(V_2),\ldots,skyQ(V_w)$时,TPAOSS 算法基于定理 1,将查询预处理时的数据传输划分为 3 个阶段:(1) 在 TPAOSS 算法的第 1 阶段,我们只将全空间 skyline 集合 S_f 从简单计算机 SC 传输到与之关联的超点计算机 SP(super processor)上.由于 skyline 计算是 CPU 敏感的[5],因此,为了提高子空间 skyline 计算的效率,我们把从 S_f 获取 w 个种子 skyline 集合 $seed(V_1),seed(V_2),\ldots, seed(V_w)$的所有计算过程放于高计算性能的超点计算机 SP 上完成,并将这 w 个种子 skyline 集合所包含的所有对象回送给 SC;(2) 在 TPAOSS 算法的第 2 阶段,我们在 SC 上接收 SP 回送给它的所有种子 skyline 对象;(3) 在 TPAOSS 算法的第 3 阶段,当 SC 接收到所有种子 skyline 对象之后,我们获取并传送这些种子 skyline 对象在其相应子空间上的重复对象给 SP.TPAOSS 算法的基本框架如下所示:

算法 1. TPAOSS 算法.

/*分 3 个阶段传送子空间 skyline 查询所需的数据;d 维对象集合 AD,子空间 $V_1,V_2,\ldots,V_w$.*/

begin

[1] 产生全空间 skyline 集合 $S_f=\nabla^d(AD)$;

[2] 将 S_f 传送给相关联的超点计算机 SP;

[3] 等待 SP 回送所有种子 skyline 对象;

[4] 运行 ORVBF 算法获取 w 个种子 skyline 集合所对应的重复值对象列表 $L_{rep}^{(1)},L_{rep}^{(2)},\ldots,L_{rep}^{(w)}$;

[5] **for** i=1 to w **do**

[6] 将 $L_{rep}^{(i)}$ 传送给相关联的超点计算机 SP;

[7] **end for**

end

接下来,我们对 TPAOSS 算法的主要步骤进行分析:

(1) 在 TPAOSS 算法的第[2]步中,为了能够降低数据传输量,当 SP 回送种子 skyline 对象时,我们不直接传送每个种子 skyline 对象的具体数据值,而是传送该对象在$\nabla^d(AD)$中的位置值.因此在 SC 上,我们可以根据对象的位置值来获取它的完整数据信息.同时,为了提高获取完整数据信息的效率,我们在 SP 上对所有种子 skyline 对象的位置值从小到大进行排序.在第 2.2 节中,我们将具体给出 3 种有效的基于该传送策略的对象位置值组织方法.

(2) 在 TPAOSS 算法的第[3]步中,我们基于 Bloom filter 技术来获取与每个子空间种子 skyline 集合相对应的重复对象.同样,为了能够降低数据传输量,我们不直接传送每个重复对象的具体数据值,而是传送与该重复对象相对应的种子 skyline 对象在$\nabla^d(AD)$中的位置值.在第 2.3 节中,我们将给出实现这一步骤的 ORVBF 算法.

2.2 接收种子skyline对象

为了获取种子 skyline 对象在子空间上的重复对象,我们必须将超点计算机 SP 上的 w 个种子 skyline 集合 $seed(V_1),seed(V_2),\ldots,seed(V_w)$回送给简单计算机 SC.然而,如果我们直接传送这些集合中对象的实际数据,那么由于每个对象可能包括多个维属性值,因此直接传送数据对象的方式将带来巨大的数据传送量.为了能够降低数据传输量而不缺少种子 skyline 对象的信息,我们采用传送位置值策略.该策略的基本思想是:假设第 $i(1\le i\le w)$个种子集合中有 ξ 个对象,即 $seed(V_i)=\{p_1,p_2,\ldots,p_\xi\}$.那么我们构造如下映射 $\wp:p_t\to pos_t$,其中,$p_t\in seed(V_i)$,而 $pos_t\in N^+$,为 p_t 在全空间 skyline 集合 S_f 的位置值.当对所有 ξ 个对象映射完毕之后,我们将所有的位置值递增排序,

最终得到有序的位置值列表 $plist^{(i)}$.因此对每个种子集合 $seed(V_i)$,我们用有序的位置值列表 $plist^{(i)}$来代替实际的对象数据值,并将 w 个种子集合所对应的位置值列表 $plist^{(1)},plist^{(2)},\ldots,plist^{(w)}$回送给简单计算机 SC.

由于一个全空间 skyline 对象可能出现在多个子空间种子 skyline 集合中,因此,回送的 w 个位置列表中可能包含多个重复的位置值,从而增加了网络传输的数据量.为了降低从 SP 回送给 SC 的 w 个位置值列表的重复性,我们对传送位置值策略加以改进,产生一种更为有效的传送策略,即传送分组位置值策略.

定义 5(维空间树). 维空间树 $T_{dim}=\langle ND,ES\rangle$,$T_{dim}$ 的每个节点对应一个维空间.如果对于任意两个节点 $node_i$ 和 $node_j$ 满足如下条件,我们称 $node_i$ 是 $node_j$ 的父节点,$node_i$ 所对应的维空间 DS_i 是 $node_j$ 所对应的维空间 DS_j 的父空间,并且存在一条有向边从 $node_i$ 指向 $node_j$:

① $DS_i \supset DS_j$;

② $\neg\exists DS' \in ND, DS_i \supset DS' \wedge DS' \supset DS_j$;

③ $DS_i = MostLeftParent(\{DS | node(DS) \in ND \wedge DS \supset DS_j\})$.

其中,$node(DS)$为维空间 V 在 T_{dim} 中所对应的节点,函数 $MostLeftParent$ 为取 DS_j 在 T_{dim} 中最左边的父空间.

注意,父空间和子空间的概念是相对的,同一个维空间 DS 相对于维空间 DS_1 来说是父空间,而相对于另一个维空间 DS_2 来说则可能是子空间.然而,当涉及到全空间 F 时,所有其他的维空间均是子空间.

定义 6(最大维空间树). 假定维空间序列 $ODS=\langle V_1,V_2,\ldots,V_w\rangle$满足$\forall V_i,V_j \in ODS, position(V_i)<position(V_j) \Rightarrow |V_i| \geq |V_j|$,如果维空间树 $T_{dim}=\langle ND,ES\rangle$满足如下条件,我们称它是一棵以 V'为根的 ODS 上的最大维空间树:

① $V' \in ODS$;

② $\forall nd \in ND, dimSpace(nd) \in ODS \wedge position(dimSpace(nd)) \geq position(V')$;

③ $\forall V \in ODS, V \subset V' \Rightarrow node(V) \in ND$.

其中,$dimSpace(nd)$为节点 nd 所对应的维空间,$node(V)$为维空间 V 在 T_{dim} 中所对应的节点,$position(V)$为维空间 V 在 ODS 中的位置.

传送分组位置值策略基于如下定理和推论来进行有效实施.

定理 2(父空间与子空间位置值列表的关系). 假定维空间树 T_{dim} 中父空间 V_p 对应的位置值列表为 $plist(V_p)$,而子空间 V_c 对应的位置值列表为 $plist(V_c)$,那么 $plist(V_c)=plist_1 \cup plist_2$,满足如下条件:

① $plist_1 \subseteq plist(V_p)$;

② $\neg\exists pos \in plist_2, pos \in plist(V_p)$;

③ $\forall pos \in plist_2, \exists pos' \in plist_1, \forall i \in V_c, obj_{pos}[i]=obj_{pos'}[i]$.

其中,obj_{pos} 和 $obj_{pos'}$分别表示位置值为 pos 和 pos'的对象.

根据定理 2,我们可以得出如下推论:

推论 1(重构父空间与子空间位置值列表). 假定维空间树 T_{dim} 中父空间 V_p 对应的位置值列表为 $plist(V_p)$,并且 V_p 具有 m 个子空间 $V_c^{(1)},V_c^{(2)},\ldots,V_c^{(m)}$,它们对应的位置值列表分别为 $plist(V_c^{(1)}), plist(V_c^{(2)}),\ldots, plist(V_c^{(m)})$,我们如下重构父空间位置值列表以及 m 个子空间位置值列表:

① $\forall i \in [1,m], aplist(V_c^{(i)})=plist(V_c^{(i)}) \wedge plist(V_p)$;

② $aplist(V_p)=plist(V_p)-\bigcup_{i=1}^{m} aplist(V_c^{(i)})$.

那么当简单计算机 SC 接收到这些重构的位置值列表之后,我们有足够的信息获取父空间 V_p 和 m 个子空间 $V_c^{(1)},V_c^{(2)},\ldots,V_c^{(m)}$ 上完整的重复对象.

从推论 1 我们不难看出,重构前的父空间位置值列表和 m 个子空间位置值列表中包含大量重复的位置值,而重构后的列表在很大程度上降低了位置值的重复性.同时我们可以得到

$$\left|aplist(V_p) \cup \bigcup_{i=1}^{m} aplist(V_c^{(i)})\right| = |plist(V_p)| < \left|plist(V_p) \cup \bigcup_{i=1}^{m} aplist(V_c^{(i)})\right|.$$

基于定理 2 和推论 1,传送分组位置值策略的基本思想可描述如下.我们首先对 w 子空间 $V_1,V_2,\ldots,V_w$ 进行分组,每一组对应一棵维空间树,分组过程如下:

(1) 对 w 个子空间按所包含的维度个数从大到小排序,得到子空间序列 $ODS=\langle V'_1,V'_2,...,V'_w\rangle$.不难看出,

$$\forall V'_i,V'_j\in ODS,position(V'_i)<position(V'_j)\Rightarrow|V'_i|\geq|V'_j|.$$

(2) 循环如下过程直到 ODS 为空:(I) 获取 ODS 中的第 1 个维空间 V';(II) 以 V'为根构造 ODS 上的最大维空间树 T_{dim};(III) 从 ODS 中删除 T_{dim} 的所有节点对应的维空间.

当在 SP 上获得 w 个种子 skyline 集合所对应的位置值列表之后,对产生的每一个维空间树 T_{dim},我们从叶子节点层开始向根节点遍历,当访问到 T_{dim} 的节点 nd 时,我们删除与 nd 相对应的位置值列表 $plist(nd)$中满足如下条件的每一个位置值 pos:

$$pos\notin plist(parent(nd))\vee pos\in\bigcup_{cd\in Child(nd)}plist(cd),$$

其中,$parent(nd)$为节点 nd 在 T_{dim} 中的父节点,$Child(cd)$为节点 cd 在 T_{dim} 中的所有子节点.最后,我们将重构后的 w 个位置值列表 $aplist^{(1)},aplist^{(2)},\ldots,aplist^{(w)}$中所有非空的列表回送给简单计算机 SC.

2.3 ORVBF算法描述

当简单计算机 SC 接收到各子空间上的种子 skyline 对象位置值之后,我们基于 Bloom filter 技术给出一个在集合 $AD-\nabla^d(AD)$上查找所有重复对象的有效方法 ORVBF(obtain replicated values based on Bloom filter). ORVBF 算法的基本思想可描述如下:对用户查询的每个子空间 $V_i(1\leq i\leq w)$,我们为 $\Delta^d(AD)-\nabla^d(AD)$指定一个长度为 $8\cdot|AD-\nabla^d(AD)|$的一维数组 PA_i 作为 Bloom filter,数组的每个分量关联一个特征项链表,对象 p 的特征项用 $\langle BF(p),\hat{}p\rangle$来表示.其中,$BF(p)=\sum_{t\in Vi}\ln(p[t]+1)$[*****],$\hat{}p$ 为指向 p 的指针.特别地,我们用特征项$\langle -1,NIL\rangle$来标识数组分量是否为空.其中,NIL 为空指针,我们称该特征项为标识特征项.初始化时,PA_i 的各分量均不包含任何特征项.对于 $AD-\nabla^d(AD)$中的每个对象 p:我们首先使用第 1 个哈希函数 h_1 计算 p 的哈希值 $h_1(BF(p))$,如果数组 PA_i 的第 $h_1(BF(p))$分量为空,那么将标识特征项$\langle -1,NIL\rangle$添加进与 $PA_i[h_1(BF(p))]$所关联的链表中;然后,我们使用第 2 个哈希函数 h_2 计算 p 的哈希值 $h_2(BF(p))$,并在 $PA_i[h_2(BF(p))]$所关联的链表添加一个特征项$\langle BF(p),\hat{}p\rangle$.当创建完毕与 V_i 相应的 Bloom filter 之后,我们根据接收到的位置值,在 Bloom filter 中查找所有与这些位置值相对应的种子 skyline 对象在 V_i 上的重复对象.由第 2.2 节可知,SP 回送种子 skyline 对象位置值的策略不同,SC 接收到的位置值列表内容就会不同:

- 当使用传送位置值策略时,SC 接收到 w 个位置值列表 $plist^{(1)},plist^{(2)},\ldots,plist^{(w)}$.对于其中的每个列表 $plist^{(i)}(1\leq i\leq w)$,$plist^{(i)}$正好包含子空间 V_i 上的所有种子 skyline 对象的位置值.
- 当传送分组位置值策略时,SC 接收到 w 个重构后的位置值列表 $aplist^{(1)},aplist^{(2)},\ldots,aplist^{(w)}$.对于其中的每个列表 $aplist^{(i)}(1\leq i\leq w)$,$aplist^{(i)}$包含维空间 V_i 上的部分种子 skyline 对象的位置值,而那些出现在 V_i 各子空间上的位置值或者不出现在 V_i 父空间上的位置值将不包含在 $aplist^{(i)}$中.值得注意的是,由定理 2 我们可知,不出现在父空间上的位置值将不会影响获取 V_i 上重复对象的完整度.

不难看出,在使用传送位置值策略时,$plist^{(i)}$具有足够的信息来获取完整的子空间 V_i 上的重复对象;而在使用传送分组位置值策略时,$aplist^{(i)}$不具有足够的信息来获取完整的子空间 V_i 上的重复对象.因此,当创建完毕与 V_i 相应的 Bloom filter 之后,我们给出获取完整的子空间 V_i 上的重复对象的两种情况:

(1) 当使用传送位置值策略时,我们对 $plist^{(i)}$中的每个位置值 pos 进行处理:在$\nabla^d(AD)$中找出该位置上的对象 sd,并求出 sd 的 $BF(sd)$值;接着,我们首先使用第 1 个哈希函数 h_1 计算 sd 的哈希值 $h_1(BF(sd))$:如果 $PA_i[h_1(BF(sd))]$为空,说明在子空间 V_i 上不存在与 sd 取值均相同的对象;如果 $h_1(BF(sd))$不为空,那么使用第 2 个哈希函数 h_2 计算 sd 的哈希值 $h_2(BF(sd))$.同样,如果 $PA_i[h_2(BF(sd))]$为空,说明在子空间 V_i 上不存在与 sd 取值均相同的对象;否则在 $PA_i[h_2(BF(sd))]$所关联的特征项链表中查找在子空间 V_i 上与 sd 取值均相同的对象,若存在对象 r'在子空间 V_i 上与 sd 取值均相同,则 r'为 V_i 上的 skyline 对象.同时,为了避免其他种子 skyline 对象重复定位 r',我们将对象 r'的特征项从特征项链表中删除.当处理完 L 中的所有位置值之后,将 Bloom filter 以及所有特

[*****] 通常,对于两个对象 p 和 r,如果 $BF(p)\neq BF(r)$,那么 p 与 r 在 V_i 上不可能为重复对象,从而减少了 Bloom filter 的错误定位概率.

征项从内存中删除.

(2) 当使用传送分组位置值策略时,我们从叶子节点层中的维空间开始往根节点创建 Bloom Filter.因为叶子节点层中的每一个维空间 V_{ld} 都不具有子空间,所以 $aplist(V_{ld})$具有足够的信息来获取所有 V_{ld} 上的重复对象,此时,我们使用与情况(1)中相同的查找重复对象的方法来处理 $aplist(V_{ld})$中的每个位置值.然后,对于每个非叶子层的维空间 V_{int},我们假定维空间 V_{int} 具有 m 个子空间 $V_c^{(1)},V_c^{(2)},...,V_c^{(m)}$,它们对应重构后的位置值列表分别为 $aplist(V_c^{(1)}),aplist(V_c^{(2)}),...,aplist(V_c^{(m)})$,那么由于 $aplist(V_{int})\cup\bigcup_{i=1}^{m}aplist(V_c^{(i)})$ 具有足够的信息来获取完整的 V_{int} 上的重复对象,因此,我们同样使用与情况(1)中相同的查找重复对象的方法来处理 $aplist(V_{int})\cup\bigcup_{i=1}^{m}aplist(V_c^{(i)})$ 中的每个位置值.然而不难看出,对于任意两个对象,它们只有在子空间上具有重复性,才有可能在父空间上具有重复性.基于此,为了提高获取 V_{int} 上重复对象的效率,对于 V_{int} 的每个子空间 $V_c^{(i)}$,如果 Bloom filter 中的对象 p 与 $aplist(V_c^{(i)})$中的对象 r 在 $V_c^{(i)}$ 上具有重复性,即$\forall t\in V_c^{(i)},p[t]=r[t]$,那么我们继续比较 p 与 r 在 $V_{int}-V_c^{(i)}$ 上的取值,如果$\forall u\in V_{int}-V_c^{(i)},p[u]=r[u]$,那么说明 p 与 r 在 V_{int} 上具有重复性,所以我们可以提取将 p 放入与维空间 V_{int} 相对应的重复对象列表 L_{rep} 中.因此,当访问到维空间 V_{int} 时,我们只需对 $aplist(V_{int})$中的位置值进行处理,并将根据 $aplist(V_{int})$来获取的所有重复对象添加进 L_{rep} 中即可.

另一方面,为了缩减传送的重复对象的数据量,当我们将重复对象从简单计算机 SC 传送给超点计算机 SP 时,同样使用传送对象位置值的策略.同时,每个被传送的对象位置值关联一个计数器,表示该对象重复的次数.

ORVBF 算法的伪代码如下所示:

算法 2. ORVBF 算法.

输入:w 个维空间 $V^{(1)},\ldots,V^{(w)}$,以及它们的有序位置值列表 $list^{(1)},\ldots,list^{(w)}$;当使用传送位置值策略时,

- $\forall i\in[1,w],list^{(i)}=plist^{(i)}$;当使用传送分组位置值策略时,$\forall i\in[1,w],list^{(i)}=aplist^{(i)}$;
- $\wp$棵维空间树 $T_{dim}^{(1)},...,T_{dim}^{(\wp)}$;
- 两个数据集合 $S_f=\nabla^d(AD)$和 $S_e=AD-\nabla^d(AD)$;

输出:w 个维空间的重复对象列表 $L_{rep}^{(1)},L_{rep}^{(2)},...,L_{rep}^{(w)}$.

方法:

begin

[1] **for** i=1 to w **do** $L_{rep}^{(i)}\leftarrow\varnothing$; **end for**

[2] **for** i=1 to $\wp$ **do**

[3] **for** 从叶子节点层开始访问 $T_{dim}^{(i)}$ 的每个维空间 $V^{(u)}$ **do**

[4] 为 S_e 指定一个长度为 $8\cdot|S_e|$的一维数组 PA,每个分量关联一个特征项链表,初始化各分量为空;

[5] **for** S_e 中的每个对象 p **do**

[6] 计算 $h_1(BF(p))$和 $h_2(BF(p))$;

[7] **if** $PA[h_1(BF(p))]$为空 **then** 将$\langle-1,NIL\rangle$插入与 $PA[h_1(BF(p))]$关联的链表中; **end if**

[8] **if** $PA[h_2(BF(p))]$只包含特征项$\langle-1,NIL\rangle$ **then**

[9] 从 $PA[h_1(BF(p))]$关联的链表中删除$\langle-1,NIL\rangle$;

[10] **end if**

[11] 将$\langle BF(p),\hat{}p\rangle$插入与 $PA[h_2(BF(p))]$关联的链表中;

[12] **end for**

[13] **for** $list^{(u)}$中的每个位置值ξ **do**

[14] 在 S_f 中获取位置为ξ的对象 sd;

[15] 计算 $h_1(BF(sd))$和 $h_2(BF(sd))$;

[16] **if** $PA[h_1(BF(sd))]$和 $PA[h_2(BF(sd))]$均非空 **then**

[17] $count_1\leftarrow0$; $count_2\leftarrow0$;

[18] **for** $PA[h_2(BF(sd))]$相关联链表的每个特征项$\langle BF(r),\hat{}r\rangle$ **do**
[19] **if** $BF(sd)=BF(r)$,并且 r 与 sd 在维空间 $V^{(u)}$上的取值均相等 **then**
[20] $L_{rep}^{(u)} \leftarrow L_{rep}^{(u)} \cup \{\langle \xi, count_1{+}{+}\rangle\}$;
[21] 从 $PA[h_2(BF(sd))]$关联的链表中删除$\langle BF(r),\hat{}r\rangle$;
[22] **if** 使用传送分组位置值策略 **then**
[23] 获取 $V^{(u)}$在 $T_{dim}^{(i)}$ 中的父空间 $V^{(p)}$;
[24] **if** $\xi \notin L_{rep}^{(p)}$,并且 r 与 sd 在维空间 $V^{(p)}-V^{(u)}$上的取值均相等 **then**
[25] $L_{rep}^{(p)} \leftarrow L_{rep}^{(p)} \cup \{\langle \xi, count_2{+}{+}\rangle\}$;
[26] **end if**
[27] **end if**
[28] **end if**
[29] **end for**
[30] **end if**
[31] **end for**
[32] 删除数组 PA 以及所有的特征项链表;
[33] **end for**
[34] **end for**
[35] 将 $L_{rep}^{(1)}, L_{rep}^{(2)}, ..., L_{rep}^{(w)}$ 传送给相关联的超点计算机 SP;
end

3 算法理论分析

在这一节中,我们主要对本文给出算法的正确性、时间开销以及数据传输量进行理论上的分析.

引理 1(ORVBF 算法的正确性). 假定 AD 是 d 维对象全集.对于 w 个维空间 $V_1,V_2,\ldots,V_w$,ORVBF 算法能够根据 SC 所接收的 w 个位置值列表 $list^{(1)},list^{(2)},\ldots,list^{(w)}$获取这 w 个维空间上完整的重复对象集合.

基于引理 1 我们不难得出,在子空间 skyline 查询预处理时,本文给出的 3 阶段传输必要数据的算法 TPAOSS 的正确性,如定理 3 所示.

定理 3(TPAOSS 算法的正确性). 假定 AD 是 d 维对象全集.当用户发出 w 个维空间 skyline 查询 $skyQ(V_1)$, $skyQ(V_2),\ldots,skyQ(V_w)$时,在预处理阶段,TPAOSS 算法能够将正确的数据传输到超点计算机上.

定义 7(最大覆盖子集). 假定 AD 是 d 维对象全集,如果集合 RD 满足如下条件,我们称 RD 是 AD 的最大覆盖子集:

① $RD \subset AD$;

② $\exists p \in RD, \forall r \in AD-RD, \max_{t=1}^{d} p[t] < \min_{t=1}^{d} r[t]$;

③ $\neg\exists e \in RD, \exists p \in RD, \max_{t=1}^{d} p[t] < \min_{t=1}^{d} e[t]$.

定理 4(ORVBF 算法时间开销). 假定 AD 是 k 维对象全集,SF 和 SE 分别为 AD 上的全空间 skyline 集合以及扩展 skyline 集合,用户发出查询的 w 个维空间为 $V_1,V_2,\ldots,V_w$,那么,

(I) 如果使用传送位置值策略,并且对于 $V_i(1\le i\le w)$,假定数据在其上分布的概率密度函数和联合分布函数分别为 $f(V_i)$和 $F(V_i)$,则 ORVBF 算法的时间开销 C_F 可表示为

$$O(w\cdot(|SE|-|SF|)+|SF|\cdot\sum_{i=1}^{w}\int_{[0,1]^{|V_i|}} f(V_i)\cdot(1-F(V_i))^{|SF|-1}dV_i).$$

(II) 如果使用传送分组位置值策略,假定 $\wp$ 棵维空间树为 $T_{dim}^{(1)},\ldots,T_{dim}^{(\wp)}$,对于 $T_{dim}^{(i)}(1\le i\le \wp)$,它包含 LT_i 个叶子层维空间,并且数据在其根节点维空间 DS_i 上的概率密度函数和联合分布函数分别为 $f(DS_i)$和 $F(DS_i)$,则 ORVBF

算法的时间开销 C_F 可表示为

$$O(w\cdot(|SE|-|SF|)+|SF|\cdot\sum_{i=1}^{\wp}LT_i\int_{[0,1]^{|DS_i|}}f(DS_i)\cdot(1-F(DS_i))^{|SF|-1}dDS_i).$$

定义 8(维空间的重构度). 假定维空间 V 在维空间树 T_{dim} 中的父空间以及 u 个子空间分别为 $V_p, V_c^{(1)},...,V_c^{(u)}$,则维空间 V 在 T_{dim} 中的重构度 $RC(V)$定义为

$$RC(V)=\frac{|plist(V)\cap plist(V_p)-\bigcup_{i=1}^{u}V_c^{(i)}|}{|plist(V)|}=\frac{|aplist(V)|}{|plist(V)|}.$$

不难看出,维空间 V 在 T_{dim} 中的重构度 $RC(V)$等于重构后的位置值列表长度与重构前的位置值列表长度的比值.注意,当使用传送位置值策略时,每个维空间 V 的重构度 $RC(V)$均等于 1.

定理 5(TPAOSS 算法的数据传输量). 假定 AD 是 k 维对象全集,SF 为全空间 V_0 上的 skyline 集合,用户发出查询的 w 个维空间为 $V_1,V_2,\ldots,V_w$.对于 $V_i(0\le i\le w)$,假定数据在 V_i 上分布的概率密度函数和联合分布函数分别为 $f(V_i)$和 $F(V_i)$,重复率为ρ_i;对象的每一维占用χ个字节,对象的位置值占用ψ个字节,计数器长度为ζ个字节,则 TPAOSS 算法的数据传输量可表示为(字节):

$$VOL_{fs}+VOL_{sd}+VOL_{rp},$$

其中,$VOL_{fs}=|SF|\cdot k\cdot\chi$.如果使用传送位置值策略,则 $VOL_{sd}=\psi\cdot|SF|\cdot\sum_{i=1}^{w}\int_{[0,1]^{|V_i|}}f(V_i)\cdot(1-F(V_i))^{|SF|-1}dV_i$;如果使用传送分组位置值策略,假定 $\wp$棵维空间树为 $T_{dim}^{(1)},...,T_{dim}^{(\xi)}$,并且对于 $T_{dim}^{(i)}$,假定它含有ϕ_i 个维空间,它上面维空间 V_j $(1\le j\le\phi_i)$的重构度为 $RC(V_j)$,则

$$VOL_{sd}=\psi\cdot|SF|\cdot\sum_{i=1}^{\wp}\sum_{j=1}^{\phi_i}RC(V_j)\cdot\int_{[0,1]^{|V_j|}}f(V_j)\cdot(1-F(V_j))^{|SF|-1}dV_j,$$

$$VOL_{rp}=|SF|\sum_{i=1}^{w}\rho_i\cdot(\psi+\zeta)\int_{[0,1]^{|V_i|}}f(V_i)\cdot(1-F(V_i))^{|SF|-1}dV_i,$$

$$|SF|=|AD|\cdot\int_{[0,1]^{|V_0|}}f(V_0)\cdot(1-F(V_0))^{|AD|-1}dV_0.$$

4 实验评估

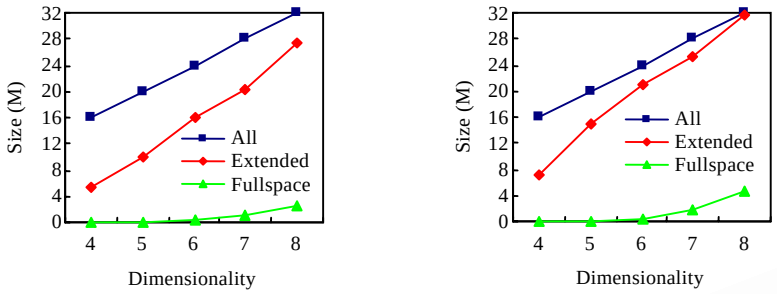
在这一节中,我们通过实验来评估本文所给方法和技术的有效性.实验评估环境为:PIII 1.60G CPU、512M 内存、60G 硬盘;Windows XP 操作系统.所有实验代码在 Java 编译器中运行通过.我们使用两类数据集,即独立分布数据集和反相关分布数据集[1].为了简单而不失一般性,对象的所有维属性、对象位置值以及重复值计数器均为整型,长度为 4Bytes.同时,我们假定维属性取值越小,那么在该维上的优越程度就越高.

4.1 3种数据结构的数据量评估

在这一节中,我们评估原始数据集、扩展 skyline 集合以及全空间 skyline 集合这 3 种数据结构的数据量大小.数据集分布采用独立分布和反相关分布两种类型.这组实验包括:① 当数据对象个数固定时,评估这 3 种数据结构的数据量随维度个数的变化情况.其中,数据对象个数固定为 1×10^6,而维度个数从 4~8 之间变化;② 当维度个数固定时,评估这 3 种数据结构的数据量随数据对象个数的变化情况.其中,维度个数固定为 8 维,而数据对象个数从 4×10^5 到 2×10^6 之间变化.实验结果如图 2 和图 3 所示.注意,在图中,原始数据集、扩展 skyline 集合以及全空间 skyline 集合分别用符号 All,Extended 和 Fullspace 来表示.

从图 2 和图 3 我们可以看出,当数据独立分布时,原始数据集中的扩展 skyline 对象和全空间 skyline 对象较少;而当数据反相关分布时,原始数据集中的扩展 skyline 对象和全空间 skyline 对象较多.这是因为,当数据反相关分布时,较多的对象具有如下性质:某些维度上取值较优,而在另外一些维度上取值较差.并且从图中我们发现,文献[10]使用扩展 skyline 集合来降低预处理阶段的数据传输量的效果较差.因为随着对象维度个数以及对象总个数的增大,扩展 skyline 集合的数据量趋近于原始数据集.特别地,当数据反相关分布时,扩展 skyline 集合

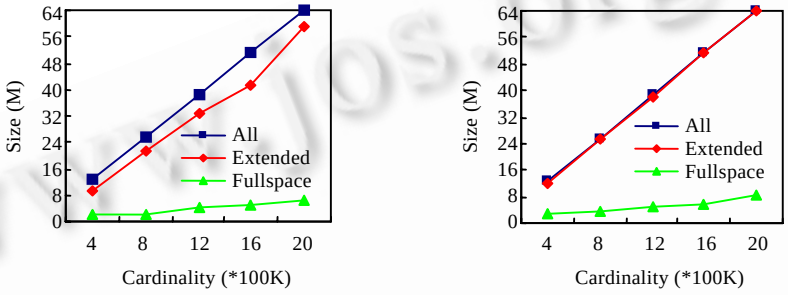
的数据量几乎等于原始数据集的数据量.例如在图2(b)中,当维度为8时,扩展skyline集合的数据量等于31.75M,而原始数据集的数据量为 32M.同样地,在图 3(b)中,当对象个数为 2×10^6 时,扩展 skyline 集合的数据量等于63.94M,而原始数据集的数据量为 64M.



(a) Independent dataset (a) 独立分布 (b) Anticorrelated dataset (b) 反相关分布

Fig.2 Fixed number of objects

图 2 对象个数固定



(a) Independent dataset (a) 独立分布 (b) Anticorrelated dataset (b) 反相关分布

Fig.3 Fixed dimensionality of objects

图 3 对象的维度个数固定

4.2 两种传送策略的位置值个数评估

在这一节中,我们评估当分别使用传送位置值策略和传送分组位置值策略时,简单计算机SC接收到的位置值个数.数据集分布仍然采用独立分布和反相关分布两种类型,全空间维数固定为 8,原始数据集的对象个数取 1.6×10^6.我们产生 60 个子空间上的 skyline 查询,由于传送分组位置值策略涉及到维空间树的个数,因此我们分4种情况来实施本节的实验评估:(1) 产生 60 棵维空间树,每棵维空间树只含有一个维空间,记为 $Q1\_60$; (2) 产生 15 棵维空间树,每棵维空间树含有 4 个维空间,记为 $Q4\_15$;(3) 产生 4 棵维空间树,每棵维空间树含有 15 个维空间,记为 $Q15\_4$;(4) 产生 1 棵包含 60 个维空间的维空间树,记为 $Q60\_1$.表 1 分别给出在这 4 种情况下,不同维数的子空间的个数,实验结果如图 4 和图 5 所示.注意,在图中,传送位置值策略和传送分组位置值策略分别用符号 SPos 和 GPos 来表示.

从图 4 和图 5 我们可以看出,在情况(1)中,即每棵维空间树只含有一个维空间的情况,这两种传送策略所得到的位置值个数相等.这是因为,当使用传送分组位置值策略时,每个维空间对应的位置值列表在重构前后所包含的位置值个数相同.此时,传送分组位置值策略退化为传送位置值策略.然而,在其余的 3 种情况中,使用传送分组位置值策略所得到的位置值个数显著少于使用传送位置值策略所得到的位置值个数.例如在图 5 中,当原始数据集的对象总数为 1.6×10^6 时,使用这两种不同传送策略所得到的位置值个数分别为:(1) 当维空间树为

$Q4\_15$ 时,使用传送位置值策略的位置值个数是 87 115,而使用传送分组位置值策略所得到的位置值个数是 50 918; (2) 当维空间树为 $Q15\_4$ 时,使用传送位置值策略的位置值个数是 54 878,而使用传送分组位置值策略所得到的位置值个数是 20 148;(3) 当维空间树为 $Q60\_1$ 时,使用传送位置值策略的位置值个数是 57 477,而使用传送分组位置值策略所得到的位置值个数是 16 036.

Table 1 Number of subspaces with difference dimensionality

表 1 不同维度的子空间个数分布

Dim.	$Q1\_60$	$Q4\_15$	$Q15\_4$	$Q60\_1$
7	0	0	0	1
6	0	15	4	5
5	0	18	13	11
4	60	23	16	21
3	0	4	17	13
2	0	0	10	9

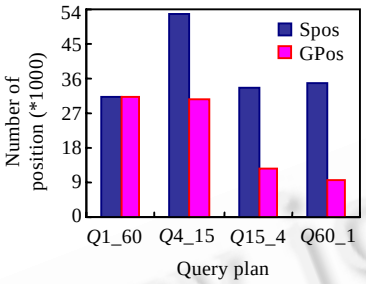


Fig.4 Independent dataset

图 4 数据集独立分布

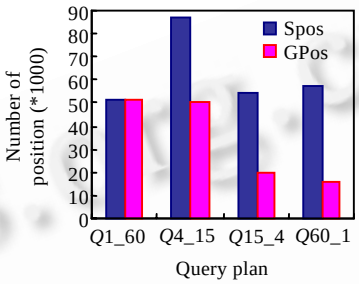
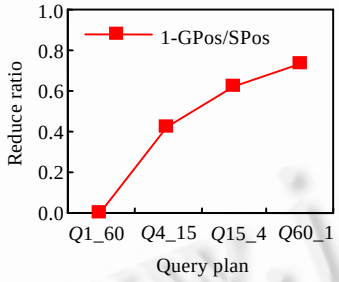


Fig.5 Anticorrelated dataset

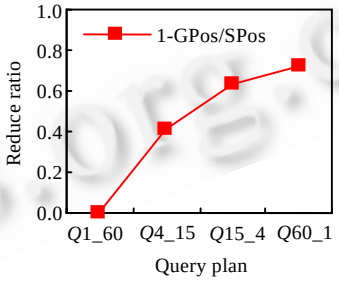
图 5 数据集反相关分布

为了显示相对于传送位置值策略、传送分组位置值策略在这 4 种情况下的有效程度,我们给出了当原始数据集对象总数为 1.6×10^6 时,在这 4 种情况下传送分组位置值策略的缩减率,如图 6 所示.从图 6 我们可以看出,在这 4 种情况下传送分组位置值策略缩减能力的强弱顺序为:情况(4)>情况(3)>情况(2)>情况(1).这是因为在通常情况下,当维空间树包含的节点越多时,被删除的重复值就越大.例如在图 6(a)中,在这 4 种情况下传送分组位置值策略的缩减率分别为 0,0.424,0.625 和 0.728;而在图 6(b)中,在这 4 种情况下传送分组位置值策略的缩减率分别为 0,0.416,0.633 和 0.721.



(a) Independent dataset

(a) 独立分布



(b) Anticorrelated dataset

(b) 反相关分布

Fig.6 Number of objects equals 1.6×10^6

图 6 对象个数为 1.6×10^6

4.3 本文方法传输的数据量评估

在这一节中,我们评估使用本文方法(TPAOSS 算法)时的数据传输量.在该组实验中,与 TPAOSS 算法比较的有 3 种方法:(1) 直接传输所有原始数据集,称为 TAll 方法;(2) 文献[10]中使用的传输扩展 skyline 集合,称为

TExtend 方法;(3) 在简单计算机上求出所有子空间上的 skyline 集合,然后将所获取的结果集进行传输,称为 TSUB 方法.实验环境和设置与第 4.2 节的相同.同时,为了简单起见,在 TPAOSS 算法的第 2 阶段,我们只考虑使用传送分组位置值策略.实验结果如图 7 和图 8 所示.

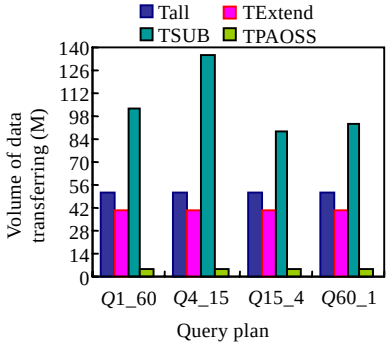


Fig.7 Independent dataset

图 7 数据集独立分布

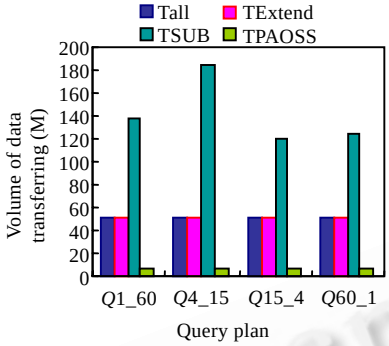


Fig.8 Anticorrelated dataset

图 8 数据集反相关分布

从实验结果我们可以看出,本文给出的 TPAOSS 算法在各种实验设置下均显著优于其余的 3 种方法.例如:在图 7 的 *Q*4_15 中,Tall 方法的数据传输量为 51.2M,TExtend 方法的数据传输量为 41.01M,TSUB 方法的数据传输量为 134.7M,而 TPAOSS 算法仅为 5.13M;在图 8 的 *Q*4_15 中,Tall 方法的数据传输量为 51.2M,TExtend 方法的数据传输量为 51.1M,TSUB 方法的数据传输量为 183.75M,而 TPAOSS 算法仅为 5.86M.同时,我们还可以得出,Tall 和 TExtend 这两种方法与维空间树中节点的构成无关;并且对于 Tall 方法来说,当数据对象个数确定以后,其数据传输量也随之确定;而 TExtend 方法除了与数据对象个数相关之外,还与数据集的分布有关.特别地,当数据反相关分布时,TExtend 方法的数据传输量几乎等于 Tall 方法.我们还注意到,在大多数情况下,TSUB 方法的数据传输量显著超过原始数据集的大小(即 Tall 方法的数据传输量).这是因为,一个对象可能同时出现在多个子空间 skyline 查询的结果集中,因此,当传输这 60 个子空间 skyline 查询所对应的结果集时,同一个对象可能进行了多次的网络传输.

5 结论与将来工作

查询预处理阶段的数据传输开销是 P2P 网络中子空间 skyline 查询应用的性能瓶颈,文献[10]通过传输扩展 skyline 集合来避免在预处理阶段传输所有的数据.然而我们的实验结果表明,文献[10]中所使用的方法在大多数情况下是低效的.为了解决查询预处理阶段的数据传输量的问题,本文基于全空间 skyline 集合与子空间 skyline 集合之间的语义关系,提出一种 3 阶段传输查询所需数据的有效算法 TPAOSS.在 TPAOSS 算法的第 1 阶段,我们只传送全空间 skyline 对象;在第 2 阶段,我们接收超点计算机回送的种子 skyline 对象的位置值;而在第 3 阶段,我们基于 Bloom filter 技术来传送种子 skyline 对象在子空间上重复对象的位置值以及重复频率.为了降低 TPAOSS 算法第 2 阶段的数据传输量,我们提出两种用来传输种子 skyline 对象的优化策略,即传送位置值策略以及传送分组位置值策略.这两种策略均使用一维数值替代多维对象来进行传输,从而显著降低了网络传输量.另外,传送分组位置值策略通过维空间的继承关系来进一步缩减网络的数据传输量.最后,我们从理论上分析和证明本文算法的正确性以及有效性.同时,在不同分布的数据集上实施大量的实验评估,并从多个不同的角度来综合考察本文算法的实用性和可扩展性.

References:

[1] Borzsonyi S, Kossmann D, Stocker K. The skyline operator. In: Ehrgott M, Greco S, Figueira J, eds. Proc. of the Int'l Conf. on Data Engineering. Melbourne: IEEE Computer Society, 2001. 421−430.

[2] Chomicki J, Godfrey P, Gryz J, Liang D. Skyline with pre-sorting. In: Lee JW, Gaewon Y, Ikchan, S, eds. Proc. of the Int'l Conf. on Data Engineering. Melbourne: IEEE Computer Society, 2003. 717−719.

[3] Tao YF, Xiao XK, Pei J. SUBSKY: Efficient computation of skylines in subspaces. In: Vladimirskyiy I, Bernhard S, Yannis T, eds. Proc. of the Int'l Conf. on Data Engineering. Melbourne: IEEE Computer Society, 2006. 65−75.

[4] Pei J, Jin W, Ester M, Tao YF. Catching the best views of skyline: A semantic approach based on decisive subspaces. In: Maurizio F, Francesco C, Francesco M, eds. Proc. of the Int'l Conf. on Very Large Data Bases. Norway: VLDB Endowment, 2005. 253−264.

[5] Godfrey P, Shipley R, Gryz J. Maximal vector computation in large data sets. In: Kossmann D, Ramsak F, Rost S, eds. Proc. of the Int'l Conf. on Very Large Data Bases. Norway: VLDB Endowment, 2005. 229−240.

[6] Pei J, Yuan YD, Lin XM, Jin W, Ester M, Liu Q, Wang W, Tao YF, Jeffrey XY, Zhang Q. Towards multidimensional subspace skyline analysis. ACM Trans. on Database Systems, 2006,31(4):1335−1381.

[7] Kossmann D, Ramsak F, Rost S. Shooting stars in the sky: An online algorithm for skyline queries. In: Lance P, Ehtesham H, Huan L, eds. Proc. of the Int'l Conf. on Very Large Data Bases. Norway: VLDB Endowment, 2002. 275−286.

[8] Papadias D, Tao YF, Fu G, Seeger B. An optimal and progressive algorithm for skyline queries. In: Ying Z, George K, Usama F, eds. Proc. of the 2003 ACM SIGMOD Int'l Conf. on Management of Data. ACM Press, 2003. 467−478.

[9] Yang B, Molina HG. Designing a super-peer network. In: Lee JW, Gaewon Y, Ikchan, S, eds. Proc. of the Int'l Conf. on Data Engineering. Melbourne: IEEE Computer Society, 2003. 49−60.

[10] Akrivi V, Christos D, Yannis K, Michalis V. SKYPEER: Efficient subspace skyline computation over distributed data. In: Pablo R, Rodriguez B, Hector C, eds. Proc. of the Int'l Conf. on Data Engineering. Melbourne: IEEE Computer Society, 2007. 416−425.

[11] Mitzenmacher M. Compressed bloom filters. IEEE Trans. on Networking, 2002,10(5):604−612.

黄震华(1980－),男,福建泉州人,博士,CCF学生会员,主要研究领域为数据库查询优化,数据挖掘.

王智慧(1975－),男,博士,讲师,主要研究领域为数据流查询,隐私挖掘.

郭建魁(1980－),男,博士,主要研究领域为数据流查询,数据挖掘.

汪卫(1970－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为数据库,数据仓库,数据挖掘.

施伯乐(1936－),男,教授,博士生导师,CCF高级会员,主要研究领域为数据库,知识库,数字图书馆,数据挖掘.