

数据流上高效计算子空间 Skyline 的算法

孙圣力 黄震华 李金玖 郭建奎 朱扬勇

(复旦大学计算机与信息技术系 上海 200433)

摘 要 流数据处理和多维空间中子空间上 Skyline 的计算是近年来数据管理与数据挖掘领域的研究热点. 此前相关工作只专注于滑动窗口上 Skyline 的维护问题, 未涉及到滑动窗口中子空间 Skyline 的计算. 文中提出了一个基于网格索引的高效维护滑动窗口上 Skyline 的算法, 以此为基础采用自顶向下的方式通过两个阶段增量式地返回目标子空间上的结果; 开发的多个剪枝策略和启发式优化方法显著地提高了全空间 Skyline 的维护以及子空间 Skyline 的计算效率. 理论分析和实验结果表明: 与同类算法相比, 文中提出的 StreamSubsky 算法以极少的时间开销就能输出第一个结果, 并且算法具有良好的可扩展性.

关键词 Skyline 计算; 数据流; 子空间 Skyline; 网格索引; 增量方法

中图法分类号 TP311

Efficient Computation of Subspace Skylines over Data Streams

SUN Sheng-Li HUANG Zhen-Hua LI Jin-Jiu GUO Jian-Kui ZHU Yang-Yong

(Department of Computing and Information Technology, Fudan University, Shanghai 200433)

Abstract Data stream processing and Skyline computing within subspaces have recently received a lot of attentions in database and data mining community. Previous works only sought to maintain full space Skylines over sliding windows. No one considered the problem of computing subspace Skylines over sliding windows. In this paper, the authors adopt an efficient full space Skyline maintaining method leveraging grid index structure, and based on this, propose an efficient top-town algorithm to incrementally output the Skyline objects within specified subspaces. Moreover, the authors present a set of optimizing heuristics to speed up these processes. Theoretical analysis and extensive experiments demonstrate that the proposed algorithm can generate the first result only taking a little overhead compared with previous works, and the methods are both efficient and scalable.

Keywords Skyline computing; data stream; subspace Skyline; grid index; incremental method

1 引 言

多维空间上 Skyline 的计算是国际上近年来数

据库与数据挖掘领域的一个研究重点和热点^[1]. Skyline 支持用户在复杂的情况下进行决策, Skyline 查询返回一组有趣的对象, 这些对象在各维上都不被其它对象所支配. 它在许多领域有着广泛的

收稿日期: 2007-03-05; 修改稿收到日期: 2007-05-25. 本课题得到国家“九七三”重点基础研究发展规划项目基金(2005CB321905)资助.
孙圣力, 男, 1979 年生, 博士研究生, 主要研究方向为流数据管理与挖掘、分布式数据库. E-mail: slsun@fudan.edu.cn. 黄震华, 男, 1980 年生, 博士研究生, 主要研究方向为数据库查询优化、数据挖掘. 李金玖, 男, 1979 年生, 硕士研究生, 主要研究方向为数据挖掘. 郭建奎, 男, 1980 年生, 博士研究生, 主要研究方向为数据流查询、Web 数据挖掘. 朱扬勇, 男, 1963 年生, 教授, 博士生导师, 主要研究领域为数据库与知识库、数据挖掘、生物信息学.

应用,如多因素决策支持、城市导航以及用户偏好查询等.此前的研究工作大多基于如下两个假设:(1)查询所涉及的维属性组合不发生变化,即所有用户所关心的度量指标相同;(2)对象更新不频繁,即数据集很少发生变化.然而在现实中存在着大量的这两个假设同时不成立的情况,如在线拍卖、股票推荐等基于动态数据的应用.因此文献[1-2]中给出的计算方法在现实应用中存在很大的局限性.最近关于 Skyline 计算的文献只针对某一假设不满足的情况提出解决方案,而不考虑另一假设不满足的情况.文献[3-4]首次提出了子空间上 Skyline 的概念.然而它们提出的方法更新代价巨大.文献[5]采用的方法虽然提高了更新效率,但依然远远不适用于像数据流这种频繁更新的环境.文献[6]提出的 SUBSKY 算法则采用的是一种维度压缩的方案,但它对维度敏感并且要全部的结果计算完毕方才产生第一个输出.文献[7-8]则考虑了第 2 个假设不满足的情况,即数据流环境下的维护 Skyline 的问题,然而它未考虑子空间上的计算问题,而且更新维护代价较高.

本文首次研究了在以上两个假设均不满足的情况下,如何在基于滑动窗口模型的数据流中高效地计算子空间上 Skyline 的问题.本文提出的 Stream-Subsky 算法较好地解决了该问题,算法基本思路是:首先动态地维护滑动窗口中全空间上的 Skyline,再根据全空间与子空间 Skyline 之间的关系,通过两个阶段以增量的方式来返回子空间上的结果.本文借鉴文献[7]中的 Eager 算法来维护滑动窗口上的 Skyline.本文的主要贡献是:

(1)本文提出了在滑动窗口中计算任意子空间上 Skyline 的问题,并给出了一个高效的增量式的解决方案.

(2)本文对 Eager 算法^[7]进行了大幅度的改进,采用更合适的索引结构,开发了多个新颖的剪枝策略,显著地提高了在滑动窗口中维护全空间 Skyline 的效率.

(3)采用多个启发式方法,显著地加速了由全空间 Skyline 到目标子空间上 Skyline 的计算过程.

(4)本文进行了大量严格细致的实验,实验表明 StreamSubsky 算法高效而且具有良好的可扩展性.

本文第 1 节介绍本文研究的问题及相关工作;第 2 节阐述算法的基本思想;第 3 节讨论算法详细的实现策略及启发式方法;第 4 节进行算法的代价分析;第 5 节给出实验结果;第 6 节对全文工作进行

总结,并给出后续研究方向.

2 算法基本思想

2.1 节首先给出本文中所用到的术语及其定义,2.2 节给出算法所采用的基本方案,接着给出一个例子说明算法的基本思想.

2.1 术语定义

在阐述算法基本思想之前,先给出本文用到的几个重要术语.假设有 D 维空间 \mathcal{D} , $\mathcal{D} = \{a_1, a_2, \dots, a_D\}$. 其中属性 a_1, a_2, \dots, a_D 的定义域均为区间 $(0, 1]$. 集合 U 中的对象均来自空间 \mathcal{D} .

定义 1. 支配. 任意给定两个对象 $p, p' \in U$. 若 p 在任一属性上的取值都不大于对象 p' , 且至少在某一属性上的取值 p 比 p' 要小, 则说 p 在空间 \mathcal{D} 上支配 p' , 记为 $p < p'$. 若 p 在 \mathcal{D} 的某个子空间 \mathcal{B} 上支配 p' , 则记为 $p <_{\mathcal{B}} p'$.

定义 2. Skyline. D 维空间 \mathcal{D} 上, U 中所有不被其他对象所支配的对象组成的集合, 记为 S .

定义 3. 子空间 Skyline. 在 \mathcal{D} 的某一子空间 \mathcal{B} ($\mathcal{B} \subseteq \mathcal{D}, \mathcal{B} \neq \emptyset$) 上, U 中所有不被其他对象所支配的对象组成的集合, 记为 $S_{\mathcal{B}}$.

定义 4. 子空间 \mathcal{B} 上的种子 Skyline. 它是这样集合, 若 $s \in S$, 且 $\exists t \in S, t <_{\mathcal{B}} s$, 则 s 为该集合的成员. 子空间 \mathcal{B} 上的种子 Skyline 记为 $Seed(\mathcal{B})$. 在不引起混淆的情况下也可简记 $Seed$.

本文将数据流中对象到达系统的时间戳看作对象的一个属性, 时间戳为整数, 自 0 开始递增. 这样任意对象 p 就可以记为 $p(p.t_{arr}, p.val_1, \dots, p.val_D)$, 其中 $p.t_{arr}$ 表示其到达系统的时间戳, $p.val_i$ 表示其在属性 a_i 上的取值. 本文主要讨论基于时间的滑动窗口, 设滑动窗口宽度为 W , 即表示对象 p 仅在时间区间 $[p.t_{arr} \dots p.t_{arr} + W - 1]$ 上是有效的. 本文提出的算法能够直接地应用到基于计数的滑动窗口中去. 例如, 可以人为地给后者中到达的对象关联一个虚拟的到达时间, 该虚拟时间可以以对象在数据流中的次序号充任. 下表给出本文中常用的符号及其意义.

表 1 符号说明

符号	描述
$U(t_i)$	时间戳 t_i 上到达系统的对象集合
$U = \bigcup_i U(t_i)$	当前窗口上所有对象的集合
S	当前窗口上的全空间 Skyline
$S_{\mathcal{B}}$	当前窗口上子空间 \mathcal{B} 上的 Skyline
$Seed$	种子 Skyline 对象集合
π	投影运算符

2.2 StreamSubsky 算法基本思想

StreamSubsky 算法的基本思想是动态维护当前窗口中的对象集 U 和全空间 \mathcal{D} 上的 Skyline \mathcal{S} . 当需要计算某子空间 \mathcal{B} 上的 Skyline 时, 采用两阶段的方案求解: 首先依据 \mathcal{S} 计算出子空间 \mathcal{B} 上的种子 Skyline $Seed$, 计算 $Seed$ 时产生的第一个对象就是最早的响应结果; 接着再基于 $Seed$ 中的对象回查 $U-\mathcal{S}$, 找出 $\forall u \in U-\mathcal{S}, u$ 满足条件 $\exists s \in Seed, \pi_{\mathcal{B}}(u) = \pi_{\mathcal{B}}(s)$, 即找出 $U-\mathcal{S}$ 中全部这样的对象, 该对象在 \mathcal{B} 上的投影在 $Seed$ 在 \mathcal{B} 上的投影结果集中出现过. 全部这样的 u 构成集合 $\Delta Seed$, $Seed \cup \Delta Seed$ 就是完整的结果 $\mathcal{S}_{\mathcal{B}}$. 当然由 $U-\mathcal{S}$ 中的对象回查 $Seed$, 效果也一样. 由上述叙可知该算法实质上是一种自顶向下的方法, 工作在一个通常要比数据集 U 小得多的集合 \mathcal{S} 上. 设 $M = |\mathcal{S}|, D = |\mathcal{D}|, N = |U|$, 在各对象各维取值相互独立的条件下文献[9]给出 \mathcal{S} 的规模, 见式(1).

$$M = \theta\left(\frac{(\ln N)^{D-1}}{(D-1)!}\right) \tag{1}$$

利用较小的 \mathcal{S} 来求得目标子空间上的结果, 可以达到较高效率. 实验结果也表明: 与相关的算法 (如 SUBSKY^[6]) 相比, StreamSubsky 算法以极少的时间就能使用户得到响应, 以少得多的时间就能输出完整的结果. 为了方便算法的理解, 以下先给出一个例子.

例 1. 假设滑动时间窗口宽度 W 为 3, 全空间维度为 3, 目标子空间 \mathcal{B} 由 xy 两维组成. 现要求当前窗口中 \mathcal{B} 上的 Skyline. 表 2 是一个示例的数据集, 其中: $\{p_1, p_2\}, \{p_3, p_4, p_5\}, \{p_6, p_7, p_8\}$ 分别在时间戳 t_1, t_2 和 t_3 上到达.

表 2 示例数据集								
	t_1		t_2			t_3		
	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8
X	0.1	0.3	0.2	0.4	0.1	0.5	0.1	0.7
Y	0.4	0.5	0.3	0.1	0.3	0.2	0.3	0.8
Z	0.2	0.3	0	0.1	0.4	0.3	0.2	0.3

算法维护集合 \mathcal{S} , 该例中 $\mathcal{S} = \{p_3, p_4, p_7\}$. 首先在 \mathcal{S} 上计算对象集 $Seed$. 在 \mathcal{S} 上存在 $p_7 \prec_{\mathcal{B}} p_3$, 所以 $Seed = \{p_4, p_7\}$. 接着再用 $Seed$ 中所有对象回查 $U-\mathcal{S} = \{p_1, p_2, p_5, p_6, p_8\}$, 只发现 $\pi_{xy}(p_5) = \pi_{xy}(p_7)$. 所以得到 $\Delta Seed = \{p_5\}$. 完整的结果集 $\mathcal{S}_{\mathcal{B}} = Seed \cup \Delta Seed = \{p_4, p_5, p_7\}$. 以下给出算法的正确性证明.

定理 1. StreamSubsky 算法输出的结果是正确的而且是完备的.

证明. 首先进行正确性证明, 只需要证明: $\forall r \in Seed$, 则 $r \in \mathcal{S}_{\mathcal{B}}$. 采用反证法. 已知 $r \in Seed$, 则必有 $r \in U$ 且 $r \in \mathcal{S}$. 若 $r \notin \mathcal{S}_{\mathcal{B}}$ 则 $\exists t \in U, t \prec_{\mathcal{B}} r$. 设 $T = \{t | t \in U \wedge t \prec_{\mathcal{B}} r\}$, 则 $\exists v \in T, v$ 不为 T 中任何对象所支配. 再考查集合 $U-T$, 对于 $\forall o \in U-T, o \not\prec_{\mathcal{B}} r$. 此处只存在两种可能, 第一种: $\pi_{\mathcal{B}}(o) = \pi_{\mathcal{B}}(r)$; 第二种: $\exists a_i \in \mathcal{B}, r.val_i < o.val_i$. 现已知 $v \prec_{\mathcal{B}} r$. 如果第一种情况成立, 则 $v \prec_{\mathcal{B}} o$, 则必 $\exists a_j \in \mathcal{B}, v.val_j < o.val_j$. 如果第二种情况成立, 因为 $r.val_i < o.val_i$, 而 $v.val_i \leq r.val_i$, 则有 $v.val_i < o.val_i$. 综合两种情况: o 至少在某一维上的取值比 v 大, 故 $o \not\prec v$. 故在 U 和 $U-T$ 中都不存在支配 v 的对象, $v \in \mathcal{S}$. 又因为 $v \prec_{\mathcal{B}} r$, 则 $r \notin Seed$, 这与已知条件相悖, 故命题得证.

接着进行完备性证明, 只需证明 $\forall r \in \mathcal{S}_{\mathcal{B}}$ 均能由算法输出. 对于 $\forall r \in \mathcal{S}_{\mathcal{B}}$, 设 $T = \{t | t \in U \wedge \pi_{\mathcal{B}}(t) = \pi_{\mathcal{B}}(r)\}$. 则 $\exists v \in T, v$ 不为 T 中任何对象所支配. 再考查集合 $U-T$. 因为 $v \in \mathcal{S}_{\mathcal{B}}$, 则对于 $\forall o \in U-T$, 关系 $o \not\prec_{\mathcal{B}} v$ 成立. 又 $\pi_{\mathcal{B}}(o) \neq \pi_{\mathcal{B}}(v)$, 则至少在 \mathcal{B} 上的某一属性上 v 的取值比 o 小. 所以 $\forall o \in U-T, o \not\prec v$ 成立. 所以 $v \in \mathcal{S}$. 因为 $v \in \mathcal{S}_{\mathcal{B}}$, 所以在 \mathcal{S} 中不存在子空间 \mathcal{B} 上支配 v 的对象, 按照算法设计 v 进行入集合 $Seed$ 中. $Seed$ 中的 v 要回查 $U-\mathcal{S}$, 所以 T 中属于集合 $U-\mathcal{S}$ 的对象都能被算法找到进入 $\Delta Seed$ 中, 而 T 中属于集合 \mathcal{S} 的对象事先已经进入 $Seed$ 中 (和 v 一道), 故 r 一定会进入 $Seed \cup \Delta Seed$ 中形成输出. 命题得证. 综上所述, 定理 1 得证. 证毕.

3 算法实现策略

3.1 节首先详细地讨论如何高效地维护全空间 Skyline, 3.2 节接着讨论以此为基础如何高效地进行子空间 Skyline 的计算.

3.1 全空间 Skyline 维护方案

基于数据流的算法关键是高效地进行更新维护. StreamSubsky 算法的维护工作包括当前窗口上的对象以及全空间 Skyline 的更新与维护. 受文献[7]的 Eager 算法的启发, 本文也采用事件列表 (event list) 机制来支持维护全空间上的 Skyline. 即将对象的过期或开始成为 Skyline 都看作一个事件, 事件依据其发生的时间点来触发. 该方案的最大优点是当某 Skyline 对象过期时, 系统中已存在该对象的替补, 从而避免在其排它支配域中计算 Skyline. 但是 Eager 算法在求新到对象的影响时间与处理被新来对象所支配的对象这两个关键问题上, 存

在很大不足:将全部对象按链表或 R-树^[10]结构来组织,再使用 d-向范围搜索(d-sided search)^[10]来支持后续操作. d-向范围搜索无论是在链表还是在 R-树来进行都是相当低效的. 在链表上搜索理论上需要穷尽整个链表,而在 R-树上搜索同样效率也不高. 因为构造 R-树所产生的 MBR 之间往往存在较严重的相互重叠,且当维度稍高时搜索 R-树的效率会急剧下降^[11]. 更重要的是:由于 R-Tree、R*-Tree 之类的内存索引只适合于固定维上的操作,而本文的应用将涉及到子空间上的计算,因此这类索引不适合本文的环境. 基于此本文采用网格索引^[12]来组织对象,并提出两个剪枝策略来加速以上两关键问题的执行.

在阐述具体策略之前,先考查数据流中各对象的性质. 本文将当前窗口中的对象称为活动对象,活动对象根据其不同性质进一步分为 Skyline 对象(当前窗口中 \mathcal{D} 上的 Skyline)、候选 Skyline 对象(在未来的某个时间可能成为 \mathcal{D} 上的 Skyline)和平凡对象(在当前和未来某个时间可能成为某 \mathcal{B} 上的 Skyline),本文分别称它们为第 I、II 和 III 类对象. 并将第 I、II 类对象合称为非平凡对象. 文献^[13]提出的 ext-skyline 大大地减少需缓存的第 III 类对象. StreamSubsky 算法中只保存 ext-skyline 对象.

不失一般性,图 1 给出了全空间维度为 2 时 StreamSubsky 所用的数据结构. 采用队列来缓存活动对象,新到达的对象插入到队列尾(右端),而过期的对象从队列头(左端)删除. 这样该队列中的对象是头到尾时间戳由小到大有序的. 采用网格作为活动对象的索引. 网格可以看作成一个多维数组,已知一个格(cell)的坐标,可以很容易地求出该格所包含活动对象的坐标范围. 例如,对于 D 维的网格索引 G , G 中的格 $c[pos_1, pos_2, \dots, pos_D]$ 所包含的活动对象的第 $j(j \in \{1, 2, \dots, D\})$ 个坐标的取值范围为 $((pos_j - 1) \cdot \delta, pos_j \cdot \delta]$, 其中 pos_j 为格 c 在第 j 维上的坐标, $pos_j \in \{1, 2, \dots, K\}$, δ 为格的宽度($\delta = 1/K$, K 为每维的等份数, D 为全空间维度,后续对 K 和 D 不作额外声明就采用该意义). 反之已知一个活动对象的坐标也可以很容易地求出其所在的格. 每个格关联 3 个队列,分别用来保存指向上述 3 类活动对象的指针. 格中的每一个队列也同样地进行动态更新,队列所关联的各对象的时间戳也是由头到尾、由小到大有序的.

与文献^[7]不同的是,本文采用一个长度为用户所定义的滑动窗口的最大宽度的数组来实现事件

链. 该数组的每个分量关联一个队列,该队列中保存具有相同触发时间的事件. 一个事件用 3 个分量来表示^[7],即对象位置;事件标志(标识对象过期或成为 Skyline);触发时间. 下面将详细分析依据此数据结构而提出的剪枝策略.

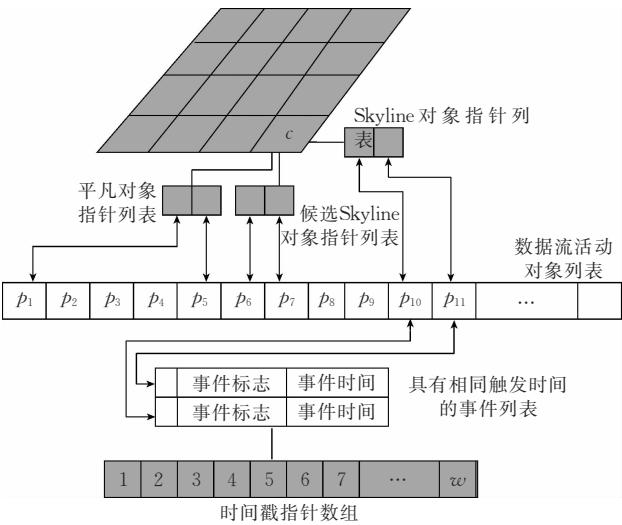


图 1 StreamSubsky 所用数据结构

剪枝策略之一:基于网格的剪枝策略. 求对象的影响时间就是计算落在其反支配区域(ADR)内的非平凡对象的最大时间戳;而处理被新来对象支配的对象,则需要在其支配区域(DR)内进行. 支配域与反支配域如图 2(a)所示. 首先可以依据网格中各格的相互关系进行剪枝. 求 p 的影响时间,对 p .ADR 所完全覆盖的格(图 2(b)左下部浅灰色区域)最多需要读取两次(分别为读取格中第 I、II 类对象队列中尾对象时间戳);而对 p .DR 所完全覆盖的格(图 2(b)右上部浅灰色区域)中的所有对象可以直接删除. 在执行求影响时间与处理被支配对象时,仅需要展开被 p .ADR 和 p .DR 部分重叠的格(图 2(b)中深灰色区域),与其中的非平凡对象先进行支配关系测试,再进行后续操作. 不妨分别设 p .ADR 与 p .DR 部分覆盖的格中所含的非平凡对象数与当前窗口中非平凡对象数目之比为 r_{gp} 和 r'_{gp} ,称它们为格剪枝

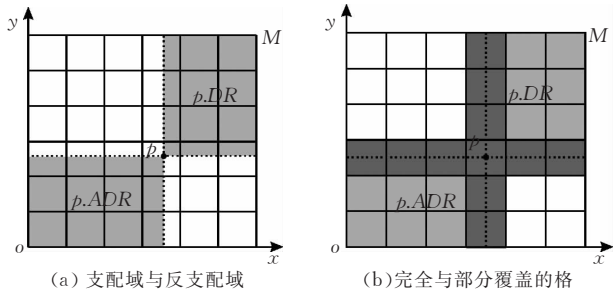


图 2 基于网格的剪枝策略示意

因子. 支配关系测试最坏情况下要在 D 维上进行比较, 减少支配关系测试次数, 就能提高算法的性能.

剪枝策略之二: 基于时间的剪枝策略. 时间戳是递增的, 较迟到达系统的对象的时间戳较大. 分析 Eager 算法中新到达的对象其支配与反支配域中所包含的非平凡对象在时间上的相互关系, 可以得到以下定理.

定理 2. 如果某对象反支配域与支配域中均存在非平凡对象, 则其反支配域中所包含的非平凡对象的最大时间戳小于其支配域中所包含的非平凡对象的最小时间戳.

证明. 不妨设新到达的对象为 p , $p.DR$ 和 $p.ADR$ 中均存在非平凡对象. 设 $p.DR$ 中所包含的非平凡对象的最小时间戳为 t_{\min} , $p.ADR$ 中所包含的非平凡对象的最大时间戳为 t_{\max} . 现在 $p.DR$ 中任意选取一个时间戳等于 t_{\min} 的非平凡对象, 设为 p_{\min} ; 在 $p.ADR$ 中任取选取一个时间戳等于 t_{\max} 的非平凡对象, 设为 p_{\max} , 显然 $p_{\max} < p_{\min}$. 现分两种情况来讨论. 若 $t_{\max} = t_{\min}$, 说明 p_{\max} 与 p_{\min} 在相同的时间戳上到达, 按照 Eager 算法的设计在处理完 t_{\max} 上到达的对象之后, p_{\min} 不可能还出在非平凡对象集中, 故 $t_{\max} \neq t_{\min}$. 若 $t_{\max} > t_{\min}$, p_{\max} 较迟到达系统, 因为 $p_{\max} < p_{\min}$, p_{\max} 到达系统后立即将 p_{\min} 从非平凡对象集中移除. p 迟于 p_{\max} 到达, 等到 p 到达系统, p_{\min} 已经不存在于非平凡对象集中. 故 t_{\max} 不大于 t_{\min} . 综合以上两情况 $t_{\max} < t_{\min}$, 定理得证. 证毕.

依据以上剪枝策略可以设计出高效的求影响时间与处理被支配对象的方法. 求影响时间的算法由图 4 给出. 第 2 步初始化是利用 $p.ADR$ 所完全覆盖的格(图 3(a)左下角浅灰色区域)中的非平凡对象的最大时间戳初始化时间戳下界 $t_{\text{lowerbound}}$; 利用 $p.DR$ 完全覆盖的格(图 3(a)右上角浅灰色区域)中的非平凡对象的最小时间戳初始化上界 $t_{\text{upperbound}}$. 第 3 步逐一展开 $p.ADR$ 所部分覆盖的格(见图 2(b)). 第 8 步进行支配测试前, 先依据定理 2 在第 7 步对格中的非平凡对象进行筛选. 只有条件 $o.\text{timestamp} < t_{\text{upperbound}}$ 满足将 o 与 p 进行支配测试. 又因为队列中对象的时间戳是由首到尾是递增的, 对非平凡对象队列读取采用由尾到头的顺序进行, 一旦支配测试成功, 即可结束操作. 第 4、10 步对队列读取也采用由尾向头的方向进行, 每次取第 I, II 类对象队列中的尾对象, 返回较大的时间戳. 在此提出一个符号 r_{ip} , 称为时间戳剪枝因子, 即反支配域部分覆盖的格中满足第 7 步的条件非平凡对象数占反支配域

部分覆盖的格中所含非平凡对象总数的比例.

处理被支配的对象算法的详细过程由图 5 给出. 需要指出的是, 逐一展开 $p.DR$ 所部分覆盖的格(见图 2(b)), 在第 6 步进行支配测试之前, 先在第 5 步依据定理 2 对格中的非平凡对象进行筛选. 在此给出一个符号 r'_{ip} , 同样也称为时间戳剪枝因子, 意义是 p 的支配域所部分覆盖的格中满足条件 $o.\text{timestamp} > t_{\text{lowerbound}}$ 的非平凡对象数占支配域部分覆盖的格中所含非平凡对象总数的比例. 第 3、7 步对队列读取同样采用由尾向头的方向进行.

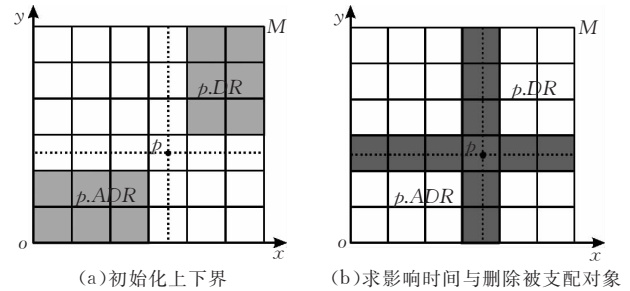


图 3 求影响时间与删除被支配对象示意

Procedure ComputeInfTime

Input: p // 待求影响时间的对象

1. $t_{\text{lowerbound}} \leftarrow -1, t_{\text{upperbound}} \leftarrow \text{maximum timestamp};$
2. 初始化 $t_{\text{upperbound}}$ 和 $t_{\text{lowerbound}};$
3. for 每一个被 $p.ADR$ 部分覆盖的格 c
4. $o \leftarrow c$ 中具有最大时间戳的非平凡对象, 若 c 中两队列均为空则返回 $\emptyset;$
5. while $o \neq \emptyset$
6. if $o.\text{timestamp} \leq t_{\text{lowerbound}}$ break;
7. if $o.\text{timestamp} < t_{\text{upperbound}}$
8. if o dominate p $t_{\text{lowerbound}} \leftarrow r_{ip} \cdot t_{\text{arr}};$ break;
9. $o \leftarrow c$ 中下一个具有最大时间戳的对象, 若两队列均到头部则返回 $\emptyset;$
10. Return $t_{\text{lowerbound}}$

图 4 计算影响时间的过程

Procedure ProcessDomObjects

Input: $p, t_{\text{lowerbound}}$ // 后者为 ComputeInfTime 的结果

1. 删除 $p.DR$ 完全覆盖的格中对象及其 EL 项;
2. for 每一个被 $p.DR$ 部分覆盖的格 c
3. $o \leftarrow c$ 中具有最大时间戳的非平凡对象, 若 c 中两队列均为空则返回 $\emptyset;$
4. while $o \neq \emptyset$
5. if $o.\text{timestamp} > t_{\text{lowerbound}}$
6. if p dominate o 将 o 迁入平凡对象队列;
删除 o 所对应的 EL 项;
7. $o \leftarrow c$ 中下一个具有最大时间戳的非平凡对象, 若两队列均到头部则返回 $\emptyset;$
8. else break;

图 5 处理被支配对象的过程

图 6 给出了依据事件链机制^[7]进行过期处理的步骤. 其中 W 为用户所定义的滑动窗口宽度. pos 是以当前时间戳来触发的事件集在事件列表中的入

口. 下图 7 给出了 StreamSubsky 算法的主模块, 每当时间戳发生跳变时就触发该模块. 新时间戳上到达的对象存放在缓冲区 BF 中. 第 1 步进行过期处理; 第 3 步计算影响时间; 5~10 步将对象插入到其所属格中适当的队列中并在事件列表中增加一项; 11 步处理被支配的对象.

Procedure ProcessExpiration

1. $pos \leftarrow \text{current timestamp MOD } W$;
2. for $EL[pos]$ 所指的列表中的每一项 $Item$
3. r : $Item.pointer$ 所指的对象;
4. if $Item.tag = 'EX'$ // 如果 r 过期了
5. 从活动对象列表中删除 r ;
6. 将 $Item$ 从 $EL[pos]$ 所指列表中删除;
7. else // r 此时成为 Skyline 对象
8. 将 r 迁入到其所属格的 Skyline 队列;
9. $Item.tag \leftarrow 'EX'$; $Item.timestamp \leftarrow r.t_{arr} + W$;
10. 将 $Item$ 迁入 $EL[r.t_{arr} + W \text{ MOD } W]$ 所指列表;

图 6 过期处理过程

Algorithm MaintainSkyline

1. 调用 ProcessExpiration 进行过期处理;
2. for BF 中的每一个对象 p
3. 调用 ComputeInfTime 获得 p 的影响时间 t_{inf} ;
4. if $t_{inf} < p.t_{arr}$
5. if $t_{inf} = -1$ // 不存在支配 p 的对象
6. 将 p 加入到其所属格的 Skyline 队列中;
7. 插入一项 $\langle (a \text{ pointer to } p), p, p.t_{arr} + W, 'EX' \rangle$ 到 $EL[p.t_{arr} + W \text{ MOD } W]$ 列表中;
8. else
9. 将 p 加入到其所属格候选 Skyline 队列中;
10. 插入一项 $\langle p, t_{inf} + W, 'SK' \rangle$ 到 $EL[t_{inf} + W \text{ MOD } W]$ 所指列表中;
11. 调用 ProcessDomObjects 处理 p 支配的对象;

图 7 StreamSubsky 算法维护主程序

3.2 子空间 Skyline 计算方案

子空间 Skyline 计算分为两个阶段来进行. 第一阶段, 基于 S 计算 $Seed$, 可以采用文献[2]提出的 SFS 方法来实现. 第二阶段, 利用第 II、III 类对象回查 $Seed$ 求 $\Delta Seed$. 查询处理过程的主要瓶颈在第二阶段. 一种简单的回查方法是利用每一个 II、III 类对象直接去与 $Seed$ 中的对象进行比较, 最坏情况下每对象需要的比较次数为 $|Seed|$. 这显然是低效的. 本文提出两个启发式方法来解决这一问题.

启发式方法之一: 基于网格的回查. 充分利用维护过程所建立的网格结构来加速回查过程. 给定查询请求 $Q(B)$, B 为目标子空间, $|B| = d$. 对于某一对象, 需要在 B 上与 $Seed$ 中对象进行等值比较. 因为该对象在 B 上有确定的数值, 所以 $Seed$ 对象在 B 上的取值是受限的, 在剩下的 $D-d$ 维上的则取值是自由的. 这样该平凡对象仍需访问的格数目为 K^{D-d} ,

占格总数 K^D 的比重为 $(1/K)^d$. 例如, 在例 1 中, p_8 进行回查时只需要与下图 8 所示的阴影部分的格中 $Seed$ 对象进行比较. 本文称阴影部分的格为候选格.

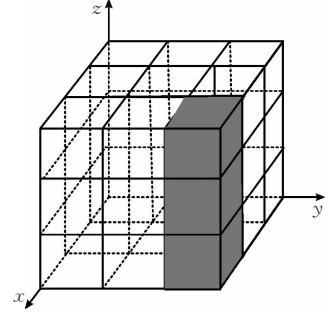


图 8 基于网格的回查示意

启发式方法之二: 利用 Bloom Filter 进行过滤. 本文使用 Bloom Filter 对平凡对象进行过滤. Bloom Filter 是一种能保证错误率并只会出现假阳性(false positive)的成员测试方法, 它使用短的 Hash 串来再现一个长字符串所代表的空间, 工作原理参见文献[14], 本文不再赘述. 使用 Bloom Filter 进行过滤的具体分为两个步骤. 首先初始化 Hash 表(图 9 第 3 步), 对 $Seed$ 中的各对象以其在目标子空间 B 上的取值构造 Key, 向 Hash 表进行映射. 第二步, 对 II、III 类对象以同样的方式构造 Key, 再进行成员测试(图 9 第 5 步). 若测试为真则回查 $Seed$ 作进一步验证, 若为假则被过滤掉不作进一步处理. 本文中 Bloom Filter 的关键是如何构造 Key, 在保证不出现假阴性(false negative)的条件下使假阳性尽可能小. 出现假阴性则导致结果不完备; 出现假阳性则导致回查次数增加, 降低算法性能. 本文采用式(3)来构造 Key, 其中 $p.coord[u]$ 代表对象 p 在属性 $u \in D$ 上的取值; i 代表属性 u 在 D 上的次序号(假设将 D 中属性从左到右从 1 开始编号); R 代表精度, 即各维的最大精度(取值位数). 例如, 在例 1 中, R 为 1, 属性 x, y, z 对应的次序号分别为 1, 2, 3, 则 p_8 的在子空间 $B(xy)$ 上的 Key 值为 $0.7 \times 10^1 + 0.8 \times 10^2 = 87$. 不难证明此种构造方法既不会

Algorithm ProcessQuery(B)

Input: S, B // 全空间 Skyline 及目标子空间

1. $Seed = \emptyset, \Delta Seed = \emptyset$;
2. 使用 SFS 算法依据 S 计算 $Seed$;
3. 依据 $Seed$ 初始化 Hash 表;
4. for 活动对象列表中的每一个 II、III 类对象 o
5. if o 存在于 Hash 表中
6. for o 的候选格中的每一个 $Seed$ 对象 s
7. if $\pi_B(s) = \pi_B(o)$ $\Delta Seed = \Delta Seed \cup \{o\}$;
8. Return $Seed \cup \Delta Seed$.

图 9 StreamSubsky 算法计算子空间 Skyline 的模块

出现假阴性也不会出现假阳性。

$$Key(p) = \sum_{u \in B} (p.coord[u] \times 10^{iR}) \quad (2)$$

综合上述两种启发式方法,图 9 给出了计算子空间 Skyline 的过程。

4 算法代价分析

StreamSubsky 算法的时间开销分为两部分:全空间 Skyline 维护开销和子空间 Skyline 计算开销。首先分析前者。每个新时间戳到来,维护做以下操作:首先作过期处理,该代价和以本时间戳作为触发时间的事件数目相关。数据流中每个对象在其生命周期内最多产生两个事件^[7],故过期处理每时间戳上分摊的开销为数据流的平均数据率。接着利用缓冲区 BF 中的对象作后续处理,其代价为 $|U(t_w)|(C_{InfTime} + C_{PocDom})$,其中, $|U(t_w)|$ 代表单个时间戳上到达的最多的对象数, $C_{InfTime}$ 为求影响时间的代价, C_{PocDom} 为处理被支配对象的代价。在此着重分析 $C_{InfTime}$ 和 C_{PocDom} 的规模。

求影响时间的代价 $C_{InfTime}$ 由初始化 $t_{lowerbound}$, $t_{upperbound}$ 与展开反支配域所部分覆盖格进行后续操作两部分组成。初始化开销仅与支配域和反支配域完全覆盖的格的数目相关,在每一格里最多读取两次,这些格的数目最多为 $(K-1)^D$,故初始化开销为 $O((K-1)^D)$ 。展开部分覆盖的格进行支配测试,需要展开的最大格数为 $K^D - (K-1)^D$;接着再对这些格中的对象进行支配测试,其代价为 $r_{tp} \times r_{gp} \times (|S_{max}| + |S_{Cmax}|) \times C_{dominate} + (1 - r_{tp}) \times r_{gp} \times (|S_{max}| + |S_{Cmax}|)$ 。其中前项为通过筛选的对象发生的开销,后项为没有通过筛选的对象发生的开销, r_{tp} , r_{gp} 为此前讨论过的时间戳剪枝因子和格剪枝因子, $|S_{max}|$, $|S_{Cmax}|$ 为窗口上最大的 I、II 类对象数, $C_{dominate}$ 为支配测试代价。所以求影响时间总的开销 $C_{InfTime}$ 为 $O(K^D + r_{gp} \times (|S_{max}| + |S_{Cmax}|) \times (r_{tp} \times C_{dominate} - r_{tp} + 1))$ 。处理被支配对象的开销 C_{PocDom} 为 $O(K^D + (|S_{max}| + |S_{Cmax}|)(1 + r'_{tp} \times r'_{gp} \times (C_{dominate} - 1)))$,其分析方法与前者类似,在此不作冗述。综上所述得出以下定理。

定理 3. StreamSubsky 维护全空间 Skyline 过程单个时间戳上分摊的开销为 $O(|U(t_w)|(C_{InfTime} + C_{PocDom} + 1))$,其中 $C_{InfTime}$ 的规模为 $O(K^D + r_{gp} \times (|S_{max}| + |S_{Cmax}|) \times (r_{tp} \times C_{dominate} - r_{tp} + 1))$; C_{PocDom} 的规模为 $O(K^D + (|S_{max}| + |S_{Cmax}|)(1 + r'_{tp} \times r'_{gp} \times (C_{dominate} - 1)))$ 。

下面讨论计算目标子空间上 Skyline 的开销。产生全部结果的代价包括计算 *Seed* 和 $\Delta Seed$ 两部分。本文采用 SFS 算法^[2]计算 *Seed*。*S* 经排序后产生的第一个对象即为 *Seed* 对象,也即为第一个结果。故产生第一个结果的代价即为 $O(|S_{max}| \ln |S_{max}|)$ 。

引理 1. 计算 *Seed* 的代价为 $O(|S_{max}| \times (\ln |S_{max}| + |Seed_{max}|))$,其中 $|Seed_{max}|$ 代表窗口上具有的最大的 *Seed* 对象数。

证明. 略。

引理 2. 计算 $\Delta Seed$ 的代价为 $O(|U_{max}| + (|U_{max}| \times r_{fp} + |\Delta Seed_{max}|) \times (|Seed_{cand}| + K^{D-d}))$ 。其中 $|U_{max}|$ 代表窗口上具有的最大对象数; $|\Delta Seed_{max}|$ 是窗口上 $\Delta Seed$ 的最大规模; r_{fp} 是 Bloom Filter 发生假阳性的概率; $|Seed_{cand}|$ 为候选格中的最大 *Seed* 对象数, d 为目标子空间维度。

证明. 计算 $\Delta Seed$ 的过程是先对平凡对象进行过滤,通过过滤的对象再回查 *Seed* 对象。过滤过程的开销为 $O(|U_{max}|)$ 。而回查开销由回查次数和每次回查的代价来决定。先考虑回查次数。发生回查的情形有两种: Bloom Filter 出现假阳或成员测试为真。假阳性发生的概率可由式(3)给出^[15]。其中 k 为相互独立的 Hash 函数的个数, m 为 Hash 表的长度(比特数), n 为 Key 的数目。当 k 与 m/n 确定, r_{fp} 为常数。

$$r_{fp} = \left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^k \approx (1 - e^{-\frac{kn}{m}})^k \quad (3)$$

例如:当 $k=2$, $m/n=8$ 时, $r_{fp}=0.0489$ 。回查次数为 $|U_{max}| \times r_{fp} + |\Delta Seed_{max}|$ 。每次回查的代价由候选格的数目及其中的 *Seed* 对象数目决定,由 3.2 节的分析其规模为 $O(|Seed_{cand}| + K^{D-d})$ 。回查部分的开销为 $O((|U_{max}| \times r_{fp} + |\Delta Seed_{max}|) \times (|Seed_{cand}| + K^{D-d}))$ 。综合以上分析引理得证。证毕。

综上所述得出以下定理。

定理 4. StreamSubsky 产生第一个结果的代价为 $O(|S_{max}| \ln |S_{max}|)$,产生全部结果的代价为 $O(|S_{max}| \times (\ln |S_{max}| + |Seed_{max}|) + |U_{max}| + (|U_{max}| \times r_{fp} + |\Delta Seed_{max}|) \times (|Seed_{cand}| + K^{D-d}))$ 。

5 实验验证与分析

实验对 StreamSubsky 算法从全空间 Skyline 维护、子空间 Skyline 计算以及内存空间消耗三方面进行测试,着重与相关的 SUBSKY 算法进行比较。SUBSKY 算法不是为数据流而设计的,对它进行了必要的适应性改造:支持对象的动态更新,选取

前 n 个对象为 anchor, 此后不改变. 在此不妨确定 n 为 15^[6]. 全空间 Skyline 维护测试还与文献[7]中的 LL-Eager 进行了比较, 以显示本算法的优越性以及剪枝策略的有效性. 算法由 VC++ 6.0 实现, 实验采用文献[7]提供的合成数据. 实验在 Windows XP 平台, 主频 2.0GHz 奔腾 4 处理器, 1GB 内存的配置上进行. 维护实验每次进行 100 次, 结果取平均值. 查询条件的构造则参考文献[6, 13], 即各种维度的子空间组合每种随机抽取 10 个, 不足的则重复选取, 结果取查询的平均时间. 测试默认设置: 相关分布全空间维度为 8, 而反相关分布则为 6; 窗口上数据集规模为 200K; 数据流速 200 元组/s. 由此前分

析可知, 算法性能与网格中格的数目有关. 关于最优的网格划分方案文献[16]作为类似的探讨, 本实验在 D 为 4~10 时, k 分别取 10, 6, 5, 4, 3, 3, 2.

5.1 全空间 Skyline 维护代价测试

维护代价测试从不同的数据维度、规模和流速三个角度进行. 图 10 是在两种分布下维度可扩展性的实验结果. 与 LL-Eager 相比, StreamSubsky 在各种情况下均表现较优, 时间消耗比前者要少一个数量级. 与 SUBSKY 相比在高维时稍差. 因为 SUBSKY 只是简单地将对象插入 B 树中, 不需作复杂的计算.

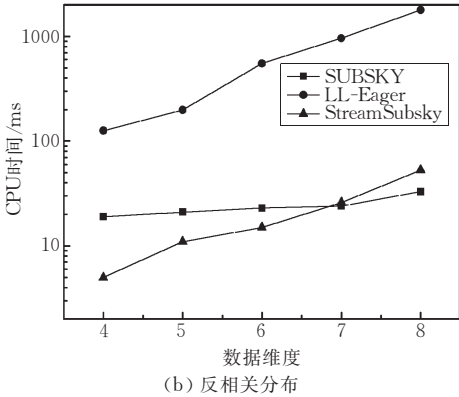
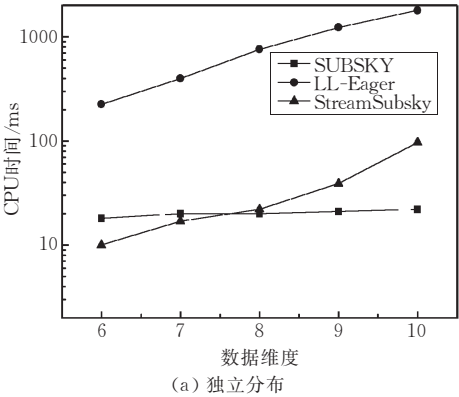


图 10 数据维度可扩展性

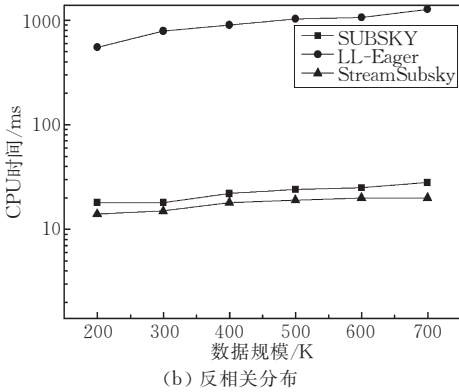
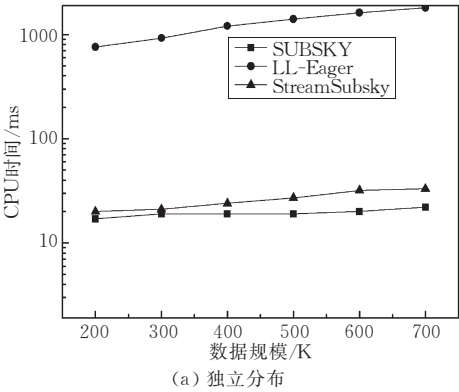


图 11 数据规模可扩展性

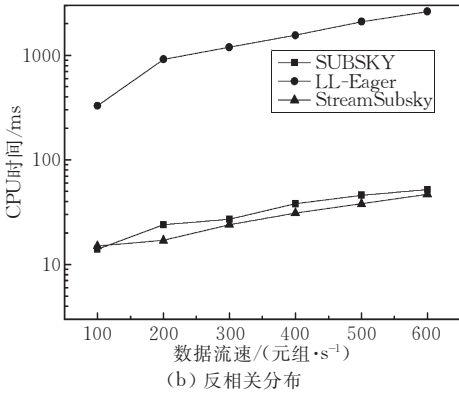
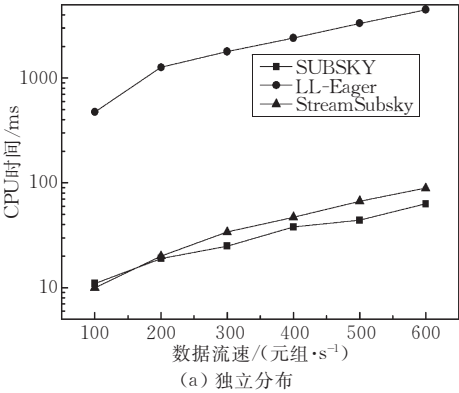


图 12 数据流速可扩展性

图 11 和图 12 分别是不同数据规模、不同流速下的实验结果. LL-Eager 的时间消耗是 StreamSubsky 的 15~20 倍,可见后者所采用的实现方案及剪枝策略的有效性. 与 SUBSKY 相比则性能差别不大,独立分布时 SUBSKY 略优,反相关分布时 StreamSubsky 略优.

5.2 子空间 Skyline 计算代价测试

从不同的查询条件维度、数据维度和数据规模三个方面测试子空间 Skyline 的计算代价. 主要与 SUBSKY 算法进行对比. 图 13 是查询条件维度不同的结果. SUBSKY 对查询条件维度很敏感,随

着维度的增加性能急剧下降. StreamSubsky 则受维度影响较小,并且以低于 SUBSKY 2~3 个数量级的代价就产生第一个输出;以使用其不到一半的时间输出全部结果. 分析其原因是因为 StreamSubsky 工作在一个小得多的数据集上. 全部时间与计算 Seed 时间之差即计算 $\Delta Seed$ 的时间,从图中看出花费时间也很少,说明了本文采用的启发式方法的有效性. 需要指出的是图 13(b)查询条件为 6 的情形,因为数据全空间维度为 6,故能直接将所维护的全空间上的结果直接输出. 图 14 和 15 在不同数据维度和数据规模下也能得到类似的结论.

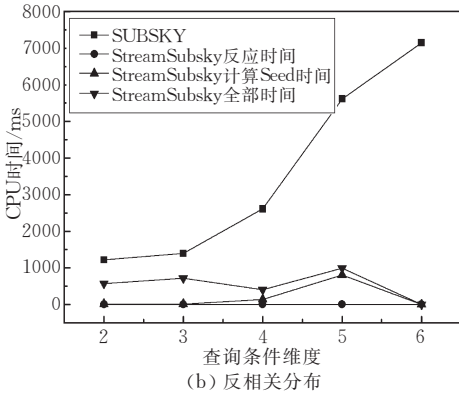
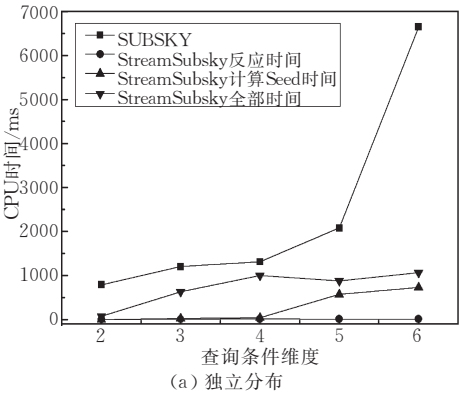


图 13 查询条件维度可扩展性

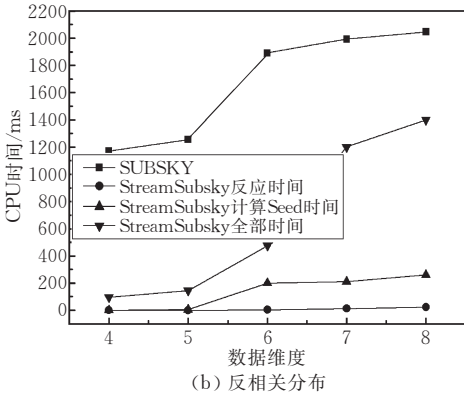
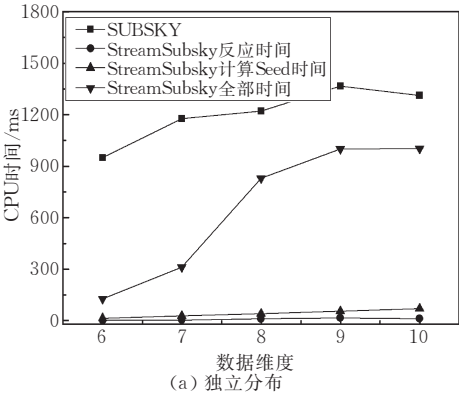


图 14 数据维度可扩展性

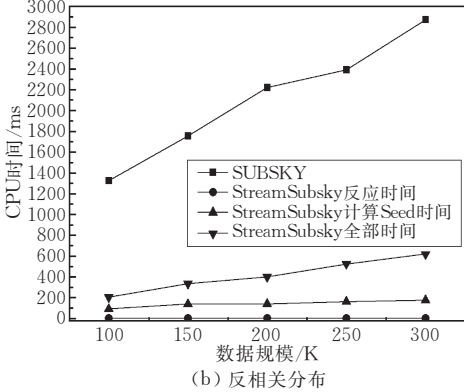
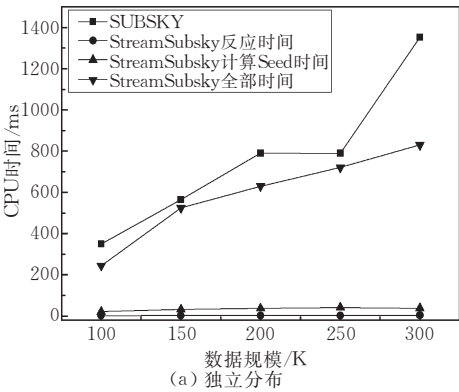
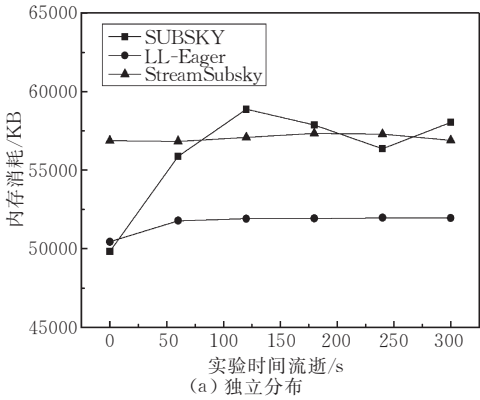


图 15 数据规模可扩展性

5.3 空间消耗测试

本节测试在不同的数据分布下各算法的空间消耗. 取各算法在初始化完毕后正式工作时随时间的流逝在不同的时间点上内存占用量. SUBSKY 空间消耗



有小许波动, LL-Eager 与 StreamSubsky 则相对平稳. 独立分布时 LL-Eager 占用空间最少, StreamSubsky 和 SUBSKY 次之. 反相关分布时则 StreamSubsky 占用空间最小, LL-Eager 和 SUBSKY 次之.

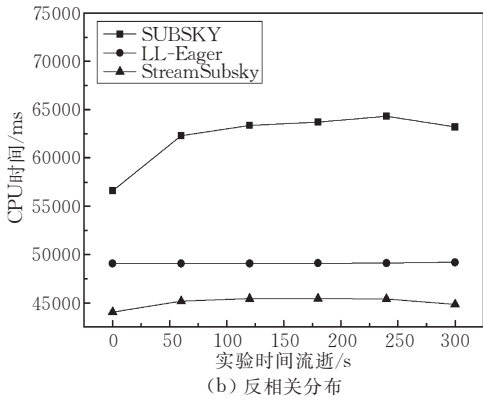


图 16 内存消耗

6 结 论

本文首次研究了具有重要实际应用意义的在数据流上计算子空间 Skyline 的问题. 提出了一个新颖高效的维护数据流 Skyline 的方法, 以此为基础通过两个阶段以增量的方式来返回目标子空间上的结果. 大量实验表明, 本文提出的算法具有较高的维护效率, 并能以极小的代价使查询得到响应, 以较少的时间输出全部结果, 算法具有良好的可扩展性和适应性且空间消耗较少. 下一步的研究方向是将问题扩展到分布式数据流的环境, 以支持基于水平划分的分布式数据流的环境.

致 谢 衷心地感谢为本文提供过帮助和指导的香港中文大学于旭(Jeffrey Yu Xu)和陶宇飞(Yufei Tao)博士!

参 考 文 献

[1] Borzsonyi S, Kossmann D, Stocker K. The skyline operator//Proceedings of the ICDE. Heidelberg, Germany, 2001: 421-430

[2] Chomicki J, Godfrey P, Gryz J, Liang D. Skyline with pre-sorting//Proceedings of the ICDE. Bangalore, India, 2003: 717-719

[3] Yuan Y, Lin X, Liu Q, Wang W, Yu J X, Zhang Q. Efficient computation of the skyline cube//Proceedings of the VLDB. Trondheim, Norway, 2005: 241-252

[4] Pei J, Jin W, Ester M, Tao Y. Catching the best views of skyline: A semantic approach based on decisive subspaces//Proceedings of the VLDB. Trondheim, Norway, 2005:

253-264

[5] Xia T, Zhang D. Refreshing the sky: The compressed sky-cube with efficient support for frequent updates//Proceedings of the ACM SIGMOD. Chicago, Illinois, USA, 2006: 491-502

[6] Tao Y, Xiao X, Pei J. SUBSKY: Efficient computation of skylines in subspaces//Proceedings of the ICDE. GA, USA, 2006: 65-74

[7] Tao Y, Papadias D. Maintaining sliding window skylines on data streams. IEEE Transactions on Knowledge and Data Engineering (IEEE TKDE), 2006, 18(3): 377-391

[8] Lin X, Yuan Y, Wang W, Lu H. Stabbing the sky: Efficient skyline computation over sliding windows//Proceedings of the ICDE. Tokyo, Japan, 2005: 502-513

[9] Buchta C. On the average number of maxima in a set of vectors. Information Processing Letters, 1989, 33(2): 63-65

[10] Guttman A. R-tree a dynamic index structure for spatial searching//Proceedings of the ACM SIGMOD. Boston, Massachusetts, USA, 1984: 47-57

[11] Theodoridis Y, Sellis T. A model for the prediction of R-tree performance//Proceedings of the ACM PODS. Montreal, Canada, 1996: 161-171

[12] Wang W, Yang J, Muntz R. STING: A statistical information grid approach to spatial data mining//Proceedings of the VLDB. Athens, Greece, 1997: 186-195

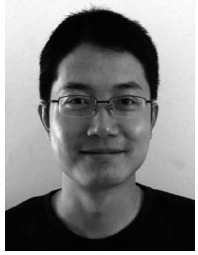
[13] Vlachou A, Doukeridis C, Kotidis Y, Vazirgiannis M. SKYPEER: Efficient subspace skyline computation over distributed data//Proceedings of the ICDE. Istanbul, Turkey, 2007

[14] Bloom B H. Space/Time trade-offs in hash coding with allowable errors. Communications of the ACM, 1970, 13(7): 422-426

[15] Fan L, Cao P, Almeida J, Broder A Z. Summary cache: A scalable wide-area web cache sharing protocol. Department of Computer Science, University of Wisconsin-Madison: Technical Report 1361, 1998

- [16] Mouratidis K, Bakiras S, Papadias D. Continuous monitoring of top- k queries over sliding windows//Proceedings of

the ACM SIGMOD. Chicago, IL, USA, 2005: 635-646



SUN Sheng-Li, born in 1979, Ph. D. candidate. His research interests focus on stream data mining and management, distributed and parallel data processing.

HUANG Zhen-Hua, born in 1980, Ph. D. candidate. His research interests focus on data mining and database

query optimization.

LI Jin-Jiu, born in 1979, M. S. candidate. His research interests focus on data mining and query processing.

GUO Jian-Kui, born in 1980, Ph. D. candidate. His research interests focus on query processing over data stream and Web data mining.

ZHU Yang-Yong, born in 1963, professor, Ph. D. supervisor. His research interests focus on data mining and bioinformatics.

Background

The work is supported by the National Basic Research Program (973 Program) of China under grant No. 2005CB321905: "Survivability Analysis Method of Massive Information System and Software Survivability Enhancement Technology". Survivability is the ability of a system to fulfill its mission in the presence of attacks, failures and accidents in a timely manner. Survivability is an increasing important dependability attribute for critical systems. The survivability of a system can be enhanced by predicting attacks then modifying the architecture of the system to provide enhanced resistance, recognition, recovery and adaptive capabilities. The research group has done a lot of researches in the past two years in the field including intrusion detection and intelligent decisions over data stream, several papers have been published in some international conferences, just like ACM SIGROUP and IEEE ICDM etc.

Skyline queries help users make intelligent decisions over multiple-dimensional data, where different and often conflicting criteria are considered. Most of the Skyline processing methods recently are based on fixed combination of dimen-

sions and static datasets. On one hand, applications that require Skyline analysis usually provide numerous candidate attributes, and various users may issue queries regarding different subsets of the dimensions depending on their interests. On the other hand object naturally occurs in the form of a stream of data values and needs real time, continuous processing. To the best of our knowledge, none of the previous works considered Skyline computation within subspaces over continuous data stream. SUBSKY is the best to compute subspace Skyline among previous works, but its delay of response time is too big to be accepted by the users. The authors develop a novel solution to handle the problem. They propose a much more efficient full space Skyline maintenance method leveraging grid index structure. Based on this, they present an efficient top-town two-phase algorithm to incrementally output the Skyline objects within specified subspaces. Moreover, they propose a set of optimizing heuristics to speed up the processing. Theoretical analysis and extensive experiments demonstrate that our methods are both efficient and scalable.