ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.20, No.7, July 2009, pp.1839−1853
doi: 10.3724/SP.J.1001.2009.03340

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

高效处理分布式数据流上 skyline 持续查询算法*

孙圣力+, 李金玖, 朱扬勇

(复旦大学 计算机与信息技术系,上海 200433)

Efficient Processing of Continuous Skyline Query over Distributed Data Streams

SUN Sheng-Li+, LI Jin-Jiu, ZHU Yang-Yong

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

+ Corresponding author: E-mail: slsun@fudan.edu.cn

Sun SL, Li JJ, Zhu YY. Efficient processing of continuous skyline query over distributed data streams. *Journal of Software*, 2009,20(7):1839−1853. http://www.jos.org.cn/1000-9825/3340.htm

Abstract: To reduce system delay and overall communication load, this paper proposes an efficient algorithm BOCS (based on the change of skyline) which is out of share-nothing strategy. BOCS is to solve this issue through progressive refinement by two steps. This paper provides analytical results and they show that BOCS is optimal in terms of the communication overhead among all algorithms which are out of share-nothing strategy. Theoretical analysis and extensive experiments demonstrate that these methods are efficient, stable and scalable.
Key words: distributed data stream; skyline; continuous query; communication optimal

摘 要: 基于非共享策略,围绕着降低系统反应延迟与通信负荷的目标,提出了一种分两阶段渐进求解的分布式算法 BOCS(based on the change of skyline),并对算法的关键实现环节,如协调站点与远程站点间的通信、skyline 增量的计算等进行了系统优化,使算法在通信负荷与反应延迟上达到了较好的综合性能.理论分析证明,在所有基于非共享策略的算法中,BOCS 算法通信最优.大量的对比实验结果也表明,所提出的算法高效、稳定且具有良好的可扩展性.
关键词: 分布式数据流;skyline;持续查询;通信最优
中图法分类号: TP311 **文献标识码**: A

分布式数据流广泛存在于环境监测、交通监控、网络通信以及金融股票交易等应用中.分布式数据流分析与挖掘是当前数据库与数据挖掘领域的研究热点.作为一种重要的数据挖掘技术,skyline 查询[1]近来引起了研究者们的广泛关注.考虑如下股票推荐系统[2]:存在某股票数据库,股票数据具有风险和佣金两个属性.现某用户浏览该数据库,试图从最近 3 天内到达的股票中挑选一支风险和佣金均尽可能低的股票进行投资.Skyline 查询为这类应用提供了一个可行的解决途径,它返回给定对象集中所有不被其他对象所支配的对象.这样,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对象.Skyline 查询对于多约束决策支持、城市导航以及用户偏好查询等具有重要意义.此前的大量工作[1,3,4]都专注于静态数据集上的 skyline 计算问题,近年来也逐渐出现了一些考虑集中式数据流上计算 skyline 的研究成果[2,5].然而,在更具挑战性的分布式数据流上处理

* Supported by 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 2007-08-13; Revised 2008-02-04; Accepted 2008-04-01

skyline 查询的成果未见于任何文献中.再考虑上述股票推荐系统,设想它是一个全球性的系统,由一个中央服务器与多个相互独立的远程服务器组成.分布在世界各地(如上海、纽约、法兰克福)的远程服务器监控本地的股票市场动态并相应地更新本地数据库,中央服务器与远程服务器进行通信交互以回答用户的查询请求.此类基于分布式数据流的事件监控与持续查询已经渗透到越来越多的应用之中[6−10].文献[6]首次提出了分布式数据流模型,它是指数据流来自多个水平分割的数据源,需要在各数据源产生的数据的并集上进行计算才能得到最终结果.分布式数据流体系结构如图 1 所示[9],系统由 M 个远程站点和一个中央协调站点组成.图 1 中的 $S_i(t_s)$是远程站点 $site_i$ 时间戳 t_s 上到达的对象集;$Skt_i(t_s)$是在时间戳 t_s 上 $site_i$ 向协调站点发送的概要数据结构, $Skt_i(t_s)$因算法不同而采取不同的方式构造.远程站点之间通常也能进行通信,但为了与以前的相关工作[8−10]保持一致,本文仅考虑远程站点仅能和协调站点进行双向通信的情形.本文致力于设计高效处理分布式数据流上 skyline 持续查询的算法,决定算法性能的主要因素是基本集中式算法的效率以及远程站点与协调站点间的通信效率.本文的主要贡献是:

1) 提出了在分布式数据流上处理 skyline 持续查询的问题,并给出了形式化描述;
2) 开发了一种具有较高整体性能的分布式处理算法 BOCS(based on the change of skyline),该算法实现了通信效率与反应延迟的合理均衡,并且在所有采取非共享策略的算法中,该算法通信最优;
3) 进行了大量严格而细致的实验,实验结果表明,BOCS 算法高效、稳定,具有良好的可扩展性,对数据分布具有较高的鲁棒性.

本文第 1 节阐述背景知识及相关工作.第 2 节对此前的相关工作进行扩展,给出一种基准算法.第 3 节阐述 BOCS 算法的设计思想及其优化策略.第 4 节进行算法的代价分析.第 5 节给出对比实验结果.第 6 节对全文工作进行总结.

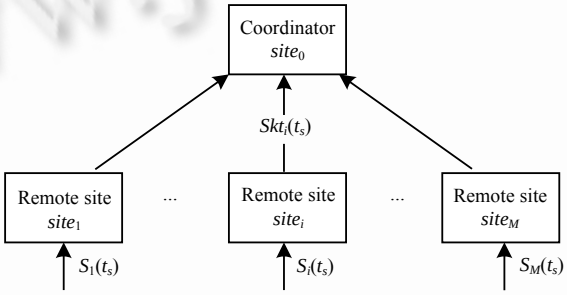


Fig.1 Distributed stream monitoring architecture

图 1 分布式数据流处理体系结构

1 背景知识

1.1 相关工作

自分布式数据流模型提出以来,分布式数据流上的查询处理与挖掘逐步成为研究热点.特别是近 3 年以来研究成果大量涌现,这些工作包括:在分布式数据流上计算近似分位数[9]、估计数据集的基(F_0)[10]、计算数据集的吉尼系数(F_2)[8]、统计恰好出现特定频次的对象数[11]、挖掘分布式数据流中的频繁项[12]、分部式数据流上 Top-k 元素监控[13]以及分布式数据流上的聚类分析[7]等.Skyline 查询在数据库领域最早由 Borzsonyi 等人在文献[1]中提出,此外,该文还提出了 BNL(block nested loops)和 DC(divide and conquer)两种算法:BNL 算法采用迭代的方法将对象进行逐一对比,每次迭代产生一组 skyline 对象;DC 算法则采用递归的方法进行,将整个数据集划分为若干个内存能容纳的子块,先分别计算各子块上局部的 skyline,最后将局部结果合并得到全局结果.此后,涌现出大量适用于不同场合的更高效的算法,比较有代表性的是依赖索引结构的 BBS(branch and bound skyline)算法[4]和基于排序的 SFS(sort first skyline)算法[3].在 BBS 算法中,对象采用 R-树[14]来组织,然后采用分

支定界的方法来逐步输出数据集上的 skyline,BBS 算法被证明为 I/O 最优.SFS 算法则采用先排序后比对的方法来计算 skyline.在集中式数据流上处理 skyline 的研究成果主要有文献[2,5].处理该问题的关键有两点:一是当前窗口上的 skyline 对象过期后,如何直接输出其"继任者"而避免计算其排它支配域上的 skyline;二是高效地求新到对象的影响时间与淘汰被新来对象所支配的对象.针对问题 1,文献[2]采用事件链(event list)机制而文献[5]则利用支配图来解决.但对于问题 2,文献[2,5]所采用的方法效率较低.某对象 p 的影响时间是其加入 skyline 的时间,求影响时间的关键是找到落在 p 的反支配区域(记为 ADR)内的对象的最大时间戳;淘汰 p 所支配的对象则是指将 p 的支配区域(记为 DR)内的所有对象从系统中删除.文献[2]将对象按链表(或采用 R-树索引结构)来组织,再直接依靠 D-向搜索(D-sided search)[14]来实施上述操作.在链表上搜索理论上需要穷尽整个链表,维度稍高时在 R-树上进行搜索也是相当低效的:维度较高时,组成 R-树的 MBR(minimum bounding rectangle)之间会产生较严重的相互重叠,导致在 R-树上搜索的急剧下降[15].文献[5]存在的问题与此类似.本文提出的 BOCS 算法建立在更高效的集中式算法[16]上,该集中式算法采用网格索引和一系列优化策略显著地提高了问题 2 的效率.与本文工作比较接近的还有文献[17],它提出了在传感器网络中计算 skyline 的算法 SWSMA(sliding window skyline monitoring algorithm),该算法基于共享策略开发了元组过滤和网格过滤等方法来减少对象的传输数.SWSMA 算法仅追求较高的通信效率,不考虑系统反应延迟.该算法中,当过滤元组(或网格)过期需要重新选定时,会产生大量的通信负荷,并需要进行大量的计算,从而使系统产生严重的反应延迟.故 SWSMA 算法应用在传感器网络上,而并不适用于高速的分布式数据流环境.本文提出的 BOCS 算法则在反应延迟和通信效率上达到了较合理的均衡,具有较高的综合性能.

1.2 术语与问题定义

令 $\mathcal{A}=\{A_1,A_2,\ldots,A_D\}$ 是一组有界并有序的域,$\mathcal{O}=A_1\times A_2\times\ldots\times A_D$ 是一个 D 维空间.称 $A_1,\ldots,A_D$ 为空间 $\mathcal{O}$ 的属性或维.不失一般性,本文假设 $\forall A_i\in\mathcal{A}$ 的定义域均为实数区间(0,1].考虑一个集合 $U=\{u_1,u_2,\ldots,u_n\}$,其中任何 u_i 均来自空间 $\mathcal{O}$.将 u_i 在属性 A_i 上的取值记为 $u_i.val_i$.

定义 1(支配). 任意给定两个对象 $p,q\in U$.如果对于 $\forall A_i\in\mathcal{A}$,有 $p.val_i\le q.val_i$,且 $\exists A_j\in\mathcal{A}$,有 $p.val_j<q.val_j$,则称 p 支配 q,记为 $p\prec q$.若 p 不支配 q,则记为 $p\not\prec q$.

定义 2(skyline). 集合 U 中所有不被其他对象所支配的对象组成的集合称作 U 上的 skyline,记为 SKY.形式化地,$SKY=\{p\in U|(\forall q\in U)(q\not\prec p)\}$.

本文主要考虑数据流中基于时间的滑动窗口模型[18],这样,任意对象 p 就可以记为 $p(p.t_{arr},p.val_1,\ldots,p.val_D)$,其中,$p.t_{arr}$ 表示其到达系统的时间戳(不妨设时间戳为自 0 开始递增的整数),$p.val_i$ 为其在属性 $A_i\in\mathcal{A}$ 上的取值.不妨假设滑动窗口的宽度为 W,即各对象在系统中的生命周期跨度为 W 个时间戳.如果 p 到达系统的时间戳为 t_{arr},则其在区间 $[p.t_{arr}+1,p.t_{arr}+W]$ 上是活动的,而它在时间戳 $p.t_{arr}+W+1$ 上过期而失效.

定义 3(时间戳 skyline). 将在某时间戳 $t_s(t_s\ge 0)$ 上到达的对象集记为 $S(t_s)$.集合 $S(t_s)$ 上的 skyline 称为时间戳 skyline,记为 $TS(t_s)$.

滑动窗口上的 skyline 会随着时间的推移而演化,skyline 实际上是一个快照.将 $t_s(t_s\ge 1)$ 上的 skyline 快照记为 $SKY(t_s)$.若 $t_s\ge W$,则 $SKY(t_s)$ 是对象集 $S(t_s-W)\cup\ldots\cup S(t_s-1)$ 上的 skyline;若 $1\le t_s<W$,则它是对象集 $S(0)\cup\ldots\cup S(t_s-1)$ 上的 skyline.本文考虑分布式环境,将时间戳 t_s 上到达远程站点 $site_i$ 的对象集记为 $S_i(t_s)$,将 $site_i$ 时间戳 t_s 上的 skyline 快照记为 $SKY_i(t_s)$,将基于全局数据流的 skyline 快照记为 $SKY_0(t_s)$.本文要解决的问题是,随着数据流中对象的不断到达与过期,动态持续地维护全局数据流上的 skyline 快照 $SKY_0(t_s)$.对于基于计数的滑动窗口模型[18],可以给每个到达的对象关联一个虚拟时间戳(时间戳可用该对象在数据流中的次序号充当),这样,本文提出的算法就能直接地应用于基于计数的数据流环境.表 1 给出本文中常用的符号及其意义.

Table 1 List of symbols

表 1 本文所用的符号及说明

Symbol	Brief description
W	Width of sliding window
D	Dimensionality of data object
S_i	Local stream at remote $site_i$
$S_i(t_s)$	Set of objects arriving at remote $site_i$ on time t_s
$S_0=\cup_i S_i$	Global stream of the whole system
$Skt_i(t_s)$	Sketch sent to coordinator by remote $site_i$ on time t_s
$SKY_i(t_s)$	Skyline over stream of remote $site_i$ on time t_s
$SKY_0(t_s)$	Skyline over global stream on time t_s
$TS_i(t_s)$	Timestamp skyline of remote $site_i$ on time t_s

2 基准算法

作为对比,本节给出一种基准算法.将此前已知的集中式算法(Eager 或 Lazy[2])进行少许扩展,即可以设计出一种简单的适用于分布式数据流环境的算法 BODS(based on direct sending).该算法如下设计:远程站点简单地监控本地的数据流,不作任何处理,直接将其监测到的对象发送到协调站点.协调站点收集各远程站点发送来的对象,时间戳发生跳变即调用集中式算法(Eager 或 Lazy)进行维护处理.这样,图 1 中远程站点向协调站点传送的概要结构 $Skt_i(t_s)$就等于 $S_i(t_s)$.进一步考察 $S_i(t_s)$中的对象,可以发现一些有趣的性质,基于此便能设计出效率更高的算法.处理分布式数据流上 skyline 持续查询,除了基本的正确性要求之外,本文提出以下目标:

(1) 降低系统的反应延迟.在高速的数据流环境中,降低算法处理每一对象的平均 CPU 时间,对提高系统的吞吐率非常关键.此外,还要保证系统具有较高的稳定性,避免采用可能使性能出现较大波动的处理策略,以提高系统的服务质量.

(2) 降低通信量.减少通信量对于减少网络拥塞、提高分布式系统的性能非常关键.在特定的分布式环境下,例如在无线传感器网络中,要尽可能地延长系统的使用寿命,就必须使用消耗能量尽可能少的通信协议[19].

目标 1 是数据流算法的内在要求,而目标 2 则体现了分布式算法的特点.以上设计目标可能会相互冲突,如何达到两者的合理均衡,则是算法设计的关键.

3 分布式算法 BOCS

本节将围绕着第 2 节提出的两个目标对 BODS 进行优化,最终得到本文提出的解决方案.第 3.1 节从减少通信量的角度出发,先后设计出了基于传输时间戳 skyline 的 BOTS(based on timestamp skyline)算法和基于传输 skyline 增量的 BOCS 算法;第 3.2 节从减少系统反应延迟的角度出发,采用更高效的基本维护算法,并对 BOCS 算法的关键实现环节进行优化,进一步提高了系统的性能;第 3.3 节对算法进行详细描述,并给出一个实例加以说明.

3.1 通信负载优化

提高通信效率就是要在保证系统具有较低反应延迟的前提下,尽可能地减少不必要的对象传输.BODS 算法所发送的对象集 $S_i(t_s)$中存在不少冗余对象,考察其中的对象,可以得到以下引理.

引理 1. 时间戳 t_s 上到达远程站点 $site_i$ 的任意对象 p,若 $p\notin TS_i(t_s)$,则 p 在其整个生命周期内都不属于 $site_i$ 上的 skyline.

证明:已知 $p\in S_i(t_s), p\notin TS_i(t_s)$,由时间戳 skyline 的定义,则一定$\exists q\in S_i(t_s): q\prec p$.而 q 与 p 具有相同的时间戳,则在 p 的整个生命周期内都存在支配 p 的对象.由 skyline 的定义,该引理得证. □

由引理 1 进一步得出以下定理:

定理 1. 时间戳 t_s 上到达远程站点 $site_i$ 的任意对象 p,若 $p\notin TS_i(t_s)$,则 p 在其整个生命周期内都不会属于整个分布式系统上的 skyline.

根据定理 1 可以设计出一种通信效率较高的基于传输时间戳 skyline 的算法 BOTS.该算法的基本步骤如图 2 所示,$BOTS\_Remote_i$ 和 $BOTS\_Coord$ 是分别配置在远程站点 $site_i$ 和协调站点上的处理模块.远程站点缓存本地上一个时间戳上到来的对象.时间戳发生跳变即计算时间戳 skyline(可以调用 SFS 算法[3]实施),完毕后发送到协调站点.协调站点接收各远程站点发送来的时间戳 skyline,然后调用集中式算法对全局 skyline 进行更新维护.这样,图 1 中的概要结构 $Skt_i(t_s)$就等于 $TS_i(t_s)$.

进一步考察数据集 $TS_i(t_s)$中的对象可以发现,即使某对象 $p\in TS_i(t_s)$,p 在其整个生命周期内也不一定会出现在 $site_i$ 上的 skyline 之中.例如,假设当前时间戳为 t_s,$\exists p\in TS_i(t_s)$,若此时在 $site_i$ 上$\exists o\in TS_i(t_{s'}):o\prec p$,其中,$t_s-W\le t_{s'}<t_s$,则 p 不能立即成为 skyline 对象,p 被置为候选 skyline 对象(未来可能会成为 skyline 对象)缓存起来.若此后在区间$[t_s+1,t_{s'}+W+1]$上到来某对象 s:$s\prec p$,则 p 不可能再加入 skyline,其被 s 从系统中淘汰.这样,p 在其整个生命周期内都没有机会加入 skyline.故时间戳 skyline 中仍然存在冗余对象,下面提出一个更高效的解决方案.

对 BODS 算法进行优化,还可以进一步从时间轴上来考虑,下面给出引理.

引理 2. $\exists p\in S_i$,若 p 在其整个生命周期内都不属于站点 $site_i$ 上的 skyline,则 p 在其整个生命周期内也不会属于全局 skyline.

证明:已知 $p\in S_i$,p 在其整个生命周期内都不属于 $site_i$ 上的 skyline.由 skyline 的定义,则在 $site_i$ 上在 p 的整个生命周期内都存在支配 p 的对象.而 $S_0=\cup_i S_i$,则在 p 的整个生命周期内在 S_0 中存在支配 p 的对象.所以 p 在其整个生命周期内都不会属于全局 skyline.引理得证. □

根据引理 2 并受文献[17]的启发,可以设计出一种通信效率很高的算法(based on the change of skyline,简称 BOCS).该算法的基本思想是:采取分两阶段的方式渐进求解,即远程站点先计算出本地局部 skyline,再将本地结果传送给协调站点,协调站点据此维护全局 skyline.算法的基本步骤如图 3 所示.各远程站点缓冲本地当前时间戳上到达的对象,时间戳发生跳变即调用集中式算法对本地 skyline 进行更新维护.维护完毕后计算出与上一时间戳相比本地 skyline 的增量,最后将增量发送到协调站点.协调站点,接收并合并各远程站点发送来的 skyline 增量,再调用集中式算法更新维护全局 skyline.图 3 给出了 BOCS 算法的简要描述,$BOCS\_Remote_i$ 和 $BOCS\_Coord$ 分别部署在远程站点 $site_i$ 和协调站点上.这样对于 BOCS 来说,图 1 中的 $Skt_i(t_s)$就等于$\Delta SKY_i(t_s)$.

$BOTS\_Remote_i$
1. Save all objects observed on $t_{current}-1$ in BF_i;
2.If timestamp changes, $TS_i(t_{current}-1)\leftarrow$compute timestamp skyline over BF_i;
3. Send($TS_i(t_{current}-1)$) to coordinator.

$BOTS\_Coord$
1. $BF_0\leftarrow\cup_i TS_i(t_{current}-1)$;
2. Maintain global skyline;

Fig.2 Brief description of algorithm BOTS
图 2 BOTS 算法描述

$BOCS\_Remote_i$
1. Save all objects received on $t_{current}-1$ in BF_i;
2. If timestamp changes, maintain local skyline $SKY_i(t_{current})$;
3. $\Delta SKY_i(t_{current})\leftarrow SKY_i(t_{current})-SKY_i(t_{current}-1)$;
4. $Send(\Delta SKY_i(t_{current}))$.

$BOCS\_Coord$
1. $BF_0\leftarrow\cup_i\Delta SKY_i(t_{current})$;
2. Maintain global skyline.

Fig.3 Brief description of algorithm BOCS
图 3 BOCS 算法描述

3.2 计算负荷优化

本节围绕着第 2 节中的目标 1,在采用更高效的基本集中式算法的基础上对 BOCS 算法的关键实现环节进一步进行优化,以尽可能地降低系统的反应延迟.正如第 1.1 节所分析的那样,文献[2,5]中提出的集中式算法在计算对象的影响时间与淘汰被支配对象的问题上存在明显的缺陷.我们在文献[16]中提出了解决该问题的方法:采用网格索引来组织对象,并开发多个启发式剪枝策略来提高算法的效率.该算法的具体细节参见文献[16],这里不再赘述.但该算法并不能直接应用于本文所考虑的问题中,需要进行较大的改进才能为 BOTS 和 BOCS 算法所采用.BOTS 中协调站点上所采用的集中式算法与 BOCS 中远程站点上所采用的集中式算法相一致,下面主要阐述 BOCS 所采用的集中式算法.

不失一般性,图 4 和图 5 给出了空间维度为 2 时 BOCS 算法分别部署在远程站点和协调站点上的集中式算法的结构描述.先考察部署在远程站点上的算法.数据流中所有活动对象由一个列表(active object list,简称

AOL)来保存,新到达的对象在队尾插入,过期对象从队头删除.活动对象根据其不同特性进一步分为 skyline 对象、候选 skyline 对象和无效对象(不属于前两者的对象).本文称前两者为有效对象,而无效对象将不会保留在系统中.采用网格索引,每个格关联 2 个指针列表,分别用来保存本格所属 skyline 和候选 skyline 对象的地址,算法结果即由前者体现.依据事件链机制进行过期处理.部署在协调站点上的算法(如图 5 所示)与远程站点上的(如图 4 所示)相比,不同点在于活动对象(或有效对象)的组织方法不同:图 4 采用简单的队列,而图 5 采用哈希表来实现.这是因为远程站点发送来的是 skyline 增量$\Delta SKY_i(t_s)$,而$\Delta SKY_i(t_s)$中的各对象可能持有不同的时间戳.为了提高算法的灵活性与减少插入新到对象和删除过期对象时搜索 AOL 的工作量,故将 AOL 以哈希表的方式来实现.对于计算新到达对象的影响时间、删除被新来对象所支配的对象以及过期处理等细节详见第 3.3 节.

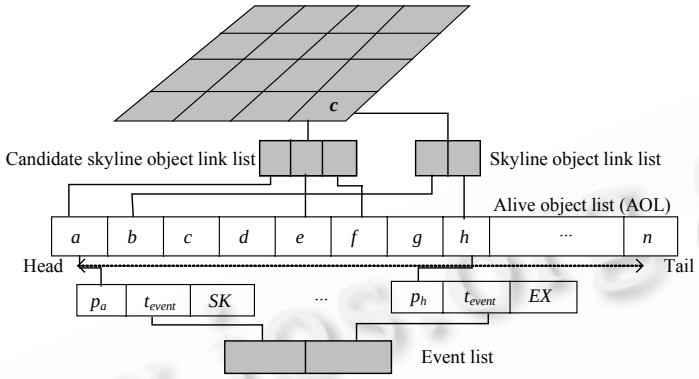


Fig.4 Data structure of algorithm on remote site

图4 远程站点上集中式算法的数据结构

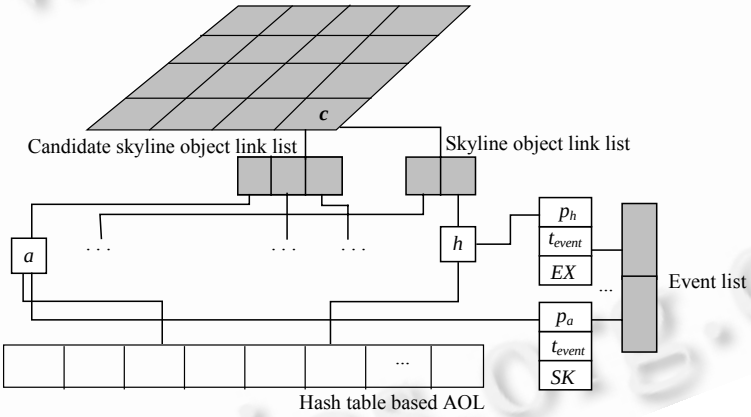


Fig.5 Data structure of algorithm on coordinator

图 5 协调站点上集中式算法的数据结构

除了采用更高效的集中式算法,下面进一步对 $BOCS\_Remote_i$ 的关键细节进行优化.考虑图 3 中 $BOCS\_Remote_i$ 的第 3 步:如果直接采用两集合中元素一一对比的方式来计算两集合之差,这将产生 $O(n^2)$的开销.在此,我们采用更巧妙的方法来解决这一问题.考察某对象变为 skyline 对象而得以进入集合$\Delta SKY_i(t_s)$的时机,只存在两种可能:(1) 处理 SK(skyline)事件,候选 skyline 对象升格为 skyline 对象;(2) 新时间戳上到达的对象不为其他任何有效对象所支配,则其立即成为 skyline 对象.以上两个时机分别出现在后文图 9 所示的过期处理模块中和图 10 所示的集中式算法主程序中.我们将这两个时机出现的对象保存在一个列表(delta skyline list,简称 DSL)中,其每个成员皆为对象的地址.当某对象 p 变为 skyline 对象时,将 e 加入到 DSL 中(e 代表 p 在 AOL 中的

地址).此外,还需要为每个活动对象新设置一个属性 *ptr*,*ptr* 指向 DSL 中 *p* 所对应的成员(即 *e* 在 DSL 中的地址,若 *p* 的地址没有被加入到 DSL 中,则 *p.ptr*=*null*).依据以上结构就可以很容易地在维护 skyline 的同时,顺便将 $\Delta SKY_i(t_{current})$作为副产品计算出来.具体细节见下节所述.这时,计算$\Delta SKY_i(t_s)$所发生的开销就降为 $O(n)$.

3.3 算法细节及示例

本节给出 BOCS 算法部署在远程站点上的集中式算法的细节.除了$\Delta SKY_i(t_{current})$的计算以外,协调站点上的集中式算法与此类似,不再给出.BOCS 算法所采用的集中式算法由 3 个模块组成(如图 6 所示).算法的工作流及各模块的合作情况简述如下:新到达的对象缓存在 BF(buffer)中.时间戳发生跳变,即调用模块 *procExpObj* 依据事件链进行过期处理,即将 EX(expiration)事件关联的对象从系统中淘汰,将 SK 事件所关联的对象迁入到 skyline 对象列表.接着,调用模块 *computeInfTime* 计算 BF 中对象的影响时间,依据不同的影响时间,置其为不同类型的对象.若其为有效对象,则插入到 AOL 中,然后将其在 AOL 中的地址插入到其所属格的相应列表(skyline 或候选 skyline 列表)中.最后,调用模块 *procDomObj* 将当前窗口上被新来对象所支配的对象淘汰出局.下面图 7~图 10 给出以上 3 个模块 *computeInfTime*,*procDomObj* 和 *procExpObj* 以及集中式算法主程序的详细描述.

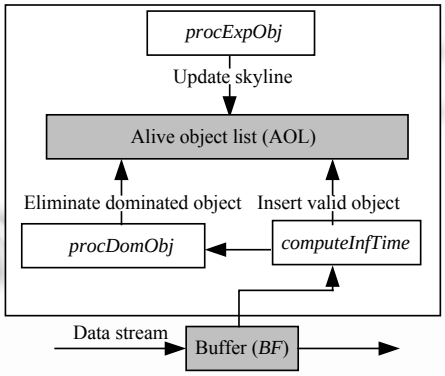


Fig.6 Architecture of centralized algorithm

图 6 集中式算法模块结构

```
Procedure computeInfTime
Input: p //the object to compute Influence Time
1  t_lowerbound←−1, t_upperbound←Maximum Timestamp;
2  Initialize t_upperbound and t_lowerbound;
3  for each cell c partially covered by p.ADR
4     o←valid object holding the largest timestamp
      in c, if both queues are empty returns ∅;
5     while o<>∅
6        if o.timestamp≤t_lowerbound,   break;
7        if o.timestamp<t_upperbound
8           if o dominate p t_lowerbound←o.t_arr; break;
9        o←valid object holding the next largest
         timestamp in c, if both queues are empty
         returns ∅;
10 Return t_lowerbound
```

Fig.7 Procedure of computing influence time

图 7 计算影响时间的过程

```
Procedure procDomObj
Input: p, DSL, t_inf // t_inf is result of ComputeInfTime
1 for each object o located in cells that totally covered by
  p.DR
2    if o.ptr<>null, delete the object pointed by o.ptr in
     DSL;
3    Delete o and its associate item from AOL and EL
     respectively;
4 for each cell c partially covered by p.DR
5    o← valid object holding the largest timestamp in c,
     if both queues are empty returns ∅;
6    while o<>∅
7       if o.t_arr>t_inf
8          if p dominate o
9             if o.ptr<>null, delete object pointed by o.ptr in
              DSL;
10            Delete o and its associate item in EL;
11         o← valid object holding the largest timestamp in
           c, if both queues are empty returns ∅;
12      else break;
```

Fig.8 Procedure of processing dominated objects

图 8 处理被支配对象的过程

```
Procedure procExpObj
Input: DSL
1 t←Current Timestamp;
2 for each event Item in EL triggered by t
3    p: the object pointed by Item.pointer;
4    if Item.tag='EX' //if p go expired
5       Delete p from AOL;
6       Delete Item from EL;
7    else //p becomes a skyline object
8       p is transferred to skyline array of the cell that p located
        in;
9       e←Item.pointer, e is added to DSL, p.ptr←the address
        of e within DSL;
10      Item.tag←'EX'; Item.timestamp←p.t_arr+W+1;
11      Item is inserted into EL;
```

Fig.9 Procedure of expiration processing

图 9 过期处理过程

```
Algorithm BasicCentralizedAlgorithm
1 DSL←∅;
2 Call procExpObj(DSL); //expiration processing
3 for each object p in BF
4    call computeInfTime(p) to obtain p's influence time t_inf;
5    if t_inf<>p.t_arr
6       if t_inf=−1   //there is no object dominating p
7          p is added to skyline array of the cell that p located in;
8          e←the address of p, e is added to DSL, p.ptr←the
           address of e within DSL;
9          ⟨(a pointer to p)p,p.t_arr+W+1,'EX'⟩ is added to EL;
10      else
11         p is added to the candidate skyline array;
12         one new item ⟨p,t_inf+W+1,'SK'⟩ is added to EL;
13      call procDomObj(p,DSL,t_inf);
```

Fig.10 Main program of centralized algorithm

图 10 基本集中式算法主程序

以下给出具体的例子,对 BODS,BOTS 和 BOCS 算法的通信过程加以说明,并描述在 BOCS 算法中 skyline 增量是如何作为副产品被计算出来的.

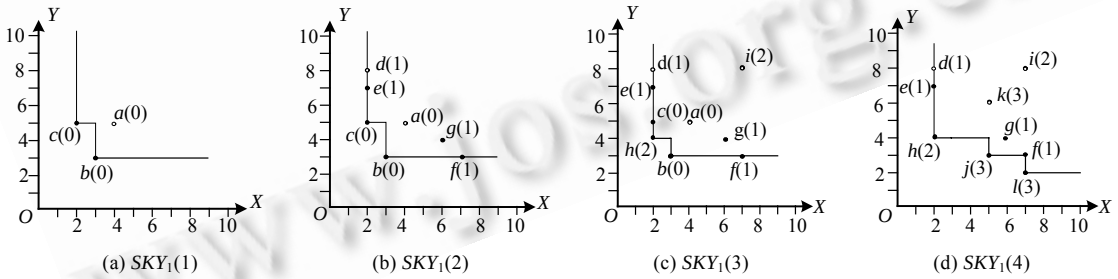
例 1:为了简化说明,假设系统中除了协调站点外仅有一个远程站点 $site_1$.W 为 3,D 为 2.表 2 给出了 $site_1$ 时间戳 0~3 上的对象到达情况.图 11 是远程站点 $site_1$ 不同时间戳上的 skyline 快照,为了方便说明,将被较后到来的对象支配而须从系统中淘汰出去的对象仍然保留在图中.某对象若属于其到达时间上的时间戳 skyline,则用实心点表示,否则用空心点表示,扩号中的数字代表对象到达系统的时间.

Table 2 An example data set

表 2 一个示例数据集

Time	0			1				2		3		
Object	*a*	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*	*j*	*k*	*l*
X	4	3	2	2	2	7	6	2	7	5	5	7
Y	5	3	5	8	7	3	4	4	8	3	6	2

先考察 BODS 算法,它发送本站点上到达的全部对象,总通信量为 12.BOTS 算法计算时间戳 skyline,仅将时间戳 skyline 进行发送.时间戳 0,1,2 和 3 上的时间戳 skyline 分别是{*b*,*c*},{*e*,*f*,*g*},{*h*}以及{*j*,*l*},故总通信量为 8.BOCS 算法仅将 skyline 增量进行发送,由图 11 可知,站点 $site_1$ 上时间 1,2,3 和 4 上的 skyline 快照分别是{*b*,*c*},{*b*,*c*},{*h*,*b*}和{*h*,*j*,*l*}.Skyline 增量$\Delta SKY_1(1)$~$\Delta SKY_1(4)$分别为{*b*,*c*},∅,{*h*}和{*j*,*l*},总通信量为 5.BOCS 与 BOTS 相比,少发送了 *e*,*f*,*g* 等 3 个对象.



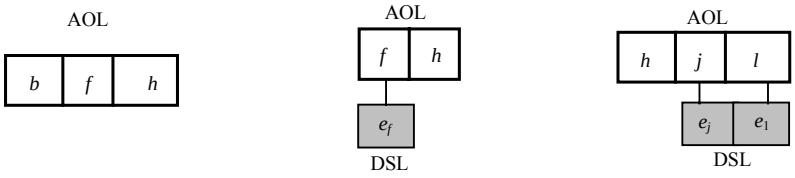
(a) $SKY_1(1)$ (b) $SKY_1(2)$ (c) $SKY_1(3)$ (d) $SKY_1(4)$

Fig.11 Skyline snapshot over different times on $site_1$

图 11 $site_1$ 上不同时间戳上的 skyline 快照

下面再给出 $BOCS\_Remote_1$ 构造$\Delta SKY_1(4)$的过程.在时间戳 3 时,$site_1$ 上的有效对象为{*b*,*f*,*h*}(图 11(c)、图 12(a)所示).时间戳 4 到来调用模块 *procExpObj* 进行过期处理,触发所有以 4 为触发时间的事件:*b* 在时间戳 4 上过期,将其从活动对象列表中删除;*f* 由候选 skyline 对象升格为 skyline 对象,在 DSL 中加入一项 e_f(即指向 AOL

中对象 f 的指针),再置 $f.ptr$ 为 DSL 中 e_f 的地址.经过该过程后的 skyline 增量由图 12(b)所示.接着进行后续维护工作,对象 j 支配 f,将 f 和 e_f 分别从 AOL,DSL 中除去.另外,j 不为任何对象所支配,故在以上两列表中分别加入 j 和 e_j;同理也加入 l 和 e_1.图 12(c)是 $BOCS\_Remote_1$ 维护处理完毕后 DSL 的状态,DSL 中的内容刚好对应 $\Delta SKY_1(4)$.



(a) AOL of timestamp 3 (b) $\Delta SKY_1(4)$ after *procExpObj* (c) Final $\Delta SKY_1(4)$

(a) 时间戳 3 时的 AOL (b) *procExpObj* 后的 $\Delta SKY_1(4)$ (c) 最终的 $\Delta SKY_1(4)$

Fig.12 Description of ΔSKY construction

图 12 ΔSKY 构造示意图

4 算法分析

本节首先对 BOCS 算法的正确性进行证明,然后着重从通信负载的角度对 BODS,BOTS 和 BOCS 进行分析,最后给出 BOCS 的通信最优性质.以下定理给出算法的正确性保证.

定理 2. 算法 BOCS 输出结果是正确且是完备的.

证明:假设 $SKY_0(t_s)$ 是 t_s 时在任何正确算法下协调站点上的输出,假设 $SKY_c(t_s)$ 是 BOCS 算法下协调站点上的输出,这样只需证明 $SKY_c(t_s)=SKY_0(t_s)$.

先进行正确性证明,只需证明 $\forall r\in SKY_c(t_s): r\in SKY_0(t_s)$.采用反证法.假设 $\exists r\in SKY_c(t_s)$,而 $r\notin SKY_0(t_s)$,则 r 必被某远程站点 $site_i$ 上的某一活动对象 p 所支配.在 $site_i$ 上存在两种可能性:

(1) $\exists p'\in SKY_i(t_s), p'\prec p$;

(2) $p\in SKY_i(t_s)$,$SKY_i(t_s)$ 是远程站点 $site_i$ 上 t_s 时的 skyline 快照.

在此,不妨另设一个符号 q,如果第(1)种情况成立,则用 q 代替 p';若第(2)种情况成立,则用 q 代替 p.显然,$q\prec r$.根据 BOCS 的设计,q 一定会出现在 $site_i$ 的输出中且被发送到协调站点.在协调站点上也存在两种可能:

(1) $\exists q'\in SKY_c(t_s), q'\prec q$;

(2) $q\in SKY_c(t_s)$.

在此,不妨再用 o 代替 q'(第(1)种情况成立时)或 q(第(2)种情况成立时),显然,$o\prec r$.根据 BOCS 的设计,协调站点在时间戳 t_s 上不可能输出对象 r 为 skyline 对象,这与假设矛盾.所以正确性得证,故有 $SKY_c(t_s)\subseteq SKY_0(t_s)$.

然后进行完备性证明,只需证明 $\forall r\in SKY_0(t_s)$,均能由 BOCS 输出,即 $r\in SKY_c(t_s)$.假设某 $r\in SKY_0(t_s)$,不妨设 r 来自远程站点 $site_i$.因为 $r\in SKY_0(t_s)$,所以在 t_s 上,整个系统中都不存在支配 r 的活动对象,故 $r\in SKY_i(t_s)$.由算法设计,r 一定会出现在 $site_i$ 的输出中且被发送到协调站点.还因为 $r\in SKY_0(t_s)$,协调站点上也不可能存在支配 r 的活动对象,故在 t_s 时,r 必定出现在协调站点输出的 skyline 中.完备性得证,故又有 $SKY_0(t_s)\subseteq SKY_c(t_s)$.

综上所述,$SKY_c(t_s)=SKY_0(t_s)$,定理得证. □

设在 BOCS,BOTS 和 BODS 算法下远程站点 $site_i$ 截止至时间戳 N(时间戳 N 刚开始)时发送的对象集分别为 U_i,V_i 和 W_i,整个分布式系统发送的对象集为 $U(U=\cup_i U_i)$,$V(V=\cup_i V_i)$ 和 $W(W=\cup_i W_i)$.由算法设计,则 W,V 和 U 的规模由以下式子给出:

$$|W|=\sum_{i=1}^{M}|W_i|=\sum_{i=1}^{M}\left(\sum_{j=0}^{N-1}|S_i(t_j)|\right) \tag{1}$$

$$|V|=\sum_{i=1}^{M}|V_i|=\sum_{i=1}^{M}\left(\sum_{j=0}^{N-1}|TS_i(t_j)|\right) \tag{2}$$

$$|U|=\sum_{i=1}^{M}|U_i|=\sum_{i=1}^{M}\left(\sum_{j=1}^{N}|\Delta SKY_i(t_j)|\right) \tag{3}$$

其中,公式(3)中的$\Delta SKY_i(t_j)=SKY_i(t_j)-SKY_i(t_{j-1})$.分析以上 3 种算法所发送的对象集,可以得出以下定理:

定理 3. 若 BOCS,BOTS 和 BODS 算法下整个系统发送的对象集分别为 U,V 和 W,则关系 $U\subseteq V\subseteq W$ 成立.

证明:因为 $U=\cup_i U_i$,$V=\cup_i V_i$,$W=\cup_i W_i$.只需证明在任意远程站点 $site_i$ 上,$U_i\subseteq V_i\subseteq W_i$ 成立.关系 $V_i\subseteq W_i$ 是非常直观的,在此着重证明 $U_i\subseteq V_i$.假设 $r\in U_i$,则在算法 BOCS 下,远程站点 $site_i$ 在时间戳 t_s 上向协调站点发送了对象 r,即 $r\in\Delta SKY_i(t_s)$.而$\Delta SKY_i(t_s)=SKY_i(t_s)-SKY_i(t_s-1)$,所以 $r\in SKY_i(t_s)$.不妨设 r 到达 $site_i$ 的时间为 $t_{s'}(t_s-W\le t_{s'}\le t_s-1, t_s\ge W$,这里的 W 为滑动窗口的宽度),由引理 1 有 $r\in TS_i(t_{s'})$,则在算法 BOTS 下,$site_i$ 在时间戳 $t_{s'}$ 上一定会将 r 向协调站点发送,即 $r\in V_i$.所以有 $U_i\subseteq V_i$,定理得证. □

就通信量而言,BOCS 不仅比 BODS 和 BOTS 算法更优,实际上,在不共享其他远程站点任何信息的前提下,不存在比 BOCS 更优的算法.下面给出定理.

定理 4. 在所有采用非共享策略的算法中,BOCS 通信开销最小.

证明:只需证明 BOCS 算法下,系统传送的每一个对象都是必要的.假设在时间戳 t_s 上,某远程站点 $site_i$ 向协调站点发送了对象 r.由 BOCS 的设计可知,$r\in\Delta SKY_i(t_s)$.而$\Delta SKY_i(t_s)=SKY_i(t_s)-SKY_i(t_{s-1})$,所以 $r\in SKY_i(t_s)$.即在时间戳 t_s 上,$site_i$ 中不存在支配 r 的任何活动对象.在不共享其他远程站点上的任何信息的前提下,一定要将 r 发送到协调站点上,如果不发送,则会出现如下后果:在时间戳 t_s 时,除 $site_i$ 外,在其他任何远程站点上也不存在支配 r 的活动对象,则 $r\in SKY_0(t_s)$.在这种情况下,若不将 r 发送到协调站点上,则导致结果不完备甚至可能会出现错误的输出.所以一定要将 r 进行发送,不然无法保证输出结果的完备性与正确性.因此在采用非共享策略的情况下,BOCS 算法传送的每一个对象都是必要的,它不发送任何冗余的对象.BOCS 通信最优性得证. □

还可以从负载均衡的角度对以上 3 种算法进行分析.BODS 算法下,各远程站点只承担对象的监控与发送工作,计算任务都集中在协调站点上.另外,协调站点还需要规模最大的缓冲区,该算法下数据与查询负载的分配都最不均衡.BOTS 算法下,各远程站点需要计算时间戳 skyline,协调站点负责维护工作;而 BOCS 算法下远程站点与协调站点都要进行 skyline 维护操作.BOTS 和 BOCS 算法的查询负载较为均衡,而 BOCS 算法的数据负载最为均衡.如果单纯地从降低通信量的角度来考虑,还可以采用共享信息的策略对 BOCS 算法进一步优化,但这可能会导致严重的系统延迟,下节中与 SWSMA[17]进行的对比实验可以验证这一点.而 BOCS 做到了反应延迟和通信效率较合理的均衡.在此需要指出的是,BODS 算法和 BOCS 算法不仅适用于基于时间的滑动窗口模型,也适用于基于计数的滑动窗口模型,而 BOTS 算法则仅适用于前者.实际上,在基于计数的滑动窗口模型下就通信负载而言,BOTS 退化成为 BODS 算法.

5 实验评价

本文提出的算法由 VC++6.0 实现,实验在 Windows XP 平台,主频 2.0G 奔腾 4 处理器和 1G 内存的配置上进行,采用文献[2]提供的合成数据(包括独立分布与反相关分布数据)进行实验.着重测试在不同的数据分布、维度与规模之上算法的效率、敏感性与可扩展性(scalability).主要考虑基于时间的滑动窗口数据流模型,在本节的所有实验中,滑动窗口的宽度(W)设定为 1 000,每一个时间戳代表 1s.实验参数默认值:数据维度(D)为 6,各远程站点上的数据规模(N)为 300K.实验参数的变化区间:D 为 4~8,而 N 为 200,300,500,800,1 000(K).当给定数据规模时,数据流速(rate)就能直接计算出来.例如 N 为 300K,则 $rate$ 为 300 对象/秒.每组实验执行 100 次,结果取平均值.BOCS 和 BOTS 所采用的集中式算法的性能与网格粒度相关,下面第 5.1 节首先测试网格粒度对 BOCS 所采用的集中式算法性能的影响.第 5.2 节接着再从通信负载、时间开销以及性能的稳定性这 3 个角度全面地对 BODS, BOTS,BOCS 和 SWSMA 算法进行对比.

5.1 网格粒度的影响

本节测试 K 值(格索引中每维的等份数)对 BOCS 和 BOTS 所采用的集中式算法性能的影响,试图求出最佳的 K 值.在本节的实验中,基本参数配置为 D=6,N=1M.我们将 K 值由 1 逐步增长到 9,图 13 给出了在不同 K 值

和数据分布情况下算法处理每个对象的平均 CPU 时间.

由图 13 所示,无论独立还是反相关分布数据集,所对应的曲线都类似于一条抛物线:刚开始时,随着 K 值的增大,CPU 开销显著下降,当到达最低点后,开销开始反弹.其原因是因为过于稀疏的网格将导致单个格中存在过多的对象,而过于精细的网格将导致访问格本身的开销激增.当 D=6、K 由 1 增长到 9 时,网格索引中格的数目由 1 增加到 9^6.在保持较低的 CPU 开销的前提下,网格索引所消耗的空间越小越好.其他维度下的实验结果与此类似.综合考虑,维度 4~8 时我们所选定的 K 值分别为 4,3,3,2,2,它们将应用于后续的实验中.因此可见,网格索引的空间消耗非常之小.

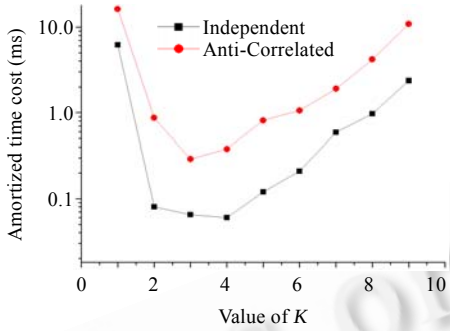


Fig.13 Performance under different K

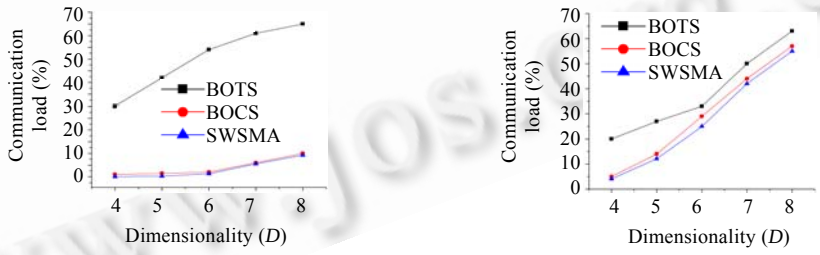
图 13 不同 K 值下的算法性能

5.2 算法性能测试

本节对 BOCS 算法着重从通信负载、时间开销与算法稳定性这 3 个面进行测试,主要与 BOTS,BODS 以及 SWSMA 进行对比.BODS 采用文献[2]中就 CPU 开销来说表现最为优秀的 I-Lazy 方法作为基本维护算法;SWSMA 采用自适应的过滤方式实现并加以优化.在模拟的分布式系统中存在 5 个远程站点和 1 个协调站点,分别部署在 6 台计算机上.协调站点与远程站点间的通信带宽设为 56KB/s.

5.2.1 通信负载测试

本文以百分比的形式来衡量通信负载:即分别以 BOCS,BOTS 和 SWSMA 算法之下 100 个时间戳内全部远程站点向协调站点发送的对象总数与 BODS 下发送的对象总数之比来表示.对于 SWSMA 算法,还将其过滤器比特数折合成对象数.第 1 组实验考察算法的维度可扩展性,将 D 由 4 逐步增长到 8,图 14 给出了在不同维度下各算法的通信代价.



(a) Independent
(a) 独立分布数据集

(b) Anti-Correlated
(b) 反相关分布数据集

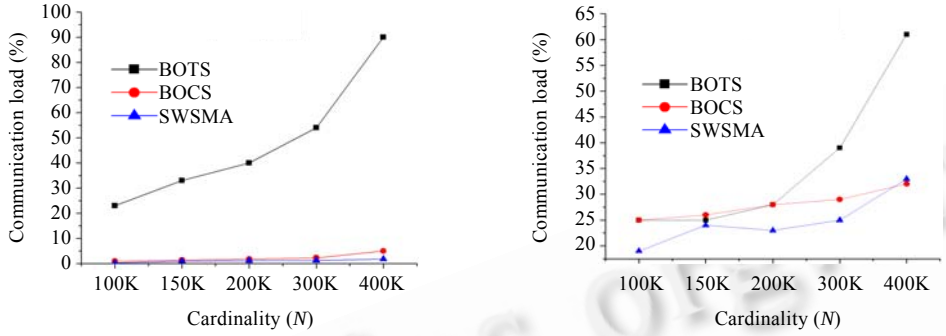
Fig.14 Communication-Overhead scalability on dimensionality

图 14 通信负载的维度可扩展性

如图 14(a)所示,在独立分布数据集上,BOTS 产生的通信量是基准算法 BODS 的 30%~65%,而 BOCS 算法产生的通信量是 BODS 的 3%~10%,这与 SWSMA 算法大体相当.图 14(b)是在反相关分布数据上的结果,BOTS

和 BOCS 产生的通信量则分别是 BODS 的 20%~70%与 5%~60%,而 SWSMA 算法则要比 BOCS 低 1~3 个百分点.总的来说,BOCS 产生的通信量比 BODS 少得多,也要明显低于 BOTS,这与第 4 节的理论分析相一致.

在第 2 组实验中,我们将 *rate* 由 100 逐步增长到 400 对象/秒,则各远程站点上的数据规模由 100K 增长到 400K.图 15 给出了不同数据规模下各算法的通信代价.无论是在相关分布还是反相关分布数据集上,BOCS 与 BOTS 相比都具有更好的可扩展性.而 SWSMA 对 BOCS 的优势也仅在反相关分布数据集上略有体现,究其原因是因为:独立分布数据集上的 skyline 规模相当小,过滤器发挥作用的空间非常有限;而在反相关分布数据集上,skyline 规模较大,留给过滤的空间相对较大.



(a) Independent
(a) 独立分布数据集

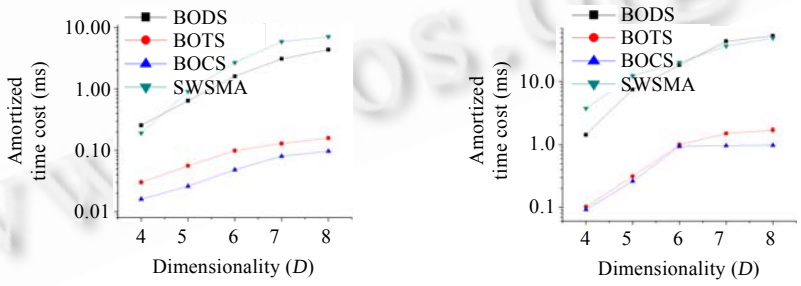
(b) Anti-Correlated
(b) 反相关分布数据集

Fig.15 Communication-Overhead scalability on cardinality

图 15 通信负载的规模可扩展性

5.2.2 时间开销测试

本节测试算法的反应延迟.在基于时间的滑动窗口模型下,BOCS 与 BOTS 算法的反应延迟包括 3 部分:远程站点上算法消耗的时间、通信过程消耗的时间以及协调站点上集中式算法消耗的时间.BODS 算法的反应延迟仅包含协调站点上的 I-Lazy 算法所消耗的时间,因为远程站点不作任何维护操作而传输过程可以在 I-Lazy 启动之前完成.SWSMA 算法与 BOCS 算法相似,也包括 3 部分,另需将其过滤器的维护开销分摊到每个对象中去.本节中,算法的反应延迟以分摊的时间开销即系统处理每一个对象的平均 CPU 开销来衡量.在第 1 组实验中,将对象的维度由 4 逐步增长到 8,图 16 给出了在不同维度下各算法的结果.无论是在独立分布还是反相关分布的数据集下,BOCS 与 BODS 和 SWSMA 相比都具有更好的可扩展性.当维度由 4 增长到 8 时,BOCS 对 BODS 和 SWSMA 的优势也由 1 个数量级逐步增长到近 2 个数量级.BOTS 的对 BODS 和 SWSMA 的优势也很明显,但不如 BOCS.



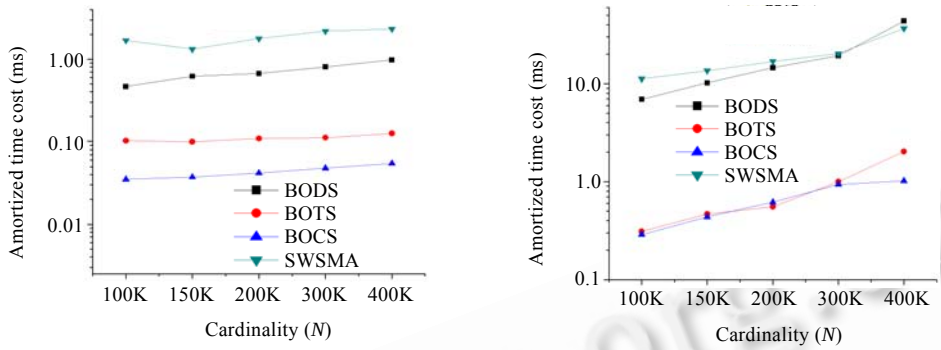
(a) Independent
(a) 独立分布数据集

(b) Anti-Correlated
(b) 反相关分布数据集

Fig.16 Processing-Time scalability on dimensionality

图 16 处理时间的维度可扩展性

在第 2 组实验中,将 *rate* 由 100 逐步增长到 400 对象/秒,则各远程站点上的数据规模由 100K 增长到 400K.图 17 给出了不同数据规模下各算法的时间开销.无论是在独立分布还是反相关分布的数据集上,BOCS 对 BODS 和 SWSMA 的优势都在 1~2 个数量级左右,这也与图 16 反映的情况大体一致.各算法在独立分布数据集上与反相关数据集相比,均具有更好的可扩展性.需要指出的是,在基于计数的滑动窗口模型下,BODS 算法的反应延迟应包括两部分:协调站点上 I-Lazy 所消耗的时间以及通信过程所消耗的时间.因此,在基于计数的滑动窗口模型下,BOCS 与 BOTS 算法对 BODS 算法的优势会更加明显.



(a) Independent
(a) 独立分布数据集

(b) Anti-Correlated
(b) 反相关分布数据集

Fig.17 Processing-Time scalability on cardinality

图 17 处理时间的规模可扩展性

5.2.3 算法稳定性测试

BOCS 算法基于 skyline 增量来设计,增量的规模是可变的;而 SWSMA 算法中过滤器失效后需要重新计算,这可能会导致通信和计算负荷的波动.在此,有必要对以上两种算法的稳定性进行验证与对比,即测试其通信负载与 CPU 开销是否存在着冲突.本节的实验采用默认设置.第 1 组实验,当算法充分预热之后(从 0 开始经历 1 000 个时间戳之后),我们记录下连续 300 个时间戳上各算法所产生的通信量,结果如图 18 所示(与第 5.2.1 节相同,采用百分比形式表达).先考察 BOCS 算法:在独立分布数据集上,其产生的通信量在 1%~3%的区间内波动,在反相关分布数据集上的波动区间则是 22%~28%.再考察 SWSMA 算法:在独立分布数据集上,其产生的通信量主要在 0.5%~2%的区间内波动,在反相关分布数据集上,主要在 20%~27%区间上波动.但在 SWSMA 算法中,则会周期性地出现明显的离群点,而在 BOCS 中则不存在.这是因为,在 SWSMA 中,当过滤器更新时,一方面需要将远程站点上大量的对象传送到协调站点,另一方面还需要将新设置的过滤器广播到各远程站点上,这将产生巨大的通信量.而在数据规模比较大的情况下,skyline 增量的规模则变化得不是很明显,故 BOCS 算法的通信要稳定得多.第 2 组实验考察 CPU 开销,当算法充分预热之后,我们记录连续 300 个时间戳上各算法处理每个对象的平均时间,结果如图 19 所示.与图 18 类似,SWSMA 算法的结果中存在着离群点,BOCS 的结果中则不存在.这是因为 SWSMA 的过滤器更新时需要进行大量的计算,而 BOCS 中不会出现类似情况.总之,无论是在独立分布还是反相关分布的数据集上,也无论是从通信还是 CPU 开销的角度来看,算法 BOCS 的稳定性都要高得多,这决定了 BOCS 算法能够提供更可靠的服务.

6 结 论

Skyline 查询是近来数据库与数据挖掘领域的研究热点,它返回给定数据集上所有不被其他对象所支配的对象.此前的相关工作都是针对静态数据集或集中式数据流而设计的.近年来涌现的大规模分布式应用激发了基于分布式数据流的 skyline 查询处理的研究.本文提出了一种高效处理分布式数据流上 skyline 持续查询的算法 BOCS.它建立在更高效的集中式算法上,在 BOCS 中仅将 skyline 增量进行传输,并采取了一系列措施对算法

的实现进行优化.理论分析证明,在所有采用非共享策略的算法中,BOCS 通信最优.此外,BOCS 实现了通信负载和反应延迟的合理折衷,算法具有较好的综合性能.详细的对比实验结果表明,BOCS 算法高效、稳定且具有良好的可扩展性.

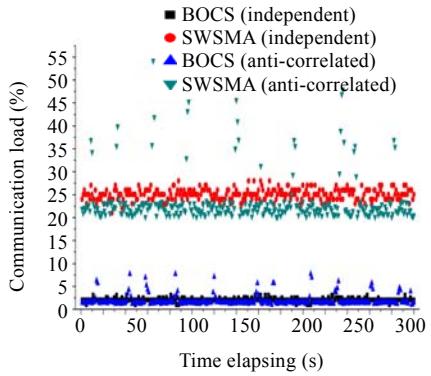


Fig.18 Communication load stability verification

图 18 通信负载稳定性验证

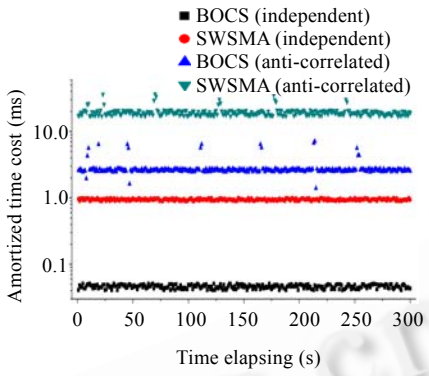


Fig.19 CPU load stability verification

图 19 CPU 负载稳定性验证

致谢 衷心地感谢香港中文大学陶宇飞(Yufei Tao)博士及其研究团队为我们提供相关源代码及其实验数据.

References:

[1] Borzsonyi S, Kossmann D, Stocker K. The skyline operator. In: Proc. of the 17th Int'l Conf. on Data Engineering. Washington: IEEE Computer Society, 2001. 421−430. http://www.fernuni-hagen.de/ICDE/D-2001/

[2] Tao YF, Papadias D. Maintaining sliding window skylines on data streams. IEEE Trans. on Knowledge and Data Engineering, 2006, 18(3):377−391.

[3] Chomicki J, Godfrey P, Gryz J, Liang D. Skyline with presorting. In: Proc. of the 19th Int'l Conf. on Data Engineering. Washington: IEEE Computer Society, 2003. 717−719. http://www.ouhk.edu.hk/HK2003/

[4] Papadias D, Tao YF, Fu G, Seeger B. Progressive skyline computation in database systems. ACM Trans. on Database Systems, 2005,30(1):41−82.

[5] Lin X, Yuan Y, Wang W, Lu H. Stabbing the sky: Efficient skyline computation over sliding windows. In: Proc. of the 21st Int'l Conf. on Data Engineering. Washington: IEEE Computer Society, 2005. 502−513. http://icde2005.is.tsukuba.ac.jp/

[6] Gibbons P, Tirthapura S. Estimating simple functions on the union of data streams. In: Proc. of the ACM Symp. on Parallel Algorithms and Architectures. Crete Island, 2001. 281−291. http://www.cs.dartmouth.edu/SPAA/2001/

[7] Zhou A, Cao F, Yan Y, Sha C, He X. Distributed data stream clustering: A fast EM-based approach. In: Proc. of the 23rd Int'l Conf. on Data Engineering. Washington: IEEE Computer Society, 2007. 736−745. http://www.icde2007.org/icde/

[8] Cormode G, Garofalakis M. Sketching streams through the net: Distributed approximate query tracking. In: Bohm K, Jensen CS, Haas LM, Kersten ML, Larson P, Ooi BC, eds. Proc. of the 31st Int'l Conf. on Very Large Data Bases. New York: ACM Press, 2005. 13−24.

[9] Cormode G, Garofalakis M, Muthukrishnan S, Rastogi R. Holistic aggregates in a networked world: Distributed tracking of approximate quantiles. In: Proc. of the ACM SIGMOD Int'l Conf. on Management of Data. New York: ACM Press, 2005. 25−36. http://cimic.rutgers.edu/~sigmod05/

[10] Das A, Ganguly S, Garofalakis M, Rastogi R. Distributed set-expression cardinality estimation. In: Nascimento MA, Tamer M, Kossmann D, Miller J, Blakeley A, Schieferl KB, eds. Proc. of the 30th Int'l Conf. on Very Large Data Bases. San Francisco: Morgan Kaufmann Publishers, 2004. 312−323.

[11] Cormode G, Muthukrishnan S, Rozenbaum I. Summarizing and mining inverse distributions on data streams via dynamic inverse sampling. In: Bohm K, Jensen CS, Haas LM, Kersten ML, Larson P, Ooi BC, eds. Proc. of the 31st Int'l Conf. on Very Large Data Bases. New York: ACM Press, 2005. 25−36.

[12] Manjhi A, Shkapenyuk V, Dhamdhere K, Olston C. Finding (recently) frequent items in distributed data streams. In: Proc. of the 21st Int'l Conf. on Data Engineering. Washington: IEEE Computer Society, 2005. 767−778. http://icde2005.is.tsukuba.ac.jp/

[13] Babcock B, Olston C. Distributed top-*k* monitoring. In: Proc. of the ACM SIGMOD Int'l Conf. on Management of Data. New York: ACM Press, 2003. 28−39. http://db.ucsd.edu/sigmodpods03/

[14] Guttman A. R-Tree: A dynamic index structure for spatial searching. In: Proc. of the ACM SIGMOD Int'l Conf. on Management of Data. New York: ACM Press, 1984. 47−57. http://www.sigmod.org/

[15] Theodoridis Y, Sellis T. A model for the prediction of R-tree performance. In: Proc. of the 15th ACM SIGACT-SIGMOD-SIGART Symp. on Principle of Database Systems. New York: ACM Press, 1996. 161−171. http://www.sigmod.org/

[16] Sun SL, Huang ZH, Li JJ, Guo JK, Zhu YY. Efficient computation of subspace skylines over data streams. Chinese Journal of Computers, 2007,30(8):1418−1428 (in Chinese with English abstract).

[17] Xin J, Wang G, Chen L, Zhang X, Wang Z. Continuously maintaining sliding window skylines in a sensor network. In: Ramamohanarao K, Krishna PR, Mohania MK, Nantajeewarawat E, eds. Proc. of the 12th Int'l Conf. on Database Systems for Advanced Applications. Berlin: Springer-Verlag, 2007. 509−521.

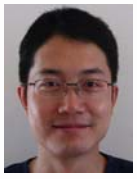
[18] Jin CQ, Qian WN, Zhou AY. Analysis and management of streaming data: A survey. Journal of Software, 2004,15(8):1172−1181 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/15/1172.htm

[19] Madden S, Franklin MJ, Hellerstein JM, Hong W. The design of an acquisitional query processor for sensor networks. In: Proc. of the ACM SIGMOD Int'l Conf. on Management of Data. New York: ACM Press, 2003. 491−502. http://db.ucsd.edu/sigmodpods03/

附中文参考文献:

[16] 孙圣力,黄震华,李金玖,郭建奎,朱扬勇.数据流上高效计算子空间 Skyline 的算法.计算机学报,2007,30(8):1418−1428.

[18] 金澈清,钱卫宁,周傲英.流数据分析与管理综述.软件学报,2004,15(8):1172−1181. http://www.jos.org.cn/1000-9825/15/1172.htm



孙圣力(1979－),男,湖南澧县人,博士生,CCF 学生会员,主要研究领域为基于流数据的查询处理与数据挖掘,分布式数据库.

李金玖(1979－),男,硕士生,主要研究领域为数据挖掘与知识发现,查询优化.

朱扬勇(1963－),男,博士,教授,博士生导师,主要研究领域为数据挖掘,生物信息学,数据库.