



基于 MapReduce 的连续 Skyline 查询^{*}

单观敏,董一鸿,何贤芒

(宁波大学信息科学与工程学院 宁波 315211)

摘 要:随着 Web 网络和无线传感器网络的不断发展,大数据的出现对数据查询和处理产生了挑战。研究了 MapReduce 框架下移动对象的并行 Skyline 计算问题,采用基于角度划分的方法解决负载平衡问题,并提出了一种过滤策略进行剪枝提高计算效率,在此基础上分别实现了静态 Skyline 算法和基于事件跟踪的 MR-Track 算法。最后通过对比实验验证了算法的有效性。

关键词: Skyline 查询; MapReduce; Hadoop; 移动对象; 连续查询

doi: 10.3969/j.issn.1000-0801.2014.05.014

Continuous Skyline Queries Based on MapReduce

Shan Guanmin, Dong Yihong, He Xianmang

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

Abstract: With the development of Web and wireless sensor networks, the appearance of big data has brought a great challenge to data query and process. The parallel Skyline computing issue of a moving query object under the MapReduce framework was studied. To solve the problem of load balance, a method based on angle partition was adopted. Then a filtering method was introduced to improve the efficiency. On the basis of those methods, a static Skyline algorithm and an event tracking algorithm named MR-Track were implemented respectively. Finally, many experiments were made to verify the effective and efficient of MR-Track algorithm.

Key words: Skyline query, MapReduce, Hadoop, moving object, continuous query

1 引言

Skyline 查询作为多目标决策应用的一种重要方式已经在数据库社区获得了广泛而深入的研究。在 Skyline 查询中,数据点 p 支配数据点 q ,当且仅当 p 在任一维上的取值都不比 q “差”且至少在一个维度上比 q 更“好”(Skyline 查询中的“好”和“差”是由用户的喜好而定的)。Skyline 查询返回的是不被其他数据对象支配的数据对象。例如,手机用户想要找到一个超市去购物,假设影响用户选择的有两个因素:价格和距离。显然不同超市的商品价格不会完全相同;由于用户是处于匀速直线的步行中,因

此用户到这些超市的距离会发生变化。如图 1 所示, $a \sim f$ 为用户位于初始位置时各个超市的属性坐标,从图中可以看出 a 不可能被用户选择,因为 d 在所有属性上要优于 a ,同样 c 和 f 也不在用户选择范围内,因此 Skyline 集合为 $\{b, d, e\}$ 。当用户移动到另一个位置时,所有数据点的距离属性发生了变化, $a' \sim f'$ 为各点变化后的坐标,此时没有其他点在所有属性上优于 b 和 c ,因此 Skyline 集合变为 $\{b, c\}$ 。

Skyline 查询最早是由 Borzsonyi 等人^[1]在 ICDE 2001 会议上提出的,此后 Skyline 查询成为数据库和数据挖掘的研究热点。虽然不同形式的 Skyline 查询已经被深入研究,但是随着网络技术的不断发展,大数据的出现对传统的

^{*} 国家自然科学基金资助项目(No.61202007),宁波市自然科学基金资助项目(No.2013A610063)

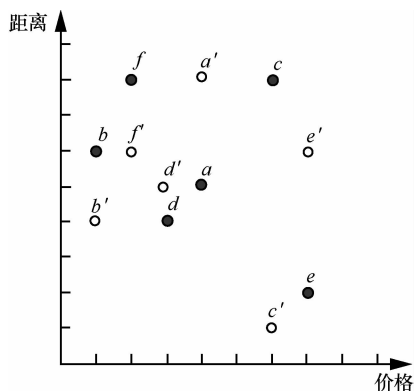


图1 Skyline点举例

Skyline 查询产生了挑战。MapReduce^[2]作为云计算的一种重要分布式并行框架,能够满足对海量数据集的处理。目前基于 MapReduce 的 Skyline 查询还处于初步研究状态,而对于移动环境下的连续 Skyline 查询更是未涉及,因此本文的工作将着重于查询点匀速直线移动、其他数据点静止情形下的 Skyline 查询问题。

2 相关工作与研究现状

2.1 传统 Skyline 查询

Borzsanyi 等人^[1]首次将 Skyline 操作引入数据库系统,提出了 BNL 和 D&C 两种算法。BNL 算法的优点是对各种数据分布、各种大小的数据集,都可以直接应用而不需要对数据进行任何索引或预处理。参考文献[3]将原始数据集编码成位图(bitmap),提出了一种基于索引(index)的方法。Kossmann 等^[4]基于 R-tree 索引结构提出的 NN 算法。Papadias 等^[5]分析了 NN 方法在 I/O 和存储方面存在的不足并对其进行了改进,提出了 BBS 算法。

随着集中式 Skyline 研究的深入,分布式环境下的 Skyline 查询也进入研究领域。Balke 等^[6]首次提出了垂直分布数据库中用轮询的方式按序访问各个划分找到“锚点”,将在“锚点”之前未被访问过的点删除,再访问后续的点并根据支配关系更新“锚点”和 Skyline,从而提出了 BDS 算法和 IDS 算法。Cui 等^[7]在分布式环境下研究了 Skyline 查询处理,对 MBR 按“不可比较关系”进行分组,将过滤点和计划发送给后续站点,同时将 Skyline 集反馈给组头,进而提出了 PaDSkyline 算法。

除了集中式和分布式环境,其他形式的 Skyline 查询如移动环境下的 Skyline 查询研究和并行 Skyline 计算等也是研究热点。Huang 等人^[8]研究了查询点移动情况下的 Skyline 问题,提出了连续跟踪算法 CSQ,首先计算出初始

时刻的 Skyline,计算下一时刻造成 Skyline 集合变化的事件,使用查找边界有效减少了不必要的计算。Tian 等人^[9]研究了数据点移动而查询点不动情况下的 Skyline 查询,使用网格划分空间,算法只需动态地维护受影响区域和 Skyline 集合。Fu 等人^[10]研究了位置不确定移动对象的 Skyline 查询,提出一种不确定移动对象连续概率 Skyline 查询的 U-CPSC 算法。Vlachou 等^[11]从数据划分的角度研究了并行计算 Skyline 的问题,提出了基于角度的划分,并且针对不同的数据分布提出了格划分、等量划分和动态划分。Park 等^[12]研究了多核架构下的并行 Skyline 计算问题,通过对 R-tree 中同父节点的节点进行并行计算提出了并行 BBS 算法,同时基于“划分合并”的思想并行地对 Skyline 合并计算实现了 pskyline 算法。Kohler 等^[13]提出了超平面映射划分的方法并行计算 Skyline,与基于角度的划分不同,超平面映射将各数据点映射到超平面上再进行划分数据集。

2.2 MapReduce 框架下的 Skyline 查询

MapReduce 框架最早是由 Dean 等^[2]提出的。MapReduce 作为一种编程模型,用于大规模数据集的并行运算。在 MapReduce 框架中,任务主要有两个阶段——map 和 reduce。worker 节点功能主要是执行不同阶段的计算任务;主节点的主要职责是将用户提交的任务分派给各个 worker 节点,并负责监控和控制 worker 节点的任务执行状态信息。MapReduce 计算模型的基本流程如下:数据文件(input file)被分成若干个文件块,这些文件块的每条记录形成键值对(key, value)被 map 阶段的 worker 节点(又称 slave 节点,从节点)处理生成中间键值对,所有的中间剪枝对根据键值被散列函数分到不同的组中(相同键值的键值对分到一组),reduce 阶段的 worker(slave)一组组读取键值对并进行处理,最后生成输出文件(output file)。

当前基于 MapReduce 的 Skyline 研究还处于初步阶段。张波良等^[14]首次研究了 MapReduce 框架下海量数据集的 Skyline 查询问题,提出了 MR-BNL 算法、MR-SFS 算法和 MR-Bitmap 算法。MR-BNL 算法和 MR-SFS 算法将数据空间以网格形式划分成若干等份,计算出各个部分的局部 Skyline,再合并计算出全局 Skyline。MR-Bitmap 算法预计计算出每一维的全局二进制位串,再对每个区域的数据点进行位计算,最终判断出是否是 Skyline 点。Chen 等^[15]提出了 MR-Angle 算法。与 MR-BNL 算法不同的是,MR-Angle 算法将数据空间按角度划分,这样保证了点的剪枝能力。丁琳



琳等^[16]通过改进 Hadoop 框架,引入全局过滤的方法,提出了延迟 Skyline 查询算法、贪婪 Skyline 查询算法和混合 Skyline 查询算法,有效地提高了海量数据 Skyline 查询效率。雷婷等^[17]采用空间分区树的启发式方法处理合并局部 Skyline 集合,提出了 HSPD-Skyline 算法。

3 相关概念

首先,本文给出一些定义如下。

定义 1 (静态支配^[11])给定 d 维静态属性集 D 中两个对象 $p_1, p_2, p_1.A_i$ 为 p_1 在第 i 个静态维的坐标,不失一般性,假设对象的属性值越小越好,若在所有静态维度 i 上有 $p_1.A_i \leq p_2.A_i (1 \leq i \leq d)$ 且对于至少一个静态维 j 上有 $p_1.A_i < p_2.A_i (1 \leq j \leq d)$, 则称 p_1 静态支配 p_2 , 记作 $p_1 <_s p_2$ 。

定义 2 (支配^[18])设 d 维静态和一维距离属性集 D 中两个对象 $p_1, p_2, p_1.A_i$ 为 p_1 在第 i 个静态维上的坐标,不失一般性,假设对象的属性值越小越好,若在所有静态维度 i 上有 $p_1.A_i \leq p_2.A_i (1 \leq i \leq d)$ 、在距离属性上有 $dist(q, p_1, t) \leq dist(q, p_2, t)$ 且对于至少一个静态维 j 上有 $p_1.A_i < p_2.A_i (1 \leq j \leq d)$ 或者 $dist(q, p_1, t) < dist(q, p_2, t)$, 则称 p_1 支配 p_2 , 记作 $p_1 < p_2$ 。

定义 3 (局部静态 Skyline 集^[11])部分数据集 $D_i \in D$ 中静态属性上所有不被任何其他点静态支配的点的集合称为 D_i 的局部静态 Skyline 集,记为 SSK_{local} 。其中每个点称为局部静态 Skyline 点。

定义 4 (局部 Skyline 集) t 时刻部分数据集 $D_i \in D$ 中静态属性和动态属性上所有不被任何其他点支配的点的集合称为 t 时刻的局部 Skyline 集合,记为 $SKY_{local}(t)$ 。

定义 5 (全局 Skyline 集^[18]) t 时刻数据集 D 中所有不被任何其他点支配的点的集合称为全局动态 Skyline 集,简称全局 Skyline 集,记为 $SKY_{global}(t)$ 。

本文使用的符号及定义见表 1。

4 基于 MapReduce 的连续 Skyline 查询

MapReduce 框架下移动对象的 Skyline 并行计算总的框架如下:如图 2(a)所示, t_1 时刻, map 对数据集进行分区,计算出查询点到数据点的距离, reduce 对区域数据集(根据 key 值分组)根据预处理的部分结果(存储在 HDFS 中)更新计算出局部 Skyline 集(分别为 $\{p_1, p_{15}, p_{13}, p_{16}\}$ 和 $\{p_2, p_5, p_{14}\}$),最后再使用一个 MapReduce 合并各区域的局部 Skyline 计算出全局 Skyline(即 $\{p_5, p_{13}, p_{16}\}$);如图 2(b)

表 1 符号及定义

符号	定义
D_i	第 i 个区域的数据集
D	全部数据集
$u.A_i$	u 在第 i 维的属性
$u < v$	u 支配 v
$u <_s v$	u 静态支配 v
$dist(q, p, t)$	t 时刻查询点 q 到 p 的距离
SSK_{local_i}	第 i 个区域的局部静态 Skyline
$NSSKY_{local_i}$	第 i 个区域的局部非静态 Skyline 集
$SKY_{local_i}(t)$	t 时刻第 i 个区域的局部 Skyline 集
$SKY_{global}(t)$	t 时刻的全局 Skyline 集
t_{in}	非 Skyline 点成为 Skyline 点的时刻
t_{out}	Skyline 点成为非 Skyline 点的时刻

所示, t_2 时刻按照前面的步骤计算, 此时的局部 Skyline 和全局 Skyline 发生了变化。对于每一时刻循环进行类似计算。

4.1 数据区域划分

对于分布式并行计算, 首先需要考虑负载均衡问题。这里采用了参考文献[11]和参考文献[15]提出的基于角度的分割方法。其基本过程是:首先将数据集中每个点的笛卡尔坐标转化为球坐标,根据球坐标将数据集均匀划分为若干区域,如图 3 所示,再计算每个区域数据集的局部 Skyline 集,最后合并计算全局 Skyline 集。

球坐标计算式如式(1)所示,其中角坐标维度 $\varphi_k (1 \leq k \leq n-1)$ 的取值范围为 $[0, \pi/2)$, A_i 为数据点在第 i 维的属性:

$$\begin{aligned}
 r &= \sqrt{A_n^2 + A_{n-1}^2 + \cdots + A_1^2} \\
 \phi_1 &= \arctan\left(\frac{\sqrt{A_n^2 + A_{n-1}^2 + \cdots + A_2^2}}{A_1}\right) \\
 &\vdots \\
 \phi_{n-2} &= \arctan\left(\frac{\sqrt{A_n^2 + A_{n-1}^2}}{A_{n-2}}\right) \\
 \phi_{n-1} &= \arctan\left(\frac{\sqrt{A_n^2}}{A_{n-1}}\right)
 \end{aligned} \tag{1}$$

4.2 局部区域过滤点策略

正如前文所述,主节点只负责对从节点的任务执行情况等进行监控,不参与具体计算,因此传统的分布式 Skyline 计算中全局过滤点策略已不再适用。根据前面提到的算法,在算法最后都需要使用一个 reduce 来计算全局 Skyline,如果 Skyline 点过多,则其负载量是相当大的,很有可能成为算法的性能瓶颈。已有的基于 MapReduce 的

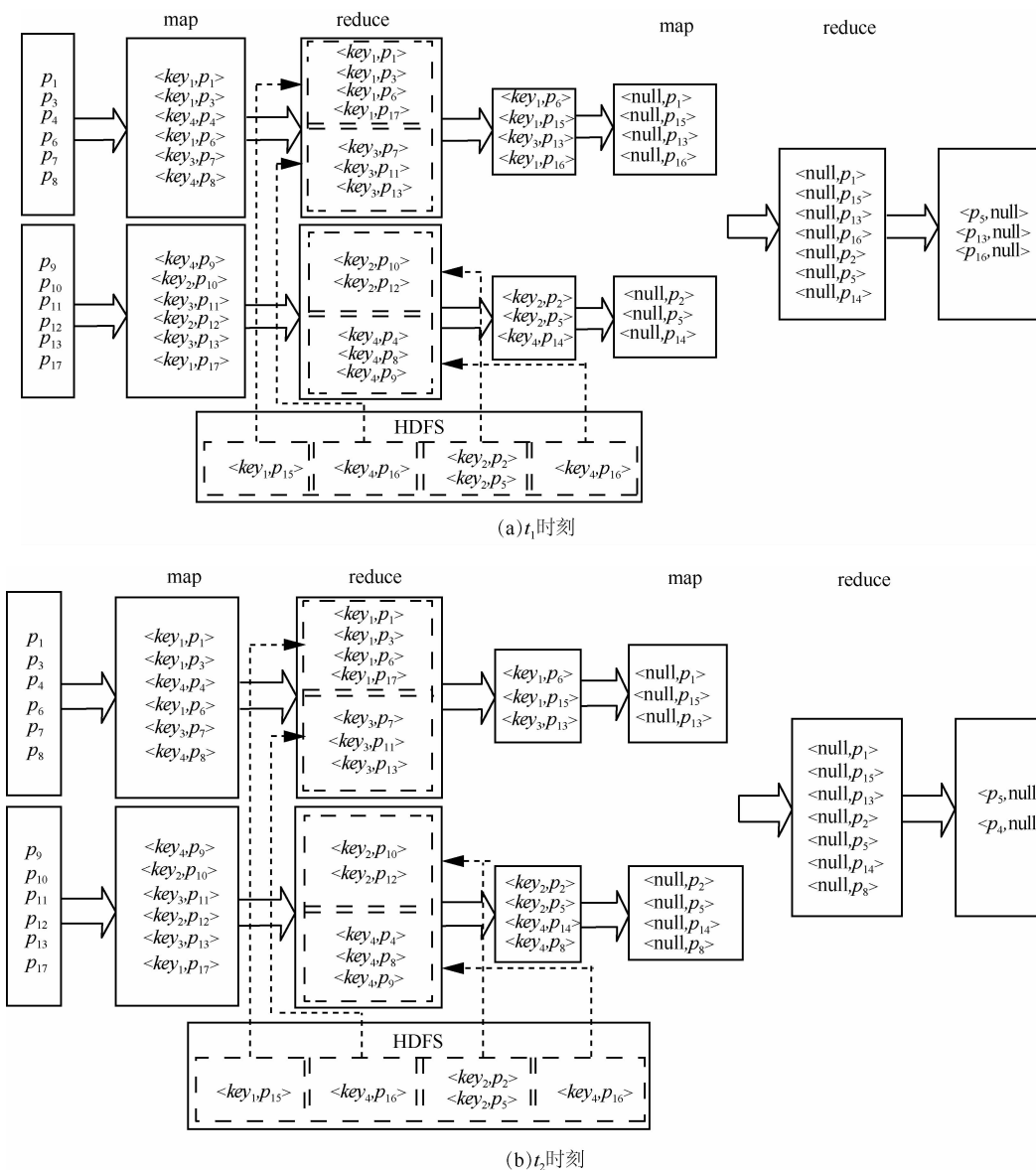


图 2 算法流程举例

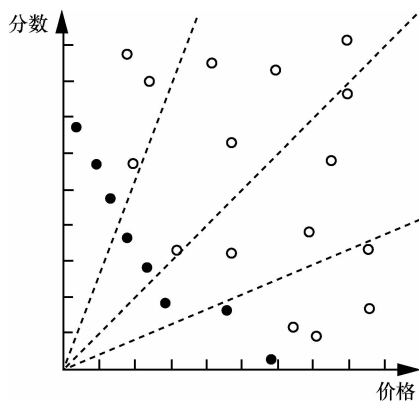


图 3 基于角度的划分

Skyline 计算算法需要将数据进行分块,再将中间键值对按 key 值散列到不同的 reduce 节点上,因此一个 reduce 可能需要处理若干个区域的数据块。因此本文提出了局部过滤点传递剪枝的策略,以剪枝掉一些不可能成为全局 Skyline 的局部 Skyline 点。

局部过滤点剪枝的主要思想是:首先 reduce 计算出某个区域的 Skyline 集合,选出 f 值最小的 Skyline 点作为下一块区域的过滤点, f 值的计算式如下:

$$f(p) = \sum_{i=1}^d p \cdot A_i + \text{dist}(q, p, t) \quad (2)$$

然后在计算出下一块区域的局部动态 Skyline 时使用



过滤点剪枝掉被支配的数据点,最后计算出新过滤点。如图4所示,假设 reduce 阶段的某个 reducer_i 需要计算 3 个区域数据集 (区域号分别为 key_1 、 key_2 和 key_3) 的局部 Skyline 集,则 reducer_i 首先计算出 key_1 区域的局部 Skyline 集合,选出 f 值最小的点作为过滤点 ($filterpoint_1$),用于计算 key_2 区域时的剪枝,同样计算完 key_2 区域后再选出 f 值最小的点作为 key_3 区域的过滤点 ($filterpoint_2$),以此类推。

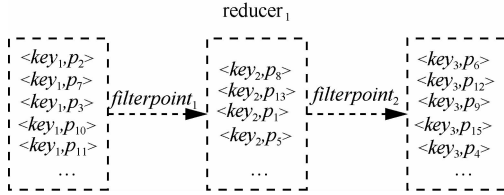


图4 过滤点示意

下面将结合以上区域划分和过滤策略实现 MapReduce 框架下的两种 Skyline 查询算法。

4.3 基于 MapReduce 的静态算法

4.3.1 算法描述

基本的算法主要是不断地循环重复计算局部 Skyline 集和合并局部 Skyline 集计算全局 Skyline 集,这导致了一些不必要的开销。由参考文献[18]可知,静态 Skyline 点必为 Skyline 点。如果考虑首先进行预计算局部静态 Skyline 点并保留这些点,这样在后续查询计算时无需再计算这些静态 Skyline 点,这能减少后续的重复计算。因此,本文提出了基于 MapReduce 的静态算法 (static algorithm based on MapReduce, MR-Static)。

算法 1 首先计算出每个点的球坐标,输出的键值对形式为<区域号,数据点>。在 reduce 阶段使用 BNL 算法计算出静态 Skyline,将数据集分成静态 Skyline ($NSSKY_{local_i}$) 和非静态 Skyline ($NSSKY_{local_i}$) 两个集合。

算法 1 StaticPreprocess

输入:原始数据集 D

输出:局部非静态 Skyline 集合 $NSSKY_{local_i}$

Preprocess Job

Map Task

for each point P_i in dataset D do

 compute the coordinates of P_i using Eq. (1)

 compute the partition R_i that P_i belongs to

 output (R_i, P_i)

end for

Reduce Task

for each partitioned sectors R_i do

 compute $SSKY_{local_i}$ and $NSSKY_{local_i}$ using BNL

 save $SSKY_{local_i}$ in file sst_i

 output ($R_i, NSSKY_{local_i}$) in file st

end for

算法 2 的 Division Job (划分作业) 中, map 的输入不再是全部数据点,只需将非静态 Skyline 作为输入数据集, map 阶段计算非静态 Skyline 数据点到查询点之间的距离。在 reduce 阶段,先将静态 Skyline 集放入局部 Skyline 队列中,再使用 BNL 算法计算出局部 Skyline 集,接着根据局部过滤点 $filterpoint$ 对局部 Skyline 点 SP_i 的支配关系对 Skyline 点进行标记,如果 $filterpoint < SP_i$,则将 $SP_i.gskyflag$ 标记为“0”,即 SP_i 不是全局 Skyline 点,然后按前文提到的方法更新局部过滤点。最后使用一个 MapReduce 过程 (Merging Job (合并作业)) 合并局部 Skyline, map 根据 $gskyflag$ 标记可以将不可能是全局 Skyline 的数据点丢弃,这样可以减少 reduce 阶段的输入量,从而提高计算效率, reduce 合并所有局部 Skyline 集并计算出全局 Skyline。

算法 2 StaticProcess 算法

输入: D 的非静态 Skyline 集合 $NSSKY$

输出: t 时刻 D 的 Skyline 集合 $SKY_{global}(t)$

Division Job

Map Task

for each point P_i in dataset $NSSKY$ do

 compute the distance $dist(q, P_i, t)$ between P_i and

query point q

 output (R_i, P_i)

end for

Reduce Task

for each partitioned sectors R_i do

 add $SSKY_{local_i}$ into $SSKY_{local_i}(t)$

 compute Skyline candidate $SKY_{local_i}(t)$ using BNL

 if ($filterpoint \neq null$)

 for each point P_i in SKY_{local_i} do

 if ($filterpoint$ dominates P_i)

$P_i.gskyflag=0$;

 end for

 for each point SP_i in $SKY_{local_i}(t)$ do

```

    if(f(SPi)<f(filterpoint))
        filterpoint=SPi;
    output (Ri, SKYlocal(t)) in file st
end for
Merging Job
Map Task
for each point Pi in file st do
    if(Pi.gskyflag! =0)
        output (null, Pi)
    end for
Reduce Task
compute global Skyline SKYglobal(t) using BNL
output (SKYglobal(t), null)

```

算法 3 是完整的 MR-Static 算法, 先使用算法 1 预计算出局部静态 Skyline 集和局部非静态 Skyline 集, 再通过算法 2 对每个时刻的查询进行计算。

算法 3 MR-Static 算法

输入: 数据集 D

输出: t 时刻 D 的 Skyline 集合 SKY_{global}(t)

preprocess the data set using algorithm 1

for t=0 to t_{max} do

 compute Skyline at time t using algorithm 2

end for

4.3.2 时间复杂度

若只考虑 map 和 reduce 的影响, 算法的性能分析如下: Division Job 中, 本文假设 N 为全部数据集数量, 第 i 个 map 分割到的数据点数量为 m_i (假设共有 c 个 map, $\sum_{i=1}^c m_i = N$), 对于每个点而言, 计算球坐标、所属区域号和输出所需时间 t_a , 因此 map 阶段计算所需最长时间为 $\max(m_i \cdot t_a)$ 。本文假设 shuffle 将数据集分为 u 个区域, 第 j 个区域的数据点数量为 n_j , 对于 BNL 算法来说, 其最好情况下 (即第一个数据点支配后续所有点) 的所需比较次数为 n_j (即 Skyline 集合大小为 1), 此时输出非静态 Skyline 集所需次数为 1; 而最坏情况下 (数据点两两互相比) 的比较次数是 $\frac{n_j(n_j-1)}{2}$ (即 Skyline 集合大小为 n_j), 此时输出非静态 Skyline 集次数为 n_j 。因此可以计算得到最好情况下其时间复杂度为 $O(\max(m_i \cdot t_a) + \max(n_j + n_j))$, 近似为 $O(N)$; 最坏情况下其时间复杂度为 $O(\max(m_i \cdot t_a) + \max(\frac{n_j(n_j-1)}{2} + n_j))$, 近似 $O(N^2)$ 。

在查询处理的 Division Job 中, 假设第 i 个 map 分割到的非静态数据点数量为 r_i (假设共有 d 个 map), 计算距离和输出所需时间为 t_b , map 阶段计算所需最大时间为 $\max(r_i \cdot t_b)$ 。本文假设 shuffle 将数据集分为 u 个区域, 因为按静态属性划分区域, 所以第 j 个区域的局部数据点全集数量始终不变为 n_j , 本文假设局部非静态 Skyline 集合数量为 w_j ($0 \leq w_j \leq n_j \leq 1$), BNL 算法最好情况下 (即静态数据点支配后续所有非静态点) 的所需比较次数为 w_j (即所有局部非静态 Skyline 被第一个静态 Skyline 支配), 使用过滤点剪枝和选出过滤点都为 $O(n_j - w_j)$, 此时输出所需次数为 $n_j - w_j$, 因此查询处理阶段最好情况下的时间复杂度为 $O(\max(r_i \cdot t_b) + \max(n_j + n_j - w_j + n_j - w_j + n_j - w_j))$, 近似为 $O(N)$ 。而最坏情况下的比较次数是 $\frac{w_j(2n_j - w_j - 1)}{2} + n_j$ (即 Skyline 集合大小为 n_j), 使用过滤点剪枝最多需要 n_j , 选出过滤点最多需要 n_j (假设有 s_j 个被过滤点标记) 输出最多需要 n_j , 因此查询处理阶段最差情况下的时间复杂度为 $O(\max(r_i \cdot t_b) + \max(n_j + \frac{w_j(2n_j - w_j - 1)}{2} + n_j + n_j))$, 近似为 $O(N^2)$ 。

在 Merging Job 中, 假设第 k 个 map 分割到的数据点数量为 l_k (假设共有 e 个 map), 则 map 阶段所需最长时间为 $\max(l_k)$ 。而对于最终的 reduce 而言其最好的时间为

$\sum_{j=1}^u (n_j - w_j - s_j)$, 此时输出为 1; 最差时间为 $\frac{N(N-1)}{2}$, 输出最差为 $\frac{N(N-1)}{2}$ 。因此这个阶段的最好时间复杂度为 $O(\max(l_k) + \sum_{j=1}^u (n_j - w_j - s_j) + 1)$, 近似为 $O(N)$; 最差时间复杂度为 $O(\max(l_k) + \frac{N(N-1)}{2} + \frac{N(N-1)}{2})$, 近似为 $O(N^2)$ 。

4.4 基于 MapReduce 的跟踪算法

4.4.1 算法描述

对于局部 Skyline 点而言, 不是所有点同时进入或移除局部 Skyline 集合, MR-Static 算法的缺点在于重复计算了部分没有发生变化的 Skyline 点, 如果能够知道某个点影响 Skyline 集合的时间就能很大程度上减少计算量。本文将非 Skyline 数据点进入 Skyline 集合的事件 (INSERT) 发生的时刻为 t_{in} , 同样定义 Skyline 数据点离开 Skyline 集合的事件 (DELETE) 发生时刻为 t_{out} 。为了计算出某个点的 t_{in} 和 t_{out} , 需要将该点与数据集中其他点进行比较。对于两个数据点 p 和 u 而言, 假设两者到 q 距离相等的时刻为 t_{equal} , 且 $u <_s p$,



u 对 p 支配关系的变化有两种情况: 若 t_{equal} 时刻之前有 $\text{dist}(q, u) < \text{dist}(q, p)$, 此时 $u < p$, 则 t_{equal} 时刻之后, 因为 $\text{dist}(q, u) > \text{dist}(q, p)$, 所以此时 u 无法支配 p ; 若 t_{equal} 时刻之前有 $\text{dist}(q, u) > \text{dist}(q, p)$, 此时 u 不支配 p , 而 t_{equal} 时刻之后, 由于 $\text{dist}(q, u) > \text{dist}(q, p)$, 则 $u < p$, 此时 p 不是 Skyline 点。对于数据点 p 而言, 其与其他数据点产生第一种情况的最晚时间即所求的 t_{in} , 因为在这个时间之前至少存在一个数据点支配 p , 也就是之前 p 不可能是局部 Skyline 点, 而之后又无其他点支配 p 。类似地, p 发生第二种情况的最早时间即 t_{out} 。

根据参考文献[9], 对于二维空间内的匀速直线运动的查询点 q 而言, 假设其初始时刻的位置坐标为 (x_0, y_0) , 速度为 (v_x, v_y) , 数据点 p 的坐标为 (x_p, y_p) , 那么数据点 p 到查询点 q 的欧式距离函数可以表示为:

$$\text{dis}(q, p, t) = \sqrt{at^2 + bt + c} \quad (3)$$

其中, 参数 a, b, c 是由 q 的起始位置、移动速度和数据点的位置决定的: $a = v_x^2 + v_y^2$, $b = 2[(x_0 - x_p) \cdot v_x + (y_0 - y_p) \cdot v_y]$, $c = (x_0 - x_p)^2 + (y_0 - y_p)^2$ 。

由于速度 (v_x, v_y) 为常量, 因此只需要计算出 b, c 即可, 这样对于两个数据点 p_1 和 p_2 而言, 它们到查询点 q 距离相等的时刻可以使用以下计算式:

$$t_{\text{equal}} = \frac{c_1 - c_2}{b_2 - b_1} \quad (4)$$

与 MR-Static 算法一样, 基于 MapReduce 的跟踪算法 (MR-Track 算法) 主要分为两个步骤: 预处理和事件处理。

如算法 4 所示, 预处理的基本步骤如下: 在 map 阶段将数据集按角度划分并计算出查询点 q 到各数据点的初始距离。在 reduce 阶段, 首先计算出两个数据点之间的静态支配关系和支配关系, 然后根据式 (3) 和式 (4) 计算出相应的参数 b, c 和相交时间 t_x , 根据 t_x 和 t_x 时刻的事件类型更新最晚 INSERT 的时刻 t_{last} 和最早 DELETE 的时刻 t_{first} 。在与其他所有点比较完之后, 如果 P_j 不被其他点支配, 则将其加入初始的局部 Skyline 集合中, 然后将 t_{last} 和 t_{first} 的事件加入局部事件队列 EQ_i 中, 最后需要对 EQ_i 进行升序排序。这样循环对每个数据点进行处理。

算法 4 TrackPreprocess

输入: 原始数据集 D

输出: 初始局部 Skyline 集合

Preprocess Job

Map Task

```

for each point  $P_i$  in dataset  $D$  do
  compute the coordinates of  $P_i$  using Eq. (1)
  compute the partition  $R_i$  that  $P_i$  belongs to
  compute the initial distance between  $P_i$  and query
  point  $Q$ 
  output ( $R_i, P_i$ )
end for
Reduce Task
for each partitioned sectors  $R_i$  do
  for each point  $P_j$  in  $R_i$  do
    for each point  $S_k$  in  $R_i$  do
      if ( $S_k$  statically dominate  $P_j$ )
        compute  $b, c$  and the intersection time  $t_x$ 
        using using Eq.(3) and Eq.(4) separately;
        Update  $t_{\text{in}}$  and  $t_{\text{out}}$ 
      end for
      add <INSERT,  $t_{\text{in}}$ > and <DELETE,  $t_{\text{out}}$ > of  $P_j$ 
      into  $EQ_i$ 
      if ( $P_j$  is not dominated by all other points in  $R_i$ )
        insert  $P_j$  into initial skyline  $SKY_{\text{local}}(t)$ ;
      end for
    quicksort( $EQ_i$ );
    output ( $R_i, SKY_{\text{local}}$ ) in file  $st$ 
  end for

```

跟踪处理(算法 5)主要针对时刻 t 的事件进行处理更新局部 Skyline 并计算出全局 Skyline。其基本过程如下: 第一个 MapReduce 阶段的 map 计算出时刻 t 查询点和数据点 P_i 的距离, reduce 阶段先从局部事件队列中取出 t 时刻之前的事件存放到 HQ_i 队列中, 再根据 HQ_i 中的事件更新局部 Skyline 集合, 如果事件类型为 INSERT, 则将相应的数据点插入局部 Skyline 集合中, 反之需要将对应的数据点从局部 Skyline 集合中移除, 最后采用过滤策略剪枝并计算出全局 Skyline。

算法 5 TrackProcess

输入: 上一时刻的局部 Skyline 集合 $SKY_{\text{local}}(t-1)$

输出: t 时刻 D 的 Skyline 集合 $SKY_{\text{global}}(t)$

Event Job

Map Task

for each point P_i in dataset $SKY_{\text{local}}(t-1)$ do

```

    compute the distance between  $P_i$  and query point  $q$ 
at time  $t$ 
    output ( $R_i, P_i$ )
end for
Reduce Task
for each partitioned sectors  $R_i$  do
    for each event  $E$  in  $Eq_i$  do
        if ( $E.t_x \leq t_{current}$ )
            update( $SKY_{local_i}(t), E$ );
        else
            break;
        end for
    if (filterpoint! = null)
        for each point  $P_i$  in  $SKY_{local_i}(t)$  do
            if (filterpoint dominates  $P_i$ )
                 $P_i.gskyflag = 0$ ;
            end for
        for each point  $SP_i$  in  $SKY_{local_i}(t)$  do
            if ( $f(SP_i) < f(filterpoint)$ )
                filterpoint =  $SP_i$ ;
            end for
        output ( $R_i, SKY_{local_i}(t)$ ) in file st
    end for
Merging Job
Map Task
for each point  $P_i$  in file st do
    if ( $P_i.gskyflag! = 0$ )
        output (null,  $P_i$ )
    end for
Reduce Task
compute global Skyline  $SKY_{global}(t)$  using BNL
output ( $SKY_{global}(t)$ , null)

```

完整的 MR-Track 算法(算法 6)先使用算法 4 预计算出数据点两两之间的支配关系发生变化的事件,再通过算法 5 更新每个时刻的局部 Skyline 集合计算出全局 Skyline。

算法 6 MR-Track 算法

输入:数据集 D

输出: t 时刻 D 的 Skyline 集合 $SKY_{global}(t)$

preprocess the data set using algorithm 4

```

for  $t=0$  to  $t_{max}$  do
    compute Skyline at time  $t$  using algorithm 5
end for
4.4.2 时间复杂度

```

本节将从预处理和跟踪处理两个阶段分析算法的时间复杂度。

预处理中,假设第 i 个 map 分割到的非静态数据点数量为 r_i (假设共有 d 个 map),计算球坐标、计算所属区域号、计算初始距离和输出所需时间为 t_a ,map 阶段计算所需时间为 $\max(r_i \cdot t_a)$ 。本文假设 shuffle 将数据集分为 u 个区域,第 j 个区域的数据点数量为 n_j ,判断支配关系、计算相交时刻、更新 t_{in} 和 t_{out} 可以在 t_b 时间内完成,输出的局部 Skyline 最多为 n_j ,而插入事件和 Skyline 可在 t_d 时间内完成。快速排序最多需要 $(2n_j)^2$ 。因此预处理阶段的时间复杂度为 $O(\max(r_i \cdot t_a) + \max(n_j \cdot (n_j \cdot t_c + t_d) + 4n_j^2 + n_j))$,近似为 $O(N^2)$ 。

跟踪处理的 Event Job(事件作业)中,假设第 k 个 map 分割到的数据点数量为 l_k (假设共有 e 个 map),计算 t 时刻的距离和输出可在 t_c 内完成,则 map 阶段所需时间为 $\max(l_k \cdot t_c)$ 。因为按静态属性划分区域,所以第 j 个区域的局部数据点全集数量始终不变为 n_j 。而事件队列中的事件最多为 $2n_j$,更新局部 Skyline 最多需要 $2n_j \cdot t_{update}$ (每次更新可在 t_{update} 内完成),使用过滤点剪枝最多需要 n_j ,选出过滤点最多需要 n_j (假设有 s_i 个被过滤点标记),输出最多需要 n_j 。因此最差情况下查询处理的时间复杂度为 $O(\max(m_i \cdot t_c) + \max(2n_j + 2n_j \cdot t_{update} + n_j + n_j + n_j))$,近似为 $O(N)$,显然要优于 MR-Static 算法。

Merging Job 中,假设第 k 个 map 分割到的数据点数量为 l_k (假设共有 e 个 map),则 map 阶段所需时间为 $\max(l_k)$ 。与 MR-Static 算法相同,这个阶段的最好时间复杂度为

$O(\max(l_k) + \sum_{j=1}^u (n_j - w_j - s_j) + 1)$,近似为 $O(N)$;最差时间复杂度为 $O(\max(l_k) + \frac{N(N+1)}{2} + \frac{N(N+1)}{2})$,近似为 $O(N^2)$ 。

5 实验评估

5.1 实验环境

实验在 4 台 PC 机组成的集群上进行,其中 1 台作为主节点,其余 3 台作为从节点。每台 PC 配置相同:CPU 为 Intel Core i3 3.4 GHz,4 GB 内存,操作系统为 CentOS 6.3, Hadoop 版本为 1.0.3, JDK 版本为 1.6。



实验采用了人工合成数据集和真实数据集。利用参考文献[1]的生成工具生成3种人工数据集,每种数据集默认包含 2×10^6 个对象,对象的位置属性随机生成,静态维度为3维,每一维属性的取值范围为 $(0, 1\ 000]$ 的双精度浮点数。这些数据集分别服从以下3种分布。

- 独立(independent):对象各维属性值服从均匀分布且独立同分布。
- 正相关(correlated):如果对象的某一维属性值大(小),则其他属性值也很大(小)。
- 反相关(anti-related):如果对象的某一维属性值大(小),则其他属性值很小(大)。

真实数据集的地理位置属性采用北美地区33 696个人口稠密地点和文化地标^[9],静态属性为6维独立分布数据。

为了直观比较,本文采用参考文献[15]中的MR-Angle算法作为基本算法。实验通过调整数据规模 N 、静态维度 d 、从节点数 n 等参数设置来考察MR-Angle算法和本文的两种算法在这3种数据集上的运行时间和map和reduce过程中输入输出的元组数,从而验证算法的有效性,其中运行时间为处理10次查询的平均时间。

5.2 实验结果和分析

5.2.1 人工数据集

5.2.1.1 数据规模对算法的影响

本次实验主要考察当数据点个数依次设为 2×10^6 、 4×10^6 、 6×10^6 和 8×10^6 时对算法的影响,其他参数设置如下:节点数 $n=4$,静态维度 $d=3$,移动对象速度 $= (6, 10)$,球坐标的每一维划分4部分(d 维数据空间可分为 4^d 部分)。从图5上可见在数据规模比较小时,MapReduce的性能对于算法的影响较大,随着数据规模的增大,MR-Angle算法和MR-Static算法的响应时间都出现了增长,但MR-Static算法好于MR-Angle算法,而MR-Track算法的响应时间基本稳定。在

反相关数据集上,3个算法相比独立数据集时出现了增长,这是因为反相关数据的数据之间不支配的可能性增大,导致在计算全局Skyline时的输入也出现相应增长,从而导致计算开销增加。同样对于正相关数据集响应时间较小是因为Skyline点明显少于其他两种数据集。又可从图6看出MR-Track算法的平均元组数明显少于其他两种算法,这是因为MR-Track算法无须再重复计算Skyline,只需更新部分Skyline。

5.2.1.2 静态维度对算法的影响

在这个实验中,静态数据维度数 d 依次调整为1、2、3和4,其他参数如下:节点数 $n=4$,数据规模 $N=2 \times 10^6$,移动对象速度 $= (6, 10)$,球坐标的每一维划分4部分。图7显示了随着维度的增长,算法的响应时间的变化。尤其对于独立数据集和反相关数据集来说,静态维度 $d=4$ 和5时最后合并的Skyline数量超过了分配内存大小,此时所需响应时间明显增长;而对于正相关数据集而言,响应时间变化不大,这是因为这种数据集中的Skyline点数量相对增加较少,MapReduce过程所需时间影响较大。同样在I/O开销上,3种算法都出现了一定增长(如图8所示),这是因为随着维度的上升,数据点的不被其他点支配的可能性越来越大,导致局部Skyline增多。从图7和图8中看出MR-Static算法和MR-Track算法仍比MR-Angle算法好。

5.2.1.3 Slave节点数对算法的影响

本次实验考察节点数对算法的影响。实验依次将slave节点数量设为2、3、4和5,并且设置相应的reduce数量,其他参数不变。从图9中可见MR-Track算法优于前两种算法。可以发现从节点数量对正相关数据的影响较小,这是因为正相关数据中包含的Skyline较少,导致对算法性能的影响也较小。而在独立数据集和反相关数据集上,算法的执行时间随着节点数的增加而有所降低,因此增加节点数可以在一定程度上提高数据处理的能力。

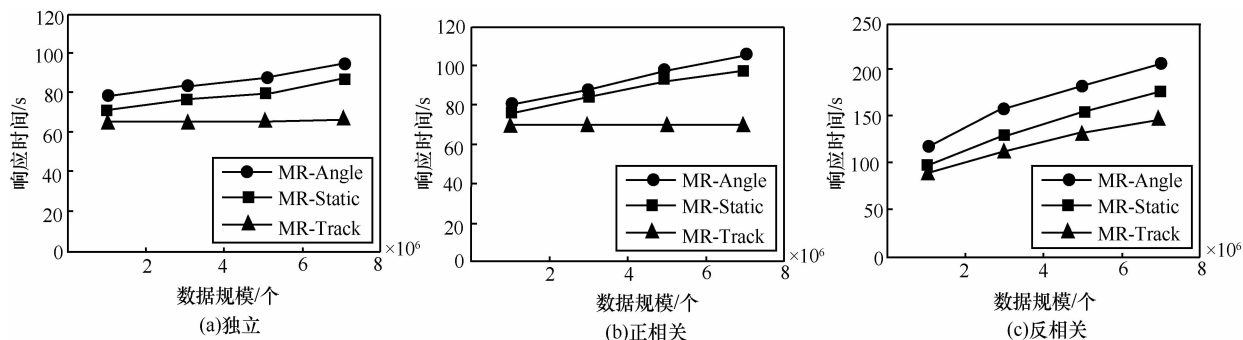


图5 数据规模对算法运行时间的影响

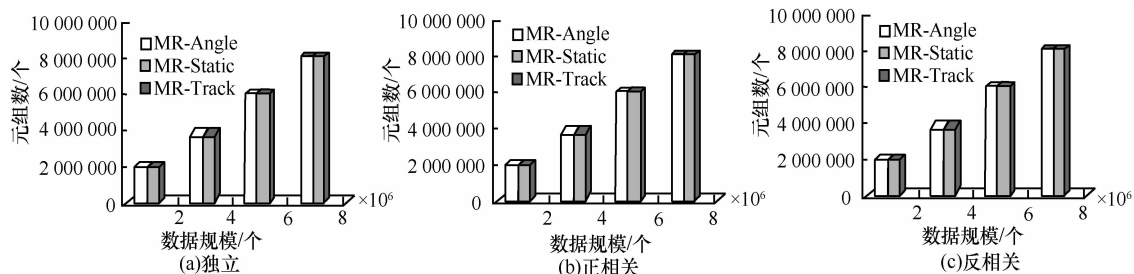


图 6 数据规模对元组数的影响

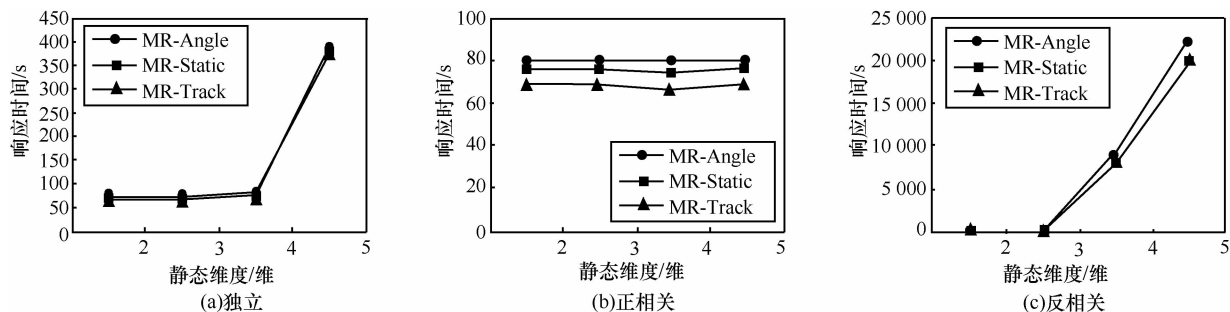


图 7 静态维度对算法运行时间的影响

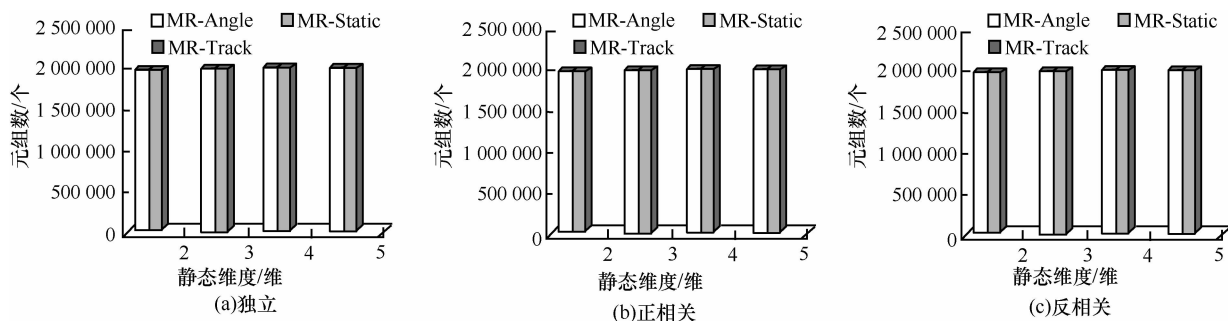


图 8 静态维度对元组数的影响

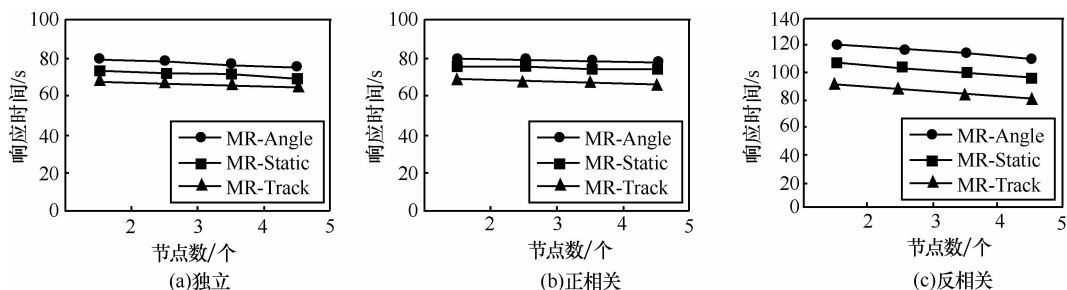


图 9 从节点数对算法运行时间的影响

5.2.2 真实数据集

本实验主要考察了在真实数据集上从节点数(依次设为 2、3、4、5)对算法的影响。从图 10 中可以看出由于数据集较小的因素,3 种算法在响应时间上很接近,而且有下降的趋势,这是因为真实数据集的规模不够大导致响应时间主要受 MapReduce 框架的限制。

6 结束语

本文研究了 MapReduce 框架下查询点移动、其他点固定的 Skyline 查询,通过预处理提前计算出局部静态 Skyline 来减少后续的计算量,提高计算效率,并通过使用局部过滤部分区域数据的方法进行剪枝:通过预处理提前计算保留

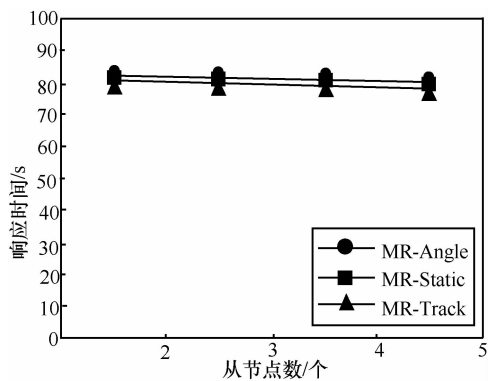


图 10 从节点数对算法运行时间的影响

局部静态 Skyline 集,实现了 MR-Static 算法;通过预计算事先分别计算出数据点由非 Skyline 点成为 Skyline 点的时刻和 Skyline 点成为非 Skyline 点的时刻,后续更新局部 Skyline 集,实现了基于事件跟踪的 MR-Track 算法。实验结果证明了 MR-Track 算法的有效性和高效性。进一步的研究将着重于查询点不确定情况下的 Skyline 查询和 MapReduce 框架的改进。

参考文献

- 1 Borzsonyi S, Kossmann D, Stocker K. The Skyline operator. Proceedings of the International Conference on Data Engineering (ICDE), Heidelberg, Germany, 2001: 421~430
- 2 Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters. Communications of the ACM, 2005, 51(1):107~113
- 3 Tan K L, Eng P K, Ooi B C. Efficient progressive skyline computation. Proceedings of the VLDB, Roma, Italy, 2001: 301~310
- 4 Kossmann D, Ramsak F, Rost S. Shooting stars in the sky: an online algorithm for skyline queries. Proceedings of the Very Large Data Bases (VLDB), Hong Kong, China, 2002:275~286
- 5 Papadias D, Tao Y F, Fu G, *et al.* An optimal and progressive algorithm for skyline queries. Proceedings of ACM Management of Data (SIGMOD), San Diego, California, USA, 2003:467~478
- 6 Balke W T, Guntzer U, Zheng J X. Efficient distributed Skylining for web information systems. Proceedings of International Conference on Extending Database Technology (EDBT), Heraklion, Crete, Greece, 2004: 256~273
- 7 Cui B, Lu H, Xu Q Q, *et al.* Parallel distributed processing of constrained Skyline queries by ltering. Proceedings of International Conference on Data Engineering (ICDE), Cancun, Mexico, 2008:546~555
- 8 Huang Z Y, Lu H, Ooi B C, *et al.* Continuous skyline queries for moving objects. IEEE Transactions on Knowledge and Data Engineering, 2006, 18 (12): 1645~1658
- 9 Tian L, Wang L, Zou P, *et al.* Continuous monitoring of skyline

query over highly dynamic moving objects. Proceedings of the ACM International Workshop on Data Engineering for Wireless and Mobile Access, Beijing, China, 2007:59~66

- 10 付世昌,董一鸿,唐燕琳等. 基于事件的位置不确定移动对象连续概率 Skyline 查询. 自动化学报, 2011, 37(7): 836~848
- 11 Vlachou A, Doukeridis C, Kotidis Y. Angle-based space partitioning for efficient parallel skyline computation. Proceedings of ACM Management of Data (SIGMOD), Vancouver, BC, Canada, 2008:227~238
- 12 Park S, Kim T, Park J, *et al.* Parallel skyline computation on multicore architectures. Proceedings of IEEE 25th International Conference on Data Engineering (ICDE), Shanghai, China, 2009:760~771
- 13 Kohler H, Yang J, Zhou X F. Efficient parallel skyline processing using hyperplane projections. Proceedings of the ACM SIGMOD International Conference on Management of Data, Athens, Greece, 2011:85~96
- 14 张波良,周水庚,关佑红. MapReduce 框架下的 Skyline 计算. 计算机科学与探索, 2011, 5(5):385~397
- 15 Chen L, Hwang K, Wu J. MapReduce Skyline query processing with a new angular partitioning approach. Proceedings of Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), Shanghai, China, 2012:2262~2270
- 16 丁琳琳,信俊昌,王国仁. 基于 MapReduce 的海量数据高效 Skyline 查询处理. 计算机学报, 2011, 34(10):1785~1796
- 17 雷婷,王涛,曲武等. 云环境下基于超球面投影分区的 Skyline 计算. 计算机科学, 2013, 40(6):164~171
- 18 Xiao Y Y, Lu K, Deng H F. Location-dependent skyline query processing in mobile databases. Proceedings of Web Information Systems and Applications Conference(WISA), Hohhot, China, 2010
- 19 ChoroChronos. <http://www.chorochronos.org/>

[作者简介]



单观敏,男,宁波大学信息科学与工程学院硕士生,主要研究方向为数据挖掘。

董一鸿,男,博士,宁波大学信息科学与工程学院教授,主要研究方向为移动数据库、数据挖掘和人工智能。

何贤芒,男,博士,宁波大学信息科学与工程学院讲师,主要研究方向为数据挖掘和隐私保护。

(收稿日期:2013-10-13)