一个面向需求扩展的不确定数据 Top-k 查询改进算法

俞闽敏 陈宁江
(广西大学计算机与电子信息学院 南宁 530004)

摘 要 已有的不确定数据 top-k 查询语义只返回在可能世界中聚集概率最大的一个应答,并不能很好地满足用户差异化的查询需求。针对这个问题,通过引入反映查询需求的指标"需求扩展度",定义了基于需求扩展的不确定数据查询语义 RU-Top$k$,并且提出了在新语义下的查询算法。实验表明,RU-Top$k$ 算法具有较小的平均单位查询运行时间,且在满足用户需求的情况下,具备更高的查询效率。

关键词 不确定数据,top-k,需求扩展
中图法分类号 TP311 **文献标识码** A

Algorithm of Improved Top-k Query on Uncertain Data for Requirement Extension

YU Min-min CHEN Ning-jiang
(College of Computer, Electronic, and Information, Guangxi University, Nanning 530004, China)

Abstract The semantic of the existing top-k query on uncertain data only returns one response, which has the largest gathered probability in all possible worlds. As a result, it cannot meet the consideration of users' individuation requirement. To address this issue, the concept of requirement extend degree was introduced and a top-k query semantic on uncertain data for requirement extension was defined. Further, an algorithm named RU-Top$k$ to process the semantic was presented. The Experimental results show the superiority of RU-Top$k$ algorithm in terms of run time for per query answer and query efficiency compared with the existing algorithms under the condition of meeting users' requirements.

Keywords Ucertainty data, Top-k, Requirement extension

1 引言

不确定数据广泛存在于文本分析、信息检索、传感器网络和射频识别等领域。随着数据采集手段的发展,客观世界中普遍存在的数据不确定性逐渐被人们所认识,不确定数据查询处理问题已经引发了学术界和工业界的共同关注与重视,成为新近发展起来的研究热点[1]。top-k 查询在传统的查询处理中得到了广泛研究,其通过用户为数据对象指定的一个打分函数,返回给用户最满意的 k 个数据对象,避免返回大量的查询结果,在大规模数据集中能够有效提高查询效率。在不确定数据集上,top-k 查询结果不仅依赖于数据对象的打分值,还与它自己的置信度以及其它对象的打分值与置信度有关。打分函数与置信度的相互作用决定返回哪些数据对象,考量二者不同的结合方式则产生不同的不确定 top-k 查询语义。

近年不确定数据查询上的 top-k 查询取得了不少成果,根据定义与应用场景的不同,主要有以下几类:

Soliman 等[2]最早研究了不确定数据 top-k 查询处理问题,提出了 U-Top$k$ 查询,其返回在所有可能世界中发生概率最大的元组向量;OptU-Top$k$ 则是利用状态空间来优化 U-Top$k$ 查询的算法,其核心思想是将每一个可能世界实例映射为一个状态,然后持续地生成状态空间,并通过剪枝技术不断压缩空间,直至获得所需的查询结果。Chen 和 Yi 等人[3]考虑了在数据源动态变化情况下的 U-Top$k$ 查询问题,设计了一种数据结构,其能随着数据源变化在较短时间内动态更新查询结果。OptU-kRanks 算法[4]计算每一个新到来的元组在所有可能世界中某个排名上出现的概率之和,并将这个值与已计算过的概率和进行比较,直到根据剪枝策略判断出来的元组都不可能获得更大的概率时,已考虑过的元组中概率和最大者即为该排名所求的查询结果。

Hua 等[5]提出的 PT-k 语义与 Jin 等[6]提出的 Pk-Top$k$ 查询类似,核心思想是计算元组在所有可能世界中成为 top-k 的聚集概率,然后根据用户指定的概率阈值 p 对查询结果进行剪枝,使返回的结果元组超过 p 的概率在整个可能世界空间中排名位于前 k 个。根据 PT-k 查询的特点,若能重复使用每个元组计算得到的概率值,则可以避免将所有的元组纳入计算范围,从而减少计算开销,达到提高查询效率的目的。在他们后续的工作中给出了一种基于泊松近似规则的算法,对元组的聚集概率的计算过程进行剪枝[7]。

Yi 等人[8]在包含元组存在级与属性级的 x-relations 模

到稿日期:2011-07-14 返修日期:2011-10-17 本文受国家科技支撑计划课题(2009BAH53B03),国家自然科学基金项目(61063012),广西高校优秀人才资助计划(桂教人[2011]40号),广西教育厅科研项目(桂教科研[2010]10号)资助。
俞闽敏(1986-),女,硕士生,主要研究方向为网络分布计算、软件工程,E-mail:sunshineyu@126.com;陈宁江(1975-),男,博士,教授,CCF 高级会员,主要研究方向为软件工程、网络分布计算,E-mail:calmriver@263.net(通信作者)。

型上定义了 U-Top$k$ 和 U-kRanks 查询语义，并给出了具有多项式运行时间与空间的查询处理算法。刘德喜等人[9]将 x-tuple 作为一个整体，为 U-kRanks 查询定义了新的语义 U-x-kRanks，返回最可能排在前 k 的 x-tuple 而非元组，并在此语义下提出了一种基于动态规划的查询算法。

王爽等人[10]提出了分布式环境中的不确定数据 top-k 查询算法 UDTop$k$。它定义了一个候选集，仅使用候选集中的数据而不用访问数据集中所有数据，就可以得到正确的 top-k 查询答案，并通过动态维护候选集与仅传输少量数据，达到减少网络中数据传输的目的。

总体而言，迄今为止的各种查询语义和其使用的算法都是基于各自所强调的应用场景而提出的。目前不确定数据 top-k 查询处理领域的研究工作都聚焦于以下两个方面：依据特定的实际应用环境定义不同的不确定数据 top-k 查询语义，以及设计与应用环境相适应的查询算法。在实际应用中，用户常常会对查询结果提出质量要求。比如，由于数据具有不确定性，查询的最优解发生在某个特定概率下，用户很可能并不满足于只获得最优解，而是想了解其它次优解的情况，甚至是次优解与最优解之间的距离。然而，已有的绝大多数研究对用户差异化需求的考虑都有所欠缺。为了解决这个问题，本文探讨将查询质量相关的用户需求约束引入到不确定 top-k 查询语义中，定义了新的语义 RU-Top$k$，当满足某个用户需求阈值设定时，输出 top-k 查询结果。

2 问题分析

首先以一个应用场景为例来说明本文重点研究的问题。在化工、核电等行业中，高温作业容易引发爆炸等事故。在一个工业生产监视系统中，基于安全要求，监测生产车间的温度。温度数据从分布在厂房各处的无线传感器获得。在这个无线传感器网络中，传感器节点由于受到周围环境干扰、机械故障、仪器精度限制等影响，所监测到的数据往往是不精确的。假设各传感器获取温度事件是相互独立的，我们给每个数据附加置信度来描述所获数据准确的概率，并按照温度测量值从高到低排序。表 1 列出了在某一时刻，各个传感器测得的温度数据。

表 1 测量温度数据集

tuple ID (元组标识)	Sensor ID (传感器器 ID)	Score (温度测量值)	Confidence (Pr(t))
t_1	A	100	0.8
t_2	B	96	0.3
t_3	C	92	0.4
t_4	D	76	0.9

对于 top-k 查询，用户首先需要知道的信息是最可能的实际温度最高的数个传感器。由于高温易引发爆炸，即使传感器结点发生高温的可能性较小，但是一旦发生就将造成巨大事故，故不可掉以轻心，因此用户更为关注温度测量值高的传感器结点；温度测量值较低的结点即使是最有可能发生的实际温度最高的结点，也不是用户所关心的，因此用户希望能限定查询结果在其所关心的范围与程度之内，故产生如下需求。

需求①：查询最有可能的实际温度最高的 k 个传感器结点。

需求②：在温度测量值排名前 k 个传感器结点中恰好有 i 个发生的情况下，查询最可能的实际温度最高的 k 个传感器结点（即查询返回的最可能的实际温度最高的 k 个传感器结点中，恰好包含 i 个结点，并且这 i 个结点温度测量值都排名在前 k 个，i 是由用户自定义的整数，且 $i\leqslant k$）。

需求③：在温度测量值排名前 k 个传感器结点中至少有 80% 发生的情况下，查询最可能实际温度最高的 k 个传感器结点（即查询返回的可能的实际温度最高的 k 个传感器结点中，至少包含了温度测量值都排名在前 k 个结点中的 80%）。

需求④：在温度测量值排名前 v 个传感器结点中恰好有 i 个发生的情况下，查询最可能的实际温度最高的 k 个传感器结点（即查询返回最可能的实际温度最高的 k 个传感器结点中，恰好包含 i 个结点，这 i 个结点温度测量值都排名在前 v 个，i 和 v 是由用户自定义的整数，且 $i\leqslant v\leqslant k$）。

需求⑤：通过指定不同的 i 值，在一次查询中一次性返回基于需求②的多个查询应答。

需求⑥：在一次查询中，能比较基于需求⑤返回的多个查询应答的概率大小。

在现有的不确定数据 top-k 查询语义中，只能满足用户需求①，剩下的需求都难以满足。对于需求②－④，已有的查询只是考虑最大的概率，没有相应的指标来反映用户的需求。在已有的 top-k 查询中，需求①的结果在很多情况下，最终查询结果不在 score 函数排名前 k 的元组中，比如假设查询 top-2 返回的结果为 $\{t_5,t_6\}$ 时，最有可能的实际温度最高的传感器都不在元组集合 $\{t_1,t_2\}$ 中，$\{t_1,t_2\}$ 是测量温度最高的 2 个元组。对于需求⑤、需求⑥，现有的查询语义在一次查询中，只能返回概率最大的一个查询应答，并不能得到多个查询应答，也无从比较。

为了满足用户的需求，扩展已有的不确定 top-k 查询语义，引入一个新指标“需求扩展度”，以更精确地反映用户对于不确定 top-k 查询的深层需求。由此提出一种面向用户需求扩展的不确定数据 top-k 查询算法 RU-Top$k$（Requirement Extension Uncertain Top-k）。

3 算法思想及描述

首先列出算法中用到的相关概念。

- D：不确定数据库；
- D_i：不确定数据库 D 中按照打分函数降序排列后排名在前 i 的元组集合；
- $\overline{D_i}$：属于 D 但不属于 D_i 的元组的集合；
- n：OptU-Top$k$[2]算法的扫描深度；
- $Pr(X)$：X 发生的概率，X 是一个可能事件，或一个元组，或一个元组向量，或一个元组集合；
- $\overline{W}$：由 D 生成的所有可能世界组成的集合；
- W：一个可能世界；
- W_i：由 D_i 生成的所有可能世界组成的集合。

为了反映用户的需求，从用户的角度引入了需求扩展度 q 的概念。

定义 1（需求扩展度） 长度为 k 的元组向量/集合 X 的需求扩展度 $q(X)=|U|/k$，$q(X)\in[0,1]$，$U=\{t\mid t\in X\cap D_k\}$，$|U|$ 表示集合 U 中包含的元素个数。

定义 2（面向需求扩展的不确定 top-k 查询，RU-Top$k$） 设 D 是一个具有可能世界空间 $\overline{W}$ 的不确定数据库，T 是一个长度为 k 的元组向量并且满足 $q(T)\geqslant q_e$（$q_e\in[0,1]$），其中 q_e

是用户自定义的需求扩展阈值约束。令 $\Phi_k(W)$ 是 W 中按照打分函数逆序排列的前 k 个元组的集合；当 $|W|<k$ 时，$\Phi_k(W)=0$。RU-Top$k$ 查询返回一个序列 $R=\{T_{k,j}^{q_e^*}\}(1\leqslant j\leqslant \lfloor(1-q_e)k\rfloor+1)$，序列 R 中总共包含 $\lfloor(1-q_e)k\rfloor+1$ 个查询应答，而每个查询应答都是长度为 k 的元组向量，可表示为

$$T_{k,j}^{q_e^*}=\mathrm{argmax}_T\Big(\sum_{W\in W,\phi_k(W)=T,q(T)\geqslant q_e}(\Pr(W))\Big) \quad (1)$$

式中，$T_{k,j}^{q_e^*}$ 是在所有可能世界中，满足 q_e 的约束条件下，成为 top-k 聚合概率从大到小排名第 j 的长度为 k 的元组向量。特别地，若 $q_e=0$ 且 $j=1$，则 $T_{k,1}^{0^*}=\mathrm{argmax}_T\Big(\sum_{W\in W,\phi_k(W)=T}(\Pr(W))\Big)$，即为 U-Top$k$ 查询结果。

基于以上定义，RU-Top$k$ 查询算法的主要思想如下：

(1)用户可以根据自身查询需求设定需求扩展度的值($q_e=i/k$)，其用来约束 RU-Top$k$ 查询的结果集中最少应该包含的分值排名前 k 的元组的个数。

(2)RU-Top$k$ 查询结果可以看成由两部分组成，一部分来源于打分函数排名前 top-k 的元组，另一部分来源于剩下的元组。查询结果只能属于以下情况：

前 k 个元组取 i 个，第二部分中取 $k-i$ 个；

前 k 个元组取 $i+1$ 个，第二部分中取 $k-i-1$ 个；

……

前 k 个元组取 $k-1$ 个，第二部分中取 1 个；

前 k 个元组取 k 个，第二部分中取 0 个。

(3)RU-Top$k$ 算法采用分治的思想，在查询中把元组分为两部分来进行处理：第一部分是由打分函数排名前 k 的元组组成，剩下的所有元组归属到第二部分。对于需求④，同样地，第一部分可以取打分函数排名前 v 的元组，剩下的归为第二部分。

(4)将第一部分中的 k 个元组根据概率排序，由于元组间相互独立，因此先将概率值最大的 i 个元组找出来做乘积，再逐步按照概率逆序地将元组加入其中，并求乘积。

(5)对于第二部分，使用 OptU-Top$k$ 算法来计算 top-1 到 top $k-i$ 的值；OptU-Top$k$ 算法在计算 top k 元组的过程中，中间结果恰巧已经包含了对 top 1 到 top $k-1$ 的计算，只不过 OptU-Top$k$ 算法舍弃了这些计算结果。在 RU-Top$k$ 算法中，需要计算第二部分中 top 1 到 top $k-1$ 的值，因此使用 OptU-Top$k$ 算法思想，只需要稍加改进，就能在计算 top k 过程中，不增加时间复杂度的情况下，较快地获取 top 1 到 top $k-1$ 的值。

(6)最终将两个部分中对应的元组向量进行合并。

基于以上的思想和定义，RU-Top$k$ 算法的具体过程如下所述。

输入：已按打分函数逆序排序的元组数据流 TS；k；q_e；

输出：长度为 k 的元组向量组成的一个序列 $R=\{T_{k,j}^{q_e^*}\}$。

BEGIN

(1) 初始化长度为 k 的元组向量 $F_k=\{t_1,t_2,\cdots,t_k\}=D_k$；

(2) 用快速排序算法将 D_k 中元组按照概率排序，并用 $t_l^p(1\leqslant l\leqslant k)$ 标记，而后存储在栈 Sta 中；

(3) $T^0\leftarrow F_k$；//将前 k 个元组赋值给 T^0

(4) $\Pr(T^0)\leftarrow P_k=\prod_{i=1}^{k}\Pr(t_i)$；/ * 将前 k 个元组的概率乘积作为 T^0 的概率 * /

(5) while($1\leqslant i\leqslant k-\lceil q_e\cdot k\rceil$) {

(6) $t_{k-i+1}^p\leftarrow$Sta. pop()；//从栈中弹出元组概率最小的元组

(7) $P_{k-i}=P_{k-i+1}\cdot(1-\Pr(t_{k-i+1}^p))/\Pr(t_{k-i+1}^p)$；

/ * 栈中剩下的 k－i 个元组组成向量的概率 * /

(8) 长度为(k－i)的元组向量 $F_k\leftarrow$从 F_{k-i+1} 删除 t_{k-i+1}^p 元组；

(9) 通过调用空间搜索状态算法 OptU-Topk 计算 $\overline{D_k}$ 上的 i 元组向量 $T_{i,1}^{0^*}$，结果记为 L_i，概率记为 $\Pr(L_i)$；

(10) 得到长度为 k 的元组向量 $T_i\leftarrow$ 拼接 F_{k-i} 和 L_i；

(11) $\Pr(T^i)=P_{k-i}\cdot\Pr(L_i)$；} //end while

(12) 查询返回序列 $R=\{T_{k,j}^{q_e^*}\}\leftarrow$用快速排序算法根据 $\Pr(T_i)$ 给得到的所有 T_i 排序，其中，$0\leqslant i\leqslant k-\lceil q_e\cdot k\rceil=\lfloor(1-q_e)k\rfloor$；

END

在 RU-Top$k$ 算法中，初始时令 $T^0=F_k=D_k$，然后将 D_k 中的元组依据其置信度排序。使用快速排序算法花费 $O(k\log k)$的时间和 $O(\log k)$的空间开销。计算 P_i 序列总共花费 $O(k)$的时间开销。文献[11]不仅证明了 OptU-Top$k$ 算法时空复杂度分别为 $O(nk)$ 和 $O(k^2)$，还证明了 OptU-Top$k$ 算法的扫描深度 n 与打分函数和元组概率分布有关。OptU-Top$k$ 算法采用搜索状态空间的策略来计算 $T_{k,1}^{0^*}$，与此同时，$T_{i,1}^{0^*}$ $(1\leqslant i\leqslant k-1)$作为算法的中间结果也得到了。因此，计算 $T_{i,1}^{0^*}$ $(1\leqslant i\leqslant k)$总的时间花销是 $O(nk)$。存储向量 $T^i(i\in[0,k])$ 需要 $O(k^2)$ 空间。最后，RU-Top$k$ 算法使用快速排序将向量 T^i 按照其概率 $Pr(T^i)$ 排序的时间开销为 $O(k\log k)$。综上所述，RU-Top$k$ 算法的时间开销为 $O(nk)$，空间开销为 $O(k^2)$。

使用上述的应用场景中的一个实例来展示 RU-Top$k$ 的查询过程，如表 2 所列。在表 2 中的不确定数据只是元组数据流的一部分，这些元组已按照打分函数排序。为简便起见，在本例输入中，忽略了元组的打分函数，并采用小数来表示各元组的置信度。本例的查询目标是满足 20%需求扩展度的 RU-Top4 查询。长度为 4 的元组向量 $T4,i0.6^*$ 组成的序列即为 RU-Top$k$ 算法的查询结果。以上实例显而易见，RU-Top$k$ 算法是有效的，并能在一次查询中返回符合用户需求的多个查询应答。

表 2　RU-Top$k$ 算法实例

input	score-ranked tuple stream：t_1:0.3；t_2:0.2；t_3:0.8；t_4:0.6；t_5:0.4；t_6:0.7；t_7:0.6；t_8:0.1；t_9:0.2 k=4；q_e=20%	
divide and conquer	$D_k=\{t_1,t_2,t_3,t_4\}$	$\overline{D_k}=\{t_5,t_6,t_7,t_8,t_9\}$
Initialize (i=0)	$F_4=\{t_1,t_2,t_3,t_4\}=T^0$， $P_4=\Pr(t_1)\Pr(t_2)\Pr(t_3)\Pr(t_4)=0.0288$	
Sort t	$Sta(t_l^p)$：$\{t_3,t_4,t_1,t_2\}$	
i=1	$P_3=P_4(1-\Pr(t_2))/\Pr(t_2)=0.1152$ $F_3=\{t_1,t_3,t_4\}$	$L_1=\{t_6\}$ $\Pr(L_1)=(1-0.4)\times0.7=0.42$
	$T^1=\{t_1,t_3,t_4,t_6\},\Pr(T^1)=0.1152\times0.42=0.048384$	
i=2	$P_2=P_3(1-\Pr(t_1))/\Pr(t_1)=0.2688$ $F_2=\{t_3,t_4\}$	$L_2=\{t_5,t_6\}$ $\Pr(L_2)=0.4\times0.7=0.28$
	$T^2=\{t_3,t_4,t_5,t_6\},\Pr(T^2)=0.2688\times0.28=0.075264$	
i=3	$P_1=P_2(1-\Pr(t_4))/\Pr(t_4)=0.1792$ $F_1=\{t_3\}$	$L_3=\{t_5,t_6,t_7\}$ $\Pr(L_3)=0.4\times0.7\times0.6=0.168$
	$T^3=\{t_3,t_5,t_6,t_7\},\Pr(T^3)=0.1792\times0.168=0.0301056$	
Sort T	$T_{4,1}^{0.6^*}=T^2=\{t_3,t_4,t_5,t_6\},\Pr(T^2)=0.075264$ $T_{4,2}^{0.6^*}=T^1=\{t_1,t_3,t_4,t_6\},\Pr(T^1)=0.048384$ $T_{4,3}^{0.6^*}=T^3=\{t_3,t_5,t_6,t_7\},\Pr(T^3)=0.0301056$ $T_{4,4}^{0.6^*}=T^0=\{t_1,t_2,t_3,t_4\},\Pr(T^0)=0.0288$	

4 实验

为比较 RU-Top$k$ 算法和传统的 U-Top$k$ 算法的查询结果并分析 RU-Top$k$ 语义算法的效率，我们进行了实验验证。实验采用 VS2005 软件为开发工具，运行环境为 PC 机，配置为 CPU Intel Core 2，2.66 GHz，内存为 2GB，操作系统为 Windows XP。在实验中，由数学软件 MATLAB7.0 生成模拟数据。打分函数满足[0，1]上的均值分布；元组的置信度分别服从概率为均匀分布、高斯分布、指数分布，每一个查询都随机生成满足相应概率分布的 10000 个元组。如表 3 所列，0e0.5、0e0.2、0u0.5、0u0.9、0uu 分别表示元组概率服从指数分布、高斯分布、[0，1]上均匀分布的数据集在 U-Top$k$ 语义下算法的实验结果，而 e0.5、e0.2、u0.5、u0.9、uu 则代表相应的数据集在 RU-Topk 语义下的实验结果。

表 3 数据集

元组概率分布	均值	方差	U-Topk 算法	RU-Topk 算法
均匀分布	0.5	无	0uu	uu
高斯分布	0.5	0.2	0n0.5	n0.5
高斯分布	0.9	0.2	0n0.9	n0.9
指数分布	0.5	无	0e0.5	e0.5
指数分布	0.2	无	0e0.2	e0.2

传统 U-Top$k$ 查询只返回一个查询应答，而 RU-Top$k$ 在一次查询中返回符合用户需求的多应答，二者的信息量存在较大的差别。运行时间是衡量算法性能的一个重要指标，为了更为准确地评价一个算法，引入一个新指标——平均单位查询运行时间(perAnswer_runtime)来衡量获得查询的性能，计算方法如下：

平均单位查询应答运行时间＝ 一次查询的运行时间/一次返回应答的个数

图 1 的实验结果展示了 U-Top$k$ 和 RU-Top$k$ 算法($q_e=0$ 时)在不同元组概率分布下，平均单位查询运行时间的对比。

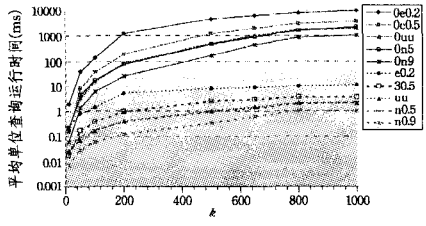

图 1 不同概率分布下 k 与平均单位查询运行时间

由实验可知，在各种数据集元组的概率分布下，RU-Top$k$ 算法相对于传统算法都具有较小的平均单位查询运行时间，即在满足用户需求的情况下，同样的时间可提供更多的信息量，具备更高的查询效率。

RU-Top$k$ 语义引入了需求扩展度来表达用户差异化的查询需求，通过实验来探讨需求扩展度的选取对查询性能的影响。RU-Top$k$ 算法的时间复杂度为 $O(nk)$，而 n 为 RU-Top$k$ 算法分层迭代扫描部分的搜索深度，即为了取得正确的查询结果最少需要扫描的层数。在 RU-Top$k$ 与 U-Top$k$ 算法中，分层迭代扫描部分要在庞大的可能世界集合中搜索，是阻碍算法性能提高的瓶颈。因此，在 k 值一定的情况下，倘若能有效减小扫描深度，就能有效提高查询算法效率。图 2 反映了当 $k=200$ 时，在不同的概率分布下 RU-Top$k$ 算法的扫描深度随着用户选取不同的需求扩展度约束而变化的情况。图 2 表明，随着需求扩展度的减少，在相同的 k 值下，扫描深度在不断地增加。这是因为需求扩展度减少时，对查询内在的约束实际上是降低了，因而符合用户查询需求的可能世界增多，需要搜索的范围也变大。在 RU-Top$k$ 算法中，用户通过选定需求扩展度约束，能快速过滤其不关心的信息，极大地缩减搜索空间，有效提高查询性能。比如在 e0.2 中，q 为 85%，扫描深度从 3000 锐减到 300，提升了查询效率。其次，由图 2 可知，对于元组概率的均值越低、分布得越零散的数据集而言，这种剪枝优化的效果越明显。

图 3 反映了扫描深度比值受到需求扩展度变化的影响情况。其中，扫描深度比是指在同一个 Top-k 查询中，当前需求扩展度对应的扫描深度与无约束状况下($q_e=0$)的扫描深度的比值。图 3 显示了需求扩展度与扫描深度的变化呈线性相关。因而，通过无约束情况下($q_e=0$)RU-Top$k$ 算法的执行情况，能较为容易地估计出在其他需求扩展度约束条件下的情况。

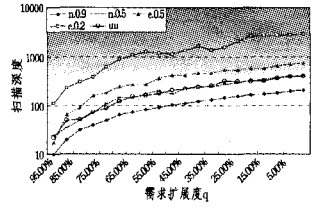

图 2 不同概率分布下需求扩展度与扫描深度

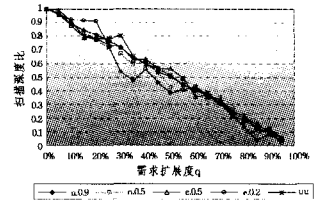

图 3 不同概率分布下需求扩展度与扫描深度

结束语 本文根据不确定数据 top-k 查询的特点，探讨了用户在不确定数据 top-k 查询中的深层需求，并引入了能精确反映用户需求的指标，由此提出了一个新的不确定数据 top-k 查询语义 RU-Top$k$，并给出了对应的有效算法。文中通过实验得出该算法具有较小的平均单位查询时间，在考虑用户需求的情况下，可以具有更高的查询效率。下一步的研究工作将主要考虑以下问题：本文仅讨论了元组间相互独立的情况，对于互斥以及其他关联关系，将做进一步探讨；继续完善实验，以更全面地验证 RU-Top$k$ 算法的时间性能和查询效果；探讨将用户需求扩展度引入到其它不确定数据 top-k 语义(如 U-kRanks)中。

参 考 文 献

[1] Cheng R, Kalashn I D, Prabhakar S. Evaluating probabilistic queries over imprecise data [C]// Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data. New York: ACM Press, 2003, 551-562

[2] Soliman M A, Ilyas I F, Chang K C. Top-k query processing in uncertain databases [C]// Proceedings of the 23rd IEEE International Conference on Data Engineering. Istanbul, 2007: 896-905

[3] Chen Jiang, Yi Ke. Dynamic structures for top-k queries on uncertain data [C]// Proceedings of the 18th International Symposium on Algorithms and Computation. Sendai, 2007: 427-438

[4] 周帆，李树全，肖春静，等. 不确定数据 Top-k 查询算法[J]. 电子测量与仪器学报，2010，24(7)：650-656

[5] Hua Ming, Pei Jian, Zhang Wen-jie, et al. Efficiently answering probabilistic threshold top-k queries on uncertain data [C]// Proceedings of the 24th IEEE International Conference on Data Engineering. 2008: 1403-1405

(下转第 174 页)

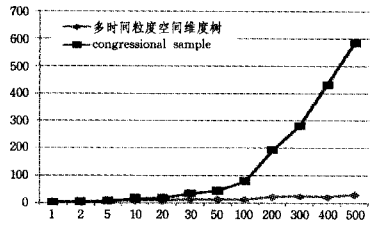

图 6　不同数据插入频率下，查询时间和更新时间测量结果

由图可知，随着数据插入频率的增长，congressional sample算法的增长幅度较大，而多时间粒度空间维度树算法由于采用近似聚集查询思路，因此查询相应时间的变化则不大。

3.2　多时间粒度空间维度树查询误差实验及分析

图 7 和表 1 表示，随机查询下的准确聚集结果以及使用多时间粒度空间维度树得到的近似聚集结果及误差。

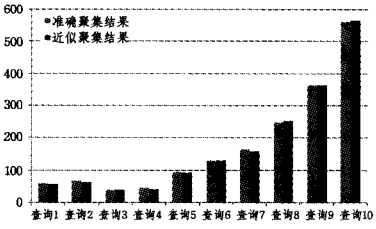

图 7　随机查询下的准确聚集结果以及近似聚集结果

表 1　随机查询下的准确聚集结果、近似聚集结果及其误差

	准确聚集结果(ms)	近似聚集结果(ms)	误差(%)
查询 1	59	57	3.39
查询 2	68	65	4.41
查询 3	39	39	0.00
查询 4	46	42	8.70
查询 5	95	92	3.26
查询 6	130	131	7.69
查询 7	165	159	3.64
查询 8	250	253	0.12
查询 9	365	365	0.00
查询 10	563	566	5.33

由图 7 和表 1 可以看出，使用多时间粒度空间维度树的近似聚集结果的误差最低为 0%，最高为 8.7%。

3.3　传统数据仓库与列式数据仓库性能对比

图 8 表示多时间粒度空间维度树在传统行式数据仓库和列式数据仓库环境中，对于不同数据集进行同一种查询的相应时间对比。图中，横坐标表示已有数据集的大小，纵坐标表示响应时间，单位为 ms。

由图 8 可知，数据集在 1M～1G 的情况下，两种数据仓库的查询响应时间还没有明显差距。但当数据集大小为 10G 或者更大的情况下，响应时间的差距就十分明显，行式数据仓库环境下的查询时间增长较快，而列式数据仓库则比较平缓。

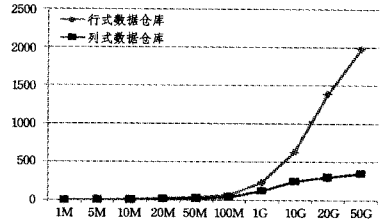

图 8　不同数据集下的行式/列式数据仓库查询响应时间

结束语　在数据仓库环境中，由于大数据量以及查询的复杂性使得一个查询的执行通常需要很长时间，并且目前对实时查询的需求越来越迫切，因此，近似查询技术成为处理该类查询的一个有效手段。本文结合 RFID 数据的特点以及聚集查询技术设计了支持连续不同时空粒度的聚集查询的数据结构——多时间粒度空间维度树，用以对 RFID 数据的聚集信息进行快速索引，以去除部分在连续实时聚集查询中的冗余操作。通过实验证明，该模型与算法在不同数据集大小以及数据插入频率的情况下，查询响应时间较小，符合对于 RFID 数据在列式数据仓库中进行实时查询的基本要求。但是，本文的模型和算法还存在许多不足，比如模型的构造是静态的，无法动态改变其结构；模型的查询效率较高，但更新时间仍会随着数据集大小以及数据插入频率的增加而显著提高。这些不足都是我们下一阶段改进的重点。

参 考 文 献

[1] Want R. An Introduction to RFID Technology[J]. Pervasive Computing, IEEE, 2006(6): 25-33

[2] 李战怀，聂艳明，陈群，等. RFID 数据管理的研究进展[J]. 中国计算机学会通讯，2007(8)

[3] Abadi D J, Boncz P A, Harizopoulos S. Column-oriented Database Systems[C]// VLDB '09. 2009: 24-28

[4] Han Jia-wei, Gonzalez H, Li Xiao-lei, et al. Warehousing and Mining Massive RFID Data Sets[C]// ADMA. 2006: 1-18

[5] Wang Fu-sheng, Liu Pei-ya. Temporal Management of RFID Data[C]// Proceedings of the 31st VLDB Conference. 2005: 1128-1139

[6] Chawathe S S, Krishnamurthy V, Ramachandrany S, et al. Managing RFID Data[C]// VLDB. 2004: 1189-1195

[7] Chawathe S S, Krishnamurthy V, Ramachandrany S, et al. Managing RFID Data[C]// VLDB. 2004: 1189-1195

[8] Acharya S, Gibbons P B, Poosala V. Congressional Samples for Approximate Answering of Group-By Queries[C]// SIGMOD. 2000

(上接第 154 页)

[6] Jin Che-qing, Yi Ke, Chen Lei, et al. Sliding-window top-k queries on uncertain Streams [J]. Proceedings of the 34th International Conference on Very Large Data Bases, 2008, 1(1): 301-312

[7] Hua Ming, Pei Jian, Zhang Wen-jie, et al. Ranking queries on uncertain data: A probabilistic threshold approach [C]// SIGMOD. 2008: 673-687

[8] Yi Ke, Li Fei-fei, Kollios G. Efficient Processing of Top-k Queries in Uncertain Databases with X-Relations [J]. Knowledge and Data Engineering, 2009, 21(1): 1-14

[9] 刘德喜，万常选，刘喜平. 不确定数据库中基于 X-tuple 的高效 Top-k 查询处理算法[J]. 计算机研究与发展，2010，47(8)：1415-1423

[10] 王爽，王国仁. 基于不确定数据的分布式 Top-k 查询算法[J]. 东北大学学报：自然科学版，2010，31(2)：177-180

[11] 王晓伟，贾焰，杨树强，等. 存在级不确定数据上的概率 Skyline [J]. 计算机研究与发展，2011，48(1)：68-76