TKEP:海量数据上一种有效的 Top-K 查询处理算法

韩希先[1)] 杨东华[2)] 李建中[1)]

[1)](哈尔滨工业大学计算机科学与技术学院 哈尔滨 150001)

[2)](哈尔滨工业大学基础与交叉科学研究院高性能计算中心 哈尔滨 150001)

摘 要 在许多应用领域中,top-k 查询是一种十分重要的操作,它根据给定的评分函数在潜在的巨大的数据空间中返回 k 个最重要的对象.不同于传统的 TA 算法,NRA 算法只需要顺序读就可以处理 top-k 查询,从而适合于随机读受限或不可能的场合.文中详细地分析了 NRA 算法的执行行为,确定了增长阶段和收缩阶段中每个文件需要扫描的元组个数.文中发现在海量数据环境中,NRA 在增长阶段需要维护大量的候选元组,严重影响了算法的执行效率.所以,文中提出一种新的海量数据上的 top-k 查询算法 TKEP,该算法在查询的增长阶段就执行早剪切,从而大大减少增长阶段需要维护的候选元组.文中给出了早剪切操作的数学分析,确定了早剪切操作的理论和实际剪切效果.据作者所知,该文是第一篇提出在 top-k 查询的增长阶段执行早剪切的文章.实验结果表明,和传统的 NRA 相比,TKEP 在增长阶段维护的元组数量减少 3 个数量级,需要的内存量减少 1 个数量级,TKEP 算法获得 1 个数量级的加速比.

关键词 海量数据;top-k;早剪切;TKEP

中图法分类号 TP311 **DOI 号**:10.3724/SP.J.1016.2010.01405

TKEP: An Efficient Top-K Query Processing Algorithm on Massive Data

HAN Xi-Xian[1)] YANG Dong-Hua[2)] LI Jian-Zhong[1)]

[1)](*School of Computer Science and Technology, Harbin Institute of Technology, Harbin* 150001)

[2)](*Center for High Performance Computing, The Academy of Fundamental and Interdisciplinary Sciences, Harbin Institute of Technology, Harbin* 150001)

Abstract In many application fields, top-k is an important operation since it returns k most important objects according to a given ranking function. Different from traditional TA algorithms, NRA only requires sequential access to return top-k results so that it can be used in environment where random access is limited or impossible. This paper analyzes the execution behavior of NRA and determines tuple number to scan in increasing and shrinking phase. It is found that in massive data context, NRA needs to maintain large quantity of candidate tuples in increasing phase which affects algorithm efficiency significantly. This paper proposes a novel top-k algorithm TKEP (Top-K with Early Pruning) on massive data which performs early pruning in increasing phase to prune most of candidate tuples. This paper provides mathematical analysis of early pruning and proves its theoretical and practical pruning effect. To the best of our knowledge, it is the first paper to provide early pruning in top-k processing. The extensive experiments show that compared to NRA, TKEP maintains less tuples by a factor of three orders of magnitude, it consumes less memory by a factor of an order of magnitude and TKEP achieves substantial performance speed-up of an order of magnitude.

Keywords massive data; top-k; early pruning; Top-K with Early Pruning

收稿日期:2010-06-11.本课题得到国家“九七三”重点基础研究发展规划项目基金(2006CB303005)、国家自然科学基金(60903016, 60533110, 60773063)、新世纪优秀人才支持计划(NCET-05-0333)、黑龙江省教育厅科学技术研究项目(11531276)和 NSFC-RGC of China (60831160525)资助.**韩希先**,男,1981 年生,博士研究生,主要研究方向为海量数据管理和数据密集型计算. E-mail: xxhan1981@163.com. **杨东华**,男,1976 年生,博士,讲师,主要研究方向为海量数据管理和数据密集型计算等. **李建中**,男,1950 年生,教授,博士生导师,主要研究领域为数据库和传感器网络等. E-mail: lijzh@hit.edu.cn.

1 引 言

在许多应用领域中,例如 Web 搜索、信息检索、多媒体数据库和数据挖掘等,top-k 查询是一种十分重要的操作,它根据给定的评分函数在潜在的数据空间中返回 k 个最重要的结果.处理 top-k 查询时,系统通常根据用户的要求对涉及到的每个对象的属性计算一个数值来反映该属性的特点,然后利用一个单调评分函数聚集多个属性的数值作为该对象的权重,top-k 查询返回 k 个权重最高的对象作为查询的结果.

处理 top-k 查询的一种方式是利用多维索引,可是当查询涉及的属性超过 6 时,多维索引的查询性能急剧下降(curse of dimensionality)[1].相对于多维索引,另一种更具有扩展性的方法是把数据表列存储化并且按属性值降序排列存储.Fagin 等[2]提出 TA 算法在多个有序文件上执行 top-k 查询.top-k 查询处理涉及到两种数据读方式来获得对象 x 在指定列文件的数值:顺序读和随机读.顺序读意味着处理完该文件中分数比 x 大的元组后可以获得 x 的分数,而随机读则直接获得该文件中 x 对应的属性值.在某些情况下,例如数据流和海量数据环境,随机读操作是受限的或过于昂贵的,人们利用 NRA 算法[3-6]来处理只支持顺序读的 top-k 查询.NRA 的执行可以分为两个阶段:增长阶段(不断累积 top-k 候选元组直到找到 top-k 结果的阈值)和收缩阶段(不断剪切候选元组直到获得最终 top-k 结果).由于属性文件的降序排列和评分函数的单调性,NRA 算法通常不需要处理完属性文件所有数据就可以结束查询并返回结果.随着涉及的属性数量、元组数量和返回的结果数量的增大,NRA 的增长阶段需要维护的候选元组也越来越多,其维护费用也越来越大.尤其是增长阶段维护的数据超过给定的内存容量时,NRA 算法将不得不把内存中的数据反复输出到磁盘,这必然会引起较高的费用,从而影响 top-k 查询的执行效率.

本文详细地分析了 NRA 算法的执行行为,确定了增长阶段和收缩阶段中每个文件需要扫描的元组数,同时还分析了 NRA 算法在增长阶段需要维护的候选元组的数量.分析表明,在海量数据上,NRA 在增长阶段需要维护大量的候选元组,严重影响了算法的执行效率.基于对 NRA 行为的分析,本文提出一种新的海量数据上的 top-k 查询处理算法 TKEP(Top-K with Early Pruning).该算法和现有 NRA 算法的最大不同在于,该算法可以在查询的增长阶段就执行早剪切,从而减少增长阶段需要维护的元组数量.本文给出早剪切操作的数学分析,确定了早剪切的理论和实际剪切效果.指数间距 bloom filter 表(EGBFT)是 TKEP 算法执行剪切时用到的数据结构,EGBFT 的第 j 个元组是构建在对应文件的第 1 到第 2^j 元组上的 bloom filter.本文提供了较全面的实验来评价 TKEP 的性能,实验结果表明,和传统的 NRA 相比,TKEP 在增长阶段维护的元组数量减少 3 个数量级,需要的内存量减少 1 个数量级,TKEP 算法获得 1 个数量级的加速比.

本文的主要贡献在于:

(1)本文提出了一种新的海量数据上的 top-k 算法 TKEP.不同于传统的 NRA 算法,TKEP 算法在增长阶段就开始执行早剪切操作,从而大大减少了需要维护的候选元组.据我们所知,本文是第一篇提出在 top-k 查询中执行早剪切的文章.

(2)本文给出了 NRA 算法的行为分析,确定了增长阶段和收缩阶段需要扫描的元组数以及增长阶段需要维护的候选元组数量.

(3)本文提出了早剪切操作的数学分析,并且确定了早剪切操作的理论和实际剪切效果.

(4)本文利用较全面的实验比较了 TKEP 和 NRA 算法的性能.实验表明,不管在执行时间还是内存消耗方面,TKEP 比 NRA 算法都具有较大的优势.

本文第 2 节给出本文的背景和问题定义;第 3 节给出 NRA 的行为分析;TKEP 算法在第 4 节中介绍;实验和性能分析在第 5 节中给出;第 6 节和第 7 节分别是相关工作和本文的结论.

2 背景和问题定义

给定具有 N 个元组的表 T,每个元组具有 M 个属性 $A_1, A_2, \cdots, A_M$.表 T 存储为列文件集合 $LS=\{L_1, L_2, \cdots, L_M\}$,每个列文件 L_i 的模式为 $L_i(RID, A_i)$,其中 RID 表示元组的标识符,A_i 表示元组在属性 A_i 上的值.$\forall t \in T$,如果 $t.RID=rid_i$,$t.A_i=a_i$,则 $(rid_i, a_i) \in L_i$.L_i 根据 A_i 的值降序排列.F 是定义在 m 个属性上的评分函数,不失一般性,假设 F 定义在 $A_1, A_2, \cdots, A_m$,$F(t)=\sum_{i=1}^{m}(w_i \times t.A_i)$,$w_i$ 是评分函数 F 定义在属性 A_i 上的权重.通常,F 是单调函数,即 $\forall t_1, t_2 \in T$,如果对所有有 $1 \leq i \leq m$,$t_1.A_i \leq t_2.A_i$,那么 $F(t_1) \leq F(t_2)$.

Top-k 查询. 给定表 T 和涉及到 m 个属性的评分函数 F，top-k 查询返回表 T 的 k 子集 $R(k<N)$，$\forall t_i \in R$，$\forall t_j \in (T-R)$，$F(t_i) \geq F(t_j)$. 通常，$m \leq 10$[7].

经典的 Threshold Algorithm(TA)算法要求同时支持顺序读和随机读. TA 以 round-robin 方式读取 m 个列文件的元组，如果元组 t 第一次出现，则利用随机读从其它 $(m-1)$ 个列文件中获得 t 的其它属性值，得到 t 的分数 $F(t)$. TA 利用优先队列 P_Q 维护分数最大的 k 个元组，令 $P_Q.min$ 表示 P_Q 中最小的分数. 假设所有列文件最后读取的元组为 (rid_1, a_1)，…，(rid_m, a_m)，则 TA 算法的阈值 threshold 定义为 $\tau = F(a_1, \cdots, a_m)$. 随着列文件的扫描，TA 算法不断更新 P_Q 和 τ. 当 $P_Q.min \geq \tau$ 时，P_Q 维护的就是 top-k 的结果，TA 算法结束.

当 m 个列文件可以放入内存时，TA 算法可以处理 top-k 查询. 但是，在海量数据环境中，内存甚至无法完全容纳单个列文件. 此时，top-k 查询的列文件都存储在磁盘，每次随机读需要完全扫描其他 $(m-1)$ 个列文件，费用极高. NRA 算法只需要顺序读就可以完成 top-k 查询. NRA 以 round-robin 方式依次读取 m 个列文件的元组，并且维护见到的每个元组 t 的上界 F_t^{ub} 和下界 F_t^{lb}，其中 $F_t^{ub} = \sum_{i=1}^{m}(w_i \times t.A_{iu})$，在 NRA 的处理过程中，如果 $t.A_i$ 已经出现，则 $t.A_{iu} = t.A_i$，否则 $t.A_{iu} = MAX(A_i)$，其中 $MAX(A_i)$ 表示属性 A_i 的最大值；$F_t^{lb} = \sum_{i=1}^{m}(w_i \times t.A_{il})$，在 NRA 的处理过程中，如果 $t.A_i$ 已经出现，则 $t.A_{il} = t.A_i$，否则 $t.A_{il} = MIN(A_i)$，其中 $MIN(A_i)$ 表示属性 A_i 的最小值. NRA 利用优先队列 P_Q 维护 F_t^{lb} 最大的 k 个元组. NRA 算法阈值 τ 的定义和 TA 算法的类似. NRA 的执行分为两个阶段：增长阶段和收缩阶段. 增长阶段利用 Hash 表维护见到的候选元组集合 C 直到 $P_Q.min \geq \tau$，此时我们可以肯定 top-k 结果包括在 C 中；在收缩阶段，NRA 逐渐剪切候选元组并且更新 P_Q，当 $P_Q.min \geq F_t^{ub}$ ($\forall t$, $t \in (C-P_Q)$)时，P_Q 维护的就是 top-k 的结果，NRA 算法结束.

虽然 NRA 只需要顺序读来处理 top-k 查询，但是它也存在着如下问题：NRA 在增长阶段累积候选元组，直到满足给定阈值条件，根据下节的分析，这使得增长阶段将维护大量的数据，甚至超过内存容量. 此时，NRA 算法将不得不把内存中的数据反复输出到磁盘，这会引起较高的费用. TKEP 算法采用早剪切的方法在增长阶段就执行剪切操作，从而减少了增长阶段需要的内存量，加快了 top-k 查询的执行.

3　NRA 行为分析

我们先给出本节需要用到的定义.

定义 1(位置索引).　给定表 T，元组 $t \in T$ 的位置索引 PI(Positional Index)是 i，如果 t 是 T 的第 i 个元组.

我们用 $T(i)$ 来表示表 T 中位置索引为 i 的元组. 令 T_1 和 T_2 分别表示增长阶段和收缩阶段每个列文件需要扫描到的位置索引值. 3.1 节介绍如何确定 T_1 和 T_2，3.2 节分析 NRA 在增长阶段需要维护的候选元组数. 由于本文重点在于讨论早剪切操作，为更清晰地分析问题，假设评分函数 F 的权重都为 1，属性在值域[0,1]内均匀分布并且属性间独立分布.

3.1　确定 T_1 和 T_2

保守地估计，当有 k 个元组前 m 个属性 A_1，…，A_m 都在增长阶段出现时，增长阶段结束，因为此时肯定满足条件 $P_Q.min \geq F(L_1(T_1).A_1, \cdots, L_m(T_1).A_m)$. 令 T_1' 表示 T_1 的估计值，T_1' 的值对应着需要扫描 m 个列文件的元组个数使得有 k 个元组的对应 m 个属性都出现. $\forall t \in T$，$P(t.A_i \geq L_i(T_1').A_i) = T_1'/N$，$t.A_1$，…，$t.A_m$ 同时分别出现在 L_1，…，L_m 的前 T_1' 个元组的概率 $p = (T_1'/N)^m$.

如果把任意元组 t 的前 m 个属性出现在列文件 L_1，…，L_m 的前 T_1' 个元组的结果看作一次成功的事件 S，则成功的次数满足二项式分布 $BD(N,p)$，该二项式分布的期望是 Np. 要获得 T_1' 的值，一种简单的方法是令 $Np=k$，可以得出 $T_1' = N \times (k/N)^{\frac{1}{m}}$，可是分布 BD 的方差是 $Np(1-p)$，该方法无法保证能够获得至少 k 个满足条件的元组. 根据棣莫弗-拉普拉斯定理，当 n 较大时，参数为 (n,p) 的二项分布可以很好地近似为均值为 np、方差为 $np(1-p)$ 的正态分布. 由于海量数据中，N 很大，$p = (T_1'/N)^m$ 很小，所以分布 BD 可以用正态分布 $ND(\mu, \sigma^2)$ 来代替，其中 $\mu = Np$，$\sigma^2 = Np(1-p)$.

定理 1.　$T_1' = N \times [(-b + \sqrt{b^2 - 4ac})/(2a)]^{\frac{1}{m}}$ 以 99.9968% 的概率保证事件 S 成功的次数不少于 k，其中，$a = N^2 + 16N$，$b = -(2Nk + 16N)$，$c = k^2$.

证明.　正态分布 $ND(\mu, \sigma^2)$ 如图 1 所示. 已知

$$\int_{\mu-4\sigma}^{+\infty} N(\mu, \sigma^2)\,\mathrm{d}x = 99.9968\%.$$

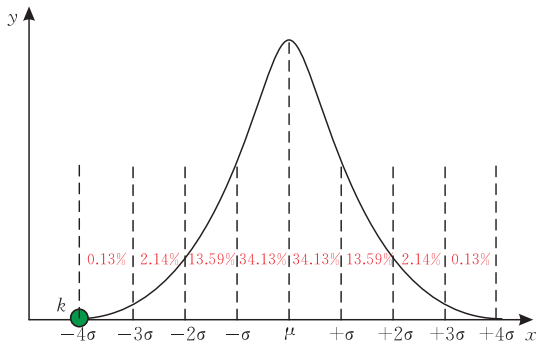


图 1 正态分布示例

如果 $k=\mu-4\sigma$，那么以 99.9968% 的概率可以保证正态分布 $ND(\mu,\sigma^2)$ 的 x 轴的值不小于 k，即事件 S 成功的次数不少于 k. $(k=\mu-4\sigma)\Rightarrow(k=[Np-4\sqrt{Np(1-p)}])\Rightarrow(N^2+16N)p^2-(2Nk+16N)p+k^2=0$.

解上述一元二次方程得

$$p=(-b+\sqrt{b^2-4ac})/(2a),$$

其中，$a=N^2+16N$，$b=-(2Nk+16N)$，$c=k^2$. 由 $p=(T_1'/N)^m$，得 $T_1'=N\times[(-b+\sqrt{b^2-4ac})/(2a)]^{\frac{1}{m}}$（解方程时我们丢弃了另一个不合理的解）. 证毕.

增长阶段结束后，NRA 进入收缩阶段. 我们同样保守地获得 T_2 的估计值 T_2'.

定理 2. $T_2'=mT_1'$ 时，NRA 算法结束.

证明. 已知，当增长阶段结束时，$P_Q.min\geq F(L_1(T_1').A_1,\cdots,L_m(T_1').A_m)$. 在收缩阶段，$P_Q.min$ 的值单调递增. 令 C 表示增长阶段维护的候选元组集合，当 $P_Q.min\geq F_t^{ub}(\forall t,t\in(C-P_Q))$，NRA 算法结束. 令 P_m 表示出现 m 个属性的元组集合. 可以肯定，如果 $F(L_1(T_1').A_1,\cdots,L_m(T_1').A_m)\geq F_t^{ub}(\forall t,t\in(C-P_m))$，$P_Q\subseteq P_m$，则 NRA 算法结束. 因为，$\forall t\in P_m-P_Q$，必然有 $P_Q.min\geq F(t)$.

$\forall t\in(C-P_m)$，要排除 t，需要扫描 m 个列文件的最大元组数 $T_{\max}$ 对应着如下情况：t 的 $(m-1)$ 个属性取最大值 1，剩余一个属性值较小，即 $F(L_1(T_1').A_1,\cdots,L_m(T_1').A_m)=(1-T_1'/N)\times m\geq F_t=1\times(m-1)+(1-T_{\max}/N)$，解此不等式，我们得到 $T_{\max}\geq N+N\times(m-1)-m\times(N-T_1')=mT_1'$，则当 $T_2'=mT_1'$ 时，NRA 算法结束. 证毕.

定理 1 和定理 2 证明 T_1' 和 T_2' 以极高的概率（99.9968%，根据后文早剪切的操作方法，实际的概率还要远高于该值）保证 $T_1'\geq T_1$，$T_2'\geq T_2$，所以本文的剩余部分认为 T_1' 和 T_1 可以互换，T_2' 和 T_2 可以互换.

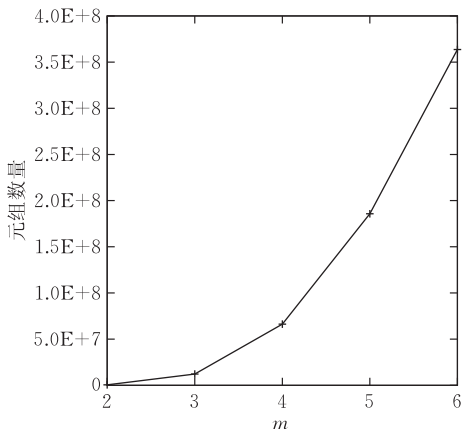
3.2 NRA 在增长阶段需要维护的数据量

NRA 算法在增长阶段不对元组进行剪切[4-5]，由定理 1 知，$T_1'=N\times p^{1/m}$，$p=(-b+\sqrt{b^2-4ac})/(2a)<1$，则 T_1' 的值随着 m 的增大而指数级增长，随着 N 和 k 的增大而多项式级增长.

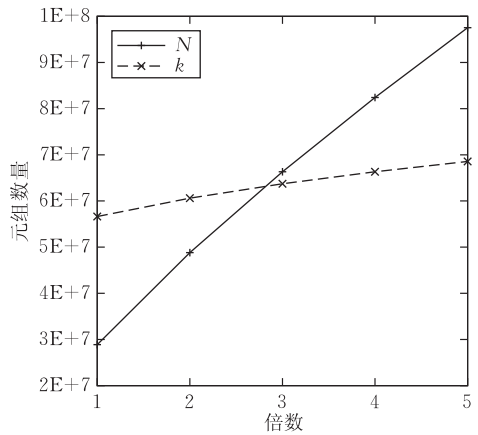
接下来，我们分析在增长阶段 NRA 需要维护的元组数，从而更清楚地理解 NRA 在增长阶段所需要的内存量. $\forall t\in T$，$P(t.A_i\geq L_i(T_1').A_i)=T_1'/N(1\leq i\leq m)$，令 $NUM(t,T_1')$ 表示 t 满足条件 $t.A_i\geq L_i(T_1').A_i$ 的属性个数，则 $NUM(t,T_1')$ 满足二项式分布 $BD_2(m,T_1'/N)$，$P(NUM(t,T_1')=m')=\binom{m}{m'}\times(T_1'/N)^{m'}\times(1-T_1'/N)^{(m-m')}$，所以 NRA 在增长阶段维护的元组数量

$$NUM_{\text{tuple}}=\sum_{i=1}^{m}[N\times P(NUM(t,T_1')=i)].$$

图 2(a)说明了当 $N=12\times10^8$，$k=20$ 时，随着 m 的变化，增长阶段需要维护的元组数量. 可以看到，随着 m 的增长，维护的元组数量呈指数级增长，



(a) m 变化的效果($N=12\times10^8$, $k=20$)



(b) $N(k)$ 变化的效果($m=4$)

图 2 增长阶段维护的元组个数

其增长趋势和 T_1' 的增长趋势一致. 图 2(b)说明了当 $m=4$ 时，N 和 k 的变化对维护元组数量的影响. 固定 $k=20$，随着 N 从 4×10^8 增长到 20×10^8，元组数量基本呈线性增长；固定 $N=12\times10^8$，随着 k 从 5 增长到 25，元组数量也呈线性增长，但是其增长速度较慢.

总的来看，由于传统的 NRA 算法在增长阶段不执行剪切操作，在海量数据上执行 top-k 查询时，增长阶段维护大量的候选元组，从而严重影响查询的执行效率. 接下来介绍 TKEP 算法如何在增长阶段执行早剪切操作.

4　TKEP 算法

本节主要介绍 Top-K with Early Pruning (TKEP)算法. 4.1 节给出了早剪切的数学分析，4.2 节介绍了指数间距 bloom filter 表的构造，4.3 节描述如何实现 TKEP 算法.

4.1　早剪切的数学分析

早剪切在 top-k 的增长阶段就开始对每个候选元组执行剪切操作，从而尽可能地减少 top-k 处理需要维护的候选元组，同时加快 top-k 的执行速度.

早剪切基本规则. 由定理 2，$\forall t\in C$，如果 $\exists i(1\le i\le m)$，$t.A_i<L_i(T_2').A_i$，t 肯定不是 top-k 的结果.

如果 t 的属性在增长阶段出现了 i 次，t 无法被早剪切的概率是

$$P_i=\left(\frac{T_2'-T_1'}{N-T_1'}\right)^{(m-i)}.$$

P_i 的计算公式解释如下：假设元组 t 有 i 个属性出现在对应列文件的前 T_1' 个元组，要使得 t 不会被早剪切，则 t 的其余 $m-i$ 个属性必须全部在对应的列文件的第 T_1' 和第 T_2' 元组之间出现. 给定 $t.A_i\le L_i(T_1').A_i$，$P(L_i(T_2').A_i\le t.A_i\le L_i(T_1').A_i)=(T_2'-T_1')/(N-T_1')$，从而得到 P_i 的计算公式. 令 NUM_{remain} 表示候选元组集合 C 中无法被早剪切的元组数量，计算如下：

$$NUM_{\text{remain}}=\sum_{i=1}^{m}(NUM_i\times P_i).$$

利用 NUM_{tuple} 和 NUM_{remain}，早剪切操作在理论上可以剪切的元组个数 $NUM_{\text{pru}}=NUM_{\text{tuple}}-NUM_{\text{remain}}$，即

$$NUM_{\text{pru}}=\sum_{i=1}^{m}(NUM_i-NUM_i\times P_i)$$
$$=\sum_{i=1}^{m}[NUM_i\times(1-P_i)].$$

令 f_p 表示早剪切操作可以剪切的元组占增长阶段见到的所有元组的比例，

$$f_p=\frac{NUM_{\text{pru}}}{NUM_{\text{tuple}}}=\frac{\sum_{i=1}^{m}[NUM_i\times(1-P_i)]}{\sum_{i=1}^{m}NUM_i}.$$

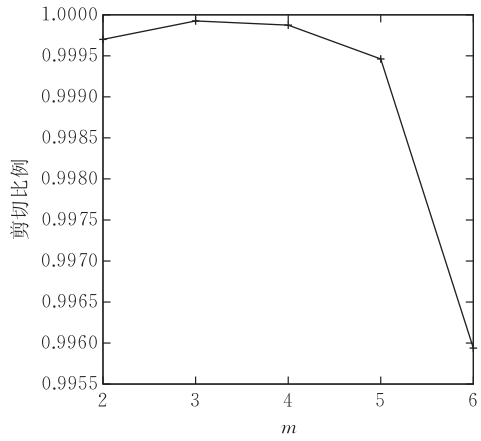
所以早剪切的理论效果取决于 P_i 的取值. 利用定理 1 和定理 2 的结果，

$$P_i=\left(\frac{T_2'-T_1'}{N-T_1'}\right)^{(m-i)}=\left[\frac{(m-1)\times T_1'}{N-T_1'}\right]^{(m-i)}$$
$$=\left[\frac{(m-1)\times[(-b+\sqrt{b^2-4ac})/(2a)]^{1/m}}{1-[(-b+\sqrt{b^2-4ac})/(2a)]^{1/m}}\right]^{(m-i)},$$

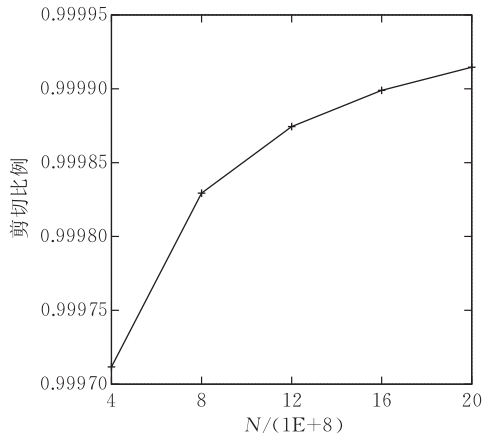
其中，a、b 和 c 的取值如定理 1 所示. 可以看出，P_i 的值较小，因为 $P_i\approx[(m-1)\times(2k/N)^{1/m}]^{(m-i)}$. 当 $N=12\times10^8$，$m=4$，$k=20$ 时，$P_i=0.043^{(4-i)}$. 给定 m，随着 i 的减小，P_i 指数级地减小，这也符合直观的感觉，即如果元组 t 在增长阶段出现的属性越少，则元组 t 越不可能是 top-k 的结果.

观察 NUM_i 的计算公式，我们发现在增长阶段，绝大多数的元组的属性只出现 1 次或者 2 次，即 NUM_1 和 NUM_2 的值比 $NUM_i(3\le i\le m)$ 要大很多，例如 $N=12\times10^8$，$k=20$，$m=4$ 时，$NUM_1/NUM_{\text{tuple}}=97.87\%$，$NUM_2/NUM_{\text{tuple}}=2.1\%$. 再加上 P_i 的值很小，尤其在 i 较小时，$1-P_i$ 的值就接近于 1，所以 f_p 的值接近于 1. 这就意味着早剪切算法在理论上可以剪切绝大多数的无用元组.

图 3(a)给出了 $N=12\times10^8$，$k=20$，m 变化时的剪切效果. 可以看到，剪切比例都在 99% 以上，虽然从 $m=3$ 开始，剪切比例开始降低，但是即使 $m=6$ 时，剪切比例仍维持在 99.59%. 这是由于随着 m 的增大，T_1' 随着 m 的增长而指数级增加，给定 $N=12\times10^8$ 不变，增长阶段有更多的元组找到了更多的属性，从而降低了剪切比例. 从 $m=2$ 到 $m=3$，剪切比例有所增加，这是因为 $m=2$ 时，早剪切操作只能剪切属性在增长阶段只出现过 1 次的元组，而 $m=3$ 时，早剪切操作还可以剪切属性在增长阶段出现 2 次的元组，所以剪切比例增加. 可是，当 $m\ge3$ 时，T_1' 的增加对剪切比例的影响超过了属性增加对剪切比例的影响，所以此时剪切比例开始下降.



(a) m变化的剪切效果



(b) N变化的剪切效果

图 3 早剪切的理论剪切效果

图 3(b)给出了 $m=4, k=20, N$ 变化时的剪切效果. 我们看到随着 N 的增大,剪切比例越来越大,这是由于更大的 N 使得属性值在列文件中的分布越分散,从而早剪切可以剪切的元组也越来越多.

通过分析,早剪切操作理论上可以剪切绝大多数的候选元组,从而大大减少了 top-k 处理需要的内存量,也加快了 top-k 的执行速度.

4.2 指数间距 bloom filter 表(EGBFT)

Bloom filter[8]是一种简洁的概率数据结构,用来判断给定元素是否是集合的成员. 用于集合 $S=\{x_1, x_2, \cdots, x_n\}$ 成员查询的 bloom filter 实现为一个 a 位的比特向量,初始时每个位都是 0. bloom filter 用 b 个值域为 $[1, \cdots, a]$ 的独立 Hash 函数 $h_1, h_2, \cdots, h_b$ 来构建和探测 bloom filter. $\forall x \in S$, bloom filter 的第 $h_i(x)(1 \leq i \leq b)$ 位设为 1. 要判断给定元素 y 是否属于集合 S, bloom filter 判断是否所有的第 $h_i(y)(1 \leq i \leq b)$ 位都是 1. 如果有任何一位为 0,则 y 肯定不是 S 的元素. 否则,bloom filter 认为 y 属于 S. 当然,即使 y 不属于 S,bloom filter 也可能返回 y 属于 S 的结论(false positive). 但是,bloom filter 对于属于 S 的元素肯定返回正确的结论. 令 fpr 表示 bloom filter 固有的 *false positive rate*,即不属于集合 S 的元素被 bloom filter 误认为是集合元素的概率. 给定 $b=(a\times\ln 2)/n$ 时,*false positive rate* 最小,$fpr=(1/2)^b$.

下面给出 TKEP 用到的指数间距 bloom filter 表的定义.

定义 2(指数间距 bloom filter 表). 给定包括 N 个元组的列文件 L_i, $EGBFT_i$ 是 L_i 上的指数间距 bloom filter 表,如果 $EGBFT_i$ 满足条件:(1) $|EGBFT_i|=\log_2 N$,(2) $EGBFT_i(j)$ 是构建在 $L_i(1)$ 到 $L_i(2^j)$ 的 RID 属性上的 bloom filter.

EGBFT 的构造如图 4 所示. 给定任意列文件 L_i, $EGBFT_i$ 占据的磁盘空间与元组数量 N 和 fpr 有关. $EGBFT_i(j)$ 是定义在 2^j 个元素上的 a 位并且具有 b 个 Hash 函数的 bloom filter,如果 $b=(a\times\ln 2)/2^j$, fpr(false positive rate)最小,$fpr=(1/2)^b$. 给定 fpr 和元组数量 2^j,我们得到最优的 bloom filter 长度

$$a=\frac{2^j\times\log_2\left(\frac{1}{fpr}\right)}{\ln 2}.$$

已知,$|EGBFT_i|=\log_2 N$,令 $SIZE_{EGBFT_i}$ 表示 $EGBFT_i$ 占据的磁盘空间,则

$$SIZE_{EGBFT_i}=\sum_{i=0}^{\log_2 N}\left[\frac{2^i\times\log_2\left(\frac{1}{fpr}\right)}{8\times\ln 2}\right]=\frac{(2N-1)\times\log_2\left(\frac{1}{fpr}\right)}{8\times\ln 2}.$$

根据 $SIZE_{EGBFT_i}$ 的计算公式,EGBFT 表占据的磁盘空间和元组数量呈线性关系. 由于本文中 TKEP 采用的 $fpr=0.01$, $EGBFT_i$ 占据的磁盘空间不超过列文件大小的 30%. TKEP 在每个列文件 $L_1, L_2, \cdots, L_M$ 上预先构建各自的 EGBFT 表.

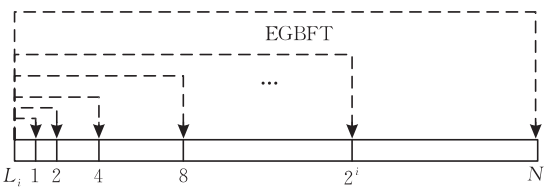


图 4 EGBFT 示例

4.3 算法实现

4.1 节给出了令人欣喜的早剪切的理论效果,可是在实际中,理论效果是很难实现的,因为它要求

快速返回任意位置索引范围的 RID 集合的成员检测结果. Bloom filter 是一种简洁的成员检测数据结构，考虑到早剪切涉及的位置索引范围的不固定，所以 TKEP 利用 EGBFT 表来执行早剪切操作.

利用定理 2 给出 T_2' 的值，TKEP 首先把 $EGBFT_i(j)(1\leq i\leq m)$ 读入内存，其中 $j=\lceil\log_2 T_2'\rceil$，然后利用内存中维护的 bloom filter 对增长阶段碰到的元组执行早剪切操作.

早剪切的代码如算法 1 所示. 算法 1 的实际效果不如早剪切的理论效果，其主要原因有两个：

(1) $EGBFT_i(j)$ 涉及的元素数量要大于 T_2'；

(2) bloom filter 固有的 false positive 问题.

算法 1.　EarlyPruning (RID *rid*).

```
//function testInBF(bf, rid) is used to determine
//whether rid is contained in S on which bloom filter
//bf is constructed, true is in, false is not
1. int index=⌈log2 T2'⌉
2. for i=1 to m
3.    boolean inflag=testInBF(EGBFTi(index),rid)
4.    if(!inflag)
5.       return true;
6.    enf if
7. end for
8. return false;
```

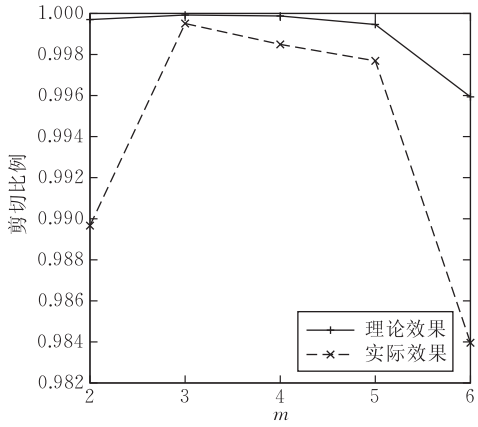
接下来，我们分析算法 1 的实际剪切效果. $\forall t\in C$，假设 t 的属性在增长阶段出现了 i 次，t 无法被算法 1 剪切的概率是

$$P_i'=\left(\frac{2^j-T_1'}{N-T_1'}+\frac{N-2^j}{N-T_1'}\times fpr\right)^{(m-i)}.$$

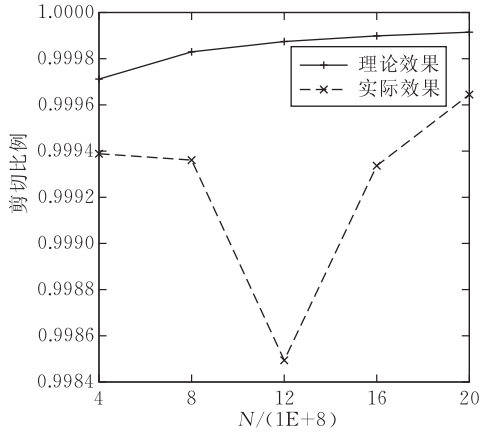
P_i' 解释如下：假设元组 t 有 i 个属性 $A_1,\cdots,A_i$ 出现在列文件 $L_1,\cdots,L_i$ 的前 T_1' 个元组，要使得 t 不会被算法 1 剪切，那么给定 $(i+1)\leq d\leq m$，存在如下两种情况：(1) $L_d(2^j).A_d\leq t.A_d\leq L_d(T_1').A_d$，(2) $t.A_d<L_d(2^j).A_d$，但是以概率 fpr，$EGBFT_d(j)$ 认为 $L_d(2^j).A_d\leq t.A_d\leq L_d(T_1').A_d$. 令 f_p' 表示算法 1 的实际剪切比例，计算公式如下：

$$f_p'=\frac{\sum_{i=1}^{m}[NUM_i\times(1-P_i')]}{\sum_{i=1}^{m}NUM_i}.$$

由于 $\forall i$，$P_i'>P_i$，所以 $f_p'<f_p$，但是算法 1 仍然可以较好地完成剪切工作. 如图 5(a)所示，即使 $m=6$ 时，算法 1 仍然可以获得 98.4%的剪切比例，而在图 5(b)中，算法 1 最低可以获得 99.85%的剪切比例.



(a) m变化的剪切效果



(b) N变化的剪切效果

图 5　早剪切的实际剪切效果

如图 5(b)所示，在 $N=12\times10^8$ 时，实际剪切比例要低于其它点的剪切比例，这是由于 $N=12\times10^8$ 时，$T_2'=67738932$ 刚好超过 $EGBFT(26)$，TKEP 载入 $EGBFT(27)$ 执行早剪切操作，$EGBFT(27)$ 是构建在第 1 和第 134217728 元组的 RID 属性上的 bloom filter，该范围几乎是 T_2' 的两倍，所以 $N=12\times10^8$ 时的剪切比例较低，而 N 取其它数值时并没有出现类似的最差情况.

令 $SIZE_{\text{added}}$ 表示需要读入内存的 EGBFT 元组大小，则

$$SIZE_{\text{added}}=m\times\frac{2^{\lceil\log_2 T_2'\rceil}\times\log_2\left(\frac{1}{fpr}\right)}{8\times\ln 2}.$$

令 $SIZE_{\text{pruned}}$ 表示早剪切操作剪切掉的元组大小，L_{tuple} 表示每个元组在 Hash 表中占据的空间，则

$$SIZE_{\text{pruned}}=L_{\text{tuple}}\times\sum_{i=1}^{m}[NUM_i\times(1-P_i')].$$

通过本文的实验可知，读入内存的 EGBFT 元组的空间比剪切掉的元组占用的空间要小 1 个数量级，而且，内存中的 EGBFT 元组不需要参与任何更新操

作,绝大多数候选元组的剪切不但节省了大量的内存空间,还减少了数据维护和更新的费用,这使得在内存中维护 EGBFT 的代价远小于早剪切带来的收益.

TKEP 的执行代码如算法 2 所示.算法 1 可以较好地执行单次剪切操作.可是,在增长阶段,如果同一元组的多个不同属性在不同列文件出现,算法 1 需要被调用多次,而其中只有一次剪切是必要的.一种解决方法是记录已经剪切的元组,如果已剪切的元组的其他属性再次出现,则不需要再执行剪切.可是,这样会增加内存的消耗,违背早剪切操作的初衷.TKEP 采用的方法是:即使增长阶段出现单个元组的多个属性,TKEP 也仍然执行多次剪切操作.这样做的原因是,在增长阶段,绝大多数的元组其属性只出现过 1 次或者 2 次,当 $N=12\times10^8$,$k=20$,$m=4$ 时,增长阶段属性数量只出现 1 次的元组占所有元组的 97.87%,属性数量值出现 2 次的占 2.1%,需要的剪切判断次数只比最优情况多 2.14%,而且,bloom filter 的检测操作可以在内存中快速执行,从而 TKEP 可以较小的代价获得较大的收益.

算法 2. $TKEP(L_1,\cdots,L_m,F)$

```
//P_Q: priority queue to maintain k highest lowerbound
       of tuple
//C: tuple set in hash table
1. boolean beGrowing=true;
2. hashtable ht;
3. loop from 4~29;
4. read next object x in round-robin way and update
   threshold
5. if(beGrowing)
6.    if(earlyPruning(x.rid))
7.       continue;
8.    else
9.       Hashtable_Value value=update_hashtable(ht,
                                   x.rid)
10.      if(value.partialScore≥P_Q.min)
11.         update (P_Q, value)
12.      end if
13.      if(P_Q.min≥threshold)
14.         beGrowing=false;
15.      end if
16.   end if
17. else
18.    Hashtable_value value=ht.get(x.rid)
19.    if(value==null)
20.       continue;
21.    else
22.       Hashtable_Value value=
                                   update_hashtable(ht, x.rid)
23.       if(value.partialScore≥P_Q.min)
24.          Update (P_Q, value)
25.       end if
26.       if(P_Q.min≥F_t^ub (t∈C−P_Q))
27.          return P_Q
28.       end if
29.    end if
30. end if
```

TKEP 算法对增长阶段见到的每个元组执行早剪切操作,从而减少了需要维护的候选元组数量.在处理过程中,TKEP 不断更新 *threshold* 和 P_Q,当 $P_Q.min\geq threshold$ 时,TKEP 离开增长阶段进入收缩阶段.由于此时 TKEP 维护的元组数量很少,所以收缩阶段可以较快地执行.在收缩阶段,TKEP 采用类似于 LARA[4-5] 的方式维护候选元组的上界,同时忽略不在候选元组集合中的元组,因为它们不可能是最终的 top-k 结果.当 $P_Q.min\geq F_t^{ub}(t\in C-P_Q)$ 时,TKEP 执行结束,P_Q 维护的 k 个元组就是 top-k 的最终结果.

5 性能评价和分析

5.1 实验设置

本节通过实验评价 TKEP 的性能,我们利用 Java 语言实现 TKEP 和 NRA 算法,jdk 版本为 jdk-6u17-windows-x64,实验在 HP xw8600 工作站(8×2.8GHz Xeon CPU+32GB 内存+1.4TB 硬盘+64bit windows)上执行.实验从 3 个方面来比较算法的性能:元组数量 N、返回结果数量 k 和涉及到的属性数量 m.实现的 NRA 算法结合了传统的 NRA[3]、LARA[4-5] 和 TBB[6] 算法.

根据 m 的数值,我们生成 m 个属性的表,每个属性的值在[0,1]上均匀分布,该表的每个属性单独存储为列文件,同时实例化每个属性所在元组的标识符,列文件根据属性值降序排列后存储.实验用到的评分函数 $F=\sum_{i=1}^{m}A_i$,参数设置如表 1 所示.

表 1 实验参数设置

参数	默认值	变化范围
N	12×10^8	$4\times10^8\sim20\times10^8$
k	20	5~25
m	4	2~6

元组数量 N 分别取值 4×10^8、8×10^8、12×10^8(默认值)、16×10^8 和 20×10^8,返回结果数量 k 分别取值 5、10、15、20(默认值)和 25,属性数量 m 分别

取值 2、3、4(默认值)、5 和 6.

在实验中，我们设置内存 Hash 表维护的最大元组数量是 4×10^7，此时系统占用的内存空间大约是 6G. 当需要维护的元组数量超过该数量时，我们采用类似于 LARA 的处理方法，把 Hash 表的元组输出到磁盘，并且和指定缓存文件(初始为空)合并，同时更新 P_Q，然后清空 Hash 表，继续执行接下来的操作. 该种方法可以处理维护的元组数量超过内存最大容量的情况，可是该方法延长了 top-k 的增长阶段的执行时间，因为在两次文件合并之间，当前 Hash 表维护的不是增长阶段已经看到的所有元组信息，而只是上次 Hash 表清空后看到的部分元组信息，只有当下一次文件合并操作时，我们才可以更新已经看到的所有元组信息.

5.2　实验 1：N 的效果

固定 $k=20$ 和 $m=4$，实验 1 评价 N 变化时 TKEP 算法的性能. 如图 6(a)所示，和 NRA 算法相比，TKEP 获得的平均加速比是 16.08. 在图 6(a)中，我们注意到当 $N=12$、16 和 20($\times10^8$)时，NRA 的执行时间增长的幅度不大，其原因在于此时需要维护的所有元组数量超过最大限制，只有当前 Hash 表的部分元组信息和磁盘上的缓存文件合并时，我们才可以获得已经看到的所有元组信息和更新 P_Q 的信息，这就把检查增长阶段是否结束的检测推迟到了下次文件合并操作，这可以在图 6(b)中得到证实，当 $N=12$、16 和 20($\times10^8$)时，NRA 在增长阶段维护的基本相等的元组数量. 由于采取早剪切操作，TKEP 可以剪切掉在增长阶段见到的绝大多数元组，TKEP 维护的元组数量比 NRA 维护的平均少 1448.85 倍，并且本文对早剪切的理论分析结果和实际运行结果相符.

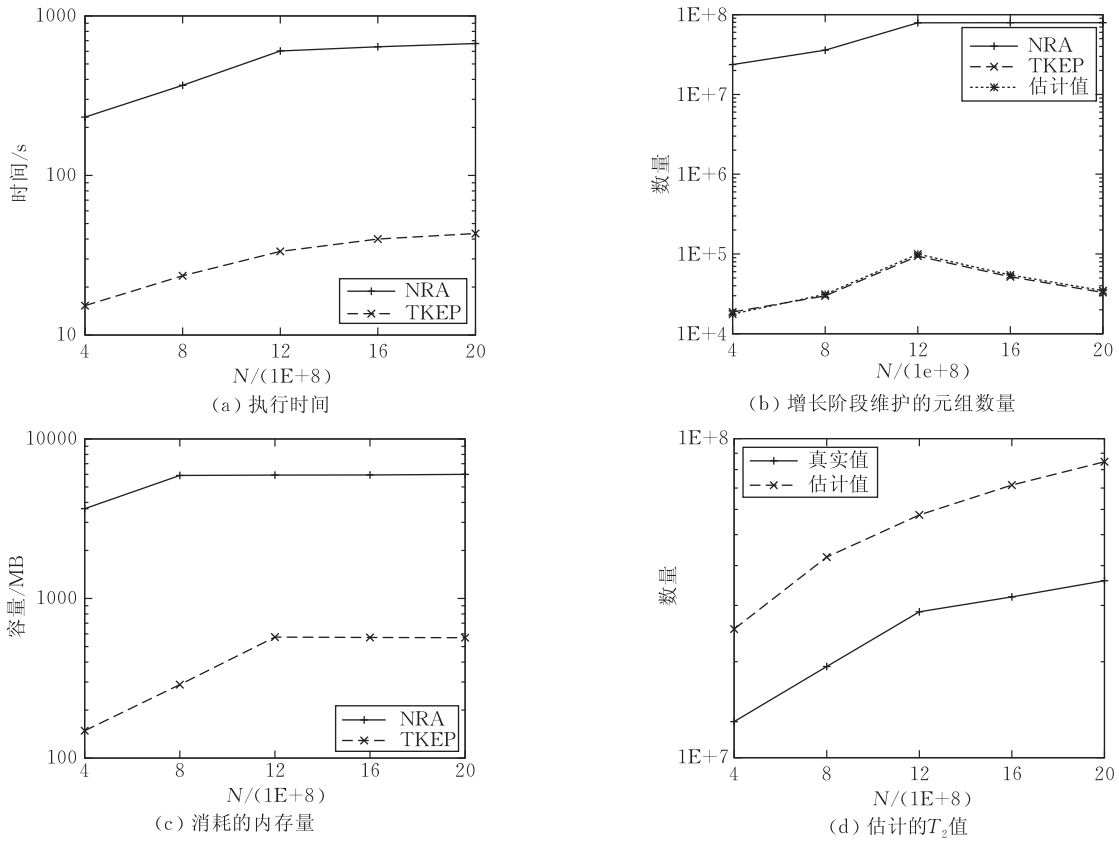


图 6　元组数量的影响

图 6(c)说明了 TKEP 和 NRA 的执行所需要的内存量，平均来看，NRA 需要的内存量是 TKEP 的 15.29 倍，其中 TKEP 所需要的大部分内存用来维护对应的 EGBFT 元组，因为绝大多数的候选元组都被剪切了. 从图 6(c)可以看到，在 N 大于 8×10^8 时，NRA 所需要的内存量基本不变，这是由于我们限制了内存的 Hash 表最多维护 4×10^7 个元组，而 $N=12$、16 和 20($\times10^8$)时，NRA 需要维护的元组数量都超过内存的最大限制，当 $N=8\times10^8$，NRA 需要维护的元组数量(35913291)虽然没有超过可是也很接近最大限制. 当 $N=12$、16 和 20($\times10^8$)时，TKEP 所需要的内存量也基本相等，这是由于在 TKEP 的执行过程中，绝大多数的内存是用来维护 EGBFT 元组的，当 $N=12$、16 和 20($\times10^8$)时，

TKEP 需要维护同样的 $EGBFT_i(27)(1\leq i\leq m)$ 元组，所以需要的内存量基本相等. 图 6(d)比较了估计的 T_2' 和实际的 T_2，我们发现 $T_2'>T_2$，从而验证了本文对 NRA 的行为分析.

5.3 实验 2：k 的效果

固定 $N=12\times10^8$ 和 $m=4$，实验 2 评价返回结果数量 k 变化时 TKEP 的性能.

如图 7(a)所示，TKEP 的执行时间平均比 NRA 的执行时间快 20.07 倍，这是由于 TKEP 的早剪切操作及时地剪切了绝大多数的候选元组，如图 7(b)，TKEP 在增长阶段需要维护的元组数量比 NRA 需要维护的平均少 3194.66 倍. 我们注意到，在 $k=10$、15、20 和 25 时，NRA 的执行时间基本不变，其原因和 5.2 节的讨论类似，此时 NRA 需要维护的元组数量超过内存的最大限制，内存溢出处理把增长阶段结束的结论检查推迟到下次文件合并操作，使得 NRA 在 $k=10$、15、20 和 25 时维护基本相等数量的元组. 更少的元组数量使得 TKEP 需要更少的内存，如图 7(c)所示，TKEP 需要的内存量比 NRA 少 16.57 倍，TKEP 需要的绝大多数内存是用来维护 EGBFT 元组，而在 $k=5$、10 和 15 时，TKEP 需要同样 $EGBFT_i(26)(1\leq i\leq m)$，所以其消耗的内存量基本不变，$k=20$ 和 25 时，TKEP 需要同样的 $EGBFT_i(27)$，所以其内存量也相等. 在 $k=10$、15、20 和 25 时，NRA 消耗的内存量达到内存的最大限制，所以内存量基本维持不变. 当 $k=5$ 时，NRA 需要维护的元组数(38181274)虽然没有超过但是也很接近内存最大限制，所以 NRA 在这组实验中对内存量的消耗基本不变. 图 7(d)比较了估计的 T_2' 和实际的 T_2，我们发现 $T_2'>T_2$，从而验证了本文对 NRA 的行为分析.

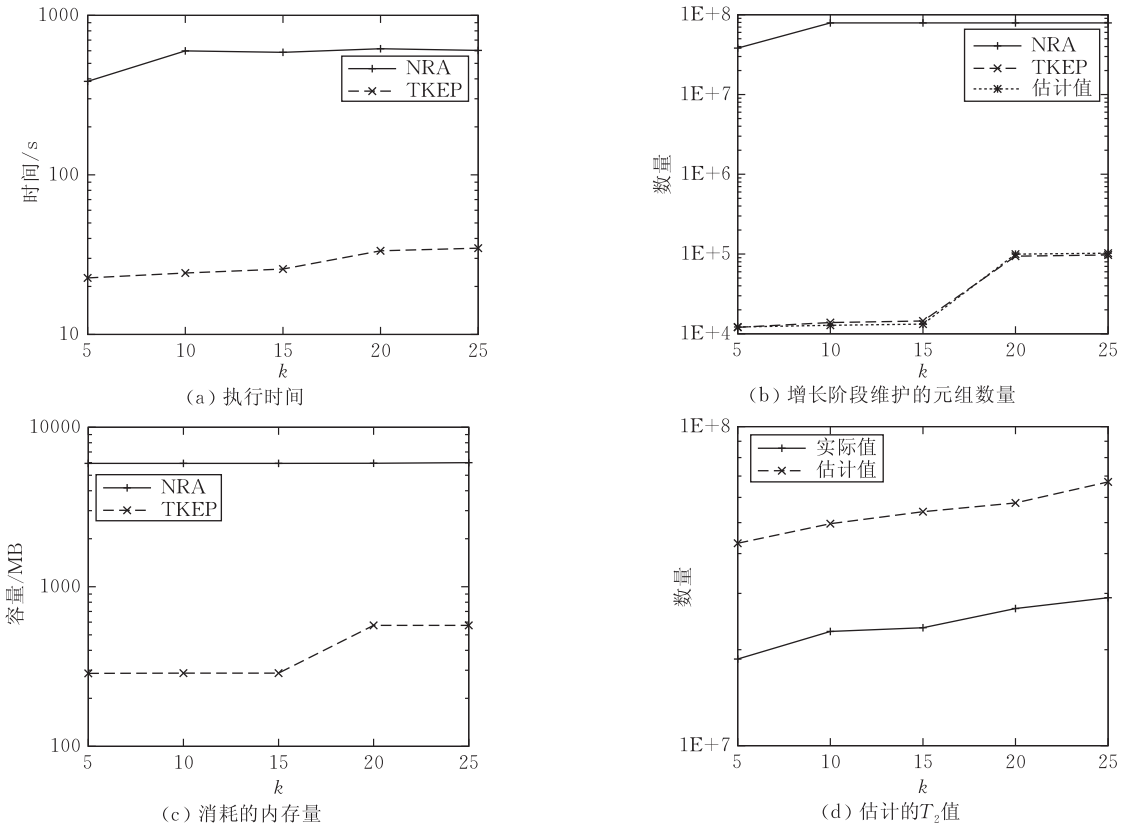


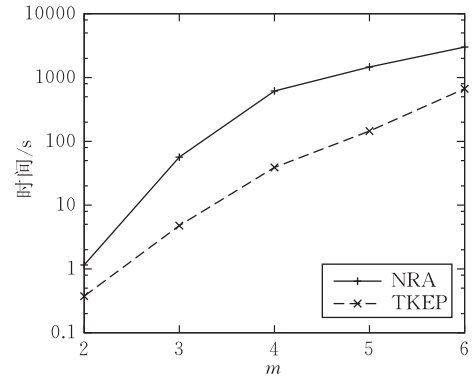
图 7 返回结果数量的影响

5.4 实验 3：m 的效果

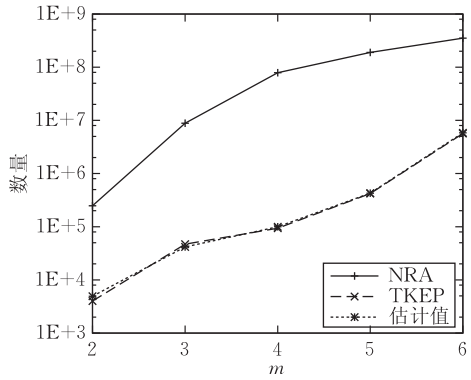
固定 $N=12\times10^8$ 和 $k=20$，实验 3 评价涉及到的属性数量 m 变化时 TKEP 的性能.

如图 8(a)所示，TKEP 的执行时间比 NRA 快了 9.07 倍，而且基本上从 $m=3$ 开始，TKEP 对 NRA 的加速比在逐渐减小. 这是由于随着 m 的增大，T_1 随着 m 的增长而指数级增加，给定 $N=12\times10^8$ 不变，增长阶段有更多的元组找到了更多的属性，从而降低了剪切比例，如图 8(b)，这使得 TKEP 需要维护更多的元组，但是 TKEP 维护的元组数量仍然比 NRA 少了 320.56 倍. 如图 8(c)，TKEP 需要的内存量比 NRA 少 9.56 倍. 我们发现，在 $m=5$ 和 6 时，NRA 需要的内存量超过了预定的最大限制，这是因为在 $m=5$ 时，NRA 需要清空 Hash 表 5

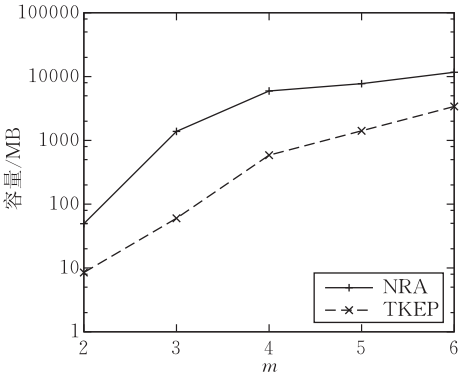
次，而 $m=6$ 时，NRA 需要清空 Hash 表 9 次，而 Java 采用的自动内存回收机制，这就使得在多次反复清空、插入 Hash 表操作时，有一部分内存没有被 Java 及时回收，从而造成 NRA 运行期间内存超过最大限制. 这从另一个方面也说明了早剪切操作的必要性. 图 8(d)给出了估计的 T_2'和实际的 T_2的比较，与之前的分析类似，我们发现 $T_2'>T_2$，从而验证了本文对 NRA 的行为分析.



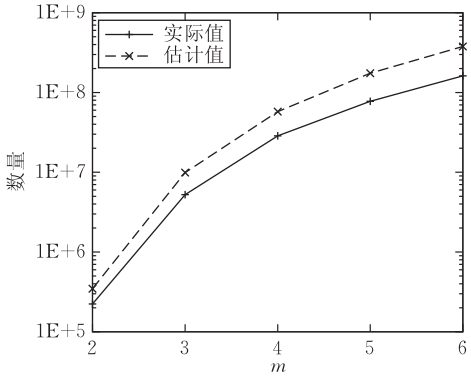
(a) 执行时间



(b) 增长阶段维护的元组数量



(c) 消耗的内存量



(d) 估计的T_2值

图 8　属性数量的影响

6　相关工作

作为多个领域中一种重要的数据操作，top-k 查询得到了研究人员的广泛关注[9]. Bruno 等[10]把 top-k 查询转换为数据库上的范围查询，从而可以利用现有数据库技术来直接回答 top-k 查询. Fagin 等[2]提出 Threshold Algorithm(TA)算法处理有序链表上的 top-k 查询. TA 算法要求系统同时支持顺序读和随机读操作. 相对于 TA 算法的 round-robin 读取方式，Guntzer 等[11]提出 Quick-Combine 算法优先读取使算法阈值下降最快的链表，从而加快了 TA 算法的执行. Bast 等[12]把 top-k 优化问题转化为顺序读和随机读的调度问题，从而获得最优的调度和性能. Akbarinia 等[13]提出 Best Position 算法来改进传统的 TA 算法，Best Position 算法利用随机读过程中获得的辅助信息来加快 top-k 处理.

TA 算法要求系统支持随机读操作，可是在有些情况中，例如数据流和海量数据环境，随机读操作是受限制的或者不可能的. Fagin 等[3]提出 NRA 算法(No Random Access)来处理只支持顺序读环境下的 top-k 查询. Guntzer 等[14]提出的 Stream-Combine 算法改进 NRA 算法，优先读取使得算法阈值下降最快的链表. Mamoulis 等[4-5]通过对 NRA 算法的分析，把 NRA 的执行过程分为增长和收缩两个阶段，并且提出 LARA 算法减少了 top-k 处理的计算费用和内存需求. Bruno 等[15]和 Marian 等[16]提出 Upper 和 Pick 算法处理 Web-accessible 数据库的 top-k 查询. Hwang 等[17]提出 Minimal Probing 算法来最小化 top-k 查询中的谓词评价费用. Pang 等[6]提出 TBB 算法把 NRA 的 round-robin 数据读取过程转换为链表的顺序扫描，从而提高了磁盘效率.

另一种处理 top-k 的方法是利用索引和视图. Chang 等[18]提出 Onion Indices 来处理 top-k 查询，该方法把元组看作多维空间的点，利用预计算的数据点的闭包来回答 top-k 查询. Xin 等[19]提出了

robust索引的概念来加快 top-k 处理，该方法把元组划分为多个连续的层次，任何 top-k 查询可以最多利用 k 层元组来回答. Zou[20]等通过构建支配图结构来改进 top-k 查询效率，预计算得到的支配图表示元组间的支配关系，top-k 查询转换为图的遍历问题. Hristidis 等[21]和 Das 等[22]提出利用实例化视图来加快 top-k 处理.

7 结 论

本文详细地分析了 NRA 算法的执行行为，确定了增长阶段和收缩阶段中每个文件需要扫描的元组个数. 为解决 NRA 在增长阶段维护元组过多的问题，本文提出一种新的海量数据上的 top-k 查询算法 TKEP，该算法在查询的增长阶段就执行早剪切，从而大大减少增长阶段需要维护的候选元组. 本文给出了早剪切操作的数学分析，证明了早剪切操作的理论和实际剪切效果. 实验表明，不管在执行时间还是内存消耗方面，TKEP 比 NRA 算法都具有较大的优势.

本文给出的 TKEP 算法假设涉及到的属性在其值域内均匀分布并且属性间独立分布，在后续工作中，我们将继续考虑把 TKEP 扩展到属性不均匀分布和属性相关的环境.

参 考 文 献

[1] Korn Flip, Pagel Bernd-Uwe, Faloutsos Christos. On the 'Dimensionality Curse' and the 'Self-Similarity Blessing'. IEEE Transactions on Knowledge and Data Engineering, 2001, 13(1): 96-111

[2] Fagin Ronald, Lotem Amnon, Naor Moni. Optimal aggregation algorithms for middleware//Proceedings of the 20th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems(PODS'01). California, USA, 2001: 102-113

[3] Fagin Ronald, Lotem Amnon, Naor Moni. Optimal aggregation algorithms for middleware. Journal of Computer and System Sciences, 2003, 66(4): 614-656

[4] Mamoulis Nikos, Cheng Kit Hung, Yiu Man Lung, Cheung David W. Efficient aggregation of ranked inputs//Proceedings of the 22nd International Conference on Data Engineering(ICDE'06). Atlanta, GA, USA, 2006: 72-83

[5] Mamoulis Nikos, Yiu Man Lung, Cheng Kit Hung, Cheung David W. Efficient top-k aggregation of ranked inputs. ACM Transactions on Database Systems(TODS), 2007, 32(3): 19

[6] Pang HweeHwa, Ding Xuhua, Zheng Baihua. Efficient processing of exact top-k queries over disk-resident sorted lists. VLDB Journal, 2010, 19(3): 437-456

[7] Fagin Ronald, Kumar Ravi, Sivakumar D. Efficient similarity search and classification via rank aggregation//Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data (SIGMOD'03). San Diego, California, USA, 2003: 301-312

[8] Bloom Burton H. Space/time trade-offs in Hash coding with allowable errors. Communications of the ACM, 1970, 13(7): 422-426

[9] Ilyas Ihab F, Beskales George, Soliman Mohamed A. A survey of top-k query processing techniques in relational database systems. ACM Computing Surveys, 2008, 40(4): 11

[10] Bruno Nicolas, Chaudhuri Surajit, Gravano Luis. Top-k selection queries over relational databases: Mapping strategies and performance evaluation. ACM Transactions on Database Systems(TODS), 2002, 27(2): 153-187

[11] Guntzer Ulrich, Balke Wolf-Tilo, Kiessling Werner. Optimizing multi-feature queries for image databases//Proceedings of the 26th International Conference on Very Large Data Bases(VLDB'00). Cairo, Egypt, 2000: 419-428

[12] Bast Holger, Majumdar Debapriyo, Schenkel Ralf, Theobald Martin, Weikum Gerhard. IO-Top-k: Index-access optimized Top-k query processing//Proceedings of the 32nd International Conference on Very Large Data Bases(VLDB'06). Seoul, Korea, 2006: 475-486

[13] Akbarinia Reza, Pacitti Esther, Valduriez Patrick. Best position algorithms for Top-k queries//Proceedings of the 33rd International Conference on Very Large Data Bases(VLDB'07). Vienna, Austria, 2007: 495-506

[14] Guntzer Ulrich, Balke Wolf-Tilo, Kiessling Werner. Towards efficient multi-feature queries in heterogeneous environments//Proceedings of the 2001 International Symposium on Information Technology: Coding and Computing(ITCC'01). Las Vegas, NV, USA, 2001: 622-628

[15] Bruno Nicolas, Gravano Luis, Marian Amelie. Evaluating Top-k queries over web-accessible databases//Proceedings of the 18th International Conference on Data Engineering (ICDE'02). San Jose, CA, 2002: 369-380

[16] Marian Amelie, Bruno Nicolas, Gravano Luis. Evaluating Top-k queries over web-accessible databases. ACM Transactions on Database Systems(TODS), 2004, 29(2): 319-362

[17] Hwang Seung-Won, Chang Kevin Chen-Chuan. Probe minimization by schedule optimization: Supporting Top-K queries with expensive predicates. IEEE Transactions on Knowledge and Data Engineering, 2007, 19(5): 646-662

[18] Chang Yuan-Chi, Bergman Lawrence D, Castelli Vittorio et al. The onion technique: Indexing for linear optimization queries//Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data(SIGMOD'00). Dallas, Texas, USA, 2000: 391-402

[19] Xin Dong, Chen Chen, Han Jiawei. Towards robust indexing for ranked queries//Proceedings of the 32nd International Conference on Very Large Data Bases(VLDB'06). Seoul, Korea, 2006: 235-246

[20] Zou Lei, Chen Lei, Graph Dominant. An efficient indexing structure to answer Top-K queries//Proceedings of the 24th International Conference on Data Engineering (ICDE'08). Cancun, Mexico, 2008: 536-545

[21] Hristidis Vagelis, Papakonstantinou Yannis. Algorithms and applications for answering ranked queries using ranked views. VLDB Journal, 2004, 13(1): 49-70

[22] Das Gautam, Gunopulos Dimitrios, Koudas Nick. Dimitris tsirogiannis, answering Top-k queries using views//Proceedings of the 32nd International Conference on Very Large Data Bases (VLDB'06). Seoul, Korea, 2006: 451-462

HAN Xi-Xian, born in 1981, Ph. D. candidate. His research interests include massive data processing and data intensive computing.

YANG Dong-Hua, born in 1976, Ph. D., lecturer. His research interests include massive data processing and data intensive computing.

LI Jian-Zhong, born in 1950, professor, Ph. D. supervisor. His research interests include database and sensor network.

Background

This paper focuses on the research for query processing issues on massive data. Top-k is an important operation in multiple fields since it returns k most important objects among potential huge answer space according to a given ranking function. Threshold algorithm (TA) is proposed to solve it by sequential and random access on sorted files. But on massive data, random access is prohibitively expensive. Under these circumstances, NRA is used to perform top-k query processing by sequential access only. NRA does not perform any pruning in growing phase. So it maintains a large quantity of candidate tuples in execution which affects its performance significantly. TKEP is the first to perform early pruning in top-k to execute pruning in growing phase. This paper first analyzes the execution behavior of NRA and determines positional index values to which each sorted file is scanned. By general rule for early pruning proposed in this paper, most of candidate tuples can be pruned. Mathematical analysis for early pruning is presented in this paper also. The theoretical and practical pruning effect is presented in this paper. We conduct extensive experiments to evaluate the performance of TKEP. Experimental results show that TKEP maintains up to three orders of magnitude fewer tuples in growing phase than NRA and obtains an order of magnitude speed-up comparing to NRA.

This work is supported in part by the National Grand Fundamental Research 973 Program of China, the Key Program of National Natural Science Foundation of China, the NSF of China and the NSFC-RGC of China. These foundations focus on the research of various areas of data intensive super computing. Our group has been working on the research of database for many years, and many good papers have been published in worldwide conferences and transactions, such as SIGMOD, VLDB, ICDE, KDD, INFOCOM, TKDE, VLDB Journal et al. This paper proposes a novel top-k query processing on massive. It gives a solution on topic of query processing on massive data.