ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.21, No.4, April 2010, pp.620−631
doi: 10.3724/SP.J.1001.2010.03498
© by *Institute of Software*, *the Chinese Academy of Sciences*. All rights reserved.
E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

缓存敏感的封闭冰山立方体计算*

栾 华[1,2+], 杜小勇[1,2], 王 珊[1,2]

[1](数据工程与知识工程教育部重点实验室(中国人民大学),北京 100872)

[2](中国人民大学 信息学院,北京 100872)

Cache-Conscious Computation of Closed Iceberg Cubes

LUAN Hua[1,2+], DU Xiao-Yong[1,2], WANG Shan[1,2]

[1](Key Laboratory of Data Engineering and Knowledge Engineering (Renmin University of China), Ministry of Education, Beijing 100872, China)

[2](School of Information, Renmin University of China, Beijing 100872, China)

+ Corresponding author: E-mail: luanhua@ruc.edu.cn

Luan H, Du XY, Wang S. Cache-Conscious computation of closed iceberg cubes. *Journal of Software*, 2010,21(4):620−631. http://www.jos.org.cn/1000-9825/3498.htm

Abstract: The computation of data cubes usually produces huge outputs. There are two popular methods to solve this problem: Iceberg cube and closed cube, which can be combined together. Due to the importance and usability of closed iceberg cube, how to efficiently compute it becomes a key research issue. A cache-conscious computation method is proposed in this paper. The data are aggregated in a bottom-up manner. In the meantime, the closed cells covering the aggregate cells are discovered and output. Two pruning strategies are used to save unnecessary recursive calls. The Apriori pruning is utilized to support iceberg cube computation. To reduce the number of memory-related stalls and produce the aggregate results efficiently, multiple dimensions are pre-sorted and the software prefetching technology is introduced into data scans. A comprehensive and detailed performance study is conducted on both synthetic data and real data sets. The results show that the proposed closed iceberg cube computation method is efficient and effective.
Key words: OLAP (on-line analytical processing); closed iceberg cube; cache-conscious; memory-related stalls

摘 要: 数据立方体计算通常会产生大量的输出结果,冰山立方体和封闭立方体是解决这个问题的比较流行的两种策略,二者可以结合使用.鉴于封闭冰山立方体(closed iceberg cube)的重要性和实用性,如何高效地计算封闭冰山立方体是一个值得研究的问题.提出一种缓存敏感(cache-conscious)的计算封闭冰山立方体的方法,在自底向上对数据进行聚集的同时,寻找覆盖聚集单元的封闭单元,将其输出,使用两种策略进行剪枝,去掉不必要的递归,同时使用Apriori剪枝技术,支持冰山立方体(iceberg cube)的计算.为了减少与内存相关的延迟,快速得到聚集结果,对多个维进行预排序,并将软件预取技术引入到数据扫描中.在模拟数据和真实数据上进行了详细而全面的实验研究,结果表明,封闭冰山立方体的计算方法是快速、有效的.

---

* Supported by the National Natural Science Foundation of China under Grant Nos.60496325, 60873017 (国家自然科学基金); the Grant from HP Labs China (惠普中国实验室资助项目)

Received 2008-05-14; Accepted 2008-10-09

© 中国科学院软件研究所 http://www.c-s-a.org.cn

关键词: 联机分析处理;封闭冰山立方体;缓存敏感;内存相关延迟
中图法分类号: TP311 **文献标识码:** A

在数据仓库和 OLAP(on-line analytical processing)领域,数据立方体计算[1]是一项重要而又耗时的操作,引起了研究者的很大关注.对数据立方体的研究包含很多方面:计算完全立方体(full cube)和冰山立方体(iceberg cube)[2−5],这是一个基本的问题,它的发展可能会影响立方体其他方面的发展;当基本表发生改变时(如增加或删除数据记录、维的层次有了变化),数据立方体的维护[6,7];数据立方体的近似计算[8,9];对压缩立方体的计算、存储、查询以及维护[10−14];在数据立方体上进行高级分析[15,16]等.

众所周知,数据立方体的计算结果是非常巨大的,随着立方体维数的增加,这一结果呈指数级增长.目前,在已有的研究中,两种比较有效的解决这一问题的途经[14]是计算冰山立方体和封闭立方体(closed cube).在冰山立方体中,只需计算和输出那些满足冰山约束条件(如 $Count(*)\geq 10$)的单元,尽可能避免不必要的聚集操作;封闭立方体将聚集单元分成封闭单元和非封闭单元,只考虑封闭单元的计算和输出,其他单元的度量值可以由相应的封闭单元得到.这两种技术可以结合起来使用,即封闭冰山立方体(closed iceberg cube),计算满足冰山约束条件的封闭单元,进一步减少输出结果.鉴于封闭冰山立方体的重要性和实用性,如何快速而有效地计算封闭冰山立方体是一个具有较高研究价值的问题.

在数据立方体技术发展的同时,计算机硬件技术也在快速发展.首先,内存容量越来越大,以指数级的速度在增长着.现在配备 GB 级内存的计算机普遍存在,有的机器甚至具有 TB 级的内存容量(如 HP Integrity 有 2TB 的内存),更多的数据可以保存在内存中.其次,CPU 和内存的速度发展不匹配.CPU 以摩尔定律的速度在发展,远远快于内存的速度(大约每年 10%的发展趋势)[17],这使得 CPU 访问内存的相对代价越来越高,与内存相关的延迟对性能产生越来越大的影响.另外,现代的处理器支持很多新的技术,如 SMT 和多核.这些硬件方面的巨大变革促使研究者重新设计数据库的一些核心算法[18−21].同样地,在数据仓库和 OLAP 领域,如果能够有效地利用硬件资源,性能也会获得很大的提高.

本文提出一种缓存敏感(cache-conscious)** 的计算封闭冰山立方体的方法,这种方法从整体上看采用自底向上的聚集顺序,在聚集单元的同时,计算单元相应的封闭单元,利用封闭单元以及已经处理过的单元进行剪枝,减少不必要的递归调用,同时使用 Apriori 剪枝策略,支持冰山立方体的计算.为了减少内存相关延迟,使用预排序和软件预取技术,降低扫描数据的次数,隐藏访问内存导致的延迟,提高封闭冰山立方体计算的性能.在模拟数据和真实数据上所进行的详细的实验,其结果表明了这种方法的高效性.

本文第 1 节主要给出所使用的基本概念.第 2 节介绍给出的计算封闭冰山立方体的基本方法.第 3 节给出改进的缓存敏感计算方法.第 4 节是详细的实验研究.第 5 节介绍与本文相关的工作.第 6 节总结全文并提出未来工作方向.

1 基本概念

在本节中,我们首先介绍数据立方体中的基本概念(基于文献[14]),然后,为了更好地理解算法的优化,简要介绍现代计算机的层次存储结构.

1.1 数据立方体

例 1:图 1 显示的是一个具有 4 个维(A,B,C,D)的基本表(base table).假设度量函数是 $Count$.在这个基本表上得到的数据立方体(data cube)是 16 个分组查询的结果,$\{A,B,C,D\}$的每个子集形成一个分组.每个分组对应一个单元(cell)集合.

** 我们使用对 cache-conscious 习惯的翻译方法:缓存敏感,cache-conscious 通常是指采用的方法能够更好地利用 CPU 硬件资源,提高资源利用率,减少缺失,从而降低运行时间,提高性能.

© 中国科学院软件研究所 http://www.c-s-a.org.cn

定义 1(单元(cell)). 在一个 n 维的数据立方体中,单元 $c=(a_1,a_2,\ldots,a_n:m)$(m 是度量值)叫做一个 k 维的分组单元,当且仅当在$\{a_1,a_2,\ldots,a_n\}$中有 $k(k\le n)$个值不是“*”.这里,“*”的意思是这个维被概化(generalized)了,可以匹配域中的任意值.

定义 2(冰山单元(iceberg cell)). 如果在度量值上存在冰山约束条件,当一个单元满足这个条件时,则被称为冰山单元.

为了表述方便,设 $M(c)=m,V(c)=(a_1,a_2,\ldots,a_n)$.对于冰山约束,常见的情况是满足 $M(c)\ge$min_sup,其中,min_sup是用户定义的最小支持度.

定义 3(覆盖(cover)). 假设有两个单元 $c=(a_1,a_2,\ldots,a_n:m)$和 $c'=(a_1',a_2',\ldots,a_n':m')$,如果对每个不是“*”的 a_i ($i=1,\ldots,n$)都有 $a_i'=a_i$,那么称 $V(c)\le V(c')$.对于 c 和 c',如果所有满足条件 $V(c)\le V(c'')\le V(c')$的单元 c''都有 $M(c'')=M(c')$,那么单元 c 被 c'覆盖.

定义 4(封闭单元(closed cell)). 如果一个单元不被任何单元覆盖,那么它是一个封闭单元.

定义 5(封闭冰山单元(closed iceberg cell)). 如果一个封闭单元满足冰山约束条件,那么它是一个封闭冰山单元.

封闭冰山立方体(closed iceberg cube)的计算就是要计算所有的封闭冰山单元;如果不存在冰山约束条件,则计算封闭完全立方体(closed full cube).

例 2:在如图 1 所示的基本表中,假设冰山约束是 $Count\ge 2$,那么 $c_1=(a_1,b_1,*,d_1:2)$和 $c_2=(a_1,*,*,*:3)$是封闭冰山单元,单元 $c_3=(a_1,b_1,*,*:2)$和 $c_4=(a_1,b_2,c_2,d_2:1)$不是封闭冰山单元,因为 c_3 被单元 c_1 覆盖,而 c_4 不满足冰山约束.

A	B	C	D
a_1	b_1	c_1	d_1
a_1	b_1	c_2	d_1
a_1	b_2	c_2	d_2

Fig.1 An example of base table

图 1 一个基本表例子

1.2 层次存储结构

在现代计算机体系中,CPU 缓存-内存-磁盘形成了一个层次的存储结构,如图 2 所示.

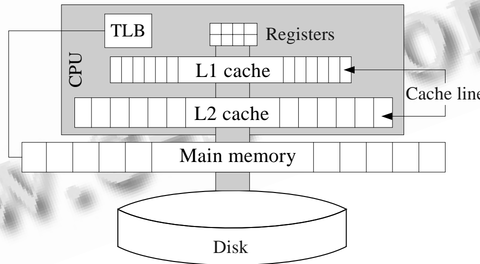

Fig.2 Hierarchical memory system

图 2 层次存储结构

CPU 缓存(CPU cache)[17]是容量较小、速度较快的 RAM 存储器,缓存内存中的数据和指令.现代的处理器通常有两级缓存:L1 和 L2,有的处理器还有 L3 缓存,如 Itanium 2 处理器.越靠近 CPU 的缓存,容量越小,但处理速度越快.缓存由多个缓存行(cache line)组成,缓存行是缓存与内存之间的传输单元.

当 CPU 需要的数据在 L1 缓存中时,程序很快继续运行,否则引起缓存缺失(cache miss),数据需要从较低层的缓存(L2)中取出.如果 L2 中也没有数据,则再从内存中读取.L1 基本上以 CPU 的速度运行,访问 L2 会引起几十个时钟周期的延迟,但若访问内存,则要有几百个时钟周期的延迟,而且由于 CPU 和内存速度发展得不一致,这个延迟的相对代价会越来越大.因此应该尽量减少缓存缺失,尤其是 L2 的缺失.

除了 L2 缓存之外,TLB(translation lookaside buffer)也要与内存进行交互.TLB 是一种页表缓存,由多个条目

© 中国科学院软件研究所 http://www.c-s-a.org.cn

(entry)组成,用来存放数据和指令的虚拟地址到物理地址的转换信息.当 CPU 执行机构接到虚拟地址后,先到TLB 中查找相应的页表条目,如果 TLB 中正好有此条目,那么命中;否则,引起 TLB 缺失,需要访问内存.有时 TLB缺失也会对性能产生很大影响.

文献[22]将 L1,L2 和 TLB 引起的缺失统称为内存相关延迟(memory-related stalls),并通过研究发现,对于数据库来说,花费在这一部分上的时间大约占整个运行时间的 50%.所以,如果能够充分利用 CPU 资源,数据库可以运行得更快.

2 基本的计算方法

为了更加清楚地阐述封闭冰山立方体的计算过程,本节首先给出一种基本方法(简称 BASE_ALG),在下一节中,我们将介绍进一步优化的方法.

从整体上看,BASE_ALG 采用自底向上的计算顺序(类似于 BUC 算法[3]),先对第 1 个维进行划分,然后对第1 个维的每个划分在剩下的维上进行递归计算;处理完第 1 个维后,将这一维的值设为“*”,再划分第 2 个维,同样,在得到的每个划分上计算剩下的维;然后是第 3 个维,直到最后一个维,处理完一个维后将其值设为“*”.在得到一个划分之后,首先判断划分对应的单元是否满足冰山约束条件,如果是那么继续进行,否则,停止在此划分上聚集及递归;然后判断这个划分的封闭单元是否已经被输出了,如果是,则停止聚集及递归,否则,计算相应的封闭单元并输出;在执行过程中,如果某个维在封闭单元中的值不是“*”,那么跳过这个维.下面我们给出BASE_ALG 算法的具体过程(像以前的研究[3−5,14]一样,我们以 count 为例).

算法 1. *BASE_ALG*(*P*,*d*,*size*).

输入:要计算的划分 *P*,开始维 *d* 和划分的大小 *size*;

输出:封闭冰山立方体.

D 表示维数,C_i 表示第 *i* 个维的势,*result* 是一个单元,*CA*[*i*][*j*]是在划分 *P* 中第 *i* 维的第 *j* 个划分的元组数目.

```
Begin
01     if (size==1) then
02        if 每个小于 d 的维 i 都满足 result[i]!=“*” then
03           输出封闭冰山单元 P;          /*P 只有一条元组,即封闭冰山单元*/
04        return;
05     if 存在小于 d 的维 i,result[i]==“*”,且在划分 P 中这一维只有一个不同值 then
06        return;                        /*直接返回即可,因为对应的封闭冰山单元已经输出*/
07     for 每个大于等于 d 的维 i do
08        if 在划分 P 中这一维只有一个不同值 V_i then
09           result[i]=V_i;
10     输出封闭冰山单元 result;
11     记住 result 的值;
12     for 每个维 i,d≤i≤D do
13        if result[i]!=“*” then         /*跳过这一个维*/
14           continue;
15        Partition(P,i);                /*在划分 P 中对第 i 维再进行划分*/
16        for 每个划分 j,1≤j≤C_i do
17           if CA[i][j]≥min_sup then    /*Apriori 剪枝*/
18              P_ij 代表 P 中基于维 i 的第 j 个划分;
19              result[i]=P_ij[1][i];
20              BASE_ALG(P_ij,i+1,CA[i][j]);
```

21 恢复 *result* 的值为第 11 行记住的值;

End

在调用 BASE_ALG 函数之前,首先判断 *size*≥min_sup 是否成立,如果成立,则开始第 1 次调用 BASE_ALG 函数.这时,*result* 中所有维的值都是“*”,*P* 是整个基本表,*d* 是 1.第 1 行到第 4 行代码是对只有单条元组划分的优化,如果当前维之前的所有维在 *result* 中的值都不是“*”,那么这个元组作为封闭冰山单元输出,否则,说明这个划分的封闭冰山单元已经输出了,不需要再次输出.例如,假设当前单条元组是(a_1,b_1,c_1),*d* 是 3,*result* 是$(*,b_1,*)$,即 *result*[1]=“*”,那么不需要再计算$(*,b_1,*)$的封闭单元(a_1,b_1,c_1).这是由于 BASE_ALG 算法是按照维 *A*,*B*,*C* 的顺序执行的,在对维 *A* 的 a_1 分片进行处理时,(a_1,b_1,c_1)已经作为封闭冰山单元输出了.第 5 行和第 6 行使用相同的技术进行剪枝.第 7 行~第 9 行计算划分 *P* 的封闭冰山单元,假设 *P* 中有 3 条元组(a_1,b_1,c_1),(a_2,b_1,c_1),(a_2,b_1,c_2),*d* 是 1,这时,*result* 是(*,*,*).因为第 2 维只取一个不同的值 b_1,因此 *result* 成为$(*,b_1,*)$,根据定义,这是一个封闭冰山单元,在第 10 行中输出.第 11 行是为了防止 *result* 的值在递归调用中被改变,导致第 13 行不能进行正确的判断,暂时记下递归之前的值,在第 21 行进行恢复.第 13 行、第 14 行对立方体计算再次进行剪枝,如果某个维在 *result* 中的值不是“*”,如上面例子中 *result*[2]=b_1,则说明没有必要在这个维上再进行递归,因为它形成的封闭冰山单元$(*,b_1,*)$已经输出了.第 15 行基于维 *i* 进一步划分 *P*,第 18 行中的 P_{ij} 通过这一步得到,对维 *i* 的划分 *j*,如果满足冰山约束条件(第 17 行),则在设置好 *result* 的第 *i* 维的值后(第 19 行),对新的划分进行递归调用(第 20 行).

3 缓存敏感的计算方法

BASE_ALG 方法使用 CountingSort 算法对维进行排序(类似 BUC 算法),CountingSort 算法主要由 3 个步骤组成:计数阶段、累加阶段和移动阶段.在 BASE_ALG 中,这 3 个步骤分两次完成,算法 1 中的第 7 行、第 8 行完成计数功能(或者说第 7 行、第 8 行的功能通过计数实现),第 15 行的 Partition 函数进行累加和移动,实现排序划分功能.

3.1 预排序

我们来看 Partition 函数的执行过程.每次对划分 *P* 中的一个维进行排序,排序顺序和聚集顺序是一样的.假设有 4 个维 *A*,*B*,*C*,*D*,当对维 *A* 进行划分时,其他属性也会被从内存读入到缓存中,因为它们在同一个缓存行中.但无论一个缓存行中包含多少个属性,只有属性 *A* 是有用的,因此缓存行的利用率很低.在 *A* 及其后代都被聚集完成后,处理维 *B*(如果 *B* 需要处理的话).同样地,为了对 *B* 进行划分,不仅包含 *B* 还包含其他属性的数据又被扫描 1 次,如果数据不在缓存中,这会引起额外的缓存缺失(以及 TLB 缺失).这种情况在处理维 *C*,*D* 时都会发生.大体上讲,为了对剩余维进行排序,父分组的每个划分需要被读入缓存多次,这会导致很多的内存相关延迟.

我们对 BASE_ALG 进一步优化,得到缓存敏感的改进算法(简称为 CC_ALG),主要采用两种优化技术,第 1 个是预排序,基本原理如下:假设当前的划分是维 *A* 的 a_1 划分,要在这个划分上依次对维 *B*,*C*,*D* 进行排序,那么在对维 *B* 进行排序时,对维 *C* 和 *D* 也同时进行排序,即把 *C*,*D* 两个维预先排好,当 *C*,*D* 需要处理时,直接使用排序结果即可.这样只需将数据读入缓存一次,就完成所有维的划分,减少了数据读入的次数,因此也就降低了缓存缺失(以及 TLB 缺失).具体过程见算法 2.

算法 2. *CC_ALG*(*P*,*d*,*size*).

输入:要计算的划分 *P*,开始维 *d* 和划分的大小 *size*;

输出:封闭冰山立方体.

D 表示维数,C_i 表示第 *i* 个维的势,*result* 是一个单元,*CA*[*i*][*j*]是在划分 *P* 中第 *i* 维的第 *j* 个划分的元组数目.

Begin

01~10 行与算法 1 中的 01~10 行类似;

11 *PrePartition*(*P*,*d*,*CA*,*size*); /*在划分 *P* 中对从 *d* 到 *D* 的每个维进行划分*/

12 记住 *result* 的值;

13 **for** 每个维 *i*,*d*≤*i*≤*D* **do**

© 中国科学院软件研究所 http://www.c-s-a.org.cn

```
14          if result[i]!="*" then               /*跳过这一个维*/
15              continue;
16          for  每个划分 j,1≤j≤Ci do
17              if CA[i][j]≥min_sup then         /*Apriori 剪枝*/
18                  Pij 代表 P 中基于维 i 的第 j 个划分;
19                  result[i]=Pij[1][i];
20                  CC_ALG(Pij,i+1,CA[i][j]);
21                  恢复 result 的值为第 12 行记住的值;
End
```

算法 2 和算法 1 大部分步骤是相同的,主要有 3 处不同:前两处是第 5 行和第 7 行,完成的功能是一样的,只是算法 2 使用软件预取指令来实现,具体细节将在下一节中给出陈述;重点观察原来第 15 行的 Partition 函数,它在算法 2 中被出现在第 11 行的 PrePartition 函数代替,这个函数完成的功能是对从 d 到 D 的每个维进行划分,具体实现过程将在下一节中结合预取技术一起说明,预取技术是 CC_ALG 使用的第 2 个优化技术.

3.2 预取技术

图 3 是预取排序的一个简单示例.要在维 D_1 的一个划分(属性值是 d_{11})上对维 D_2 排序(假设 $d_{21}<d_{22}<d_{23}<d_{24}$,实际上,维 D_3 到 D_n 都会同时排序,这里只用 D_2 的排序来说明预取).结果是一个数组 *SP*,保存指向元组的指针,这些指针指向的元组在属性 D_2 上是有序的.指针数组 *P* 和关系中相应的属性被扫描了两遍,一遍是为了得到数组 *CA* 中的计数值,另一遍是为了得到有序指针数组 *SP*.如果属性值在缓存中没有被找到,则会引起缓存缺失(有时 TLB 缺失也会发生),那么执行时间会浪费在这些缺失上.为了减少由于缺失带来的延迟,CC_ALG 将软件预取技术应用到对待排数据的扫描上,预取通过时间重叠降低了显示出来的缺失延迟.在属性值 $P[i][2]$使用之前,属性值 $P[i+k][2]$的读取指令发布出去(k 是预取距离),$P[i+k][2]$的读取和 $P[i][2]$的使用以及其他读取操作同时进行.在算法需要访问属性值 $P[i+k][2]$时,它已在缓存中.指针数组 *P* 保存了元组的地址,使得预取易于实现.

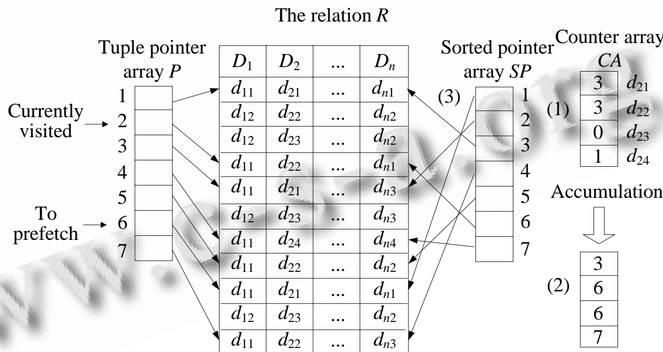

Fig.3 Illustration of prefetching sorting

图 3 预取排序的示意图

图 3 中的(1)~(3)显示了 CountingSort 的 3 个步骤,其中,步骤(1)和步骤(3)阶段都需要扫描原数据,可以在扫描数据的过程中加入软件预取指令.除此之外,上一节已经提到,在算法 2 的第 5 行判断是否剪枝时也要读取原数据,这同样是一个可使用预取提高性能的地方.下面在算法 3 中给出使用预取技术的 PrePartition 函数,算法 2 中第 5 行和第 7 行预取指令的加入方式类似于 PrePartition 函数中预取技术的使用.

算法 3. *PrePartition*(*P*,*d*,*CA*,*size*).

输入:要计算的划分 *P*,开始维 *d*,计数数组 *CA* 和划分的大小 *size*;

输出:计数数组 *CA* 和有序指针数组 *SP*.

D 表示维数,C_i 表示第 i 个维的势,k 是预取距离.

```
Begin
01    for i=d;i≤D;i++ do                 /*累加阶段.计数阶段已在算法 2 的第 7 行、第 8 行得以实现*/
02      for j=2;j≤C_i+1;j++ do
03        CA[i][j]+=CA[i][j−1];
04    for j=1;j≤size;j++ do              /*移动阶段*/
05      if (j+k)≤size then
06        prefetch(P[j+k][d]);           /*发出预取指令*/
07      for i=d;i≤D;i++ do
08        SP[i][CA[i][P[j][i]]]=P[j];
09        CA[i][P[j][i]]−=1;
10    for i=d;i≤D;i++ do                 /*恢复计数*/
11      for j=1;j≤C_i;j++ do
12        CA[i][j]=CA[i][j+1]−CA[i][j];
End
```

第 1 行~第 3 行是累加阶段,得到计数的累加值.结果数组 *SP* 在第 4 行~第 9 行获得,我们增加预取指令预取 *k* 个元组之后的属性值(第 6 行).当维 *d* 的值读入到缓存中时,其他在同一缓存行的属性值也都在缓存中了.*CA*[*i*][*P*[*j*][*i*]]存储的是指向元组的指针在数组 *SP*[*i*]中的位置,所指向的元组包含属性值 *P*[*j*][*i*].移动阶段结束时,*SP* 保存的是从维 *d* 到维 *D* 的有序指针,*CA*[*i*][*j*]表示的是维 *i* 的划分 *j* 在数组 *SP*[*i*]中的起始位置.在第 10 行~第 12 行,维 *i* 的划分 *j* 的大小可以通过 *CA*[*i*][*j*+1]−*CA*[*i*][*j*]计算出来.

预取距离 *k*(在实验中为 4)可以通过下面的公式估计出来[23]:

$$k=\left\lceil \frac{T_t+T_l}{\max(T_t,T_c)} \right\rceil .$$

T_c 是每个循环的计算延迟;T_l 是内存定位延迟,包括芯片组延迟、总线仲裁等;T_t 是数据传输延迟,等同于每个循环的缓存行数目乘缓存行的延迟.T_c 可以通过测试得到,T_l 和 T_t 分别使用下面的公式计算得到:

$$T_l=T_{miss}-T_{line},$$

$$T_t=N_{line}\times T_{line}.$$

T_{miss} 指得是访问内存的缺失延迟,T_{line} 是从内存传输一个缓存行到 CPU 缓存的延迟,N_{line} 是每次循环访问的缓存行的数目.

4 实验研究

为了验证封闭冰山立方体计算方法的有效性,我们进行了详细的实验研究,将 BASE_ALG,CC_ALG 方法和文献[14]中提出的 C-Cubing(StarArray)和 C-Cubing(MM)方法进行了比较.据我们所知,这是目前计算封闭冰山立方体最快的方法.由于这 4 种算法的输出结果完全一样,因此我们来比较它们的计算时间.使用的数据包含模拟数据和真实数据.

4.1 封闭完全立方体计算

在这一节中,我们比较 BASE_ALG,CC_ALG,C-Cubing(StarArray)和 C-Cubing(MM)这 4 种算法在计算封闭完全立方体(即 min_sup=1)时的性能差异.使用的是模拟数据,分别在维的势(如图 4 所示)、维的数目(如图 5 所示)、基本表大小(如图 6 所示)以及倾斜度(如图 7 所示)这 4 个方面比较 4 种算法.图中“C-Cubing(S)”代表算法 C-Cubing(StarArray), “C-Cubing(M)”代表算法 C-Cubing(MM).

图 4 中的数据集包含 1 000K 条元组、7 个维,数据均匀分布,每个维的势从 20 变到 300.图 5 显示了维从 5 增加到 9 时不同算法的计算时间,数据集具有 1 000K 条元组,维的势是 50.图 6 中的元组数目从 200K 增加到

© 中国科学院软件研究所 http://www.c-s-a.org.cn

1 000K,数据集包含势是 50 的 7 个维.图 7 显示了使用 Zipf 分布,当数据集倾斜度从 0 变到 3 时,4 种算法性能的变化趋势.倾斜度为 3 表示最经常出现的值占整个元组的 83%[3],数据集有 1 000K 个元组、7 个维,每个维的势为 100.

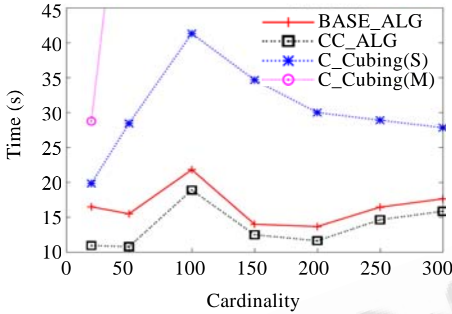

Fig.4 Computation w.r.t cardinality

图 4 势变化时的计算

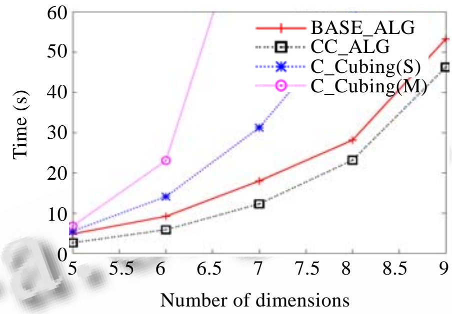

Fig.5 Computation w.r.t dimensions

图 5 维数目变化时的计算

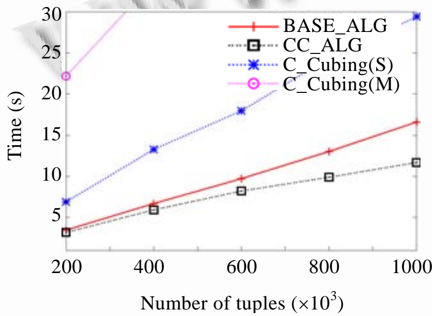

Fig.6 Computation w.r.t tuples

图 6 元组数目变化时的计算

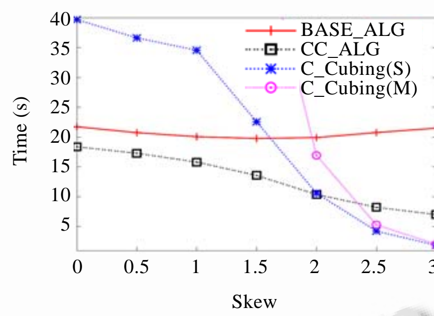

Fig.7 Computation w.r.t skewness

图 7 倾斜度变化时的计算

从图 4~图 7 显示的结果中可以得到以下两点结论:第一,在绝大多数情况下,BASE_ALG 和 CC_ALG 算法比 C-Cubing(StarArray)和 C-Cubing(MM)算法快很多,例如在图 4 中,当势是 50 时,C-Cubing(StarArray)和 C-Cubing(MM)分别需要 28.42s 和 74.66s,而 BASE_ALG 和 CC_ALG 仅需要 15.46s 和 10.76s.由于 C-Cubing(MM)花费的时间太长,为了能够清楚地区别其他算法的曲线,我们没有将它的时间完全显示在图中(其他的图也如此处理计算时间太长的算法).C-Cubing(MM)没有任何封闭单元剪枝功能,它只是在输出单元时判断是否是封闭单元,如果是,则输出,否则,不输出,因此它的性能完全依赖于 MM-Cubing 算法本身的性能.当 MM-Cubing 算法表现不好时,C-Cubing(MM)表现也不好.第二,任何情况下,CC_ALG 算法都比 BASE_ALG 算法性能要好,尤其是当数据集比较倾斜的时候.在图 7 中,当倾斜度是 3 时,CC_ALG 比 BASE_ALG 算法快 3 倍.这是因为倾斜的数据集比较稠密,而稠密数据的大小递归划分时不易减少,大划分容易产生较多的缺失,CC_ALG 算法能够有效地降低缺失,所以效果较明显.C-Cubing(StarArray)和 C-Cubing(MM)算法只在倾斜度非常大的时候性能才比较好,这是因为 StarArray 和 MM-Cubing 能够较快地处理特别稠密的数据.

4.2 封闭冰山立方体计算

本节中的实验比较各种算法在计算封闭冰山立方体时的表现.图 8~图 11 分别是算法随着势、维数目、元组数目和最小支持度变化时的运行结果.图 8~图 10 中最小支持度为 100,图 8 和图 9 的数据集有 1 000K 条元组,前者有 7 个维,维的势从 20 变到 500,后者维的势是 100,维的数目从 5 增长到 9.图 10 使用的数据集包含 7 个势为 100 的维.图 11 中最小支持度从 1 增加到 100,数据集有 1 000K 条元组、7 个维,势是 100.

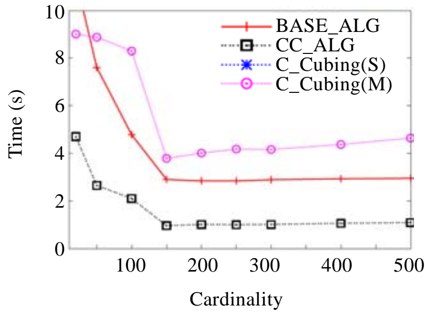

Fig.8 Iceberg computation w.r.t cardinality

图 8 势变化时的冰山计算

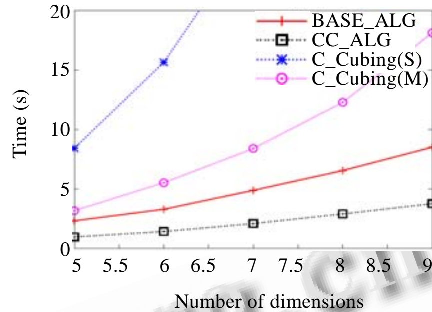

Fig.9 Iceberg computation w.r.t dimensions

图 9 维数目变化时的冰山计算

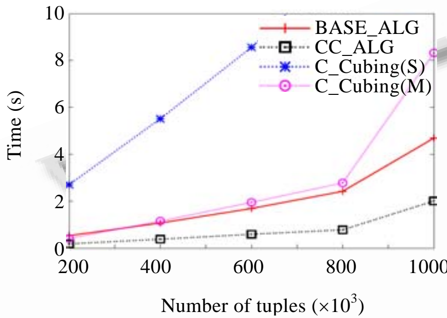

Fig.10 Iceberg computation w.r.t tuples

图 10 元组数目变化时的冰山计算

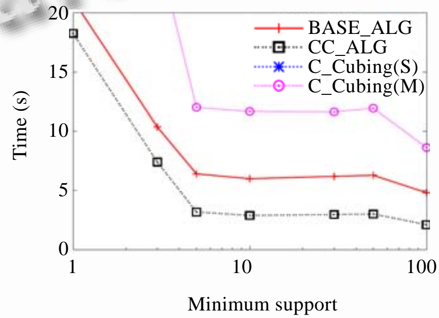

Fig.11 Iceberg computation w.r.t min_sup

图 11 最小支持度变化时的冰山计算

从结果中我们可以看出,对于冰山立方体计算,C-Cubing(StarArray)的性能比 C-Cubing(MM)的要差(图 8 和图 11 没有显示 C-Cubing(StarArray)的曲线是因为它的运行时间太长,在图 8 中最差,它比 CC_ALG 算法慢 22 倍),这和文献[14]的研究结果是一致的.因为这时封闭单元剪枝的影响减弱了,所以,即使 C-Cubing(MM)没有封闭单元剪枝,但其性能仍然较好.

虽然 C-Cubing(MM)比 C-Cubing(StarArray)的性能好一些,但仍然不如 BASE_ALG 和 CC_ALG 算法.在每个图中,CC_ALG 算法都能比 C-Cubing(MM)算法快 4 倍以上,这是因为两种算法都支持有效的冰山剪枝,但CC_ALG 算法还存在封闭单元剪枝策略,而且 CC_ALG 算法采用的是类似 BUC 算法的聚集方法,BUC 算法本身在很多情况下比 MM-Cubing 性能要好.比较 BASE_ALG 和 CC_ALG 算法我们可以发现,在大多数情况下,CC_ALG 算法的计算时间可以比 BASE_ALG 算法的时间缩短 2~3 倍.CC_ALG 算法能够取得如此大的性能改进,说明我们使用的预排序和软件预取技术是有效的.与上一节的实验结果比较可以发现,对于冰山立方体计算,CC_ALG 算法能够更大程度地改进 BASE_ALG 方法,这是因为冰山剪枝去掉了很多划分,不需要对这些划分判断是否封闭单元已经输出了,因而减少了这一步的代价之后,优化技术的作用更加明显了.

4.3 真实数据

这一节的实验是在真实数据上进行的.我们使用的数据是天气数据集 May95L.DAT[24],包含 28 个维,共有 1 195 149 个元组.我们使用了 9 个维,属性(势)分别是:lower cloud amount (10),high cloud amount x100(901), middle cloud amount (10),change code (10),solar altitude (1 798),relative lunar illuminance x100 (220),sea level pressure (11 001),wind speed (979)和 wind direction (352).

图 12 是最小支持度为 1 时,随着维数目的变化,4 种算法的运行结果,我们取前 5,6,7,8,9 个维.C-Cubing(MM)算法性能最差.大多数情况下,BASE_ALG 和 CC_ALG 算法的表现都比 C-Cubing(StarArray)算法要好,而CC_ALG 算法又比 BASE_ALG 算法要快.图 13 中最小支持度从 1 增加到 100,取前 7 个维.C-Cubing(MM)算法

© 中国科学院软件研究所 http://www.c-s-a.org.cn

的曲线随着最小支持度的增加下降得很快,当最小支持度比较大时,性能较好,但仍然没有超过 CC_ALG 算法.CC_ALG 算法在最好的情况下(min_sup=100),比 BASE_ALG 算法快 2.3 倍,比 C-Cubing(StarArray)算法快 2.8 倍.

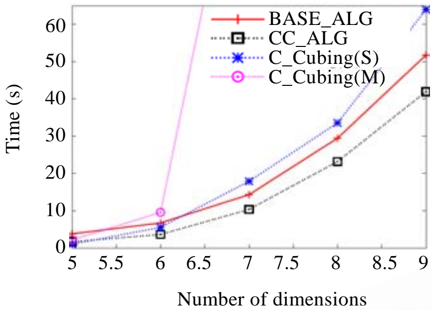

Fig.12 Real data computation w.r.t dimensions
图 12 维数目变化时的真实数据计算

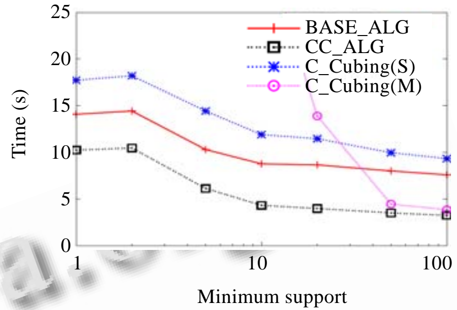

Fig.13 Real data computation w.r.t min_sup
图 13 最小支持度变化时的真实数据计算

4.4 缓存性能

为了清楚地理解各种算法的执行过程,我们将执行时间分解为 3 个部分[22]:忙碌时间(busy,在图中用 Busy 表示)、内存相关延迟(memory-related stalls,在图中用 Memory 表示)和其他延迟(other stalls,在图中用 Other 表示).忙碌时间指的是 CPU 做有用工作的时间;内存相关延迟包括 L1 缺失、L2 缺失和 TLB 缺失造成的延迟;其他延迟包含两部分:分支误测代价和资源相关的延迟.图 14 和图 15 显示了 CC_ALG,C-Cubing(StarArray)和 C-Cubing(MM)算法相对于 BASE_ALG 算法的执行时间分解结果.使用的是上一节中介绍的真实数据,图 14 包含前 6 个维,最小支持度是 1;图 15 包含前 7 个维,最小支持度是 20.

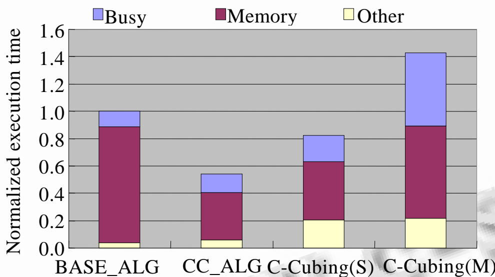

Fig.14 Execution time breakdown (min_sup=1)
图 14 执行时间分解(min_sup=1)

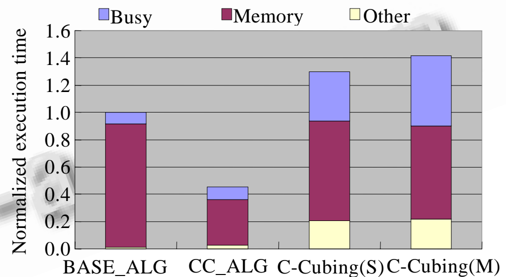

Fig.15 Execution time breakdown (min_sup=20)
图 15 执行时间分解(min_sup=20)

从图中可以看出,这 4 种算法的内存相关延迟占整个执行时间的比例都是最高的,说明大部分时间都是由各种内存相关的缺失引起的,CPU 真正忙碌的时间并不是很多.C-Cubing(MM)算法的忙碌时间比例是最高的,占执行时间的 37%(如图 14 所示)和 36%(如图 15 所示),但仍然要低于 CPU 空闲时间.与 BASE_ALG 算法相比,CC_ALG 算法有效地降低了内存相关延迟,因此缩短了立方体计算时间,这说明我们的预排序和软件预取技术确实如我们分析的那样,显著地减少了缺失.

5 相关工作

文献[10−14]提出一些无损压缩立方体大小的方法,与我们的工作比较相关.Condensed Cube[10]将“单个基本元组”压缩和“投影单个元组”压缩方法作为压缩立方体的基础,生成算法很慢.Dwarf[11]采用共享单元前缀的方式来压缩立方体,存储的是所有单元,而不是封闭单元.Quotient Cube[12,13]维护立方体的语义关系,输出很多包

© 中国科学院软件研究所 http://www.c-s-a.org.cn

含上界和下界的临时类,根据文献[14]的研究,其所提出的方法比 C-Cubing 方法要慢一个数量级.文献[14]引进一个可有效增量计算的度量“closedness”,提出基于聚集的 C-Cubing 方法,并与 MM-Cubing 和 Star 算法结合起来,形成计算封闭立方体的方法,这是目前和我们的工作最相关而且最快的方法,我们在实验中与它进行了详细的比较.

文献[2−5]给出计算完全立方体或是冰山立方体的算法.MultiWay[2]在多个维上同时聚集,但无法利用 Apriori 剪枝;BUC[3]首次提出冰山立方体的概念,采用自底向上的计算顺序;Star-Cubing[4]共享计算,利用 star-tree 结构将同时聚集和冰山剪枝结合起来;MM-Cubing[5]将数据分成不同的子空间,使用类似 MultiWay 的方法共享计算.这些基本算法对封闭冰山立方体计算方法的研究有很大的借鉴作用.

在研究计算机硬件对数据库的影响以及充分利用硬件资源方面已经有很多成果出现[18−22,25],对数据库在现代计算机硬件以及不同工作负载下的性能表现作了实验研究,针对数据库的索引、连接等核心算法提出了优化策略.这些方面的成果有助于我们更好的设计 OLAP 领域的方法.

6 总 结

本文提出了一种缓存敏感的封闭冰山立方体计算方法,这种方法采取自底向上的计算顺序,使用多种剪枝技术进行冰山剪枝和封闭单元剪枝,利用预排序和软件预取策略减少内存相关延迟,提高计算速度.与目前最快的方法在大量的模拟数据和真实数据上作了比较测试,实验结果表明,这种方法是快速、有效的.今后,我们将进一步研究基于封闭冰山立方体的查询及维护方法.另外,OLAP 领域其他方法的优化,如 what-if 分析,也是我们感兴趣的研究方向.

References:

[1] Gray J, Chaudhuri S, Bosworth A, Layman A, Reichart D, Venkatrao M, Pellow F, Pirahesh H. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. Data Mining and Knowledge Discovery, 1997,1(1):29−53. [doi: 10.1023/A:1009726021843]

[2] Zhao Y, Deshpande PM, Naughton JF. An array-based algorithm for simultaneous multidimensional aggregates. In: Peckham J, ed. Proc. of the ACM SIGMOD Int’l Conf. on Management of Data. New York: ACM Press, 1997. 159−170.

[3] Beyer K, Ramakrishnan R. Bottom-Up computation of sparse and iceberg CUBEs. In: Delis A, Faloutsos C, Ghandeharizadeh S, eds. Proc. of the ACM SIGMOD Int’l Conf. on Management of Data. New York: ACM Press, 1999. 359−370.

[4] Xin D, Han JW, Li XL, Wah BW. Star-Cubing: Computing iceberg cubes by top-down and bottom-up integration. In: Freytag JC, Lockemann PC, Abiteboul S, Carey MJ, Selinger PG, Heuer A, eds. Proc. of the 29th Int’l Conf. on Very Large Data Bases. San Francisco: Morgan Kaufmann Publishers, 2003. 476−487.

[5] Shao Z, Han JW, Xin D. MM-Cubing: Computing iceberg cubes by factorizing the lattice space. In: Hatzopoulos M, Manolopoulos Y, eds. Proc. of the 16th Int’l Conf. on Scientific and Statistical Database Management. Washington: IEEE Computer Society, 2004. 213−222.

[6] Hurtado CA, Mendelzon AO, Vaisman AA. Maintaining data cubes under dimension updates. In: Kitsuregawa M, Maciaszak L, Papazoglou M, Pu C, eds. Proc. of the 15th Int’l Conf. on Data Engineering. Washington: IEEE Computer Society, 1999. 346−355.

[7] Lee KY, Kim MH. Efficient incremental maintenance of data cubes. In: Dayal U, Whang KY, Lomet DB, Alonso G, Lohman GM, Kersten ML, Cha SK, Kim YK, eds. Proc. of the 32nd Int’l Conf. on Very Large Data Bases. New York: ACM Press, 2006. 823−833.

[8] Barbara D, Sullivan M. Quasi-Cubes: Exploiting approximations in multidimensional databases. SIGMOD Record, 1997,26(3): 12−17. [doi: 10.1145/262762.262764]

[9] Shanmugasundaram J, Fayyad U, Bradley PS. Compressed data cubes for OLAP aggregate query approximation on continuous dimensions. In: Proc. of the ACM SIGKDD Int’l Conf. on Knowledge Discovery and Data Mining. New York: ACM Press, 1999. 223−232. http://www.informatik.uni-trier.de/~ley/db/conf/kdd/kdd99.html

[10] Wang W, Feng JL, Lu HJ, Yu JX. Condensed cube: An effective approach to reducing data cube size. In: Agrawal R, Dittrich K, Ngu AH, eds. Proc. of the 18th Int’l Conf. on Data Engineering. Washington: IEEE Computer Society, 2002. 155−165.

[11] Sismanis Y, Deligiannakis A, Roussopoulos N, Kotidis Y. Dwarf: Shrinking the PetaCube. In: Franklin MJ, Moon B, Ailamaki A, eds. Proc. of the ACM SIGMOD Int’l Conf. on Management of Data. New York: ACM Press, 2002. 464−475.

© 中国科学院软件研究所 http://www.c-s-a.org.cn

[12] Lakshmanan LVS, Pei J, Han JW. Quotient cube: How to summarize the semantics of a data cube. In: Bernstein PA, Ioannidis Y, Ramakrishnan R, eds. Proc. of the 28th Int'l Conf. on Very Large Data Bases. San Francisco: Morgan Kaufmann Publishers, 2002. 778−789.

[13] Lakshmanan LVS, Pei J, Zhao Y. QC-Trees: An efficient summary structure for semantic OLAP. In: Halevy AY, Ives ZG, Doan AH, eds. Proc. of the ACM SIGMOD Int'l Conf. on Management of Data. New York: ACM Press, 2003. 64−75.

[14] Xin D, Shao Z, Han JW, Liu HY. C-Cubing: Efficient computation of closed cubes by aggregation-based checking. In: Liu L, Reuter A, Whang KY, Zhang JJ, eds. Proc. of the 22nd Int'l Conf. on Data Engineering. Washington: IEEE Computer Society, 2006.

[15] Balmin A, Papadimitriou T, Papakonstantinou Y. Hypothetical queries in an OLAP environment. In: Abbadi AE, Brodie ML, Chakravarthy S, Dayal U, Kamel N, Schlageter G, Whang KY, eds. Proc. of the 26th Int'l Conf. on Very Large Data Bases. San Francisco: Morgan Kaufmann Publishers, 2000. 220−231.

[16] Lakshmanan LVS, Russakovshy A, Sashikanth V. What-If OLAP queries with changing dimensions. In: Proc. of the 24th Int'l Conf. on Data Engineering. 2008. 1334−1336. http://www.informatik.uni-trier.de/~ley/db/conf/icde/icde2008.html

[17] Hennessy JL, Patterson DA. Computer Architecture: A Quantitative Approach. 3rd ed., San Francisco: Morgan Kaufmann Publishers, 2002.

[18] Boncz PA, Manegold S, Kersten ML. Database architecture optimized for the new bottleneck: memory access. In: Atkinson MP, Orlowska ME, Valduriez P, Zdonik SB, Brodie ML, eds. Proc. of the 25th Int'l Conf. on Very Large Data Bases. San Francisco: Morgan Kaufmann Publishers, 1999. 54−65.

[19] Rao J, Ross KA. Making B^+-Trees cache conscious in main memory. In: Chen WD, Naughton JF, Bernstein PA, eds. Proc. of the ACM SIGMOD Int'l Conf. on Management of Data. New York: ACM Press, 2000. 475−486.

[20] Chen S, Gibbons PB, Mowry TC. Improving index performance through prefetching. In: Aref WG, ed. Proc. of the ACM SIGMOD Int'l Conf. on Management of Data. New York: ACM Press, 2001. 235−246.

[21] Chen S, Ailamaki A, Gibbons P, Mowry T. Improving hash join performance through prefetching. In: Proc. of the 20th Int'l Conf. on Data Engineering. Washington: IEEE Computer Society, 2004. 116−127. http://www.informatik.uni-trier.de/~ley/db/conf/icde/icde2004.html

[22] Ailamaki A, DeWitt DJ, Hill MD, Wood DA. DBMSs on a modern processor: Where does time go? In: Atkinson MP, Orlowska ME, Valduriez P, Zdonik SB, Brodie ML, eds. Proc. of the 25th Int'l Conf. on Very Large Data Bases. San Francisco: Morgan Kaufmann Publishers, 1999. 266−277.

[23] IA-32 Intel architecture optimization reference manual. 2005. http://srl.cs.jhu.edu/manuals/24896612.pdf

[24] Hahn C, Warren S. Extended edited synoptic cloud reports from ships and land stations over the globe. 1952~1996. 1999. http://cdiac.esd.ornl.gov/ftp/ndp026c/ndp026c.txt

[25] Liu DW, Luan H, Wang S, Qin B. Main memory database TPC-H workload characterization on modern processor. Journal of Software, 2008,19(10):2573−2584 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/19/2573.htm [doi: 10.3724/SP.J.1001.2008.02573]

附中文参考文献:

[25] 刘大为,栾华,王珊,覃飙.内存数据库在 TPC-H 负载下的处理器性能.软件学报,2008,19(10):2573−2584. http://www.jos.org.cn/1000-9825/19/2573.htm [doi: 10.3724/SP.J.1001.2008.02573]

栾华(1980－),女,山东龙口人,博士,CCF学生会员,主要研究领域为高性能数据库新技术,数据仓库技术.

王珊(1944－),女,教授,博士生导师,CCF高级会员,主要研究领域为高性能数据库新技术,数据库信息检索,数据仓库与 BI 技术.

杜小勇(1963－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为智能信息检索,高性能数据库,知识工程.

© 中国科学院软件研究所 http://www.c-s-a.org.cn