维普资讯 http://www.cqvip.com

时态数据的可变 Hash 索引*)

蒋夏军　吴慧中　李蔚清
(南京理工大学计算机系　南京 210094)

摘　要　索引技术是时态数据库查询优化的重要方法之一。本文提出的可变 Hash(VH)索引是建立在时间属性上的一种新的动态索引技术,主要目的是提高时态数据库快照查询的效率。由于时间的不确定性,在时态数据的时间属性上建立 Hash 索引比较困难。VH 索引克服了 Hash 索引这一难点,提出了索引参数可变的思想,并应用 B^+-树对 Hash 参数进行组织。查询时由时间值在 B^+-树上获得 Hash 参数,进而确定数据的存储地址。通过对其时间复杂度和空间复杂度的理论分析以及实验验证,表明该索引技术可以减少索引查找以及读取数据的 I/O 次数,并具有理想的空间利用率。

关键词　时态数据库,可变 Hash 索引,快照查询,时间复杂度

Variable Hashing for Temporal Data

JIANG Xia-Jun　WU Hui-Zhong　LI Wei-Qing
(Department of Computer, Nanjing University of Science & Technology, Nanjing 210094)

Abstract　Index technology is one of the important factors during the process of data query optimizing, especially for temporal database. A new hashing method for temporal data is designed to improve the efficiency of database snapshot query in this paper, and the method is called Variable Hashing (VH). Generally, it's difficult to establish hashing index for time value because of the time's indeterminacy. VH solves this problem, and it is based on the start time of a database's transaction time attribute. The parameters of hashing function are variableaccording to the time attribute of tuples, and they are organized as a B^+-tree. Using a time value to query the B^+-tree can get the hashing parameters, and the parameters can be used to calculate the address of target data. Carefull analysis and experimental test show that the time complexity of VH's snapshot query is better than other snapshot index methods, and its space complexity is also optimal.

Keywords　Temporal database, Variable hashing method, Snapshot query, Timecomplexity

1　引言

时态数据库是记录随时间变化的信息(即时态数据)的数据库。大量数据库应用系统在本质上具有时态特性,如果将与时间有关的操作排除在数据库系统之外,将给应用程序开发人员增加许多额外的负担,同时导致系统运行效率低下。因此,有关时态数据库技术的研究工作自从数据库技术诞生以来就相当活跃,并取得了许多研究成果,尤其是在时态数据库模型等方面。

由于数据库应用领域的复杂性,研究人员尚不能总结出一套普遍适用的有关时态数据的组织和使用的规律,对时态数据的有效访问就是一个典型的例子。时态数据库往往同时记录应用系统的当前状态和历史状态,使得数据量非常庞大,对数据的访问方法提出了挑战。索引技术是实现数据高效访问的主要方法。对时态数据的索引方法通常有两种:树索引和 Hash 索引。前者可以同时满足时间属性和其它属性的索引需求,后者对于时间属性往往是低效的,有时甚至是不可行的。原因在于 Hash 函数的确定性和实体状态变化时刻(时间属性)的不确定性之间存在着矛盾,所以 Hash 索引在时态数据库中的应用仅仅局限在非时间属性上,如 PPLH(Partially Persistent Linear Hashing)只是在实体号上建立 Hash 索引,从而将不同的实体分布到相应的存储单元中,在时间属性上还需要其他索引方法。

Hash 索引的主要优势在于具有相对固定的查询费用(通常是 1~2 次 I/O),而基于平衡树的索引具有对数级的查询费用($\log_B N$,其中 N 为元组数,B 为每存储单元(一般为页面)包含的索引项数。当 N 较大时,通常需要 4 次以上的 I/O)。此外,Hash 索引本身所需的存储空间与树索引相比几乎可以忽略。因此,在时间属性上设计 Hash 索引是一项很有意义的工作。我们提出了一种新的时态数据的索引方法——可变 Hash(VH)索引,该索引技术为不同的时间窗口确定不同的 Hash 参数,以达到提高查询效率的目的。通过理论分析和实验验证,证明了该索引技术的有效性,并得出在实体状态变化率接近的情况下效果最好的结论。

2　可变 Hash(VH)索引

VH 索引的设计思想是:将应用系统的时间轴划分为若干个分段,在每个分段上确定相应的 Hash 函数,力求在时间的不确定性和 Hash 索引的确切查询之间找到平衡点。

时态数据库系统通常使用两类时间来描述客观世界:有

*)受国防科技预先研究项目支持。**蒋夏军**　博士生,主要研究方向为仿真支撑环境、数据库技术等;**吴慧中**　教授,博导,主要研究方向为虚拟现实、系统仿真、图像建模与处理等;**李蔚清**　博士生,主要研究方向为虚拟现实、图形学、系统仿真等。

维普资讯 http://www.cqvip.com

效时间和事务时间，本文提出的 Hash 索引将建立在事务时间之上。事务时间具有单向递增性，时间的起点为数据库的产生时间，暂时的终点为系统的当前时间值。时态数据库采用下述模型：设关系 $R=(Eid,A_1,A_2,\cdots,A_n,T)$，其中 T 为时间属性，Eid 是关系 R 中的实体标识符。同一实体在不同时间的不同状态由 R 中具有相同 Eid 值的元组表示，且 $Key(R)=(Eid,T)$组成关系 R 的唯一键值。时间属性 T 记录元组的事务时间的起始值，其终止值由该元组表示的实体状态的下一次改变确定，即由 Eid 值相同、T 值最接近的下一个元组确定。如果遇到实体的删除操作，则对记录该操作的元组做特殊标记。

为了方便阐述 VH 索引的实现原理，我们先列出相关符号及其意义：

TS_i——第 i 个时间分段，在该分段上具有确定的 Hash 参数。

$T_{0,i}$——TS_i 的起始时间点。

ΔT_i——TS_i 内的滑动时间窗。

(以上两个变量为 Hash 函数在 TS_i 分段上的参数)

SU——存储单元，其大小为磁盘存储页面容量的整数倍。

TL——元组大小(通常为元组所有属性的字节和)。

TN——每个存储单元所能容纳的最大元组数，$TN=SU/TL$。

AN_j——第 j 个存储单元的活动实体总数(即该存储单元的时间区间内，数据库快照中存在过的实体总数)。

$LT\%$——每个存储单元存放的数据占存储容量的下限百分比。

$MT\%$——每个存储单元存放的数据占存储容量的中间百分比(在产生新的 TS_i 时确定ΔT_i 时使用)。

$UT\%$——每个存储单元存放的数据占存储容量的上限百分比。

(以上三项满足：$0<LT\%<MT\%<UT\%\leqslant 100\%$)

UN——存储地址，即 SU 号。

UN_i——TS_i 时间分段的起始存储单元号，即 TS_i 的起始地址。

TS_i 分段上的 Hash 函数如图 1 所示，每一对参数($T_{0,i}$，ΔT_i)确定一个 Hash 函数，Hash 函数的参数和 TS_i 的起始地址UN_i 是时间查询的重要信息，用 B^+-树对($T_{0,i}$,ΔT_i,UN_i)进行组织(在 $T_{0,i}$上建立索引)。实体状态改变时(元组的更新操作)，根据其起始时间存入相应时间分段的某个存储单元。给定时间值 t，查找该时刻数据库的快照(或某些实体的状态)的步骤如下：首先在参数树上由 t 值查到相应的($T_{0,i}$，ΔT_i,UN_i)，然后根据式(1)确定其存储地址 UN(即存储单元号)。下面介绍了存储单元的数据结构以后，我们知道UN 号和($UN-1$)号存储单元中的数据决定了数据库在 t 时刻的快照。

$$UN=UN_i+\lfloor (t-T_{0,i})/\Delta T_i \rfloor \tag{1}$$

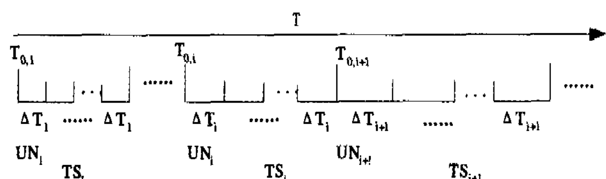


图 1 VH 索引的时间分段及其 Hash 参数

3 Hash 参数的确定

确定每个时间分段上的 $T_{0,i}$、ΔT_i、UN_i 值是 VH 索引的核心问题。为了避免查询费用的波动(极端情况下可能查询整个存储空间)，对存储单元的活动实体数和存放的元组作如下限制：(1)$AN_j\leqslant TN*MT\%$；(2)存储单元对应的时间区间内活动实体的元组在该存储单元至少出现一次。数据库在 $T=0$ 时刻创建，对实体的操作包括插入(实体加入系统)、删除(实体退出系统)和更新(实体状态改变)，每一个操作在数据库表中体现为插入一个元组。我们规定数据库中的时间为事务时间，所以元组的操作只在最近的存储单元中进行(具有 Append-Only 特性)。

我们首先通过一个例子说明 Hash 参数的确定过程。参见图 2，设 TC 为存储单元中的元组数，m 为同一 TS_i 上的存储单元的序号。(1)为插入元组(6,4)、(2,7)、(4,9)和(6,24)后存储单元 $B1$ 的状态。此时 $AN_1=3$。由于 $Eid=6$ 的元组在前 AN_1 个存储位置已经出现，因此将元组(6,24)插入末端地址向前的第一个空位。(2)为插入元组(4,44)后 $B1$ 的状态。接着将插入元组(2,46)。此时，$TC=9=TN*MT\%$，元组(2,46)将插入下一个存储单元，TS_1 的 Hash 参数已经可以确定：$T_{0,1}=0,\Delta T_1=46-T_{0,1}=46,UN_1=1$。初始化下一个存储单元 $B2$(参见(3))，此时 $AN_2=4,TC=4$，并假定 $B2$ 为 TS_1 内的存储单元(各项 Hash 参数同 $B1$)，$m=2$。从元组(2,46)到(2,72)的时间值在区间$[T_{0,1}+\Delta T_1*(m-1),\ T_{0,1}+\Delta T_1*m)$(即[46,92))上，插入元组(2,72)前存储单元 $B2$ 已经存满(参见(4))，必须重新确定 Hash 参数，开始新的时间分段 TS_2。方法如下：将 $B2$ 恢复到(3)的状态，从元组(2,46)开始依次插入(按时间值)各元组，直到元组(3,65)，此时 $TC=9=TN*MT\%$，确定 TS_2 的 Hash 参数为：$T_{0,2}=T_{0,1}+\Delta T_1*(m-1)=46,\Delta T_2=65-T_{0,2}=19,UN_2=2$(参见(5))。初始化新的存储单元，$AN_3=7,TC=7,m=2$。从元组(3,65)开始插入，直到元组(5,87)，插入元组的时间值区间$[T_{0,2}+\Delta T_2*(m-1),\ T_{0,2}+\Delta T_2*m)$(即[65,84))上，而元组(5,87)是首个超出时间区间的元组。此时 $TC=9$，满足 $TN*LT\%\leqslant TC\leqslant TN*UT\%$(即 $8\leqslant 9\leqslant 10$)，不必调整 Hash 参数，$B3$ 仍属于 TS_2 时间分段(参见(6))。

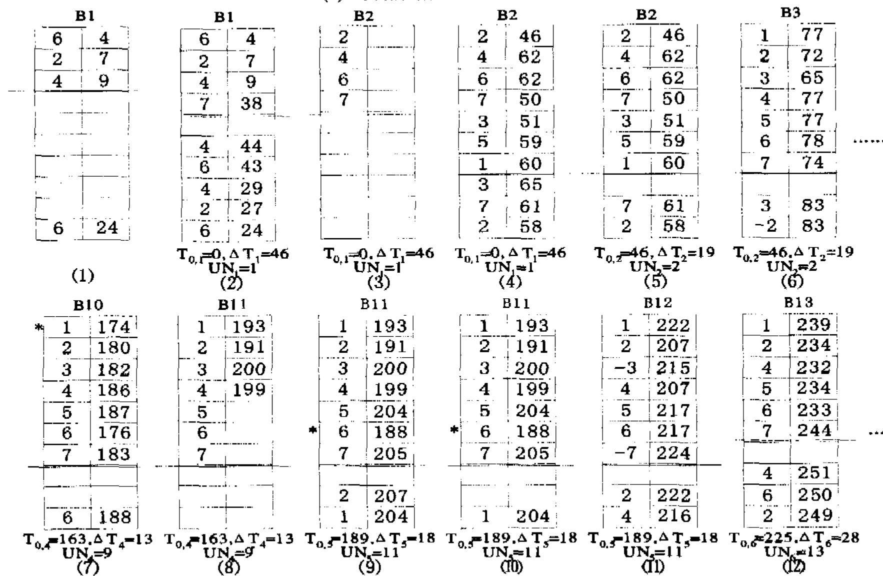
$B10$ 为 TS_4 时间分段上的第 2 个存储单元，此时 $T_{0,4}=163,\Delta T_4=13,UN_4=9$。我们开始下一个存储单元的创建过程，初始化 $B11$：$AN_{11}=7,TC=7,m=3$。元组(2,191)、(1,193)、(4,199)和(3,200)的时间值在区间$[T_{0,4}+\Delta T4.(m-1),\ T_{0,4}+\Delta T_4*m)$(即[189,202))上，插入后存储单元的状态见(8)所示，$TC=7$。因为下一个元组(1,204)的时间值已经超出时间区间[189,202)，而 $TC<TN*LT\%$，所以必须调整 Hash 参数，开始新的时间分段 TS_5，继续插入元组，直到元组(4,207)，此时 $TC=9=TN*MT\%$，TS_5 的 $Hash$ 参数为：$T_{0,5}=T_{0,4}+\Delta T_4*(m-1)=189,\Delta T_5=207-T_{0,5}=18,UN_5=11$。初始化新的存储单元 $B12$，继续插入元组(4,207)，由于该元组的时间值与插入前一单元的元组(2,207)相同，为保证查询时(1)式的正确性，将(2,207)移入 $B12$(参见(9)、(10)和(11))。由于 $Eid=6$ 的实体在时间区间[189,207)未发生操作，根据存储单元 $B10$，该实体在此时间区间上仍然活动，我们根据 $B10$ 将与实体 6 的状态相对应的元组加入 $B11$“*”所示的位置。我们在删除操作的实体号前加“－”号，如(6)中的－2、(11)中的－3 和－7。执行了删除操作的实体可能

影响下一个存储单元的 AN_i 值，参见(11)和(12)所示，$Eid=3$ 的实体在存储单元 $B13$ 中不活动，因此未出现 $Eid=3$ 的元组。

T=0

Eid	T	Eid	T	Eid	T	Eid	T	Eid	T	Eid	T	Eid	T	Eid	T
6	4	2	46	3	65	5	87	6	176	4	199	5	217	1	239
2	7	7	50	2	72			2	180	3	200	6	217	7	244
4	9	3	51	7	74			3	182	1	204	1	222	2	249
6	24	2	58	1	77	⋮		7	183	5	204	2	222	6	250
2	27	5	59	4	77			4	186	7	205	-7	224	4	251
4	29	1	60	5	77			5	187	2	207	4	232	-5	253
7	38	7	61	6	78			6	188	4	207	6	233	⋮	
6	43	4	62	-2	83			2	191	-3	215	2	234		
4	44	6	62	3	83			1	193	4	216	5	234		

(a) 原始数据



(b) 存储单元中的数据及相关 Hash 参数

图 2　Hash 参数的确定过程

$TN=10, LT\%=80\%, MT\%=90\%, UT\%=100\%$，$Eid$ 从 1 到 7。“*”表示该实体在本存储单元是活动的，但未发生任何操作，对应元组是由系统根据前一存储单元的内容加入的，目的是提高快照查询的效率。Eid 前加“－”，表示元组对应的为删除操作。

在一个新的时间分段上确定 ΔT_i 的过程可以分为 3 类：初始化 ΔT_i，缩短 ΔT_i 和延长 ΔT_i。$B1$ 是从数据库初始状态开始的处理过程，$B1$ 和 $B2$ 以及 $B10$ 和 $B11$ 之间都经历了 Hash 参数的改变。$B2$ 因为原 ΔT_i 太大(时间区间内的元组数量超过了 $TN*UT\%$)而对其进行了缩减，$B11$ 因为原 ΔT_i 太小(时间区间内的元组数量低于 $TN*LT\%$)而对其进行了扩大。Hash 参数改变后的首个存储单元通常包括($TN*MT\%$)个元组，只是在出现相等 T 值的情况下发生例外，如 $B11$ 中将(2,207)移入了下一个存储单元，因此元组数小于($TN*LT\%$)。

我们给出确定 Hash 参数的详细算法。从初始状态出发，第一个时间分段的 $T_{0,1}=0, UN_1=1$。ΔT_1 通过循环执行下述过程确定($AN_1=0, T=0$)：

```
1   根据实体状态的改变和 T 值确定插入元组，记 t 为该元组的 T 值；
2   if(TC< TN* MT%){
3     if(该存储单元中不存在相同 Eid 值的元组){
4       AN1 ++；
5       在存储单元的第 AN1 个存储位置存入；
6     }
7     else
8       在存储单元中由末端地址开始向起始地址寻找，找到第一个空位后存入；
9     TC ++；
10  }
11  else
12  ΔT1 =t−0；//元组的插入将进入新的循环
```

以后新产生存储单元(设为第 B_j 个)时，根据 AN_{j-1} 值(前一个存储单元的活动实体数目)以及前一个存储单元内与删除操作相关的元组，确定本存储单元的 AN_j 值，保留存储单元的前 AN_j 个元组的存储空间，并将对应活动实体的 Eid 值(仅存放该值)存入前 AN_j 个元组的存储空间(参见图 2 中(3))，使 $TC=AN_j$。每产生一个元组(T 值为 t)，执行下述过程(设该存储单元所在时间分段的 $Hash$ 参数为：$T_{0,i}$、ΔT_i 和 UN_i，令 $m=B_j-UN_i+1$，m 表示存储单元在该时间分段上的序号)：

```
13  if(t< T0,i+ m*ΔTi){
14      if (TC< TN*UT%){
15          if(该存储单元中不存在相同 Eid 值的元组){
16              ANj ++；
17              在存储单元的第 ANj 个位置存放该元组；
18          }
19          else{
20              if(存储单元的前 ANj 个位置中已经存放相同 Eid 的元组) //除 Eid 值外，其它值也已存入
21                  在存储单元中由末端地址开始向起始地址寻找，找到第一个空位后存入；
22              else //除 Eid 值外，其它值未存入
23                  存储单元的前 ANj 个位置中 Eid 值相同处存入该元组；
24          }
25          TC ++；
26      }
27      else{
28          将该存储单元中的所有元组存入一个临时存储单元，并恢复到新产生时的状态，再将临时存储单元中的元组根据 T 值由小到大存入该元组，更新 ANj 和 TC，直到 TC=TN* MT%，记下一个元组的 T 值为 t；
29          T0,i +1 = T0,i + (m−1) * ΔTi，ΔTi+1 = t− T0,i+1，UNi+1 =Bj；
30          产生新的存储单元 Bj+1 并初始化(包括 ANj+1 和 TC)，m=2；//Bj+1 =j+1
31          将临时存储单元中的剩余元组和新插入的元组加
```

```
        入该存储单元,更新 N_{j+1} 和 TC;
32      }
33  }
34  else{
35          if((TC> = TN * LT%)&&(! Enlarge)){ // Enlarge 为布尔变量,初始为 false,判断 ΔT 是否处在扩大过程中
36          产生新的存储单元 B_{j+1} 并初始化(包括 AN_{j+1} 和 TC),m++;//B_{j+1}=j+1
37          将元组插入该存储单元,更新 N_{j+1} 和 TC;
38      }
39    else{
40      if(TC<TN * MT%){
41          Enlarge=ture;
42          将元组插入该存储单元,并更新 AN_j 和 TC;
43      }
44      else{
45          Enlarge=false;
46          T_{0,i+1}=T_{0,i}+(m-1)*ΔT_i,ΔT_{i+1}=t-T_{0,i+1},UN_{i+1}=B_j;
47          产生新的存储单元 B_{j+1} 并初始化(包括 AN_{j+1} 和 TC),m=2;//B_{j+1}=j+1
48          将元组插入该存储单元,更新 N_{j+1} 和 TC;
49      }
50    }
51  }
```

28、31、37、42 和 48 行的元组插入过程与 15~25 行相同。如果两个或若干个 T 值相等的元组分别位于不同的存储单元,必须将其全部移入后一个存储单元。我们在上述算法中省略了这一过程。

4 性能分析

数据库的快照查询是本索引的主要应用。具有分页特性的树索引(如 B^+-树、R-树等),实现快照查询的最优时间复杂度为:$O(\log_B N+A/B)$(B 为磁盘页面所容纳元组的数量,N 为元组总量,A 为查询结果的元组数量)。要使整个查询费用优于前式,只能通过减少查询索引的 I/O 次数(因为 A/B 值已经最小)实现,即尽量使 $\log_B N$ 变小。VH 索引可以实现这一优化。

设 λ_i 为第 i 个时间分段所包含的存储单元数(即($T_{0,i}$,ΔT_i,UN_i)控制的存储单元数),存储单元的平均有效空间利用率为 $U\%$,λ 为所有时间分段控制的平均存储单元数,C 为时间分段总数,则 $\lambda=\frac{1}{C}\sum_{i=1}^{C}\lambda_i$。用 B^+-树组织 Hash 参数时,树高为 $\log_B\frac{N}{\lambda*TN*U\%}$,查询目标节点的时间复杂度为 $O(\log_B\frac{N}{\lambda*TN*U\%})$ 次 I/O,因此 VH 索引快照查询的时间复杂度为

$$O(\log_B N-\log_B\lambda-\log_B(TN*U\%)+A/B) \quad (2)$$

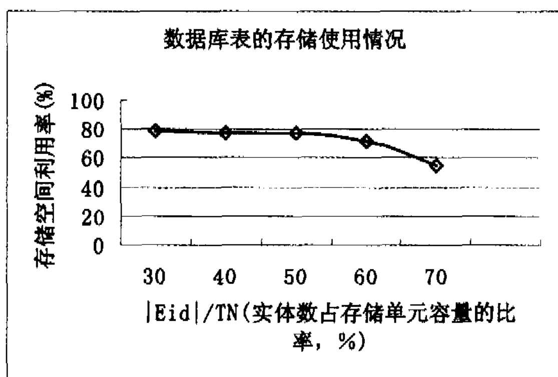
λ 越大,优化性能越明显,即使在极端情况下(每个存储单元对应一个 Hash 函数,$\lambda=1$),也能减少 $\log_B(TN*U\%)$ 次 I/O。下面的实验表明,在某些情况下可以取得较大的值。

浪费的空间主要由两部分组成:(1)存放本存储单元对应的时间段内未发生操作,但依旧活动的实体所对应的元组(图 2 中由"*"标出);(2)尚空着的存储位置。存放数据库元组所需的空间为:$\frac{N}{TN*U\%}$,通常可以保证 $U\%>70\%$(参见实验部分),所以数据库元组的空间复杂度为 $O(N/TN)$。这样的空间利用率与经过优化的其它以快照查询为目标的索引(如 Snapshot Index)相同。

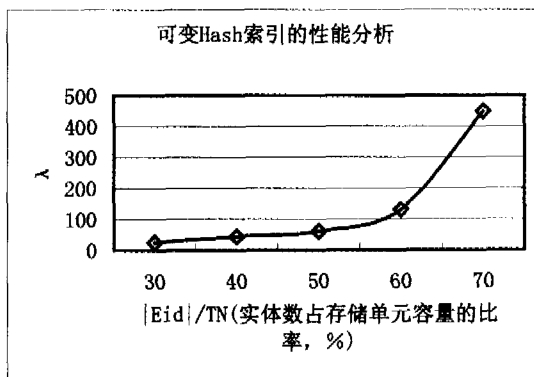
对于给定实体号(可以是集合)和时间点的查询,其时间复杂度与整个数据库的快照查询相同。如果在数据库系统的 Eid 值域内做一个划分(可以应用建立在实体号上的 Hash 方法),在不同的部分分别建立 VH 索引,可以进一步优化此类查询。对于包含时间区间的查询,利用区间起点和终点确定存储单元的范围,并在该范围内进行遍历。当时间区间较大时,查询费用将很高,极端情况下可能遍历整个数据存储空间。

5 实验结果及分析

VH 索引对快照查询(包括实体的时间点查询)具有确定的 I/O 费用,因此我们主要通过实验考察其存储性能以及 λ 的变化趋势。实验的有关参数配置如下:$TN=100$,$LT\%=60\%$,$MT\%=80\%$,$UT\%=100\%$,实验的仿真时间为 100000,元组删除操作约占所有操作的 9%。设实体状态改变的时间区间(即事务时间区间)的长度在[1,100]上均匀分布,改变 $|Eid|$(表示不同 Eid 值的数目,即实体数)值,实验结果如图 3 所示。线性 Hash (LH)索引的存储空间利用率在 65%~95%范围内波动(见参考文[4])。由图 3(a)可知,当 $|Eid|/TN\leqslant 60\%$ 时,存储空间利用率在 70%~80%之间波动,因此 VH 的存储空间利用情况是比较理想的。由图 3(b)可知,当 $|Eid|TN\geqslant 50\%$ 时,$\lambda>60$。若 $B=TN=100$,则 $\log_B\lambda+\log_B(TN*U\%)>1.81$。根据(2)式,可以平均减少近 2 次 I/O。



(a) 数据库表的存储性能



(b) VH 索引的性能

图 3 可变 Hash 索引的存储性能及的变化趋势

VH 索引的参数 $T_{0,i}$ 和 ΔT_i 的确定与元组的事务时间区间直接相关,因此实体相邻两次状态改变的时间区间长度的分布状况是决定可变 Hash 索引性能的重要因素。设 $|Eid|=50$,同时改变时间区间长度的分布状况(在一定的区间上均匀分布),实验的其它配置与上述相同。得到的实验结果如表

(下转第 242 页)

维普资讯 http://www.cqvip.com

为了改善聚类的目标函数值,算法需要识别出来什么样的聚类是不好的聚类,并改变该聚类的中心。当算法所找到的两个聚类的中心实际上应该属于一个聚类时,就会出现不好的聚类。这两个聚类竞相提高目标函数的值,但是,却因为算法可以识别的聚类的个数在初始化时已经确定,所以有一个真正的聚类并没有被识别出来,而这个聚类所包含的对象被列入孤立点集。这种情形是不合理的。

为了解决这个问题,在算法中需要注意,是不是某个聚类在所有相关维上的函数值即 ϕ_i 特别低,或者说是出现两个特别相似的聚类。这时,就说明相关的聚类是不好的,应使用一个新的聚类中心。

3.4 带有输入知识的高维聚类算法

算法在初始化时,先确定一些聚类的种子,即潜在的聚类的中心。然后,将数据集中的每个对象分配到每一个聚类,使得目标函数的值最大,暂时使用聚类中心在 V_j 上的投影替代 $\tilde{\mu}_{ij}$ 计算目标函数的值。如果某个对象不能改善任何聚类的 ϕ_i 值,则它被认为是一个孤立点,放入孤立点集。为所有的对象分配了所在聚类之后,重新确定每个聚类的相关维集,并使用实际的中心再次计算目标函数值。如果新得到的目标函数值是目前为止最大的值,记录所形成的聚类。否则,保持前面记录的最好的聚类不变。对于新得到的聚类,重新计算每个聚类的中心。在下一次迭代中,使用新得到的聚类中心代替 $\tilde{\mu}_{ij}$,并重新为数据集里的每个对象分配聚类。重复上述操作,直到在几次迭代过程后,所得到的目标函数的值没有大的改善为止。该算法类似于 K-medoids 算法,具体的步骤如下:

① 加载数据集。

② 输入聚类个数以及相关知识(假设所输入的知识全部正确)。

③ 为每一个聚类分配一个聚类中心,并把每个对象分配到与之最相似的聚类,使得在这种情形下的目标函数值最大。

④ 选择相关维。

⑤ 如果所得聚类使得目标函数值是到目前为止的最大值,则记录该组聚类结果。否则,替换聚类中心。

⑥ 重复③,④,⑤直到所得到的目标函数在几次迭代中没有大的改善为止。

3.5 复杂性分析

算法的时间复杂度和空间复杂度分别为 $O(knd)$ 和 $O(nd)$。相对于其他同类问题的算法而言,线形的时间复杂度使得该方法对于解决大规模数据集的聚类问题更实际。

结论 本文从聚类高维数据中的应用出发,提出了带有输入知识的高维数据聚类算法,并对算法所涉及的主要技术进行讨论。本文从高维数据的聚类的相关维着手,提出正确地使用输入知识来指导聚类过程,以便把学习所获得的知识或者专家知识应用到聚类中,指导聚类进行进一步学习,利于知识的增长,扩大了聚类的应用前景。

参考文献

1 Aggarwal C C, Yu P S. Finding generalized projected clusters in high dimensional spaces. In: ACM SIGMOD Intl. Conf. on Management of data, 2000.

2 Berkhin P. survey of clustering data mining techniques. http://www. accrue. com/products/ researchpapers. html

3 Talavera L, Bejar J. Integrating declarative knowledge in hierarchical clustering tasks. In:Intl. Symposium on Intelligent Data Analysis, 1999

4 Xing E P, Ng A Y, Jordan M I, Russell S. Distance metric learning, with application to clustering with side-information. In Advances in Neural Information Processing Systems 15, 2003

5 Cohn D, Caruana R, McCallum A. Semi-supervised clustering with user feedback, 2000

6 Han Jiawei, Kamber M. 数据挖掘概念与技术. 北京:机械工业出版, 2001

(上接第 133 页)

1 所示。事务时间区间在区间[1,10]和[90,100]上均匀分布时,λ 值均超过了 100,存储空间利用率也较高;在区间[1,100]上,λ 值下降明显;在区间[1,10]和[90,100]各占50%时,λ 值下降为 10.94,存储有效利用率同时下降为 69.21%。由此可见,VH 索引比较适用于状态变化的时间间隔相对接近的实体集。具体应用时,可以根据先验知识,将实体分成不同的簇,在各簇上分别建立 VH 索引,例如在仿真系统中记录实体状态的数据库可以采用这一方法。

表 1 VH 索引的性能随事务时间区间分布状况的变化

时间区间长度	[1,10]	[90,100]	[1,100]	[1,10]和[90,100]各 50%
存储利用率(%)	78.03	87.00	78.13	69.21
λ	106.95	141.92	60.81	10.94

结论与展望 上世纪 90 年代以来,研究人员对时态数据的索引技术进行了大量的研究,对时间属性的索引主要集中在树索引方面。我们在分析数据库时间属性特点的基础上,设计了一种新的基于时间属性的 Hash 索引——VH 索引。通过对其查询时间复杂度和数据库表的空间复杂度的理论分析以及实验验证,对其有效性和适用性得出了初步结论,但基于 VH 索引的时间区间相关查询及其优化方法仍需进一步研究。另外,VH 索引的聚簇特性使其适用于数据查询的并行处理,因此对于分布式应用环境下的 VH 索引的研究也很有意义。

参考文献

1 Kollios G, Tsotras J. Hashing Methods for Temporal Data. IEEE Transactions on Knowledge and Data Engineering, 2002, 14(4): 902~919

2 Jensen C S, Snodgrass T. Temporal Data Management. IEEE Transactions on Knowledge and Data Engineering, 1999, 11(1): 36~44

3 Salzberg B, Tsotras V J. A Comparison of Access Methods for Time-Evolving Data, ACM Computing Surveys, 1999, 31(2): 158~221

4 Litwin W, Neimat Ma, Donovan. LH *-A Scalable, Distributed Data Structure. ACM Transactions on Database Systems, 1996, 21(4): 480~525

5 Nϕrvag K. A study of object declustering strategies in parallel temporal object database systems, Information Science, 2002 (146): 1~27

6 Nascimento M A, Dunham M H. Indexing Valid Time Database via B+-Tree. IEEE Transactions on Knowledge and Data Engineering, 1999, 11(6): 929~947

7 Gunadhi H, Segev A. Efficient Indexing Methods for Temporal Relation. IEEETransactions on Knowledge and Data Engineering, 1993, 5(3): 496~509

8 Vram K. Temporal databases: Access structures, search methods, migration strategies, and declustering techniques: [Ph D dissertation]. Arlington: The University of Texas, 1994