维普资讯 http://www.cqvip.com

内存数据库中一种改进的图论存取方法及其性能分析*)

万常选[1,2] 刘云生[1]

(华中科技大学计算机科学与技术学院 武汉430074)[1] (江西财经大学信息管理学院 南昌330013)[2]

摘 要 内存数据库是支持高性能信息处理的一种方法,对于内存数据库而言,数据库的存储结构与存取方法是关键。本文给出了一种内存数据库组织与存取的改进图论方法,并对该方法给几个基本的查询处理操作所能带来的优势进行了探讨,最后从存储空间和操作执行时间两方面进行了性能分析,结果显示改进的图论方法能提供很好的性能。

关键词 内存数据库,数据库组织,内存数据库图,性能分析

Upgraded Graph-Theoretic Access Methods and Performance Analysis for Main Memory Databases

WAN Chang-Xuan[1,2] LIU Yun-Sheng[1]

(School of Computer Science & Technology, Huazhong University of Science & Technology, Wuhan 430074)[1]

(School of Information & Technology, Jiangxi University of Finance & Economics, Nanchang 330013)[2]

Abstract One approach to achieve high performance in a database management system is to store the database in main memory rather than on disk. For main memory databases, structure and access methods of databases are the key of a system. In this paper, a upgraded gragh-theoretic organization and access method of MMDBs are presented, and the advantages of several basic query operations on our method are discussed. Finally, the performance in storage cost and execution time of the operations are analyzed and evaluated carefully, the results show that our method provides better performance than other methods in storage cost and execution time of the operations.

Keywords Main memory database, Database organization, Main memory database graph, performance analysis

1 引言

数据库关系表一般都是以平淡文件(即表格)的方法进行存取,为了加快查询速度,再建立一些索引。文[2]中提出了一种以边附带属性(名)的对偶图的方式来存储内存数据库的新方法,并对基于对偶图的存取方法与表格的方法在存储空间、执行时间方面的性能进行了初步的对比分析。这些研究在提高内存数据库的空间利用率方面是非常有价值的。

所谓内存数据库,简单地说就是任何时刻任意一个活动事务所操作的数据集都要存放在内存中,换句话说,内存数据库系统就是数据库的"工作版本"常驻内存的数据库系统[1,2]。显然,它要求较大的内存量,至少应能存储当前处理的数据,但并不要求在任何时刻整个数据库都存放在内存。内存数据库是支持高性能信息处理的有力工具,是实现诸如实时数据库、智能数据库等的基础,其核心问题是数据库的存储结构及存取方法,以提高空间利用率。

内存数据库的图论存取方法就是将数据库 DB 的内容以内存数据库图(记为 MM-DBG)的方式进行存储。MM-DBG 是一个边带属性(名)标记的对偶图,即 MM-DBG$=G(T_{DB},V_{DB},L)$,其中 T_{DB},V_{DB}为 G 的两个顶点集,T_{DB}为元组顶点集,V_{DB}为属性值的顶点集;L 为其边集,边需附带属性(名)标记,设关系 R 的属性 A 记为 R_A,则对于附带属性 R_A 的边子集 $L_A\subseteq L$,有:$L_A=\{l|l_t\in T_{DB}\wedge l_v(V_{DB}\wedge(R_A(l_t)=l_v)\}$。其中,$l_t$,$l_v$ 分别表示边 l 所连接的顶点集 T_{DB},V_{DB}中的顶点,$R_A(l_t)$表示元组 l_t 在属性 R_A 上的取值。

图论存取方法的物理实现的基本原理是:①将存储空间分成变长的存储段(segment),每一值域的属性值及其索引成簇地单独存于一个片段中,每一个关系的所有元组也分别存于一个片段,片段中的每一存储对象赋以一个不变的唯一标识 OID(即存取指针),于是各元组和属性值都可通过其 OID 存取,为了便于区分,记元组 OID 为 TID;②元组顶点集 T_{DB}中每一个关系元组存储它的每个属性取值在值顶点集 V_{DB}中的 OID 序列,这种 OID 的顺序关系就反映了 t→v 的边所附带属性的含义;③值顶点集 V_{DB}中的每个值域中的值、域索引、以及域中每个值所对应的元组 TID 的倒排表等成簇地单独存于一个片断中;④由于一个值域可能对应于若干个属性在该域中取值,而在每一个属性上又可能有多个元组的取值对应于同一个值,因此倒排表的结构分两层来组织,第一层是一个指针矩阵(记为 M),第二层是倒排子表,倒排子表是 TID 系列的集合,它按属性进行分片段存储,假设该值域在数据库 DB 中是 n 个不同值的集合$\{v_i|i=1,2,\cdots,n\}$,在该值域取值的属性集合为$\{A_j|j=1,2,\cdots,m\}$,则指针矩阵 M 共有 $n\times m$ 个存储单元,其中单元 m_{ij}中存储的指针是指向倒排子表中的一个 TID 序列,该 TID 序列中存储了 DB 中属性 A_j 取值为 v_i 的所有元组的 TID;⑥由于键域的每个值只唯一对应一个元组,即键域的倒排子表中只包含一个元组 TID,因此无需倒排子表,直接将该元组 TID 存储在指针矩阵中;⑦域索引中仅包含访问该域索引关键字值的 OID 一个索引项;⑧查

*)本课题得到国家自然科学基金(编号60073045)和国防预研基金(编号00J15.3.3.JW0529)资助。**万常选** 博士生,教授,主要研究方向为现代数据库技术、Web 信息管理及电子商务技术。**刘云生** 教授,博导,主要研究方向为现代数据库理论与技术及其集成实现、数据库与信息系统开发。

维普资讯 http://www.cqvip.com

询操作的中间结果只存储一个元组 TID 矩阵和一个模式描述，其中元组 TID 矩阵中的每一行存储的是该结果关系该元组所对应的相关源关系元组的 TID 序列，而模式描述中存储的是结果关系的属性名以及该属性所对应的源关系名。图1给出了一个 MM-DBG 存储结构的例子，其中假设关系 S 的属性 A、B、D 和关系 T 的属性 E、G 的取值类型相同，因此它们的取值在同一值域。

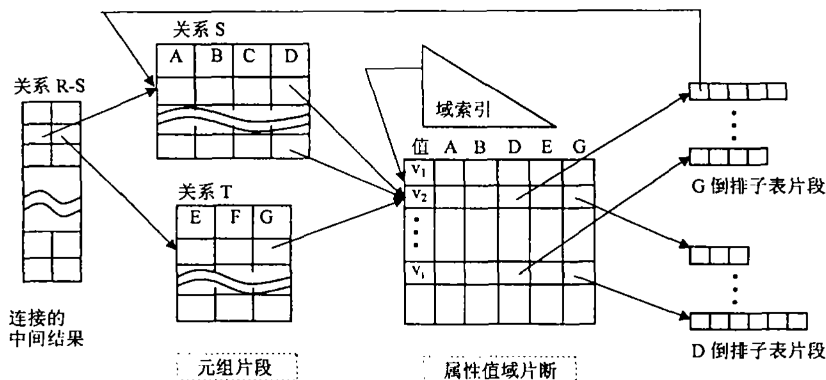

图1 一个 MM-DBG 物理实现的例子

MM-DBG 方法的主要目标是对 MMDB 系统提供紧凑的数据结构和高效的空间利用率。在传统的表格存储方法下，关系中的值是直接存储的，这就意味着重复的值也要重复存储，另外，为了获得高效的查询性能，还需要建立许多索引，因此索引关键字值还得重复在索引中存储一遍。MM-DBG 方法将元组与值分开存储，DB 中每一个可能取值都只存储一遍，相应要付出的代价是需要存储大量的 OID、TID 及指针，但总的存储代价可能会更低；同时对关系的处理是采用间接访问的方式进行的（即通过元组中存放的属性值的 OID 再去访问它的实际值），因此对总的执行效率可能会有点影响。

文[2]中得出的 MM-DBG 方法的主要不足之处是指针矩阵 M 的存储代价太大。为了进一步提高存储效率，本文提出一种改进的 MM-DBG 方法，记为 MM-DBUG 方法，并对它们在存储空间、执行时间方面的性能进行了进一步的深入分析。

2 改进的内存数据库图论存取方法及其查询操作

在 MM-DBG 方法中，将 DB 中所有关系的具有相同类型的属性的取值纳入到一个值域中，这样使得该值域所对应的指针矩阵 M 变得非常庞大，但事实上具有相同类型的不同属性的取值重复的概率是非常小的，因此付出（庞大的指针矩阵）与收效（重复的取值只需存储一遍值）相比是得不偿失的。本文的改进方法是：①在 DB 的所有关系中，除了类型相同且取值含义相同的属性（即意味着它们的取值有较大重复）的取值范围作为同一个值域外，其它每一个属性的取值范围均作为一个单独的值域；②对于关系的外键，在关系元组的外键域单元中不是存储它的取值在值域中的 OID，而是直接存储一个 TID，该 TID 直接指向所参照的另一关系中的键值所在元组，该 TID 的值是在元组增加或修改时通过查找定位找到的，因此称为预处理 JOIN 方法（即事先建立好 JOIN 连接的意思），这种存储策略能使得外键对键的 JOIN 操作变得非常简单和高效。图2给出了一个 MM-DBUG 存储结构的例子。其中属性 A、D 分别是关系 S、T 的（主）键，同时 D 又是关系 S 相对于关系 T 的外键。

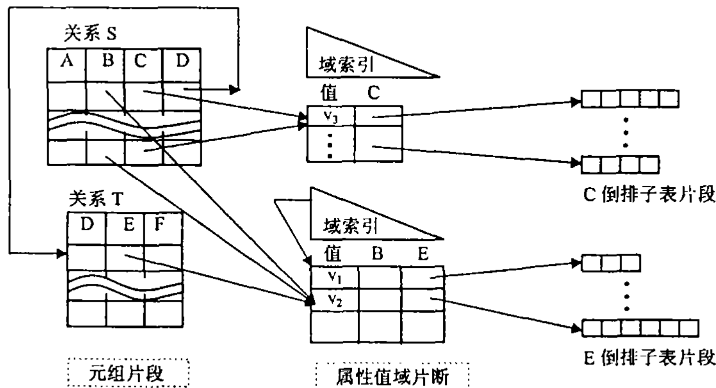

图2 一个 MM-DBUG 物理实现的例子

MM-DBUG 方法不仅比 MM-DBG 方法在存储空间方面有较大优势，而且在执行选择操作(σ_Q)、连接操作($\bowtie|_P$)时也有较大的性能改善。图论存取方法将索引与值域集成在一起存储，因此在执行选择操作或连接操作时，它并不是按选择条件 Q 或连接条件 P 在关系表上进行扫描，而是先在值域中进行配匹，对于满足条件的值再在倒排子表中按关联的 TID 去查找相应的元组。由于 MM-DBG 方法的值域太大（如果在一个值域中有 n 个取值不重复的属性，则值域的大小是 n 个属性分别取值范围的并集），它的执行性能会受到很大影响。而对于 MM-DBUG 方法，由于每一个值域中的值的个数不会大于相应关系中的元组个数（因为一个关系中除了键之外的属性的取值一般会有重复），因此会有较好的执行性能。

维普资讯 http://www.cqvip.com

对于连接操作,出现频率最大的是外键与键之间的等值连接,由于 MM-DBUG 方法采用的是预处理 JOIN 方法,因此它可以大大地提高系统在处理连接操作方面的性能。例如,假设有如下两个学生与院系的关系,以及在它们之上的两个查询要求。

学生关系:Students (sno, sname, sex, age, address, dep_no)

院系关系:Departments(dno, dname, phone)

查询1:显示所有所龄在18岁以下的学生的姓名、性别和所在院系名称。

查询2:显示计算机学院和信息学院的所有女同学的姓名和年龄。

对于查询1而言,首先执行的是对 Students 关系的一个关于年龄的选择操作,生成的中间结果按前面所述方法进行存储(即存储中间结果的元组 TID 矩阵和模式描述,这样可以节省存储空间和生成中间结果关系的执行时间);然后应该执行一个中间结果关系到 Departments 关系的连接操作并将查询结果进行输出,由于学生关系中的外键 dep_no 中直接存储了指向院系关系中所参照元组的 TID,因此,这个连接操作实际上并未进行,就可以直接将查询结果进行输出。

对于查询2而言,首先分别在 Students 和 Departments 关系上执行一个选择操作,生成的中间结果的存储方法同前;然后应该执行一个在两个中间结果关系上的连接操作,该操作有两种执行方案,一种是从 Departments 的中间结果关系开始来配匹 Students 的中间结果关系,由于并没有建立从前者到后者的指向联系,因此该执行方案还得进行连接操作,但是在连接过程中所需进行的比较是关于 TID 值的比较(即将前者的一个元组 TID 与后者的所有元组的 dep_no 值进行比较,相同者即是查询结果),它一般是更加高效的(特别是与字符串的比较而言);另一种执行方案是从 Students 的中间结果关系开始来配匹 Departments 的中间结果关系,根据前者的 dep_no 值与后者的元组指针矩阵中存储的 TID 进行比较,它仍然是一种 TID 值的比较。为了提高对中间结果关系的连接操作性能,可以对一个或两个中间结果关系建立临时索引(传统存储方法下也需要建立这种临时索引),以便能够使用更高效的连接操作算法。

对于不是外键与键之间等值连接的其它连接操作而言,正如前面所述,它的操作是在值域中进行配匹,由于值域已经建立了索引,并且值域中值的个数不会比相应关系的元组个数多,因此它的执行性能与传统存储方法是一个数量级的。对于元组属性没有重复取值或很少重复取值的值域而言,与传统存储方法相比,连接操作所需要的值的比较开销差不多,但是它要多增加从查找到的 TID 返找关系元组及再从关系元组中存储的 OID 找属性取值的开销,对于内存操作而言,这种开销是不大的;而对于元组属性有较大重复取值的值域而言,由于值域中值的个数比相应关系元组的个数要小得多,因此,与传统存储方法相比,在进行连接操作时可以节省一些值的比较时间,这种节省可以弥补需要增加的从 TID 找元组及从 OID 找值的开销。总之,对于不是外键与键之间等值连接的其它连接操作,总的性能也是在一个数量级的。上述分析对于选择操作亦是如此,详细的性能分析见第3部分。

由于图论存取方法的中间结果关系是一种描述性的,并没有真正生成,因此没有必要去先执行投影操作来减少中间结果的元组大小,也就是说,除非经过投影后可以使中间结果关系的元组重复有一个非常大的减少,否则投影操作将推迟到最后来执行。如果查询的目的仅仅是为了输出,即结果关系无需保存的话,则还可能不需要执行投影操作,只需直接根据存储的描述性结果关系将查询结果进行输出即可。在执行投影操作时才由描述性的关系变成真正的关系存储,由于所生成的结果关系中的数据在已有关系中都存在,因此只需生成结果关系元组,并在值域中的倒排子表中增加指向这些结果关系元组的 TID 即可。

对于在一个关系中增加一个元组的操作,一方面要增加一个元组来存储各属性取值的 OID;另一方面要将该元组所有属性的取值增加到相应值域中去(对于原值域中已经存在的值则无需增加),并增加该值指向元组的 TID。对于修改操作和删除操作也同样是非常简单的。

3 性能分析

通常,系统性能以其在存储空间和执行时间两个方面的代价来描述,这里我们也分别从空间和时间的开销两方面来对 MM-DBUG 方法、MM-DBG 方法以及传统的平淡文件(FF—Flat File)方法进行性能对比分析。

3.1 存储代价分析

假设 DB 中有 r 个关系,每个关系均有 N 个元组;每个关系可以有若干个键,这里假设每个关系各有一个键,且均为字符串类型,因此共有 r 个键;假设 r 个关系中共有 r_1 个外键,且每个键至多有一个以该键作为外键的关系,因此有 $r_1 \leqslant r$;在 r 个关系中,除键和外键之外,假设还有 a_1 个字符串类型的属性,a_2 个实数类型的属性,因此 r 个关系中共有 $r+r_1+a_1+a_2$ 个属性;假设字符串类型属性的取值的平均长度为 L 字节,实数类型的长度为4字节,每个指针、OID、TID 的长度为 m 字节;假设关系中除键和外键之外的属性取值的平均不重复率为 p,其中 $0<p\leqslant 1$,即100个元组中属性的值有 $100\cdot p$ 个不相同;对于 MM-DBUG 方法,假设除外键之外的每个属性单独一个值域;对于 MM-DBG 方法,假设除外键外不同属性之间的取值不重叠。记 MM-DBUG 方法、MM-DBG 方法的存储代价分别为 S_{UG}、S_G,则

$S_{UG}=(r+r_1+a_1+a_2)\cdot N\cdot m$ (关系表中 OID 或 TID 大小)

$+r\cdot N\cdot L+a_1\cdot(N\cdot p)\cdot L+a_2\cdot(N\cdot p)\cdot 4$ (值域中的数据域大小)

$+(r\cdot N\cdot 1+(a_1+a_2)\cdot(N\cdot p)\cdot 1)\cdot m$ (值域中的指针矩阵大小)

$+(r\cdot N+(a_1+a_2)\cdot(N\cdot p))\cdot m$ (所有域索引中 OID 大小)

$+(a_1+a_2)\cdot N\cdot m$ (所有倒排子表 TID 大小)

$=N((3r+r_1)m+2(a_1+a_2)(1+p)m+(r+a_1p)L+4a_2p)$

$S_G=(r+r_1+a_1+a_2)\cdot N\cdot m$ (关系表中 OID 大小)

$+(r\cdot N+a_1\cdot(N\cdot p))\cdot L+a_2\cdot(N\cdot p)\cdot 4$ (值域中数据域大小)

$+((r\cdot N+a_1\cdot(N\cdot p))\cdot(r+r_1+a_1)+a_2\cdot(N\cdot p)\cdot a_2)\cdot m$ (值域中指针矩阵大小)

$+\{[r\cdot N+a_1\cdot(N\cdot p)]+a_2\cdot(N\cdot p)\}\cdot m$ (所有域索引 OID 大小)

$+(r_1+a_1+a_2)\cdot N\cdot m$ (所有倒排子表 TID 大小)

$=N((r^2+2r+2r_1)m+2(a_1+a_2)m+r(r_1+a_1)m+(a_1^2+a_2^2+a_1+a_2+a_1r+a_1r_1)pm+(r+a_1p)L+4a_2p)$

几点说明:①对于 MM-DBUG 方法,有 r 个键的值域,每

维普资讯 http://www.cqvip.com

个键的值域有 N 个不同值；其余 a_1+a_2个属性分别对应一个值域，且每个值域有$(N\cdot p)$个不同值；每一个值域的指针矩阵均为1列。②对于 MM-DBG 方法，有两个值域，一个是字符串值域，共有 $r\cdot N+a_1\cdot(N\cdot p)$个不同值，该值域的指针矩阵有 $r+r_1+a_1$列；另一个是实数值域，共有 $a_2\cdot(N\cdot p)$个不同值，该值域的指针矩阵有 a_2列。

假设在传统的 FF 方法下按单个属性建立索引，每个键属性建立一个索引，除键之外的其它属性建立索引的比例为 q，其中$0\leqslant p\leqslant 1$，q=1时表示关于每一个属性都建立了索引；每个索引为按索引键值 v 排序的对偶(v,TID)的阵列。记 FF 方法的存储代价为 S_{FF}，则

$S_{FF}=(r+r_1+a_1)\cdot N\cdot L+a_2\cdot N\cdot 4$ （关系表中元组数据大小）

$+(r+(r_1+a_1)\cdot q)\cdot N\cdot L+(a_2\cdot q)\cdot N\cdot 4$ （所有索引中索引键数据大小）

$+(r+(r_1+a_1+a_2)\cdot q)\cdot N\cdot m$ （所有索引中 TID 大小）

$=N(rm+(r_1+a_1+a_2)qm+(2r+r_1+a_1+r_1q+a_1q)L+4a_2(1+q))$。

因此，$S_G-S_{UG}=Nm(r^2-r+r_1+r(r_1+a_1)+(a_1^2+a_2^2-a_1-a_2+a_1r+a_1r_1)p)$

由上式可知，改进的图存取方法比原来的图存取方法可以节省大量的存储空间。

假设:DB 中有5个关系，即 r=5；$r_1=3$，$a_1=10$，$a_2=10$，即共28个属性，平均每个关系有5.6个属性；p=0.7，m=2，L=8，N=1000。则 $S_G=836000$字节，$S_{UG}=296000$字节，$S_{FF}=329000$字节(q=0.5时)，$S_{FF}=424000$字节(q=1时)。$S_{UG}/S_G\approx 35.41\%$，即改进的图存取方法所占用存储空间只相当于原来的图存取方法的35.41%。

3.2 时间代价分析

由于一个查询最终是由相应关系代数操作来实现，因此只需对三种基本关系操作执行的时间代价进行分析。

假设关系 R 中共有 N 个元组，m 个属性（均为同一类型），属性取值的不重复率为 p，对于键属性有 p=1；假设在内存中定位一个索引所需 CPU 时间为 d；在 CPU 中执行一次两个值的比较所需时间分别为 c_1（两个指针值的比较）和 c_2（两个属性值的比较），典型地有 $c_1\ll c_2$，特别是对于字符串数据更是如此；根据地址指针进行一次内存读数据操作所需 CPU 时间分别为 a_1（访问一个指针值）和 a_2（访问一个属性的数据值），典型地有 $a_1\ll a_2$，特别是对于字符串数据更是如此，同时有 $a_1\ll c_1$，$a_2\ll c_2$；假设对索引的查找采用简单的二分查找法，对值域中的指针矩阵中的某行的查找采用顺序查找法。记 MM-DBUG 方法、MM-DBG 方法和 FF 方法的时间代价分别为 T_{UG}、T_G 和 T_{FF}。

3.2.1 选择操作的时间代价分析

仅考虑最简单的选择操作谓词 $Q=A\theta C$，其中 A 为属性名，C 为常量，θ 为等值比较。

第一步，在索引中进行查找定位的时间代价分析。三种方法的时间代价分别为：

$$T_{UG}=(\frac{N\cdot p+1}{N\cdot p}\log_2(N\cdot p+1)-1)\cdot(a_1+a_2+c_2)$$

$$T_G=(\frac{m\cdot N\cdot p+1}{m\cdot N\cdot p}\log_2(m\cdot N\cdot p+1)-1)\cdot(a_1+a_2+c_2)$$

$$T_{FF}=(\frac{N+1}{N}\log_2(N+1)-1)\cdot(a_2+c_2)$$

其中，对于图方法，每一次查找需先访问一次索引中的 OID 值，再根据此 OID 访问一次属性值，最后再将该值与 C 进行一次比较；对于 FF 方法，每一次查找需要在索引中访问一次属性值，并将该值与C 进行一次比较。

第二步，访问并处理结果元组。对于 FF 方法，只需根据索引中查找到的元组地址进行元组内容的访问并进行结果处理（如显示输出结果或写结果关系，假设结果处理所需时间为 S）；而对于图论方法则首先需要根据在值域的指针矩阵和倒排子表中进行元组定位并根据元组进行属性值定位，然后才能进行元组内容的访问。因此，三种方法的时间代价分别为：

$$T_{UG}=(1+\frac{1}{p}-|p|+\frac{1}{p}\cdot m)\cdot a_1+\frac{1}{p}\cdot m\cdot a_2+S$$

$$T_G=(\frac{m+1}{2}+\frac{1}{p}-|p|+\frac{1}{p}\cdot m)\cdot a_1+\frac{1}{p}\cdot m\cdot a_2+S$$

$$T_{FF}=\frac{1}{p}\cdot m\cdot a_2+S$$

几点说明：①由于属性取值的不重复率为 p，因此每次选择操作平均可获得$1/p$ 个结果元组；②$|p|$为对 p 取整，对于键属性它的值为1，对于其它属性它的值为0；③对于 MM-DBUG 方法，在指针矩阵的访问需要1次，以获得倒排子表指针（对于键属性则获得的是元组 TID）；对于非键属性还需在倒排子表中访问$1/p$ 次，以获得$1/p$ 个元组 TID；④对于 MM-DBG 方法，在指针矩阵中的平均访问次数是$(m+1)/2$次，以获得属性 A 所对应的倒排子表指针（对于键属性则获得的是元组 TID）；对于非键属性还需在倒排子表中访问$1/p$次，以获得$1/p$ 个元组 TID；⑤由于每个元组有 m 个属性，因此对于图方法，还需要在每个元组中访问 m 次以获得 m 个属性值的指针。

综上所述，三种方法对执行选择操作所需总的时间代价分别为：

$$T_{UG}=d+(\frac{pN+1}{pN}\log_2(pN+1)-1)(a_1+a_2+c_2)+(1-|p|+\frac{1+m}{p})a_1+\frac{ma_2}{p}+S$$

$$T_G=d+(\frac{mpN+1}{mpN}\log_2(mpN+1)-1)(a_1+a_2+c_2)+(\frac{m+1}{2}-|p|+\frac{1+m}{p})a_1+\frac{ma_2}{p}+S$$

$$T_{FF}=d+(\frac{N+1}{N}\log_2(N+1)-1)(a_2+c_2)+\frac{ma_2}{p}+S$$

由此可知，MM-DBUG 方法与 FF 方法相比，在执行选择操作时，一方面是多了一块访问指针(OID)的时间$(1-|p|+(1+m)/p)a_1$，它是与元组数无关的；另一方面是在索引中的每次查找定位多了一次指针的访问时间 a_1；相应地，对于非键属性的索引空间会更小，从而可以节省在索引中的查找定位次数。显然三种方法的时间复杂度均为 $O(\log_2 N)$。假设 N=1000，m=6，p=0.7，$a_2=4\cdot a_1$，$c_2=8\cdot c_1$，$c_1=4\cdot a_1$。则上述三式分别变为：

$$T_{UG}=d+358.56\cdot a_1+S$$

$$T_G=d+456.24\cdot a_1+S$$

$$T_{FF}=d+357.46\cdot a_1+S$$

由上面三个式子可以看出，它们的性能差距是很小的。这里还没有考虑图方法对于中间结果的描述性存储所节省的时间。

如果选择操作的属性是外键，对于 MM-DBG 方法与 FF 方法而言，与在其它属性上的选择操作完全相同；但是对 MM-DBUG 方法的执行性能有较大影响。不过幸运的是，由于键一般都是代码化的，它并没有直接的含义，而外键主要是

（下转第174页）

维普资讯 http://www.cqvip.com

另外，将底层设备驱动程序替换成能够访问其他处理器(如 Intel Pentium 4等)的硬件计数器的驱动程序，并对接口处作相应的修改，可以使 PTracker 能够适用于这些处理器。

附录 A 本文用到的 PAPI 参数的含义

PAPI_L1_DCA	L1数据 Cache 访问数
PAPI_L1_DCM	L1数据 Cache 不命中数
PAPI_L2_TCA	L2 Cache 总的访问数
PAPI_L2_TCM	L2 Cache 总的不命中数
PAPI_BR_CN	所执行的条件分支指令数
PAPI_BR_MSP	条件分支指令预测错误数
PAPI_FP_INS	所执行的浮点指令数

参考文献

1 Ghosh S,et al. Cache Miss Equations: A Compiler Framework for Analyzing and Tuning Memory Behavior. In ACM Transactions on Programming Languages and Systems,1999,21(4):702～745
2 http://www.cs.wisc.edu/～mscalar/simplescalar.html
3 Merten M C,et al. An Architectural Framework for Run-Time Optimization. IEEE Transactions on Computers,2001,50(6):567～589
4 Lambert, et al. Profiling I/O Interrupts in Modern Architectures. In: 8th Intl. Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, San Francisco, California,2000
5 Hirzel M, et al. Bursty Tracing: A Framework for Low-Overhead Temporal Profiling. In: 4th Workshop on Feedback-Directed and Dynamic Optimization (FDDO), Dec. 2001
6 http://icl.cs.utk.edu/projects/papi/
7 http://www.gz-juelich.de/zam/PCL/
8 http://research.compaq.com/SRC/dcpi/
9 http://developer.intel.com/vtune/
10 IA-32 Intel® Architecture Software Developer's Manual: Volume 3: System Programming Guide. Intel Corporation. 2002
11 http://www.sdsc.edu/PMaC/Benchmark/MAPS/
12 Zeyao M, Zhang Baolin. Multilevel averaging weight method for load imbalance problems. International J. Comp. Math, 2000, 77 (3/4)
13 Browne S, et al. A Portable Programming Interface for Performance Evaluation on Modern Processors. The International Journal of High Performance Computing Applications, 2000, 14 (3):189～204
14 Ding C, Kennedy K. Memory bandwidth bottleneck and its amelioration by a compiler. In: Proc. of the 2000 Intl. Parallel and Distributed Processing Symposium, Cancun, Mexico, May, 2000
15 Carr S. Combining Optimization for Cache and Instruction-Level Parallelism. In: Proc. of PACT '96, Boston, USA, Oct. 1996
16 Bailey D H. Performance Metrics: Out of the Dark Ages. In: High-Speed Computing Conf. Gleneden, OR, Apr. 2001

(上接第83页)

用来进行参照用的，因此在实际应用中关于外键的选择操作是基本上不会发生的。对 MM-DBUG 方法，如果要执行外键上的选择操作，一种可能的执行方案是：直接在外键所在关系中进行选择操作，即对于每一个元组，首先通过外键中存储的 TID 值去间接访问外键的真正取值，再与 C 进行比较；为了提高执行性能(如果有必要的话)，还可以关于外键域专门建立一个索引。

3.2.2 *连接操作的时间代价分析* 与选择操作的分析类似，三种方法的时间代价是同一数量级的。对于 FF 方法，文[3]对几种索引形式下的连接操作算法的性能进行了分析，结论是：①如果两个连接的关系 R_1 和 R_2(设 R_2 为内连接关系)在连接属性上都有索引，则 Tree Merge 方法有最好的性能，此时如果 T Tree 索引也存在的话，则通过扫描该索引还可以进一步减少连接过程中的比较次数，比较的次数大致为($|R_1|+2\cdot|R_2|$)。如果连接属性取重复值的话，则比较的次数会增加。②如果两个连接的关系上只有一个索引(设内连接关系上有索引)，则 Hash Join 方法有最好的性能，在一个 Hash 表上查找一个值，有一个固定的代价(记为 k)，它与索引的大小无关，Hash Join 方法的比较次数大致为($|R_1|+k\cdot|R_2|$)，其中 $2<k<\log_2(|R_2|)$。

但是需要指出的是，在 MM-DBUG 方法下，对于外键与键之间的等值连接操作基本上是无时间代价的(因为它不需要执行任何比较)，而在实际应用中，绝大部分的连接操作都是外键与键之间的等值连接操作，因此，MM-DBUG 方法对于连接操作的执行性能是明显优于其它方法的。

对于投影操作，在第2部分已经进行过分析，这里就不重复了。

结束语 本文对文[2]中提出的内存数据库的图论存取方法进行了改进，改进后的图论存取方法不仅可以大大节省存储空间，而且对执行性能也有较大提高，特别是对于连接操作的执行性能会有很大的提高。本文还在文[2]的基础上，对几种存取方法在存储空间、执行时间方面的性能进行了进一步的深入分析，分析结果表明改进图论方法是更加有效的。

下一步的工作主要在内存数据库的并发控制、索引结构、查询算法、优化策略等方面。不过对于 MM-DBS 而言，查询优化将比磁盘数据库要简单得多，至少集簇问题以及通过投影来减少元组大小的问题可以不必考虑，因此也可以简化对算法的选择。另一方面，并发控制可能又会引发新的矛盾，由于在 MM-DBS 中所有的数据操作都是在内存中，这样相比之下锁的代价就会更大，例如，如果在元组粒度加锁的话，加锁的代价将与存取的代价相当，这样总的代价就是存取代价的2倍了，这太昂贵了。因此，应该在更大粒度的数据上加锁，这样可能更合算，相应地 MM-DBS 下的事务可能会更短(因为没有磁盘访问了)，因此锁数据的时间也就会更短。另外，如果整个 DB 不能全部装入内存的话，这时就还有一个内存与外存交换数据的策略与算法的问题。最后，内存数据库的恢复也是一个重要的问题，它与磁盘数据库的恢复策略与算法是相差很远的，今后我们将致力于这些问题的研究。

参考文献

1 Liu Yun-Sheng, et al. Data organization and management of real-time main memory databases. Journal of Computer Research & Development, 1998, 35(5): 469～473 (in Chinese)
2 Liu Yun-Sheng, et al. Graph-Theoretic Access Methods for Main Memory Databases. Chinese Journal of Computers, 2001, 24 (10): 1095～1101 (in Chinese)
3 Lehman T J, Carey M J. Query Processing in Main Memory Database Management Systems. In: Proc of the ACM SIGMOD Int'l Conf on Management of Data, Washington, D.C., May 1986. 239～250
4 Ammann A C, et al. Design of a Memory Resident DBMS. In: Proc of the IEEE Compcon, San Francisco, 1985. 54～57
5 DeWitt D J, et al. Implementation Techniques for Main Memory Database Systems. In: Proc of the ACM SIGMOD Annual Meeting, Boston, Massachusetts, June 1984. 1～8
6 Bitton D, et al. Performance of Complex Queries in Main Memory Database Systems. In: Proc of the Third int'l Conf on Data Engineering, Los Angeles, California, Feb. 1987. 72～81