一种主存数据库的存取方法*

刘云生　　胡国玲

（华中理工大学计算机系　武汉　430074）

摘要　本文给出了一个主存数据库管理系统 ARTs－MMDBS，着重讨论了它的数据库管理机制与存取方法，并具体阐述了它的一种新的索引结构—SB 树.

关键词　主存数据库，存取方法，索引结构，数据库组织，SB 树.

主存数据库系统 MMDBS(main memory database system)是指“当前”数据库或数据库的“工作版本”常驻主存的数据库系统. 它能够对实时数据库系统的实现提供有力的支持.

实时数据库系统是事务和数据都可以具有定时特征或显式定时限制的数据库系统. 系统的正确性不仅依赖于逻辑结果，而且依赖于逻辑结果产生的时间.[1]如果将实时数据库建立在主存数据库基础上，那么在事务运行过程中不仅可以消除内外存间的数据 I/O，为系统准确估算事务的运行时间，从而使之具有一定的动态可预报性提供有力支持，而且也可为满足实时事务的定时限制打下基础.

由于存储介质性能的不同，MMDBS 和通常的磁盘数据库系统(DRDBS)在数据空间的组织和管理、数据的存储结构和存取策略等方面存在很大差异，这些也就是主存数据库系统需要研究的主要方面. 本文就我们研制的一个支持主动实时数据库的主存数据库管理系统 ARTs－MMDBS 对这些问题进行讨论.

1　ARTs－MMDBS 的存取策略

采用灵活而高效的数据存储管理机制是实现主存数据库管理系统的关键. 由于主存可以直接被 CPU 访问，因此主存数据库管理系统应该开发比常规的数据库管理系统更为灵活高效的数据空间组织管理机制.

1.1　ARTs－MMDBS 的数据组织

ARTs－MMDBS 采用如图 1 所示的数据空间组织管理机制. 主存数据库 MMDB 逻辑上由若干分区组成，一个分区用来存放一个关系的数据；分区物理地由若干段组成，一个段是主存中一块固定长度的连续区域，它是内、外存 I/O 的单位，也是内、外存分配和回收及

---

* 本文研究得到国家自然科学基金与国防预研资助. 作者**刘云生**，1940 年生，教授，主要研究领域为现代数据库与信息系统，实时计算，软件开发方法学. **胡国玲**，女，1967 年生，博士生，主要研究领域为现代数据库技术与信息系统，实时计算，主动规则系统.

本文通讯联系人：刘云生，武汉 430074，华中理工大学计算机系

本文 1995－11－06 收到修改稿

对 MMDB 进行恢复的单位.

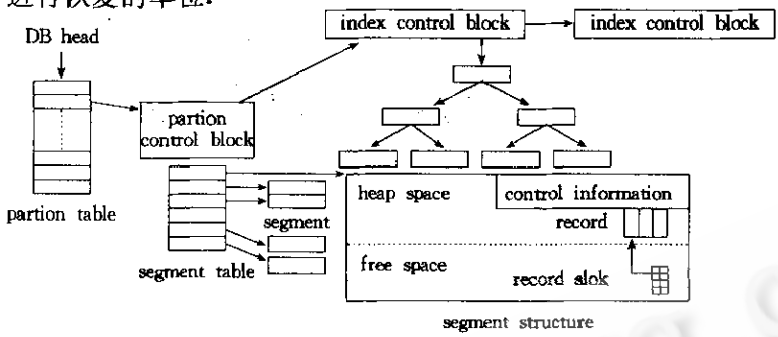

图1 ARTs－MMDBS的数据空间组织管理机制

每个记录都有一个唯一的标识符 RID，一个 RID 为一个三元组$\langle P,S,L\rangle$. 其中 P,S,L 分别为分区号、段号、段内的记录槽号. 这样记录可以在段内自由移动而其 RID 不变. 记录槽的结构如图 2 所示. 其中记录槽内的指针直接指向主存地址.

1.2 ARTs－MMDBS 的数据存取

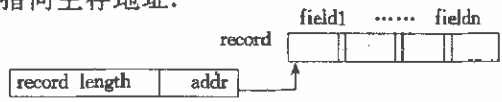

图2 记录槽的结构

存取一个记录可由下列算法完成：

FUNC ACCEMD(RID)

输入：记录的 RID

输出：存放记录值的缓冲区 buf

(1) 根据记录的 RID 二分查找分区表和段表得到该记录所在段 s 的地址；

(2) 在 s 中根据 RID 中的记录槽号直接定位记录槽；

(3) 根据记录槽中指向记录值的指针及记录长度，将记录值复制到 buf 中.

设分区表和段表的长度分别为 PL 和 SL，则由上述算法易知其时间复杂度为：

$$O(\log_2(PL))+O(\log_2(SL))$$

2 索引结构

通常的 B^+ 树对基于磁盘的数据库能提供方便快速的存取，但对于主存空间极为宝贵的 $MMDBS$ 而言它不太适用. 研究表明，B^+ 树中每个结点的覆盖率仅为 55%[2]，其存储效率太低. 主存数据库应该开发适合于数据的直接地址访问特征的存取方法；且与传统数据库所考虑的相反，在处理“时”、“空”关系时，空间应为第一位. 实验证明，在整个树结点或树结点的 80% 都置于主存时，AVL 树比 B^+ 树能提供更好的存取性能.[3]但 AVL 树每个结点只存储一个元素，故每个元素都带有 2 个指针的附加信息，因此它虽有存取效率高的特点但其存储效率仍很低. 为此，我们设计了一种新的索引结构——“顺序二分”(SB)树，它可看作是 AVL 树的一个变种. 每个结点包含了多个顺序存储的元素.

2.1 SB 树的结构

定义 2.1 (标识偏序关系). 设 $D=\{RID_i \mid 1\leqslant i\leqslant n\}$ 为一个记录标识符有限集，$KEY(RID_i)$ 为 RID_i 所对应的记录关键字值. $\propto$ 为 D 上的一个标识偏序关系是指 $\forall\ RID_i, RID_j\in D, RID_i\neq RID_j, RID_i\propto RID_j\ iff\ KEY(RID_i)<KEY(RID_j)$.

定义 2.2 (SB 树). SB 树是一种树结构：$SB-tree=(D,E)$. 其中 D 为 SB 树的结点

集,E 为其边集. 若 $D=\Phi$, 则 $E=\Phi$, 称其为空 SB 树;若 $D\neq\Phi$,则 E 是 D 上的一个"父-子"关系 H 的集合,即 $E=\{H\}$且:

(1) D 中存在唯一的称为根的结点 r,它在关系 H 下无父辈;

(2) 若 $D-\{r\}\neq\Phi$,则 $D-\{r\}=\{D_L^r,D_R^r\}$且 $D_L^r\cap D_R^r=\Phi$;

(3) 若 $D_L^r\neq\Phi$,则在 D_L^r 中存在唯一的结点 x_L^r,$\langle r,x_L^r\rangle\in E$ 且存在 D_L^r 上的边集 $E_L^r\subset E$;若 $D_R^r\neq\Phi$,则在 D_R^r 中存在唯一的结点 x_R^r,$\langle r,x_R^r\rangle\in E$ 且存在 D_R^r 的边集 $E_R^r\subset E$;即 $E=\{(r,x_L^r),(r,x_R^r),E_L^r,E_R^r\}$;

(4) (D_L^r,E_L^r),(D_R^r,E_R^r)是符合本定义的 SB 树,分别称为根 r 的左子树和右子树,它们的根分别为 x_L^r 和 x_R^r.

(5) $|h_L^r-h_R^r|\leqslant 1$(h_L^r 和 h_R^r 分别为 r 的左、右子树高度);

(6) 每一结点 N 由结点头(控制信息)和 $2n+1$ 个索引项 2 部分组成:

$$(bf,lcount,rcount,lp,rp;\ e_1,e_2,\ldots,e_n,e_{n+1},\ldots,e_{2n+1}).$$

其中 $bf=h_L^N-h_R^N$. 称为 N 的平衡因子;lp 和 rp 分别为指向 N 的左子树和右子树的指针;$lcount$ 为 N 中左半部 $e_i(1\leqslant i\leqslant n)$的个数;$rcount$ 为 N 中右半部 $e_i(n+2\leqslant i\leqslant 2n+1)$的个数;元素 e_i 就是 $RID_i(i=1,2,\ldots,2n+1)$,它们构成一个标识偏序集.

若 D_L^N 和 D_R^N 均为空,即 N 为叶结点,则 $1\leqslant(lcount+rcount+1)\leqslant(2n+1)$,否则($N$ 为非叶结点)有 $2n-1\leqslant(lcount+rcount+1)\leqslant 2n+1$;

(7) 若 $D_L^N\neq\Phi$ 且 $D_R^N\neq\Phi$,设 $maxkey_L^N$ 为 N 的左子树中所有 e_i 所指向的记录中最大的关键字值所对应的元素,其对应的结点为 $maxe_L^N$,称为 N 的最大下界结点;$minkey_R^N$ 为 N 的右子树中所有 e_i 所指向的记录中最小的关键字值所对应的元素,其对应的结点为 $mine_R^N$,称为 N 的最小上界结点,则

$$KEY(maxkey_L^N)<KEY(e_{n+1-lcount}),\ KEY(e_{n+1+rcount})<KEY(minkey_R^N)$$

从定义中可以看出:SB 树的结点只存储了记录的 RID,通过 RID 调用算法 $ACCEMD$ 便可得到记录值,从而得到其关键字值,这样节省了存储关键字副本的空间;用定长的、占用字节少的 RID 作为索引项,使 SB 树更简单、更紧凑.

SB 树的结构及其每个结点的结构如图 3 和图 4 所示:

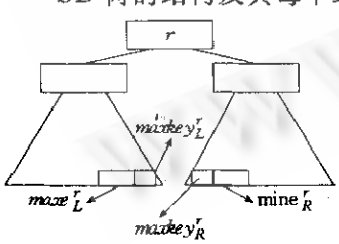

图3 SB树的结构

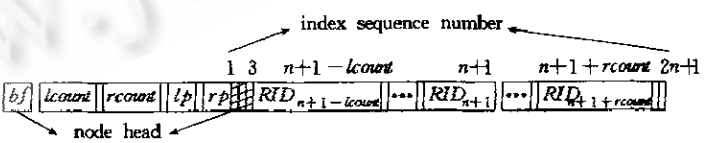

图4 SB树的结点结构

2.2 SB 树的维护

下面用类 $PASCAL$ 语言来描述 SB 树的各种操作算法.

2.2.1 查　找

检索操作从 SB 树的根结点开始,首先调用存取方法找出被检索结点所对应的最大关键字值和最小关键字值,如果检索值小于最小关键字值,则在其左子树中查找;如果大于最

大关键字值，则在其右子树中查找；否则在当前结点中以二分法查找，若找不到所要检索的值则查找失败.

```
FUNC SB_search(sb_root,keyi): RID
  {输入：sb_root—SB 树的根；keyi —记录的关键字值；输出：成功则输出被检索记录的 RID；失败则输出－1.}
v:=sb_root; x:=NIL;
WHILE v≠NIL DO
  【minkey:=getkey(ACCEMD(v↑.e[n+1−v↑.lcount])); {调用算法 ACCEMD 得到 v↑.e[n+1−v↑.lcount]所对
    应的记录值，然后取出其关键字值放入 minkey 中}
    maxkey:=getkey(ACCEMD(v↑.e[n+1+v↑.rcount]));
    CASE
      keyi < minkey: v:=v↑.lp;
      keyi > maxkey: v:=v↑.rp;
      minkey < keyi < maxkey:
        【x:=bin_search(keyi,v); {在结点 v 中二分查找 keyi 所对应记录的 RID}
          IF x≠NIL THEN RETURN(x); ELSE RETURN(−1); 】】
RETURN(−1)
ENDFUNC
```

2.2.2 插 入

插入算法首先查找插入结点，若找到则插入(此时若该结点已满，则挤出其最大或最小元素，该元素再作一个新的插入)；若找不到则生成一个新的结点，将索引项插入，然后转入树的平衡性检查. 每当对一个结点进行插入时，都自中心点开始，分别向左右两边找到其适当位置而插入.

```
FUNC ins_SB_tree(t,RIDi):integer
  {输入：t —SB 树的根；RIDi—记录标识符；输出：0—插入成功；−1—插入失败；}
IF t=NIL THEN
  【new(s); s↑.e[n+1]:=RIDi; s↑.lp:=NIL; s↑.rp:=NIL; s↑.rcount:=0; s↑.lcount:=0; s↑.bf:=0; t:=s;
  RETURN(0)】
ELSE {查找 RIDi 的插入位置后插入并记下 a 以便检查树的平衡性}
  【f:=NIL; a:=t; p:=t; q:=NIL;
    WHILE p≠NIL DO
    【IF p↑.bf≠0 THEN【 a:=p; f:=q】
      q:=p;
      keyi:= getkey(ACCEMD(RIDi));
      keymin:=getkey(ACCEMD(p↑.e[n+1−p↑.lcount]));
      keymax:=getkey(ACCEMD(p↑.e[n+1+p↑.rcount]));
      CASE
      keyi<keymin: p:=p↑.lp;
      keyi>keymax: p:=p↑.rp;
      (keymin<keyi<keymax) AND ((p↑.lcount+p↑.rcount+1)<(2n+1));
    【loc:=sbin_search(RIDi,p); {在结点 p 中二分查找 RIDi 所应插入的索引顺序号} ins_node(RIDi,p,loc); {在 p 中
    的 loc 处插入 RIDi，必要时还需向左或右移动记录} RETURN(0); 】
      (keymin<keyi<keymax) AND ((p↑.lcount+p↑.rcount+1)=(2n+1));
      【loc:=sbin_search(RIDi,p); {loc 为 RIDi 应插入的索引顺序号}
        IF loc>(n+1) THEN
          【tmprid:=p↑.e[2n+1];
          FOR k:=2n+1 DOWNTO (loc+1) DO p↑.e[k]:=p↑.e[k−1];
          p↑.e[loc]:=RIDi; ins_SB_tree(p↑.rp,tmprid); RETURN(0);】
        ELSE {loc≤(n+1)}
            【tmprid:=p↑.e[1];
            FOR k=1 TO (loc−1) DO p↑.e[k]:=p↑.e[k+1];
            p↑.e[loc]:=RIDi; ins_SB_tree(p↑.lp,tmprid); RETURN(0);】】】】
IF (q↑.lcount+q↑.rcount+1)<(2n+1) {q 为叶结点}
```

【$loc:=sbin\_search(RID_i,p)$；$ins\_node(RID_i,p,loc)$；RETURN(0)；】
ELSE
【$new(s)$；$s\uparrow.e[n+1]:=RID_i$；$s\uparrow.lp:=NIL$；$s\uparrow.rp:=NIL$；$s\uparrow.rcount:=0$；$s\uparrow.lcount:=0$；$s\uparrow.bf:=0$；
$upd\_bf(a,q)$；{从 a 到 q 修改平衡因子}
$b:=is\_bal(a)$；{检查以 a 为根的子树的平衡性}
IF b=FALSE $rotate(a)$；{进行旋转操作} RETURN(0)；】
ENDFUNC.

2.2.3　删　除

输入：RID_i—记录标识符；

输出：0—执行成功；−1—执行失败.

(1)查找 RID_i 所在结点.

(2)若找不到则报告一个错误并停止操作，否则设它在结点 q 中，则在 q 中删除 RID_i.

(3)如果 q 为叶结点：如果删除 RID_i 后 q 中元素个数为 0，则释放该结点然后转(5). 否则算法结束.

(4)若 q 为非叶结点：如果删除 RID_i 之后($q\uparrow.lcount+q\uparrow.rcount+1$)≥($2n-1$)则算法结束. 否则：

(Ⅰ)如果 $q\uparrow.bf=1$：则从 $maxe\ell$ 中移出 $maxkey\ell$ 到 q 中，若移出后 $maxe\ell$ 中元素个数为零，则释放该结点，然后转(5). 否则算法结束.

(Ⅱ)如果 $q\uparrow.bf=-1$：则从 $mine_k$ 中移出 $minkey_k$ 到 q 中，若移出后 $mine_k$ 中元素个数为零，则释放该结点，然后转(5). 否则算法结束.

(Ⅲ) 如果 $q\uparrow.bf=0$ 则

(a)如果($maxe\ell\uparrow.lcount+maxe\ell\uparrow.rcount+1$)＞($mine_k\uparrow.lcount+mine_k\uparrow.rcount+1$)，则从 $mine_k$ 移出 $minkey_k$ 到 q 中，若移出后 $mine_k$ 中元素个数为零，则释放该结点，然后转(5). 否则算法结束.

(b)如果($maxe\ell.\uparrow lcount+maxe\ell\uparrow.rcount+1$)≤($mine_k.\uparrow lcount+mine_k\uparrow.rcount+1$)，则从 $maxe\ell$ 移出 $maxkey\ell$ 到 q 中，若移出后 $maxe\ell$ 中元素个数为零，则释放该结点，然后转(5). 否则算法结束.

(5)重新计算结点的平衡因子，进行树的平衡性检查，如果不平衡，则进行旋转.

2.2.4　SB 树的旋转

SB 树的旋转有 LL，LR，RL，RR 4 种，与 AVL 树相同，在此略述.

2.3 性能分析

由上述定义及算法描述我们可以看出：SB 树吸收了 AVL 树和 B 树 2 者的优点，其非叶结点最多只有 2 个元素的空闲空间(之所以留出 2 个元素的空闲空间是为了大大减少 SB 树旋转操作的次数)，因此它即具有较高的空间利用率又具有较高的存取效率.

定理 1. 具有 m 个索引项(RID)的 SB 树的最大高度 h 为：

$$h=\log_\varphi(m+2n-1)+\log_\varphi\sqrt{5}-\log_\varphi[(2n-1)\times\varphi^2-(2n-2)](\varphi=(1+\sqrt{5})/2) \quad (1)$$

证明：设 N_h 为高度为 h 的 SB 树中含有的最少结点数(反之，h 为由 N_h 个结点所能构成的 SB 树的最大高度)，其中叶结点数有 L_h 个，非叶结点数有 NL_h 个，由定义 2.2，我们有

$N_0=0$，$N_1=1$，$N_2=2$，$N_h=N_{h-1}+N_{h-2}+1$ ($h\geqslant 3$)；

$L_0=0$，$L_1=1$，$L_h=L_{h-1}+L_{h-2}$($h\geqslant 2$)

$N_h=L_h+NL_h$

显然 L_h 为一个斐波那契序列(*Fabonacci numbers*),即 $L_h=F_h$. 用归纳法易证:当 $h\geqslant 0$ 时有:$N_h=F_{h+2}-1$. 即 $L_h+NL_h=F_{h+2}-1$,则 $NL_h=F_{h+2}-L_h-1$,而 $L_h=F_h$,则 $NL_h=F_{h+2}-F_h-1$. 又由定义 2.2,每个叶结点中至少有 1 个索引项,每个非叶结点至少有$(2n-1)$个索引项,显然应有

$$F_h+(2n-1)*(F_{h+2}-F_h-1)=m \tag{2}$$

已知 $F_h\approx\varphi^h/\sqrt{5}$ [4],将式(2)两边取对数即得(1),定理得证.

定理 2. SB 树的查找算法在最坏情况下的时间复杂度为:

$$O(\log_\varphi m)+O(\log_2(n))*[O(\log_2(SL))+O(\log_2(PL))]$$

其中 SL 为分区表的长度,PL 为段表的长度.

证明:由定理 1 及算法 *SB_search* 可知本定理成立.

3 结束语

本文以主存数据库管理系统 ARTs—MMDBS 为例就主存数据库的数据空间的组织管理、索引及存储结构等主要方面进行了具体的探讨,其主要工作在于:给出了一种 MMDB 的体系结构和数据空间的组织管理机制;提出了一种适合 MMDB 的新的索引结构—SB 树,给出了它的形式定义并具体描述了它的各种操作算法.

主存数据库是目前数据库领域研究的一个热点,已取得了若干可喜的成果,但因其与磁盘数据库有不同特性与设计目标,仍有很多问题需要研究解决.

参考文献

1 刘云生,卢炎生. 实时数据库的特征及其与主动数据库的联系. 计算机工程与应用,1993,3:38~43.
2 周龙骧. 数据库管理系统实现技术. 武汉:中国地质大学出版社,1990.
3 David J *et al*. Implementation technologies for main memory Database System. ACM, 1984. 1~8.
4 Horowitz E, Sahni S. Fundamentals of data structures. Pitmen Publishing Limited, 1976.

AN ACCESS METHOD FOR MAIN MEMORY DATABASES

Liu Yunsheng Hu Guoling

(*Department of Computer Science Huazhong University of Science and Technology Wuhan* 430074)

Abstract In this paper, the authors present a main memory database system ARTs—MMDBS, and discuss with the emphasis on its database management mechanisms and access methods. And the paper describes in detail a new index structure—SB tree used in ARTs—MMDBS.

Key words Main memory database, access method, index structure, database organization, SB tree.