并行数据库上的并行CMD_Join算法*

李建中 都 薇

(黑龙江大学信息技术研究所 哈尔滨 150080)

摘要 并行数据库在多处理机之间的分布方法(简称数据分布方法)对并行数据操作算法的性能影响很大.如果在设计并行数据操作算法时充分利用数据分布方法的特点,可以得到十分有效的并行算法.本文研究如何充分利用数据分布方法的特点,设计并行数据操作算法的问题,提出了基于CMD多维数据分布方法的并行CMD_Join算法.理论分析和实验结果表明,并行CMD_Join算法的效率高于其它并行Join算法.

关键词 并行数据库,数据分布方法,CMD数据分布方法,并行Join算法,并行CMD_Join算法.

中图法分类号 TP311.13

在并行数据库系统研究领域中,并行Join算法一直是人们关心的重要问题,很多并行Join算法已经被提出.[1,2]这些并行Join算法可以分为3类:并行循环嵌套Join算法[3]、并行排序合并Join算法[4]和并行Hash算法.[5~8]在这些算法的基础上,人们又研究了数据偏斜对并行Join算法的影响,提出了抗数据偏斜影响的并行Join算法.[7~9]

在并行关系数据库系统中,每个关系都由某种方法划分为多个子集合,分布到多个处理机.关系的数据分布方法对并行数据操作算法的性能影响很大.如果在设计并行数据操作算法时充分利用数据分布方法的特点,我们会得到十分有效的并行算法.我们曾以CMD数据分布方法[10]为基础,设计了并行多维区域查询算法.[11]理论分析和实验结果表明,充分利用数据分布方法的特点可以获得高效率的并行算法.

本文深入研究利用数据分布方法的特点,设计并行数据操作算法的问题,提出了基于CMD多维数据分布方法的并行CMD_Join算法.CMD_Join算法具有以下几个特点:①利用CMD关系部分排序的特点,剔除两个Join关系中不需要Join的子集合,打破了Join操作至少存取整个操作关系一遍的传统复杂性下界;②具有Hybrid-Hash-Join算法的所有优点,但I/O工作量远小于Hybrid-Hash-Join算法;③CMD_Join算法的效率高于其它并行Join算法.

不失一般性,本文以SHARED-NOTHING并行计算结构[12]为基础来讨论CMD_Join算法.在这种结构中,每个处理机都有独立的CPU、存储器和磁盘系统.

1 预备知识

我们简单介绍一下CMD数据分布方法(详见文献[10]).给定一个具有d个属性的关系$R(A_1,\ldots,A_d)\in D_1\times\ldots\times D_d$,其中$D_i$是属性$A_i$的定义域,CMD方法把$R$视为$d$维空间$S=D_1\times\ldots\times D_d$的子集合.CMD方法首先根据$R$的数据分布情况把空间$S$划分为多个子空间,使得$R$在每个子空间中具有近似相同的元组数.于是,关系$R$被划分成大小近似相等的子集合.然后,CMD方法使用坐标和求模函数把S的每个子空间分配到一个处理机.为了叙述简单,令$S=[0,1)^d$.设P是处理机个数,P个处理机的内存容量(元组数)分别为$M_1,M_2,\ldots,M_P$.

为了划分空间S,CMD方法把空间S的第i维的值域划分成长度为$\frac{1}{n_iP}$的n_iP个子区间:

$$[0,\frac{1}{n_iP}),[\frac{1}{n_iP},\frac{2}{n_iP}),\ldots,[\frac{n_iP-1}{n_iP},1)$$

其中n_i是调整因子,n_i必须充分大,使得下列两个条件成立:

(1) 划分后的每个超方体所包含的R的元组可装入一个磁盘页;

---

* 本文研究得到国家自然科学基金、国家863高科技项目基金、国家杰出青年基金和黑龙江省杰出青年基金资助.作者李建中,1950年生,教授,博士导师,主要研究领域为数据库与并行计算.都薇,女,1973年生,硕士生,主要研究领域为数据库.

本文通讯联系人:李建中,哈尔滨 150080,黑龙江大学信息技术研究所

本文1997-04-14收到原稿,1997-07-25收到修改稿

(2) R 的任一维 i 的任一区间 $[\frac{j}{n_iP},\frac{j+1}{n_iP})$ 满足：$|\{t|t\in R,t[A]\in[\frac{j}{n_iP},\frac{j+1}{n_iP})\}|\leqslant\sum_{i=1}^{P}M_i$. 其中 $|X|$ 是关系 X 的元组数，$t[A]$ 是元组 t 的 A 属性值，A 是 R 的第 i 维对应的属性.

条件(2)保证了关系 R 与每维的每个划分区间对应的子集合(即 A 属性值属于该划分区间的元组集合)可以存储在 P 个处理机的内存中，有效地支持并行数据操作算法的设计与实现.

设 $I_{ij}=[\frac{j}{n_iP},\frac{j+1}{n_iP})$ 是第 i 维上的第 j 个子区间，j 是这个子区间的坐标. CMD 方法把 S 划分为 $(n_1P\times\ldots\times n_dP)$ 个超长方体，每个超长方体都是 d 个子区间的笛卡尔积 $I_{1x}\times\ldots\times I_{dx_d}$，$(x_1,\ldots,x_d)$ 定义为该超长方体的坐标. 超长方体 $(x_1,\ldots,x_d)$ 被分配到处理机 $CMD(x_1,\ldots,x_d)$，其中 CMD 函数为：$CMD(x_1,\ldots,x_d)=(x_1+\ldots+x_d)\bmod P$. 图 1 给出了一个 $S=[0,1)^2$ 在 4 个处理机间分布的实例，其中 $n_1=n_2=2$，$P=4$，最左列和最底行的标记是各维上的子区间的坐标，方框内的数字表示相应的超方体被分配到的处理机号. 例如，超方体(6,6)被分配到处理机 0.

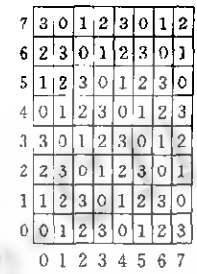

图 1　$S=[0,1)^2$ 在 4 个处理机间划分的实例

定义 1.1. 关系 R 的 CMD 存储结构是一个二元组 $R_{CMD}=(\{PV_1,\ldots,PV_d\},\{R_1,\ldots,R_N\})$，其中 PV_i 是关系 R 的第 i 个属性定义域的划分向量，按划分点递增顺序排列，R_j 是 R 留驻在第 j 个处理机的数据子集合. 用 CMD 方法存储的关系称为 CMD 关系.

CMD 关系具有很多好性质[12]，除整个关系均匀地分布在多个处理机上以外，下面几个性质对于并行数据操作算法的设计十分重要：

(1) 在任一维 k 上都部分有序：设 I_{ki} 和 I_{kj} 是第 k 维(对应属性为 A)上的两个划分区间，如果 $i<j$，则对于任意 $u\in\{t|t\in R,t[A]\in I_{ki}\}$，$v\in\{t|t\in R,t[A]\in I_{kj}\}$，$u[A]<v[A]$；

(2) 对于任意维 k 上的任意一个划分区间 I_{kj}，$\{t|t\in R,t[A]\in I_{kj}\}$ 的数据量不超过全部处理机的内存空间总容量，并且均匀地分布在 P 个处理机上；

(3) 如果 I_{ki} 和 I_{kj} 是第 k 维(对应属性为 A)上两个不同的划分区间，则 $\{t|t\in R,t[A]\in I_{ki}\}$ 和 $\{t|t\in R,t[A]\in I_{kj}\}$ 具有近似相等数量的元组.

2　基本概念

CMD 关系的每个属性对应一维或一个坐标轴. Join 属性对应的坐标轴称为 Join 轴.

定义 2.1. 设 $R\subseteq D_1\times\ldots\times D_d$ 是一个 CMD 关系，a 是 R 的 Join 轴，被划分为 k 个区间，R 沿 Join 轴的第 i 个 $(0\leqslant i\leqslant k)$ Join 区域定义为集合 $(D_1\times\ldots\times D_{i-1}\times I_i\times D_{i-1}\times\ldots\times D_d)\cap R$，记作 $JR(R,a,i)$，其中 I_i 是 R 的 Join 轴的第 i 个区间.

下面我们分别用 $L(R,a,i)$ 和 $U(R,a,i)$ 表示 $JR(R,a,i)$ 的 a 属性值的下界和上界. 图 2 用二元关系 $R(a,b)$ 举例说明这些概念，其中属性 a 是 Join 属性.

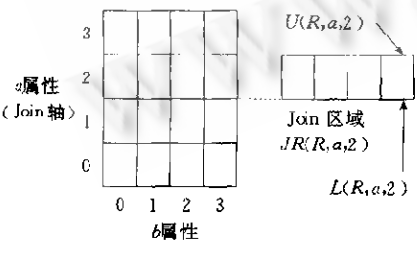

图2　二元关系 R 和它的一个Join区域

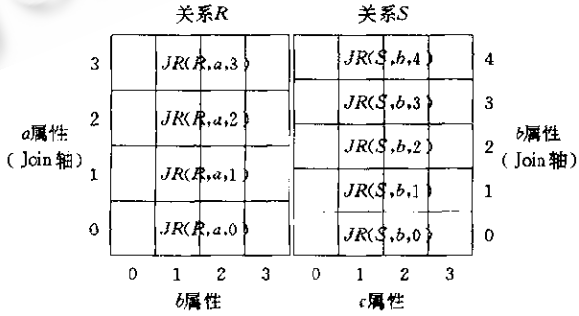

图3　一个说明Join相关和Join无关概念的实例

定义 2.2. 设 R 和 S 是等值 Join 操作的操作关系，a 是 R 的 Join 属性，b 是 S 的 Join 属性，R 和 S 都是 CMD 关系. 如果条件(简称 Join 条件)

$$(L(R,a,i)\leqslant L(S,b,j)\leqslant U(R,a,i))\vee(L(S,b,j)\leqslant L(R,a,i)\leqslant U(S,b,j))$$

成立,则称 $JR(R,a,i)$和 $JR(S,b,j)$ Join 相关,否则,称 $JR(R,a,i)$和 $JR(S,b,j)$ Join 无关.

图 3给出了一个说明 Join 相关和 Join 无关概念的实例.例如,R 的 Join 区域 $JR(R,a,2)$与 S 的 Join 区域 $JR(S,b,2)$和 $JR(S,b,3)$ Join 相关,S 的 Join 区域 $JR(S,b,4)$与 R 的 Join 区域 $JR(R,a,2)$ Join 无关.

引理 2.1. 设 CMD 关系 R 和 S 是等值 Join 操作 J 的操作关系,a 是 R 的 Join 属性,b 是 S 的 Join 属性.$JR(R,a,i)$和 $JR(S,b,j)$分别是 R 和 S 的 Join 区域.如果 $JR(R,a,i)$和 $JR(S,b,j)$ Join 无关,则 R 和 S 的 Join 结果不包括 $(JR(R,a,i)\times JR(S,b,j))$元组.

证明:如果 $JR(R,a,i)$和 $JR(S,b,j)$ Join 无关,则 Join 条件 $(L(R,a,i)\leqslant L(S,b,j)\leqslant U(R,a,i))\vee(L(S,b,j)\leqslant L(R,a,i)\leqslant U(S,b,j))$为假.从而 $L(S,b,j)>U(R,a,i)$或 $L(S,b,j)<L(R,a,i)$,同时 $L(S,b,j)>L(R,a,i)$或 $L(R,a,i)>U(S,b,j)$.如果 $L(S,b,j)>U(R,a,i)$,则$\forall\ t\in JR(R,a,i)\times JR(S,b,j)$,$t[a]\leqslant U(R,a,i)<L(S,b,j)$,$L(S,b,j)\leqslant t[b]$.从而 $t[a]\neq t[b]$,即 t 不属于 R 和 S 的 Join 结果.如果 $L(S,b,j)<L(R,a,i)$,必有 $U(S,b,j)<L(R,a,i)$.于是,$\forall\ t\in JR(R,a,i)\times JR(S,b,j)$,$L(R,a,i)\leqslant t[a],t[b]\leqslant U(S,b,j)<L(R,a,i)$.从而 $t[a]\neq t[b]$,即 t 不属于 R 和 S 的 Join 结果. □

引理 2.1 说明,计算 R 和 S 的 Join 结果时,我们只需计算 R 和 S 的 Join 相关区域的 Join 结果,不必考虑 Join 无关区域.所有 Join 相关区域的 Join 结果的并集即为 R 和 S 的 Join 结果.任意一个 CMD 关系 R 的 Join 区域 $JR(R,a,i)$的 $L(R,a,i)$和 $U(R,a,i)$可由 R 的划分向量计算出来.所以,两个 CMD 关系的 Join 区域的相关性可以由它们的划分向量来确定.

3 CMD_Join 算法

CMD_Join 算法分为两个阶段.第 1 阶段确定两个 Join 关系的 Join 相关区域集合,每个关系的 Join 无关区域将被剔除;第 2 阶段进行相关 Join 区域的 Join 处理.在处理过程中,一个关系的 Join 区域仅与另一个关系的 Join 相关区域进行 Join 操作,不需要与其它 Join 区域进行 Join 操作.例如,在图 3 中,关系 R 的 Join 区域 $JR(R,a,1)$只需要与关系 S 的区域 $JR(S,b,1)$和 $JR(S,b,2)$进行 Join 操作.对于每对 Join 相关区域,我们可以用 Hash,Sort_Merge 和 Nested Loop方法[2]完成它们的 Join 操作.于是,我们有如下 3 个 CMD_Join 算法.

算法. $\text{CMD\_Join}_{\text{Hash\_Join}}(R,S)$

输入:R:CMD 关系,Join 属性为 a;S:CMD 关系,Join 属性为 b.

输出:R 与 S 的 Join 结果.

(1) 使用 R 和 S 的划分向量确定 R 和 S 的 Join 相关区域:
$JR(R,a,x),JR(R,a,x+1),\ldots,JR(R,a,y);\ JR(S,b,z),JR(S,b,z+1),\ldots,JR(S,b,w)$;

(2) $j:=z$;

(3) FOR $i=x$ TO y DO (P 个处理机并行地)

(4) 读 $JR(R,a,i)$;

(5) 用 Hash 函数分布 $JR(R,a,i)$到所有可用处理机;

(6) 每个处理机建立所接受的 $JR(R,a,i)$子集合的 Hash 表;

(7) WHILE ($j\leqslant w$) $\wedge$ ($JR(S,b,j)$和 $JR(R,a,i)$ Join 相关) DO (并行地)

(8) IF 第 1 次访问 $JR(S,b,j)$

(9) THEN 读 $JR(S,b,j)$,并使用 Hash 函数分布到 $JR(R,a,i)$所在处理机集合;
ENDIF;

(10) 每个处理机用 $JR(S,b,j)$子集搜索 $JR(R,a,i)$子集的 Hash 表,完成 $JR(S,b,j)$子集与 $JR(R,a,i)$子集的的 Hash Join;

(11) $j:=j+1$;

(12) ENDWHILE;

(13) IF ($L(R,a,i)<L(S,b,j-1)$) THEN

(14) $j:=j-1$; /* 若 $JR(S,b,j-1)$与 $JR(R,a,i+1)$ Join 相关,$JR(S,b,j-1)$需与 $JR(R,a,i+1)$进行 Join */

(15) ENDIF;

(16) ENDFOR.

算法. $\text{CMD\_Join}_{\text{Sort\_Merge}}(R,S)$

输入:R:CMD 关系,Join 属性为 a;S:CMD 关系,Join 属性为 b.

输出:R 与 S 的 Join 结果.

(1) 同 $\text{CMD_Join}_{\text{Hash_Join}}(R,S)$算法的(1);

(2) $j:=z$;

(3) FOR $i=x$ TO y DO (P 个处理机并行地)

(4) 读 $JR(R,a,i)$;

(5)　用Hash函数分布$JR(R,a,i)$到所有可用处理机;
(6)　每个处理机对所接受的$JR(R,a,i)$子集合进行排序;
(7)　WHILE ($j\leqslant w$)∧($JR(S,b,j)$和$JR(R,a,i)$ Join相关) DO (并行地)
(8)　　IF 第1次访问$JR(S,b,j)$
(9)　　THEN 读$JR(S,b,j)$,并使用Hash函数分布到$JR(R,a,i)$所在处理机集合;
　　ENDIF;
(10)　　每个处理机对$JR(S,b,j)$子集合进行排序;
(11)　　每个处理机进行$JR(R,a,i)$子集和$JR(S,b,j)$子集的合并Join;
(12)　　$j:=j+1$;
(13)　ENDWHILE;
(14)　IF ($L(R,a,i)<L(S,b,j-1)$) THEN
(15)　　$j:=j-1$;　　/* 若$JR(S,b,j\ \ 1)$与$JR(R,a,i+1)$ JOIN相关,$JR(S,b,j-1)$需与$JR(R,a,i-1)$进行Join */
(16)　ENDIF;
(17) ENDFOR.

算法. CMD_Join$_{\text{Nested_Loop}}$(R,S)

输入:R:CMD关系,Join属性为a;S:CMD关系,Join属性为b.
输出:R与S的Join结果.
(1) 同CMD_Join$_{\text{Hash_Join}}$(R,S)算法的(1);
(2) $j:=z$;
(3) FOR $i=x$ TO y DO (并行地)
(4)　读$JR(R,a,i)$;
(5)　用Hash函数分布$JR(R,a,i)$到所有可用处理机;
(6)　WHILE ($j\leqslant w$)∧($JR(S,b,j)$和$JR(R,a,i)$ Join相关) DO (并行地)
(7)　　IF 第1次访问$JR(S,b,j)$
(8)　　THEN 读$JR(S,b,j)$,并使用Hash函数分布到$JR(R,a,i)$所在处理机集合;
　　ENDIF;
(9)　　每个处理机进行所接收到的$JR(R,a,i)$子集和$JR(S,b,j)$子集的Nested_Loop_Join;
(10)　　$j:=j+1$;
(11)　ENDWHILE;
(12)　IF ($L(R,a,i)<L(S,b,j-1)$) THEN
(13)　　$j:=j-1$;　　/* 若$JR(S,b,j-1)$与$JR(R,a,i+1)$ JOIN相关,$JR(S,b,j\ \ 1)$需与$JR(R,a,i+1)$进行Join */
(14)　ENDIF;
(15) ENDFOR.

4　算法的复杂性分析

设CMD关系R和S的划分向量存储在主存储器中,R和S的任意一对Join相关区域都可以存储在主存储器中,各算法的Hash函数能够把R和S每个Join区域均匀地分布到P个处理机.下面是复杂性分析中使用的参数:T_{comp}=CPU完成一个算术逻辑计算或算术逻辑比较的平均时间;T_{comm}=在处理机之间传输一个字节数据的平均时间;T_{move}=在内存中移动一个元组的时间;$T_{I/O}$=读写一个磁盘物理页数据的时间;t=每个磁盘物理页存储的元组数;b=每个元组的字节数;$|x|$=关系x的元组数;n=关系R的元组数;m=关系S的元组数.

我们首先分析CMD_Join$_{\text{Hash_Join}}$算法的并行执行时间.设n_RP是R的连接属性a的划分区间数,n_SP是S的连接属性b的划分区间数.算法第1步的并行执行时间为$O(T_{\text{comp}}(n_RP+n_SP))$,第2步的执行时间是常数.于是,第1和第2步的并行执行时间为$T_{1\sim2}=O(T_{\text{comp}}(n_RP+n_SP))$.

由第2节最末介绍的CMD关系的特点(2)、(3),$JR(R,a,i)$均匀地分布在P个处理机上,而且每个处理机上具有$\frac{n}{n_RP^2}$个$JR(R,a,i)$的元组.算法第4步的执行时间是$O(\frac{n}{n_RP^2}(T_{I/O}+T_{\text{move}}))$.第5步的执行时间是$O(\frac{n}{n_RP^2}(T_{\text{comp}}+bT_{\text{comm}}))$.第6步的执行时间是$O(\frac{n}{n_RP^2}(T_{\text{comp}}+T_{\text{move}}))$.第4～6步总的执行时间为$T_{4\sim6}=O(\frac{n}{n_RP^2}(T_{I/O}+T_{\text{move}}+T_{\text{comp}}+bT_{\text{comm}}))$.

第8和第9步的执行时间为$O(\frac{m}{n_SP^2}(T_{I/O}+T_{\text{move}}+T_{\text{comp}}+bT_{\text{comm}}))$.第10步的执行时间是$O(\frac{m}{n_SP^2}(T_{\text{comp}}+T_{\text{move}}+\frac{n}{n_RP^2}T_{I/O})$,其中$\frac{m}{n_SP^2}\frac{n}{n_RP^2}T_{I/O}$是写Join结果的时间上界.第11步的执行时间是常数.第7～12步循环的执行次数等于与$JR(R,a,i)$ Join相关的S的Join区域个数.设与R的每个Join区域Join相关的S的Join区域数平均为α.第7～

12步的并行执行时间为

$$T_{7\sim12}=O(\alpha\frac{m}{n_SP^2}((1+\frac{n}{n_RP^2})T_{I/O}+T_{move}+T_{comp}+bT_{comm})).$$

第13～15步的执行时间为常数.第3～16步循环次数至多为n_RP.于是,$CMD_Join_{Hash_Join}$算法的并行执行时间为$O(T_{1\sim2}+n_RP(T_{4\sim6}+T_{7\sim12}))$.

现在我们来分析$CMD_Join_{Sort_Merge}$算法的并行执行时间.$CMD_Join_{Sort_Merge}$算法第1和第2步的执行时间为$T_{1\sim2}=O(T_{comp}(n_RP+n_SP))$.类似于$CMD_Join_{Hash_Join}$算法,$CMD_Join_{Sort_Merge}$算法第4步的执行时间是$O(\frac{n}{n_RP^2}(T_{I/O}+T_{move}))$.第5步的执行时间是$O(\frac{n}{n_RP^2}(T_{comp}+bT_{comm}))$.第6步的执行时间为$O(\frac{n}{n_RP^2}\log\frac{n}{n_RP^2}(T_{comp}+T_{move}))$.第4～6步的并行执行时间为

$$T_{4\sim6}=O(\frac{n}{n_RP^2}(T_{I/O}+T_{move}+T_{comp}+bT_{comm})+\frac{n}{n_RP^2}\log\frac{n}{n_RP^2}(T_{comp}-T_{move})).$$

第8和第9步的执行时间为$O(\frac{m}{n_SP^2}(T_{I/O}+T_{move}+T_{comp}+bT_{comm}))$.第10和第11步的执行时间分别为$O(\frac{m}{n_SP^2}\times\log\frac{m}{n_SP^2}(T_{comp}+T_{move}))$和$O((T_{comp}+T_{move})(\frac{n}{n_RP^2}+\frac{m}{n_SP^2})+\frac{m}{n_SP^2}\frac{n}{n_RP^2}T_{I/O})$.其中$\frac{m}{n_SP^2}\frac{n}{n_RP^2}T_{I/O}$是写Join结果的时间上界.第7～13步循环的执行次数等于与$JR(R,a,i)$ Join相关的S的Join区域数,设其均值α.第7～13步的并行执行时间为

$$T_{7\sim13}=O(\alpha(\frac{m}{n_SP^2}(T_{I/O}+T_{move}+T_{comp}+bT_{comm})+\frac{m}{n_SP^2}\log\frac{m}{n_SP^2}(T_{comp}+T_{move})+$$
$$(T_{comp}+T_{move})(\frac{n}{n_RP^2}+\frac{m}{n_SP^2})+\frac{m}{n_SP^2}\frac{n}{n_RP^2}T_{I/O})).$$

第14～16步的执行时间为常数.第3～17步循环次数至多为n_RP.于是,$CMD_Join_{Sort_Merge}$算法的并行执行时间为$O(T_{1\sim2}+n_RP(T_{4\sim6}+T_{7\sim13}))$.

最后,我们来分析$CMD_Join_{Nested_Loop}$算法的并行执行时间.$CMD_Join_{Nested_Loop}$算法的第1和第2步的执行时间为$T_{1\sim2}=O(T_{comp}(n_RP+n_SP))$.

类似于$CMD_Join_{Sort_Merge}$算法,$O(\frac{n}{n_RP^2}(T_{I/O}+T_{move}))$是$CMD_Join_{Nested_Loop}$算法第4步的执行时间.第5步的执行时间是$O(\frac{n}{n_RP^2}(T_{comp}+bT_{comm}))$.第4和第5步的并行执行时间为

$$T_{4\sim5}=O(\frac{n}{n_RP^2}(T_{I/O}+T_{move}+T_{comp}+bT_{comm})).$$

第7和第8步的执行时间为$O(\frac{m}{n_SP^2}(T_{I/O}+T_{move}+T_{comp}+bT_{comm}))$.第9步的执行时间为$O((T_{comp}+T_{move})\frac{n}{n_RP^2}\times\frac{m}{n_SP^2}+\frac{m}{n_SP^2}\frac{n}{n_RP^2}T_{I/O})$,其中$\frac{m}{n_SP^2}\frac{n}{n_RP^2}T_{I/O}$是写Join结果的时间上界.第6～11步循环的执行次数等于与$JR(R,a,i)$ Join相关的S的Join区域数,设其均值为α.第6～11步的并行执行时间为

$$T_{6\sim11}=O(\alpha(\frac{m}{n_SP^2}(T_{I/O}+T_{move}-T_{comp}+bT_{comm})+(T_{comp}+T_{move})\frac{n}{n_RP^2}\frac{m}{n_SP^2}+\frac{m}{n_SP^2}\frac{n}{n_RP^2}T_{I/O})).$$

第12～14步的执行时间为常数.第3～15步循环次数至多为n_RP.于是,$CMD_Join_{Nested_Loop}$算法的并行执行时间为$O(T_{1\sim2}+n_RP(T_{4\sim5}+T_{6\sim11}))$.

5 实验结果

我们在一个由8台586微型计算机组成的计算机机群环境下实现了CMD_Join算法,并进行了加速比、响应时间、性能比较等方面的实验分析.多计算机之间的通信由以太网实现.

(1) 加速比.这里的加速比是指用一台处理机执行CMD_Join算法的时间与用多台处理机执行CMD_Join算法的执行时间的比值.为了考察算法的加速比,我们随机地产生了具有5万、10万、15万、20万个元组的4对关系,每个元组长为32字节,然后分别使用1,3,4,5,6,7,8台处理机在每对关系上执行$CMD_Join_{Hash_Join}$,$CMD_Join_{Sort_Merge}$以及$CMD_Join_{Nested_Loop}$算法.对不同处理机台数,CMD_Join算法的加速比随数据量增加而变化的情况如图4～6所示.

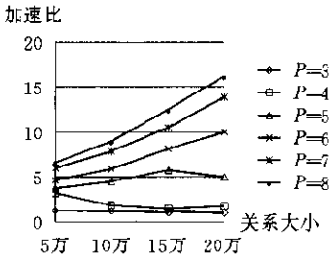

图4　$CMD_Join_{Hash_Join}$算法的加速比

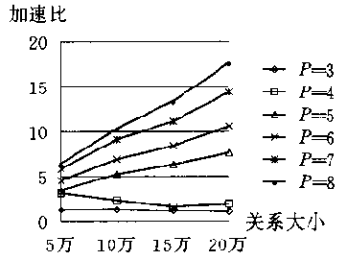

图5　$CMD_Join_{Sort_Merge}$算法的加速比

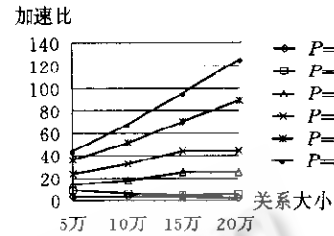

图6　$CMD_Join_{Nested_Loop}$算法的加速比

图中的 P 是处理机个数. 从图 4～6 可以看出，处理机的个数越多，算法的加速比越高. 当处理机个数大于 5 时，加速比随着关系大小的增加十分迅速地增长. 当处理机个数小于 5 时，由于内存容量的限制，I/O 时间构成了整个执行时间的主要部分，加上通信开销，关系大小的增加不能使加速比明显增加. 从这 3 幅图也可以看出，$CMD\_Join_{Nested\_Loop}$ 算法的加速比最高，$CMD\_Join_{Sort\_Merge}$ 算法次之，$CMD\_Join_{Hash\_Join}$ 算法最低. 这是因为单处理机上 $CMD\_Join_{Nested\_Loop}$ 算法的效率最低，$CMD\_Join_{Sort\_Merge}$ 算法次低，$CMD\_Join_{Hash\_Join}$ 算法的效率最高. 从图中可以看到，加速比出现了超线性现象. 超线性加速比的出现是因为单处理机的主存容量远小于多处理机的累计主存容量. 这样，在单处理机情况下，很多 Join 区域的容量远大于主存缓冲区的容量，每次 Join 操作需要反复多次存取 Join 区域，引起了大量的磁盘读写操作. 这种磁盘读写开销随着处理机的增加而减小. 从图中也可以看出，加速比超线性现象随 Join 关系规模的增加而明显增大. 超线性加速比说明，CMD_Join 算法在并行计算系统中具有特殊的优越性.

(2) 执行时间. 为了分析比较 3 种 CMD_Join 算法的执行时间，我们随机地产生了具有 5 万、10 万、15 万、20 万个元组的 4 对关系，每个元组长为 32 字节，然后分别使用 1、3、4、5、6、7、8 个处理机在这些关系上执行 3 个 CMD_Join 算法. 对于不同处理机台数，$CMD\_Join_{Hash\_Join}$，$CMD\_Join_{Sort\_Merge}$ 和 $CMD\_Join_{Nested\_Loop}$ 算法的执行时间随数据量增加而变化的情况分别如图 7～9 所示. 从图中可以看出，对于所有关系大小，处理机个数(在一定范围内)越多，执行时间越短. 执行时间随数据量的增加而增加.

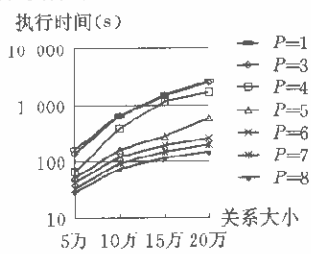

图7

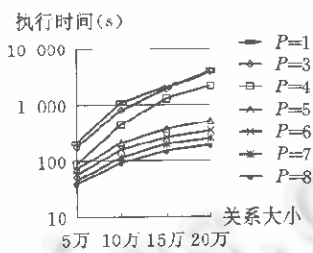

图8

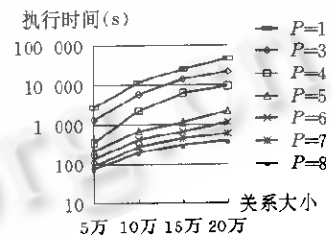

图9

(3) 性能比较. 为了把 3 个 CMD_Join 算法与 Hybrid_Hash_Join 算法、Grace_Hash_Join 算法和并行 Sort_Merge_Join 算法进行比较，我们使用 8 个处理机，在具有 5 万个元组(每个元组 32 个字节)的一对关系上执行了这些算法. 图 10 给出了这些算法的执行时间的比较. 图中的 Ch，Csm，Cnl 分别表示 $CMD\_Join_{Hash\_Join}$，$CMD\_Join_{Sort\_Merge}$ 和 $CMD\_Join_{Nested\_Loop}$ 算法，Hy，Gr，Sm 分别表示 Hybrid_Hash_Join，Grace_Hash_Join 和并行 Sort_Merge_Join 算法. 从图 10 中可以明显地看出，3 种 CMD_Join 算法的效率高于所有其它算法.

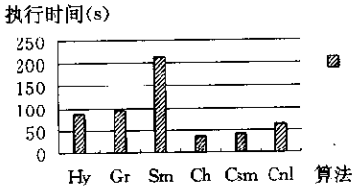

图10

参考文献

1　李建中. 并行数据库的查询并行化技术与物理设计方法. 软件学报，1994，5(10)：1～10
(Li Jian-zhong. Parallelization techniques for query processing and data declustering methods. Journal of Software, 1994, 5(10):1～10)

2　李建中. 并行数据操作算法和查询优化技术. 软件学报，1994，5(10)：11～23
(Li Jian-zhong. Parallel data operation algorithms and query optimization techniques. Journal of Software, 1994, 5(10):11

～23)

3 BoralH *et al*. Join on a cube: analysis, simulation and implementation. In: Kitsuregawa M, Tanaka H eds. Database Machines and Knowledge Base Machines. Boston: Kluwer, 1988. 61～74

4 Schneider D A, DeWitt D J. A performance evaluation of parallel in algorithms in a shared-nothing multiprocessor environment. In: Maier D ed. Proceedings of the ACM SIGMOD'89, USA, 1989. Baltimore: ACM Press, 1989. 110～121

5 Kitsuregawa M, Tanaka H, Motooka T. Application of hash to data base machine and its architecture. New Generation Computing, 1983,1(1),25～39

6 Omiecinski E R, Lin E T. Hash-based and index-based join algorithms for cube and ring connected multicomputers. IEEE Transactions on Knowledge and Data Engineering, 1989,1(3):329～343

7 Kitsuregawa M, Nakayama M, Takagi M. The effect of bucket size tuning in the dynamic hybrid GRACE hash join method. In: Proceedings of the International Conference on Very Large Data Bases'89. San Mateo: Morgan Kaufmann Publishers, Inc., 1989. 257～266

8 Kitsuregawa M, Ogawa Y. Bucket spreading parallel hash: a new bobust, parallel hash join method for data skew in the super database computer (SDC). In: Proceedings of the International Conference on Very Large Data Bases'90. Palo Alto: Morgan Kaufmann Publishers, Inc., 1990. 210～221

9 Hua K A, Lee C. Handling data skew in multiprocessor database computer systems using partition tuning. In: Proceedings of the of the International Conference on Very Large Data Bases'91, Barcelona, 1991. San mateo: Morgan Kaufmann publishers, Inc., 1991. 525～536

10 Li Jian-zhong, Jaideep Srivastava, Doron Rotem. CMD: a multidimensional declustering method for parallel database systems. In: Proceedings of the 18th International Conference on Very Large Data Bases Conference. Canada, 1992

11 Li Jian-zhong. Range query procession in multi-disk systems. Journal of Computer Science and Technology, 1992,7(4),316～327.

12 Stonebraker M. The case for shared nothing. Database Engineering, 1986,9(1):17～24

Parallel CMD_Join Algorithms on Parallel Databases

LI Jian-zhong DU Wei

(*Institute of Information Technology Heilongjiang University Harbin* 150080)

Abstract Data declustering methods of parallel databases have significant effect on the performance of the parallel algorithms on the databases. If the characteristics of the declustering methods of the parallel database can be adequately exploited in designing parallel algorithms for implementing the database operations in parallel database systems, high performance parallel algorithms could be resulted. Unfortunately, the advantages of the underlying declustering methods of the parallel databases are rarely considered in existent parallel algorithms on the parallel databases. The focus of this paper is on how to utilize the advantages of the underlying declustering methods of the databases in designing parallel join algorithms. A set of parallel join algorithms based on the CMD declustering method are proposed. Theoretical and experimental results show that the proposed algorithms are more efficient than other parallel join algorithms.

Key words Parallel database, data declustering method, CMD data declustering method, parallel Join algorithm, parallel CMD_Join algorithm.