一种高效的累进式空间连接查询处理算法

唐桂芬[1],杨伟锋[2],黄双临[1],李 炜[1]
(1. 解放军 61081 部队,北京 100094;2. 空军指挥学院,北京 100045)

摘 要: 累进式空间连接查询广泛应用于空间数据集成、在线空间聚集查询系统中. 本文提出一种高效的累进式空间连接查询处理算法,在现有累进式等值连接查询处理框架之上增加空间连接处理策略,使之适于处理空间连接查询;提出一种基于驻留度的动态同步替换策略处理内存溢出,提高了驻留内存空间对象的利用率;还提出了一种基于 BEA 的不完全连接查询处理算法有效减少磁盘连接计算冗余的 I/O 和 CPU 计算代价. 实验表明,所提出的算法明显优于现有累进式空间连接查询处理算法.

关键词: 累进式空间连接; 替换策略; 不完全连接
中图分类号: TP392 **文献标识码**: A **文章编号**: 0372-2112 (2009) 02-0318-07

An Efficient Progressive Spatial Join Query Processing Algorithm

TANG Gui-fen[1], YANG Wei-feng[2], HUANG Shuang-lin[1], LI Wei[1]
(1. 61081 *Troops of PLA, Beijing* 100094, *China*; 2. *Air Force Command College, Beijing* 100045, *China*)

Abstract: Progressive spatial Join query can be extensively applied in spatial data integration, online spatial aggregation query etc. systems. This paper proposes an efficient progressive spatial join query algorithm. We add spatial query strategy over existing progressive join query architecture, so it can be applied to spatial join query. We present a dynamic concurrent flush policy based on resident degree to process memory overflow, which makes memory-join phase more efficiently. We also propose an incomplete join query algorithm based on BEA, which reduces redundant I/O and CPU cost in disk join phase. Extensive experiments prove that our technique delivers results significantly faster than the previous methods.

Key words: progressive spatial Join; flush policy; incomplete join

1 引言

在空间数据集成系统中[7],用户查询常常涉及多个空间数据源,如查询穿过街区的公交线路,当有关街区和公交线路的空间数据位于不同的服务器上,空间数据集成系统需要对来自这些数据服务器的空间数据进行空间连接运算才能得到用户需要的查询结果. 同样,在线空间聚集查询系统[1],在线空间数据可视化系统[6]也经常遇到类似的情形. 这类查询的共性是:输入空间数据位于远程服务器上,并通过网络传输到本地进行查询处理. 有两种方式处理这类查询[6]:阻塞式查询和累进式查询. 阻塞式查询在消耗所有输入数据之后才会输出查询结果;而累进式查询则采用增量计算方式,通常在接收数据的同时就进行连接运算,并增量式输出连接结果,这样即使输入集阻塞也能快速得到部分查询结果[2]. 显然,累进式查询更适于 Web 应用. 当前,累进式查询已成为新的研究热点[1~3,5].

现有累进式连接查询处理算法大多采用内存连接和磁盘连接两阶段处理框架[2,3,5]. 内存连接阶段完成新接收数据对象与驻留内存数据对象之间的连接计算,内存溢出时依据一定的替换策略将内存中的部分数据调度到磁盘上,以处理后续到达的数据;输入数据全部到达本地后,再由磁盘连接阶段处理内存连接阶段未完成的连接计算. 现有累进式连接查询算法应用于空间连接查询时性能低下. 当前面向累进式空间连接查询的研究成果还很少.

本文研究并提出一种高效的累进式空间连接查询处理算法,其思想是在现有的累进式连接查询处理框架上添加空间处理策略以处理空间连接查询,并通过有效的替换策略和磁盘连接算法提高累进式空间连接查询处理性能.

2 相关工作

针对累进式等值连接查询,Wilschut 等人[8]首先提

收稿日期:2007-01-30;修回日期:2008-09-01
基金项目:自然科学基金(No. 40601080)

出了基于流水线的对称哈希连接算法 SHJ,该算法要求输入数据总量小于系统可用内存. 基于 SHJ,Urhan[3]、Mohamed[2]以及 Dittrich[4]先后提出了扩展哈希连接处理算法 XJoin、累进式合并连接算法 PMJ 和哈希合并连接算法 HMJ. 这些算法都是对 SHJ 的自然扩展,为了处理内存溢出,它们采用了不同的替换策略. XJoin 采用了最大替换策略,即将数据量最大的 Hash 桶调度到磁盘上;PMJ 使用了全部替换策略,该策略降低了查询系统内存利用率和算法性能;HMJ 采用了适应性启发式替换策略. 这些替换策略都没有考虑数据传输特性、空间数据相对分布等影响因素,不能保证驻留内存数据的利用率.

在累进式空间连接查询方面,GangLuo[6]提出了非阻塞式并行空间连接查询算法 NPSJ,该算法同样采用两阶段的处理方法并将输入空间数据分两部分处理,先到达的空间对象在内存中按照 R 树方式组织,后到达的空间对象直接在磁盘上按照空间划分组织. 本质上,NPSJ 采用的是一种不替换策略,该策略对于两个输入集不同步时效率非常低.

另外,上述算法[2~4,6]在磁盘连接阶段均采用传统合并连接算法或其变体. 事实上,由于内存连接阶段已经完成了部分连接计算,因此,磁盘连接阶段所处理的是一种不完全连接. 此时仍然采用合并连接算法不可避免地产生较大冗余 I/O 和 CPU 计算. 目前还没有算法考虑到磁盘链接阶段的优化问题.

3　问题描述与 PSJ 查询处理框架

3.1　问题描述

设 R_1 和 R_2 是分别位于两个远程数据服务器上空间数据集. 我们采用类 SQL 方式[6]来表达对 R_1 与 R_2 的空间连接查询:

SELECT　　*

FROM　　*online* R_1, R_2

WHERE *Intersect* (R_1. *sa*1, R_2. *sa*2) = *TRUE*.

其中,*sa*1 和 *sa*2 分别是 R_1 和 R_2 的空间属性;*online* 表明空间连接查询的输入集是以数据流的方式传输到本地进行连接运算的.

累进式空间连接(Progressive Spatial Join,PSJ)查询不仅能够快速找出 R_1 和 R_2 中所有满足空间谓词 *Intersect*(空间谓词还可以是其它空间关系)的空间对象偶,还具有以下特性[4]:(1)尽快输出初始部分查询结果;(2)数据传输拥塞也能持续输出查询结果.

3.2　PSJ 查询处理框架

图 1 给出了 PSJ 查询处理框架,对 R_i (i = 1 or 2),PSJ 将已接收的空间数据分两部分存储 R_i^{mem} 和 R_i^{disk},分别表示内存和磁盘中的数据. 与现有的累进式连接查询算法[2,3,5]类似,PSJ 也是基于划分的,采用空间规则划分函数[10]将输入空间数据集划分为 N 个部分,对应地,R_i^{mem} 和 R_i^{disk} 中的数据分别组织到 N 个数据桶中. R_i^{mem} 和 R_i^{disk} 的第 j 个划分分别表示为 $R_i^{mem}[j]$ 和 $R_i^{disk}[j]$ ($j \in [1, N]$).

同样,PSJ 采用两阶段处理框架[2,3,5]. 如图 1,内存连接阶段 PSJ 的基本步骤为:(1)将新接收的空间对象 r (假定 $r \in R_1$, $r \in R_2$ 时处理与此类似)插入(*insert*)相应的数据桶 $R_1^{mem}[j]$ ($j \in [1, N]$);(2)用 r 探测(*probe*) $R_1^{mem}[j]$,并将匹配的结果输出;(3)内存溢出时,使用第 4 节提出的替换策略将部分驻留内存数据调度(*flush*)到磁盘上. 当输入数据全部到达本地后,PSJ 开始磁盘连接阶段(*d-d join*).

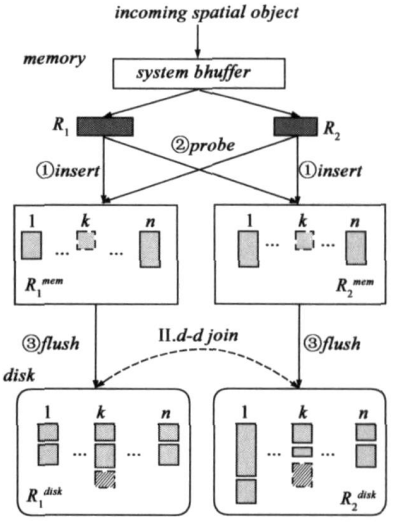

图1　PSJ的查询处理框架

由于空间连接比等值连接的查询代价要高得多,因此,累进式空间连接查询比等值连接查询更容易出现数据滞留现象[9],即处理新接收空间对象与驻留内存空间对象之间连接运算所耗费的时间大于系统连续接收两个空间对象的时间间隔,导致未处理空间对象滞留在缓存中,影响系统查询的性能. 为此,PSJ 在内存连接阶段采用了适应性过滤-提炼两步骤的空间连接策略处理空间连接,与传统空间数据库中顺序执行过滤-提炼两步骤[10]不同,PSJ 中过滤操作和提炼操作可以灵活调度与切换,过滤操作保持较高优先级别,当数据传输拥塞或缓冲区空闲内存较多时,PSJ 会自动触发提炼操作处理候选对象集,并及时将匹配结果输出. 这样 PSJ 不仅能够避免数据滞留,还实现了查询结果的持续输出. 相应地,PSJ 在内存中主要存储空间对象的概要信息:*key-tuple* < *OID*, *MBR* > 二元组,其中 *OID*、*MBR* 分别为空间对象的唯一标识和最小矩形包围框. 本文后

续工作主要针对过滤操作进行的.

4 基于驻留度的动态同步替换策略

替换策略是指在内存溢出时将内存数据调度到磁盘上的准则,替换策略对累进式连接查询算法性能影响很大[9].理想的替换策略能够将未来利用率较低的数据替换到磁盘上,从而提高内存连接阶段的输出率[11].考虑第 j 个划分,若未来到达数据属于 $R_1^{mem}[j]$ 的比较少,那么 $R_2^{mem}[j]$中的数据在未来利用率就比较低.另一方面,如果未来到达 $R_1^{mem}[j]$的数据与当前 $R_2^{mem}[j]$中数据的空间分布倾斜程度非常大,那么当前 $R_2^{mem}[j]$数据未来的利用率也不高.

基于此,我们提出一种基于驻留度的动态同步替换策略(Dynamic Concurrent Flush Policy,DCFP).所谓同步替换是指将一个数据划分中两个数据桶($R_1^{mem}[j]$和 $R_2^{mem}[j]$)同时替换到磁盘上[2,3].我们使用驻留度来评价数据划分被替换的程度,驻留度越大该数据划分被替换的机会越小,反之该数据划分被替换机会越大.

对于 $R_1^{mem}[j]$,其驻留度由两个因素决定:(1) 当前 $R_1^{mem}[j]$中的空间对象与 $R_2^{mem}[j]$中未来到达空间对象之间的相对分布;(2) 未来到达空间对象属于 $R_2^{mem}[j]$的概率 $R_2^{mem}[j]$.据此我们定义数据桶驻留度.

定义 1 数据桶增量连接率(BIJR). 设 $n_1^{mem}[j]$为当前 $R_1^{mem}[j]$的空间对象数,$n_1^{mem}[j]$为当前 $R_1^{mem}[j]$的空间对象与 $R_2^{mem}[j]$未来到达的空间对象产生的连接结果数,则 $R_1^{mem}[j]$的增量连接率定义为:$BIJR_1[j] = n_1^{reslt}[j]/\ n_1^{mem}[j]$同理:$BIJR_2[j] = n_2^{reslt}[j]/\ n_2^{mem}[j]$.

定义 2 数据桶驻留度(BRD). $R_1^{mem}[j]$的驻留度是 $BIJR_1[j]$与 $P_2^{arr}[j]$的乘积.即 $BRD_1[j] = BIJR_1[j]\ *\ P_2^{arr}[j]$,$BIJD_2[j] = BIJR_2[j]\ *\ P_2^{arr}[j]$.

由定义 1,2 可知,数据桶驻留度是基于未来到达数据定义的,且随时间而变化.在实际操作中,我们只能依据历史统计信息来预测驻留度.下面分别给出 $P_i^{arr}[j]$和 $BIJR_i[j]\ (i = 1\ or\ 2, j \in [1, N])$的计算方法.

对于 $BIJR_i[j]$,我们使用 $R_i^{mem}[j]$最近一个替换区间内得到的BIJR来近似表示.该过程需要维护统计信息有:最近一次替换区间内 $R_i^{mem}[j]$包含的空间对象数 $n_i^{mem}[j]$及其新产生的连接结果数 $n_i^{reslt}[j]$.

对于 $P_i^{arr}[j]$,有 $P_i^{arr}[j] = P(R_i)\ *\ P(j|\ R_i)$,其中 $P(R_i)$是未来到达的空间对象属于 R_i 的概率,而 $P(j|\ R_i)$则是未来到达空间对象属于的 $R_i^{mem}[j]$条件概率.$P(R_i)$依赖于 R_1 和 R_2 的相对传输速度,我们使用最近一个替换区间内 R_1 和 R_2 的平均相对传输速度表示未来 R_1 和 R_2 之间的相对传输速度.设 n_i^{resct} 表示最近替换区间内到达 R_i 的对象数,则 $P(R_i) = n_i^{resct}/(n_1^{rect} + n_2^{rect})$.

$P(j|\ R_i)$与空间对象的分布有关,根据已接收空间对象的分布情况可以预测 $P(j|\ R_i)$.$P(j|\ R_i)$等于 $R_i^{mem}[j]$和 R_i 中累计接收空间对象总数的比值:$P(j|\ R_i) = n_i^{totol}[j]/\sum_{k=1}^{N} n_i^{totol}[k]$,其中 $n_i^{totol}[j]$是整个处理过程中 $R_i^{mem}[j]$累计接收的空间对象数目.因此,$P_i^{arr}[j] = n_i^{rescl}\ *\ n_i^{totol}[j]/((n_1^{rescl} + n_2^{rescl})\ *\ \sum_{k=1}^{N} n_i^{totol}[k])$.

在同步替换的约束下,我们将数据划分的驻留度定义为该数据划分包含的两个数据桶驻留度之和.因此,DCFP 可以进一步表述为:将驻留度最小的数据划分替换到磁盘上.

5 基于 BEA 的不完全连接查询

使用合并连接查询算法处理磁盘连接[2~4]产生较高的冗余 I/O 和 CPU 计算代价.以图 2 为例,2(a) 表示第 k 个划分中各数据段之间的空间连接关系.每个圈表示一个数据段,圈中数字为该数据段包含的页面数.数据段的数目表示了内存连接阶段该划分发生替换的次数,图 2(a) 表示第 k 个划分替换了 4 次.假定系统的可用内存为 10 个页面.若采用传统合并连接,此时数据段总数为 8,小于 10,只需一次合并即可,此时连接计算所访问的页面总数为 21,寻道时间也是 21;而按图 2(b) 的方式分解空间连接图,子图依据 $G2 \rightarrow G3 \rightarrow G1$ 的顺序载入内存进行连接计算,相邻子图共同的数据段无需重复载入,如图中的 $sk2$,此时访问页面总数为 20,寻道时间为 7.按新方法连接计算的 I/O 代价减少了 $1 - ((20 + 7)/(21 + 21)) \approx 36\%$.由此可见,有效的数据载入顺序能大大提高磁盘连接的执行效率.

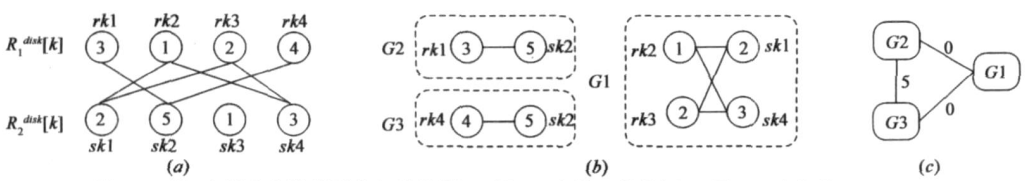

图2 (a)第k个划分内数据段之间连接图;(b)图2(a)中的一种分解;(c)图2(b)对应的WOG

为此,我们研究更有效磁盘连接算法.针对不同的划分,分两种情况考虑:(1) 划分中包含的数据量小于

系统可用内存,此时,将该划分中所有参与连接计算的数据段全部载入内存,并使用嵌套循环加平面扫描技术完成连接计算[4];(2)划分中包含的数据量超过系统可用内存,则使用本节将要介绍的基于 *BEA* 的不完全连接查询算法来执行连接计算,其基本思想是通过优化数据访问顺序减少连接计算过程冗余的 I/O 和 CPU 计算代价. 本节主要讨论第二种情况.

设第 k 个划分包含的数据量超过系统可用内存. 我们引入带权重的二部图 $G_F=(V_F,E_F,w)$ 来表达划分 k 中各数据段之间的连接关系. 因此该图又称为空间连接图(Spatial Join Graph,SJG),其中顶点集 $V_F=\{rk1,rk2,\cdots,rkn,sk1,sk2,\cdots,skn\}$ 是数据段集合,边集 $E_F=\{(rki,skj)|\ rki>,<skj,rki\in R_1^{disk}[k],skj\in R_2^{disk}[k]\}$ 表示各数据段之间的连接关系,$w(v)\geq 0$ 表示每个数据段包含页面数. 设 $d(v)$ 表示与数据段 v 有连接关系的数据段总数,那么磁盘连接 I/O 代价 $C_{I/O}$ 的范围是:$C_{I/O}\in[\sum_{v\in V_F}w(v)\ \sum_{v\in V_F}d(v)\ w(v)]$,即各数据段在整个磁盘连接过程中被访问次数的下限为一次,而上限为 $d(v)$. 减少 $C_{I/O}$,就是要把具有连接关系的数据段尽可能同时载入内存. 不同的数据访问顺序产生不同的 I/O 代价,基于 SJG,降低空间连接计算冗余 I/O 包括两个方面:

(1)有效分解 SJG,尽可能减少相同数据数据段重复载入的次数;

(2)优化子图访问顺序,连续载入的两个子图中相同的数据段无需重复载入,因此,优化的子图载入顺序也能降低冗余 I/O 代价.

5.1　基于 BEA 的 SJG 分解算法

将给定 $SJG=(V_F,E_F,w)$ 的边集 E_F 划分为 m 个子集:$E_1,E_2,\cdots,E_m$,使 $E_1\cup E_2\cup\cdots\cup E_m=E_F$,$\forall 1<i\neq j<m$,$E_i\cap E_j=\varnothing$,且 $V_i=\{v|\ \exists u:(u,v)\in E_i or(v,u)\in E_i\}$,则 $SJG_i=(V_i,E_i,w)$,$1\leq i\leq m$ 构成 SJG 的一个连接子图. 每个子图包含数据总量必须小于或等于系统的可用内存 M,即:$\sum_{v\in V_i}w(v)\leq M$. 整个磁盘连接的 I/O 代价为:$Cost_{I/O}\sum_{v\in V_i}|\{i|\ v\in V_i\}|\ *w(v),i\in[1,m]$,其中$|\{i|\ v\in V_i\}|$ 表示包含节点 v 的子图数. 那么连接查询冗余 I/O 的代价为:$Cost_{RI/O}=\sum_{v\in V_F}.(|\{i|\ v\in V_i\}|-1)\ *w(v),i\in[1,m]$ 有效分解 SJG,即 $Cost_{I/O}$ 最小化,该问题已经被证明是一个 NP 完全问题[14]. 我们使用一种基于 BEA (Bond Energy Algorithm) 矩阵变换算法来实现连接矩阵的有效分解. BEA 算法最早由 McCormick[12] 提出并用于处理数据分解与重组问题. 其思想是用连接矩阵来描述对象之间的关联关系,并通过矩阵的行列变换得到具有全局最优关联关系的对象分布. 这里我们将 BEA 算法应用于分解 SJG. 同样,我们定义一个空间连接矩阵 $SJM=[S_{ij}]_{n*n}$ 来描述空间连接图 $SJG=(V_F,E_F,w)$. *SJM* 定义为:

$$s_{ij}=\begin{cases}w(v_i) & ,\text{if } i=j\\ \dfrac{(w(v_i)+w(v_j))}{2} & ,\text{if } i\neq j\text{and}(v_i,v_j)\in E_F\text{or}(v_j,v_i)\in E_F\\ 0 & ,\text{oterwise}\end{cases}$$

其中,$n=|V_F|$,$v_i,v_j\in V_F$. SJM 对角线元素表示相应数据段包含的页面数. 图 2(*a*) 对应的空间连接矩阵如图 3(*a*) 所示. 连接图的分解等价于连接矩阵的分解. 该过程分两个步骤完成:

(1)使用 BEA 算法将紧密关联的数据段在连接矩阵中聚集放置;

(2)选择一个分解点,将矩阵分解为子矩阵集合.

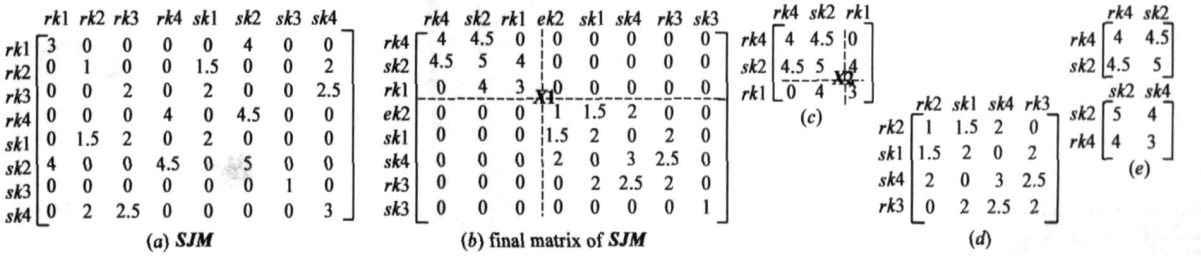

图3　连接矩阵分解示例

下面我们定义关联度和全局关联度.

定义 3　关联度(A)[14]. 度量连接矩阵中两个数据段之间的关联程度:$A(i,j)=\sum_{k=1}^{n}s_{ik}\ *\ s_{kj}$.

对任意的两个数据段 $v_i,v_j\in E_F$,若 $(v_i,v_j)\in E_F$ 或 $\exists v_k$ 满足 $(v_k,v_j)\in E_F$ 且 $(v_k,v_i)\in EF$ 时,则有 $A(i,j)>0$,否则 $A(i,j)=0$. $A(i,j)$ 越大,v_i,v_j 之间的关联程度越大,划分到同一个子图中的概率也越大

定义 4　全局关联度(GA)[14]

度量矩阵中各数据段的整体分布:$GA(SJM)=\sum_{i=1}^{n}(A(i,i+1)+A(i,i-1))$. 其中,$A(i,i-1)$ 和 $A(i,i+1)$ 表示数据段 v_i 与矩阵中相邻的两个数据段之间的关联度. 对于 v_1 和 v_n,约定 $A(1,0)=A(0,1)=A(n+1,n)=A(n,n+1)=0$. $GA(SJM)$ 越大,连接矩阵 *SJM* 的全局关联程度越高,矩阵中各数据段整体分布的聚类性越好.

连接矩阵的全局关联度最大时,在矩阵的主对角

线上可以选择一个分解点,将矩阵 SJM 分解为四个子矩阵:$SJM_{1,1}$(左上),$SJM_{1,2}$(右上),$SJM_{2,1}$(左下)和 $SJM_{2,2}$(右下). $SJM_{1,1}$. 和 $SJM_{2,2}$表示子图内部的连接关系,而 $SJM_{1,2}$和 $SJM_{2,1}$表示子图之间的连接关系. 连接矩阵分解的目标是使各子图内部连接尽可能大,而子图之间连接尽可能少,子矩阵所需的 I/O 代价为:$Cost_{SJM_{k,l}} = \sum_{(i,j)\in SJM_{k,l}} S_{ij}\ (2 \leq k, l \leq 2)$. 因此,分解点确定的依据是使式(1)最大:

$$Cost_X Cost_{SJM_{1,1}} * Cost_{SJM_{2,2}} - Cost_{SJM_{1,2}} * Cost_{SJM_{2,1}} \quad (1)$$

多数情况下,矩阵不能实现完全分解,在矩阵中表现为 $SJM_{1,2}$和 $SJM_{2,1}$中包含非 0 元,为了保持子图的独立性,我们将 $SJM_{1,2}$和 $SJM_{2,1}$中非 0 元对应的数据段在 $SJM_{1,1}$和 $SJM_{2,2}$各保留一个副本,如图 3(c)中的 *sk*2. 若分解后子图所包含的数据量仍然大于系统可用内存,则需要对其进行迭代分解,直至所有子图包含的数据量均小于系统可用内存. 不含迭代过程的连接矩阵分解算法如图 4 算法 1 所示.

算法 1 *Decompose*(SJM)

```
1) Select a row from SJM to TMatrix;
2) FOR(each remaining row v_i in SJM) {
3)     NCGA ←0; pos ←0;
4)     FOR(each position j of the TMatrix) {
5)     CompGA(j) ←2A(TMatrix[j], SJM[i]) +
6) 2A(SJM[i], TMatrix[j+1]) - 2A(TMatrix[j], TMatrix[j+1]);
7)         IF (CompGA(j) > NCGA){
8)           NCGA ←CompGA(j); pos ←j;}
9)     }
10)    Insert v_i into TMartix[pos];
11) }
12) Recalculate SJM according to the TMartix.
13) Cost_X ←0; Dpoint ←0;
14) FOR(each decomposition point i){
15)     TempCost = Cost_SJM1,1 * Cost_SJM2,2 - Cost_SJM1,2 * Cost_SJM2,1;
16)     IF (TempCost > CostX){
17)     Cost_X ←TempCost; Dpoint ←i}
18) }
19) Return SJM1,1 and SJM2,2 of SJM according to Dpoint
```

图 4 连接图分解算法

算法 1 包含了矩阵分解的两个步骤:1)利用 *BEA* 算法进行矩阵变换使全局关联度最大(1-11 行),其做法是依次将 SJM 中各数据段插入临时矩阵 TMatrix 中,确定数据段在 TMatrix 中最终位置的依据是该数据段对 TMatrix 的全局关联度贡献最大. 文献[12]已证明了算法 1 中第 1,2 行数据段选取方法对最终矩阵排列的影响非常小. 因此,本文采用的是顺序选取;2)确定分解点分解矩阵(12-18 行). 图 3(a)中 SJM 按照 BEA 矩阵变换算法得到的具有最优全局关联度的矩阵如图 3(b)所示. 图 3(c)与 3(d)是图 3(b)分解得到的子矩阵. 图 3(e)是 3(c)二次迭代分解的结果.

5.2 优化子图访问顺序

SJG 分解后还要考虑子图的访问顺序,在讨论这个问题之前,我们先给出加权重叠图的定义和加权重叠图 I/O 代价定理.

定义 5 加权重叠图(WOG) 设 $SJG = (V_F, E_F, w)$ 边集 V_F 划分为 m 个子集:$E_1, E_2, \cdots, E_m$, $WOG = (V_m, E_m, wm)$ 是一个带权重的无向图,其中 V_m 表示连接图分解后得到子图集合,$\forall 1 \leq i \neq j \leq m$,边 $(V_i, V_j) \in E_m$ 的权重 wm 为 V_i 和 V_j 中包含相同数据段总的页面数,记为:$wm(V_i, V_j) = \sum_{v\in V_i \wedge v\in V_j} w(v)$,若 V_i 和 V_j 之间没有相同数据段,则 $wm(V_i, V_j) = 0$. 图 2(c)即为图 2(b)对应的 WOG.

定义 6 加权重叠图的路径 给定加权重叠图 $WOG = (V_m, E_m, wm)$,则 WOG 中的一条路径 $P = (Vi_1, Vi_2, \cdots, Vi_m)$,其中,$P$ 经过 WOG 中每个节点一次且仅一次,$wm(Vi_j, Vi_{j+1})$ 为边 (Vi_j, Vi_{j+1}) 的权重,路径的长度为 $\sum_{j=1}^{m-1} wm(Vi_j, Vi_{j+1})$,用 $|P|$ 表示.

由定义 5 和 6,不难证明定理 1 成立.

定理 1 给定空间连接图 $SJG = (V_F, E_F, w)$ 及其加权重叠图 $WOG = (V_m, E_m, wm)$,若 WOG 上的最长路径为 $|P|$(边的权值表示路径长度),则 SJG 连接计算的 I/O 代价可以表示为:$Cost_{I/O} \geq \sum_{v\in V_m} w(v) - |P|$.

我们将 WOG 边权值定义为边的长度,根据定理 1,WOG 的访问顺序问题可归约为求 WOG 最长路径问题. 具体描述为:对于给定的 $WOG = (V_m, E_m, wm)$,优化 WOG 访问顺序的目标是求 WOG 的一条 $|P|$ 值最大的路径 $P = (Vi_1, Vi_2, \cdots, Vi_m)$.

优化 WOG 访问顺序是一个典型的旅行商问题,同样属于 NP-难解问题[13]. 文献[13]给出了遗传算法求解该问题的方法. 考虑到空间连接图分解后,在子图数目较小以及加权重叠图中边权重较小的情况下,子图的访问顺序相对来说对 *I/O* 代价影响较小,因此,我们采用文献[15]所提出的近似算法予以求解.

6 实验仿真与结果分析

6.1 实验设置

表 1 是两组测试数据的元数据. (1) TIGER Line 标准测试数据,包括美国威斯康星州水系(Hy)和公路(Rd1)[16]数据;(2)中国上海的城市空间数据库,包括上海街区(Dt)和道路(Rd2)数据. 实验机器配置为:Celeron 2.8GHz CPU,256MB DDR 内存和 7200RPM 硬盘;操作系统为 Windows XP,测试代码利用 C 语言编写. 仿真环境

设置为:磁盘/内存页面的大小设为 1024byte,系统的可用内存设为 1000 个页面,变化连续读取两个空间对象的时间间隔可模拟不同的数据传输特性. 具体分为:(1) 可靠网络与不可靠网络:若数据传输过程中不发生拥塞即为可靠网络,反之则为不可靠网络. 为了仿真可靠网络,设 $l^{intv} = 10^{-3}$(秒) 为常数. 模拟不可靠网络,根据 Zipfian 分布产生 $l^{intv} \in [10^{-3}, 0.03]$,即最常的网络延迟设为 0.03 秒;(2) 相同传输速率与不同传输速率:设两个空间数据集的传输速率分别记为 $Sp(R_1)$ 和 $Sp(R_2)$. 比较两种情形:①$Sp(R_1) = Sp(R_2)$;②$Sp(R_1) = 5Sp(R_2)$.

表 1　测试数据特性

Dara Sets	Data Type	Object Num	Total Size	Key-tuple	Result Num
TIGER	Hy(Line)	122,149	25.2M	4.4M	34,166
Line	Rd1(Line)	456,613	62.4M	14.6M	
Shang-	Dt(Polygon)	406	0.36M	14.6K	758
Hai	Rd2(Line)	14,714	8.324M	529.7K	

6.2　实验与结果分析

比较 PSJ 与 NPSJ[6] 和 PMJ[4] 在不同的测试条件下的性能差别. PSJ 采用了 DCFP 策略处理内存溢出,磁盘连接阶段使用了基于 BEA 的不完全连接算法;PMJ 和 NPSJ 在内存连接阶段分别采用了全部替换策略[4] 和不替换策略[6]. 磁盘连接阶段 PMJ 和 NPSJ 都采用合并连接算法.

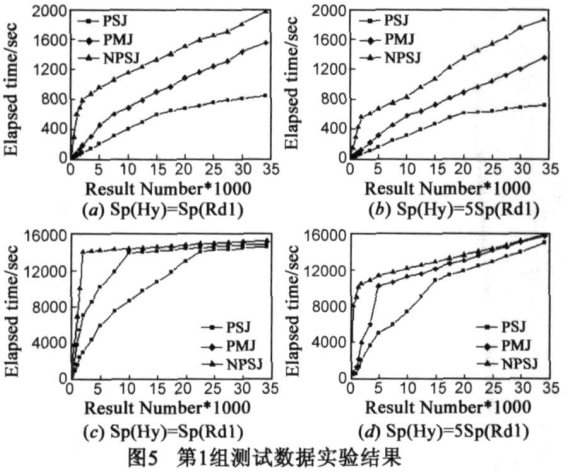

图5　第1组测试数据实验结果

图 5 为第 1 组测试数据的实验结果. 此时,数据量大,超过系统的可用内存,PSJ 将数据集依据文献[10] 的方法划分为 25 个格网进行实验. 在可靠网络环境下,图 5(a) 和(b) 分别表示输入数据集传输速率相同和传输速率不同的条件下 PSJ、PMJ 和 NPSJ 的实验结果;相应地,不可靠网络环境下实验结果如图 5(c) 和(d). 图 5 表明,PSJ 比 PMJ 和 NPSJ 具有更好的查询处理性能,并且输入数据的传输速率对 PSJ 的影响也比 PMJ 和 NPSJ 小. 另外,对于不可靠网络,网络传输代价占整个查询代价的绝大部分,而 PSJ 在内存连接阶段能够输出一半以上的连接结果,PMJ 和 NPSJ 在内存连接阶段产生的连接结果相比 PSJ 要少得多,这也说明 PSJ 大大提高了内存连接阶段驻留内存数据的利用率. 由于输入数据空间分布不均分,这 25 个划分中有 5 个划分的数据量超过系统可用内存,图 6 统计了 PSJ 和 PMJ 磁盘连接阶段的平均 I/O 和 CPU 计算代价. PSJ 使用基于 BEA 的不完全连接算法(UCSJ),而 PMJ 使用传统的合并连接算法(THMJ). 图 6 表明,UCSJ 比 THMJ 查询处理性能提高了很多,整个 I/O 代价减少了 45 %左右,而整个 CPU(提炼步骤 UCSJ 与 THMJ 算法的代价是相同的) 代价则节约了近 30 %.

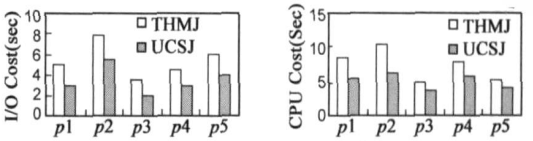

图6　比较UCSJ和THMJ计算性能

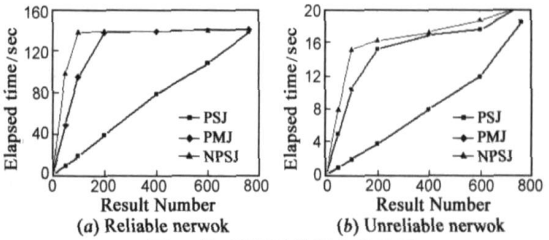

图7　第2组测试数据实验结果

在相同传输速率的条件下使用第 2 组测试数据实验,图 7(a) 和(b) 分别表示 PSJ、PMJ 和 NPSJ 在可靠网络和不可靠网络下的实验结果. 此时空间对象数目较少,两个输入数据集对应的 key-tuple 能全部放在内存中,因此,PSJ 在内存连接阶段不会出现溢出,其磁盘连接阶段完全退化为空间连接的提炼步骤;而 PMJ 和 NPSJ 没有使用 key-tuple 的内存结构,仍然会出现内存溢出,因此查询性能大大降低. 当输入数据集传输速率不同时的实验结果与此类似.

参考文献:

[1] Hass P J, et al. Ripple join for online aggregation[A]. In SIGMOD[C]. New York, USA, ACM, 1999. 287 - 298.

[2] Mokbel M F, et al. Hashr-merge Join: A non-blocking join algorithm for producing fast and early join results[A]. In ICDE[C]. USA: IEEE Computer Society, 2004. 251 - 263.

[3] Urhan T, et al. XJoin: A reactively scheduled pipelined join operator[J]. IEEE Data Engineering Bulletin, 2000, 23(2): 27 - 33.

[4] Dittrich J P, et al. Progressive merge join: A generic and non-blocking sort-based join algorithm[A]. In VLDB[C]. Hong Kong: VLDB Endowment, 2002. 299 - 310.

[5] Ives, Z G, et al. An adaptive query execution system for data integration[A]. In SIGMOD[C]. USA: ACM, 1999. 299 - 310.

[6] Luo G, et al. A non-blocking parallel spatial join algorithm [A]. In ICDE[C]. USA: IEEE Computer Society, 2002. 697 - 705.

[7] Boucelma O, et al. VirGIS: Mediation for geographical information system[A]. In ICDE[C]. USA, IEEE Computer Society, 2004. 855.

[8] Wilschut A N, et al. Dataflow query execution in a parallel main-memory environment [A]. In PDIS [C]. USA: IEEE Computer Society, 1991. 68 - 77.

[9] Tao Y F, et al. RPJ: producing fast join results on streams through ratebased optimization[A]. In SIGMOD[C]. USA: ACM, 2005. 371 - 382.

[10] Patel J M, et al. Partition based spatial-merge join[A]. In SIGMOD[C]. Canada: ACM, 1996. 259 - 270.

[11] Viglas S D, et al. Rate-based query optimization for streaming information sources[A]. In SIGMOD[C]. USA, ACM, 2002. 37 - 48.

[12] McCirmick J. Problem decomposition and data reorganization by a clustering technique[J]. Operations Research, 1972, 20 (5): 993 - 1009.

[13] 王小平, 曹立明. 遗传算法-理论、应用与软件实现[M]. 西安交通大学出版社. 2004 年.

[14] Xiao J, et al. Clustering non-uniform-sized spatial objects to reduce I/O cost for spatial-join processing[J]. The Computer Journal, 2001, 44(5): 384 - 397.

[15] 熊伟, 等. 空间连接处理中提炼步骤的遗传优化[J]. 电子学报, 2006, 34(6): 1069 - 1073.
Xiong Wei, ect. Gentic optimization to the refinement step of spatial join processing[J]. Acta Electronic Sinica, 2006, 34 (6): 1069 - 1073. (in Chinese)

[16] US Bureau of the Census. TIGER/Line Files [DB/OL]. http://www.census.gov, 2006-12 - 02.

作者简介:

唐桂芬 女, 1977 年生于湖北武汉. 解放军 61081 部队工程师, 2007 年在国防科技大学获博士学位. 研究方向为空间数据集成技术, GPS 导航技术.
E-mail: guifentang@nudt.edu.cn.

杨伟锋 男, 1978 年生于河北邢台. 空军指挥学院博士研究生. 研究方向为 GIS 与战场仿真.

黄双临 女, 1975 年生于北京, 解放军 61081 部队工程师, 硕士, 研究方向为 GPS 导航技术.

李 炜 男, 1975 年生于北京, 解放军 61081 部队工程师, 硕士, 研究方向为 GPS 导航技术.