一种避免数据偏斜的动态Hash连接方法*

洪晓光 王新军 董继润

(山东大学计算机系 济南 250100)

摘要 本文提出了一种新的动态Hash连接方法——DHJ(dynamic hash join),以解决并行数据库连接操作中的数据偏斜现象.为避免目前某些算法提出的预处理中隐含的高额费用,该方法在划分阶段通过增添附加桶的方法来平衡输出,然后依据计算确认哪些附加桶被映射到处理器上并确定处理器分配,在最后阶段完成连接.本文最后给出了该算法的性能分析.

关键词 连接,数据偏斜,动态Hash,桶,平衡输出.

中图法分类号 TP311.13

Hash连接在并行数据库查询中得到大量运用,许多文章对此进行了讨论,并给出了相应算法.[1~4]通常在连接之前,为减少参与连接的关系元组的数目,可先进行选择和投影操作,因此无法预知这些关系的分布情况,同时尚无一种Hash函数能够在此前提下得到均匀的工作负载.不平衡负载对并行操作会产生很大的负面影响,因此为了充分利用Hash连接算法,研究负载平衡是很重要的.数据偏斜导致不平衡负载有很多原因:连接属性值的不均匀分布会导致内在固有偏斜(Intrinsic Skew),处理器间隐含着的连接中的负载不平衡将会导致划分偏斜(Partition Skew).因此负载的不平衡可能发生在连接算法的不同阶段.

有关偏斜处理的连接算法已有许多,如GRACE[1],它们大多是通过在源数据分布中获得知识以减少连接阶段不平衡的发生,为此需要进行有效的预处理.如文献[2]中的BSJ算法是在数据划分处理结束之后,在已知桶的大小的基础上再将桶分配到各处理器上.Wolf在文献[3]中提出了一种排序Hash算法,读取两个源关系,运用局部选择和投影,并将结果当作一个粗(Coarse)Hash桶的集合回写,另外基于一个细化(Finer)Hash函数的统计被保存在每个桶中,然后一个协处理器收集所有finer和coarse的统计集合,并且计算出一个桶的分配给各处理器.文献[4]中给出了一种动态Hash连接算法,它是通过在划分阶段动态计算各处理器的当前负载量,发现不平衡即进行调整,划分阶段结束即进入连接阶段,不象前面两种方法在划分和连接阶段之间增加一个schedula阶段来平衡负载.不过,该方法虽能对全局的数据进行平衡,但它要求更多的数据通信和消息传递,因为它在划分阶段随时计

---

* 本文得到国家自然科学基金资助.作者洪晓光,1964年生,讲师,主要研究领域为分布式数据库,并行数据库.王新军,1968年生,讲师,主要研究领域为分布式数据库,并行数据库.董继润,1935年生,教授,主要研究领域为分布式数据库,并行数据库,演绎数据库,面向对象数据库.

本文通讯联系人:洪晓光,济南 250100,山东大学计算机系

本文1996-10-16收到修改稿

算和调整向处理器的输出.

本文算法综合上述几种方法的优点,在划分阶段动态计算局部负载量,利用附加桶调整对处理器的分配,避免了上述几种方法的缺陷. 本文第 1 节给出新的 Hash 方法;第 2 节对该方法进行分析,并和其它方法进行比较;第 3 节讨论执行中的一些实验结果.

1　动态 Hash 连接方法——DHJ

有关 Hash 方法的一般介绍在此不再赘述. 我们提出的 DHJ(dynamic hash join)方法采用并行结构中的 SN 模式,有一部分处理器拥有自己的磁盘存储,主要用来保持和组织源关系数据,我们称之为数据服务器. 另一部分则只作为一般处理器,用作查询处理. 下面是 DHJ 方法的描述:

假设 R 和 S 是两个要进行连接的关系,二者相比,S 较小,它们被水平划分,并且均匀分布到了各数据服务器上,设 R_m 和 S_m 是 R 和 S 在数据服务器 m 上的部分,$|R_m|$ 和 $|S_m|$ 是关系基数,又设数据服务器的个数是 $D+1$,其标识分别为 $0,1,2,\ldots,D$,一个 Hash 函数初始划分每个 R_m 到 $B+1$ 个桶,$R_m^0,R_m^1,R_m^2,\ldots,R_m^B$ 桶的序号为 $0,1,2,\ldots,B$.

在某一时间点,数据服务器 m 已经划分了 Q 个元组,每个桶中分别有 $Q_m^0,Q_m^1,Q_m^2,\ldots,Q_m^B$ 个元组. 我们有这样几个分配比例:

$$q_1=\frac{Q_m}{|R_m|}\qquad q_2^i=\frac{Q_m^i}{Q_m}\qquad q_3^i=\frac{Q_m^i}{|R_m|}$$

其中 $Q=\sum_{i=0}^{B}Q_m^i$(i 为桶号).q_1 反映了当前 R_m 中已被划分的元组占总元组数的比例;q_2^i 反映了当前第 i 个桶中元组数占已被划分的总元组数的比例;q_3^i 反映了当前第 i 个桶中元组数占总元组数的比例. 依据上述三等式,我们可对已相对较满的桶进行平衡处理直到 R_m 全部划分完成,再将其它处理器的划分信息综合起来得到一个负载平衡的桶分配. 最后,根据某种策略完成连接阶段. 与其它 Hash 方法相比,这里要增加对附加桶的处理.

下面我们将仔细说明上述处理方法中桶平衡问题和处理器分配方案以及连接阶段附加桶的处理. 这两点也是本文的新意所在.

2　平衡处理

正常情况下,R_m 应当被 Hash 分配到 $B+1$ 个桶中,但随着划分阶段的进行,根据上述 3 个划分比例,我们可以知道哪个桶负载比较大. 假设在某一时间点,对第 i 个桶而言 q_1 和 q_2^i 分别满足以下条件:

$$\frac{1}{D+1}\left(\sum_{i=0}^{D}q_2^i\right)>\beta*\alpha \tag{1}$$

和

$$q_1=\frac{Q_m}{|R_m|}<\beta' \tag{2}$$

在(1)式中,$\alpha=\frac{|R_m|}{B+1}$表示平均每个桶应分得的元组数,β 为常数,随时间点的变化而变化,是一实验值,可取大于 0.5 的值. 如在此时间点我们取 $\beta=0.6$. (2)式中的 β' 也是常量,表示当前划分的数据量,也随时间点变化而变化,可取小于等于 0.5 的值. 在此时间点取 β'

$=0.5$.

上述条件表明已划分的数据量还不到一半时,第 i 个桶中就已有 2/3 以上的元组数. 这时我们将为其动态生成一个新的附加桶,标号为 i'. 后面的元组将全部往 i' 桶中存放. 照此方法,到划分阶段完成,即当 $q_1=1$ 时,桶的总个数就可能比 $B+1$ 多出若干个. 附加桶与原桶之间通过某种机制衔接,但由于我们的动态调整,每个桶中的数据应当比较均匀.

上面我们只给出了划分阶段中每个数据服务器独立完成的桶的划分,在划分过程的同时,我们也象文献[4]中的 DBJ 方法那样找一个函数 $f(x)$,将每个桶映射到一个处理器标号上. 当某个桶中数据出现不平衡时,该桶即被附加桶替代,而该附加桶则可能被映射到另外的处理器,即与原桶被分配的处理器不相同. 但该处理器必须与原桶对应的处理器有对应的标记,因为在连接阶段第 2 个关系的桶需向这两个处理器发送.

假设有 $P+1$ 个一般处理器,标号分别为 $0,1,2,\ldots,P$,则平均来说每个处理器上应被映射 $\gamma=[(B+1)/_{(P+1)}]$ 个桶. 但由于在划分时出现偏斜,增加了附加桶,因此应有 $\gamma=[(B+1+L)/_{(P+1)}]$,这里 L 是附加桶的个数. 也就是说,在开始划分阶段,桶的真正个数不是 $B+1$ 而是 $B+1+L$. 而 Hash 函数初始时对 R_m 的划分则是映射到 $0,1,2,\ldots,B$ 桶上. L 不会很大,因为若 $L=B+1$ 就表示每一个桶都被增加了一个附加桶,这是很不现实的,因为不可能所有值都发生偏斜.

与一般 Hash 方法不同,由于数据偏斜,本文算法在划分阶段有附加桶产生,而且附加桶与对应的原桶并不一定被分配映射到相同的处理器,但如前所述,这些处理器之间都有相互对应的标志,很容易衔接. 因此在 probing 阶段,关系 S 的一个桶在被分到某一处理器 k 时,如果发现有另外的处理器 k' 与其有对应关系,那么该桶将同时被传送到 k'. 在 k,k' 都进行 probing 操作. 最后输出连接结果.

3 分 析

这里我们利用文献[4]中的一些相同假设并给出了费用函数,在下面分析中用到的诸如 T_{DHJ},T_{part} 等符号的意义参见文献[4]. 假设通讯、磁盘 I/O 和 CPU 计算是完全重叠的;两个源关系有相同的大小和初始均匀分布;并行结构为 SN 结构. 设 N 为处理器个数,并假定连接操作时在偏斜条件下,关系被均匀分布到所有处理器上. 只有一个例外,它有额外的数据,比其它处理器多出 $\sigma\times 100\%$ 的元组. 设 $|P_s|=\dfrac{|R|}{1+(N-1)(1-\sigma)}$,$|P_u|=\dfrac{(1-\sigma)|R|}{1+(N-1)(1-\sigma)}$,在这里 $|P_s|$ 和 $|P_u|$ 分别是偏斜划分和不偏斜划分的基数. 我们有以下等式:

$T_{DHJ}=T_{part}+T_{join}$ 即总费用等于划分与连接阶段的费用之和.

$T_{part}=\max(T_{dio}+T_{dcpu}+T_{dcomm})$ 划分阶段的费用为 I/O,CPU 运算和通讯费用中最大者.

$T_{dio}=\dfrac{t|R|}{N\omega_{io}}+\dfrac{\frac{t|R|}{N}-M}{\omega_{io}}$ I/O 费用为从磁盘读第 1 个关系的费用和超出内存后回写磁盘的费用之和.

$T_{dcpu}=\frac{2|R|}{N}\times\frac{Icpu}{u}$ CPU 的费用为划分阶段处理所有元组的费用.

$T_{dcpu}=2(\frac{|R|}{N}\cdot\frac{N-1}{N}\cdot\frac{t}{W_{Com}})+T_{dextra}$ 通讯费用为 $N-1$ 个一般传输与因为偏斜而产生的额外传输的费用之和. 其中 $T_{dextra}=\frac{(|P_s|-|P_u|)}{\omega_{comm}})*\beta*t$ 表示额外数据被移动到其它处理器时的费用，其中 β 表示目前已划分的数据量，它可以是我们前面提到的 3 个比例中的 q_1，这里我们可以取在中间 50%，即数据已划分完一半的位置上.

$$T_{join}=\left\lceil\frac{t|R|}{NM}\right\rceil(T_{jhash}+T_{jprobe})$$

表示连接阶段的费用由修建 hash 表和进行 probing 操作费用的总和构成.

其中 $$T_{jhash}=\max(\frac{M}{\omega_{io}},\frac{M}{t}\cdot\frac{I_{cpu}}{u})$$

表示建 hash 表的费用由 I/O 费用与 CPU 费用中的大者决定.

$$T_{jprobe}=T_{jlocal}+T_{jremot}$$

probing 操作的费用等于匹配本地处理器数据与匹配异地处理器数据费用之和.

其中 $$T_{jlocal}=\max(\frac{M}{\omega_{io}},\frac{M}{t}\cdot\frac{I_{cpu}}{u})\qquad T_{jremote}=\frac{|R|}{N}\cdot\frac{M}{\omega_{comm}}$$

从上述分析可以看出，本文给出的算法比文献[4]提出的 DBJ 算法有一个很大的优点，即在划分阶段基本没有数据通讯，而只需对附加桶采用同样的处理器分配规则进行分配.

4 讨　论

本算法的核心是在数据出现不平衡时通过附加桶予以保存，同时对原桶的处理器映射将被改变到新的处理器，这样实际象文献[4]中 DBJ 一样，也只需经过两个阶段即可完成，但它不象 DBJ 那样需要大量的数据移动和消息传递，因而效率是非常高的. 尤其在表中多个数据发生偏斜时效果更好. 本文前面提到的几个比例等式中的系数值，尤其是 β 尚需进一步通过实验加以确认.

参考文献

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

2 Kitsuregawa M, Ogawa. Bucket spreading parallel hash: a new robust. Parallel Hash Join Method for Data Skew in the Super Database Computer (SDC). In: Proc. of 16th VLDB Conference, Brisbane, Australia, 1990. 210～221.

3 Wolf J L, Dias D M, Yu P S *et al*. An effective algorithm for parallelizing hash join in the presence of data skew. IBM T. J. Watson. Research Center Tech Report RC 1990.

4 Zhao X, Johnson R G. DBJ—a dynamic balancing hash join algorithm in multiprocessor database systems. LNCS 779, Advances in Database Technology-EDBT'94, 1994. 301～308.

A DYNAMIC HASH JOIN ALGORITHM TO AVOID DATA SKEW

HONG Xiaoguang WANG Xinjun DONG Jirun

(*Department of Computer Science Shandong University Ji'nan* 250100)

Abstract In this paper, a new hash join algorithm——DHJ (the dynamic hash join) is proposed to resolve the problem of skewed data in the join operation in parallel database. The objective of the algorithm is to avoid the high cost of processing inherent in some early work. Additional buckets are used in the algorithm to balance output during the data partition. Then they are mapped to different processors before the completion of the join operation. The performance analysis of the algorithm is provided in this paper.

Key words Join, data skew, dynamic hash, bucket, balance output.

Class number TP311.13