ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.17, No.4, April 2006, pp.770−781
DOI: 10.1360/jos170770

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

一种并行 XML 数据库分片策略*

王国仁+, 汤 南, 于亚新, 孙 冰, 于 戈

(东北大学 信息科学与工程学院,辽宁 沈阳 110004)

A Data Placement Strategy for Parallel XML Databases

WANG Guo-Ren+, TANG Nan, YU Ya-Xin, SUN Bing, YU Ge

(College of Information Science and Engineering, Northeastern University, Shenyang 110004, China)

+ Corresponding author: Phn: +86-24-83681250, Fax: +86-24-23893138, E-mail: wanggr@mail.neu.edu.cn, http://mitt.neu.edu.cn

Wang GR, Tang N, Yu YX, Sun B, Yu G. A data placement strategy for parallel XML databases. *Journal of Software*, 2006,17(4):770−781. http://www.jos.org.cn/1000-9825/17/770.htm

Abstract: This paper targets on parallel XML document partitioning strategies to process XML queries in parallel. To describe the problem of XML data partitioning, a concept, intermediary node, is presented in this paper. By a set of intermediary nodes, an XML data tree can be partitioned into a root-tree and a set of sub-trees. While the root-tree is duplicated over all the nodes, the set of the sub-trees can be evenly partitioned over all the nodes based on the workload of user queries. For the same XML data tree, there are a number of intermediary nodes sets, and different intermediary nodes sets will generate different partitions. It can be evaluated if a partitioning is good based on the workload of user queries. It is obviously an NP hard problem to choose an optimal partitioning. To solve this problem, this paper proposes a set of heuristic rules. Based on the idea described above, this paper designs and implements an XML data partitioning algorithm, WIN, and the extensive experimental results show that its speedup and scaleup performances outperform the existing strategies.

Key words: parallel database; XML document; workload; data partitioning; intermediary node

摘 要: 主要研究 XML 文档的并行数据分片策略,以便能够并行处理 XML 查询.为了描述 XML 数据分片,提出了媒介节点的概念.一组媒介节点的集合可以将一棵 XML 数据树分割成一棵根树和一组子树的集合:根树将在所有站点中复制;而子树集合则可以根据用户查询的工作负载被均匀地分片到各个站点中.对于同一棵 XML 数据树,会有很多种媒介节点的集合;而不同的媒介节点集合会产生不同的数据分片结果.然后,依据各个数据分片中的用户查询工作量是否均衡,来衡量一个分片的好坏.选择一组最佳的媒介节点集合是一个 NP-hard 问题.为了解决此问题,设计了一组启发式优化规则.基于这一思想,提出并实现了一种基于媒介节点的 XML 数据分片算法 WIN(workload-aware intermediary nodes data placement strategy).大量实验结果证明:WIN 算法的性能要优于以往的并行 XML 数据分片策略.

关键词: 并行数据库;XML 文档;工作负载;数据分片;媒介节点

* Supported by the National Natural Science Foundation of China under Grant Nos.60273079, 60473074 (国家自然科学基金); the National Research Foundation for the Doctoral Program of Higher Education of China under Grant No.DP0345710 (国家教育部博士点基金)

Received 2004-01-21; Accepted 2005-10-20

中图法分类号: TP311 文献标识码: A

随着 XML 的广泛应用和数据规模的急剧膨胀,其查询处理日趋复杂,单处理机环境由于 I/O 和主存限制,整体处理能力存在着极限.并行数据库系统为大型 XML 文档的高效处理提供了有效的解决途径.在并行数据库系统中,数据分片是影响并行数据库系统整体性能的重要因素.

并行数据库研究者已经提出了很多种经典的数据分片策略.在并行关系数据库中,主要有一维数据划分方法和多维数据划分方法:一维数据划分方法包括 Round-Robin,Hash,Range 和 Hybrid-Range-Partition;多维数据划分方法包括 k-d-B-Tree,hB-Tree[1]和 X-Tree[2]等.在并行对象数据库中,各对象之间通过指针任意引用,对象之间拓扑结构呈图状.因此,并行对象数据库中分片算法都是图分片算法.Metha 和 Dewitt[3]对无共享并行关系数据库系统进行了详细的模拟研究;He 和 Yu[4],Ghandeharizadeh[5]提出了不同的并行对象数据库分片算法.

XML 是一种半结构化数据,通常表示成数据树的形式.与关系数据库中结构化的表结构相比,XML 包含复杂的嵌套结构,而将 XML 转换成关系数据表的形式会造成大量数据冗余.因此,并行关系库中传统的数据分片策略不能很好地适用于并行 XML 数据库;与面向对象数据库相比,XML 的树结构比对象库中图结构更直观和清楚地表示了元素之间的关系.因此,并行对象数据库中的数据分片策略也不能很好地适合于并行 XML 数据库.基于路径表达式的 XML 查询语言,如 XPath[6]和 XQuery[7],也给并行 XML 数据库的研究工作提出了新的挑战.

文献[8]提出了两种并行 XML 数据库的分片算法:NSNRR 和 PSPIB.NSNRR 的基本思想是:以元素为粒度,根据轮循法将具有相同名字的元素存储在同一台处理机上.在查询时,这种分片算法能够很好地利用管道并行性.但是,该分片算法造成大量数据倾斜,随着文档的增大,该算法加速比和缩放比性能明显降低.PSPIB 是一种以路径实例为粒度的分片算法.从根到叶子的一条完整路径分支上所有元素节点的元素标记构成的一个有序集合或者其子集就是一个路径模式.从根到叶子的所有元素节点组成的有序序列称为一个路径实例.XML 查询语言主要是基于结构的,因此,PSPIB 的主要思想是将具有相同路径模式的路径实例平均打散到各个处理机.由于 PSPIB 将相同模式打散的基本思想与本文提出的 WIN(workload-aware intermediary nodes data placement strategy)类似,我们在这里给出一个简单的 PSPIB 分片算法实例,以便性能分析中进行比较.图 1 是一个轻量级 XML 数据树,假设有两台处理机,图 2 显示了 PSPIB 分片结果.实心圆元素代表该元素被复制存储在两台处理机上;实线圆代表该元素存储在处理机 1;虚线圆代表该元素存储在处理机 2.

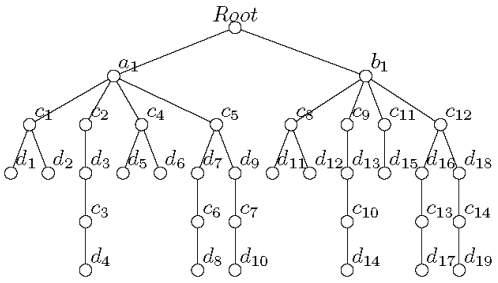

Fig.1 A light weighted XML data tree

图 1 一棵轻量级 XML 数据树

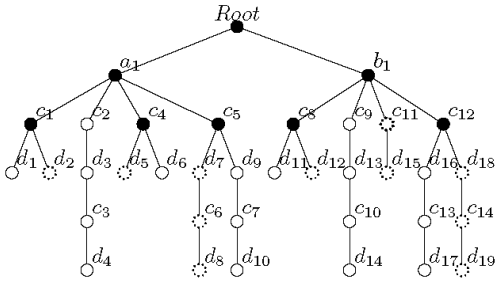

Fig.2 Partitioning result of PSPIB algorithm

图 2 PSPIB 算法数据分片结果

与上述思路不同,对一棵 XML 数据树进行分片的另一种思路是以子树为单位来进行数据树的分片.观察一棵 XML 数据树,树的上层节点个数很少,而这些节点往往出现在大部分 XML 查询中,因此,很自然地应该以牺牲少量磁盘空间为代价,将这些节点复制到所有站点上,称为复制根树;而对那些访问频率比较低且数据量又比较大的子数据树,则应该分片到各个站点上,称为分片子树.在每个站点将分片子树与复制根树的节点合并构成一棵局部数据树.各个站点以数据树形式进行存储的目的就是保证 XML 数据的局部完整性,降低各个站点之间的网络传输工作量,从而使并行系统并行性得到更好的发挥.由于不同的根树和相应的子树集会给并行系统带来

不同的复制代价和并行性,因此,必须考虑在复制代价和并行性方面进行权衡,以达到两个目标:一个是各个站点工作量分布尽量平均;另一个是系统响应时间尽量小.为了描述这一 XML 数据分片问题,本文提出了媒介节点的概念.一组媒介节点的集合可以将一棵 XML 数据树分割成一棵根树和一组子树的集合.根树将在所有站点中复制,而子树集合则可以根据用户查询的工作负载被均匀地分片到各个站点中.对于同一棵 XML 数据树,会有很多种媒介节点的集合,而不同的媒介节点集合会产生不同的数据分片结果.然后,依据各个数据分片中的用户查询工作量是否均衡来衡量一个分片的好坏.选择一组最佳的媒介节点集合是一个 NP-hard 问题.为了解决这一问题,本文设计了一组启发式优化规则.基于这一思想,本文设计并实现了一个基于媒介节点的 XML 数据分片算法 WIN.

本文第 1 节说明基于媒介节点的分片原理.第 2 节详细描述基于媒介节点的并行 XML 数据分片算法 WIN.第 3 节进行性能评价与分析.第 4 节总结全文.

1 基于媒介节点的分片原理

在本节中,我们首先提出媒介节点的概念,然后给出基于媒介节点的数据分片概念,最后给出查询负载的估算模型.

1.1 媒介节点

我们将一棵 XML 数据树 *XT* 上的元素节点分成 3 类:复制节点 DN(duplicate nodes),媒介节点 IN(intermediary nodes)以及普通节点 TN(trivial nodes).这 3 种节点具有以下一些性质:

(1) *DN* 的祖先节点是 *DN* 或者 *NULL*;

(2) *IN* 的祖先节点是 *DN* 或者 *NULL*;

(3) *TN* 的祖先节点是 *IN* 或 *TN*;

(4) *DN* 全复制存储在各个处理机,而 *IN* 和 *TN* 节点是全局唯一的;

(5) $\{IN\}\cap\{DN\}=\varnothing;\{IN\}\cap\{TN\}=\varnothing,\{DN\}\cap\{TN\}=\varnothing$;

(6) $\{IN\}\cup\{DN\}\cup\{TN\}=XT$.

DN 节点集合构成了需要在各个站点复制的根树,而 *IN* 节点集合就是待分片子树根的集合.不同的媒介节点集合会产生不同的数据分片结果,因此,如何找到一组合适的媒介节点集合是本文中新分片算法的关键.对于同一棵 XML 数据树,会有很多种媒介节点的集合.如图 3 所示,$\{b,c\}$是一组媒介节点集合,而$\{b,f,g\}$,$\{d,e,c\}$等也是媒介节点集合,媒介节点集甚至可以包含叶子节点,如$\{b,l,m,g\}$,$\{b,f,n,o\}$,$\{b,l,m,n,o\}$等,媒介节点集的极限情况是所有媒介节点均为叶子节点$\{h,i,j,k,l,m,n,o\}$.选择一组最佳的媒介节点集合是一个 NP-hard 问题.

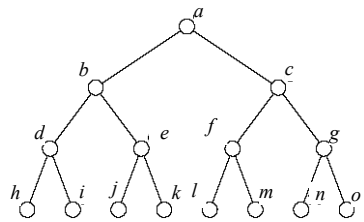

Fig.3 Illustration of intermediary nodes

图 3 媒介节点示意图

1.2 基于媒介节点的数据分片

基于媒介节点的数据分片问题可以描述为:对于一棵 XML 数据树,给定一组媒介节点集合和相应的根树,首先,将根树在各个站点上复制存储;然后,将媒介节点集合中的媒介节点根据相同父节点进行分组——在每一组中,将媒介节点构成的子树集合根据子树工作量均衡的原则分配到不同站点,然后,在各个站点将根树与该站点上分片子树合并生成局部 XML 数据树.结合 XML 文档特殊结构和 XML 查询正则路径表达式的特征,分组

目的是使同一组内子树具有相似的结构,从而提高查询并行性.

基于媒介节点的数据分片技术涉及到两个关键技术:一个是空间搜索问题(即找到所有可能的媒介节点集的方法);另一个是分片结果(一个媒介节点集表示一种分片)的评价问题.衡量一个媒介节点集(即一种分片)的好坏主要有两点:首先是分片后,各个站点上的查询工作量分布是否均衡;另一个是复制代价.高复制将会带来好的并行性,但无疑也增加了整个并行数据库系统的复制代价,极限情况就是全复制,这显然不是我们需要的结果.因此,第二个衡量标准就是:复制代价的增加是否低于其带来的利益.

根节点以及叶子节点集合是媒介节点集的两种极限情况.空间搜索的一种策略是:从根节点开始,对媒介节点集中的某个节点用其子节点集合进行扩展,以找到最优解.假设有一组媒介节点集合 IN,对其进行扩展后会产生$|IN|$个媒介节点集 $IN_1,IN_2,\ldots,IN_{|IN|}$,其中$|IN|$为媒介节点集 IN 中节点的个数.然后依次对 $IN_1,IN_2,\ldots,IN_{|IN|}$进行扩展,这显然是一个 NP 爆炸问题.因此,为了减少解空间搜索范围、提高算法效率,在对媒介节点集 IN 进行扩展时,设计了两个启发式规则来限制需要继续扩展到媒介节点集数目,这两个规则如下所示:

(1) 如果 IN_i 与 IN 相比能带来更大的数据倾斜,类似于爬山算法思想,则认为:在这种情况下没有继续扩展的意义,因此不对 IN_i 进行继续扩展;

(2) 即使 IN_i 能比 IN 带来更好的数据分布,但它所带来的复制代价的增加要高于它所能带来的利益,在这种情况下也不进行扩展.

1.3 查询负载估算模型

在基于工作负载均衡的分片策略中,工作量估算是十分重要的一个环节,因为工作量估算的准确度会直接影响分片结果,从而影响并行数据库系统的性能.这一部分我们提出了 XML 数据树的工作量估算公式.这些公式不仅基于 XML 文档的固有结构,而且考虑到了查询语句、索引结构和查询策略的采用.不同于文献[9,10]中索引正则路径表达式的路径索引结构,我们采用了 XML 数据库中基本的 PNameIndex,NameIndex, TextValueIndex 和 AttributeValueIndex[11].至于一些关键问题,如祖先后代和 twig 查询问题,虽然在文献[12−14]中提出了十分有效的解决方案,但我们只采用查询重写和哈希连接技术.这是因为,我们的分片方法应该具有普遍性,而不是针对某一类特殊问题和特定系统.表 1 列出了本文需要使用的变量.

Table 1 Variables list

表 **1** 变量列表

Variable	Description
n_a	The number of element a
N	The number of sites
$n_{a\,\theta\,b}$	The result number of operation $a\theta b$
S_a	The size of element a
Q_i	The i-th query in statistics
f_i	Query frequency of Q_i
$\delta_{a\theta b}$	Selectivity of join operation $a\theta b$
$t_{I/O}$	Avg. disk access time
v_{net}	Net speed
W_a	Workload of a data tree of root element a
$W_{label(a,b)}$	Workload of $label(a,b)$
$F_{label(a,b)}$	Frequency of $label(a,b)$

接下来将给出我们进行工作量估算的基本模型公式.一棵 XML 数据树工作量定义如下:

定义 **1**. 选择率 $\delta_{a\theta b}$ 定义如下:

$$\delta_{a\theta b}=n_{a\theta b}/n_b \tag{1}$$

选择率反映了结构连接运算θ后的结果数 $n_{a\theta b}$ 与 b 个数的百分比.

定义 **2**. 边 $label(a,b)$出现频率定义如下:

$$f_{label(a,b)}=\sum_{i=1}^{n}((Q_i \lhd label(a,b)).f_i) \tag{2}$$

其中 $Q_i \lhd label(a,b)$代表边 $label(a,b)$在查询 Q_i 中累计出现的次数.

定义 **3**. 在 XML 数据树中,从一个节点 *a* 到另一个节点 *b* 构成的边的工作量定义如下:

$$W_{label(a,b)}=\left(\left(\frac{3n_a+3n_b+n_b.\delta_{a\theta b}.(S_a+S_b)}{S_{page}}\right).t_{I/O}+\frac{n_b.\delta_{a\theta b}.(S_a+S_b)}{v_{net}}\right).f_{label(a,b)} \tag{3}$$

大括号中:前一部分计算 I/O 工作量,包括建立哈希表所需 I/O 次数 $\frac{3n_a+3n_b}{S_{page}}$ 和查询结果存储所需 I/O 次数 $\frac{n_b.\delta_{a\theta b}.(S_a+S_b)}{S_{page}}$;后一部分是网络传输工作量.由于 CPU 工作量的数量级要比磁盘 I/O 和网络传输高,所以我们忽略了 CPU 工作量.

定义 **4**. 根为 *a* 的数据树工作量定义如下:

$$W_a=\begin{cases}0, & a是叶子节点\\ \sum_{b\in chilr(a)}W_b+W_{label(a,b)}, & a是非叶子节点\end{cases} \tag{4}$$

表 2 列出了在进行工作量估算时实际使用的重要的系统参数.本文使用一个具有 5 台处理机的无共享并行系统结构,一个百兆高速局域网,采用的面向对象数据库中对象标识(oid)大小为 4 个字节.

Table 2 System parameters

表 **2** 系统参数

System characteristic	Value
Number of sites	5
Aggregate network bandwidth (MB/s)	100
Main memory per site (MB)	128
Average I/O cost per page (ms)	10~20
Page size (KB)	4
Size of object identity (Byte)	4

2 基于媒介节点的数据分片算法

根据第 1.1 节的节点分类,由于 *IN* 节点和 *TN* 节点是唯一的,因此,*IN* 和 *TN* 组成的子树相互不重叠.如果我们能在一棵 XML 数据树 *XT* 中将 *IN* 和 *TN* 组成的这些不重叠子树进行分组,每一组中子树包含相似的结构和相当的工作量,然后将每组内子树按工作量均衡的原则分片到不同处理机上.这种分片策略可以保证数据树结构的局部完整性,可以使并行数据库系统获得很高的查询无关并行性、较低的通信开销和数据倾斜.接下来的部分,我们会详细描述如何找到一组最合适的 *IN*,并根据以上基本原则进行数据分片.

首先,我们介绍 WIN 数据分片算法,见算法 1.

算法 **1**. *WIN*(*XT*,*N*).

(1) *interList*←{{*root*}}

(2) *finalIN*←*NIL*

(3) WHILE *interList*≠*NIL*

(4) do *IN*←*interList*.*popFront*()

(5) IF *Benefit*(*IN*,*finalIN*)

(6) then *finalIN*←*IN*

(7) *Expand*(*interList*,*IN*)

(8) RETURN *Partition*(*finalIN*)

该算法描述如何将一棵 XML 数据树进行数据分片.其中,数 *XT* 代表一棵 XML 数据树,*N* 是处理机个数,*interList* 代表 *IN* 节点集合的集合,*finalIN* 代表我们找到的 *IN* 节点集合.这个算法可以简短描述如下:

(1) 当 *interList* 不是 *NULL* 时,从 *interList* 得到一个 *IN* 集;

(2) 如果新的 *IN* 能比 *finalIN* 带来更多利益,用新的 *IN* 代替 *finalIN*;

(3) 将当前 *IN* 节点集进行扩展并将扩展结果插入 *interList*.

其中,*Partition*()描述如何将一个给定 *IN* 节点集的 XML 数据树进行分片;*Expand*()描述如何对 *interList* 进行扩展;*Benefit* ()计算是否扩展后的媒介节点集可能带来更高效率.

接下来,我们介绍 *Partition*()算法,见算法 2.该算法描述如何将一个给定 *IN* 节点集的 XML 数据树进行数据分片.为了获得高的查询无关并行性和低的数据倾斜,结合 XML 文档的特殊结构和 XML 查询正则路径表达式的特征.我们首先将具有相同父亲的 *IN* 节点集进行分组,目的是使每一组内 *IN* 节点构成子树具有相似结构;然后,每一组 *IN* 节点子集根据工作量均衡的原则分配到不同处理机,并在各个处理机上进行子树合并操作.本算法省略了各个处理机上的子树合并操作,合并操作主要是在处理机上建立 *DN* 节点集和 *IN* 节点集之间的父子关系.这个算法执行完毕后,每个站点 P_i 形成一棵局部 XML 数据树 XT_i.

算法 **2**. *Partition*(*IN*).

(01) $DN \leftarrow \bigcup_{E\in IN} Ancessor(E)\ DN$

(02) duplicate each node in *DN* to all sites $P_1,P_2,\ldots,P_N$

(03) for each element $E\in IN$

(04) do group *E* according to *getParent*(*E*)

(05) for each group *G*

(06) do $num\leftarrow 1$

(07) for each element $E\in G$

(08) do $workload_{AVG}\leftarrow W_{(getParent(E))}/N$

(09) $workload_{ADD}\leftarrow 0$

(10) if $workload_{ADD}<workload_{AVG}$

(11) then $workload_{ADD}\leftarrow workload_{ADD}+W_{(E)}$

(12) else $workload_{ADD}\leftarrow 0$,

(13) $num\leftarrow num+1$

(14) distribute *Subtree*(*E*) to P_{num}

(15) return distribution of *IN*

然后,我们介绍 *Expand*()算法,见算法 3.该算法描述如何对 *interList* 进行扩展,每次扩展只对 *interList* 中的一个 *IN* 节点集进行操作,扩展一共进行|*IN*|次(|*IN*|代表 *IN* 节点集所含节点个数),每次取出原 *IN* 节点集中一个节点并加入该节点所有孩子节点构成新的 *IN* 节点集,如果新的 *IN* 节点集可以比原 *IN* 节点集带来更好的效益,就将新 *IN* 节点集插入 *interList*,比较效益算法定义如算法 4 中的 *Benefit*()算法.

算法 **3**. *Expand*(*interList*,*IN*).

(1) for each node $in\in IN$

(2) do $IN'\leftarrow IN-in\cup in.getChildren()$

(3) if *Benefit*(*IN*′,*IN*)

(4) then *interList*.*pushBack*(*IN*′)

(5) return *interList*

并行数据库系统中,从所有可能的分片方法中选出最优分片方法是一个 NP-hard 问题.因此,在计算新的媒介节点集 IN_1 是否比老的媒介节点集 IN_2 带来更高利益时,算法 *Benefit*()利用了两个启发式规则.

算法 **4**. *Benefit*(IN_1,IN_2).

(1) *Partition*(IN_1)

(2) $w_1,w_2,\ldots,w_N$ represents IN_1' s workload in site P_1 to P_N

(3) *Partition* (IN_2)

(4) $w_1', w_2', \ldots, w_N'$ represents IN_2' s workload in site P_1 to P_N

(5) $w_{IN_1} \leftarrow \sum_{I=1}^{N} w_i, w_{IN_2} \leftarrow \sum_{i=1}^{N} w'_i$

(6) $Avg_1 \leftarrow W_{IN_1}/N$, $Avg_2 \leftarrow W_{IN_2}/N$

(7) $\mathrm{Max}_1 \leftarrow \max(w_1, w_2, \ldots, w_N)$, $\mathrm{Max}_2 \leftarrow \max(w'_1, w'_2, \ldots, w'_N)$

(8) $Ben_1 \leftarrow \mathrm{Max}_1 - Avg_1$, $Ben_2 \leftarrow \mathrm{Max}_2 - Avg_2$

(9) if $Ben_1 \geq Ben_2$

(10) then return false

(11) else $\Delta_{ben} \leftarrow \mathrm{Ben}_2 - \mathrm{Ben}_1$

(12) $\Delta_{dup} \leftarrow (W_{IN_1} - W_{IN_2})/N$

(13) return $(\Delta_{ben} - \Delta_{dup}) > 0$

从 *Expand* ()算法中我们可以观察到:每一次对 *IN* 节点集进行扩展之后,新构成的 *DN* 节点集中元素就会增加.这样造成的趋势就是:随着进一步的扩展,各台处理机工作量分布渐渐趋于平均,但每个处理机上复制节点个数逐渐增多,并行系统复制代价增大,每台处理机工作量增加,系统响应时间变大.极限情况是全复制,而全复制会导致系统负担过大.因此,我们采用了前文所示的两个启发式规则来加速解空间的搜索并避免全复制现象.

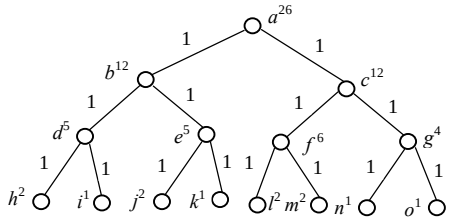

Fig.4 Illustration of workload
图 4 工作量示意图

下面,我们用实例说明 WIN 的工作流程.假设有两台处理机,图 4 给出 XML 数据树边和点的工作量,为了解释清晰,我们使用的是虚拟数据.边上数字代表计算出的每条边的工作量,每个节点上标数字代表该节点对应子树的工作量.WIN 算法流程如下:首先,以根{*a*}作为初始 *IN* 及候选集.此时,只有一个以 *a* 为根的子树,将它分配到处理机 1,此时两台处理机最大工作量为 26;当我们扩展{*a*},得到{*b*,*c*},将以 *b* 为根的子树分配到处理机 1,以 *c* 为根的子树分配到处理机 2,其余部分复制到 2 台处理机.此时,两台处理机最大工作量为 14<26,因此,{*b*,*c*}作为新候选 *finalIN*;依此类推,当我们扩展 *b*,得到{*d*,*e*,*c*},以 *d*,*e* 为根的子树分配到处理机 1,以 *c* 为根的子树分配到处理机 2,其余部分复制.此时,两台处理机最大工作量为 16>14,因此,我们不对{*d*,*e*,*c*}继续进行扩展.同理,扩展{*b*,*c*}中的 *c* 得到{*b*,*f*,*g*},最大工作量 16>14,仍不继续扩展.此时,找到最优 *IN* 为{*b*,*c*}.然后进行子树分配及复制,WIN 算法运行终止.

3 性能分析

由于基于工作负载均衡的并行 XML 数据分片算法 WIN 和基于路径模式的路径实例分片算法 PSPIB 都是将路径实例平均分片在不同处理机上(其中 WIN 基本思想是基于工作量均衡而 PSPIB 基于路径实例平均),因此,为了对这两种分片算法进行比较分析,我们在一个本地化 XML 数据库上完成了这两种分片策略的数据分片,并完成了查询性能测试.本节,我们首先介绍测试平台及数据集,然后对两种分片算法的各类查询进行性能分析.

3.1 测试平台及数据集

我们使用 6 台 PC 机组成一个同构的无共享 NOPC 环境进行测试.每台 PC 机拥有 CPU 为 PIII 800MHz,硬盘为 20G,内存为 128M,10/100M 自适应网卡.操作系统均为 Sun Unix (Solaris 8.0),交换区(swap)大小为 1.5G.各台 PC 通过一个高速 HUB(100Mbps)连接在一起,每台 PC 机采用一个本地化 XML 数据库管理系统来存储数据,它通过 ODMG[15]绑定的 DOM 接口将 XML 文档存入一个本地面向对象数据库系统 Fish[16].

测试的数据集以德国 CWI 所提出的 XML Benchmark Project[17]为基础,按照固定 DTD 信息生成不同大小的 XML 文档,生成的文档从 20M~100M.我们使用一个复杂的包含 20 个标准 XQuery 查询语句的查询集[17]进行性能测试.

3.2 性能分析

WIN vs PSPIB:这一部分,我们主要将 WIN 的性能与 PSPIB 的性能进行比较.为了保证公平性,我们对两种分片策略采用相同的查询方法,使用文献[11]提出的 PNameIndex 索引、NameIndex 索引、TextValueIndex 索引和 AttributeValueIndex 索引技术,通过查询重写和哈希连接技术完成所有查询操作.为了保证实验数据的准确性,我们执行每条查询语句 5 次,其中第 1 次为冷结果,后 4 次为热结果.冷结果是指数据库缓冲区和磁盘缓冲区中不存在任何数据库数据时的测试结果;热结果与之相反,我们取热结果平均响应时间为最终响应时间.我们对 20M,40M,60M,80M,100M 这 5 种类型文档,从 1~5 台机器对查询集进行测试,测试工作量和实验结果数据量都非常大.由于内存限制,当处理大文档如 80M,100M 文档时,一台处理机经常成为性能曲线中的异常点,因此在加速比性能分析时,从两台处理机的性能开始分析.由于受篇幅限制,无法给出全部查询性能曲线.

我们首先给出查询 Q_5 的性能曲线,这个查询是对字符串转型类型的测试.由于字符串是 XML 文档中类属数据类型,因此,关于解释字符串的查询经常涉及到将字符串转换成具有更多语义的其他数据类型.这个语句主要查询如下内容:How many sold items cost more than 40.

该查询挑战数据库管理系统转换提供的原始类型的能力.尤其当我们没有额外的 XML 模式信息或只有 DTD 文档的时候,这种数据类型转换操作发生的频率非常高.

Q_5 在两种分片策略下的加速比性能如图 5~图 9 所示.可以看出,该查询在两种分片策略中都有很好的加速比性能.这说明两种分片策略对字符串转型操作有很好的支持,而且由于该查询中只包含单条路径表达式,并行系统具有极好的查询内无关并行性,而且结果合并工作量很小.因此,两种策略均具有很好的加速比性能.

WIN 和 PSPIB 在查询 Q_5 下的缩放比性能曲线如图 10 所示.

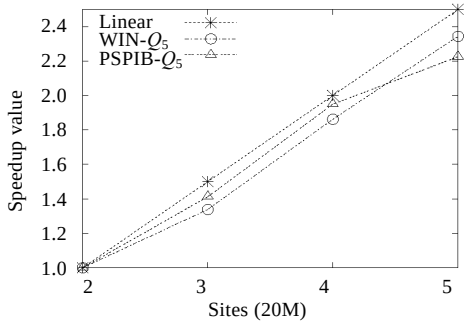

Fig.5 Q_5 speedup using the 20M document

图 5 20M 文档 Q_5 加速比

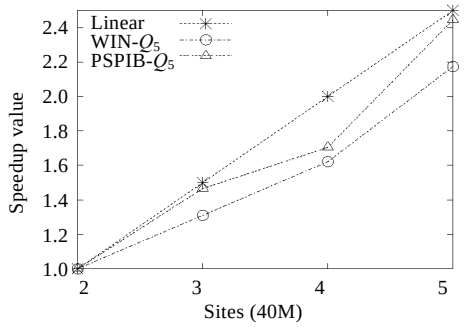

Fig.6 Q_5 speedup using the 40M document

图 6 40M 文档 Q_5 加速比

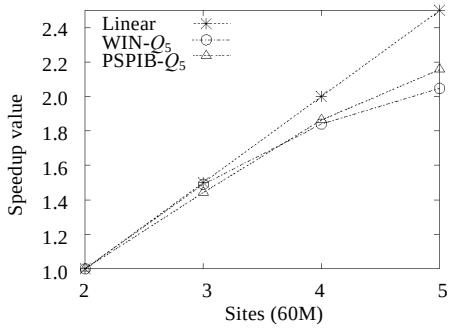

Fig.7 Q_5 speedup using the 60M document

图 7 60M 文档 Q_5 加速比

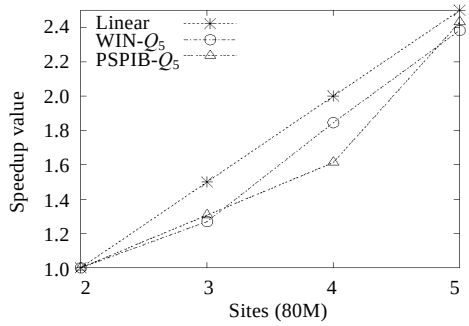

Fig.8 Q_5 speedup using the 80M document

图 8 80M 文档 Q_5 加速比

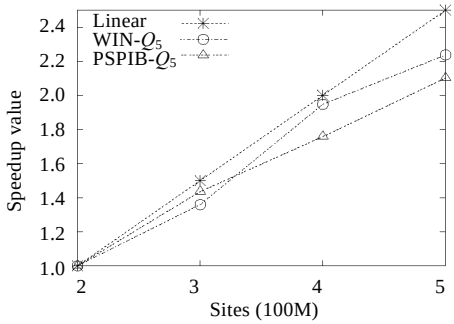

Fig.9 Q_5 speedup using the 100M document

图 9 100M 文档 Q_5 加速比

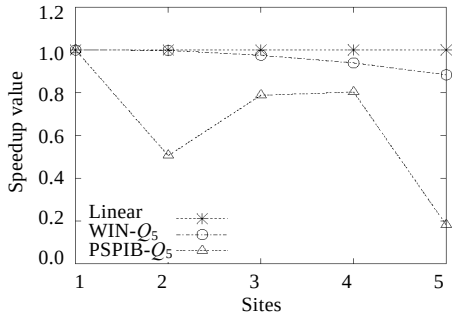

Fig.10 Q_5 scaleup

图 10 Q_5 的缩放比性能

图 10 中,1 台处理机处理 20M 文档;2 台处理 40M 文档;3 台处理 60M 文档;4 台处理 80M 文档;5 台处理 100M 文档.从图中我们可以清楚地看到:随着处理机个数的增加,WIN 的缩放比性能下降很少,并且接近线性;相反地,PSPIB 的缩放比性能不规则地变化并且呈现波浪型趋势.因此,我们可以很容易得出下面的结论:在这种模式下,WIN 比 PSPIB 具有更好的缩放比性能.

接下来,我们对标准测试集中的 Q_6 进行分析,该查询是对正则路径表达式类型性能的测试.在 XML 或者半结构化数据的每种查询语言中,正则路径表达式都是一种基本的查询类型.正则路径表达式考察查询处理器优化路径表达式和剪除不相关路径的能力.这个语句主要查询如下内容:

How many items are listed on all continents?

这个查询是对 XPath 中 descendant-or-self 轴的测试,一个好的查询引擎或者路径编码模式可以通过对数据树的优化,无须遍历所有路径而找出查询结果.最近,一些编码方式[12,18]已经为这种查询的优化提供了有效的解决方案.Q_6 在两种分片策略下的加速比性能如图 11~图 15 所示.由这些加速比性能曲线图可以看出:在各种大小文档下,WIN 的加速比性能要优于 PSPIB,这是由于我们在查询中采取了查询重写技术.在本查询中,将一个 descendant-or-self 轴的查询分解为多条子查询语句,然后对各条子查询语句的查询结果进行连接操作,得出最终查询结果.在 PSPIB 分片算法中,各条路径实例是根据自身的个数进行轮循分布的,不同路径模式的路径实例之间没有联系,这就导致各个路径实例分片结果具有随机性,具有相近关系的不同路径模式的路径实例很有可能不存储在同一台处理机.因此,很大程度上增加了查询时的通信开销和结果合并工作量.与之相反,WIN 的基本分片规则保证了路径实例的平均分配,并且 WIN 分片策略中采用的 *Partition*()保证了具有相近关系路径实例的本地化存储,因此节省了大量的网络通信代价和结果合并代价,因此该算法有很好的加速比性能.

WIN 和 PSPIB 在查询 Q_6 下的缩放比性能曲线如图 16 所示.从图中我们可以清楚地看到:随着处理机个数的增加,PSPIB 的缩放比性能下降得很快;但是与之相比,WIN 的下降趋势相对较小并且趋于稳定,因此,在这种类型的查询下,WIN 具有更好的缩放比性能.

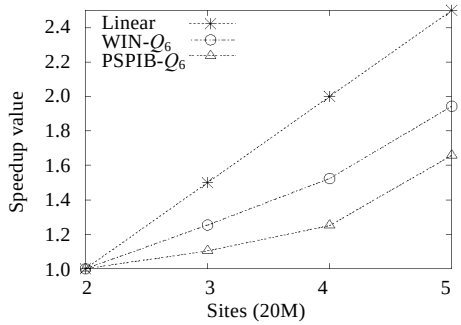

Fig.11 Q_6 speedup using the 20M document

图 11 20M 文档 Q_6 加速比

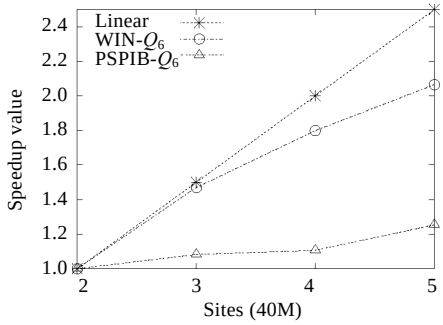

Fig.12 Q_6 speedup using the 40M document

图 12 40M 文档 Q_6 加速比

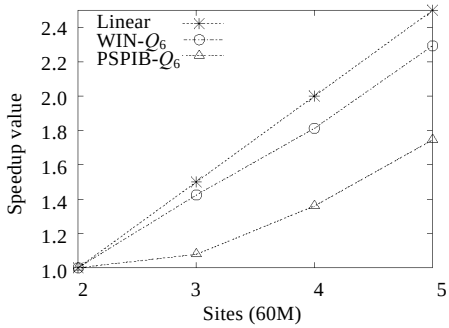

Fig.13 Q_6 speedup using the 60M document

图 13 60M 文档 Q_6 加速比

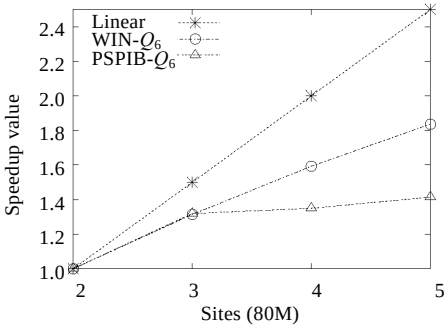

Fig.14 Q_6 speedup using the 80M document

图 14 80M 文档 Q_6 加速比

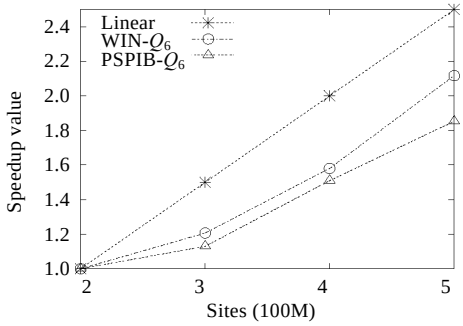

Fig.15 Q_6 speedup using the 100M document

图 15 100M 文档 Q_6 加速比

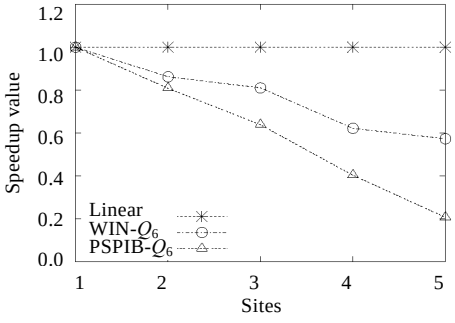

Fig.16 Q_6 scaleup

图 16 Q_6 的缩放比性能

查询测试集中还包含一些很复杂的查询表达式,我们最后对一个比较复杂的查询表达式 Q_{20} 进行性能分析.这个查询是一个聚合类查询,它将每个人根据不同数据进行分类,由于这个查询也包括对一些没有相关数据人的分类操作,因此该查询是一个典型的半结构化查询.这个查询具体内容如下:

Group customers by their income and output the cardinality of each group?

在两种分片策略下,Q_{20} 的加速比性能如图 17~图 21 所示.从这些加速比性能图中我们可以看到:在不同大小的文档中,PSPIB 性能曲线有下降的趋势,而 WIN 的性能明显优于 PSPIB.原因和前述查询相似:在 PSPIB 算法中,有相近关系路径实例是随意存储的,结果合并代价和通信开销很大;而 WIN 中,我们用 *Partition*()算法中启发式规则确保了相近路径实例尽量存储在同一处理机上,因此,WIN 的加速比性能明显好于 PSPIB.

图 22 是 Q_{20} 在两种分片策略下的缩放比性能曲线.从图中曲线走向我们可以很容易地看出:在这种很复杂的查询表达式下,WIN 比 PSPIB 具有更好的可扩展性.

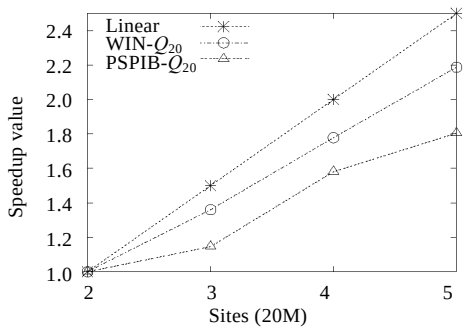

Fig.17 Q_{20} speedup using the 20M document

图 17 20M 文档 Q_{20} 加速比

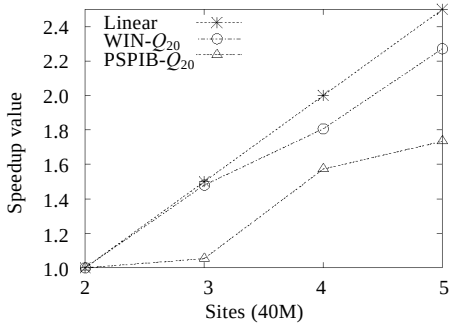

Fig.18 Q_{20} speedup using the 40M document

图 18 40M 文档 Q_{20} 加速比

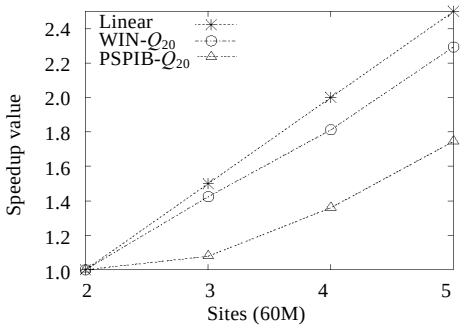

Fig.19 Q_{20} speedup using the 60M document

图 19 60M 文档 Q_{20} 加速比

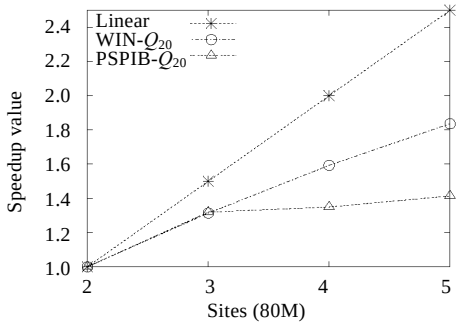

Fig.20 Q_{20} speedup using the 80M document

图 20 80M 文档 Q_{20} 加速比

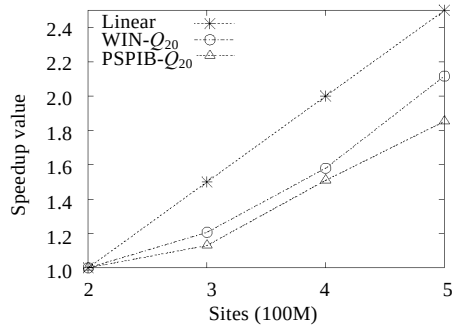

Fig.21 Q_{20} speedup using the 100M document

图 21 100M 文档 Q_{20} 加速比

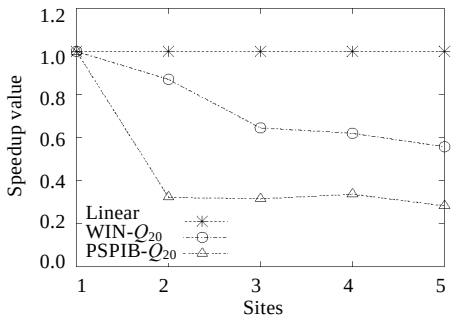

Fig.22 Q_{20} scaleup

图 22 Q_{20} 的缩放比性能

4 结束语

本文提出了一种新的基于工作负载均衡的并行 XML 数据库分片算法 WIN.该算法结合了 XML 数据树结构的特点和 XML 查询语言的特性,因此获得了很高的查询并行性能和较低的数据倾斜.基于一个标准的测试数据库[17]和测试集的完整测试,大量实验结果表明,WIN 的性能要全面优于以往的并行 XML 数据库中分片算法.本文提出的 XML 数据库分片算法能够很好地解决静态 XML 文档的数据分片问题,而 XMl 文档的动态分片策略是将来的主要研究工作.

References:

[1] Lomet DB, Salzberg B. The HB-Tree: A multiattribute indexing method with good guaranteed performance. ACM Trans. on Database Systems, 1990,15(4):625−658.

[2] Berchtold S, Keim DA, Kriegel H. The x-tree: An index struct for high-dimensional data. In: Vijayaraman TM, Buchmann AP, Mohan C, Sarda NL, eds. Proc. of the 22nd VLDB Conf. Bombay: Morgan Kaufmann Publishers, 1996. 28−30.

[3] Mehta M, DeWitt DJ. Data placement in shared-nothing parallel database systems. VLDB Journal, 1997,6(1):53−72.

[4] He Z, Yu JX. Declustering and object placement in parallel OODBMS. In: Roddick JF, ed. Proc. of the 10th Australasian Database Conf., ADC'99. Auckland, 1999. 18−21.

[5] Ghandeharizadeh S, Wilhite D, Lin K, Zhao X. Object placement in parallel object-oriented database systems. In: Agrawal R, Dittrich KR, eds. Proc. of the 10th Int'l Conf. on Data Engineering. Houston: IEEE Computer Society, 1994. 253−262.

[6] Berglund A, Boag S, Chamberlin D, Fernández MF, Kay M, Robie J, Siméon J. XML path languages (XPath), ver 2.0, W3C Working Draft, 2001. Technical Report, WD-xpath20-20011220, W3C, 2001. http://www.w3.org/TR/WD-xpath20-20011220

[7] Boag S, Chamberlin D, Fernández MF, Florescu D, Robie J, Siméon J. XQuery 1.0: An XML query language, W3C working draft, 2001. Technical Report, WD-xquery-20010607. World Wide Web Consortium.

[8] Yu Y, Wang G, Yu G, Wu G, Hu J, Tang N. Data placement and query processing based on RPE parallelisms. In: Voas J, ed. Proc. of the IEEE COMPSAC 2003 Conf. Dallas: IEEE Computer Society, 2003. 151−157.

[9] Chan CY, Garofalakis M, Rastogi R. RE-Tree: An efficient index structure for regular expressions. VLDB Journal, 2003,12(2): 102−119.

[10] Chung C, Min J, Shim K. APEX: An adaptive path index for XML data. In: Halevy AY, Ives ZG, Doan AH, eds. Proc. of the 2002 ACM SIGMOD Conf. Madison: ACM, 2002. 121−132.

[11] Lv J, Wang G, Yu JX, Yu G, Lu H, Sun B. Performamce evaluation of a DOM-Based XML database: Storage, indexing and query optimization. In: Meng XF, Su JW, Wang YJ, eds. Proc. of the WAIM 2002. LNCS 2419, Beijing: Springer-Verlag, 2002. 35−46.

[12] Al-Khalifa S, Jagadish HV, Koudas N, Patel JM, Srivastava D, Wu Y. Structural joins: A primitive for efficient XML query pattern matching. In: Proc. of the 2002 IEEE ICDE Conf. San Jose: IEEE Computer Society, 2002. 141−154.

[13] Bruno N, Koudas N, Srivastava D. Holistic twig joins: Optimal XML pattern matching. In: Halevy AY, Ives ZG, Doan AH, eds. Proc. of the 2002 ACM SIGMOD Conf. Madison: ACM, 2002. 310−321.

[14] Chen Z, Jagadish HV, Korn F, Koudas N, Muthukrishnan S, Ng R, Srivastava D. Counting twig matches in a tree. In: Georgakopoulos D, Buchmann A, eds. Proc. of the 2001 IEEE ICDE Conf. Heidelberg: IEEE Computer Society, 2001. 595−604.

[15] Cattell RG, Barry DK, Berler M, Russell C, Schadow O, Stanienda T, Velez F, Eastman J, Jordan D. The Object Database Standard: ODMG 3.0. San Francisco: Morgan Kaufmann Publishers, 2000.

[16] Yu G, Kaneko K, Bai G, Makinouchi A. Transaction management for a distributed store system WAKASHI design, implementation and performance. In: Su SYW, ed. Proc. of the 12th IEEE ICDE Conf. New Orleans: IEEE Computer Society, 1996. 460−468.

[17] Schmidt A, Waas F, Kersten M, Carey MJ, Manolescu I, Busse R. XMark: A benchmark for XML data management. In: Lochovsky FH, Bernstein PA, eds. Proc. of the 28th VLDB Conf. Hong Kong: Morgan Kaufmann Publishers, 2002. 974−985.

[18] Wang W, Jiang H, Lu H, Yu JX. PBiTree coding and efficient processing of containment join. In: Dayal U, Ramamritham K, Vijayaraman TM, eds. Proc. of the 2003 IEEE ICDE Conf. Bangalore: IEEE Computer Society, 2003. 391−404.

王国仁(1966－),男,湖北崇阳人,博士,教授,博士生导师,CCF 高级会员,主要研究领域为多媒体数据管理,P2P 数据管理,生物信息学,XML 数据管理.

汤南(1980－),男,硕士生,主要研究领域为并行 XML 数据管理.

于亚新(1972－),女,讲师,主要研究领域为并行 XML 数据管理.

孙冰(1978－),男,博士生,主要研究领域为 XML 数据管理.

于戈(1962－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为分布式数据库,数据仓库与数据挖掘,数据流管理.