BSC:一种高效的动态 XML 树编码方案*)

汪陈应　袁晓洁　王　鑫　刘众奇
(南开大学计算机科学与技术系　天津 300071)

摘　要　确定一篇 XML 文档中任意两个节点之间是否存在某种结构关系,是 XML 查询处理过程的一个重要组成部分。XML 树编码方案为每个节点分配唯一编号,仅通过比较节点编号而不必访问原 XML 文档,就可以快速有效地确定节点间的结构关系。随着 XML 应用不断普及,能否高效地支持更新操作,已成为 XML 树编码方案研究的一个重要课题。本文基于二进制小数的特性,提出了一种新的 XML 树编码方案——BSC,它可以完全高效地支持 XML 更新操作而不需要重新编码。实验结果证明,与已有的动态编码方案相比,BSC 编码无论在静态编码方面还是在动态更新方面都具有很好的性能。

关键词　XML,更新,动态,编码方案

BSC: An Efficient Numbering Scheme for Dynamic XML Trees

WANG Chen-Ying　YUAN Xiao-Jie　WANG Xin　LIU Zhong-Qi
(Department of Computer Science and Technology, Nankai University, Tianjin 300071)

Abstract　It is very important to detect whether there are structural relationships between two nodes in XML queries. The numbering scheme is designed to label the XML nodes so that the structural relationships between nodes can be easily determined by comparing their labels without accessing the original XML file. With the increasing popularity and application of XML related standards, it is urgent to find a numbering scheme that is able to support XML data updates. This paper presents a novel XML tree numbering scheme, called BSC, which takes advantage of the property of binary decimal numbers and can completely avoid re-labeling any existing nodes when the XML update is performed in any case. Our experimental results show that BSC works much better than the existing dynamic numbering schemes considering either the static numbering or the XML data updates.

Keywords　XML, Update, Dynamic, Numbering scheme

1　引言

随着 XML 相关标准的推广与应用,Web 上出现了大量的 XML 数据,需要存储到数据库中进行管理。XML 数据库管理系统的研究已经引起了国内外众多学者的关注。在查询语言方面,已经有 XPath[1] 和 XQuery[2] 等用来处理 XML 数据,而确定两个节点间的结构关系在 XML 查询处理中扮演着重要的角色。

为了优化 XML 查询处理,一般情况下,XML 树中的每个节点都被赋予唯一的编号,通过这个编号可以快速地确定任意两个节点之间是否存在父子、祖先后裔、兄弟、前驱、后继等关系。为 XML 树中的节点分配唯一编号的具体策略称为编码方案。为了有效地适应 XML 数据的查询和更新,一个高效的编码方案应该有以下三个方面的特征:

· 确定性。能够快速高效地确定 XQuery[2] 中定义的 13 种 XML 节点间结构关系的轴值(Axes)。

· 动态性。能够支持 XML 数据的动态更新,而不用重新改变任何已经存在的编号。因为在原生 XML 数据库中,索引建立在 XML 节点编号的基础上,如果改变了已有节点的编号,就破坏了整个索引结构,系统代价会大幅度增加。

· 压缩性。为了快速处理 XML 数据,一般一棵 XML 树或者 XML 数据片段的编号是需要保存在主存中的,因此节点编号所占的空间大小应该尽量小。

本文在前缀编码基础之上,利用二进制小数的特性,提出了一种新的编码方案。二进制小数的特性保证了动态更新,前缀特性保证快速高效地支持 XML 查询。

2　相关工作

在早期,XML 树编码被认为是一个颇具挑战性的研究课题,吸引了许多研究者。目前也出现了一批 XML 节点编码方案。所有编码方案可以分为两类:基于区间(region -based)的和基于前缀(prefix-based)的。

2.1　区间编码

早期的 XML 树编码方案研究都是基于区间的,文[3~5]中提出的都是基于区间的编码方案。基于区间的编码方案利用 XML 节点的有序特点,根据每一个元素节点的文档顺序位置为其赋予一个编码。这种编码通常是一个三元组[*start*, *end*, *level*]或者其变体形式[*order*, *size*, *level*](后者可以理解为 $start=order$;$end=order+size$),并且对于节点 u 和节点 v,u 是 v 的祖先节点当且仅当 $start(u)<start(v)<end(v)<end(u)$。$u$ 是 v 的父亲节点当且仅当 u 是 v 的祖先节点并且 $level(u)=level(v)-1$。

*)天津市科技发展计划基金项目(06YFGZGX05700)、天津市应用基础研究计划项目(07JCYBJC14500)。汪陈应　博士研究生,主要研究方向:XML 数据管理。

基于区间的编码方案不仅能够有效地支持包含关系的计算，而且能够有效地支持文档位置关系的计算，同时编码的长度一般较小。因此，在进行结构查询时，比较和计算所花费的时间代价也相对较小。但是，基于区间的编码方案一直无法完全实现编码的动态特性，当插入的新结点的数量超过预留空间后，重新编码所引起的时间和空间开销相当大。

2.2 前缀编码

基于前缀的编码方案利用 XML 文档的嵌套特点，节点的编号是以父节点的编号为前缀，加上该节点在本层的惟一标识，称为层标识。对于两个节点 u 和 v，u 是 v 的祖先节点当且仅当 u 的编号是 v 的编号的一个前缀；u 是 v 的父亲节点当且仅当 u 是 v 的祖先节点，并且 u 的编号比 v 的编号仅少一个层标识。文[6～10]中都是对基于前缀的编码方案的探讨。

DeweyIDs[7]和 ORDPATHs[8]是目前较好的两种动态 XML 树编码方案，但是它们有三个共同的缺点：

·最左端插入问题。为了解决这个问题，ORDPATHs 中引入了负数，DeweyIDs 把 1 作为属性节点的标志，并且认为所有的孩子节点都在属性节点之后。

·层标识溢出问题。作为层标识的整数，无论用固定长度的位数还是使用一个长度域和数据域来存储，当在同一个父节点下插入的孩子节点过多时，所有已经存在的兄弟节点及其后裔节点都需要重新编码。

·更新空间效率不高。在更新时，新插入的编号多了一个甚至多个作为标志的偶数。不仅这些偶数没有发挥实际作用，而且增大了编号的长度。另外，这些偶数也会造成同层节点编号的段数不一致，影响查询效率。

DLN[10]的层标识使用了可变长的编码，可以解决层标识溢出问题；在更新时，引入了子层编号和相应的子层分隔符。由于需要额外编码层分隔符和子层分隔符，一方面降低了存储效率，另一方面也增加了 DLN 编号的复杂度。同样 DLN 也存在最左端插入问题。

2.3 研究动机

到目前为止，除了 DLN，还没有一种编码方案在更新时可以完全避免重新编码。而 DLN 在更新时需要使用子层编码，造成空间效率下降。本文提出一种新的 XML 树编码方案——BSC(Binary Symbol Coding)，该编码方案在完全避免重新编码的前提下，使编号长度尽量减小，从而大大提高空间效率。

3 BSC 编码

DeweyID[9]是一种最基本的基于前缀的编码方案。它之所以没有动态性，是因为它的层标识是连续的正整数，而 ORDPATHs、DeweyIDs 和 DLN 都是对 DeweyID 的层标识进行了不同程度的改进，不过它们都没有摆脱层标识依然是整数的限制。本文用真分数来代替整数作为层标识，由于任意两个真分数之间都有无穷多个真分数，这个性质可以保证 BSC 编码可以在任意位置插入任意多个节点。而在物理存储方面，可以把真分数转化成对应的二进制小数。由于真分数的整数部分都为 0，因此只需要存储小数部分对应的二进制位串。在下面，分数形式被称为逻辑编码，而二进制位串形式称为物理编码。

3.1 基本定义

真分数转化成的小数有两类：有限小数和无限循环小数。无限循环小数是不能转化成有限长度的二进制位串，因此 BSC 编码的层标识只能利用能转化为有限小数的那部分真分数。为了提高效率，所有的层标识操作都是在物理编码上进行的。

定义 1(基本符号) “0”和“1”。逻辑编码中，它们是真分数转化的二进制小数中的一位；物理编码中，它们是存储的一个表示单位。

定义 2(BSC 层标识) 同一个父节点的所有孩子节点的唯一标识。逻辑编码中，它们就是一个真分数的集合；物理编码中，它们是由基本符号组成的一个二进制位串，用正则表达式表示是：(1|0)*1。

为了比较两个层标识的次序关系，并不需要将物理编码转化为逻辑编码，而是利用这些物理编码满足的字典序进行比较。

定义 3(字典序) 给定两个层标识 C_1 和 C_2 的物理编码，定义它们在字典序中的关系如下：

ⅰ. $C_1 \equiv C_2 \Leftrightarrow C_1$ 和 C_2 完全相同。

ⅱ. $C_1 \prec C_2 \Leftrightarrow$

a)“0”≺“1”。

b)从左向右比较 C_1 和 C_2 的当前基本符号，直到当前基本符号不相同停止比较，如果当前的基本符号 S_1 和 S_2 满足条件 a)即 $S_1 \prec S_2$，则 $C_1 \prec C_2$。

c)C_1 是 C_2 的前缀。

ⅲ. $C_1 \succ C_2 \Leftrightarrow C_2 \prec C_1$。

定义 4(层分隔符) 分隔层标识。可以使用逗号、小点号等，本文中一律采用小点号。

定义 5(BSC 编号) 由多个用层分隔符分隔的层标识组成的一个串。物理编码用正则表达式表示是((1|0)*1.)*(1|0)*1。

3.2 静态 BSC 编码

任何 XML 文档都可以表示为节点带有标签的树型结构，称之为 XML 树。而对 XML 文档进行编码，指的就是按照一定的规则，为 XML 树中的每个节点赋予惟一的标识。

定义 6(静态 BSC 编码) 即为 XML 树中的每个节点赋予一个惟一的 BSC 编码，用映射表示是：{XML 节点|XML 文档对应的 XML 树}→{BSC 编号}。

由定义 6 可知，静态 BSC 编码在 XML 文档载入阶段，为文档对应的 XML 树中的每个 XML 节点分配一个 BSC 编号。XML 树根节点的 BSC 编号通常为 1；而其它节点的 BSC 编号是用层分隔符将其父节点的 BSC 编号和该节点的层标识连接起来。因此为同一父节点的所有孩子节点分配层标识就是静态 BSC 编码的关键问题。

算法 1 静态层标识分配算法

```
输入：一个正整数 n
输出：n 个 BSC 层标识
①CodeArray[1,n]/*存放 N 个层标识的数组*/
②SubEncoding(CodeArray, 0, n+1,0,1)
③return CodeArray
④function SubEncoding(CodeArray, start, end, startValue, end-
  Value)
  /*SubEncoding 是个递归程序，start 和 end 分别是开始和结束位
  置，startValue 和 endValue 分别是开始和结束位置的层标识*/
⑤if start+1≥end return
⑥current:=(start+end)/2
⑦currentValue:=(startValue+endValue)/2;
⑧CodeArray[current]:=currentValue;
⑨SubEncoding(CodeArray, start,
  current, startValue, currentValue);
⑩SubEncoding(CodeArray, current,
  end, currentValue, endValue);
```

算法1的时间复杂度为O(log N),空间复杂度为O(N)。

3.3 动态BSC编码

所谓的XML动态性,是指对XML文档的插入、删除和更新操作的支持。由于XML文档的删除和更新操作不涉及XML节点编码的改变,因此对于XML树编码方案来说,动态性就是能支持XML节点的插入。

完全支持XML节点的插入指的是在任何父节点的任何两个相邻孩子节点之间插入一个新的孩子节点时,可以得到一个合法的编号,而不用修改存在的任何节点编号。

算法2 动态插入节点层标识的分配

输入:*left*, *right*
输出:*inserted*
①if 最左端插入:
②*inserted* := *right*/2;
③elseif 中间插入:
④*inserted* :=(*left*+*right*)/2;
⑤elseif 最右端插入:
⑥*inserted* :=(*left*+1)/2;

算法2的时间和空间复杂度均为O(1)。

4 实验

在这一节中,将对本文提出的BSC编码和DLN进行比较。所有的实验都是在实验室用C#语言编写的Native XML数据库系统——XNative上进行的。

表1 测试数据集

文档名称	深度 Max/Avg	扇出度 Max/Avg	节点数
Hamlet. xml	7/5.31	174/1.82	12099
Reed. xml	5/3.70	703/1.79	18855
Sigmod. xml	7/5.74	89/2.07	23814
NASA. xml	8/5.1	2435/1.77	845804
Treebank. xml	37/8.44	56385/1.57	3829512
SwissProt. xml	6/3.76	50000/2.41	7180734
XMark_1. xml	12/5.18	2550/1.93	324273
XMark_4. xml	12/5.19	10200/1.93	1290249
XMark_8. xml	12/5.18	20400/1.93	2585984

表1给出的是实验用到的10个测试数据集的特性。其中第一个数据集是文[11]提供的,第二个至第六个是文[12]提供的,后面三个是由XML自动生成工具XMark[13]生成的。其中的1,4,8表示文件的大小分别为10M,40M,80M。

在表2中,给出的是BSC和DLN静态编码的比较结果,其中的Avg和Max表示的是编号的平均和最大长度,单位是bit。为了方便比较,表2中DLN给出的是实际的实验测试数据,而BSC给出的是和DLN平均和最大编号长度的区别。

表2 静态编码性能

文档名称	DLN		BSC	
	Avg	Max	Avg	Max
Hamlet. xml	34.58	46	−4.49	−1
Reed. xml	27.79	40	−9.3	−5
Sigmod. xml	35.63	46	−6.82	−6
NASA. xml	45.30	68	−15.14	−23
Treebank. xml	58.25	200	−16.07	−15
SwissProt. xml	40.56	65	−21.75	−35
XMark_1. xml	41.49	84	−11.18	−19
XMark_4. xml	43.80	88	−13.45	−23
XMark_8. xml	45.46	88	−15.14	−23

为了测试BSC和DLN的更新效率,选取了表1中的第一个数据集Hamlet. xml作为原始的XML文档。Hamlet. xml在根节点下一共有五个Act元素节点。本文选取了表1中的第七个数据集XMark_1. xml作为其中的一个测试Act元素节点,表3给出了将它分别插在Hamlet. xml的六个位置中的DLN编码和BSC编码的比较结果。

表3 动态编码性能

插入位置	DLN		BSC	
	Avg	Max	Avg	Max
第一个Act节点前	55.49	98	−21.28	−29
一、二Act节点之间	51.49	94	−16.18	−24
二、三Act节点之间	51.49	94	−16.18	−24
三、四Act节点之间	51.49	94	−17.18	−25
四、五Act节点之间	51.49	94	−16.18	−24
第六个Act节点之后	46.49	89	−11.18	−19

由表2和表3可知,在静态编码和动态更新时,BSC都具有比DLN更好的性能,其平均编码长度比DLN要小15个bit左右,最大编号长度也比DLN要小20个bit左右,而且随着XML文档的节点增多,优势越大。

结束语 本文提出了一种新的高效的动态XML树编码方案。它不仅在静态编码中具有很好的空间效率,而且本文更注重的是:在更新时,它可以完全避免重新编码任何已有节点,并且只需要将插入节点相邻的两个节点的层标识之和的一半作为插入节点的层标识,更新效率非常高。今后的工作包括在BSC编码基础上设计和实现支持动态更新的原生XML数据库的索引结构。

参考文献

1 Clark J, DeRose S. XML Path Language (XPath) Version 1.0, W3C Recommendation, 1999
2 Chamberlin D, et al. XQuery 1.0: An XML Query Language, W3C working Draft, 2007
3 Li Q, Moon B. Indexing and Queryring XML data for Regular Path Expressions, In: VLDB, 2001
4 Zhang C, et al. On Supporting Containment Queries in Relational Database Management Systems. In: Proc. of SIGMOD, 2001. 425~436
5 Amagasa T, Yoshikawa M, Uemura S. QRS: A Robust Numbering Scheme for XML Documents. In: Proc. of ICDE, 2003. 705~707
6 Cohen E, Kaplan H, Milo T. Labeling Dynamic XML Trees. In: Proc. of PODS, 2002. 271~281
7 Duong M, Zhang Y. A New Labeling Scheme for Dynamically Updating XML Data. In: Proc. of ADC, 2005. 185~193
8 O'Neil P E, O'Neil E J, Pal S, et al. ORDPATHs: Insert-Friendly XML Node Labels. In: Proc. of SIGMOD, 2004. 903~908
9 Tatarinov I, Viglas S, Beyer K S, et al. Storing and querying ordered XML using a relational database system. In: Proc. of SIGMOD, 2002. 204~215
10 Böhme T, Rahm E. Supporting Efficient Streaming and Insertion of XML Data in RDBMS. In: Proc. 3rd Int Workshop Data Integration over the Web (DIWeb), LNCS, 2004
11 NIAGARA Experimental Data. Available at: http://www.cs.wisc.edu/niagara/data.html
12 University of Washington XML Repository. Available at: http://www.cs.washington.edu/research/xmldatasets/
13 XMark — An XML Benchmark Project. Available at: http://monetdb.cwi.nl/xml/downloads.html