ArithBi$^+$——一种基于反向算术压缩的 XML 索引结构*)

金彦钟[1] 包小源[2] 宋再生[2]

(天津科技大学计算机科学系 天津 300222)[1] (北京大学计算机科学系 北京 100871)[2]

摘 要 XML 在数据交换中的应用越来越广泛,但由于加入标记后的空间膨胀较大,对传输及存储资源耗费严重。压缩后的 XML 数据容量明显减少[1~3],但怎样基于压缩后的 XML 数据直接进行查询及处理,仍然是需要深入研究的问题。本文以反向算术压缩为基本压缩算法,提出针对 XML 数据库中压缩 XML 文件的索引结构 ArithBi$^+$。基于该索引结构,可高效实现对类似//element$_1$/element$_2$/…/elment$_m$ 的查询处理。

关键词 XML,索引,平衡二叉树,算术压缩

ArithBi$^+$—An XML Index Structure on Reverse Arithmetic Compressed XML Data

JIN Yan-Zhong BAO Xiao-Yuan SONG Zai-Sheng

(Deparatment of Computer Science, Tianjin University of Science Tehnolohy, Tianjin 300222)[1]

(Department of Computer Science, Peking University, Beijing 100871)[2]

Abstract Even XML is used as a popular data exchange standard over Internet and Intranet, its space expansion because of adding tags to every different semantic content unit makes the transmitting and storing of XML data very expensive in terms of resources. After compressed[1~3], its size will be much smaller, but how to evaluate query directly based on the compressed data is still necessary for us to do some work. We propose an XmL index structure, ArithBi$^+$, on compressed data which is result from revert arithmetic compression. Queries as the form of //element$_1$/element$_2$/…/elment$_m$ can be evaluated efficiently using ArithBi$^+$.

Keywords XML, Index, Balanced binary tree, Arithmetic compression

1 引言

XML 在数据交换中的应用越来越广泛,但由于其空间扩展较大,对传输及存储资源耗费严重。就 XML 数据库中存储 XML 而言,压缩后进行存储是必要的。但若压缩后的数据不能支持基于其上的查询等操作,则数据库的性能会受到影响。利用保持结构的 XML 压缩方法(如 XPress 压缩方法)就可以解决这个问题。为了更清楚地描述问题,假设压缩只对 TAG 进行,元素内容压缩这里不作讨论,其实一些现成的压缩方法如 LZ、字典压缩方法等都可以直接用于 XML 元素值的压缩,这里不再赘述。

我们采用了一种保持结构的压缩算法:针对简单路径的反向算术压缩,该压缩只依据路径对元素进行而不涉及元素的内容,从而保持了 XML 文件原来的结构。这样,当基于压缩后的 XML 文件回答诸如//elment$_1$/element$_2$/…/element$_n$ 或/elment$_1$/element$_2$/…/element$_n$ 时,只需要将查询路径重写为一个对应的右开压缩值区间$[L,R)$(或一个值 m),并与压缩文件中的元素(已经变为压缩值)进行比对,若某元素落入该区间,则该元素满足查询条件,进入结果集。显然,要得到所有的查询结果,需要对整个文件进行扫描,这种代价是很大的,在有些应用环境中是不允许的(如搜索引擎中)。

鉴于此,对压缩后的 XML 建立索引是有效降低这种代价的有效途径。由于压缩后的 XML 文件中,每个元素已变为对应的压缩值(相同路径的元素具有相同的压缩值),我们对该压缩值集合建立扩展的平衡二叉树(命名为 ArithBi$^+$)。这样,执行查询//elment$_1$/element$_2$/…/element$_n$ 或/elment$_1$/element$_2$/…/element$_n$ 的过程就变为判断 ArithBi$^+$ 中的哪些结点落在区间$[L,R)$中(对//elment$_1$/element$_2$/…/element$_n$)或等于查询值 m(对/elment$_1$/element$_2$/…/element$_n$)。

本文只讨论这两种基本查询路径,对于其它情形,本文讨论的方法作一定的扩展后仍然适用,文中不再详细叙述。

本文第 2 节给出反向算术压缩的基本算法及相关内容;第 3 节详细叙述 ArithBi$^+$ 的构造、维护等相关内容;第 4 节给出基于 ArithBi$^+$ 的查询算法,并有部分实验结果;第 5 节是相关工作;最后是结论及展望。

2 XML 反向算术压缩

2.1 XML 数据树

定义 1 XML 数据树(图)。采用半结构化数据模型 OEM[4](Object Exchange Model)描述 XML 的数据内容。在 OEM 中,用图来表示数据(称 XML 数据图),图中的结点和边均有标记。图中,结点表示对象,边标记为属性名称。OEM 中的对象有原子对象(Atomic object)与复合对象(Complex object)之分,原子对象的值为不可分割的量,如一个整数值;复合对象的值是由(label, id)偶对为元素构成的集合。XML 数据可用 OEM 模型表示:将 XML 元素用 OEM 结点表示;元素-子元素关系、元素-属性关系,以及参考关系等用相应的标记为对应名字的边来表示。XML 中的数据值(这里简单假设其所有数据均为字符串)与 OEM 的叶节相对应。文中,为了简单起见,没有考虑结点引用的情形,故又可

*)本文研究得到 973 国家重点基础研究发展规划(G1999032705)、863 数据库重大专项课题(2002AA4Z3440)项目支持。

称 XML 数据树。

图 1a 是文中的 XML 用例;图 1 b 是其对应 OEM 图。其中,&0, &1 等是对象标识号。对象 &13, &14 等是原子对象,而 &1, &2 等是复合对象。

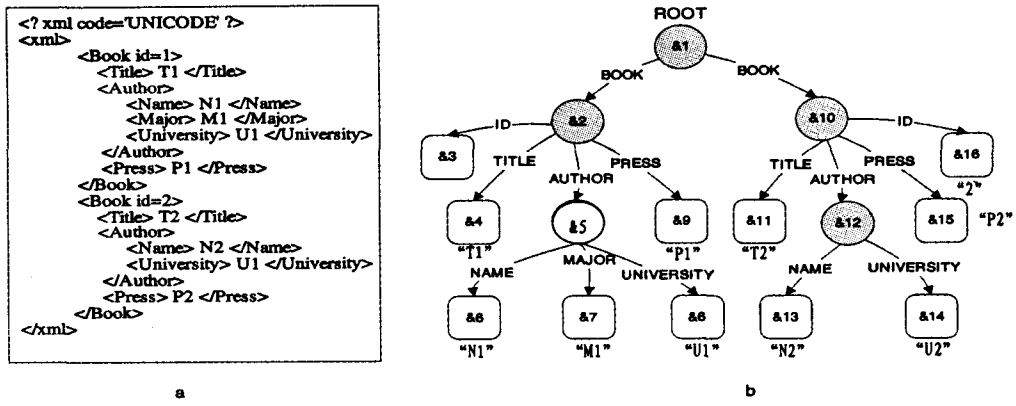

图 1 示例 XML 文档(a)及其 OEM 数据图(b)

为了简明的叙述,下面讨论主要针对 XML 的元素而不涉及其内容(如"N1"等)以及元素的属性(如结点 &3, &16 等)。

定义 2 元素(结点)路径。结点路径是指从根结点开始到该结点为止,将沿途的有向边标记(label)以"/"为连接符进行顺次联结所形成的串。根结点直接用一个"/"表示。例如结点 &4 的路径为/book/title。下文中,在不导致误解前提下,会将"结点"和"元素"作为同义词混用。

2.2 XML 反向算术压缩

·XML 反向算术压缩 算术压缩[5]的基本思想是首先利用统计方法取得压缩对象的分布信息,之后按照该信息将[0,1]区间进行分割,注意这种分割是边界连续,不互相覆盖的。这样,每个压缩原子对象都有一个区间与之对应,称该区间为该原子对象的概率区间。

以字符串 abcbbc 为例,经统计可知,a 的概率为 1/6,b 的概率为 3/6,c 的概率为 2/6。以 a、b、c 为顺序,用它们的概率值对区间[0,1]进行划分,则得出如下的基本概率区间分布。

a[0, 1/6)	简写为[a)
b[1/6, 4/6)	简写为[b)
c[4/6, 6/6)	简写为[c)

这样,对 *abcbbc* 进行压缩时,首先以第一个字符 a 的区间作为基本区间 P,以基本概率区间分布再对 P 进行划分得 Q,当下一个字符 b 到来时,串 ab 的压缩结果即为 Q 中 b 所占的区间(简写为"[a[b))",表示区间[a)经划分后 b 所对应的区间)。如果本次压缩过程仍要继续,则用基本概率区间分布再对 Q 进行划分,所得的概率区间即为前三个字符所组成字符串的压缩结果,循环往复,直到一次压缩结束。这里说的一次压缩是指由于压缩整体对象一般很大,因此将看成连续的小片段,每次压缩都是针对这样一个小片段进行。

实际实现中,一般以该区间中一个值(譬如左闭边界值)来表示压缩结果即可。可以看到,对于串 *abcbbc* 而言,如果将它作为一个压缩小片段,则有如下关系成立([a[b[c[b[b[c)))))))是串 abcbbc 的压缩结果):

引理 1 $[a)\supset[a[b))\supset[a[b[c)))\supset\cdots\supset[a[b[c[b[b[c))))))$

考察引理 1,发现上述概率区间的包含关系与 XPath 查询的包含关系有相似之处。仍以查询//paper/author/name 及//author/name 为例,观察得到:若以元素名(TAG)为压缩的原子对象,则有 $[name[author))\supset[name[author[paper)))$。那么,在对 XML 中的一个元素进行压缩时,若以该元素的路径为依据,反向进行压缩算术压缩直到根结点为止,则所得压缩值作为该元素的压缩结果,并且该压缩结果本身是包含路径信息的。我们称这种压缩方法为 XML 反向算术压缩。下文中,所有谈到压缩的地方,均指利用 XML 反向算术压缩。

设一个元素 e_x 的路径为 $/e_1/e_2/\cdots/e_x$,其压缩值为 $[e_x[\cdots[e_2[e_1))))$,有一个查询具有形式 $//e_i/e_{i+1}/\cdots/e_x$ ($i>=1$),它的对应压缩值(指对路径 $//e_i/e_{i+1}/\cdots/e_x$ 进行反向压缩)为 $[e_x[\cdots[e_{i+1}[e_i))))$。根据引理 1 及定义 1,则有:

$[e_x[\cdots[e_{i+1}[e_i))))\supseteq[e_x[\cdots[e_2[e_1))))$

为了进一步叙述,首先引入定义 3。

定义 3 查询语句的包含关系。设有 XPath 查询 P, Q,令

$P=//element_1^P/element_2^P/\cdots/element_n^P$

$Q=//element_1^Q/element_2^Q/\cdots/element_m^Q$

不失一般性,设 $n\geqslant m$,若有 $element_m^Q=element_n^P$, $element_{m-1}^Q=element_{n-1}^P,\cdots,element_1^Q=element_{n-m+1}^P$,则称 Q 包含 P,记作 $Q\supseteq P$。实际上,查询的包含关系也表示了其查询结果的包含关系。仍以查询 P=//paper/author/name 及 Q=//author/name 为例,显然 Q 的查询结果包含全部 P 的查询结果。注意到,若 P 具有形式/paper/author/name,则 $Q\supseteq P$ 仍然成立。

这样,根据定义 3 及前叙述内容,有定理 1 成立。

定理 1 设有一 XML 文件,若 $\exists\rho\in\Sigma$,且查询路径 $P=//e_i/e_{i+1}/\cdots/e_m$ 的压缩值为 q,有 $\rho\supseteq q$,则 ρ 是路径 $Q=/x_1/x_2/\cdots/e_i/e_{i+1}/\cdots/e_m$ 的压缩值。其中,Σ 是 XML 文件所有压缩元素值的集合。

证明略。

定理 1 说明了这样一个事实:对于一个查询 Q 及其压缩值 q,压缩文件中所有满足 $\rho\subseteq q$ 的元素就是该查询的结果集。其中,ρ 是元素的压缩值。

最后给出算术压缩的一般算法。

算法 1 Arithmetic compression algorithm

```
Low=0.0
High=1.0
While((c=getc(input)! =EOF ) {
```

```
    Range=high-low;
    High=low+range * high_range( c );
    Low=low+range * low_range( c );
    }
output(low );
```

解压缩是压缩的逆过程，算法也类似。

下面图 2 给出了图 1 示例 XML 文件的对应压缩 XML 文件及 XML 数据图。需要注意，在下文的叙述中，只利用了图 2 所示的压缩值的左边界。

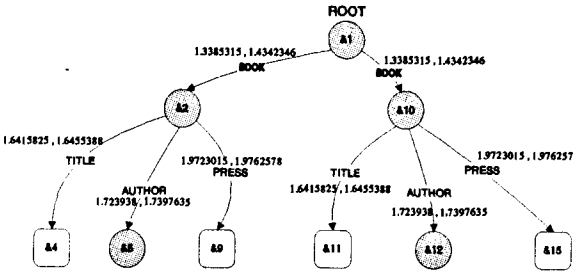

图 2　示例 XML 文档对应压缩结果及数据图

• 压缩统计信息提取方法　在 XML 反向算术压缩中，需要统计各元素出现的次数，以确定各元素所占的概率区域。当直接基于压缩 XML 文件执行查询时，首先要将查询语句重写为一个概率区域，而该重写过程需要源 XML 文件的统计信息。统计信息可以扫描整个 XML 文件后得出，但若直接采用对元素进行计数的方法，由于离 XML 数据树根近的元素其出现的次数越少，这样会造成在压缩时概率空间衰减幅度大，需要高精度的浮点运算才能胜任。这里采用一种改进的计数方法[6]：在 XML 数据树中，计算一个元素所能到达的不同名称元素结点的个数作为该元素的计数值，若某元素已经出现过，则不再进入统计计数中。下面给出基于流化的 XML 统计信息提取算法。

算法 2　压缩统计信息获取

```
Input: Streaming XML Data
Output: Statistics Information
1. EleHashTab={0, { }}//EleHashTab is Hash Table to store Tag
   Counting Info.
2. For each Element Tag of the XML file do {1}
3.   If HashFunc(Tag)∉EleHashTab then
4.     EleHashTab.EleInfo←EleHashTab.EleInfo∪{{Tag,0}}
5.     For each ti from root to Tag along path {2}
6.           EleHashTab.EleInfo[ti].count + 1 //Each item has
             structure of { TagName, Count}
7.           EleHashTab.Summary+1//EleHashTab has a field to
             store Summary cou nts
8.     End For {2}
9. End If
10.  End For {1}
```

算法 2 中，EleHashTab 是存放所有统计信息的 Hash 表，该表有两项：总体统计信息 *Summary* 及各元素统计信息 *EleInfo*。当扫描(循环{1})到一个元素(tag)时，若该元素以前没有出现过(行 3)，则首先在 EleInfo 中加入该元素的统计初始值{Tag,0}；接下来，将自 root 到该元素的路径所经过的所有结点的计数值加 1(循环{2})，同时总体统计数加 1。

分析该统计信息提取方法，可以发现，对于某些有 DTD 的 XML 文件，若 DTD 满足一定约束条件，则统计信息可以直接通过 DTD 取得。我们在实现中，对 DTD 作出如下简单规定：不考虑“?”、“*”及“|”的情形。可以证明，满足该规定的 DTD 可以作为上述统计方法的替代统计源，结果与对源文件的统计结果一致。显然，该方法要比直接扫描源文件节省时间。

接下来，简单介绍如何基于 DTD 提取压缩对应 XML 文件所需统计信息。需要分两个步骤进行：①将 DTD 转换为 DTD 图；②对 DTD 图进行遍历得出统计信息。DTD 定义了 XML 数据中元素及属性的父子关系，它可以被表示为一个 DTD 树(图)。图 3 所示为一个 XML 文件的 DTD 及其对应 DTD 图。二元组 $G=(N,C)$ 表示有向 DTD 图，其中每一个 $E \in N$ 对于于 DTD 的一个元素，且有 $c \in C, c=(E1,E2)$ $E1,E2 \in N$，表示若在 DTD 中元素 $E1$ 的子元素包括 $E2$，则有一条由 $E1$ 指向 $E2$ 的边。

```
<!ELEMENT Root (E1, E2)>
<!ELEMENT E1 ( E3)>
<!ELEMENT E2 (E4,E5)>
<!ELEMENT E3 (#PCDATA)>
<!ELEMENT E4 (#PCDATA)>
<!ELEMENT E5 (#PCDATA)>
```

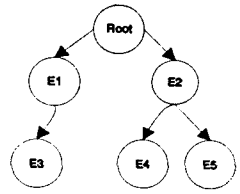

图 3　DTD 及对应 DTD 有向图

基于 DTD 图提取统计信息的具体算法与算法 2 类似。

3　压缩 XML 索引结构 ArithBi+

3.1　基本思路

反向算术压缩生成的压缩结果是一个区域值，实际一般取区域的下界为一个单一的数值即可。若要直接对压缩 XML 文件进行查询时，以查询//author/name 为例，其查询执行的过程由以下步骤组成：

①按照压缩统计信息对 author/name 实施反向压缩(即查询重写)，得到一个区间 Δ；

②对压缩数据进行扫描，把其数值落入区域 Δ 的元素及其内容并入结果集合；

③若文件扫描到末尾则结束，输出结果集；否则转②。

其实，在压缩 XML 文件中，路径相同元素的压缩结果相等。若以压缩值相等为等价条件对压缩 XML 文件中的元素(包括其内容)进行等价划分，则每个等价类中的元素压缩值相同，亦即这些元素有相同的路径。将每个等价类看作一个对象，构造一棵平衡二叉树，树中的一个结点对应一个对象，则该二叉树就是针对压缩 XML 文件的索引，命名为 ArithBi+。

这样，基于 ArithBi+ 对压缩 XML 文件执行查询//author/name 的过程，实际就是寻找 Δ 包含树中哪些叶结点的值的过程。如图 1 中结点 &6，&13 的路径为/book/author/name，它们属于同一个等价类(压缩后值相等，设为 x)。显然 $x \in \Delta$ 成立，所以该类中的所有元素 &6、&13 均为查询结果。

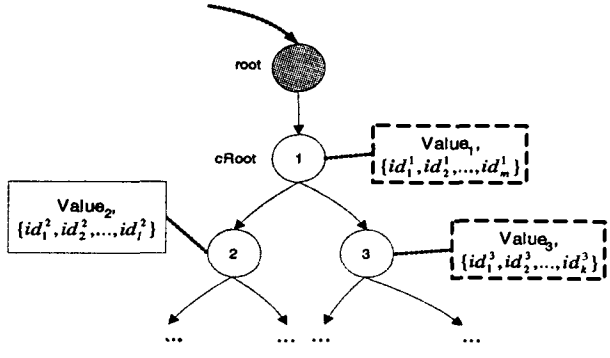

图 4　ArithBi+ 结构示意

3.2　ArithBi+

图4给出的是ArithBi$^+$结构的一个示例。树中每个结点包含了一个关键值 *value*，以及与该值对应的所有元素的集合，结点1,2,3所包含的值 $value_1$，$value_2$ 及 $value_3$ 满足关系 $value_2<value_1<value_3$，它们对应的扩展集分别为 $\{id_1^1, id_2^1, \cdots, id_m^1\}$，$\{id_1^2, id_2^2, \cdots, id_l^2\}$，及 $\{id_1^3, id_2^3, \cdots, id_k^3\}$。实际上，在具体实现中，每个 id 是对实际元素的位置索引(譬如可以是元素的开始和终止位置)。

下面给出索引结构 ArithBi$^+$ 的形式化定义。

定义4 ArithBi$^+$。设某压缩 XML 文件的压缩元素值集合为 Σ，对应原始 XML 文件元素集合 $U=\{id_1, id_2, \cdots, id_n\}$。由前述反向算术压缩的性质，可知对于每一个压缩值有多个元素(用唯一 *id* 标识来区分)与之对应，即存在影射 $f: \forall x_i \in \Sigma, \exists y_i=\{id_j \mid id_j \in U, j=1,2,\cdots,n$，且元素 id_j 的压缩值$=x\}$，并有 $\bigcap y_i=\phi(i=1,2,\cdots,P$，P 为集合 Σ 的势)，记为 $f(x_i)=y_i$。

平衡二叉树 $T=\{V,E,f,\Sigma,U\}$，V,E 为树的结点及边。T 中每个结点 $v\in V$ 包含一个且仅一个 $x\in\Sigma$ 以及 $y=f(x)$，分别作为结点值(用于搜索，记为 *value*)和扩展集(记为 *extent*，表示压缩值为 *value* 的元素均在集合 *extent* 中)，即每个结点包含二元组 $\{value, extent\}=\{x,\{id_1, id_2, \cdots, id_m\}\}$。

根据上述索引结构 ArithBi$^+$ 的定义，其构造可经以下步骤完成。

①初始化 T=ArithBi$^+$，开始只有一个根结点 root(虚拟根结点)；

②读入压缩文件的第一个元素(值为 x，元素编码为 id_r)时，创建 root 的子结点(cRoot)，并使 cRoot. value=x，cRoot. extent=$\{id_r\}$；

③读入下一个元素(y, id_t)，在 T 中查找 value 值为 y 的结点，若查找成功(设为结点 M)，则 M. extent←M. extent$\cup\{id_t\}$，转⑤；若不成功则转④；

④在自 cRoot 向下的查找过程中，若有某结点 N(value=w)，满足 $y>w$ 且 N 无右子结点，则创建结点 $L=\{y,\{id_t\}\}$，且令 L 为 N 的右子结点；若满足 $y<w$ 且 N 无左子结点，则创建结点 $L=\{y,\{id_t\}\}$，且令 L 为 N 的左子结点；

⑤到压缩文件尾否？是则结束返回；否则转②。

3.3 ArithBi$^+$ 的维护

ArithBi$^+$ 是一种基于平衡二叉树的结构，所以它的维护基本与平衡二叉树的维护类似，这里只对由于引入扩展集而导致的变化进行叙述。

插入一个新结点与上述 ArithBi$^+$ 构造中一个元素的插入过程一致。

当从压缩文件中删除一个元素(z, id_d)时，需要从对应的索引结构 ArithBi$^+$ 中删除该元素；具体过程为首先在 ArithBi$^+$ 中查找 value=z 的结点，找到后(一定可以找到，设对应 ArithBi$^+$ 中的索引结点为 D)，另 D. extent←D. extent -$\{id_d\}$，若 D. extent=ϕ 则删除结点 D，并对 ArithBi$^+$ 进行调整。

元素值的修改实际上可等价地看做两个步骤的操作：第一步从 ArithBi$^+$ 中执行删除旧结点(即原有值)；之后再插入新结点(即修改值)。

4 基于 ArithBi$^+$ 的压缩 XML 查询

4.1 查询算法

基于索引结构 ArithBi$^+$ 的查询就是查找查询条件(区域 Q)包含二叉树中的哪些结点的问题，算法 Query_Binary_Probability_Tree 详细描述了如何完成一个查询过程。

算法3 Query_Binary_Probability_Tree

```
Input：查询区间为 L,R,索引树为 iTree
Output：结果集 rData
①Node=Root
②判断[L,R)与 Node. Value 的关系，有三种可能：a. 小于转③ b. 包含转④ c. 大于转⑤
③Node=Node. Left_Child,转②
④//分成左、右子树分别进行处理
4.1 BranchNode=Node
4.2 rData=rData+BranchNode. extent; //即 L<Node. value<R 的情形
A. 左子树处理
4.3 cNode=BranchNode. Left_Child,若 cNode 不为空则 4.4,否则转 B
4.4 If L=<cNode. Value 则 rData=rData + cNode. Right_SubTree. extent + cNode. extent ; 否则转 4.3
4.5 If L>cNode. Value 则 cNode=cNode. Right_Child,若 cNode 不为空则 4.4,否则转 B
B. 右子树处理
4.6 cNode=BranchNode. Right_Child,若 cNode 不为空则 4.7,否则转⑥
4.7 If R>=cNode. Value 则 rData=rData + cNode. Left_SubTree. extent + cNode. extent ; 否则转 4.6
4.8 If R<cNode. Value 则 cNode=cNode. Left_Child, 若 cNode 不为空则 4.7,否则转⑥
⑤Node=Node. Right_Child,转②
⑥输出 rData,结束。
```

下面摘要解释算法3。

说明一点，为简明起见，算法中，行③、⑤没有判断结点是否为空。行④处理结点 M. value∈[L,R)。由于 M 的 value 值包含在查询区间[L,R)中(查询命中)，行 4.2 将 M 所包含的扩展集中的所有元素并入结果集 rData；之后行 4.3～4.5 继续处理 M 结点左子树，行 4.4 判断 M 的左子结点(用 cNode 标记)的值与查询区间左边界 L 的关系，若 L=<cNode. value，则由于 R>M. value>cNode. value 且 M. value 大于 cNode 右子树中所有结点的值，故结点 cNode 及其右子树中所有结点(所有对应元素简记为 cNode. Right_SubTree. extent)均属于查询命中结点；行 4.6～4.8 处理 M 结点的右子树，类似地，行 4.7 判断 M 的右子结点 cNode 的值与查询右边界 R 的关系，若 R>=cNode. value，则由于 L<M. value<cNode. value 且 M. value 小于 cNode 左子树中所有结点的值，故结点 cNode 及其左子树中所有结点(所有对应元素简记为 cNode. Left_SubTree. extent)均属于查询命中结点。

由算法3得到结果集合 rData 后，由于 rData 中的元素标识实际上是元素在压缩文件中的位置索引(如元素的开始 Tag 偏移量及终止 Tag 偏移量)，因此最终结果生成程序只需要按照每个元素的位置索引取得对应元素的内容即可。还需要指出，最后要依据压缩信息对查询结果中的压缩元素进行复原以供最终的查询用户使用。

4.2 实验及结果分析

表1所列4个查询语句为实验中用到的查询，实验 XML 数据取自 DBLP[8] 中 Authors 部分。实验重点就 ArithBi$^+$ 的查询执行时间进行了实验。实验硬件环境为 PIII800MHz，内存 256MB，硬盘 80GB；软件环境为 Windows XP，实验算法用 Java 语言实现。

表1 实验用查询语句

Q1	//Author/Publication/AuthorInfo/AuthorName
Q2	/Authors/Author/Name
Q3	//AuthorName
Q4	//Author[ConfInfo]/Title

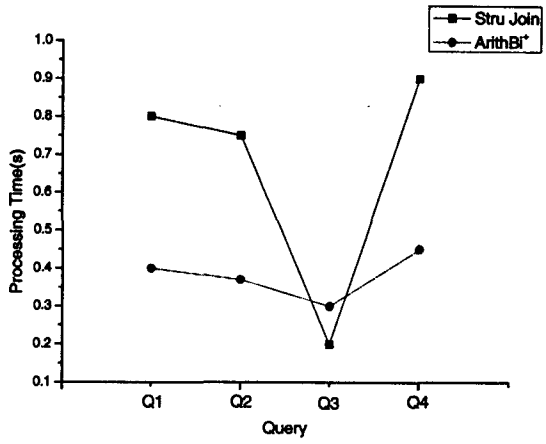

图 5 ArithBi+查询执行时间

图 5 中，我们利用基于 RDBMS 的 Structure Join 方法与 ArithBi+方法进行对比，可以看到，ArithBi+方法在各个查询语句(除 Q3)的表现都比前者要好很多。这是因为 Structure Join 方法的基本操作是 Join 操作，这种操作的代价是比较高的，并随着查询路径的增长所需要的 Join 次数越多。Q3 的情形比较特殊，这是由于 RDBMS 中是将同名的(如 AuthorName)存放在一张表中的，所以只需要一次选择操作就可以得到所有的结果。

5 相关工作

针对 XML 进行索引的工作有很多，但都不能直接用于经算术压缩的 XML 数据，参考文献中只列举了有代表性的几个[9~11]。XQzip[12]提出了一种有压缩功能的索引结构，但对基于算术压缩的 XML 数据不能进行处理；文[13]则提出了一种可根据压缩比率及查询能力选择有效 XML 压缩算法的查询处理系统，并没有讨论如何充分利用索引提高查询处理能力。

结论及展望 提出了一种压缩 XML 数据的索引结构 ArithBi+，应用该索引结构可以直接基于压缩 XML 文件进行查询，并可以较高效率实现对类似//element$_1$/element$_2$/…/elment$_m$ 语句的查询处理。后续的工作将在如何基于 ArithBi+进行复杂查询处理等方面开展进一步研究。

参 考 文 献

1 Liefke H, Suciu Dan. XMill: An efficient Compressor for XML data. In: Proc. of the 2000 ACM SIGMOD Int'l Conf. on Management of Data 2000, New York: ACM Press, 2000. 153～164
2 Tolani P M, et al. XGRIND: A Query-friendly XML Compressor. In: Proc. of the 18th Int'l Conf. on Data Engineering (ICDE' 02). Washington: IEEE Computer Society, 2002. 225～225
3 Min J-K, et al. XPRESS: A Queriable Compression for XML data. In: Proc. of the 2003 ACM SIGMOD Int'l Conf. on Management of Data, New York: ACM Press, 2003. 122～133
4 Papakonstantinou Y, Garcia-Molina H, Widom J. Object Exchange Across Heterogeneous Information Sources. In: Proc. of the 11th Int'l Conf on Data Engineering (ICDE' 95). Washington: IEEE Computer Society, 1995. 251～260
5 Lelewer D A, Hirschberg D S. Data Compression. ACM Computing Surveys, New York: ACM Press, 1987, 19(3): 261～296
6 Manolopoalos Y, Theodoridis Y, Tsotras V J. Advanced database indexing. Boston: Kluwer Academic Publishers, 1999. 61～81
7 Cormen T H, Leiserson C E, Rivest R L. Introduction to algorithms. Cambridge, MA: MIT Press, 1990
8 DBLP, XML Data. At http://dblp.uni-trier.de/xml/
9 Wang Haixun, Park Sanghyun, Fan Wei, Yu P S. ViST: Adynamic index method for querying XML data by tree structures. In: Proc. of the 2003 ACM SIGMOD Int'l Conf. on Management of Data, New York: ACM Press, 2003. 110～121
10 Chung C, Min J, Shim K. APEX: An adaptive path index for XML data. In: Proc. of the 2002 ACM SIGMOD Int'l Conf. on Management of Data, New York: ACM Press, 2002. 121～132
11 Goldman R, Widom J. DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases. In: Proc. of the 23th Int'l Conf. on Very Large Databases (VLDB'97). Athens: Morgan Kaufmann, 1997. 436～445
12 Cheng J, Ng W. XQzip: Querying Compressed XML Using Structural Indexing. In: 9th Int'l Conf. on Extending Database Technology (EDBT 2004), Heidelberg: Springer-Verlag, 2004. 219～236
13 Arion A, et al. Efficient Query Evaluation over Compressed XML Data. In: 9th Int'l Conf. on Extending Database Technology (EDBT 2004), Heidelberg: Springer-Verlag, 2004. 200～218

(上接第 63 页)

发起者没有和响应者有一致的 C_R 和 R 是因为我们不能保证最后一条消息一定到达发起者 I。这不影响安全性，因为对响应者 R 来说，他确信是与发起者 I 执行协议，与 I 共享秘密。对 I 来说，如果他在此认证基础上建立了安全联合，那么这个安全联合也一定是和 R 建立的，并且他们有一致的、秘密的密钥集。

结论 IKE2 协议的身份及消息认证是为了保证密钥的安全。我们证明了在两种认证方式下，发起者和响应者都能实现他们的认证目标。在分析上我们把 $SK\{h\}=\{h\}_{SK\text{-}e}$，$H_{SK\text{-}a}(HDR,\{h\}_{SK\text{-}e})$简化为 $SK\{h\}=\{h\}_{SK\text{-}e}$，这已经能够达到安全目标，增加的 $H_{SK\text{-}a}(HDR,\{h\}_{SK\text{-}e})$主要是验证整个消息的数据完整性。同时我们发现在主动攻击情况下，发起者的身份不能得到保护，我们认为在许多应用中需要为发起者提供主动攻击下的身份保护。其次我们可以看到虽然 IKE2 的消息结构非常复杂，但减少的消息交换回合无疑提高了协议效率，毕竟计算时间比网络传输时间要少的多，减少的消息回合也能更有效地抵抗拒绝服务攻击。IKE2 的另一个优点是协议的可选冗余度减少了，这有利于它的商业应用。

为了分析 IKE2 协议，我们扩展了串空间的诚实理论。由于增加了入侵模型，因此扩展串空间能够分析包含丰富密码原语的安全协议。本文的分析也是扩展串空间理论的一次实际应用，我们相信它能在更多的复杂协议分析中发挥作用。

参 考 文 献

1 Maneki A P. Honest Functions and their Application to the Analysis of Cryptographic Protocols. In: Proc. of the 12th IEEE Computer Security Foundations, Workshop[C], 1999. 83～89
2 Fábrega F J T, Herzog J C, Guttman J D. Honest Ideals on Strand Spaces. In: Proc. of the 11th IEEE Computer Security Foundations Workshop[C], 1998. 66～77
3 Fábrega F J T, Herzog J C, Guttman J D. Strand Spaces: Why is a Security Protocol Correct?. In: Proc. [C], 1998 IEEE Symposium on Security and Privacy, 1998. 160～171
4 Fábrega F J T, Herzog J C, Guttman J D. Strand Spaces: Proving Security Protocols Correct. Journal of Computer Security[J], 1999(7): 191～230
5 Harkins D, Kaufman C, Kent S, et al. Internet Key Exchange (IKEv2) Protocol. http://www.ietf.cnri.reston.va.us/internet-drafts/draft-ietf-ipsec-ikev2-11.txt. 2003. Work in progress
6 Moore K. The Internet IP Security Domain of Interpretation for ISAKMP. RFC 2407, November 1998
7 Maughan D, Schertler M, Chneider M, et al. Internet Security Association and Key Management Protocol (ISAKMP). RFC 2408, November 1998
8 Harkins D, Carrel D. The Internet Key Exchange (IKE). RFC 2409, November 1998
9 Aiello W, Bellovin S M. Efficient, DoS Resistant, Secure Key Exchange for Internet Protocols. In: Proc. of the ACM Computer and Communications Security (CCS) Conf, Washington, DC,. November 2002
10 Ferguson NSchneier B. A Cryptographic Evaluation of IPSec. http://www.counterpane.com/ipsec.html. 1999
11 Kaufman C, et al. Code-preserving Simplifications and Improvements to IKE. Internet Draft, Internet Engineering Task Force, July 2001
12 Kaufman C, Perlman R. Analysis of IKE. In: IEEE Trans. on Network Computing, November 2000