基于优化过滤策略的 XML 数据查询处理*)

陈海坤　李建中　骆吉洲
(哈尔滨工业大学计算机科学与技术系　哈尔滨 150001)

摘　要　如何高效地处理 XML 查询,是目前研究的热点。由于当前方法存在过多扫描无用节点引起效率下降的问题,本文设计了一种 XML 数据的二级索引结构,基于该结构给出路径查询处理算法。首先,本文对 XML 模式中每个节点按路径类型进行分类编码,然后把每个节点按该编码进行聚类存储。在查询时,就可以先根据模式信息和查询信息得到目标节点的编码,然后只需将二级索引中这些编码对应的部分载入内存,进行过滤操作。这样就不必扫描整个索引,提高 CPU 和 IO 效率。本文还对二级索引结构进行扩展,使本文的过滤索引能方便应用在有分支结构的查询上。实验结果表明,本文的 XML 数据过滤算法效率优于基于 Bit vector 的过滤算法,并且索引结构所需要的存储空间也小于 Bit vector 索引。

关键词　XML,数据过滤,路径表达式,模式图,位向量,二级索引

An Optimized Filter Strategy for XML Query Processing

CHEN Hai-Kun　LI Jian-Zhong　LUO Ji-Zhou
(School of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001)

Abstract　How to retrieve information user interested in has become a hotspot in research. In this paper, we propose a filter strategy to reduce the number of candidate nodes using double level index. In the filter algorithm, all nodes are coded according to their path type, and clustered based on their codes. In course of processing, we first calculate the target codes of the query according to the structure of XML data. Then we load parts of the index which target codes point to into memory, and use the filter expression to filter those definitely useless nodes without scanning entire index. Furthermore, we also extend the index structure to process twig pattern query for XML. The analysis and the experimental results show that our filter algorithm has a better performance than filter algorithm using Bit vector, and also needs less storage.

Keywords　XML, Data filter, Path expression, Data graph, Bit vector, Double level index

1　引言

XML 作为 Web 数据表示和交换的标准,越来越多的 Web 数据以 XML 格式进行存储和交换。目前,XQuery[1] 成为 w3c 推荐的标准,而路径表达式查询是 XQuery 的重要组成部分。因此,如何有效地对路径表达式查询进行处理,提高 XML 数据查询处理的效率成为一个重要的研究课题。

基于区间编码的 join 算法是处理路径表达式查询的一种重要方法,其中 Structural Join[2~4] 和 holistic twig matching[5] 是两种有代表性的方法。它们存在过多扫描无用节点的问题,从而降低查询效率。为此一些研究者提出了一些策略来过滤这些无用的节点[6,7],如建立基于区间编码的 XR-Tree,但这些方法不能完全过滤掉无用的节点,为此又提出了一种基于 Bit vector[8] 的索引,以进一步减少对无用节点的扫描。但基于 Bit vector 的过滤算法存在若干问题,如它未充分利用 XML 模式图的信息,在每个查询处理时都要对 XML 文档中所有的结点的索引扫描一遍,由于该索引相当大,对索引的完整扫描增大了 CPU 和 IO 的消耗;该算法不能高效地支持 twig pattern query,即带分支的查询。

本文提出的二级索引结构对 Bit vector 索引结构进行优化,可以有效降低在查询处理过程中的 IO,提高查询效率,并对二级索引结构进行扩展,使其能有效支持 twig pattern 查询。本文还给出基于该二级索引结构的查询处理算法。

本文第 2 节介绍基本概念;第 3 节介绍二级索引结构的建立;第 4 节介绍利用二级索引结构进行查询处理的算法;第 5 节是实验结果与分析;最后总结全文。

2　基本概念

XML 文档的模式可以定义成一个有向图,称其为 XML 模式图,图 1 是一个模式图实例。而 XML 模式图对应的 XML 文档可以用一棵树来表示。图 2 中的数据树就是图 1 模式图的一棵数据树,斜体数字为该节点在节点树中的先序编号。

文[8]提出的基于 Bit vector 的索引结构,对 XML 模式图中的每一个节点,不妨设为 a,都建立一个位向量。位向量的每一行是一个布尔值,说明 XML 数据树上从根节点到该行代表的节点的路径是否经过节点 a。所有位相量的同一行的布尔值放在一起就表示了该行对应的节点的路径信息。

*)该论文受到下列基金资助:国家自然基金重点项目,编号 60533110,黑龙江省自然科学基金重点项目,编号 zjg03-05,国家自然科学基金项目,编号 60473075,国家教育部新世纪创新人才计划,编号 NCEF-05-0333,黑龙江省自然基金,编号 F0208,哈尔滨市科技攻关项目,编号 2004AA1CG13213。

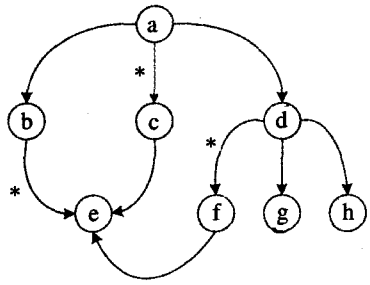

图 1　模式图实例

过滤表达式是形如 $a_1\&\cdots\&a_k\&\neg b_1\&\cdots\&\neg b_t$ 的布尔表达式。其中 a_1 到 a_k 是从根节点到目标节点必须经过的那些节点，而 b_1 到 b_t 是一定不经过的节点。这样对于给定的不带分支的查询，可以将过滤表达式中的节点对应的位向量载入内存，并利用过滤表达式对这些位向量的每一行进行求值，如果值为假则将该行代表的节点从候选节点集中过滤掉，为真则加入到结果集。文[8]给出了过滤表达式的求解算法。

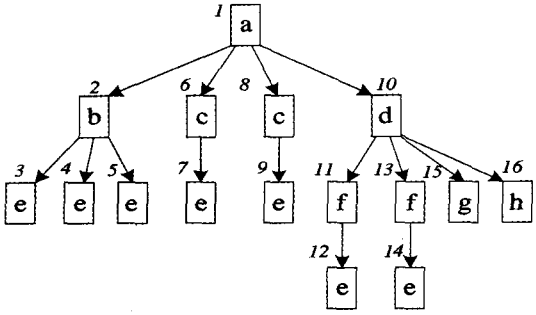

图 2　数据树实例

3　二级索引的建立

本文的过滤策略的目的是降低在过滤过程中的 IO 和 CPU 消耗，为了达到这个目的，分析 XML 模式图和路径查询的特点，发现可以对模式图中的路径进行分类，并且由路径查询可以得到模式图中对查询结果有贡献的路径类型。如果把 Bit vector 中的每行按照路径类型进行聚类存储，只用扫描对查询结果有用的路径类型所指的那些行即可得到结果。据此，本文对 Bit vector 索引结构进行改进，建立二级索引，以降低过滤时的 IO 消耗。

下面给出对 XML 模式图中路径的分类编码方法。

定义 3.1(关键节点)　模式图中的根节点和出度或者入度大于等于 1 的节点。

定义 3.2(非关键节点)　模式图中不是关键节点的节点。

考虑按照关键节点对模式图中的路径进行分类编码。设节点 $m_1,\cdots,m_k$ 为模式图中所有的关键节点，则路径类型的编码长度为关键节点的个数 k，编码是形如 $m_1\ m_2,\cdots,\ m_k$ 的二进制串。如果模式图中的一条路径经过关键节点 m_i，则在编码中置 m_i 位为 1，否则置 0。称路径的编码为该路径的类型。

下面对 XML 数据建立二级索引结构：

(1)找出模式图中所有的关键节点，对 XML 文档中的每个节点进行编码。节点编码方法为在该节点标签名后接上从根节点到该节点的路径的编码，中间以“.”分开。

(2)为每个非关键节点建立位向量(这个位向量就是 Bit vector 索引中的位向量)，并将数据树中的所有节点，按其节点编码聚类存储非关键节点的位向量。

(3)用节点编码为键来记录具有相同编码的节点在位向量中的起始终止位置。

对图 1 中的模式图以及图 2 中的数据树建立二级索引。建立新的索引如图 3 所示。

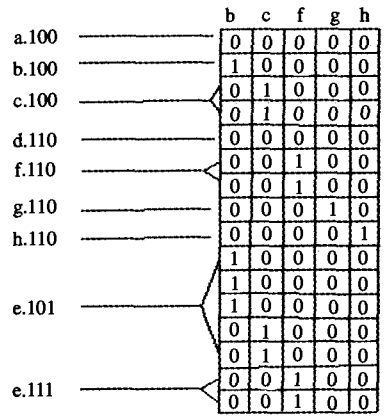

图 3　二级索引实例

关于二级过滤索引的性质，有如下结论：

引理 3.1　设对查询结果有贡献的全部节点编码为 t_1，$\cdots,t_k$，这些节点编码在二级索引结构中所指的行代表的节点集合为 A，满足查询条件的目标节点集合为 B，则有集合 B 是集合 A 的一个子集。

对给定的路径查询和模式图，可以计算模式图中对查询结果有贡献的节点的编码。文[8]给出了对单路径查询的过滤表达式的求解算法，并证明对候选节点集合应用过滤表达式进行过滤后得到的节点集合就是查询的结果节点集合。设过滤表达式为 $a_1\&\cdots\&a_k\&\neg b_1\&\cdots\&\neg b_t$，如果有关键节点属于查询路径一定经过的节点(在表达式中不带¬的节点)，则编码中该关键节点对应的那位置为 1；如果有关键节点属于查询路径一定不经过的节点(即在表达式中前面带¬的节点)，则路径的编码中该关键节点对应的那位置为 0；对于不出现在过滤表达式中的关键节点，在编码中对应的位可以是 0 或者 1。

根据引理 3.1，在得到对查询结果有贡献的节点编码集合之后，只需载入这些编码对应的那部分非关键节点的位向量进行过滤就可以了。实际上，当过滤表达式全部由关键节点组成时，无需扫描非关键节点的位向量，就可以直接得到满足查询的节点集合。

定理 3.1　对给定的不带分支的查询，当其过滤表达式全部由关键节点组成时，对查询结果有贡献的节点编码在二级索引结构中所指的行代表的节点集合就是查询结果。

4　基于二级索引的查询处理

4.1　单路径查询处理算法

对于不带分支的查询，可利用二级索引结构通过过滤候选节点快速得到结果。查询处理过程如下：首先利用模式图 G 和给定的查询 q 通过过滤表达式求解算法 FindFE(q, G)[8]得到过滤表达式 fe；再根据 fe 找出对查询结果有贡献的节点编码集合 $CodeSet$；去掉 fe 中的关键节点，得到由非关键节点组成的新的过滤表达式 $newfe$；如果 $newfe$ 为空，则由二

级索引找出 *CodeSet* 中编码的节点在位向量上的位置范围，并且将这些范围内的节点加入结果集；如果 *newfe* 不为空，则由二级索引找出 *CodeSet* 中编码的节点在位向量上的位置范围，并载入 *newfe* 中非关键节点位向量上在这些范围内的那些行，逐行代入 *newfe* 进行计算，如果计算结果为真，则将该行所代表的节点放入结果集，否则过滤掉。下面给出具体的算法描述。

算法：SPFilter(q,G)
输入：q 是不带分支的查询
　　G 是模式图
输出：查询结果集合 L
开始：
1：fe=FindFE(q, G)；
2：keyNodeList=GetKeyNode(G)；
3：CodeSet=GetCode(keyNodeList, fe)；
4：newfe=GetNewFE(keyNodeList, fe)；
5：if newfe=null
6：　for each c∈CodeSet do
7：　　找出编码 c 在位向量上的位置范围，并且将这些范围内的节点加入结果集；
8：else
9：　for each c∈CodeSet do
10：　　找出编码 c 在位向量上的位置范围，将该范围内的非关键节点位向量载入内存；
11：　　对载入的部分位向量的每一行
12：　　If(GetFilter(newfe))
13：　　将该行节点加入结果集
结束。

直观上看，算法时间复杂度为 $O(n)$，其中 n 为位向量的行数。另外过滤表达式求解算法的时间复杂度为 $O(m^2)$，m 为模式图中标签的个数。由于该算法利用了二级索引结构，因此不需要载入整个索引，所以在通常情况下 IO 仅为基于 Bit vector 索引结构算法的一小部分，尤其是在过滤表达式全部由关键节点组成时，IO 降到最小。实验和分析，在第 5 节给出。

4.2 带分支查询处理算法

对于带分支的查询，利用二级索引结构也可以快速得到查询结果。基本的带分支的查询，可以分解成三个单路径查询：从根节点到分叉节点、从根节点到分支末端节点、从根节点到目标节点。对这三个单路径查询应用基于二级索引结构的过滤策略，得到三个候选节点集合，再在这三个候选集合上做 twig pattern join[5] 就能得到查询需要的结果。

为了使二级索引结构能支持带分支的查询，记录 XML 数据树上节点的先序后序编码，并且将这两个编码组织成两列，分别为 pre 列和 post 列，添加到二级索引结构中，得到新的二级索引结构，其中 pre 列代表对应行的节点的先序编号，post 列为对应行的节点的后序编号。先序后序编码只需一次遍历 XML 数据树就可以得到。

利用修改后的二级索引，可以方便地支持带分支的查询，并且可以有效避免扫描无用的节点，从而减小查询的中间结果，提高查询效率。

5 实验结果及分析

本文采用 C++ 实现该算法，采用 XML benchmark Xmark 数据集为实验数据。实验中，对基于二级索引结构的查询处理算法和基于 Bit vector 的查询处理算法对存储空间的需求，查询效率进行对比。

索引的大小：对 factor1.0 的 XMARK 数据集合建立二级索引结构，索引结构大小为 12.4MB，而基于 Bit vector 的索引结构未压缩情况下占用空间为 20.0MB，所以本文的二级索引结构在存储空间上要优于基于 Bit vector 的索引结构。

查询效率：为测试算法过滤效果和时间效率，针对实验用的 XMark 数据集合，设计如下四个查询：

Q1：//regions//item/name　Q2：//regions/Asia/item
Q3：//category/description　Q4：//person/address

实验结果：从表 1 和表 2 看出，查询 Q1 和 Q3 的时间消耗都在 10ms 以内，并且 IO 次数都为 6，这是因为这两个查询的过滤表达式都是由关键节点组成，可以直接得到满足查询的节点。而查询 Q2 和 Q4，它们的过滤表达式中包含非关键节点，在获得满足查询的节点的编码之后，还需要载入非关键节点满足这些编码的那部分位向量，通过位运算来剔除这些编码的节点中不满足查询的节点，因此所需的 IO 次数和时间消耗都有所增加。

表 2　查询时间比较　(ms)

	Q1	Q2	Q3	Q4
二级索引	<10	15	<10	15
Bit vector 索引	71	44	21	68

表 3　IO 次数比较　(次数)

	Q1	Q2	Q3	Q4
二级索引	6	48	54	6
Bit vector 索引	550	200	200	520

结论　本文提出的基于二级索引结构的 XML 数据过滤算法，首先对 XML 数据中的每个节点按照路径类型进行编码，并按编码聚类存储。根据节点的编码和节点的路径建立二级索引。查询时，利用过滤表达式先得到满足查询的编码，只需载入非关键节点满足这些编码的那部分位向量，对这些位向量进行计算，就可以得到查询结果。该算法不必扫描整个位向量，比基于 Bit vector 的过滤算法时间和 IO 性能上有很大的提高。本文还对新的二级索引结构进行扩展，使本文的过滤算法能方便应用在带分支约束的查询上。最后的实验结果表明，本文的过滤算法可以大量降低 XML 查询要处理的数据量，索引文件占用的空间小，同时整个过滤过程时间效率相当高。

参 考 文 献

1　Boag S, Chamberlin D, Fernandez M F. XQuery 1.0: An XML Query Language. W3C Working Draft. http://www.w3.org/TR/XQuery

2　Chien S Y, Vagena Z, Zhang D. Efficient Structural Joins on Indexed XML Documents. VLDB 2002, 2002. 263～274

3　Jiang H, Lu H, Wang W, Ooi B C. XR-Tree: Indexing XML Data for Efficient Structural Joins. ICDE 2003, 2003. 253～264

4　Li Q, Moon B. Indexing and Querying XML Data for Regular Path Expressions. VLDB 2001, 2001. 361～370

5　Bruno N, Srivastava D, Koudas N. Holistic Twig Joins: Optimal XML Pattern Matching. SIGMOD 2002, 2002. 310～321

6　Jiang H, Lu H, Wang W, Yu J X. Holistic Twig Joins on Indexed XML Documents. VLDB 2003, 2003. 273～284

7　Jiang H, Lu H, Wang W. Efficient Processing of XML Twig Queries with OR-Predicates. SIGMOD 2004, 2004. 59～70

8　He Zhenying, Li Jianzhong, Chen Haikun. Using XML Structure to Reduce Candidate Nodes Participated in Query Processing. WAIM, 2005