纯 XML 数据库研究综述*

冯建华, 钱　乾, 廖雨果, 李国良, 塔　娜, 周立柱
(清华大学 计算机科学与技术系, 北京 100084)

摘　要: 虽然 XML 是一种专门为 Internet 所设计的标记语言, 但是它已经成为 Internet 上数据表示和数据交换的标准。如何对 XML 文档进行有效管理与快速查询是当前学术界的研究热点, 即所谓的 XML 数据库。主要对当前 XML 数据库的研究现状与发展趋势进行全面的论述与分析, 并对其中尚未解决的挑战性问题进行了客观的评价与分析。

关键词: 扩展标记语言; XML 数据库; Native XML 数据库; 存储策略; 查询优化; X-代数
中图法分类号: TP311.13　　**文献标识码**: A　　**文章编号**: 1001-3695(2006)06-0001-07

Survey of Research on Native XML Databases

FENG Jian-hua, QIAN Qian, LIAO Yu-guo, LI Guo-liang, TA Na, ZHOU Li-zhu
(Dept. of Computer Science & Technology, Tsinghua University, Beijing 100084, China)

Abstract: Although XML is a special markup language for Internet, it has been become the standard for data representation and exchange on Internet. With the emergence of a lot of XML documents, how to manage and query them effectively has been become a hotspot for database researchers. This paper discusses and analyzes the status quo and trend of XML database research, and gives the evaluating to the open and challenging problems in XML database research.

Key words: XML; XML Database; Native XML Database; Storage Strategy; Query Optimization; X-Algebra

1　引言

1.1　认识 XML(eXtensible Markup Language)

XML 是一种专门为 Internet 所设计的标记语言。XML 的重点是管理信息的数据本身, 而不是数据的样式, 数据的显示则交给另外的技术来解决。虽然 XML 是一种专门为 Internet 所设计的标记语言, 但是它已经成为 Internet 上数据表示和数据交换的标准。随之而来的是堆积如山的 XML 文档, 如何对这些 XML 文档进行有效管理与快速查询, 是当前学术界的研究热点, 即所谓的 XML 数据库。

1.2　XML 数据库与 XML 文档

所谓数据库就是一组相互有关联的数据集合, 而 XML 数据库是一个 XML 文档的集合, 这些文档是持久的并且是可操作的。目前 XML 数据库有三种类型(参阅 http://www.xmldb.org/faqs.html):

(1) XML Enabled Database(XEDB), 即能处理 XML 的数据库。其特点是在原有的数据库系统上扩充对 XML 数据的处理功能, 使之能适应 XML 数据存储和查询的需要。一般的做法是在数据库系统之上增加 XML 映射层, 这可以由数据库供应商提供, 也可以由第三方厂商提供。映射层管理 XML 数据的存储和检索, 但原始的 XML 元数据和结构可能会丢失, 而且数据检索的结果不能保证是原始的 XML 形式。XEDB 的基本存储单位与具体的实现紧密相关。

(2) Native XML Database(NXD), 即纯 XML 数据库。其特点是以自然的方式处理 XML 数据, 以 XML 文档作为基本的逻辑存储单位, 针对 XML 的数据存储和查询特点专门设计适用的数据模型和处理方法。

(3) Hybrid XML Database(HXD), 即混合 XML 数据库。根据应用的需求, 可以视其为 XEDB 或 NXD 的数据库, 典型的例子是 Ozone(参阅 http://www.ozone-db.org/frames/home/what.html)。

XML 文档有两种类型: 面向文档处理和面向数据处理[1]。面向文档处理的文档是利用 XML 来获取自然(人类)语言的那些文档, 如用户手册。它们以复杂或无规则的结构和混合内容为特征, 而且文档的物理结构非常重要。这些文档的处理侧重于给用户提供信息的最终表示, 因此它们也被称作面向表示的文档。面向数据处理的文档主要利用 XML 来传送数据, 这些文档包括销售订单、病人记录和科学数据等。面向数据处理的 XML 文档的物理结构, 如元素的顺序, 或者数据被存储为属性还是子元素, 通常不是很重要。它们的特征是高度有序的结构, 并且同时带有那些数据结构的多个副本, 类似于关系数据库系统中的多条记录。这些文档的处理通常侧重于应用程序间的数据交换, 因此它们也被称作面向消息的文档。这两类文档有时也简称为面向文档的(XML)文档和面向数据的(XML)文档。

面向数据处理和面向文档处理的 XML 文档之间的区别不大, 然而文档上需要执行的操作是不一样的。在一个面向文档

收稿日期: 2005-09-09; 修返日期: 2005-11-03
基金项目: 国家自然科学基金资助项目(60573094); 清华大学基础研究基金资助项目(JCqn2005022)

处理的 XML 文档中,需要执行的操作包括检索整个文档、搜索关键词、修改一部分或重排一部分。在一个面向数据处理的 XML 文档中,需要执行的操作包括检索文档的一个指定部分、搜索元素和数据的一个特定的结合、修改或删除一个简单的元素或一块简单的数据,或者给文档添加一个新元素。

1.3 XML 数据库研究的问题

与传统的关系数据库管理系统(RDBMS)一样,XML 数据库管理系统也要解决 XML 数据的存储与查询处理问题,即存储引擎与查询引擎(同关系引擎一样,将其称为 X-引擎)。在 X-引擎中主要研究 XML 数据的查询分解与优化,包括 XML 数据查询的代数表示,同关系代数一样。将其称为 X-代数。

2 XML 数据库的存储策略分析

首要问题是如何存储 XML 文档才是最好的?因为底层的存储表达对上层的查询处理和优化有着重要的性能影响[2]。因此,要实现 XML 数据库,首先必须解决 XML 文档的存储问题。根据已有的文献[3~6],XML 数据库的存储策略目前主要有以下三种:利用文件系统的平面文件、利用成熟的 RDBMS、利用对象管理器或面向对象数据库管理系统(OODBMS)。

2.1 文件系统的平面文件方法

XML 文档本质上是序列化数据。序列化数据通常采用平面文件的形式,即将每一个 XML 文档分别存储在一个文本文件里[3],并且实现一个查询引擎,当查询被执行的时候,XML 文件被解析成驻留在内存的一棵树。只要查询计算还需要树中的节点,这棵树就必须驻留在内存里。一般来说,解析的时间左右着查询计算的时间,而且这种方法慢得令人无法接受。为了提高这种方法的可用性,必须建立以下索引:利用 XML 元素在文本文件中的偏移量作为 ID,建立从标记 tag(Parent_offset, Tag) 映射到子偏移量 Child_offset 的路径索引以及从标记 tag(Child_offset, Tag) 映射到父偏移量 Parent_offset 的反向路径索引,这两个索引非常有利于遍历 XML 文档或在 XML 文档中进行导航。而其他的索引建立从标记值(Tagname, Value) 或属性值(Attribute_name, Attribute_value) 到元素偏移量 Element_offset 的映射,这些索引有助于计算查询中的选择条件(谓词)。查询引擎可以利用这些索引检索与查询相关的 XML 文件的片段,极大地减少了解析时间[2]。平面文件数据库最大优点就是容易实现,而且不需要使用数据库系统和存储管理器。但是,利用平面文件存储 XML 文档也有以下缺点[2]:

(1) 每次访问 XML 文档时都需要解析它;

(2) 在查询处理期间,整个被解析的文件都必须驻留在内存里;

(3) 为了部分解决上述两个问题,可以在 XML 文档上建立外部索引。但当 XML 文档被更新时,索引是难以维护的。

2.2 传统的关系数据库系统方法

除了平面文件以外,还可以将 XML 文档存储在传统的关系数据库系统里。最近的研究成果[4~6]说明了如何将 XML 文档映射并存储到关系数据库系统中。这种方法的缺点是当前关系数据库系统与 XML 的负载不协调,而且通过诸如 SQL 的接口访问 XML 数据会招致与存储无关的额外负担[2]。

2.2.1 利用关系的 DTD 方法

利用关系的 DTD 方法是文献[6]建议的共享内嵌法,而且需要相应 DTD 的支持。用一个独立的表来捕获具有相同标记的元素与其子元素集之间的包含关系,表中的每个元组都被赋予一个 ID,而且该表还包含一个 ParentID 列以标志该元组的父亲节点。一个元素在描述它的元素表中只能出现一次,而且它的父节点内嵌为表的一列以便表示该元素的父亲。如果描述 XML 文档模式的 DTD 图中包含环,则必须用一个独立的表来打破这个环。在这种情况下,XML 数据库的索引可以完全建立在上述各个关系表上。

2.2.2 利用关系的边的方法

文献[4]描述了边的方法,将 XML 文档的有向图存储在单个 Edge 表中。按照广度优先顺序赋予有向图中每个节点一个唯一的 ID。Edge 表中每个元组对应有向图里的一条边,并且包含该条边所连接的两个节点的 ID 值,目标节点的标记(Tag) 值以及一个顺序号,该顺序号表示同一节点与其多个子节点对应边的顺序编码。当元素只有一个文本子节点时,文本内容就内嵌到以该元素节点为目标节点的边所对应的元组里[2]。

按照文献[4]的建议,建立在(Tag, 数据) 上的索引可以缩减选择查询的执行时间。而建立在(源节点, 顺序号) 和(目标节点) 上的索引也非常重要,前者用来查找给定元素的子元素,后者用来从给定的子节点向父节点遍历。

在 Edge 表上运用簇集策略对查询的性能也会有重要影响。一般都会选择按照 Edge 表上的 Tag 字段进行簇集,使得具有相同 Tag 的元素存储在一起。当然也可以选择按照“源节点”字段进行簇集,这种策略的好处是一个 XML 元素的所有子元素都存储在一起,因此重构原始的 XML 文档将非常快。但它的缺点是具有相同标记名的元素没有被簇集,因此对于这样的查询:“把专业是计算机科学的所有学生都选择出来”将会导致大量的随机 I/O。而建立在 Tag 字段上的簇集则表现出了较好的性能,但它对 XML 文档的重构没有任何好处[2]。

2.2.3 利用关系的属性方法

文献[4]还建议了另外一种所谓的属性方法。它是按 Edge 表的 Tag 字段对边表进行水平划分,不同标记(指 Tag 字段的值) 的元组存储在各自不同的表里。这种方法以牺牲 Edge 表非常重要的属性 Tag 为代价而节省了存储空间。对属性方法来说,查询处理器需要 DTD 来决定哪些表包含子元素,因为子元素的标记没有存储在表中。需要注意的是,对于有很多 XML 文档的一个大集,属性方法可能导致大量的表[2]。

2.3 对象管理器方法

在对象管理器里存储 XML 文档的明显方法就是把每一个 XML 元素存储成一个独立的对象[5],但是由于 XML 元素通常都非常小,因此这种方法的空间开销高得令人不敢问津。取而代之,把 XML 文档的所有元素存储在一个单独的对象里,而 XML 元素本身就变成了这个对象里的轻量级对象[2]。同时,文献[2]用 lw_object,即记录来表示轻量级对象,而用 file_object 表示代表整个 XML 文档的对象。轻量级对象(lw_object) 在文件对象(file_object) 中的偏移量(Offset) 被用作该对象的

标志(lw_oid) 。

3　不同存储方法的分析与比较

3.1　不同存储方法的优缺点比较

平面文件是存储 XML 文档的最简单的机制, 但一般不支持索引查询, 也不容易修改文档。关系型或面向对象数据库按照一定的粒度来存储 XML 文档, 这使得对 XML 文档的访问比较容易也比较灵活, 同时提高了查询和修改 XML 文档的效率, 而且还可以方便地建立和维护各种索引。

3.2　不同存储方法的性能比较

上面实际上介绍了存储 XML 文档的五种方法, 文献[2]通过实验数据比较了这五种方法的性能。实验使用两个数据集: 第一个数据集包含 250 个 XML 文档, 总共 114MB[7]; 第二个数据集是 ODP, 总共 140MB(参阅 Open Directory Project, http: / /www. dmoz. org/) 。实验所用的关系数据库为 DB2 v7. 1, 对象方法的底层利用系统 Shore[8] 来实现; DB2 和 Shore 都配置使用 30MB 的内存缓冲池。文本方法中没有缓冲池, 其查询处理器使用所有可用的 256MB 物理内存; 文本方法中的索引利用 Berkeley DB(参阅 Berkeley DB Toolkit, http: / /www. sleepycat. com/) 来实现。在利用关系数据库的 DTD, Edge 和属性方法中, 将用 XQuery 表达的查询手工翻译成 DB2 可执行的 SQL 查询语句[2]。

通过对存储比、XML 文档的重构和查询速度等指标的测试, 初步的实验结果表明要想获得好的性能和紧凑的数据表示, 相关 XML 文档的 DTD 信息是至关重要的。当 DTD 信息可利用时, DTD 方法的数据表示就很紧凑, 而且该方法对不同的数据集和不同的查询均表现出优秀的性能。而利用适当的索引, 文本方法可以获得与对象管理器方法相似的性能。但是, 维护索引的代价将使这种方法只适用于 XML 文档更新不频繁的时候[2]。

3.3　利用关系的 DTD 方法不是 XML 数据库的唯一出路

由于利用关系的 DTD 方法在性能上获得了巨大的优势, 有相当多的研究人员将时间和精力花费在把 XML 模式转换成关系模式上, 似乎把利用关系的 DTD 方法看成了 XML 数据库的唯一出路。其实, 利用关系数据库系统存储 XML 文档还是有很多缺陷:

(1) 有很多没有 DTD 的 XML 文档或者是 XML 文档被用作标记语言;

(2) 一个 XML 文档一定要有 DTD, 不符合半结构化数据或非结构化数据的本意;

(3) 根据 DTD 可以很容易转换成相应的关系模式, 从本质上讲这样的数据不是非结构化或半结构化数据, 而是复杂的结构化数据;

(4) 传统的 RDBMS 不能有效解决 XML 数据的查询优化问题, 它们之间存在着天然的 “阻抗失配 ”。

那么 XML 数据库的出路到底何在呢?

4　XML 数据库研究趋势: Native XML 数据库管理系统

4.1　为什么要 Native

早期的 XML 数据以文档(文本文件) 的方式存储, 以关键字查询等信息检索手段进行查询, 简单易用, 适合 XML 文档不频繁更新的场合。由于缺乏系统的存储和查询机制的支持, 造成查询能力低, 不能满足复杂条件的查询, 更谈不上查询优化。一些现有的商业数据库管理系统扩充了处理 XML 数据的功能。利用现有数据库成熟的技术, 把 XML 查询要求转换为数据库的查询表达, 如 SQL, 由数据库的查询引擎优化查询表达, 产生查询执行计划并执行, 最后再将查询的结果转换为 XML 数据。这种方法在一定程度上解决了查询复杂性的要求, 但是多次转换带来的问题是效率的降低和查询语义的混淆[9], 即阻抗失配的问题。

(1) 从本质上讲, 传统的 RDBMS 不支持层次的和半结构化的数据形式, 只有经过转换处理才能把嵌套的 XML 数据放到简单的关系表中。XML 是具有动态结构的数据, 这正是 XML 可扩展性的关键, 而传统的 RDBMS 不能处理这种数据。

(2) SQL 适合于查询受确定模式支配的表, 它不是为具有动态、复杂特性的 XML 设计的。

(3) 传统的 RDBMS 查询引擎需要把针对 XML 的查询翻译成很多关系表的检索和连接运算, 这不仅造成查询性能的下降, 而且往往不能产生代价较小的查询执行计划。

Native XML 数据库管理系统(纯 XML 数据库管理系统, NXDBMS) 以自然的方式处理 XML 数据, 没有因数据模型转换而带来信息丢失和性能下降。NXDBMS 与非 NXDBMS 的区别在于[9]: ①有效地支持 XML 数据的自描述性、半结构化和有序性; ②系统直接存储 XML 数据, 而不是把 XML 数据转换成关系模型或者面向对象模型, 由关系数据库或面向对象数据库存储; ③直接支持 XML 查询语言, 如 XQuery, XPath, 而不是转换成 SQL 或 OQL(对象查询语言) 。

4.2　Native XML 数据库

什么是真正的 Native XML 数据库(纯 XML 数据库, NXD) ? Ronald Bourret 在 “XML and Databases ”一文中给出有关 NXD 的定义[1]。一个纯 XML 数据库是指:

(1) 相对于 XML 文档中的数据, 定义了 XML 文档的逻辑模型, 并且按照该模型来存储和检索文档。这样的模型至少应该包括元素、属性、PCDATA 以及文档顺序。如 Xpath 数据模型、XML Infoset(参阅 http: / /www. w3. org/TR/xml-infoset/) 以及由 DOM 和 SAX 1. 0(参阅 http: / /sax. sourceforge. net 和 http: / /www. saxproject. org) 中的事件所隐含的模型都是这类数据模型。

(2) 就像关系数据库以行作为表的逻辑存储基本单位一样, NXDBMS 以 XML 文档作为 NXD 的逻辑存储基本单位。

(3) 不要求有任何特殊的基本物理存储模型, 它可以建立在关系的、层次的或面向对象数据库之上, 或者使用诸如索引文件、压缩文件此类的专门存储格式。

从这个定义至少可以简单地总结以下三点: ①纯 XML 数据库是专门用来存储 XML 数据的, 而且完整无缺地存储 XML 模型的所有成分; ②文档进得去出得来; ③NXD 可能根本就不是真正独立的数据库。

随着对 NXD 的认识与技术的发展, 人们越来越不认同上述定义的第三点。在这一方面, 德国的 Software AG(参阅 http: / /www2. softwareag. com/) 公司走在了前面。他们认为建立

和设计 NXD 系统是用来处理 XML 数据的,而不仅仅是在任意数据模型的数据库系统之上附加一层对 XML 的处理。用户可能一眼看不出它们之间有什么差别,但最根本的不同是在系统的内部。在很多方面,XML 都不同于其他著名的数据模型(如关系模型、面向对象模型)。将 XML 映射到另一种数据模型常常引起"阻抗失配",并导致功能和性能上的局限[10]。因此,NXD 必须直接存储 XML 数据。

4.3 NXDBMS 的体系结构

NXDBMS 是目前 XML 研究领域的一个热点问题,也是核心问题,已经出现了一批相对独立的系统。从查询方法上可以把 NXDBMS 分为两种:①大量地吸收了文档处理和编程语言思想而不是数据库处理数据的思想,如 Kweelt[11], X-Hive(参阅 X-Hive Corp (2002) X-Hive/DB, http://www.x-hive.com)等,采用导航的方式遍历 XML 文档,把长路径转换为嵌套循环的数据搜索。这种方法在处理的数据集相对较小时效率很好,但随着数据量的增大,其效率急剧下降。例如,Xindice(参阅 ApacheWeb Site (2002) Item 10 on Xindice FAQ, http://xml.apache.org/xindice/faq.html)就规定其能够处理的文档大小不能超过 5MB。②真正基于数据库思想,支持描述化查询语言和一次一集合的查询处理方式,提供完整的查询优化策略。典型的系统有 Tamino[12], Timber[13], Natix[5] 等。

与 RDBMS 一样,NXDBMS 的体系结构也分为用户接口、查询处理、存储管理和物理存储四大部分。同时,系统的事务管理、并发控制和故障恢复都与 RDBMS 有着相同的原理。

5 Native XML 数据库的挑战性问题

5.1 NXD 的物理存储

将元数据、XML 数据、索引和统计数据如何放置在物理磁盘上永远是一个挑战性的问题,而如何物理存储 XML 文档的数据尤其关键。研究 NXD 的物理存储的另外一个重要原因是因为相当多的时候 NXD 要用到簇集策略,而很多 RDBMS 或对象管理器并不支持用户对数据进行簇集。

5.1.1 数据模式问题

NXD 既能存储合法的 XML 文档(有 DTD 或 XML Schema 支持),又能存储良构的 XML 文档(无 DTD 和 XML Schema 支持)。在 XML 文档的 DTD 或 XML Schema 不可利用的情况下,NXDBMS 还能够存储和检索任何 XML 文档。任何一个 XML 文档都可以表示成一个树型结构,如图 1 所示。由于 XML 数据具有自描述性,因此对于没有 DTD 或 XML Schema 支持的 XML 文档可以自动抽取其模式信息[14],这样的模式可以用一个有向图来表示,如图 2 所示。当然 DTD 和 XML Schema 本身也可以用有向图来表示。

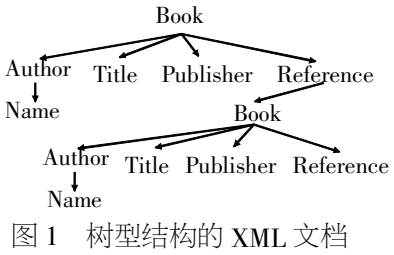


图 1 树型结构的 XML 文档

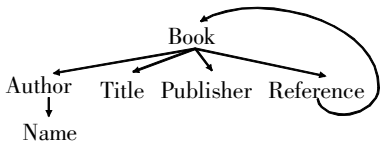


图 2 XML 文档的模式:有向图

由于 XML 数据的半结构化,数据与模式信息之间并不具有完全对应的关系,这时的模式信息只是作为查询或者了解数据的一个指导,并不具有约束数据的功能。而在传统的数据库中,数据与模式信息保持完全的对应关系,模式对数据具有约束能力,模式严格地约束数据的类型、操作和结构,数据的查询、更新和存储都要遵循模式的定义。因此,在 NXDBMS 中如何发挥 XML 模式的作用是一个挑战性的问题。

5.1.2 存储粒度问题

在数据模型上,XML 数据是一棵由各种节点组成的树,常见的节点包括元素(Element)节点、属性(Attribute)节点和文本(Text)节点。但是 NXD 中,一条记录所对应的子树有多大、包含多少个什么样的节点,这便是 NXD 的存储粒度问题。不同的存储粒度,记录的物理含义不同,可能是一棵子树,也可能是一个节点,还有可能是整个 XML 文档。从原理上讲,NXD 中记录的存储粒度有三种:

(1) 粗粒度。一个 XML 文档是一条记录,文档是作为一个整体来操作的,不需要有 DTD 或 XML Schema 等模式信息的支持。Apache Xindice(参阅 http://xml.apache.org/xindice/)就是粗粒度存储的典型例子。

(2) 细粒度。XML 文档的每一个节点就是一条记录,也不需要有文档模式信息的支持。节点可以分为广义节点和狭义节点[14]。

(3) 中粒度。XML 文档的一个片段(即一棵子树)是一条记录。子树的方法比较特殊,因为它存在一个如何划分子树的问题。目前,有两种划分子树的方法[9,14]:①根据物理块大小,使子树的大小与物理块大小相近,不需要文档模式信息的支持;②根据逻辑意义划分子树,使子树成为一个比较完整的逻辑单位,需要文档模式信息的支持。

不同的粒度对存储空间和查询的支持各不相同。对同一个 XML 文档来说,记录的粒度越细,则记录数目越多,用于表示记录之间关系的指针(逻辑指针或物理指针)就越多,从而耗费较大的存储空间,并且当 XML 文档很大时,重新生成整个文档会耗费大量的时间。但是细粒度的存储方法使每个元素和属性,包括文本都可以被单独查询、修改或删除,且对其他文档结构影响最小,具有最大的灵活性,也无须利用文档的模式信息。相反,记录的粒度越粗,为了读取某一个节点而一并读进来的节点数就越多,而这些读进来的节点可能对查询没有任何用处[14]。但是粗粒度的存储方法不需要存储过多的逻辑指针和物理指针,能够节省存储空间,而且重构整个文档会比较快一些。因此,如何根据查询的要求来确定 XML 数据的存储粒度也是一个挑战性的问题。

5.1.3 存储顺序问题

如果 NXD 中存储 XML 文档的记录粒度是狭义的节点或者子树,那么这些记录在物理块中是如何组织的呢?这就是记录的存储顺序问题,存储顺序是指记录在物理上的相邻关系。记录的存储顺序一般有以下几种[14]:

(1) 按深度优先顺序存储,这是最常见的存储顺序。

(2) 按广度优先顺序存储,这种顺序有点分层存储的意思。

(3) 按某个条件簇集存储,将满足某个条件的所有记录存储在同一个物理块内或物理上相邻的物理块内。对于 XML 文档来说,一般是将同类记录簇集存储,所谓同类记录是指记录的节点类型相同的记录。这种方法实际上在物理存储上破坏

了节点的顺序,但是对查询处理是有利的,并且在更新数据时无须在数据的中间插入,非常有利于数据的更新。

根据存储粒度和存储顺序,将会产生以下几种具体的存储方法[14]:①基于元素的深度优先(DEB)方法;②基于子树的深度优先(DSB)方法;③基于元素的广度优先(BEB)方法;④基于子树的深度优先(BSB)方法;⑤基于元素的同类簇集(CEB)方法;⑥基于子树的同类簇集(CSB)方法。

如何确定 NXD 中记录的存储顺序,以及如何针对不同的应用选取不同的存储方法,或者是针对不同的查询选择不同的存储方法,这些都是挑战性的问题。

5.2　NXDBMS 的查询处理

基于数据特征提出问题,从数据库中选择和综合数据的过程就是查询。在数据库中频繁访问的仅仅是部分信息,如果尽可能地明确表示用户最需要的信息,则能够有效地提高查询效率。所以查询处理与数据的物理存储和索引是密切相关的。根据查询的拓扑结构,XML 数据库的查询类型可以分为:①节点查询,根据唯一标志符在 XML 文档中检索单个元素节点;②路径查询,根据 XML 路径规范,即 Xpath 定义的路径(从根节点到一个叶子节点)检索一个或多个节点;③树查询,根据 XML 的树模式检索一系列节点或节点表;④图查询,根据 XML 的图模式检索一系列节点或节点表。

XML 的图模式扩展了 XML 路径支持的层次查询,增加了查询的复杂性和灵活性。当节点被文档内链接(通过 IDREF)或者文档间链接(通过 XML 链接: XLink)相连接时,XML 文档的一个树结构就成为一个有向图结构。XML 文档的逻辑模式应该是一个有向图,这个有向图可以通过适当的变化而成为一棵树。

虽然 Native XML 数据库管理系统的事务管理、并发控制、故障恢复和安全问题与 RDBMS 一样,但是对基于 XQuery 语言的查询处理是必须的,这也是实现 NXDBMS 的关键所在,即查询引擎或 X-引擎。在这里将需要潜心研究的核心问题逐一介绍如下。

5.2.1　定义完善、合理的查询代数

从理论上讲,查询代数是对遵循一定数据模型的 XML 文档集合的操作集,这样的查询代数可以简称为 X-代数(与关系代数相对应),具有一次一集合的查询能力。而在 W3C 的最新规范中也给出了“形式语义(Formal Semantics)”,即从形式上定义语言的底层代数,但这种代数是基于导航方式的,而不是数据库研究者所期望的一次一集合的 X-代数。W3C 中 XQuery 的形式语义的操作对象也已经由集合转变为序列,在这两者之间还有哪些问题没有解决? XQuery 的形式语义的其完备性和保真性的问题都需要认真研究。再有,X-代数应该是什么样子的? 包含哪些操作? 其完备性和保真性又如何? 这些更值得研究。

5.2.2　XML 查询分解

XML 查询语言,如 XPath 和 XQuery 都是将路径表达式作为核心内容。针对路径查询的处理问题,人们已经进行了大量的研究工作。在树状的 XML 数据中匹配路径查询的基本方法就是对数据进行导航式的遍历,文献[15,16]对这种方式进行了探讨。导航的方式简单直接,但执行效率不能得到保证,尤其是在大数据量的情况下。导航式遍历方法的低效性促使了类似于关系数据库中一次一集合的查询处理策略的出现[17]。NXDBMS 的查询分解也是将 XQuery 表达的查询转换为由 X-代数运算构成的查询树,具有一次一集合的操作特征。但是,复杂路径表达式的处理还有哪些更好的方法? 在从 XQuery 到查询代数树的转换过程中,有哪些有关 X-代数的等价规则可以利用? 如何转换才能得到一个等价的,但执行效率更高的 X-代数表达式?

5.2.3　XML 查询优化

由查询分解得到的查询树包含很多 X-代数运算,这些运算执行的先后顺序、在什么时候使用索引、使用什么样的索引、整个表达式的计算方法采用流水线方法还是实体化方法,都是影响查询效率的关键因素。不同执行顺序好坏的评价标准是它们的执行代价估计,这里的代价是指磁盘 I/O。在 XML 查询优化中,应该重点处理什么问题? 是不是主要针对复杂路径的代价估计? 有没有类似关系数据库的启发式优化规则? 代价估计的具体方法是什么? 是不是基于直方图的方法是最好的[9,18]? 这些都是需要重新认识和研究的问题。

5.2.4　实现 X-代数运算的各种高效算法

关系中的连接运算的算法有嵌套循环连接、索引嵌套循环连接、哈希连接和排序—归并连接等。那么 X-代数中的连接运算有哪些高效的算法? 结构连接(Structure Join)还有没有提高的空间? 诸如此类的问题都是 Native XML 数据库研究人员目前的研究重点。

5.3　XML 数据更新: XUpdate

对于 XML 数据的更新操作,无论在语言还是在操作方法上目前都没有一个统一的标准。更新操作从逻辑上是指元素的插入、删除和更新,是不是还包括其他内容,这正是需要仔细研究的问题。更新包括模式检查、节点定位、存储空间的分配和其他辅助数据的更新,如索引、编码等。

关于 XML 数据的更新语言,W3C 目前还没有这方面的工作计划,XQuery 中也没有更新 XML 数据的描述。而民间团体 XML: DB Initiative(参阅 http://www.xmldb.org/)则给出了更新语言 XUpdate 的规范(参阅 XML: DB XUpdate Use Cases, Kimbro Staken, http://www.xmldatabases.org/projects/XUpdate-UseCases/),但这是不是能被 W3C 所接受,目前还是个未知数。因此,在这个方面还有很多问题需要认真研究,当然可以借鉴 XML: DB XUpdate 的成果,如可以参考 XML: DB XUpdate 提供的基本操作。

6　典型的 NXDBMS 简介

6.1　OrientX

OrientX[9,14,17~19]是中国人民大学最近几年开发的一个纯 XML 数据库管理系统。

6.1.1　物理存储

OrientX 存储系统建立在操作系统的文件系统上,申请若干定长的文件(如 8MB),这些文件属于特定的数据集,数据集用 SetID 来标志;在文件上划分逻辑物理块,物理块定长,是操作系统物理块的整数倍(如 8KB),物理块用 LpNo 来标志;给定一对(SetID, LpNo)能马上找到对应文件相应的偏移量。

存储管理以记录为单位,不同的存储粒度记录的物理含义不同,可能是一棵子树,也可能是一个节点。虽然 OrientX 采用了 DEB, CEB, DSB 和 CSB 四种不同的存储方法,但对上层查询提供的接口都是一样的,这样可以提高数据的独立性。一个 XML 文档包含若干条记录,多个满足同一个模式定义(DTD 或者 XML Schema) 的 XML 文档存放在一个数据集里。但模式信息对 OrientX 来说不是必需的,有的存储方法一定需要模式信息,有的存储方法不需要模式信息。

6.1.2 索引管理

OrientX 的 SUPEX 索引利用了在实际应用中经常存在的 XML 数据的模式信息——DTD。从 DTD 出发构建索引结构,总结了符合 DTD 的 XML 数据可能出现的路径结构,其结构包括三个部分: 结构图、元素映射表和索引记录。结构图是路径索引的主体,它是从 DTD 中构造出来的,描述了符合 DTD 的文档中可能出现的路径。元素映射表是为了便于找到结构图中用同名元素名标记的节点而建立的入口索引。而索引记录是若干个元素记录的集合。

通过结合 XML 数据的编码,该索引可以支持多种查询方式,包括绝对路径查询、父子关系和祖先—后代关系的基本结构关系、相对路径查询等,从而能够有效地支持路径表达式的计算。XML 数据编码的引入使得查询结果可以进一步用于路径表达式的计算。

6.1.3 查询处理

OrientX 将由 XQuery 表达的查询转换为由 XML 代数运算构成的操作树,采用基于代价估计的查询优化策略,具体的代价估计方法是基于直方图的方法。数据更新是与数据查询相结合的,没有专门的数据更新语言。

6.2 Tamino

Tamino[10, 12] 是 Software AG 于 1999 年发布的 Native XML 服务器的第一个版本,其中包括 Native XML 数据库。Tamino 可以处理面向数据的文档(有正规的结构,元素顺序无关紧要,不会出现混合内容) 和面向文档的文档(几乎没有正规的结构,元素顺序非常重要,会出现混合内容) ,也可以存储其他类型的数据,如图像、HTML 文件等。Tamino 是一个完整的数据库管理系统,支持事务,提供安全保证和多用户访问,可扩展性好。Tamino 可以运行在 Windows 和 UNIX 操作系统上。

6.2.1 XML 存储

XML 文档可以有模式信息,如 DTD 或 W3C 的 XML Schema,但它们对 Tamino 来说都不是必需的。DTD 缺乏数据类型的信息,而它们是正常索引所必需的信息,因此 DTD 作为 XML 数据库模式是不够格的。2001 年,W3C 发布了 XML Schema 的推荐标准,它是一个模式定义语言,覆盖了 DTD 的所有功能并且扩展了很多新的概念。它与 DTD 的最大区别就是增加了它苦心经营的类型系统,使得 XML Schema 比较适合描述数据库模式。与关系数据库中模式与数据是紧耦合的不一样,Tamino 中模式与数据是松耦合的,而且有可能模式只描述文档的一部分。

Tamino 数据库由很多所谓的数据集组成,数据集仅仅是一组文档的包装器,每个文档只能存储在一个集里。每个集均由相关的 W3C 的 XML Schema 描述,在模式里,Doctype 是个特殊的概念,它标志了在 XML Schema 中作为根元素声明的所有元素中的一个。在集里,每个文档都只能作为一个 Doctype 的成员被存储。

文档的根元素标志了 Doctype,因此,在一个集里 Doctype 和根元素类型是 1∶1 的关系。如果文档存储在一个集里,而没有 Doctype 和文档的根元素对应,这时就会动态地创建一个 Doctype。在这种情况下,就没有相关的用户定义的模式。万一集里有一个相关的用户定义的模式存在,Tamino 就对照这个模式验证输入文档。Tamino 的元数据也以 XML 文档的形式存储在系统集里的系统 Doctypes 中。

6.2.2 索引

在数据库系统中,索引是绝对不可缺少的,否则就无法以满意的方式查询大量的数据。Tamino 支持三种类型的索引。

(1) 标准索引是基于值的索引,用于快速查找具有特定值的元素或属性。

(2) 文本索引是进行高效文本检索的先决条件。在文本索引中,包含在元素或属性中的单词被索引,这样可以加速对该词的搜索。

(3) 结构索引是 XML 特有的索引类型,俗称路径索引。精简的结构索引保存了在特定 Doctype 的任何实例中可能出现的所有路径信息。如果没有这样的索引,拼写错误就可能导致对一个 Doctype 的所有文档的顺序扫描。全结构索引不仅记录了一个 Doctype 中路径的存在,而且记录了路径所出现的文档。

6.2.3 磁盘的物理组织

Tamino 数据库由两部分空间组成: ①数据空间,包含所有的文档和对象,以簇集的方式组织 Doctypes,以便加速顺序扫描。基于文档的大小,Tamino 选择了相应的压缩技术; ②索引空间,包含索引数据。

上述两个空间都可以分布在多个不同的外部存储卷上,以便 Tamino 可以存储 TB 级的数据; Tamino 还可以根据需要自动扩展存储空间。对于事务日志,Tamino 有一个日志空间,它是一个固定大小的容器,可以循环保存所有必要的日志,以便回滚事务或在系统崩溃后重做事务。

6.2.4 查询 XML

支持 XPath/XQuery,且在两个方面对 XPath 进行了扩展。

6.3 Timber

Timber[13] 是密歇根大学于 1999 年开始开发的一个 NXDBMS,系统基于操作树的块(Bulk) 代数,主要贡献在于一次一集合的查询处理能力。这个系统参照标准 RDBMS 的设计与实现,RDBMS 的很多组件都不需要改动就可以用于 NXDBMS,如事务管理器。利用块代数可以操作树集合,代数中的每一个运算都以一个或多个树集合作为输入,而产生一个新的树集合作为输出。

6.3.1 数据存储

Timber 基于非常流行的后端存储系统 Shore[8], Shore 负责磁盘存储器管理、缓冲和并发控制。Timber 中的节点是广义节点,将所有的属性集成为一个子节点,将元素的文本内容单独作为一个子节点。文档树中的节点采用区域编码规则进行前

序编码, 但这种编码模式的更新永远是一个问题。XML 元素的物理存储顺序也是影响数据访问代价的重要因素。Timber 将与元素相关的子元素簇集在一起, 即按照文档的顺序(或者是深度优先的前序遍历顺序) 簇集存储节点, 实际上就是 DEB 的存储方法。

6.3.2　索引存储

Timber 目前只实现了单节点索引, 索引可以在属性值上或者元素内容上(当元素内容可以识别为一个数时) 构造。当然, 当元素内容为大文本时, 也可以构造基于关键字的倒排索引, 还可以在标记名上建立索引(到相应元素的映射) 。所有的索引都采用 B-树结构。在 RDBMS 中索引返回一系列 RID, 而在 NXDBMS 中索引返回一系列(D, S Œ E, L) 节点标志。

6.3.3　元数据存储

在 XML 数据库管理系统中, 很多时候, 元数据不是很显眼。元数据信息包括属性类型、数据集大小和索引结构等, 而用于代价估计目的的直方图则是新的元数据。模式在传统的数据库中扮演着一个非常关键的角色。但是 Timber 的核心设计对模式没有任何依赖性, 虽然模式对数据布局、索引选择和查询优化有非常重要的影响。

6.3.4　查询处理

支持 XQuery 查询语言, 查询分析器将 XQuery 转换成代数操作树, Timber 使用的是树代数。Timber 的查询优化器识别这棵树, 基于一系列规则和元数据信息, 并且执行将逻辑操作映射为物理操作的任务。查询计算器执行结果的查询计划树, 像管道一样一次执行一个操作。

Timber 采用基于子树的查询代数, 在查询处理中吸收了流水线计算方法的思想, 利用堆栈方法处理结构连接。Timber 支持基于代价估计的优化, 但是其代价估计只是基于位置直方图, 在估计带谓词的表达式的代价时有局限性。

此外, 还有像 eXist[10], Natix[5], Lore[20] 和嵌入式 XML 数据库管理系统 Berkeley DB XML(参阅 http: / / www. sleepycat. com/ products/ xml. shtml) 等类似的系统, 本文就不一一介绍。

7　结论

XML 数据库的研究已经从基于传统的 RDBMS 转向 NXDBMS, 这是 XML 数据库研究的发展趋势。由此也带来了许多挑战性的问题, 如 XML 的节点编码、X-代数、查询优化、实现 X-代数的各种高效算法、支持查询优化的有效索引、XML 的数据更新。这些问题给数据库研究人员提供了广阔的平台, 也为我国在数据库研究方面赶超世界先进水平提供了机遇。

参考文献:

[1] onald Bourret. XML and Databases[EB/ OL] . http: / / www. rpbourret. com/ xml/ XMLAndDatabases. htm, 2003-11.

[2] Feng Tian, David J DeWitt, *et al*. The Design and Performance Evaluation of Alternative XML Storage Strategies[J] . ACM SIGMOD Record, 2002, 31(1) : 5-10.

[3] S Abiteboul, S Cluet, *et al*. Querying and Updating the File[C] . Proc. of VLDB1993, San Francisco: Morgan Kaufmann Publishers Inc., 1993. 73-84.

[4] D Florescu, D Kossman. Storing and Querying XML Data Using an RDBMS[C] . Proc. of ACM SIGMOD2002, New York: ACM Press, 2002. 204-215.

[5] C Kanne, G Moerkotte. Efficient Storage of XML Data[C] . Proc. of ICDE2000, Washington, DC: IEEE Computer Society, 2000.

[6] J Shanmugasundaram, K Tufte, C Zhang, *et al*. Relational Databases for Querying XML Documents: Limitations and Opportunities[C] . Proc. of VLDB1999, San Francisco: Morgan Kaufmann Publishers Inc., 1999. 302-314.

[7] M Carey, D DeWitt, *et al*. The BUCKY Object-Relational Benchmark[C] . Proc. of ACM SIGMOD1997, New York: ACM Press, 1997. 12-21.

[8] M Carey, D DeWitt, *et al*. Shoring up Persistent Applications[C] . Proc. of ACM SIGMOD1994, New York: ACM Press, 1994. 383-394.

[9] 孟小峰, 王宇, 等. OrientX: 一个 Native XML 数据库系统的实现策略[J] . 计算机科学, 2003, 30(增刊) : 111-115.

[10] Tamino-Number One in XML Management[EB/ OL] . http: / / www1. softwareag. com/ Corporate/ products/ tamino/ default. asp.

[11] Arnaud Sahuguet. Kweelt: More Than Just " Yet Another Framework to Query XML! " [C] . Proc. of ACM SIGMOD2001, New York: ACM Press, 2001. 602.

[12] Harald Schoning. Tamino: A DBMS Designed for XML[C] . Proc. of ICDE2001, Washington, DC: IEEE Computer Society, 2001. 149-154.

[13] H V Jagadish, Shurug Al-Khalifa, *et al*. Timber: A Native XML Database[J] . The International Journal on Very Large Data Bases, 2002, 11(4) : 274-291.

[14] 罗道峰, 孟小峰, 等. OrientStore: Native XML 存储方法[J] . 计算机科学, 2003, 30(增刊) : 105-110.

[15] Goldman R, McHugh J, Widom J. From Semistructured Data to XML: Migrating the Lore Model and Query Language [J] . Markup Languages: Theory and Practice, 2000, 2(2) : 153-163.

[16] McHugh J, Widom J. Query Optimization for XML[C] . Proc. of VLDB1999, San Francisco: Morgan Kaufmann Publishers Inc., 1999. 315-326.

[17] 王静, 孟小峰, 等. 以目标节点为导向的 XML 路径查询处理[J] . 软件学报, 2005, 16(5) : 827-837.

[18] 王宇, 孟小峰, 等. OrientX 中的统计信息收集方法研究[J] . 计算机科学, 2003, 30(10) (增刊) : 94-98.

[19] 罗道峰, 孟小峰, 等. XML 数据扩展前序编码的更新方法[J] . 软件学报, 2005, 16(5) : 810-818.

[20] McHugh J, *et al*. Lore: A Database Management System for Semistructured Data[J] . ACM SIGMOD Record, 1997, 26(2) : 54-66.

作者简介:

冯建华(1967-) , 男, 山西运城人, 副教授, 博士, 主要研究方向为数据库、XML 数据库和 WWW 环境下的信息处理; 钱乾(1981-) , 男, 硕士, 主要研究方向为 XML 数据库的存储与索引; 廖雨果(1981-) , 女, 硕士, 主要研究方向为 XML 数据库的查询处理与优化; 李国良(1981-) , 男, 博士, 主要研究方向为用于 XML 数据库的 X-代数与编码系统; 塔娜(1982-) , 女, 硕士, 主要研究方向为 XML 数据库和 XML 数据库评测; 周立柱(1947-) , 男, 教授, 博士生导师, 主要研究方向为数据库、分布式数据库和海量存储系统。