

## OrientStore+: 一种支持高效更新的 Native XML 存储方法

张新<sup>1</sup> 孟小峰<sup>1</sup> 朱金清<sup>1</sup> 王伟<sup>1</sup> 黄静<sup>1</sup>

<sup>1</sup>(中国人民大学信息学院, 北京 100872)  
([xinzhang@ruc.edu.cn](mailto:xinzhang@ruc.edu.cn))

**摘要** XML数据在数据库中的存储模式对XML数据的查询、索引及更新有重要的影响,而目前许多XML存储方法在更新上都需较高的代价.本文提出一种Native XML存储方法OrientStore+,可以完全保留XML树结构信息,同时还具有如下特点:(1)易于对XML数据建立各种索引;(2)存储记录间相互独立,进行更新时,可以减少对XML存储及索引的修改,减小了更新的代价;(3)在Native XML数据库系统OrientX中实现了这种存储模式.另外,在这种存储模式基础上提出一种基于空间利用率的XML存储更新算法.并通过实验比较了在不同存储方法上的查询与更新效率.

**关键词** XML数据库, XML存储, XML更新

中图法分类号 TP391

## OrientStore+: a native XML storage strategy for efficient update

Zhang Xin<sup>1</sup>, Meng Xiaofeng<sup>1</sup>, Zhu Jinqing<sup>1</sup>, Wang Wei<sup>1</sup>, Huang Jing<sup>1</sup>

<sup>1</sup>(School of Information, Renmin University of China, Beijing 100872)

**Abstract** The storage strategy for XML data in database will affect the efficiency of querying, indexing and updating significantly. The cost is high for most of state-of-the-art XML storage strategies when considering update. In this paper, we propose an optimized native XML storage strategy, called OrientStore+, which can keep the structure relationship between nodes completely and the new storage can also satisfy the following virtues: (1) can facilitate building index on xml data; (2) the records in the storage are independent, which can reduce the modification to the storage data and index after update, that we can reduce the update cost; (3) the storage strategy is implemented in native XML database system OrientX. In our paper, a new update algorithm which is based on the space usage ratio is proposed. Finally, we compare the efficiency of query and update using different storage strategies through extensive experiments.

**Keywords** XML database, XML storage, XML update

目前已有许多XML数据库系统<sup>[1~4]</sup>来管理XML数据,根据管理XML数据方式,将这些数据库分为两类:Native XML数据库<sup>[1,2]</sup>和非Native XML数据库<sup>[5]</sup>,前者对XML数据的管理是建立在XML数据模型<sup>[6]</sup>上,而后者是将XML数据模型映射到其它数据模型上,如关系模型<sup>[4]</sup>.由于XML数据的半结构化特点,映射到其他数据模型往往会使问题变得复杂.因此,一些传统关系数据库厂商开始在关系数据库上采用Native方式管理XML数据<sup>[3]</sup>.本文讨论的工作也是如何在Native方式下有效地实现XML的存储与更新.

XML数据模型是把XML文档看成一棵由不同

结点组成的树,称之为文档树.为了能随机访问或者修改XML树中的结点,Native XML数据库需要将XML文档树“打散”成更小的单位进行存储.目前大部工作都是将XML文档树按结点<sup>[2]</sup>或者子树为单位进行存储<sup>[6,7,3]</sup>.XML数据的存储,不仅要存储XML结点的数据,还必须保存结点间的结构关系.因此,当具有结构关系(主要是父子关系)的结点不在同一个存储记录内时,记录之间往往要保存一定数量的“结构信息”以保证数据还原的正确性.通常,当一个结点的父亲或孩子结点不在同一记录内时,记录须保存它们的记录地址.这就使得记录之间不

收稿日期:

基金项目: \*本文得到国家自然科学基金项目(课题号: 60573091),北京市自然科学基金(课题号: 4073035),教育部新世纪优秀人才支持计划的资助



下文均以[6,7,3]中都采用的按照物理块大小来划分子树的方法来介绍OrientStore+存储方法.实际上,单个结点可以看成特殊子树,以结点为单位进行存储也可以当成按子树存储来处理.

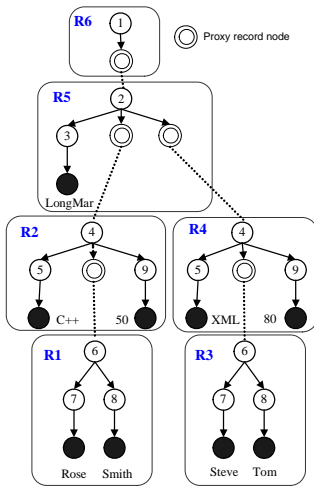


图 3. 记录树

根据物理块大小划分子树的算法在[6]介绍过, OrientStore 中,保存结点间的关系通过保存相应的记录地址,每读一个记录,可以立即得到不在同一记录内的父亲结点与孩子结点的记录地址(如果它们存在),具有很好的遍历效率.但也带来更新的代价,如图 3 中 R2 的地址发生了改变,则必须修 R1 和 R5,若还对 R2 建了索引, 还需修改其索引.

在 XML 存储时,每个结点还分配了一个编码,编码与结点的关系是一一对应的关系;在 OrientStore+ 中为结点编码与其存储地址建立映射关系 F.若一个结点的编码为 c,则 F(c)为其物理地址.易知 F(c) 的值在系统中是唯一的.

这样,在记录中不再需要存储其子记录与父亲记录的地址,只需要保存相应结点的编码 c,通过 F(c) 得到相应的存储地址.而结点的编码通常是不改变的.

建立映射关系 F 的一种方法是建立结点编码 (Code)到结点记录地址(Record-address)的索引,称之为 Code-Record index.图 4 是图 3 记录树的 B+树 Code-Record index.对于以子树为单位的 XML 存储,不需要建立每一个结点编码到记录地址的索引,只需要对每个记录的根结点建立索引即可.通过结点编码找其地址时,在 Code-Record Index 找到其最近的祖先结点编码对应的记录地址即可.如找编码 <33,37>的地址,找<29,49>的记录地址即可.

Proxy Record node 结点未分配编码,我们可以取这个记录内的结点编码中最小的 start,最大的 end 值组成的编码作为它的编码.

在引入 Code-Record index 后,若一个记录的地址发生了改变,只需要对 Code-Record index 做修改,无需修改其它记录.建立索引只需要索引结点的编码,无需索引结点的记录地址.结点的记录地址发生了改变,不需要对索引进行修改,减小了索引的维护代价.

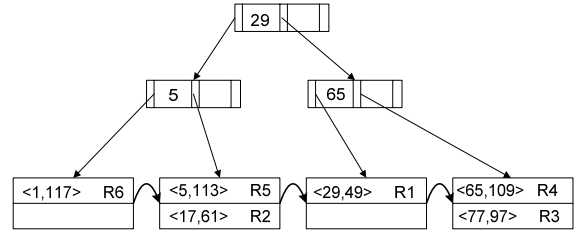


图 4 Code-Record Index

### 3 OrientStore+中的存储更新

对 XML 存储影响最大的更新操作是插入操作,包括插入结点或者子树,为了描述简单,均指插入结点.插入结点操作可以用一个三元组来描述: (S,parent,n),S 是待插入的结点,作为结点 parent 的第 n 个孩子插入.S 插入的逻辑位置是固定的,但对于以子树为单位的存储方法,在物理位置上却可能会有多种选择.如对图 5 执行(S,a,3)更新操作,即将 S 作为 a 的第 3 个孩子插入.新插入结点 S 可以选择 R1,R2,R3 中任意一个记录存储,都能保证结点间结构关系正确性,图 5 用虚线箭头表示的是 S 可选的插入位置.

插入一个新的结点时,首先必须确定插入的物理位置.Natix<sup>[7]</sup>引入split Matrix来确定结点间聚集存储的权值,在插入结点时选择与聚集存储权值高的结点聚集存储.但在实际系统中,很难自动判断那些结点要聚集存储.为此,OrientStore+引入了一种基于存储空间利用率的更新算法.

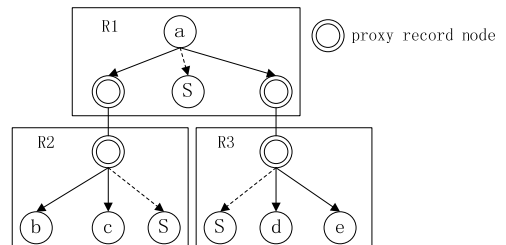


图 5. 插入 S 在物理位置的可能情况

插入新的结点时,保证文档结构的同时应使生成的记录尽可能地少(即分裂子树数目),这样既可以提高空间利用率也可以减少读取数据的 I/O.因此,插入新结点时应优先插入已存在的记录中. OrientStore+ 更新算法主要分为两步,第一步:找到

结点可插入的目标记录,第二步,将结点插入记录中,如果插入结点后记录大于记录上限值 RecordMaxSize,则进行子树分裂.

### 算法1. 插入一个新结点

```

Insert(a,S,childIndex)函数
//将结点S作为a结点的第childIndex个孩子结点插入
{
    //返回S插入位置的左兄弟,若不存在,则返回父亲结点
    ① left = GetLeft(childIndex);
    //返回S插入位置的右兄弟,若不存在,则返回父亲结点
    ② right = GetRight(childIndex);
    ③ if( left.recordID == right.recordID )
    ④     InsertToRecord(S, left.recordID);
    ⑤ else{
    ⑥     small = findSmallestEnoughSpace(left, right, a,S.size);
    ⑦     if( small != NIL )
    ⑧         InsertToRecord(S, small.recordID);
    ⑨     else
    ⑩         InsertToRecord( S, a.recordID);
    }
}
InsertToRecord(S, recordID)函数
//将结点S插入recordID记录中
{
    if(recordID.size + S.size <= RecordMaxSize ){
        //直接将S写入recordID记录中
    }
    else {
        //先将S写入recordID记录的子树中
        makeRecord(recordID.root,);
    }
}

```

图 6. OrientStore+存储中插入新结点算法

图 6 是插入一个新结点的更新算法的伪代码.限于篇幅,没有给出函数 findSmallestEnoughSpace() 及 makeRecord()的实现, findSmallestEnoughSpace() 从传入参数中的前三个(left,right,a)结点所在的物理页中,找出剩余空间不小于 S.size 中最小的一个,如果不存在,则返回空. makeRecord()函数传入参数为一子树的根结点,若这棵子树的大小大于一个记录的上限(RecordMaxSize), 将调用[6]划分子树的算法,将子树划分成更小的子树进行存储.

算法 1,第 1~3 行是判断结点可插入的记录是否有多种选择.第 1,2 行是返回 a 结点的第 childIndex 个孩子结点的左(右)兄弟结点,若存在,则返回左(右)兄弟,不存在,返回其父亲结点.如果 left,right 结点在同一记录内,则 S 结点可插入的记录只有一种选择.第 5-10 行,说明 left,right 不在同一记录内,则 S 可以选择插入 left, right,以及父亲结点所在的记录中,首先利用 findSmallestEnoughSpace() 函数找出在 left,right 与 a 三个结点所在的物理页中,剩余空间能容纳 S 且剩余空间最小的一个.若不存在满足条件的记录,则将 S 插入父亲结点中(算法第 10 行),因为孩子结点应尽可能地与父亲结点聚集存储,以提高

查询的速度.

例如:对图 5 执行 (S,a,3)更新操作, left=c, right=d,但 left 与 right 不在同一记录内,将调用算法第 6 行,从 left,right 以及父亲结点 a 所在的物理页找出剩余空间不小于 S.size 且最小的一个插入.若执行 (S,a,2),此时, left =b, right =c,S 只能插入 R2 中.

## 5 实验

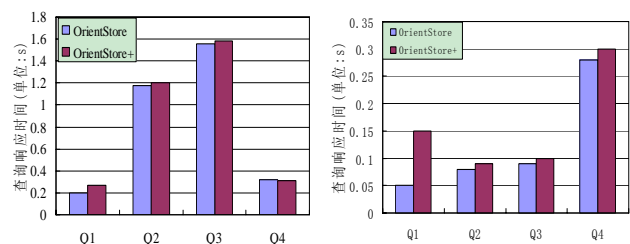
实验在处理器为 P4 2.9G, 内存 512M, Windows XP Professional 平台上运行.实验比较 OrientX<sup>[1]</sup>基于 OrientStore+ 与 OrientStore 的查询效率与更新效率.查询效率主要比较 OrientX 在两种存储上执行同一查询的响应时间.而更新效率则主要比较在两种存储上执行同一更新操作所需修改的物理页数,修改的物理页数反映了更新代价.实验对 xmark,DBLP 数据集中不同大小的 XML 文档进行了测试,测试结果反映出相似的特征,限于篇幅,以下给出的测试结果均是在 xmark 数据集的 auctions.xml 文件上测试得到的,文档大小为 23M.

Q1	for \$b in doc("auctions.xml")/site/people/person[@id = "person100"] return <name>{\$b/name/text()}</name>
Q2	for \$i in doc("auctions.xml")/item[@id="item10"] return \$i/name
Q3	for \$i in doc("auctions.xml")/item let \$q := \$i/quantity where \$q <= 5 or \$q >= 30 return \$i/name
Q4	for \$p in doc("xmark@auctions.xml")/site/*/person [profile/@income > 2000] return <person id="{ \$p/@id }">{\$p/name}</person>

表 1 测试查询举例

### 5.1 查询效率

执行查询时,有无可用的索引对查询速度有很大的影响,OrientStore+引入 Code-Record index 后,对 XML 建立索引,无需索引结点的记录地址,而是索引结点的编码,再通过 Code-Record index 找到结点的记录地址.实验分别对比了在两种存储上执行查询时有索引与无索引两种情况下的查询效率.



(a) 无索引查询响应时间 (b) 有索引时查询响应时间  
图 7 查询响应时间

图 7 执行表 1 查询的响应时间,除 Q1 外,其它查询在 OrientStore+ 与 OrientStore 上的响应时间相

差甚微,因为执行第一个查询 Q1 时,需构建文档的 Code-Record index. 建立之后,将其缓存在内存中(直到内存区满),执行其它查询时不用再生成之,只需花费在 Code-Record index 上的查找时间,而相对从磁盘上读取数据的 I/O 操作,在 Code-Record index 上的查找时间可以忽略不计. 调换 Q1~Q4 的执行顺序,发现同样的特征. 因此,在引入 Code-Record index 后,对查询影响主要是 Code-Record index 的构建时间.

## 5.2 更新代价

更新操作主要考虑插入新的结点后,需要修改的物理页数. 对存储于 OrientStore+ 与 OrientStore 的 auctions.xml 分别执行四次更新操作:根据结点编码从文档中随机选取 10,100,250,500 个结点,对每个选中的结点插入 10 个孩子结点,插入位置随机,插入结点的格式为 <a>Value</a>, a 为结点名,插入数据库时,它会被一个 TagID 代替. Value 为长度为 k 的文本,k 为 1~K 之间的随机数,实验中选取 K=100.

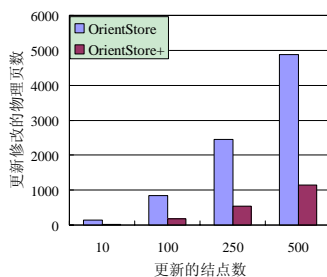


图 8. 更新时改的物理页数

图 8 是 auctions.xml 文件在 OrientStore 与 OrientStore+ 中执行上述更新所需修改的物理页数. 数据显示,执行同一更新, OrientStore 需要修改的物理页数是 OrientStore+ 的 5 倍左右. OrientStore+ 通过引入 Code-Record index 降低记录间的关联性,可以有效降低更新代价.

实验表明:在 OrientStore+ 中引入了 Code-Record index,虽然对查询效率有些影响,一般只是对查询需要构建 Code-Record index 时有影响,而且根据 B+ 树的特征,并不需要在内存中构建整个 Code-Record index,可以通过缓冲区技术减少 Code-Record index 的构建次数,它对查询的影响可限制在合理范围内.但 Code-Record index 提高了记录间独立性,大大降低了 XML 更新代价.尤其是对需要频繁更新 XML 数据.

## 6 结论和展望

OrientStore+ 通过为 XML 结点的编码与其存储地址建立映射关系,可以提高记录间的独立性,大大减小了 XML 存储及索引的更新代价,对查询效率的影响也在合理范围内. OrientStore+ 存储方法与

系统所用的编码方法无关,无论用什么编码都可以建立结点编码与其存储地址的 Code-Record index. 这种方法同样适用于其它 Native XML 数据库系统.

当然,在 XML 存储上还有许多后续工作要做,例如,如何有效划分子树以减少查询的 I/O,编码更新问题等,后继工作将关注这些问题.

## 参考文献

- [1] 孟小峰,王宇等. OrientX: 一个 Native XML 数据库系统的实现策略,第 20 届全国数据库学术会议, 2003,10, 计算机科学, 卷 30(10), 111-115.
- [2] H. V. Jagadish, Shurug AL-Khalifa, et al. TIMBER: A Native XML Database. Technical Report, University of Michigan, April 2002.
- [3] Kevin Beyer, Roberta J. Cochrane, Vanja Josifovski, et al. System RX: One Part Relational, One Part XML. In SIGMOD Conference, pages 347-358, 2005.
- [4] D. Florescu and D. Kossmann, "Storing and Querying XML Data Using an RDBMS", Data Eng. Bulletin, 22(3), 1999
- [5] Mary Fernández et al. XQuery 1.0 and XPath 2.0 Data Model : <http://www.w3.org/TR/xpath-datamodel/>. 2007-4.
- [6] 罗道峰,孟小峰等. OrientStore: Native XML 存储方法. 第 20 届全国数据库学术会议, 2003,10, 计算机科学, 卷 30(10), 105-110.
- [7] C.-C. Kanne and G. Moerkotte. Efficient Storage of XML data. In ICDE Conference, page 198-209. 2000.
- [8] 罗道峰, 孟小峰等. XML 数据扩展前序编码的更新方法. 第 20 届全国数据库学术会议, 2003,1, 计算机科学. 卷 30(10) 99-104.

**张新**, 男, 1983 年生, 硕士研究生, 研究方向: xml 数据库。

**孟小峰**, 男, 1964 年生, 教授(博导), 研究领域: Web 数据管理, XML 数据库, 移动数据管理. 曾先后在香港中文大学、香港城市大学、新加坡国立大学访问研究. 主持或参加过十多项国家科技攻关项目、国家自然科学基金以及国家 863 项目。

**朱金清**, 男, 1984 年生, 硕士研究生, 研究方向: xml 数据库。

**王伟**, 男, 1983 年生, 硕士研究生, 研究方向: xml 数据库

**黄静**, 女, 1984 年生, 硕士研究生, 研究方向: xml 数据库