DM XML 加速线性 Twig 查询的实现

徐小双[1,2],冯玉才[1,2],周英飚[1,2]
XU Xiao-shuang[1,2],FENG Yu-cai[1,2],ZHOU Ying-biao[1,2]

1.华中科技大学 计算机学院 多媒体数据库研究所,武汉 430074
2.达梦数据库有限公司,武汉 430074
1.Institute of Multimedia Database,School of Computer,Huazhong University of Science and Technology,Wuhan 430074,China
2.Dameng Database Corporation Limited,Wuhan 430074,China
E-mail:xxsh99@hgnu.edu.cn

XU Xiao-shuang,FENG Yu-cai,ZHOU Ying-biao.Implementation of accelerating linear Twig queries in DM XML. Computer Engineering and Applications,2009,45(16):9-12.

Abstract: Since every complex Twig query consists of linear ones,it is important to handle linear Twig queries effectively.DM XML system builds on homemade DM5.6,integrates structure mapping and model mapping,and realizes a particular path-partitioned encoding scheme to store XML data.After a linear Twig is analyzed in the system,its path set is generated.Each path in the set is uniquely translated into a range query for integer primary key on RDBMS.The results of experiments indicate path-partitioned encoding scheme speeds linear Twig queries so that it will make an effective way to implement complex twigs.
Key words: Extensible Markup Language(XML);tree pattern;structural join;twig query;encoding scheme

摘 要:每一个复杂的 Twig 查询都由线性 Twig 查询构成,有效地处理线性 Twig 查询显得非常重要。DM XML 系统以国产 DM5.6 关系数据库为平台,融合结构映射和模型映射,实现独特的路径分区编码方案来存储 XML 数据。在系统中,线性 Twig 查询解析后,形成线性 Twig 查询的路径集,而该集合中的每一个路径可被唯一变换为关系数据库中整型主键的范围查询。实验结果显示,路径分区编码方案能加速线性 Twig 查询,它将为高效实现复杂 Twig 查询奠定基础。
关键词:可扩展标记语言;树模式;结构连接;小枝查询;编码方案
DOI:10.3778/j.issn.1002-8331.2009.16.003 **文章编号:**1002-8331(2009)16-0009-04 **文献标识码:**A **中图分类号:**TP311

1 引言

有效地存储和查询 XML 数据是近几年来持久研究的热点。XML 数据存储方式主要有文件系统方式、关系数据库存储方式、面向对象数据库存储方式和 Native XML 数据库存储方式。其中,在第二种方式中,关系数据库把 XML 文档映射成关系模式后进行存储;目前,映射方案有主要两类:结构映射和模型映射。结构映射方案从 XML 文档依赖的 DTD 或 Schema 文档中按一定规则生成复杂的关系表[1],具有路径明晰的特点;模型映射方案侧重于对 XML 文档直接分析,具有编码简洁的特点[2]。相对而言,基于结构映射存储方案的路径查询效率较高。显然,如果能把结构映射和模型映射同时体现在关系模式中,有机结合两类映射的优点,必将有益于 XML 数据存储和查询。然而,目前还没有发现相似研究的报道。

本文以国产达梦数据库为研究平台,实现了 DM5.6 系统对 XML 的支持。原型系统 DM XML 采用独特的路径分区编码方案,体现了结构映射和模型映射的有机结合,具有编码简洁和路径明晰的特点。DM XML 高效地支持线性 Twig 查询,合理解决了线性 Twig 查询中出现的后裔边"//"和通配符"*"。它将为由多条线性分支构成的复杂 Twig 查询最大限度地减少结构连接操作,提高整体查询效率奠定基础。

2 DM XML 的路径分区编码

在实际应用中,XML 文档虽然元素数量庞大和结构深度嵌套,但文档中包含许多重复的线性路径。线性路径的数量总是远远小于节点的数量,而且在文档中必定是有限的。这里通过统计 XML 文档中出现的线性路径,建立了基于路径分区的 DM XML 存储编码方案。DM XML 将一份 XML 文档存储在 RDBMS 前,使用常量 CP 限定任一线性路径下允许出现的最大

基金项目: 国家信息产业部科技攻关课题(the Key Technologies R&D Program of Ministry of Information Industry of China under Grant No.2005BA112A02-DB-DM);湖北省发改委基金(the Research Foundation of Hubei Development and Reform Commission under Grant No.[2007]1334)。
作者简介:徐小双(1970-),男,博士生,副教授,系统分析员,主要研究领域为半结构化数据,数据库系统实现技术;冯玉才(1944-),男,教授,博士生导师,主要研究领域为现代数据库技术,数据仓库,Web 数据集成,数据挖掘;周英飚(1970-),男,博士,副教授,主要研究领域为半结构化数据,并行计算,数据库系统实现技术。
收稿日期:2009-02-03 **修回日期:**2009-03-06

元素数,建立包含 PT 表和 ET 表的关系模式。

2.1 线性路径统计

在读入 XML 文档阶段，采用 Libxml2 作为 XML 解析器，XML 解析器先序遍历 XML 文档，把依次出现在 XML 文档中的不同线性路径写入 PT 表。PT 表主要组成如下：

PT(*PID*,*Tag*,*Len*,*Path*,*MaxPrefix*)。

其中,字段 *Path* 由路径标签和间隔符号"/"组成,表征路径信息,字段 *Tag* 记录该路径的最后一个标签名,字段 *Len* 记录路径长度。字段 *PID* 标示一个唯一的线性路径，为主键。字段 *MaxPrefix* 给出了由该路径的最大前缀形成的线性路径的 *PID* 值。

图 1 给出了一份 XML 文档对应的标签树,图 2 列出了 PT 表的内容,其中 *CP*=100。从中可以看出,PT 表实质上统计了出现在 XML 文档中的不同路径。例如,设 *PID* 为 5 的元组为 p，设 *PID* 为 7 的元组为 q,则 $p.Path$=A/B/D/D,$q.Path$=A/B/D/D/D,而 A/B/D/D是A/B/D/D/D的最大前缀线性路径，故 $q.MaxPrefix$=5。同理,也可得到 $p.MaxPrefix$=3。当 PT 表较大时,在字段 *Tag*、*Len* 建立复合索引,以加快路径匹配的查找速度。

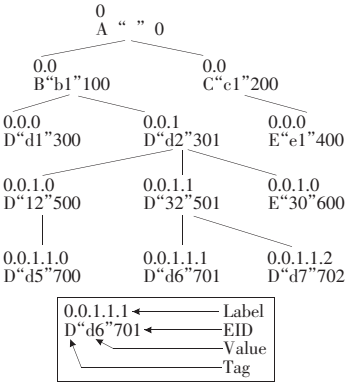

图 1 一棵 XML 文档树

PID	Tag	Len	Path	MaxPrefix
0	A	1	A	−1
1	B	2	A/B	0
2	C	2	A/C	0
3	D	3	A/B/D	1
4	E	3	A/C/E	2
5	D	4	A/B/D/D	3
6	E	4	A/B/D/E	3
7	D	5	A/B/D/D/D	5

PT 表

EID	Label	Value
0	0	null
100	0.0	b1
200	0.0	c1
300	0.0.0	d1
301	0.0.1	d2
400	0.0.0	e1
500	0.0.1.0	12
501	0.0.1.1	32
600	0.0.1.0	30
700	0.0.1.0.0	d5
701	0.0.1.1.1	d6
702	0.0.1.1.2	d7

ET 表

图 2 PT 表和 ET 表(*CP*=100)

2.2 XML 文档的编码

ET 表主要用来存储 XML 文档元素的信息,主要组成为:

ET(*EID*,*Label*,*Value*)

其中，字段 *EID* 标示 XML 文档中的唯一元素，为主键；字段 *Value* 记录了元素的值,字段 *Label* 能够推导出元素所在路径。

规则 1 若XML 文档中的元素 e 对应于 ET 表中元组 u，$u.EID$ 按如下方式编码：

(1)在 PT 表中存在一个元组 p,使得 $p.Path$ 就是从根元素到 e 的路径,则 $u.EID\in[p.PID*CP,(p.PID+1)*CP-1]$和 $\lfloor u.EID/CP\rfloor=p.PID$。

(2)若 XML 文档中的元素 e'对应于 ET 表中元组 u',根元素到 e 的路径与到 e'的路径相同，按深度遍历时 e'是 e 后继,则 $u'.EID>u.EID$,$u'.EID\in[p.PID*CP,(p.PID+1)*CP-1]$且 $\lfloor u'.EID/CP\rfloor=p.PID$。

在图 2 所示的 ET 关系表中,假设元组 u_1、u_2、u_3的 PID 字段分别为 700、701 和 702，依次对应图 1 所示的 XML 文档的三个元素。它们都有同样的路径 A/B/D/D/D,在 PT 关系表中标记为 7。因此,$u_1.EID,u_2.EID,u_3.EID\in[7*CP,(7+1)*CP-1]$=[700,799],其中 *CP*=100。根据性质 1,XML 文档的所有路径相同的元素按升序连续聚集在 ET 关系表中,因此把这种编码方案称为路径分区编码。很明显,ET 关系表能够很好地支持在字段 EID 上的索引。

规则 2 若 XML 文档中的元素 e 对应于 ET 表中元组 u，$u.Label$ 按如下方式编码：

(1)若元素 e 为根元素,$u.Label=u.EID \bmod CP$。

(2)若元素 e 为非根元素,且 e 的父亲对应 ET 表中元组 u',则 $u.Label=u'.Label+.x$,其中 $x=u.EID \bmod CP$。

在表 1 所示的 ET 关系表中，有根元素对应的元组 *Label* 字段为 0；假设元素 e 元组对应 ET 表中 *PID* 字段分别为 700 的元组 u。从 ET 表中,可以看到,元素 e 的父亲对应的元组 u'有 $u'.Label$=0.0.1.0，所以 $u.Label=u'.Label+.u.EID \bmod CP$=0.0.1.0.0。

2.3 路径分区编码的性质

定义 1 给定 XML 文档的关系模式(PT,ET)和常量 *CP*。设 p 为 PT 表中的元组,定义$\partial(p.Path)=p.PID$,$\partial^{-1}(p.PID)=p.Path$和 $\delta(p.PID)=p.MaxPrefix$。特别地,

$$\underbrace{\delta(\cdots\delta}_{i}(p.PID)\cdots)=\delta^{i}(p.PID)\text{和}\delta^{0}(p.PID)=p.PID$$

其中 $i\in N,0\leqslant i\leqslant|p.Path|-1$。

定义 2 给定 XML 文档的关系模式(PT,ET)和常量 *CP*,设 u 为 ET 表中的元组,必定存在 p 为 PT 表中的元组,使得 $\lfloor u.EID/CP\rfloor=\partial(p.Path)=p.PID$。令$|p.Path|=k$,$u.Label=n_1.n_2.\cdots.n_k$,其中 $n_1,n_2,\cdots,n_k,k\in N$,定义矢量 $\lambda(u.EID)=(n_1,n_2,\cdots,n_k)$,且 $\lambda_i(e.EID)$是 $\lambda(u.EID)$的第 i 个分量,即 $\lambda_i(e.EID)=n_i$,其中 $1\leqslant i\leqslant k,i\in N$。定义矢量$\rho(u.EID)=(\delta^{k-1}(p.PID)*CP+\lambda_1(u.EID),\delta^{k-2}(p.PID)*CP+\lambda_2(u.EID),\cdots,\delta^{0}(p.PID)*CP+\lambda_k(u.EID))$,且 $\rho_i(u.EID)=\delta^{k-i}(p.PID)*CP+\lambda_i(u.EID)$是 $\rho(u.EID)$的第 i 个分量,其中 $1\leqslant i\leqslant k,i\in N$。

定理 1 给定 XML 文档的关系模式(PT,ET)和常量 *CP*,设 u 为 ET 表中的元组,矢量$\rho(u.EID)$能顺序标示 u 所有祖先元组。

定理 1 容易从规则 1 和规则 2 推导出。例如,∂(A/B/D/D/D)=7,假设元素 e 元组对应 ET 表中 *PID* 字段分别为 702 的元组 u。$\lfloor u.EID/CP\rfloor=\partial$(A/B/D/D/D)=7,$\lambda(u.EID)$=(0,0,1,1,2),*CP*=100(见图 2)。根据 PT 表,有 $\delta^0(7)=7$,$\delta^1(7)=5$,$\delta^2(7)=\delta(\delta(7))=3$,$\delta^3(7)=\delta(\delta^2(7))=1$ 和 $\delta^4(7)=\delta(\delta^3(7))=0$。因此,$\rho(u.EID)$=(0*100+0,1*100+0,3*100+1,5*100+1,7*100+2)=(0,100,301,501,702)。可以看出,0、100、301、501 分别标示了 u 所有祖先元组。

推论 1 给定XML文档的关系模式(PT,ET)和常量CP,设u、u'为ET表中的元组,且u'是u的父亲元组,则$\rho_{k-1}(u.EID)=u'.EID$,其中,$k=|\rho(u.EID)|$。

推论 2 给定XML文档的关系模式(PT,ET)和常量CP,设u、u'为ET表中的元组,且u'是u的祖先元组,则$\rho_{k-i}(u.EID)=u'.EID$,其中,$k=|\rho(u.EID)|$,$0<i\leqslant k$,$i\in N$。

通过推论1和推论2,很容易判定来自ET表的两个元组是否存在PC(父亲/孩子)关系和AD(祖先/子孙)关系。通过上述分析,路径分区编码方案按线性路径将元组聚集存放,每个元组根据规则1(1)能求出相应路径,具有路径明晰的特点,体现了结构映射的特点。根据推论1和推论2,路径分区编码方案能明确表明两元组对应元素的PC关系和AD关系,体现了模型映射的特点。DM XML有机结合各类映射的优点,必将加快XML数据查询。

3 DM XML的线性Twig查询

Twig查询是搜索XML树得到满足树状结构的查询模式的结果。Twig查询可用XPath表达式描述,XPath表达式使用树模式(tree pattern)建模,树模式也被称作小枝模式(Twig pattern)。文中考虑最常用的三个轴:self轴(.),Child轴(/)和descendant轴(//)。按文献[3]对树模式分类,所有的树模式都属于$P^{\{/,*,//,[]\}}$类。根据缺省后裔边(//)、通配符(*)和分支([ ])的情形,可进一步划分为$P^{\{/,*,[]\}}$,$P^{\{/,//,[]\}}$,$P^{\{/,*,//\}}$等子类。因$P^{\{/,*,//\}}$没有分支而被称为线性树模式,它是构成$P^{\{/,*,//,[]\}}$类的基础。

当前,影响Twig查询求解性能的因素主要有结构连接算法的效率和Twig模式分解的规模。为解决上述问题,一方面,在关系数据库中过滤出构建结构连接所需结果的元组,以减少数据处理规模;另一方面,采取增大Twig模式分解粒度以期达到减少结构判断操作的数目,从而提高整体计算的性能[4]。DM XML系统结合上述两方面来加速线性Twig查询的匹配。

3.1 DM XML的线性Twig查询的路径集

文献[3]指出,给定线性Twig模式p,$q\in P^{\{/,//,*\}}$,线性模式p包含线性模式$q(p\supseteq q)$,当且仅当存在embedding映射$e:p\rightarrow q$;如果$p\supseteq q$,embedding映射$e:p\rightarrow q$能在多项式时间内被发现。

定义 3 设线性Twig模式p,$q\in P^{\{/,//,*\}}$,若存在embedding映射$f:p\rightarrow q$,使得p的唯一根节点和叶节点分别映射为q的根节点和叶节点,说q蕴含于p,表示$q\vDash p$。

显然,如果线性Twig模式p、q存在$p\vDash q$,则必有$p\supseteq q$。在图3中,p=A/*//D,q_1=A/B/D/D,q_2=A/B/D/F。同时,有embedding映射$e:p\rightarrow q_1$,$f:p\rightarrow q_1$和$h:p\rightarrow q_2$。因此,$p\supseteq q_1$和$p\supseteq q_2$;另外,在embedding映射$f:p\rightarrow q_1$中,p的唯一根节点和叶节点分别映射为q_1的根节点和叶节点,可知$q_1\vDash p$。

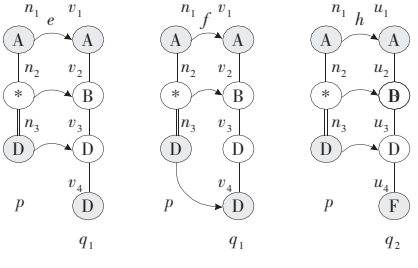

图3 $q_1\vDash p$

定义 4 给定XML文档的关系模式(PT,ET)和常量CP。设线性Twig模式$p\in P^{\{/,//,*\}}$。p的路径集定义为$A(p)=\{p_i\,|\,p_i\vDash p,\ p_i\in\pi_{Path}(\mathrm{PT})\}$,其中,$\pi$为关系投影运算。

例如,在图3中,p=A/*//D。联系图2所示的PT关系表,有$A(p)$={A/B/D,A/B/D/D,A/B/D/D}。显然,对于任一$p_i\in A(p)$,有$p_i\in P^{\{/\}}$。

当PT表较大时,首先根据建立在字段*Tag*、*Len*上的复合索引,找出可能满足条件的路径。例如,p=A/*/B/D不含descendant轴,蕴含于p的路径一定在*Tag*为*D*且*Len*为4的元组中;若p=A/*/B//D含有descendant轴,蕴含于p的路径一定在*Tag*为*D*且*Len*不小于4的元组中。然后,按照文献[3]的发现embedding映射算法从路径集中筛选出路径,求出路径集,避免了对PT表的所有路径逐个匹配。

3.2 DM XML的线性Twig查询

定理 2 给定XML文档的关系模式(PT,ET)和常量CP。对于线性Twig模式$p\in P^{\{/,//,*\}}$,Twig模式p的查询元素集为$R(p)=\cup_{p_i\in A(p)}(\sigma_F\{u|u\in\mathrm{ET}\})$,其中条件表达式$F=(\partial(p_i)*CP\leqslant u.EID\leqslant(\partial(p_i)+1)*CP-1)$,$\sigma_F$为关系选择运算。

证明 如果任一XML文档中元素e,从根元素到元素e的路径为$p_i\in A(p)$,又$p_i\vDash p$,则元素e的路径匹配线性Twig模式p,元素e是所求元素之一。根据路径分区编码规则1,元素e唯一对应的元组u,必满足$u.EID\in[\partial(p_i)*CP,(\partial(p_i)+1)*CP-1]$。因此,$u.EID$能唯一标示元素所求元素$e$。故集合$R(p)$能完全标示出匹配线性Twig模式$p$的元素。

例如,p=A/*//D,且$A(p)$={A/B/D,A/B/D/D,A/B/D/D},令p_1=A/B/D,则有$\partial(p_1)*CP$=3*100=300和$(\partial(p_1)+1)*CP-1$=(3+1)*100−1=399。则在ET表中字段*EID*值在区间[300,399]的元组都匹配线性Twig模式p(如图4)。同理,令p_2=A/B/D/D,p_3=A/B/D/D,字段*EID*值在区间[500,599]或[700,799]的元组都匹配线性Twig模式p。所以$R(p)$={300,301,500,501,700,701,702}。

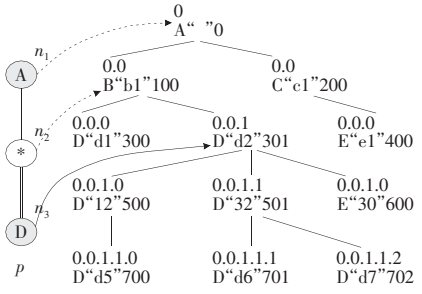

图4 线性Twig模式匹配

从定理2看出,DM XML对于求解线性Twig查询,并不需要结构连接操作。一种简单的方法是执行SQL语句查询按路径聚集在一起的元组,查询数据库的次数为$|A(p)|$。在DM XML原型系统中,直接根据字段*EID*上的索引在ET表上查找所需的元组。

如果线性Twig查询p中出现的后裔边"//"和通配符"*",根据定义3生成了路径集$A(p)$。对于任一$p_i\in A(p)$,p_i是不含"//"或"*"的线性Twig模式,这样也有机地解决了后裔边和通配符的问题。

DM XML 处理线性 Twig 查询为类属 $P^{[/,*,//,[]]}$ 的复杂查询奠定了基础。根据推论 1 和推论 2,容易判断两个元组是否满足 PC 关系和 AD 关系。因此,当线性 Twig 查询后,匹配 $P^{[/,*,//,[]]}$ Twig 查询完全脱离 ET 表。通过线性 Twig 查询,数据处理规模明显减少,舍去了那些不匹配线性模式的元组。同时,Twig 模式分解以线性 Twig 模式为基本粒度,而不是以模式中的节点为粒度,减少结构判断操作的数目。

4 DM XML 的线性 Twig 查询实验分析

4.1 实验环境

本文的实验在一台基本配置为 P4 2.8 GHz,1G RAM,160 G 硬盘的 PC 上运行,底层操作系统是 Windows XP,在 VC++6.0 环境下实现了 DM XML 原型系统,它以达梦关系数据库为平台来实现对 XML 的支持。系统主要分为 XML 文档的验证式读入、XML 数据存储、XPath 和 XQuery 查询、文档序列化输出等功能。

4.2 实验内容

实验所用数据集为 XMark[5],分别选择 110 K,1 M,10 M,50 M 和 110 M 的 XML 文档来生成不同大小的数据库。在实验中,给出了 P1~P5 共 5 个线性 Twig 查询(见表 1)。选择执行时间作为主要的评价指标,这里所给出的执行时间都是重复运行 10 次实验后得到的平均执行时间。为了说明 DM XML 系统的效能和特点,从 4 个方面设计实验:(1)XML 文档的有效线性路径数;(2)线性 Twig 查询解析特性;(3)线性 Twig 查询时间特性;(4)线性 Twig 查询的扩展性。

表 1 实验所用的 5 个 XPath 表达式

No.	Twig 查询
P1	/site/regions/africa/item/description/parlist/listitem/text/keyword
P2	/site/closed_auctions/closed_auction/annotation/description/parlist/listitem/text/keyword/bold
P3	/site/closed_auctions//emph
P4	/site/people/*//education
P5	/site/*/*/name

4.3 实验结果及分析

4.3.1 XML 文档的有效线性路径数

实验中,$CP=2^{16}$。PT 表统计了出现在 XML 文档中的不同路径。PT 表的元组数远远小于文档元素个素,且不随 XML 文档规模的成倍增长而显著增加。如表 2 所示。这表明 XML 文档中存在大量重复的路径,也表明了同质 XML 文档的结构上的相似性。表 2 表明存储标签的空间代价可以接受。

表 2 XMark 的存储特性

Data size/MB	1	11	22	52	116
Elements(*10^6)	0.02	0.17	0.33	0.76	1.78
Max depth	12	12	12	12	12
Path number	339	391	412	546	546
Label size/MB	0.17	1.19	2.31	5.32	12.4

4.3.2 线性 Twig 查询解析特性

DM XML 先根据线性 Twig 查询表达式生成路径集,构成了查询解析过程。P3 含有子孙边,在解析过程中,生成了 9 个类属 $P^{[/]}$ 的 Twig 模式的路径。P4 和 P5 虽然含有通配符,最终各自生成的路径对应一个类属 $P^{[/]}$ 的 Twig 模式。从表 3 中看出,解析过程所占时间很少。

表 3 P1~P5 的解析特性

项目	路径个数	解析时间/ms
P1	1	9 ms
P2	1	10 ms
P3	9	113 ms
P4	1	22 ms
P5	1	18 ms

4.3.3 线性 Twig 查询整体时间

测试对象是 XMark 110 M 文档生成的数据库。为了说明 DM XML 加速线性 Twig 查询算法的有效性,与建立在区间编码上的 TwigStack[6]、iTwigJoin+TL[7]算法进行了对比。TwigStack 算法可以将整体匹配 Twig 查询,避免保存无用的中间结果,对于只有祖先后代边的模式树匹配是最优的,对带有父子边的模式树不能保证最优,它需要对模式树中所有的边都进行连接。iTwigJoin+TL 采用 Tag+Level 索引过滤元素后,完成结构连接运算。DM XML 路径查询时间与解析出来的线性 Twig 模式的个数相关,而与模式的长度无关,通过线性 Twig 查询解析后,直接获得所求元素的范围,涉及的元素种类和数目大为减少,不需要进行结构连接运算。对于含有通配符(*)的 P4、P5,TwigStack 没有使用索引信息,必须遍历 XML 文档,iTwigJoin+TL 虽然采用 Tag+Level 索引,但与查询无关的元素大量存在,两者的查询速度迅速降低,而 DM XML 先从 PT 表中解析出路径集,以此确定 ET 表的待查询的主键范围。从图 5 看出,DM XML 的查询时间性能明显优于其他两种。

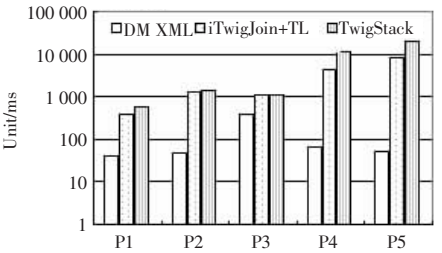

图 5 整体查询时间

4.3.4 扩展性分析

在实验中,DM XML 系统读入 XML 文档而建立相应数据库。图 6 给出了在 110 K、1 M、10 M、50 M、110 M 的 XMark 数据库下,P1 至 P5 的查询扩展性能。从图 6 看出,随着 XML 文档的规模增加,DM XML 执行线性 Twig 查询时间表现出较为缓和的增长。

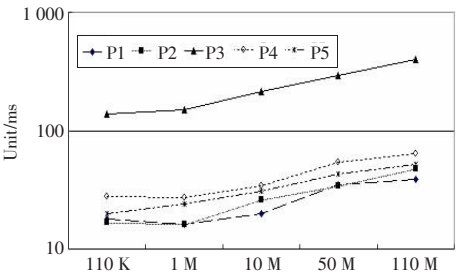

图 6 线性 Twig 查询的扩展性

4.4 实验结论

DM XML 系统采用独特的关系模式,有机结合结构映射

(下转 17 页)