文章编号: 1005-3026(2006)02-0157-04

面向对象XML存储性能分析

张晓琳[1], 谭跃生[2], 张 军[2], 王国仁[1]
(1. 东北大学 信息科学与工程学院, 辽宁 沈阳 110004; 2. 内蒙古科技大学 网络中心, 内蒙古 包头 014010)

摘 要: 针对面向对象XML数据的两种存储模式:集中式和分布式,基于两种存储模式的存储结构信息的路径仓和存储XML数据的数据仓,分别设计并实现了面向对象XML的查询代数.采用支持继承的扩展XML-RL查询语言,使用多态元素、多态引用、独占元素和独占引用四种典型查询,分析了两种存储模式的性能.结果表明查询多态元素和独占元素时,集中式优于分布式,而查询多态引用和独占引用时分布式优于集中式.

关 键 词: 面向对象的XML;路径仓;数据仓;XML-RL
中图分类号: TP 311.13 文献标识码: A

XML正在成为Internet上数据描述和交换的标准.随之而来对XML数据的管理和各种操作的要求逐步提高.由于XML是半结构化数据,所以其数据模型的表达能力要比关系模型和面向对象模型更强,XML数据所具有的特点使得运用关系模型和面向对象数据模型不能使XML数据得到完全的体现,因此XML存储和查询必将成为一个新的研究热点.众所周知,面向对象的方法具有很强的建模能力,将面向对象的特征引入到XML中来提高XML语言的建模能力可以增强对XML数据的管理[1].鉴于以上的需求,将面向对象方法加入到XML中,首先实现了对面向对象XML文档的存储和查询,其次通过采用典型查询语句对两种存储模型进行了测试与比较,分析了面向对象XML存储性能.

1 面向对象的XML文档及其存储模型

XML文档是由一系列嵌套结构构成的,元素由其子元素构成.XML文档中的结构信息由DTD来进行描述,包括为文档提供一个结构框架;为元素定义一个内容模型;数据类型和数据约束等定义.由于引入面向对象的特征到XML中,所以其文档结构就采用扩展DTD[2]来描述,相应的XML文档也具有了元素层次、继承、多重继承等信息.这样的XML文档可称为是面向对象的XML文档[3].面向对象的XML文档由于增加了一些具有面向对象特征的信息,所以现存的XML存储模型不能满足其存储要求.因此设计了两种存储模型[1]:集中式存储和分布式存储.这两种存储模型均包括两部分:路径仓和数据仓[4].路径仓和数据仓均采用二叉树结构.路径仓用来描述XML文档中的层次信息,数据仓用来存储对应于路径仓层次中类的对象.集中式存储与分布式存储在路径仓和数据仓节点具体的定义上有所不同,详细请参见文献[1].

2 面向对象XML查询代数设计

在查询中通常需要解决的一个难题就是设计一套适合的查询代数.因此在实现过程中,以上述两种存储模型为基础,通过对一些XML查询代数的研究[5~8],分别设计了两套查询代数.在这两套查询代数中有些查询代数作用是相同的,但是实现的方法不同.每套查询代数包括两部分,逻辑操作符和物理操作符.在这两套查询代数中逻辑操作符是相同的,包括:文档导航Document、路径导航Path、导航连接Chain、引用操作Reference、选择操作Select、连接操作Glue、聚集操作Congregate、排序操作Sortby、结果构造操作Construct和输出操作Variable.两套查询代数中

收稿日期: 2005-04-18
基金项目: 国家自然科学基金资助项目(60273079); 教育部高等学校优秀青年教师教学科研奖励计划基金资助项目.
作者简介: 张晓琳(1966-),女,内蒙古包头人,东北大学博士研究生; 谭跃生(1959-),男,内蒙古包头人,内蒙古科技大学教授;
王国仁(1966-),男,湖北崇阳人,东北大学教授,博士生导师.

名称相同的物理操作符为 ScanPath, ScanHierarchy, Transition, Transition_S, ScanData, Heir, Exclusive, Node, ScanALLData, Chain, NLJ, Variable_e, Variable_a, Variable_start, Variable_end.

在所设计的逻辑操作符中,路径导航 Path,引用操作 Reference,结果构造操作 Construct 和输出操作 Variable 是典型的面向对象 XML 查询代数,用以支持面向对象的查询功能.设计思想如下: ㈠ 路径导航操作: XML 文档元素间存在一种层次关系,需要通过对路径名的描述为数据的查找指定开始节点和查找路径,得到可以通过该路径到达的元素集; ㈡引用操作: 解决 XML 文档中元素之间相互关联的问题; ㈢结果构造操作: 由于对文档的查询采用的是扩展 XML-RL 查询语言,该语言的查询和结果被严格地分开,需要一个操作符来连接查询部分和结果部分; ㈣输出操作: 使结果可以按要求构造成符合 XML 文档结构的形式.逻辑操作只描述查询的功能,不涉及具体的实现,物理操作符将完成实际的查询执行.在物理操作符中,有着典型的面向对象查询代数特征的是 Heir, Exclusive, Reference_Inherit, Reference_Exclusive, Variable_e 五个物理操作符.它们要完成的功能为: ㈠ Heir(x', name, y) 表示在节点集 x' 中找到名为 name 的类及其子类的节点集,结果绑定到 y 上; ㈡Exclusive(x', name, y) 表示在节点集 x' 中找到名为 name 的类的节点集,结果绑定到 y 上; ㈢Reference_Inherit (x, name, y) 表示得到类名为 name 的节点及其子类节点的元素值或者属性值与引用值 x 相同的 name 类对象节点及其子类对象节点; ㈣ Reference_Exclusive (x, name, y) 表示得到类名为 name 的节点的元素值或者属性值与引用值 x 相同的 name 类对象节点; ㈤ Variable_e (name, node) 构造名称为 name,变量为 node 的元素.

3 存储性能分析

通过在两种存储模型上运用查询代数实现了面向对象 XML 文档的查询.这些查询采用的是扩展 XML-RL[3] 查询语言,该查询语言是在 XML-RL[9] 查询语言的基础上进行扩展,以支持多重继承、多态元素、多态引用、独占元素、独占引用等典型的面向对象查询.实验环境: Windows 2000, P4 2.0 GHz, 1 G 内存,通过采用符合扩展 DTD 但具有不同数据量的 XML 文档用四种查询实例对存储性能进行了测试,测试结果如下.

3.1 查询多态元素

例如 查询一个大学中所有的人的 pid 和 name.

```
querying
http://www.edu.cn/univ.xml/univ/person:$p,
$p/@pid:$d,
$p/name:$n
constructing
result.xml/univ/person:[@pid:$d,
                        name:$n]
```

从图 1 的结果可以看出,分布式查询多态元素的时间要大于集中式.集中式在查询多态对象的时候需要进行判断超类右节点链接的对象哪一个是要查询类的子类,而分布式结构则不需要判断而一次查出.通过测试,该部分查询时间在整个查询语句的查询时间中占的比例不大.集中式在存储如 pid 和 name 等对象中的子元素和属性的时候将其存储在数据仓中二叉树的一个节点上,而分布式则是按照对象的不同,将一个对象存储在二叉树上不同的节点上.这使得在查询对象的属性时,集中式可以在一个节点内很快的查找到 pid 和 name,而分布式则需要首先判断要查找的元素和属性是否在本地节点上,如果不在则需要查询其在超类部分的节点进而找到要查找的元素和属性,故查询时间要比集中式慢.在文档超过 52 M 后集中式和分布式的速度变慢,这是由于当文档超过 52 M 后生成的数据仓要占用的内存大概是原文档的十几倍,占用了过多的内存并且查询的节点数量与 52 M 之前比增多,因此分布式速度下降得很快,而集中式的结构决定它的数据仓大小应该为分布式的二分之一左右,因此相对下降缓慢.所以虽然集中式在查询多态对象的时候时间比分布式多,但在查询对象子元素时却比集中式省时,因此查询整体性能要比分布式好.

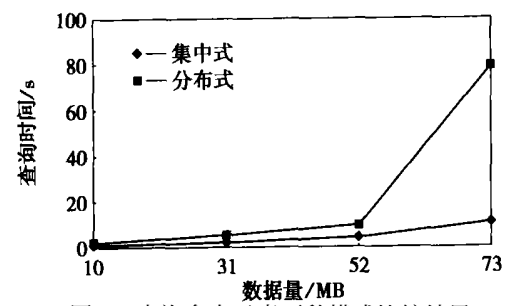

图 1 查询多态元素两种模式比较结果

Fig. 1 Comparison of polymorphic element querying results by two different models

3.2 查询独占元素

例如 查询一个大学中人的 pid 和 name,不

包括 teacher, student 和 TA.

```
querying
http: // www.edu.cn/
univ.xml/ univ/ person ↑ : $ p,
$ p/ @pid: $ d,
$ p/ name: $ n
constructing
result. xml/ univ/ person: [ @pid: $ d,
                             name: $ n]
```

从图 2 的结果可以看出, 分布式查询独占元素的时间在 10 MB 到 52 MB 之间与集中式基本相同, 这是由于查询独占元素不需要判断该元素是否为一个类的子类元素, 集中式查询该部分的时间明显缩短. 而分布式查询该部分的时间则更短, 这是由于分布式查询某个类的独占元素的时候只需要从该类右节点中链接的对象中进行选择, 比集中式搜索的节点少. 但是对于本例来说, 其搜索节点相同. 由于一个类的独占元素比查询该类的多态元素所得到的结果对象少, 因此查询这些对象的子元素和属性必然耗时相对少. 分布式查询子元素和属性所消耗时间在整个查询中所占的比例较大, 因此即使查询的对象少, 其耗时仍然要比集中式多, 而与查询多态元素该部分时间比明显减少, 但是仍然要比集中式耗时. 而分布式在52 MB 以后速度明显变慢的原因与查询多态元素相同.

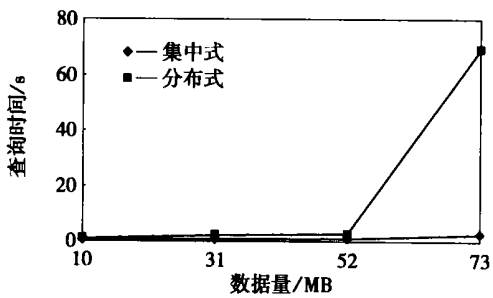

图 2　查询独占元素两种模式比较结果

Fig. 2　Comparison of monomorphis element querying results by two different models

3.3　查询多态引用

例如　查询一个大学中所有的被助教“Alice Bumbulis”教授的 course, 包含 underCourse 和 gradCourse.

```
querying
http: // www.edu.cn/ univ.xml/ / TA: $ a
$ a/ name: $ n( = “Alice Bumbulis”)
$ a/ teaches/ @(course) courses: $ b
constructing
result. xml/ results/ result: [$ b]
```

从图 3 的结果可以看出, 分布式查询多态引用要优于集中式. 这是由于在查询多态引用的时候首先需要建立该元素和 IDREF 中引用的元素的关系, 即首先查询出所有满足要求的元素, 在这个例子中则是查询所有的 course, underCourse 和 gradCourse 元素, 然后从中寻找满足条件的值. 虽然分布式在查询元素的子元素和属性上消耗很多时间, 但是由于在该查询语句中查询的量比较小, 当查到符合要求的值时, 就停止继续查询, 通过测试该部分查询时间在整个查询中不占很大比例, 这时候在该查询中查询所有的 TA 和查询引用的多态元素对象所消耗的时间就占去了查询时间的大部分, 因此集中式所消耗的时间比分布式多. 对于在 52 MB 后集中式和分布式查询速度明显下降, 其各自的原因不同, 集中式是由于两次查询继承元素消耗了很多时间, 而分布式是由于其形成的数据仓的结构占据了较多的内存, 而使查询速度变慢. 但是从图中可以看出, 这两者减慢的速度与查询多态元素和查询独占元素相比要小得多. 由于上述原因, 从整体上看分布式查询要比集中式查询有优势.

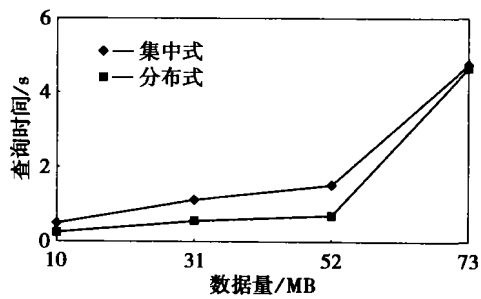

图 3　查询多态引用两种模式比较结果

Fig. 3　Comparison of polymorphic reference querying results by two different models

3.4　查询独占引用

例如　查询一个大学中所有被老师“Alley Srivastava”教授的 course. 不包含 underCourse 和 gradCourse.

```
querying
http: // www.edu.cn/ univ.xml// teacher: $ a
$ a/ name: $ n( = “Alley Srivastava”)
$ a/ teaches/ @(course ↑) courses: $ b
constructing
result. xml/ results/ result: [$ b]
```

从图 4 的结果可以看出, 分布式查询独占引用要优于集中式. 查询独占引用和查询多态引用的差别就是在于查询元素的 IDREF 属性所包含的元素时, 不需要包含该元素的子类元素, 这样就使得该查询结果中包含的对象变少, 整体时间也

要比查询多态引用减少。该曲线与查询多态元素的结果曲线相似,有一点微小的差别就是当文档超过 52 M 后分布式曲线上升的趋势较查询多态引用上升的缓慢,这主要是由于搜索 IDREF 属性所指向的元素对象相对减少,使得查询时间缩短。总体上看这些因素使得分布式在查询独占引用的时候比集中式有优势。

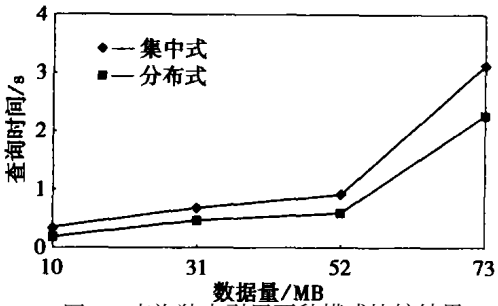

图 4 查询独占引用两种模式比较结果

Fig. 4 Comparison of monomorphis reference querying results by two different models

4 结 论

(1) 对于子类和超类继承的元素和属性存储在子类实例对象的集中式存储模式,在查询多态元素和独占元素时性能较分布式优越。

(2) 对于子类实例对象只存储子类定义的元素和属性,而从超类继承的元素和属性存储在超类实例对象中的分布式查询模式,在查询多态引用和独占引用时又强于集中式。

参考文献:

[1] 张晓琳,王国仁。面向对象的 XML 数据的存储模式研究[J]。小型微型计算机系统, 2004, 25(S): 11- 13。
(Zhang X L, Wang G R. Study on storage methods for object-oriented XML data[J]. *Mini-Micro Systems*, 2004. 25(S): 11- 13.)

[2] Wang G, Liu M. Extending XML schema with nonmonotonic inheritance[A]. *Proceedings of 1st International Workshop on XML Schema and Data Management*[C]. Chicago, 2003. 402- 407.

[3] 张晓琳,王国仁。用继承扩展 XML-RL[J]。小型微型计算机系统, 2005, 26(2): 243- 247。
(Zhang X L, Wang G R. Extending XML-RL with inheritance[J]. *Mini-Micro Systems*, 2005, 26(2): 243- 247.)

[4] 张晓琳,王国仁,赵相国。面向对象 XML 数据索引技术[J]。东北大学学报(自然科学版), 2005, 26(9): 852- 855。
(Zhang X L, Wang G R, Zhao X G. An indexing technique for object-oriented XML data[J]. *Journal of Northeastern University(Natural Science)*, 2005, 26(9): 852- 855.)

[5] Fernanadez M, Simeon J, Wadler P. A semi-monad for semi-structured data[A]. *International Conference on Database Theory*[C]. London, 2001. 263- 300.

[6] Galanis L. *Following the paths of XML data: an algebraic framework for XML query evaluation*[M]. Madison: University of Wisconsin, 2001. 46- 51.

[7] McHugh J, Widom J. Query optimization for XML[A]. *Proceedings of International Conference on Very Large Databases (VLDB)*[C]. Edinburgh, 1999. 315- 326.

[8] Christophides V, Cluet S, Simeon J. On wrapping query languages and efficient XML integration[A]. *ACM SIGMOD Conference on Management of Data*[C]. Dallas, 2000. 141 - 152.

[9] Liu M. A logical foundation for XML[A]. *Proceedings of the 14th International Conference on Advanced Information Systems Engineering (CAiSE' 02)*[C]. Toronto, 2002. 568 - 583.

Storability Analysis of Object-Oriented XML Data

ZHANG Xiao-lin[1], *TAN Yue-sheng*[2], *ZHANG Jun*[2], *WANG Guo-ren*[1]
(1. School of Information Science & Engineering, Northeastern University, Shenyang 110004, China; 2. Network Center, Inner Mongolia University of Science and Technology, Baotou 014010, China. Correspondent: ZHANG Xiao-lin, E-mail: zhangxl2003 @ eyou.com)

Abstract: Aiming at the two storage models for object-oriented XML data, i. e., centralized and decentralized, the query algebra for object-oriented XML is designed and implemented on the bases of their respective path repositories of XML structure information and data repositories of XML data. Introducing extended XML-RL query language to support inheritance, the storability of two models is analyzed by four typical queries: polymorphic element, polymorphic reference, monomorphic element and monomorphic reference. The result indicates that the centralized storage model outperforms the decentralized one at polymorphic element and monomorphic element queries and vice at polymorphic reference and monomorphic reference queries.

Key words: object-oriented XML data; path repository; data repository; XML-RL

(Received April 18, 2005)