一种基于XML小枝查询片段松弛的近似查询与结果排序方法

魏 珂 任建华 孟祥福
(辽宁工程技术大学电子与信息工程学院 葫芦岛 125105)

摘 要 提出了一种基于XML小枝查询片段松弛的近似查询与结果排序方法来实现用户在XML文档中的近似查询:通过收集用户的查询历史来推测用户偏好,并以此计算原始小枝查询分解得到的查询片段的重要程度,然后按照重要程度的排序进行查询松弛;在松弛方法中,根据查询片段数目的不同采用不同的松弛方法,若片段数目较多则以查询片段为粒度对其松弛,较少则以查询结点为粒度对数值查询与非数值查询采用不同的方法进行松弛,得到最为相关的近似查询结果;最后按近似查询结果对原始查询和用户偏好的满足程度进行排序并输出。实验证明,该近似查询方法能够较好地满足用户的需求和偏好,具有较高的查全率和准确率。

关键词 小枝查询,近似查询,查询松弛,用户偏好,结果排序
中图法分类号 TP311.131 **文献标识码** A

Approximate Query and Results Ranking Approach Based on XML Twig Query Fragment Relaxation

WEI Ke REN Jian-hua MENG Xiang-fu
(College of Electronics and Information Engineering, Liaoning Technical University, Huludao 125105, China)

Abstract Based on XML twig query fragments relaxation, this paper proposed an approximate querying and results ranking approach to achieve the approximate query results against XML documents: our method gathers the query history to speculate the user's preferences, which is used to calculate the importance for each query fragment of the twig query, and relax the original query according to the sequence of the fragments' importance; based on the number of query fragments we adopt different relax way: if the number>2, relax the original query according to the granularity of the fragment; if the number<2, relax the original query according to the granularity of query node, and adopt a different way to relax the numerical query and non-numerical query, and then obtain the most relevant query results. Finally, the relevant query results are ranked based on their satisfaction degree to the original query and the user preferences. Our experiment shows that the approximate querying and the results ranking approach can efficiently meet the user's needs and user's preferences, has the high recall and precision.

Keywords Twig query, Approximate query, Query relaxation, User preferences, Results ranking

当前,XML近似查询已经成为研究的热点,许多研究者已经开始致力于XML数据库近似查询方法的研究[1-5]。文献[1]提出了一种基于文档属性单元松弛的XML近似查询方法,即将XML文档中的叶子结点和属性结点作为处理的属性单元,找出它们之间的近似函数依赖关系并求出近似候选码和近似关键字,然后根据属性单元的支持度对原始查询进行查询松弛,从而得到近似查询结果。文献[2]是在基本XML模式中先对查询的结构进行匹配,鉴定与其他相关模式的结构是否相似,然后在查询时自动进行查询重写。文献[3]对Michigan大学Timber研究小组提出的基于约束的XML查询重写算法进行了改进,引入映射规则中的约束条件解决了内定谓词问题,消除了阻碍重写的Skolem函数。文献[4]提出了一个在XML数据库上进行关键字近似查询的系统,它对XML数据进行建模形成图形或树形结构,并利用XML模式描述查询结果,对查询系统进行优化。文献[5]开发了一个在线的XML探测系统AQAX,它能够快速实现大规模数据集的探测。

查询松弛是目前XML近似查询的主要方法,提出查询松弛的主要原因是用户输入的查询描述不能够完全反映用户实际的查询意图,而目前的查询系统又要求用户对其查询意图进行精确的描述。现有的大部分XML近似查询方法都是针对查询结构进行的,即对原始查询进行结构松弛来得到近似查询结果,但目前这些方法很少考虑用户偏好,事实上用户偏好是反映用户真实的查询意图的重要根据,它能有效避免查询空结果和信息泛滥现象。目前获取用户偏好的主要方法是利用上下文、领域知识库或XML文档结构推测用户偏好。文献[6]开发的一种偏好查询优化器根据用户对结构的特定偏好,高效地评估、鉴定和简化必要的扩展查询,它不仅决定了一个最优组扩张查询,还保存了用户偏好的特定序列。文献[7]基于领域知识库的条件偏好,提出一种在推荐系统中处

到稿日期:2011-12-06 返修日期:2012-03-16 本文受国家青年科学基金项目(61003162)资助。
魏 珂(1986—),男,硕士生,主要研究方向为XML数据库查询技术,E-mail:williekewei@gmail.com;任建华(1973—),男,副教授,硕士生导师,主要研究方向为数据库理论;孟祥福(1981—),男,博士,讲师,主要研究方向为Web数据库与XML数据个性化柔性查询技术。

理条件偏好的方法。文献[8]根据位置的上下文和偏好意识的数据库服务引入了系统结构 CareDB，使其能够根据每一位用户的偏好和上下文对其功能进行修剪。

目前 XPath 和 XQuery 是最常用的 XML 查询方式，而 XPath 表达式又是 XQuery 的基础，因此本文采用 Xpath 表达式表示 XML 查询。目前，XML 查询中最为核心的是小枝模式查询，而基于连接的查询算法是小枝模式查询最重要的执行策略：先将查询分解成若干查询片段，再对各查询片段分别查询得到相应的匹配结果，然后将各查询片段的结果合并，得到最终的查询结果。当前的 XML 小枝查询机制认为用户的查询表达是准确的，返回的查询结果也完全符合用户的查询要求，而实际上由于 XML 文档结构和内容的复杂性，普通用户很可能对 XML 文档不了解，并且用户的查询意图本身就可能不十分准确，因此，用户即使准确表达了他的查询意图，仍可能得不到想要的查询结果，此时用户会希望进行查询松弛来获得满足其真实意图的近似查询结果。

例如，图 1 是一个 XML 数据树，图 2 是在图 1 数据树上进行的小枝查询，按照基于连接的查询方法，图 2 中的小枝查询被分解为“UsedCar/Make=Ford、“UsedCar/price<10000”和“UsedCar/Color=white” 3 个查询片段。很明显，查询片段“UsedCar/Make=Ford”和“UsedCar/Color=white”完全匹配图 1 中的数据树。但是按照小枝查询的精确匹配，查询结果为空。事实上如果用户主要想获得 Make 为 Ford、Color 为 white 的二手车，而对二手车的价格的要求不那么严格，则此类查询是失败的，所以有必要基于用户偏好对 XML 小枝查询进行松弛，获得原始查询的近似结果。

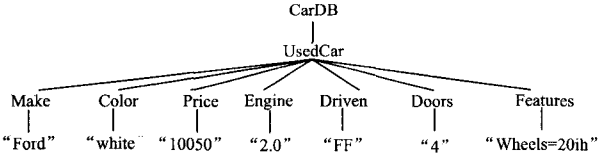

图 1　XML 数据树

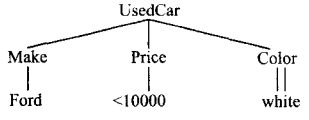

图 2　小枝查询

我们已经完成了如下工作：收集查询历史并对其分析得到用户的偏好，基于推测出的用户偏好对原始查询的查询片段区分重要程度，然后依次删除重要程度最低的查询片段进行松弛。以上工作提出了一种通用的查询松弛方法，但它没有考虑到查询片段数目较少时，以查询片段为粒度进行删剪是不合理的，因此，本文基于已有研究，由查询历史推测的用户偏好得出查询片段松弛顺序后，对查询片段较多的查询（如片段数大于 2），仍然按重要程度的顺序以查询片段为粒度对查询片段进行删剪；当查询片段较少时，将以查询结点为粒度进行查询松弛，并将数值查询作为特殊情况进行处理，从而获得与用户查询意图最为相关的近似查询结果。

本文第 1 节介绍本文所用到的相关定义；第 2 节概述本文近似查询方法的整体框架；第 3 节提出根据查询历史推测查询片段重要程度的方法；第 4 节介绍本文提出的查询松弛方法，第 5 节详细说明本文近似查询方法的实现过程；第 6 节通过实验验证了本文的改进方法；最后对全文进行综述。

1　XML 小枝查询相关定义

定义 1（XML 数据树）　XML 数据树 $T=(r_T,N,E,R)$ 是按照一定的映射关系将 XML 文档转化而成的无向多叉树。数据树 T 的根结点用 r_T 表示，N 是数据树 T 的结点集合，E 是数据树 T 的边集合，R 是数据树 T 中的关系集合[9]。

定义 2（小枝模式查询）　小枝模式查询简称小枝查询，用四元组 $Q=(r_t,V_t,E_t,R_t)$ 表示，r_t 是查询树 Q 的根，V_t 表示查询树的结点集合，E_t 表示查询树的边集合，包含孩子边（/）和后裔边（//）两个子集，R_t 表示查询树的结构关系集合。

小枝查询过程是将小枝查询中的结点映射到数据树中，使其满足：

（1）小枝查询中的结点类型与数据树中的结点类型保持一致。

（2）小枝查询中任意两个查询结点间的关系与其在数据树中对应的映射结点间的关系保持一致[10]。

定义 3（查询片段）　查询片段 $q_i=(v,e,r)$ 是将小枝查询 Q 按整体小枝连接算法进行分解得到的，查询片段是通过编码快速执行的基本单位，它满足：

（1）一个小枝查询 Q 由一个或多个查询片段 q_i 构成，各片段 q_i 是由 Q 分解得到的；

（2）查询片段 q_i 的结点 v、边 e 以及它们之间的关系 r 是小枝查询 Q 的结点 V_t、边 E_t 和它们之间的关系 R_t 的对应子集，即 $v\in V_t,e\in E_t,r\in R_t$。

2　查询松弛解决整体框架

本文提出的基于 XML 小枝查询片段松弛的近似查询方法主要包括 4 个方面：

（1）推测查询片段的重要程度：先将用户输入的小枝查询分解成查询片段，将查询片段与近期收集的查询历史进行比较，推测用户对各个查询片段的偏好程度。

（2）查询松弛方法：由查询片段的重要程度得到各查询片段松弛的先后顺序，重要程度最低的片段最先松弛，根据查询片段的数目，分两种情况进行处理：查询片段较多时，以查询片段为粒度，每一次松弛删掉一个查询片段；查询片段较少时，以查询节点为粒度进行松弛，并分为数值查询和非数值查询两种情况：对数值查询通过数值扩展实现松弛，对非数值查询按边泛化或叶子节点删除两种方式进行松弛。

（3）查询处理：按照本文提出的近似查询方法在数据集上进行查询，返回与原始小枝查询最为相关的近似查询结果。

（4）查询结果排序：对返回的近似查询结果，按照它们对原始查询和用户偏好的满足程度进行排序。

各步骤的实现方法分别在第 3 节和第 4 节中给出。图 3 给出了本文提出的 XML 小枝近似查询方法的整体框架。

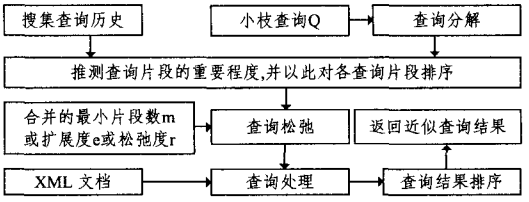

图 3　改进的近似查询方法的整体框架

3　查询片段的重要程度

查询片段排序的基本方法是：收集查询历史，以此分析用

户的偏好，然后对原始查询的查询片段区分重要程度，并根据查询片段的重要程度对原始查询的查询片段进行排序，以使重要程度最低的查询片段最先进行松弛。

查询历史是对用户近期的查询进行分解得到的查询片段的集合，它能很好地反映出用户近期对某内容的偏好。查询历史可用 $H=\{(U_1,Q_1,T_1),\cdots,(U_k,Q_k,T_k)\}$ 表示，其中 Q_i 表示用户提出的第 i 条查询，T_i 表示提交此查询的时间。将查询历史中的查询分解为查询片段，将所有不重复的查询片段组成查询历史集合 $S=\{q_1,q_2,\cdots,q_n\}$，n 为集合中的查询片段的数目。根据查询历史可以得出用户偏好的模型。

定义 4(用户偏好) 用户偏好 $P\{(q_1,w_1),(q_2,w_2),\cdots,(q_n,w_n)\}$ 由查询片段及其权重组成，其中 q_i 是 S 中的第 i 个查询片段，w_i 是第 i 个查询片段的权重，表示用户对第 i 个片段 q_i 的偏好程度。下面介绍的新鲜度和兴趣度共同决定了查询片段的权重 w_i。

3.1 查询片段新鲜度

定义 5(查询片段新鲜度) 查询片段的新鲜度定义为：

$$f_i=\begin{cases}(f_i-\gamma)^{t-T}, & f_i>\gamma\\ 0, & 0<f_i<\gamma\end{cases} \tag{1}$$

式中，f_i 是查询片段 q_i 的新鲜度，初始值为 1，γ 是 0 到 1 之间的一个极小值。t 是用户此次提交查询片段 q_i 的时间，T 是查询历史中用户提交查询片段 q_i 的时间。

用户近期的查询内容可以很好地反映用户的偏好。比如用户一天前查询过“UsedCar/Make＝Ford”，而上次查询“UsedCar/price＜10000”是在一周前，如果该用户本次输入查询“UsedCar[Make＝Ford]/price＜10000”，此查询按基于连接的查询方法可分解成“UsedCar/Make＝Ford”和“UsedCar/price ＜10000”。在本文的近似查询方法中，将首先返回满足“UsedCar[Make＝Ford]/price＜10000”的结果，然后返回只满足“UsedCar/Make=Ford”和“UsedCar/price＜10000”的结果，且满足“UsedCar/Makee＝ Ford”的结果应优先返回，因为用户对最近查询过的内容应具有较高的偏好程度。

当小枝查询包含多个查询片段时，对查询片段的新鲜度进行归一化，如下：

$$\mathrm{fre}(q_i)=f_i/\sum_{j=1}^{n}f_j \tag{2}$$

式中，$\mathrm{fre}(q_i)$是对查询片段 q_i 进行归一化得到的新鲜度。

3.2 查询片段的兴趣度

定义 6(查询片段的兴趣度) 查询片段在查询历史中的兴趣度定义为：

$$p_i(n)=\begin{cases}p_i(n-1)+\lg(n+\eta)\\ p_i(n-1)\end{cases} \tag{3}$$

式中，η 是一个偏移常数，$p_i(n-1)+\lg(n+\eta)$是第 n 次使用过系统后查询片段 q_i 的兴趣度，$p_i(n-1)$是用户第 $n-1$ 次使用过系统后查询片段 q_i 的兴趣度。

用户输入的查询次数也能够反映用户的偏好，相对于查询次数少的内容，用户应当更偏好于查询次数较多的内容。例如，用户此次的查询是“UsedCar[Make＝Ford] /price＜10000”，用户近期查询“UsedCar/Make＝Ford”的次数较多而查询“UsedCar/price＜10000” 的次数很少，则该用户可能是对二手车的品牌 Ford 比较感兴趣而对其价格不是很重视。对查询片段的兴趣度进行归一化如下：

$$\mathrm{pre}(q_i)=p_i(n)/\sum_{j=1}^{n}p_j(n) \tag{4}$$

式中，$\mathrm{pre}(q_i)$代表查询片段 q_i 归一化之后的兴趣度。

根据上面得出的查询片段的新鲜度和兴趣度，将查询片段 q_i 的重要程度 w_i 定义为：

$$w_i=a*\mathrm{fre}(q_i)+(1-a)*\mathrm{pre}(q_i) \tag{5}$$

式中，$a\in[0,1]$，用于决定新鲜度与兴趣度对 q_i 的重要程度 w_i 产生的不同程度的影响。

4 查询松弛方法

对于给定的小枝查询 Q，按照整体小枝连接算法的思想将其分解为 n 个查询片段(n 即为查询片段数)：

(1)当 $n>2$ 时，以查询片段为粒度进行查询松弛。

在查询阶段得到各个查询片段对应的查询结果，在合并阶段按上一节得到的查询片段的重要程度排序结果依次取 n 个查询片段中重要程度最高的 $n,n-1,\cdots,m$ ($m<n$)个查询片段的解进行合并，构造最终的查询模式匹配结果。

(2)当 $n\leqslant 2$ 时，若

1. 查询片段 q_i 为数值查询，对 q_i 的查询条件进行扩展。

扩展后的查询与原始查询的比值 e 是扩展查询相对于原始查询的扩展程度，即扩展度。一般情况下，$0<e<2$：

a)当查询片段 q_i 为求解大于某一数值 $value>d$ 时，数值范围扩展度为 $value>d*e$，$0<e<1$；

b)当查询片段 q_i 为求解小于某一数值 $value<d$ 时，数值范围扩展度为 $value<d*e$，$1<e<2$；

c)当查询片段 q_i 为求解等于某一数值 $value=d$ 时，数值范围扩展度为 $d*e_1<value<d*e_2$，$0<e<2$；

d)当查询片段 q_i 为求解大于某一数值 $d_1<value<d_2$ 时，数值范围扩展度为 $d_1*e_1<value<d_2*e_2$，$0<e<2$。

经过数值区间扩展的查询片段 q_i'的重要程度为 $e*w_i$，其中 w_i 为 q_i 的重要程度。若给定数值的扩展度 e，则可以计算出松弛范围，在查询阶段可以得到原始数值查询以及扩展查询的结果。

2. 查询片段 q_i 为路径查询：按以下两种方式[11]得到 q_i 的松弛查询 q_i'：

a)边泛化：q_i 中的一个“/”边由“//”边代替。

b)叶子节点删除：q_i 中的一个片段 $a/b//c$ 可由 a/b 代替。

松弛后的查询对应原始查询有一个松弛程度 r，即松弛度。若给定查询片段的松弛度 r，可以得到与原始查询片段相近程度达到松弛要求的查询片段。同样，在查询阶段得到查询片段以及对应松弛查询的结果。

5 查询松弛与结果排序

Bruno 等人[12]提出的 TwigStack 算法是第一个整体小枝连接算法，它利用链接堆栈保存中间结果，然后对中间结果进行合并得到最终的查询结果，从而在整体上处理小枝模式查询。TwigStack 算法(算法 1)分两个阶段：首先输出小枝查询中从根到叶单条路径的中间结果(第 1－11 行)；然后将中间结果合并生成最后的查询结果(第 12 行)。

算法 1 整体小枝连接算法-TwigStack 算法

输入：小枝查询 Q，n 个查询结点对应的流 $T_1,T_2,\cdots,T_n$(n 为小枝查询的查询结点数)

输出：将小枝模式查询 Q 的匹配结果按从叶到根有序输出

TwigStack (Q, T_1, T_2, …, T_n)

1. while (not end (Q))
2. qact=getNext(Q);
3. if (not isRoot (qact))
4. cleanStack (Sparent (qact), nextBegin (T_{qact}));
5. if (isRoot (qact) or (not empty (Sparent (qact))))
6. cleanStack (S_{qact}, nextBegin (T_{qact}));
7. moveStreamToStack (T_{qact}, S_{qact}, pointer to top (Sparent (qact)));
8. If (isLeaf (qact))
9. showSolutions (qact, 1);
10. pop (S_{qact});
11. else advance (T_{qact});
12. mergeAllPathsolutions ();

本文基于算法 1 提出了一种基于 XML 小枝查询片段松弛的近似查询与结果排序方法。此方法改进了算法 1 在第二个阶段(第 12 行)的合并过程中进行查询松弛，得到最终的近似查询结果，并将其排序输出。本文提出的基于 TwigStack 的查询松弛及近似结果排序方法见算法 2。

算法 2 基于 TwigStack 的查询松弛及近似结果排序方法

输入：查询片段的匹配结果 PS，进行合并的最小片段数 m 或扩展度 e 或松弛度 r，查询片段松弛的程度 t，各查询片段的重要程度 w_i

输出：按序输出小枝查询 Q 的近似查询结果 M

1. for each $q_i \in Q$ do
2. PS←查询片段 q_i 的匹配结果；
3. w_i←查询片段 q_i 的重要程度；
4. 将查询片段按其重要程度排序；
5. if(n>2)
6. m ←进行合并的最小查询片段数；
7. while (n>=m)
8. 合并 PS 中查询片段的重要程度最高的 n 个查询片段的查询结果；
9. 根据 w_i 计算近似查询结果对原始小枝查询的满足程度 A；
10. n--；
11. else
12. if (查询片段为数值查询)
13. t ←查询片段松弛的程度 t
14. 根据 t 得到数值的扩展范围，并得到相应的查询结果。
15. else if (查询片段为非数值查询)
16. r ←查询片段松弛的程度 r
17. 根据 t 得到原始查询的松弛查询，并得到相应的查询结果。
18. 根据各查询片段及松弛查询的重要程度合并满足松弛程度的查询结果；
19. 根据 w_i 计算每个结果对原始查询和用户偏好满足程度 A；
20. M←按满足程度降序输出最终的查询模式匹配结果。

算法 2 的主要执行过程如下：

(1) 按 Zhang 编码对小枝查询进行编码。

(2) 对原始查询进行查询分解，得到 n 个查询片段。

(3) 分别查询各个查询片段，得到 n 个查询片段的解，即中间结果。

(4) 根据第 3 节中查询片段重要程度的计算方法计算 fre(q_i)和 pre(q_i)得到用户对查询片段 q_i 的偏好程度。若查询片段数大于 2，根据 w_i 对各查询片段进行重要程度排序；若查询片段数小于 2，根据 w_i 由低到高对查询片段考虑扩展度和松弛度，得到查询片段 q_i 的松弛查询及其重要程度，根据 w_i 对各查询片段及松弛查询进行重要程度排序。

(5) $n>2$ 时，根据查询片段的重要程度，依次合并 n 个查询片段中重要程度最高的 $n, n-1, \cdots, m(m<n)$ 个查询片段的中间结果，生成最终的近似查询结果；$n \leqslant 2$ 时，先得到精确满足查询片段的最终结果，再根据重要程度得到重要程度大于某一阈值的最终匹配结果。

(6) 通过 $A=\sum_{i=1}^{k} w_i$ 计算近似查询结果对原始小枝查询的满足程度，其中 k 是近似查询结果对应查询的片段数，w_i 是对应各查询片段的重要程度。

(7) 降序输出近似查询结果。

在本方法中，精确匹配的查询结果首先返回，然后按满足程度的高低依次返回查询松弛的结果。

6 实验

本节主要介绍实验环境及测试数据，描述实验方法和实验结果并加以分析。

6.1 实验环境和测试数据

本次实验是在 Intel(R) Core(TM) i3 CPU M 380 @ 2.53 GHz, 2GRAM, 500G 硬盘和 Microsoft Windows XP 操作系统下进行的，由 IBM XML data generator[13] 产生测试文档，生成以下两个 XML 数据集：

(1) 数据集 CarDB. xml：从 Yahoo! Autos 网站[14] 随机获得 100,000 条二手车记录生成 CarDB. xml 数据集(见图 4)，其最大深度为 3，包含 100,000 个 UsedCar 元素。

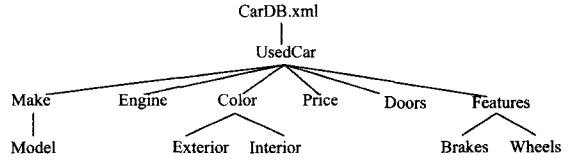

图 4 数据集 CarDB. xml

(2) 数据集 BookDB. xml：从 Amazon 网站[15] 随机获得 5000 条图书记录生成 BookDB. xml 数据集(见图 5)，其最大深度为 4，包含 5000 个 Book 元素。

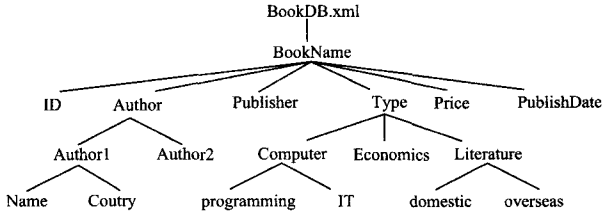

图 5 数据集 BookDB. xml

基于以上数据集，用 Java 语言和 eclipse3.0 编译工具建立了查询系统，邀请测试者提交查询，经过 1 年左右获得了各测试者的查询历史，经过整理，最终为 CarDB 和 BookDB 两个数据集保留了 2200 条和 1890 条查询历史。

6.2 查询松弛方法验证

本实验对查询系统进行测试，并对该系统的使用者进行用户调查，验证本文近似查询方法中查询松弛的查全率和准确率。查全率(Recall)是 XML 文档的元素中符合查询条件的查询结果的比率。准确率(Precision)指查询结果中与用户真实查询意图相关元素的比率[16]。在下面的实验中 10 位测试者根据各自需求和偏好，在 CarDB. xml 和 BookDB. xml 上

分别提出 1 条测试查询(见表 1)。

表 1 CarDB. xml 的数据子集 H1 上的 XPath 测试查询

Query ID	XPath expression
Q1	//UsedCar[Make="Volkswagen"/Model="Jetta"][Price<="$9000][Color="black"][Engine>="2.0"]/Features[wheel="15inch"]
Q2	//UsedCar[Make="Chevrolet"/Model="Cobalt"]/Price>="$10000"
Q3	//UsedCar[Make="Audi"/Model="A6"][Price<="$30000"][Color/Exterior="black"]/Engine>="3.0"
Q4	//UsedCar[Make="Honda"/Model="CR-V"][Price<="$6000"]/Color="white"
Q5	//UsedCar[Make/Model="Classic"][Price<="$9000"][Color/Exterior="gary"][Engine>="2.0"]/Features[Brake="Anti-lock"]
Q6	//UsedCar[Make="Toyota"/Model="Prius"]/Price<="$9000"
Q7	//UsedCar[Make="Ford"/Model="Tundra"][Price<="$10000"][Color="white"]/Features[wheel="13inch"]
Q8	//UsedCar[Make="BMW"/Model="M5"][Price<="$20000"]/Engine>="3.0"
Q9	//UsedCar[Make="Nissan"/Model="350Z"][Price<="$8000"]/Engine>="2.0"
Q10	//UsedCar[Make="Dodge"/Model="Caliber"][Price<="$8000"][Color/Interior="pink"]/Features[Brake="Anti-lock"]

在测试查询结果的查全率和准确率时,要求测试者从以上两个较大的数据集中逐条查找所有与其真实意图匹配的 UsedCar 和 Book 元素是不现实的。因此我们针对各条测试查询,取两个数据集中的 80 条与查询相关的或不相关的适当元素生成对应的小数据集 H_i,然后,在查询结果的查全率和准确率验证阶段,只需对比本文系统基于数据集 H_i 的近似查询结果与测试者在数据集 H_i 中标示出的满足用户查询意图元素的相同数目,与之相同的结果数越多,则查全率就越高,近似查询结果中与户查询意图相同的元素越多,则准确率越高。

例如,表 1 是 10 位测试者在 CarDB. xml 数据子集 H_1 上提出的 10 条查询。查询 Q_1 有 5 个查询片段,按照本文的近似匹配算法中查询片段数较多对应的方法,先对 Q_1 精确匹配获得 1 条精确结果,此结果在用户调查时在 H_1 中被标示为符合对应 Q_1 的查询意图的结果;然后对 Q_1 进行查询松弛:基于测试者的查询历史得到 Q_1 的 5 个查询片段的重要顺序依次为:Make="Volkswagen"/Model="Jetta">Color="black">Price<="$9000">Engine>="2.0">Features/wheel="15inch",第一次查询松弛 $m=n-1(n=5)$时删除重要程度最低的片段 Features/wheel="15inch",即返回的结果可以不满足此片段,因为用户对其最不关心,查询得到了 5 个 UsedCar 元素,其中 4 个符合用户的要求;第二次松弛时($m=n-2$),删掉 Features /wheel="15inch"和 Engine>="2.0"两个查询片段,得到了 11 个查询结果,其中 8 个在 H_1 中被测试者标示为相关结果。

查询 Q_2 有 2 个查询片段,测试者在 H_1 中标注的与 Q_2 相关的元素共有 35 个,按照本文的近似查询方法对 Q_2 进行严格匹配得到了 4 个查询结果,这 4 个结果都被测试者标示为相关元素;然后对 Q_2 进行查询松弛,根据查询历史得到 Q_2 的 2 个查询片段重要程度顺序是 Make="Chevrolet /Model="Cobalt">Price>="$10000"。我们首先对重要程度低的查询片段 Price>="$10000"进行松弛。根据已知松弛度 0.8,得到扩展数值区间 Price>="$8000",对//UsedCar[Make="Chevrolet/Model="Cobalt"]/Price>="$8000"进行查询,返回 21 个查询结果,19 个结果被测试者标注为相关的元素;再对 Make="Chevrolet/Model="Cobalt"进行松弛,根据已知的松弛度 0.8,经过结点叶删除得到//UsedCar[Make="Chevrolet"][Price>="$8000"],对其查询得到了 36 个查询结果,34 个结果被测试者标注为相关的元素。

同样按照本文的近似查询方法分别对表 1 中的 Q_3,…,Q_{10} 进行查询松弛,表 2 给出了按照本文方法得到的近似查询结果与测试者在 H_1 中标示出的一致结果数目的对比。

表 2 CarDB. xml 数据子集 H_1 上近似结果和相关结果数对比

Query ID	Exactly match /Relevant results	Query relaxation (m=1) /Relevant results	Query relaxation (m=2) /Relevant results	Relevant results in H_i
Q_1	1/1	5/4	11/8	11
Q_2	4/4	21/19	36/34	35
Q_3	5/5	21/20	29/26	27
Q_4	3/3	9/9	14/12	17
Q_5	3/3	21/20	24/22	23
Q_6	7/7	11/10	18/17	18
Q_7	2/2	5/4	13/9	11
Q_8	6/6	17/16	26/22	26
Q_9	6/6	12/10	18/15	19
Q_{10}	2/2	13/11	21/16	17

同样地,在 BookDB. xml 数据子集 H2 中进行了近似查询效果测试,得到表 3 所列的近似查询结果数目和相关结果数目的对比。

表 3 BookDB. xml 数据子集 H_2 上近似结果和相关结果数对比

Query ID	Exactly match /Relevant results	Query relaxation (m=1) /Relevant results	Query relaxation (m=2) /Relevant results	Relevant results in H_i
Q1	3/3	15/13	22/20	23
Q2	2/2	8/7	15/13	15
Q3	4/4	28/23	29/23	35
Q4	2/2	23/22	26/24	26
Q5	3/3	9/8	14/11	13
Q6	3/3	11/10	14/12	17
Q7	2/2	8/8	12/11	12
Q8	1/1	11/10	21/17	17
Q9	3/3	16/15	21/18	19
Q10	3/3	18/16	25/22	22

根据表 2 和表 3 中的数据,很容易计算出在 CarDB. xml 和 BookDB. xml 上对 10 条测试查询进行精确匹配的平均查全率分别为 19.8%和 13.4%,同时可以计算出准确率均为 100%;依照本文的近似查询方法,根据查询片段数目的不同使用了不同的松弛方法(若片段数>2,则删掉最不重要的查询片段;若片段数≤2,则对最不重要的片段进行松弛),第一次查询松弛在两个测试数据集上分别得 61.2%和 61.5%的平均查全率和分别为 90.3%和 89.6%的平均准确率;第二次查询松弛得到分别为 89.4%和 84.6%的平均查全率和分别为 85.7%和 84.5%的平均准确率。由以上数据可以得出,随着 m 值逐渐减小,查全率逐渐提高,而准确率逐渐降低。本文的方法得到了较高的查全率和准确率。

结束语 本文基于整体小枝连接算法 TiwgStack 提出了一种基于 XML 小枝查询片段松弛的近似查询与结果排序方

法。该方法基于收集查询历史推测用户的偏好，并根据小枝查询分解得到的查询片段数目的不同，使用不同的方法按照查询片段重要程度的高低对原始查询进行松弛。通过实验验证了该方法可以有效地提高小枝查询的查全率和准确率。在本文近似查询方法中，由查询历史推测的用户偏好是否准确直接影响近似查询结果的正确性，如果用户偏好的推测不准确，则不能保证得到的近似查询结果是用户所需的，因此，用户偏好的推测将仍然是今后工作的研究重点；同时，对查询松弛作限定条件的合并的最小片段数 m 值以及扩展度 e 和松弛度 r 的取值也是将来研究的重点。

参 考 文 献

[1] 孟祥福，严丽，张文博，等. 基于文档属性单元松弛的 XML 近似查询方法[J]. 计算机研究与发展，2010，47 (11)：1936-1946

[2] Polyzotis N, Garofalakis M, Ioannidis Y. Approximate XML query answers [C]// Proceedings of the 23rd ACM SIGMOD International Conference on Management of Data. 2004：263-274

[3] 金鑫，金远平. 一种改进的基于约束关系的 XML 查询重写算法研究[J]. 计算机研究与发展，2007，44 (5)：845-852

[4] Balmin A, Hristidis V, Koudas N, et al. A System for Keyword Proximity Search on XML Database[C]// Proceedings of the 29th VIDB Conference. Germany，2003

[5] Spiegel J, Pontikakis E D, Budalakoti S, et al. AQAX：a system for approximate XML query answers [C]// Proceedings of the 32nd International Conference on Very Large Data Bases. 2006：1159-1162

[6] Cho S, Balke W-T. Building an Efficient Preference XML Query Processor[C]// SAC' 09. USA，2009：1585-1586

[7] Yu Zhi-yong, Yu Zhi-wen, Zhou Xing-she, et al. Handling Conditional Preferencesin Recommender Systems[C]// IUI'09. USA，2009：407-411

[8] Mohamed F, Justin J. Toward Context and Preference-Aware Location-based Services[C]// MobiDE'09. USA，2009：25-32

[9] 衡星辰，覃征，邵利平，等. 基于两阶段查询重写的 XML 近似查询算法[J]. 电子学报，2007，35 (7)：1271-1278

[10] 徐超，张东站. sTwig——一种基于流的 XML 小枝匹配算法 [J]. 计算机研究与发展，2010，47：86-92

[11] Amer-Yahia S, Koudas N, Marian A. Structure and content scoring for XML[C]// Proc of International Conference on Very Large Data Bases. New York：ACM，2005：361-372

[12] Bruno N, Srivastava D, Koudas N. Holistic Twig joins：Optimal XML pattern matching [C]// Proceedings of the 21st ACM SIGMOD International Conference on Management of Data. 2002：310-321

[13] IBM Corporation XML data generator [EB/OL]. http://www.alphaworks.ibm.com/tech/xmlgenerator，2010-08-11

[14] http://autos.yahoo.com/，2009-12-23

[15] http://www.amazon.cn/，2010-06-19

[16] Su W, Wang J, Huang Q, et al. Query result ranking over e-commerce web databases [C]// Proceedings of the 15th ACM Conference on Information and Knowledge Management. 2006：575-584

(上接第 151 页)

节能方法，即利用低能耗的 Flash 存储设备减少高能耗的内存的使用，减少对磁盘的访问，从而降低整体的功耗。但 Flash 存储设备造价较高，而且作为缓存层频繁写入会严重影响设备的寿命，因此目前这类方法也没有广泛采用。另外，Zhu 等人[11]研究了通过调整内存缓存的方法来进一步延迟磁盘空闲时间，从而提高节能效果。EERAID[12]、eRAID[13]和 PARAID[14]等方法研究了如何在磁盘的冗余存储上进一步节能。

结束语 本文在对流媒体存储系统的节能和服务质量建模的基础上，得出磁盘节能随负载降低以指数关系增加的结论，并以此为基础设计并实现了冷数据集中的节能方法。通过仿真实验表明，在保障良好服务质量的前提下，该方法节能效果显著提升，是一种流媒体环境中的高效节能方法。

参 考 文 献

[1] Nebuloni G. Energy Footprint of the European Server Infrastructure, 2008, and 2009-2013 Forecast [EB/OL]. http://www.idc.com/getdoc.jsp? containerId=GE11R9，2009-10

[2] Yarow J. Videos on Youtube grew 123% year over year, while Facebook grew 239% [EB/OL]. http://www.strangelove.com/blog/2010/06/videos-on-youtube-grew-123-year-over-year-while-facebook-grew-239，2010-06

[3] Colarelli D, Grunwald D. Massive Arrays of Idle Disks for Storage Archives [C]// Proc. 2002 ACM/IEEE Conf. Supercomputing. Los Alamitos, CA, USA：IEEE，2002：1-11

[4] Pinheiro E, Bianchini R. Energy Conservation Techniques for Disk Array-Based Servers [C]// Proc. 18th Int'l Conf. Supercomputing. New York, USA：ACM，2004：68-78

[5] Ganger G, Ganger G, Worthington B, et al. The DiskSim simulation environment (v4. 0) [EB/OL]. http://www.pdl.cmu.edu/DiskSim，2009-9

[6] Hitachi. Ultrastar 36Z15 Datasheet [EB/OL]. http://www.hitachigst.com/hdd/ultra/ul36z15.htm，2003-01

[7] Carrera E V, Pinheiro E, Bianchini R. Conserving Disk Energy in Network Servers [C]// Proc. 17th Int'l Conf. Supercomputing. New York, USA：ACM，2003：86-97

[8] Gurumurthi S, Sivasubramaniam A, Kandemir M, et al. DRPM：Dynamic speed control for power management in server class disks [C]// Proc. 30th Annual Int'l Symp. Computer Architecture. New York, USA：ACM，2003：169-179

[9] Zhu Q, Chen Z, Tan L, et al. Hibernator：Helping Disk Arrays Sleep through the Winter [C]// ACM Symp. Operating Systems Principles. New York, USA：ACM，2005：177-190

[10] Kgil T, Roberts D, Mudge T. Improving NAND Flash Based Disk Caches[C]// Proceedings of the 35th International Symposium on Computer Architecture(ISCA'08). June 2008：327-338

[11] Zhu Q, Zhou Y. Power-Aware Storage Cache Management [J]. IEEE Transactions on Computers，2005，54(5)：587-602

[12] Li D, Gu P, Cai H, et al. EERAID：Energy Efficient Redundant and Inexpensive Disk Array [C]// Proc. of the 11th workshop on ACM SIGOPS European workshop. New York, USA：ACM，2004：1-14

[13] Wang J, Zhu H, Li D. eRAID：Conserving Energy in Conventional Disk-Based RAID System [J]. IEEE Transactions on Computers，2008，57(3)：359-374

[14] Weddle C, Oldham M, Qian J, et al. PARAID：A gear-shifting power-aware RAID [J]. ACM Transactions on Storage，2007，3 (3)：245-260