基于智能分组策略的XML关键字查询算法

张 永[1,2] 李泉霖[1] 刘 博[1]

(辽宁师范大学计算机与信息技术学院 大连 116081)[1]

(计算机软件新技术国家重点实验室(南京大学) 南京 210023)[2]

摘 要 XML关键字查询作为一种信息检索方式,一直是相关领域的热点研究问题。在经典查询语义SLCA的基础上,设计并实现了一种基于智能分组策略的XML关键字查询的优化算法。提出的算法通过合理的分组策略可以保证在运算过程中及时去除组内祖先节点和重复节点,减少了大量冗余计算,提高了算法的效率。最后设计多组实验在不同的XML数据上进行测试,实验结果表明了该算法的有效性和高效性。

关键词 扩展标记语言,关键字查询,智能分组,SLCA

中图法分类号 TP181 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2016.10.044

XML Keyword Search Algorithm Based on Intelligent Grouping Strategy

ZHANG Yong[1,2] LI Quan-lin[1] LIU Bo[1]

(College of Computer and Information Technology, Liaoning Normal University, Dalian 116081, China)[1]

(State Key Laboratory for Novel Software Technology at Nanjing University, Nanjing 210023, China)[2]

Abstract As an information retrieval method, the XML keyword search has been a hot issue in the related fields. On the base of the classical query semantic SLCA, an XML keyword search algorithm based on intelligent grouping strategy was designed and realized in this paper. With reasonable grouping strategy, the proposed algorithm can ensure that the ancestor nodes and repeat nodes are removed in time in the operation process, reducing the redundancy calculation and improving the efficiency of the algorithm. Finally, experiments on different XML data were designed. The results show the effectiveness and efficiency of the proposed algorithm.

Keywords Extensible markup language, Keyword search, Intelligent grouping, SLCA

1 引言

XML已经成为Web上数据交换的标准,Web上的海量数据以XML的形式进行保存和传输,如何高效准确存储和查找XML数据是目前的一个研究热点。XML的广泛应用使得从XML数据中提取信息成为一个值得研究的问题,为了从XML数据中获取所需要的信息,用户可以使用两种查询机制:1)相应的结构化查询语言,如XPath,XQuery等,但是伴随的缺点就是复杂的语法不利于大量用户的使用;2)通过关键字搜索的方式获取信息。

针对XML关键字搜索,最主要的研究方向就是最小子树根节点问题(SLCA)[1-3]及其一系列变体查询语义,如VLCA[4,5],ELCA[6,11]等,前人对此做了许多相关的研究。Xu等[1]基于SLCA语义提出了3种经典算法,即基于索引的搜索算法(Indexed Lookup Eager, ILE)、基于扫描的搜索算法(Scan Eager, SE)以及基于堆栈的搜索算法(Stack),并通过实验验证了ILE具有相对更好的表现;Sun等[2]提出了相对于ILE算法的Incremental Multiway-SLCA(IMS)算法,基于锚节点跳过一些冗余的LCA计算来提高算法的效率;Kong等[3]基于SLCA节点按层分布的规律,采取逐层求解的思路提出了LISA I算法,以及将前缀统一映射成整数后再参与计算的LISA II算法;Li等人根据VLCA语义和MDC(Meaning Dewey Code),提出了基于堆栈的VLCAStack算法[4];Liu等人基于VLCA语义,分析XML数据结构与关键字匹配模型,并设计算法返回生成结果[5];Xu Y等针对SLCA返回结果的不足提出了新的语义ELCA(Exclusive LCA),同时基于堆栈结构提出了IS(Index Stack)算法[6];Bao等基于SLCA语义,结合XReal与XSeek,提出了基于相关导向顺序的XML搜索,并对搜索返回的结果进行质量分析[8];Chen等人提出了一种Join-Based算法,并结合top-K算法返回XML关键字查找中的前K个结果[9];Zhou等人基于交叉集合操作提出了FastSLCA和FastELCA的计算过程,并提出了FwdSCLA算法和BwdSLCA算法,生成SLCA和ELCA[10];Zhao等利用极限学习机方法对概率XML文档进行节点分类,用于后续XML关键字查找操作[13];Stefan等基于XML数据DAG压缩技术提出了一种新的关键字查找算法,通过集合的交叉搜索加快了在较大的数据上的查找速度[14];Lin等人将NOT语义引入到关键字查找中并提出了valid SLCA的概念,将其作为查询结果返回[15];Aggeliki等在前人基础上基于SLCA和ELCA语义针对树型结构的数据提出了top-k-size stack-based算法,返回k个最优结果[16]。

本文基于ILE算法和IMS算法的思想,提出了一种对查

到稿日期:2015-08-29 返修日期:2015-12-23 本文受国家自然科学基金面上项目(61373127),辽宁省教育厅基金项目(L2011186)资助。

张 永(1975-),男,博士,副教授,CCF会员,主要研究方向为机器学习、智能计算,E-mail:zhyong@lnnu.edu.cn(通信作者);李泉霖(1990-),男,硕士生,主要研究方向为人工智能;刘 博(1990-),女,硕士生,主要研究方向为机器学习。

找关键字的 Dewey 码集智能分组的策略，并提出了相应的优化算法 IILE(Intelligent Indexed Lookup Eager)。IILE 算法通过合理的分组策略可以保证在运算过程中去除组内祖先节点和重复节点，减少了大量冗余计算，提高了算法的效率。最后设计多组实验在不同的 XML 数据上进行测试，实验结果验证了该算法的有效性和高效性。

本文第 2 节主要介绍了一些相关的基础性工作，包括 XML 文件的解析、Dewey 码与 SLCA 语义的定义等；第 3 节就 IILE 算法给出具体的实现步骤和伪代码；第 4 节设计实验并给出了实验的过程与结果；最后总结全文并讨论了未来的研究方向。

2 相关知识

2.1 XML 数据与 XML 文档树

XML 本身并不是一种编程语言，而是一套定义语义标记的规则，这些标记将文档分成许多部件并对这些部件加以标识。

为了便于 XML 数据的处理，在研究中通常将其映射成树结构，即 XML 文档树，如图 1 所示。

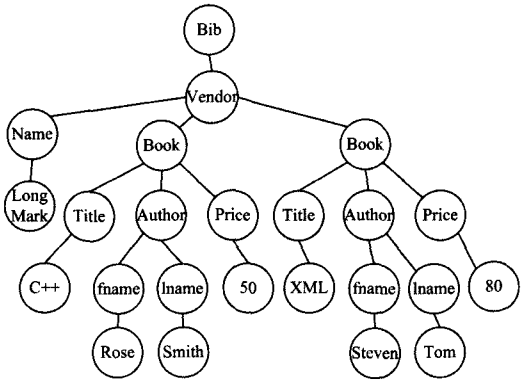

图 1 bib. xml 文档树结构

2.2 Dewey 编码

在关键字查找过程中，为了唯一标识每一个节点，采用 Dewey 编码方式。Dewey 编码是一种基于路径的编码方法，即父节点的编码是孩子节点编码的前缀，若一个节点的编码是另一个编码的前缀，则该节点一定是另一个节点的祖先节点。

Dewey 码的编码规则[3]：1)根节点的 Dewey 编码为 0；2)如果节点 v 是节点 u 的第 i 个孩子，节点 u 的 Dewey 编码为 $D(u)$，则节点 v 的 Dewey 编码为 $D(v)=D(u)\cdot(i-1)$。

例 1 如图 1 所示，根据编码规则，根节点 bib 的 Dewey 码为 0，其孩子节点 vendor 为 0.0，同理 name 为 0.0.0，C++为 0.0.1.0.0，XML 为 0.0.2.0.0。

Dewey 码的基本运算如下[3]。

性质 1 求公共前缀：表示为 $lca(v_1,v_2)$。返回 v_1 与 v_2 的最长公共前缀。若其中一个节点为空，则 $lca(v_1,v_2)$为空。

性质 2 大小判断。1)如果 v 是 u 的祖先，则称 u 大于 v ($u>v$)；2)从左至右顺序比较 u，v 的 Dewey 码各层的整数，仅在第 i 层时二者不同，整数大者的 Dewey 码大。

性质 3 右匹配与左匹配：$rm(v,S)$，$lm(v,S)$。$rm(v,S)$即节点 v 在某节点集 S 中的右匹配，在 S 中找出小于或等于 v 的最大节点，若在 S 中没有这样的节点则返回空，节点之间的大小比较即为节点之间的 Dewey 码比较。同理，$lm(v,S)$即节点 v 在某节点集 S 中的左匹配，在 S 中找出大于或等于 v 的最小节点，若在 S 中没有这样的节点则返回空。

2.3 XML 文档解析

SLCA 求解前，需要对 XML 数据进行解析，并生成对应的 Dewey 码。目前普遍使用两种技术：1)文档对象模型：DOM(Document Object Model)；2)用于 XML 的简单 API：SAX(Simple API for XML)。

DOM 这种方法不适用于大规模的 XML 文档，因此采用 SAX 方法来进行解析。

2.4 SLCA

2.4.1 SLCA 语义

SLCA 是 XML 关键字查询的重要概念，也是一种常用的查询语义。它主要定义对于给定的关键字查询，返回什么样的结果才是有意义的。这也是关键字查询研究的一个核心问题[1-3,8,11,12]。SLCA 的具体定义为求解满足以下两个条件的子树的根节点：

(1)该子树包含所有关键字序列。

(2)该子树中不包含更小的子树同样包含所有关键字序列。

XML 关键字查询首先在各个关键字的候选点集中找到关键字间的祖先关系。XML 关键字查询工作大多基于优化候选点集的生成和降低 SLCA 计算两部分。SLCA 作为 XML 关键字查询的返回结果，反映候选点集之间的包含关系。

2.4.2 SLCA 求解

求解 SLCA 的经典算法主要包括：Indexed Lookup Eager (ILE)，Scan Eager (SE) 以及 Incremental Multiway-SLCA (IMS)。

ILE 适用于关键字集中包含有低频关键字的情况，基本思路主要是设计一种 B+树结构的数据格式，用于方便地计算 lm 与 rm 操作，同时获取给定的关键字所对应的所有 Dewey 码集合，并给出公式进行说明。

$slca(\{v\},S)=\{descendant(lca(v,lm(v,S)),lca(v,rm(v,S)))\}$表示从集合中求取单个 Dewey 码的 SLCA 求解过程。而 $slca(\{v\},S_2,\cdots,S_k)=slca(slca(\{v\},S_2,\cdots,S_{k-1}),S_k)$与 $slca(S_1,\cdots,S_k)=slca(slca(S_1,\cdots,S_{k-1}),S_k)$则表示 ILE 中多个集合的求解过程[1]。

SE 则是 ILE 的一个变种，适用于所有关键字的频率波动不大的情况。这两种算法都是基于相同的规则来求解 k 个关键字的 SLCA，在计算过程中都需要 $k-1$ 个中间变量集，每一次 SLCA 的计算都需要一对(两个)节点集作为输入，然后产生一个中间结果集。

IMS 可以说是基于 ILE 算法的一个质的提升。当遇到如图 2 所示的情况，根据 ILE 规则求解 $slca(a,b)$，“元素 1”这个结果会被求解 10 次，实际上只要有一次求解就可以，同样的情况也发生在求解“元素 2”、“元素 3”等结果集上。其中很多计算都是多余的，正是针对这一点，IMS 提出了一种基于锚点的计算策略，根据锚点对参加计算的节点集进行选择，从而最大化地去除多余的计算。

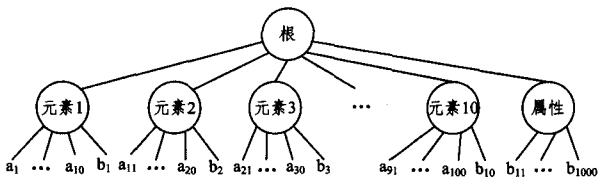

图 2 树型 XML 文档(相同关键字用下标区分)

这 3 个算法都经过巧妙的设计，将 SLCA 求解的时间复杂度降低到了 $O(|S_1|\sum_{i=2}^{k}d\log|S_i|+|S_1|^2)$[1,2]。

尽管 ILE 算法性能优越，但还是存在一些不足：1）它是将 XML 数据保存在 B+树结构上，必须修改 B+树结构以支持必要的 Dewey 码操作，实现上比较复杂。而且 B+索引结构不适用于 Dewey 编码数据[3]。2）ILE 算法中 SLCA 计算过程是“阻塞”的，即 Dewey 集 S_i 必须依次计算，也即在处理第 i 个关键字之前，必须先处理完前 $i-1$ 个关键字[1]，这大大降低了算法速度。

3 提出的 IILE 算法

3.1 基于 ILE 的优化与提升

针对前面提到的 ILE 算法中存在的问题，在此提出了 ILE 算法的改进算法——Intelligent Indexed Lookup Eager (IILE)来实现对计算过程的优化。下面将从 3 个方面阐述 ILE 算法的优化与提升。

（1）对于 k 个关键字的查询，ILE 算法的时间复杂度为 $O(|S_1|\sum_{i=2}^{k}d\log|S_i|+|S_1|^2)$，其中 S_1 为关键字集中频率最低的关键字，S 是频率最高的关键字。可见算法的效率受 S_i 的影响很大，而通过降低 S_i（$i=1$ to k）的大小，即可在 SLCA 计算前降低计算开销。

（2）将 S_1 集合拆分成 $m=\lceil|S_1|/|P|\rceil$ 份子集（组）[1]，其中 P 为大小固定的缓冲区，子集用 B_j（$1\leqslant j<m$）表示，从而达到通过求解 $slca(B,S_2,\cdots,S_k)$ 更快呈现 SLCA 结果的目的。为此给出了第一个属性。

属性 1：$slca(S_1,S_2)=removeAncestor(slca(B_1,S_2)\cup\cdots\cup slca(B_j,S_2)\cup\cdots\cup slca(B_m,S_2))$，其中 *removeAncestor* 表示去除结果节点集中的祖先节点，B_j 满足 $S_1=\bigcup_{m}^{1}B_j$，且 $B_j\cap B_{j+1}=\varnothing$（$1\leqslant j<m$）。

说明：假设 $slca(S_1,S_2)\neq removeAncestor(slca(B_1,S_2)\cup\cdots\cup slca(B_j,S_2)\cup\cdots\cup slca(B_m,S_2))$，则一定存在一个 B_j，使得 $slca(B_j,S_2)\not\subseteq slca(S_1,S_2)$。由于 $S_1=\bigcup_{m}^{1}B_j$ 且 $B_j\cap B_{j+1}=\varnothing$（$1\leqslant j<m$），因此 $B_j\subset S_1$，则 $slca(B_j,S_2)\not\subseteq slca(S_1,S_2)$ 不成立，故 $slca(S_1,S_2)=removeAncestor(slca(B_1,S_2)\cup\cdots\cup slca(B_j,S_2)\cup\cdots\cup slca(B_m,S_2))$ 是正确的。

（3）在 ILE，SE 和 IMS 算法中，都是通过间接或直接去除 SLCAs 候选集中的祖先节点来得到 SLCA，而实际上生成中间 SLCA 候选集 SLCAs 是大量计算的过程。例 2 更加直观地说明了这个问题。

例 2　设 w_1 和 w_2 两个关键字对应的 Dewey 集分别为 $S_1=\{0.0.0,0.1.1.0.1,0.1.1.1.0,0.1.1.2.1.0,0.1.1.3.0,0.1.2.0.0,0.2.0.0.0,0.2.1.0.0,0.2.2.0.0\}$，$S_2=\{0.0.1.5,0.1.0.9.0,0.1.1.2.0,0.1.2.1.0,0.2.0.0.1,0.3.0.0.0,0.3.1.0.0,0.3.2.0.0\}$。

根据 ILE 算法，求关键字 w_1，w_2（分别对应节点集 S_1，S_2）的 SLCA，取 P 大小为 2，按照$\lceil|S_1|/|P|\rceil$原则，S_1 被平均分成 5 组（即 $m=5$），分别为 $B_1=\{0.0.0,0.1.1.0.1\}$，$B_2=\{0.1.1.1.0,0.1.1.2.1.0\}$，$B_3=\{0.1.1.3.0,0.1.2.0.0\}$，$B_4=\{0.2.0.0.0,0.2.1.0.0\}$，$B_5=\{0.2.2.0.0\}$，可得 $SLCAs=removeAncestor(slca(B_1,S_2)\cup slca(B_2,S_2)\cup slca(B_3,S_2)\cup slca(B_4,S_2))\cup slca(B_5,S_2)$，即 $SLCA=removeAncestor(\{\{0.0,0.1.1\},\{0.1.1.2\},\{0.1.1,0.1.2\},\{0.2.0.0\},\{0.2\}\})$。最后求得 $SLCA=\{0.0,0.1.1.2,0.1.2,0.2.0.0\}$。易知求出只有 4 个 SLCA 的结果集，就生成了 2 个多余的 SLCA，当 XML 文档和关键字数量很大时，去除祖先节点的任务将会非常重，而且生成这些多余的 SLCAs 更是对计算能力的大量浪费。

在 ILE，SE 和 IMS 中，由于缺乏规则对 P 的大小进行动态、合适的选取，因此只能凭感觉或者实验（针对某一特定数据集）效果来选取 P 为某一固定大小，这也正是问题所在。虽然他们也考虑到了这个问题，但也只能是从结果集上采取措施。而实际上通过研究发现，通过采取合适的手段是可以对 S_i 进行智能分组的，为此给出了第二个属性。

属性 2：智能分组原则。对于 3 个 Dewey 码（节点）v_i，v_{i+1} 和 v_{i+2}（$1\leqslant i\leqslant k-2$），如果 $lca(v_i,v_{i+1})=lca(v_{i+1},v_{i+2})$，那么 v_i，v_{i+1} 分在同一组；如果 $lca(v_i,v_{i+1})>lca(v_{i+1},v_{i+2})$，那么 v_i，v_{i+1} 分在同一组，但是 v_{i+1}，v_{i+2} 不在同一组；如果 $lca(v_i,v_{i+1})<(lca(v_{i+1},v_{i+2}))$，那么 v_i，v_{i+1} 不在一组。

说明：因为相对于同一个 Dewey 码（节点）v_{i+1}，$lca(v_i,v_{i+1})\geqslant lca(v_{i+1},v_{i+2})$，说明 v_i 比 v_{i+2} 更接近于 v_{i+1}，即相对于 v_{i+1}，v_i 比 v_{i+2} 有更长的公共前缀。那么在求解 $slca(v_n,\cdots,v_i,v_{i+1},v_{i+2},\cdots,v_k)$ 的过程中分解成第一种情况：$slca(v_n,\cdots,v_i)$ 和 $slca(v_{i+1},v_{i+2},\cdots,v_k)$ 所求解的 SLCA 候选集中所含祖先的数量一定不小于分解成第二种情况：$slca(v_n,\cdots,v_i,v_{i+1})$ 和 $slca(v_{i+2},\cdots,v_k)$ 的求解方式。v_i，v_{i+1}“靠”得很近，假设 v_i，v_{i+1} 与 S_2 中的某一节点 v 生成的 SLCA 之间存在祖先关系，那么在第一种情况，为了得到准确的 SLCA，只能在分组求解 SLCA 之后去除刚刚生成的祖先节点；而第二种情况则不需等到分解求解结束，在求解的过程中就可以判断并去除祖先节点。据此可以对例子 2 利用该分组策略重新分组求解，如例 3 所示。

例 3　根据属性 2，有 $lca(0.0.0,0.1.1.0.1)<lca(0.1.1.0.1,0.1.1.1.0)$，所以“0.0.0”和“0.1.1.0.1”不分在同一组，则有 $B_1=\{0.0.0\}$。再看 $lca(0.1.1.0.1,0.1.1.1.0)=0.1.1=lca(0.1.1.1.0,0.1.1.2.1.0)=0.1.1$，所以“0.1.1.0.1”与“0.1.1.1.0”分在一组，以此类推。同理有 $B_2=\{0.1.1.0.1,0.1.1.1.0,0.1.1.2.1.0,0.1.1.3.0\}$，$B_3=\{0.1.2.0.0\}$，$B_4=\{0.2.0.0.0,0.2.1.0.0,0.2.2.0.0\}$。最终 S_1 被动态地分成 4 组，且每组的大小不是固定的。可得 $SLCAs=removeAncestor(slca(B_1,S_2)\cup slca(B_2,S_2)\cup slca(B_3,S_2)\cup slca(B_4,S_2))$，即 $SLCA=removeAncestor(\{\{0.0\},\{0.1.1.2\},\{0.1.2\},\{0.2.0.0\}\})$。直接生成 4 个 SLCA 的结果集，没有多余的结果。针对 SLCA 中的“0.1.1.2”，并不需要在分组求得的 SLCA 候选集后进行筛选，因为在求解的过程中组内产生的“祖先关系”就会被处理，而不会浪费更多的计算资源，且根据 *lemma*1 和 *lemma*2[1] 两个引理，S_{12} 求解 SLCA 的逻辑复杂性得到简化。这在多个求解的过程中效果将会更加突出。

3.2 IILE 算法描述

根据上述的分析和给出的两个改进属性，给出了 IILE 算法的描述，如算法 1 所示。

算法 1　Intelligent Indexed Lookup Eager Algorithm

Input: the inputKeywords=$\{w_1,w_2,\cdots,w_k\}$//待查询的关键字集合

Output: the slca of the inputKeywords

1. select DeweyCodeSet S_i of the inputKeyword w_i; ($1\leqslant i\leqslant k$)
2. sort S_i based on the $|S_i|$; //S_1 is the smallest DeweyCodeSet
3. S={}; //S 表示 S_1 按照智能分组之后的结果集
4. S=IGA(S_1); //调用算法 2 进行智能分组
5. T={};
6. for (each Set in S) do
7. T=Set;

```
8.    for (i=2 to k) do
9.     T=get_slca(T,Si)[1];
10.   endfor
11.   v=null; Result={};
12.   if (v!=null&&first(Set) is ancestor of v) then//first(Set)为
      Set集中第一个Dewey码
13.     removeFirstNode(Set);
14.   endif
15.   if (v!=null&&v is not ancestor of first(Set)) then
16.     Result=Result∪{v};
17.   endif
18.   v=removeLastNode(Set);
19.   Result=Result∪Set;
20.   Set={};
21. endfor
22. Result=Result∪{v};
23. removeSame(Result);
24. removeAncestor(Result);
25. return Result;
```

在算法1的描述中，S_i 表示第 i 个关键字(w_i)的所有Dewey码的集合，$|S_i|$表示 S_i 中Dewey码的个数。S_i 根据集合中Dewey码个数由小到大排列($1\leqslant i\leqslant k$)。在第3和第4行，算法1调用了算法2中的IGA函数进行了智能分组，具体实现方法如算法2所示。第6－22行为SLCA计算阶段，每个分组分别与 S_i 进行计算($2\leqslant i\leqslant k$)。第23－25行表示对结果集进行去重复去祖先的清洗并返回。

算法2 Intelligent Grouping Algorithm (IGA)

```
Input:S1
Output:S={every Set from S1},Set is a group of S1
1. S={};
2. Set={};
3. if (|S1|<=2)
4:  {Set=S1;S=S∪{Set};}
5. else
6.  {
7.    Flag=true;
8.    for (j=1 to |S1|-2) do//S1中节点从1到|S1|编号
9.      if (Flag)
10.       {Set=Set∪{vj};Flag=false;}
11.     if (lca(vj,vj+1)=lca(vj+1,vj+2))//vj,vj+1在一组
12.       {Set=Set∪{vj+1};
13.        if (j=|S1|-2)
14.          {Set=Set∪{vj+2};S=S∪{Set};}
15.       }
16.     else if (lca(vj,vj+1)>lca(vj+1,vj+2))//vj,vj+1在一组,vj+1,
        vj+2不在一组
17.       {Set=Set∪{vj+1};S=S∪{Set};Set={};
18.        if (j=|S1|-2)
19.          {Set=Set∪{vj+2};}
20.        j++;Flag=true;
21.       }
22.     else if (lca(vj,vj+1)<lca(vj+1,vj+2))//vj,vj+1不在一组
23.       {S=S∪{ Set };Set={};
24.        if (j=|S1|-2)
25.          {Set=Set∪{vj+1},Set=Set∪{vj+2};S=S∪{Set};}
26.        Flag=true;
27.       }
28.   end for
29.  }
30. return S
```

在算法2中的第3－4行，当 S_1 中Dewey码个数过少时，不需要分组。第11－15行表示当 $lca(v_j,v_{j+1})=lca(v_{j+1},v_{j+2})$时，$v_j,v_{j+1}$在一组。第16－21行表示当 $lca(v_j,v_{j+1})>lca(v_{j+1},v_{j+2})$时，$v_j,v_{j+1}$在一组，$v_{j+1},v_{j+2}$不在一组。第22－27行表示当 $lca(v_j,v_{j+1})<lca(v_{j+1},v_{j+2})$时，$v_j,v_{j+1}$不在一组。

4 实验

本文实验结合XML关键字查询的3大因素(关键字的个数、关键字频数以及XML数据集大小(总节点数))设计了多组实验。通过实验结果对比，验证了基于智能分组策略的XML关键字查询的高效性。

所有实验均在PC上实现，机器CPU为Intel(R)Core2 Duo CPU，主频为2.94GHz，物理内存为2GB，所有算法均用Java实现，JDK版本为jdk-7u2-linux-i586。

4.1 实验数据

从数据网站上下载了4个大小不同的XML数据集用于下面的实验。下载地址：https://www.monetdb.org/Downloads。其中：数据集1为FI_meta.xml，简称data1；数据集2为FR_meta.xml，简称data2；数据集3为ES_meta.xml，简称data3；数据集4为DE_meta.xml，简称data4。4个数据集大小分别为10.7M，125.2M，168.8M，247.1M。

鉴于实验的需要，从每组数据集中根据个数和频数随机选出7组关键字，关键字的个数与频数用“－”隔开[7]，分别是4－10，4－100，4－1000，4－5000，2－10，8－10，16－10。比如，4－10表示4个关键字，每个关键字的频数为10。为了方便频数的统计，允许频数在20%左右浮动(10±20%)，即关键字出现次数为8－12的频数都为10。

4.2 实验过程

基于3大查询因素设计了两组实验。第一组在同一数据下控制关键字个数相同(设为4)，改变关键字频数，且让其呈现增长趋势(10，100，1000，5000)，然后分别对每一数据集做相同的实验；第二组在同一数据下控制关键字的频数相同(设为100)，改变关键字个数(2，4，8，16)，然后分别对每一数据集做相同的实验。

实验1　关键字个数相同(设为4)；关键字频数不相同，且呈增长趋势(10，100，1000，5000)；数据集相同。实验结果如图3所示。

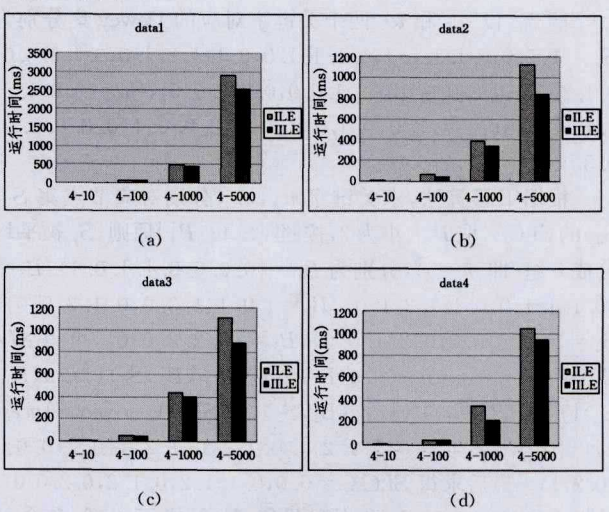

图3　实验1的实验结果

实验 1 通过保持关键字个数不变、改变关键字频数来进行实验，图 3(a)－图 3(d)分别表示 data1 到 data4 4 个不同数据集上的测试结果。从实验结果中可以看出，改进后的 IILE 算法与 ILE 相比在速度上有很大的改进，尤其体现在频数较大的情况(如 1000 和 5000)。从实验结果中可看出，当次词频增大时算法在速度上体现出很大优势，这是因为原始 ILE 算法的简单平均分组方法必然产生大量组间重复节点和祖先节点。如例 2 和例 3 所示，随着词频的增大，冗余计算操作也会大量增加，势必会影响算法的效率。而智能分组之后每一组的计算结果之间并不会有重复节点，这就避免了许多重复的计算工作。

实验 2　关键字个数不相同(2，4，8，16)；关键字频数相同(设为 100)；数据集相同。实验结果如图 4 所示。

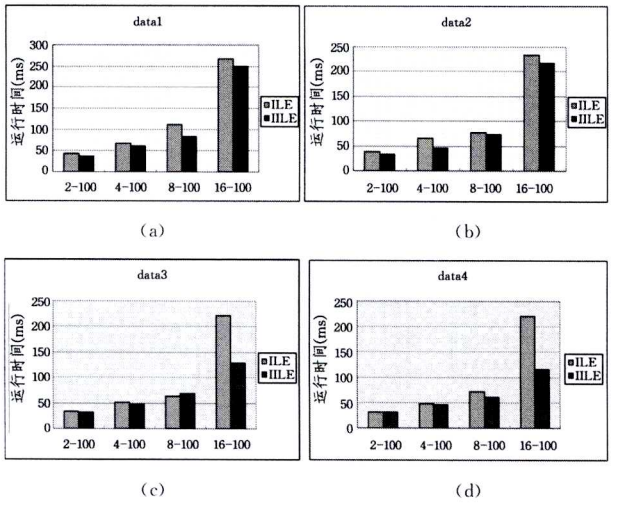

图 4　实验 2 的实验结果

实验二通过保持关键字频数不变、改变关键字个数来进行实验，图 4(a)－图 4(d)分别表示 data1 到 data4 4 个不同数据集上的测试结果。从实验结果可以看出，在频数不大(为 100)的情况下，算法的优化效果并无太大提升，如 data3 中，2 个和 4 个词频为 100 的关键字查找的效果与 ILE 基本一致，甚至 8 个词频为 100 的关键字查找的效果反而不如 ILE 算法，但是当关键字个数增加到 16 个时，IILE 算法的优势才显现出来。因为随着个数的增加，势必会引起计算量的增加，在求解的过程中组内产生的“祖先关系”就会被大量处理，而不会浪费更多的计算资源，IILE 求解 SLCA 的过程得到简化，在多个关键字求解的过程中效果将会更加突出。

4.3　实验结论

分析实验数据可以发现，改进后的 IILE 算法对算法的效率有很大的提升，根据图 3 和图 4 可知 IILE 算法对关键字频数较大和关键字个数较大的查询情况效果明显，如频数为 5000 和个数为 16 的查询情况，但是对于关键字频数小的情况(如频数为 100)，改进效果并不明显。

结束语　本文对基于 SLCA 的 XML 关键字查询算法的实现做了大量深入的研究，提出了一种基于智能分组策略的 IILE 算法并予以实现，并通过实验结果分析得出该方法可以高效地处理 XML 数据的基于 SLCA 的关键字查找。但是本文还有待改进：该算法并没有做到分组之间，产生 SLCA 之间的关系彻底分离，还是会存在一些祖先关系，造成这种问题的原因需继续分析，或将成为下一个阶段的研究方向。下一步将着重于并行化处理大规模数据，研究更加高效的查找算法，并将其应用到云计算环境中，实现大规模 XML 数据的查找。

参 考 文 献

[1] Xu Y, Papakonstantinou Y. Efficient keyword search for smallest LCAs in XML databases[C]// Proceedings of SIGMOD. 2005:537-538

[2] Sun C, Chan C Y, Goenka A K. Multiway SLCA-based keyword search in XML data[C]// Proceedings of the 16th International Conference on World Wide Web. 2007:1043-1052

[3] Kong L B, Tang S W, Yang D Q, et al. Layered Solution for SLCA Problem in XML Information Retrieval[J]. Journal of Software, 2007, 18(4):919-932(in Chinese)
孔令波，唐世渭，杨冬青，等. XML 信息检索中最小子树根节点问题的分层算法[J]. 软件学报，2007，18(4)：919-932

[4] Li G L, Feng J H, Wang J Y, et al. Effective keyword search for valuable LCAs over XML documents[C]// Proceedings of the sixteenth ACM Conference on Information and Knowledge Management. ACM Press, 2007:31-40

[5] Liu Z, Chen Y. Identifying meaning return information for XML keyword search[C]// Proceedings of SIGMOD. 2007:329-340

[6] Xu Y, Papakonstantinou Y. Efficient LCA based keyword search in XML data[C]// Proceedings of EDBT. 2008:35-546

[7] Zhang C, Ma Q, Wang X, et al. Distributed SLCA-based XML keyword search by Map-Reduce[M]// Database Systems for Advanced Applications. Springer Berlin Heidelberg, 2010: 386-397

[8] Bao Z, Lu J, Ling T W, et al. Towards an effective XML keyword search[J]. IEEE Transactions on Knowledge and Data Engineering, 2010, 22(8):1077-1092

[9] Chen L J, Papakonstantinou Y. Supporting top-k keyword search in xml databases[C]// IEEE 26th International Conference on Data Engineering (ICDE). 2010:689-700

[10] Zhou J, Bao Z, Wang W, et al. Fast SLCA and ELCA computation for XML keyword queries based on set intersection [C]// IEEE 28th International Conference on Data Engineering (ICDE). 2012:905-916

[11] Li J, Liu C, Zhou R, et al. Quasi-SLCA based keyword query processing over probabilistic XML data[J]. IEEE Transactions on Knowledge and Data Engineering, 2014, 26(4):957-969

[12] Aksoy C, Dimitriou A, Theodoratos D, et al. XReason: A semantic approach that reasons with patterns to answer XML keyword queries[M]// Database Systems for Advanced Applications. Springer Berlin Heidelberg, 2013:299-314

[13] Zhao Y, Yuan Y, Wang G. Keyword search over probabilistic XML documents based on node classification[J]. Mathematical Problems in Engineering, 2015:135-144

[14] Böttcher S, Hartel R, Rabe J. Efficient XML keyword search based on DAG-compression[M]// Database and Expert Systems Applications. Springer International Publishing, 2014:122-137

[15] Lin R R, Chang Y H, Chao K M. Locating Valid SLCAs for XML keyword search with NOT semantics[J]. ACM SIGMOD Record, 2014, 43(2):29-34

[16] Dimitriou A, Theodoratos D, Sellis T. Top-k-size keyword search on tree structured data[J]. Information Systems, 2015, 47:178-193