基于式样单划分的XML数据并行转换方法研究

李 宁 高晓光 侯 霞 张 伟 田英爱
(北京信息科技大学计算机学院 北京 100101) (网络文化与数字传播北京市重点实验室 北京 100101)

摘 要 在现有研究的基础上提出了一种基于式样单(stylesheet)划分的XML数据并行转换方法,并针对该方法所涉及的以下关键问题进行了讨论:1)如何从式样单中获得多个相互独立的转换单元,它们可以并行执行而互不影响;2)如何根据运行环境自动调整转换任务的数量和负载;3)如何将多个并行转换得到的结果进行有效的合并。将提出的算法应用于实际的Open XML-UOF文档格式转换项目,取得了很好的效果。该方法对于在并行环境下有效地提高XML的数据转换性能具有较大的应用价值。

关键词 XSLT,式样单,XML转换,并行处理,XML
中图法分类号 TP311 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2016.3.042

Research on Parallel Transform of XML Data Based on XSLT Stylesheet Partitioning

LI Ning GAO Xiao-guang HOU Xia ZHANG Wei TIAN Ying-ai
(School of Computer Science, Beijing Information Science and Technology University, Beijing 100101, China)
(Beijing Key Laboratory of Internet Culture and Digital Dissemination Research, Beijing 100101, China)

Abstract Based on the exploration of current research of XSLT acceleration, a new method to transform XML data in parallel mode based on XSLT Stylesheet partitioning was proposed. The related topics were discussed including: 1) how to obtain the independent transform units which can be executed concurrently without affecting each other, 2) what is the proper number of parallel tasks according to the underlying platform being used and how to balance their workload, 3) how to merge the individual intermediate results correctly. The proposed method was applied into the real project, i.e. the Open XML-UOF document format translation project, achieving fairly good result. This research is significant to the improvement of XML transform performance under parallel processing environment.

Keywords XSLT, Stylesheet, XML transform, Parallel processing, XML

1 背景

XSLT(eXtensible Stylesheet Language Transformation)是XML数据转换、呈现和查询处理中的关键技术。随着大数据时代的来临、数据规模和复杂度的增加,XSLT的转换性能引起了人们高度的重视,多种提高XSLT转换效率的方法被提出。如Dong和Bailey提出的XSLT式样单静态分析方法可以滤掉实际转换中用不到的模板[1];Wong和Gertz提出的对XPath表达式的优化方法试图借助DTD来提高XPath的查询效率[2];Dong和Bailey提出模板实例化方法,用以节省XSLT解析器匹配模板的时间[3];Mizumoto为了克服集中转换XML数据效率不高的问题,提出在分布式环境下对XML数据进行转换的方法[4],即在存放XML片段的不同地点进行数据转换,然后再将转换结果集中到一个特定地点,其重点是研究路径预算和缓存算法来减少不同地点间模板匹配的次数,降低通信的开销。国内杭州电子科技大学的学者陈如昌从XSLT样式表元素间关联关系的角度出发,提出了一种以样式表关联图为中心、面向特定转换过程的优化方法,以提高XSLT的转换效率[5];北京大学的高军等提出了一种基于DTD的XPath逻辑优化方法,其采用一种新的XPath树自动机和DTD树自动机的乘积运算,提高了XPath处理器的执行效率[6]。

上述方法主要针对特定的问题或特殊的转换场景,而未考虑一般情况下对XSLT更有针对性的加速方法。XPath查询优化方法主要是改进复杂XPath表达式所导致的冗余路径查询的效率问题,较大程度地提高了搜索源数据的效率,但这种方法偏向XPath表达式的优化。陈如昌的方法是面向特定转换过程的,当XML源数据的覆盖率在70%以上时,这种方法可能反而导致转换效率的降低。Dong和Bailey的方法主要针对服务器端大量数据的交换和查询,可在一定程度上减少运行时间,但是静态分析仍需要花费一些时间。

为了更好地提高XSLT的转换性能,Chen和Chu提出了利用多线程环境来提高XSLT转换效率的方法[7],其基本原理是将式样单中可并行的部分划分出来放到多线程转换模型

到稿日期:2015-02-15 返修日期:2015-05-26 本文受北京市属高等学校创新团队建设与教师职业发展计划项目(IDHT20130519)资助。
李 宁(1964-),男,博士,教授,主要研究方向为置标语言、文档信息处理,E-mail:ningli.ok@163.com;高晓光(1987-),男,硕士,工程师,主要研究方向为文档信息处理、计算机软件;侯 霞(1976-),女,博士,副教授,主要研究方向为置标语言、软件工程;张 伟(1980-),男,博士,副教授,主要研究方向为计算机系统结构;田英爱(1975-),女,博士生,副教授,主要研究方向为文档信息处理。

中作并行转换，从而达到缩短总转换时间的目的。但他们并未给出如何分解 XSLT 文档以及如何合并结果的算法。Sun 提出了在多核处理器中进行并行 XML 转换的方法[8]，该方法设计了 4 种并行转换模型，将转换任务映射到这些模型实现并行转换，还探讨了并行的颗粒度问题。该方法依赖 Intel 的 Napa XSLT 引擎，且需要较为复杂的编译过程，其用于控制 XSLT 构件并行分解和合并的思想值得借鉴，虽然并行颗粒度仍基于经验和观测。Li 提出一种基于 MapReduce 架构的 XSLT 分布式处理框架[9]，通过设计面向 MapReduce 的 XML 和 XSLT 的表示模型和算法，实现 XSLT 的并行处理。但经实验发现，在执行小规模转换任务时，MapReduce 花费在结果收集上的开销过大，效果并不理想。

为了发挥一般的多核处理器在 XSLT 并行转换中的作用，采用一般的 XSLT 处理程序，不需依赖特定的并行环境，北京信息科技大学 2010 年以来开展了基于式样单划分的 XML 数据并行转换方法的研究。罗文甜提出了一种以 XSLT 模板为最小划分单元的并行转换模型[10]，虽然效果较好，但是该方法中式样单的划分需要借助经验并通过手工完成。本文在其基础上，提出了一种基于 XSLT 的 XML 数据自动化并行转换方法，包括式样单的自动划分以及分支转换结果的自动合并。模板的划分可以根据运行环境和训练数据进行动态调整。实验结果表明，本文提出的方法有较好的可行性，能够大幅度提高转换的速度。

2 原理

XSLT 是一种基于规则的转换语言，能将 XML 数据从一种结构转换为另一种结构，被广泛地应用于不同平台或不同应用间的数据转换[11]。XSLT 通过式样单定义一组转换规则，根据数据源中的预定模式调用相应的规则产生目标数据。式样单中的每条规则表达为一个模板。XSLT 因为以下原因比较适合并行处理：1）XSLT 是一种声明性语言，其模式匹配和模板调用规则与输入节点无关；2）XSLT 中的变量仅允许一次性赋值，这保证了 XSLT 的指令互不依赖，并行分解时不需要做复杂的数据流分析。

XSLT 并行转换的原理是将单一的式样单拆分为多个可并行执行的部分，将它们安排在并行模型中运行，利用并行处理减少转换的总时间，从而达到提高性能的目的。XSLT 并行转换的难点是：1）多层次嵌套的式样单如何映射到不同的并行转换任务中去？2）如何同步并行转换的结果，使之能够还原应有的顺序？3）如何控制并行分解的粒度？一方面要避免并行任务分解过细而超过最大的线程数，或导致并发管理的开销过高；另一方面要避免并行任务分解不完全，不能充分发挥处理器的效能。

XSLT 并行转换的关键是要有一个适合的并行转换模型，使 XSLT 式样单能够按照模型的要求进行并行任务分解，以及将各个并行转换任务的结果进行合并。图 1 为本文提出的并行转换模型的示意图。其中每一个转换线程的任务都是利用一个式样单转换源 XML 数据。每个转换线程所用式样单都是原始式样单中的一部分，转换后的结果也是目标 XML 数据中的一部分，最后这些中间转换结果需要按照一定的规则合并成为目标 XML 数据。

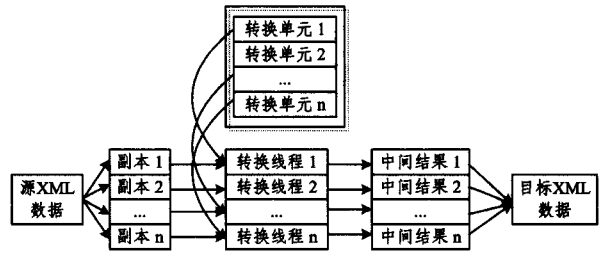

图 1 XSLT 并行转换模型

利用多线程来进行并行计算，并非线程的数量越多越好。随着线程数量的增多，用于线程启停、维护和切换的开销也将增大，当线程数达到一个阈值后，系统的性能不升反降。根据实验，最大线程数 TC 为 CPU 核数的 2 倍时可获得最佳的计算性能。假设式样单划分得到的最终转换单元数为 n，应满足 $n \leqslant TC$。

3 式样单自动划分方法

3.1 概念及原理

下面说明式样单的自动划分原理。

定义 1（丛林） 由多棵树构成的集合。

XSLT 式样单的一般形式如下。

```
〈?xml version="1.0" encoding="UTF-8"?〉
〈xsl:stylesheet version="1.1" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"〉
    〈xsl:template name="…" match="…"〉
        [b1]
        〈xsl:call-template name="t1"/〉
        [b2]
        〈xsl:call-template name="t2"/〉
        …
        [bn]
        〈xsl:call-template name="tn"/〉
        [bn+1]
    〈/xsl:template〉
〈/xsl:stylesheet〉
```

这里 $b_1, b_2, \cdots, b_{n+1}$ 为丛林。在模板 $t_1, t_2, \cdots, t_n$ 中可能会嵌套模板，从而形成子模板的嵌套层次。

假设模板 $t_1, t_2, \cdots, t_n$ 产生的转换结果为 $r_1, r_2, \cdots, r_n$，这里 $r_1, r_2, \cdots, r_n$ 也是丛林。按照 XSLT 规范，最终的转换结果为：$b_1, r_1, b_2, r_2, \cdots, b_n, r_n, b_{n+1}$。

假设模板 t_i 存在子模板 $t_{i1}, t_{i2}, \cdots, t_{im}$，这些子模板的转换结果为：$r_{i1}, r_{i2}, \cdots, r_{im}$，最终的转换结果为：$b_1, r_1, b_2, r_2, \cdots b_{i-1}, b_{i1}, r_{i1}, b_{i2}, r_{i2}, \cdots, b_{im}, r_{im}, b_{im+1}, \cdots, b_n, r_n, b_{n+1}$。

由此，作为任何一个子模板 t_i，如果能够记住从顶层模板起始到本层模板为止的各个丛林构成的路径，即（$b_1, b_2, \cdots, b_{i-1}$），就可以将该模板的转换结果填入适当的位置。当填充完所有模板的中间转换结果后，便可以得到最终的转换结果。因此，在划分模板时，需要记住这些位置信息，以供将来合并转换结果时使用。

定义 2（结果路径） 转换单元的转换结果在最终结果中的位置信息。

定义 3(转换树) 转换树是根据式样单得到的一棵树，它有一个根节点，同层模板之间的丛林作为同层节点。模板调用节点(xsl:apply-templates 或 xsl:call-template)为特殊的节点，该节点的后代是被调用的模板的转换树。

定义 4(转换分支与划分节点) 转换树中完成部分转换任务的一个分支称为一个转换分支。此分支的根节点称为该转换分支的划分节点。

定义 5(可划分节点) 式样单中可以拆分并行转换任务的节点。目前包括拥有多个子节点的非 XSLT 节点以及 XSLT 的 xsl:call-template 节点。此外，对于 xsl:apply-template 节点，如果仅有一个确定的模板与其 pattern 匹配，其也是可划分节点。

定义 6(转换单元) 一个独立的转换任务。可以形式化地描述为一个四元组 $S=\langle T,P,C,R\rangle$。其中：

1)T 为式样单。该式样单包含：a)转换分支划分节点的所有祖先节点；b)这些节点的 xsl:variable 或 xsl:param 兄弟节点；c)划分节点的所有后代节点。

2)P 为结果路径。它是一个集合，其中每个元素是从根节点到划分节点之间的各节点在转换树中的位置序号。由于除 xsl:element 外的其他 XSLT 指令节点不会在结果树中出现，因此集合中需去掉除 xsl:element 外的所有 XSLT 指令节点。位置序号的编号方式为：每层节点按从左到右的顺序从 1 开始递增。

3)C 为 S 的转换代价。由于 XSLT 不同于一般的编程语言，不能利用类似计算复杂度[11]的方法来估计转换单元的运行时间，这里是通过测量训练数据的转换时间来估算的。可以在待转换的源 XML 中随机抽取一部分数据作为训练数据。

4)R 为本转换单元的转换结果。转换之前 R 为空。

可以根据需要，按不同层次划分子模板形成转换单元。

定义 7(原子转换单元) 不存在可划分节点的转换单元。

3.2 转换单元划分规则

假设被划分的转换单元包含的转换分支如图 2 所示，其中 N_i 节点为其划分节点。

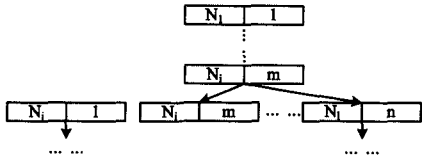

图 2 待划分的转换单元所包含的转换分支

图中省略了 N_1 到 N_i 节点之间经过的路径节点，图中的转换单元 $S=\langle T,P,C,R\rangle$ 的划分步骤如下：

1)输入待划分的转换单元 S，从根开始依次判断每个节点是否为可划分节点。如果是，则执行步骤 2)；否则执行步骤 6)。

2)计算可划分节点 N_i(设编号为 m)的子节点个数 n(不包括 xsl:variable 和 xsl:param 节点)。当 $n=1$ 时，子节点作为唯一子转换单元的划分节点，结果路径 $P=\{1,\cdots,m\}$，执行步骤 1)；当 $n>1$ 时，转至步骤 3)。

3)将转换单元划分为 n 个子转换单元：$S_1=\langle T_1,\{1,\cdots,m,1\},C_1,R_1\rangle$，$S_2=\langle T_2,\{1,\cdots,m,2\},C_2,R_2\rangle$，$\cdots$，$S_n=\langle T_n,\{1,\cdots,m,n\},C_n,R_n\rangle$，其中 $N_j,N_k,\cdots,N_l$ 节点分别为子转换单元的划分节点。

4)按照转换分支，n 个子转换单元分别从转换树中获得应包含的所有节点，形成对应的式样单 $T_1,T_2,\cdots,T_n$。

5)输出转换单元集合。

6)使用每个子转换单元对应的式样单分别转换训练数据，并记录转换时间 C。

3.3 转换单元的动态调整

为了使并行转换的整体性能最优，应当尽量平衡各转换单元的负载。式样单并行划分算法应该使划分和合并达到某种动态平衡。下面介绍相应的算法。

设转换单元集合为 $SS=\{S_i\mid i=0,1,\cdots,n-1\}$，$n$ 为转换单元的数量。

初始情况下，式样单未被拆分，即只存在一个转换单元。设 $SS=\{S_0\}$。当划分转换单元 S_0 后得到转换单元集合 $SS'=\{S_i\mid i=1,2,\cdots,n\}$，从 SS 中去掉 S_0，并将 SS' 并入 SS。SS 中各个转换单元 C 值的平均值为 T_{avg}。T_{avg} 的计算方法如下：

$$T_{avg}=\frac{\sum_{i=1}^{n}T_i}{n} \tag{1}$$

确定波动范围 $A=\{T_{avg}*(1-\eta),T_{avg}*(1+\eta)\}$，其中 η 的取值范围是(0,1)，这里取 0.2。依次判断 SS 中各转换单元的 C 值是否都落在波动范围内，判断步骤如下：

1)由 SS 得到转换单元数量 n，判断是否大于 TC。如果是，则执行步骤 4)；否则，执行步骤 2)。

2)判断 SS 中是否存在 C 值超出波动范围的转换单元。如果不存在，则执行步骤 3)；否则，选出 C 值最大的转换单元 S_j，并将其再次划分。若 S_j 为非原子转换单元，划分后可得到转换单元集合 SS''，在 SS 中去掉单元 S_j，将 SS'' 并入 SS，计算 T_{avg} 和波动范围 A；若 S_j 是原子转换单元，则 $T_{avg}=T_j$，计算波动范围 A。执行步骤 1)。

3)判断 SS 中是否存在 C 值小于波动范围 A 的转换单元。如果存在，则转步骤 4)；否则，转步骤 5)。

4)在 SS 中选出 C 值最小的两个转换单元，假设为 S_1 和 S_2，将其合并为 S_{1_2}(复合转换单元)，并在 SS 中去掉 S_1 和 S_2，填入转换单元 S_{1_2}。执行步骤 1)。

5)输出 SS 中的转换单元集合。

当动态调整过程结束后，SS 中的各个转换单元的转换代价基本均衡，最终形成的各个转换单元将分别映射到各个转换线程中进行并行转换。

4 转换结果的合并

当并行转换结束后，会得到多个中间转换结果，其数量等于转换单元的数量。要得到最终的转换结果，需要对这些中间转换结果进行有序的合并。假设转换单元集合 SS 包含 n 个转换单元，合并的步骤如下：

1)对转换结果进行排序，使得中间结果可以按增量的方式累加到最终结果中。假设各个转换单元的结果路径集合中

最多的元素个数为 Z，将不足 Z 的路径集合的其他元素填补为 0。重排 SS 中的元素，使得：

$$\sum_{j=1}^{Z} k_{i,j} 10^{Z-j} < \sum_{j=1}^{Z} k_{i+1,j} 10^{Z-j} \quad (2)$$

其中 k 为顺序。

2)每个排好序的转换单元的中间转换结果称为合并单元。将 R_1 作为基础合并单元 BR_1，其他合并单元经处理后依次合并进来。假设合并操作为 Mer，D 为对被合并的单元进行的处理，结果合并过程可表示为：

$$BR_i = Mer(BR_{i-1}, D(R_i)) \quad (3)$$

中间结果的合并模型如图 3 所示。

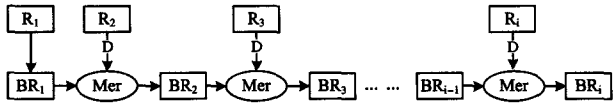

图 3　中间结果的合并模型

假设需要将合并单元 R_i 合并到 BR_{i-1} 中，步骤如下：

(1)比较基础合并单元对应的转换单元的结果路径(P_{i-1})与被合并单元对应的转换单元的结果路径(P_i)中的元素，找到第一个不同的元素出现的位置。假设位置 m 上首次出现不同的元素，则对 R_i 进行 D 操作，删除 R_i 第 m 层以上的所有祖先节点(根节点为第 1 层节点)，得到 R_i'。

(2)取出 R_{i-1} 的前 $m-1$ 个元素组成集合 G。Mer 操作过程为：

a)按照集合 G，在 P_{i-1} 中从根节点开始遍历，设最后一个遍历到的节点为 E；

b)将(1)中得到的 R_i' 作为 E 的最后子节点添加到基础合并单元中。

经过(1)和(2)两个步骤后，R_i 就被合并到 BR_{i-1} 中，得到的结果作为新的基础合并单元 BR_i，其结果路径即为 R_i 对应的转换单元的结果路径(P_i)，其转换结果为合并后的树。按此将所有的合并单元都合并到基础合并单元后，便得到最终的转换结果。

需要指出的是，上述对于结果合并的讨论并不包括复合转换单元(由多个转换单元合并后得到的转换单元)的处理。由于复合转换单元的中间结果可能不存在公共的根，因此处理起来较为复杂，具体请参见文献[12]。

5　转换实例

下面为上述算法的一个完整示例。以下为一个代表 CD 唱片目录的源 XML 数据。

```
〈?xml version="1.0" encoding="UTF-8"?〉
〈catalog〉
  〈cd〉
    〈title〉Hide your heart〈/title〉
    〈artist〉Bonnie Tyler〈/artist〉
    〈country〉UK〈/country〉
    〈company〉CBS Records〈/company〉
    〈price〉9.90〈/price〉
    〈year〉1988〈/year〉
  〈/cd〉
  〈cd〉
    〈title〉Greatest Hits〈/title〉
    〈artist〉Dolly Parton〈/artist〉
    〈country〉USA〈/country〉
    〈company〉RCA〈/company〉
    〈price〉9.90〈/price〉
    〈year〉1982〈/year〉
  〈/cd〉
  〈cd〉
    〈title〉Still got the blues〈/title〉
    〈artist〉Gary Moore〈/artist〉
    〈country〉UK〈/country〉
    〈company〉Virgin records〈/company〉
    〈price〉10.20〈/price〉
    〈year〉1990〈/year〉
  〈/cd〉
  〈cd〉
    〈title〉Eros〈/title〉
    〈artist〉Eros Ramazzotti〈/artist〉
    〈country〉EU〈/country〉
    〈company〉BMG〈/company〉
    〈price〉9.90〈/price〉
    〈year〉1997〈/year〉
  〈/cd〉
  〈cd〉
    〈title〉Tupelo Honey〈/title〉
    〈artist〉Van Morrison〈/artist〉
    〈country〉UK〈/country〉
    〈company〉Polydor〈/company〉
    〈price〉8.20〈/price〉
    〈year〉1971〈/year〉
  〈/cd〉
〈/catalog〉
```

以下为处理上述数据的 XSLT 式样单，用于列出源 XML 文件中 1990 年以前的唱片和作者。为了方便说明，分别给其中的每个节点指定一个名字，如其中的 N_1，N_2，…，N_{15}。其中名为“root”的模板为根模板，N_1 为根节点。

```
〈?xml version="1.0" encoding="UTF-8"?〉
〈xsl:stylesheet version="1.1" xmlns:xsl="http://
www.w3.org/1999/XSL/Transform"〉
  〈xsl:template name="root" match="/"〉
    〈MySelect〉                                          N1
      〈xsl:call-template name="get_amount"/〉            N2
      〈List〉                                            N3
        〈TitleList〉                                     N4
          〈xsl:call-template name="get_title_list"/〉    N5
        〈/TitleList〉
        〈AuthorList〉                                    N6
          〈xsl:call-template name="get_author_list"/〉   N7
        〈/AuthorList〉
      〈/List〉
    〈/MySelect〉
  〈/xsl:template〉
  〈xsl:template name="get_amount"〉
    〈Total〉                                             N8
```

```
〈xsl:value-of select="count(//cd[year&lt;1990])"/〉   N9
〈/Total〉
〈/xsl:template〉
〈xsl:template name="get_title_list"〉
〈xsl:for-each select="catalog/cd[year&lt;1990]"〉   N10
〈Title〉   N11
〈xsl:value-of select="title"/〉   N12
〈/Title〉
〈/xsl:for-each〉
〈/xsl:template〉
〈xsl:template name="get_author_list"〉
〈xsl:for-each select="catalog/cd[year&lt;1990]"〉   N13
〈Author〉   N14
〈xsl:value-of select="artist"/〉   N15
〈/Author〉
〈/xsl:for-each〉
〈/xsl:template〉
〈/xsl:stylesheet〉
```

根据定义 3 得到的转换树如图 4 所示。

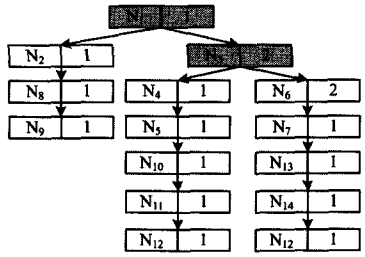

图 4　转换树示例

按照 3.2 节所述的规则，它可被划分为 2 个转换单元：$S_1=\langle T_1,\{1,1\},C_1,R_1\rangle$ 和 $S'=\langle T',\{1,2\},C',R'\rangle$。这时发现 C' 开销过高，S' 继续划分为 $S_2=\langle T_2,\{1,2,1\},C_2,R_2\rangle$ 和 $S_3=\langle T_3,\{1,2,2\},C_3,R_3\rangle$，其对应的转换分支如图 5 所示（带阴影的节点为划分节点）。

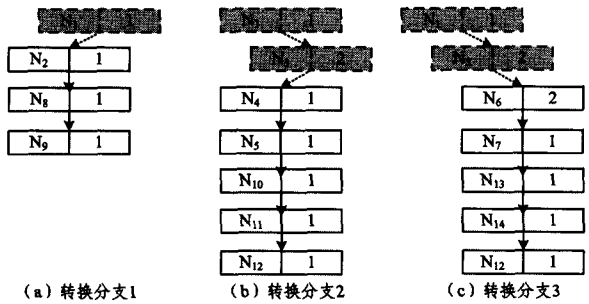

图 5　转换分支示例

下面列出相应各转换单元的式样单。假设 C_1、C_2 和 C_3 均在合理的波动范围内，不需调整。转换单元 S_1 的式样单 T_1 如下所示。

```
〈?xml version="1.0" encoding="UTF-8"?〉
〈xsl:stylesheet version="1.1" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"〉
〈xsl:template name="root" match="/"〉
〈MySelect〉
〈xsl:call-template name="get_amount"/〉
〈/MySelect〉
〈/xsl:template〉
〈xsl:template name="get_amount"〉
〈Total〉
〈xsl:value-of select="count(//cd[year&lt;1990])"/〉
〈/Total〉
〈/xsl:template〉
〈/xsl:stylesheet〉
```

转换单元 S_2 的式样单 T_2 如下所示。

```
〈?xml version="1.0" encoding="UTF-8"?〉
〈xsl:stylesheet version="1.1" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"〉
〈xsl:template name="root" match="/"〉
〈MySelect〉
〈List〉
〈TitleList〉
〈xsl:call-template name="get_title_list"/〉
〈/TitleList〉
〈/List〉
〈/MySelect〉
〈/xsl:template〉
〈xsl:template name="get_title_list"〉
〈xsl:for-each select="catalog/cd[year&lt;1990]"〉
〈Title〉
〈xsl:value-of select="title"/〉
〈/Title〉
〈/xsl:for-each〉
〈/xsl:template〉
〈/xsl:stylesheet〉
```

转换单元 S_3 的式样单 T_3 如下所示。

```
〈?xml version="1.0" encoding="UTF-8"?〉
〈xsl:stylesheet version="1.1" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"〉
〈xsl:template name="root" match="/"〉
〈MySelect〉
〈List〉
〈AuthorList〉
〈xsl:call-template name="get_author_list"/〉
〈/AuthorList〉
〈/List〉
〈/MySelect〉
〈/xsl:template〉
〈xsl:template name="get_author_list"〉
〈xsl:for-each select="catalog/cd[year&lt;1990]"〉
〈Author〉
〈xsl:value-of select="artist"/〉
〈/Author〉
〈/xsl:for-each〉
〈/xsl:template〉
〈/xsl:stylesheet〉
```

经并行转换，S_1 的中间结果 R_1 如下。

```
〈?xml version="1.0" encoding="utf-8"?〉
〈MySelect〉
〈Total〉3〈/Total〉
〈/MySelect〉
```

S_2 的中间结果 R_2 如下。

```
〈?xml version="1.0" encoding="utf-8"? 〉
〈MySelect〉
```

```
〈List〉
  〈TitleList〉
    〈Title〉Hide your heart〈/Title〉
    〈Title〉Greatest Hits〈/Title〉
    〈Title〉Tupelo Honey〈/Title〉
  〈/TitleList〉
〈/List〉
〈/MySelect〉
```

S_3 的中间结果 R_3 如下。

```
〈?xml version="1.0" encoding="utf-8"?〉
〈MySelect〉
  〈List〉
    〈AuthorList〉
      〈Author〉Bonnie Tyler〈/Author〉
      〈Author〉Dolly Parton〈/Author〉
      〈Author〉Van Morrison〈/Author〉
    〈/AuthorList〉
  〈/List〉
〈/MySelect〉
```

下面进行结果合并。3 个转换单元的结果路径为(1,1)、(1,2,1)和(1,2,2)。根据式(2),为排序,需要将之调整为:$\begin{Bmatrix}1,1,0\\1,2,1\\1,2,2\end{Bmatrix}$。调整顺序后的转换单元集合仍为$\{S_1,S_2,S_3\}$。

按照第 3 节的说明,结果树开始为空,R_1 合并到结果树后的结果如下。

```
〈?xml version="1.0" encoding="utf-8"?〉
  〈MySelect〉
    〈Total〉3〈/Total〉
  〈/MySelect〉
```

此时,基础合并单元的 $P_1'=\{1,1,0\}$。R_2 合并前需要做修整,修整过程为:比较 $P_2'=\{1,2,1\}$与基础合并单元的$P_1'=\{1,1,0\}$,出现第 1 个不相同元素的位置为第 2 项。由于 S_2 的结果路径中从第 1 项到第 2 项之间只有第 1 个元素,因此从 R_2 中去掉第 1 层节点(MySelect),修整后的 R_2'如下。

```
〈List〉
  〈TitleList〉
    〈Title〉Hide your heart〈/Title〉
    〈Title〉Greatest Hits〈/Title〉
    〈Title〉Tupelo Honey〈/Title〉
  〈/TitleList〉
〈/List〉
```

将其合并到基准结果中,其位置序号由 P_2'的第 1 项到第 1 个不相同项(1,2)指示。合并后的结果如下。

```
〈MySelect〉
  〈Total〉3〈/Total〉
  〈List〉
    〈TitleList〉
      〈Title〉Hide your heart〈/Title〉
      〈Title〉Greatest Hits〈/Title〉
      〈Title〉Tupelo Honey〈/Title〉
    〈/TitleList〉
  〈/List〉
〈/MySelect〉
```

再用该结果作为新的基础合并单元,其路径单元$P_2'=\{1,2,1\}$,合并 R_3 到中间结果。比较 $P_3'=\{1,2,2\}$与基础合并单元的 $P_2'=\{1,2,1\}$,出现第 1 个不相同元素的位置为第 3 项,所以去掉 R_3 的前 2 层节点,修整后的 R_3'如下。

```
〈AuthorList〉
  〈Author〉Bonnie Tyler〈/Author〉
  〈Author〉Dolly Parton〈/Author〉
  〈Author〉Van Morrison〈/Author〉
〈/AuthorList〉
```

将其合并到基准结果中,其位置序号由 P_3 中第 1 项到第 1 个不相同项(1,2,2)指示。合并后的结果如下。

```
〈?xml version="1.0" encoding="utf-8"?〉
〈MySelect〉
  〈Total〉3〈/Total〉
  〈List〉
    〈TitleList〉
      〈Title〉Hide your heart〈/Title〉
      〈Title〉Greatest Hits〈/Title〉
      〈Title〉Tupelo Honey〈/Title〉
    〈/TitleList〉
    〈AuthorList〉
      〈Author〉Bonnie Tyler〈/Author〉
      〈Author〉Dolly Parton〈/Author〉
      〈Author〉Van Morrison〈/Author〉
    〈/AuthorList〉
  〈/List〉
〈/MySelect〉
```

不难验证,上述结果与普通 XSLT 串行转换得到的结果是一致的。

6 实验结果

本课题将上述方法成功应用于微软、北京航空航天大学和北京信息科技大学等合作的"Open XML-UOF 转换器"第四期项目[13]。"Open XML-UOF 转换器"主要采用 XSLT 实现办公文档格式转换[14]。该项目中,Open XML 向 UOF 方向的演示文稿转换式样单共计 17 个 XSLT 文件、8683 行代码。所选取的源 XML 数据为该项目中提供的 220 个 Open XML 格式的演示文稿测试案例文档,其转换树如图 6 所示。

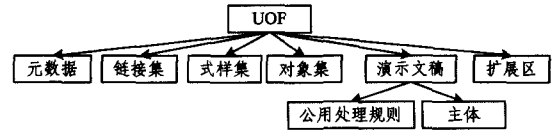

图 6 Open XML-UOF 转换器转换树

本实验从这 220 个文档中随机选取 10 个文档作为 XSLT 自动划分时的训练数据。图 7 所示为训练过程中各个转换单元划分与合并的过程,最终形成了 3 个转换单元。

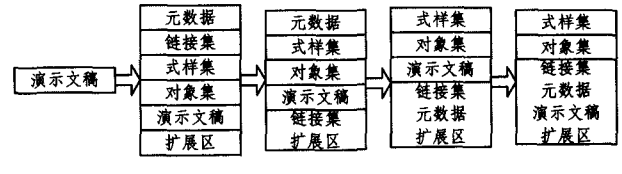

图 7 基于训练的式样单的划分与合并过程

分别按单线程串行方式和多线程方式对待测试数据进行 10 次转换,测试结果如图 8 所示,单位为 ms。其中"自动划分"为划分式样单所消耗的时间,共启用了 3 个转换线程:线程 1 用于转换"式样集";线程 2 用于转换"对象集";线程 3 用于转换"元数据、链接集、演示文稿、扩展区"。

从图 8 可以看出，并行转换方式下每个转换线程的时间消耗都小于串行方式。如果独立地并行执行这些线程，总执行时间将为以下三者之和：最大线程时间消耗、自动划分时间消耗以及结果合并时间消耗。此时还可以单独设置一个合并线程，采用流水线的方式，在转换的同时进行合并，合并时间将进一步缩短。

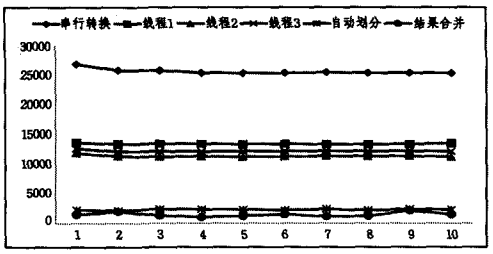

图 8　多线程转换与单线程转换的时间数据对比

图 9 展示了多线程与单线程方式下总执行时间的对比。多线程方式尽管增加了自动划分式样单带来的时间消耗，但总体性能仍大幅优于单线程方式。

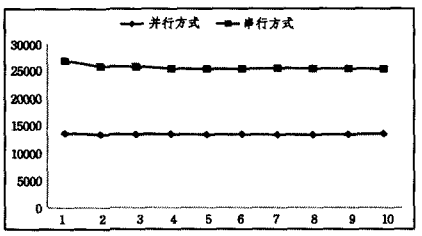

图 9　多线程与单线程总体执行时间对比

结束语　为提高 XSLT 用于 XML 数据转换的性能，本文提出了一种基于式样单划分的并行转换方法。该方法可以从式样单中自动获得若干独立的转换单元，并达到最适宜运行环境的任务数量和均衡的负载。在并行转换之后，需要将多个转换结果进行有效的合并。本文提出的方法可以在划分阶段记录下划分节点的路径信息，用于指导合并操作，保证合并时节点顺序和插入位置的正确性。实验结果表明，相对于传统的串行转换方法，本文提出的方法能够大幅提升转换效率。此外，本文提出的方法不依赖于特定的并行处理环境，无需修改 XSLT 处理器。

本方法还存在一些需要继续改进的地方。首先，并行处理的颗粒度可以进一步细化。例如 XSLT 用于循环控制的 xsl:for-each 节点和多模板匹配的 xsl:apply-templates 节点目前都被视为不可划分节点，当前这样处理是为了降低自动划分的难度，未来如果能够考虑对这些节点的深入划分，势必可以进一步提高 XSLT 的并行处理能力。其次，转换单元的动态调整依赖于 C 值的计算，而 C 值是根据训练数据得到的，不同的训练数据对 C 值的影响会有所不同，存在一定的随机性，需要研究更有效的评估式样单执行开销的方法。

本文的研究结果能够充分发挥一般多核处理器的处理能力，将之用于 XSLT 并行转换，特别对于服务器端的大规模 XML 数据转换具有很好的作用；此外，该研究结果对支持 XSLT 标准的未来发展也具有积极意义。为了提高 XSLT 处理效率，在 XSLT 2.1 之后的版本[14,15]中增加了 fork 等指令以支持流式特性。本文的方法可以用于该特性的实现，并提供更为全面的功能。

参 考 文 献

[1] Dong Ce, Bailey J. Static analysis of XSLT programs[C]// Proceedings of the 15th Australasian database conference, Volume 27. Australian Computer Society, Inc., 2004: 151-160

[2] Kwong A, Gertz M. Schema-based optimization of XPath expressions[R]. University of California, 2002

[3] Dong Ce, Bailey J. Optimization of XML transformations using template specialization [M]// Web Information Systems-WISE 2004. Springer Berlin Heidelberg, 2004: 352-364

[4] Mizumoto H, Suzuki N. An XSLT transformation method for distributed XML[C]// 2014 Fourth International Conference on Innovative Computing Technology (INTECH). IEEE, 2014: 71-80

[5] Chen Ru-chang, Yan Yi. An approach of improving XSLT conversion efficiency[J]. Mechanical & Electrical Engineering Magazine, 2009, 26(4): 80-83(in Chinese)
陈如昌，严义. 一种提高 XSLT 转换效率的方法[J]. 机电工程，2009，26(4)：80-83

[6] Gao Jun, Yang Dong-qing, Tang Shi-wei, et al. XPath Logical Optimization Based on DTD [J]. Journal of Software, 2004, 15(12): 1860-1868(in Chinese)
高军，杨冬青，唐世渭，等. 一种基于 DTD 的 XPath 逻辑优化方法 [J]. 软件学报，2004，15(12)：1860-1868

[7] Chen Peng-sheng, Chu Fu-shun. Method for fast XSL transformation on multithreaded environment: U. S. Patent Application 11/905,782[P]. 2007-10-4

[8] Sun Yuan-hao, Li Tian-you, Zhang Qi. Parallel xml transformations on multi-core processors[C]// IEEE International Conference on e-Business Engineering, 2007 (ICEBE 2007). IEEE, 2007: 701-708

[9] Li Ren, Luo Jian-hua, et al. A Scalable XSLT Processing Framework based on MapReduce[J]. Journal of Computers, 2013, 8(9): 2175-2181

[10] Luo Wen-tian. Study on the Method to Improve XSLT Transform Performance [D]. Beijing: Beijing Information Science and Technology University, 2012(in Chinese)
罗文甜. XSLT 转换性能改进方法研究[D]. 北京：北京信息科技大学，2012

[11] W3C. XSL Transformations (XSLT) Version 1.0 [S/OL]. (2013-08-04). http://www.w3.org/TR/xslt

[12] Gao Xiao-guang. Study on the Method to Improve XSLT Transform Performance by Parallel Processing [D]. Beijing: Beijing Information Science and Technology University, 2013 (in Chinese)
高晓光. 并行方式提高 XSLT 转换性能方法研究[D]. 北京：北京信息科技大学，2013

[13] Microsoft, BUAA, BISTU. UOF Translator[EB/OL]. (2011-06-30). http://sourceforge.net/projects/uof-translator

[14] Kay M. Streaming in XSLT 2.1[J/OL]. XML Prague 2010, 2010-3. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.231.473&rep=rep1&type=pdf#page=21

[15] XSL Transformations (XSLT) Version 3.0[S/OL]. (2013-12-12). http://www.w3.org/TR/xslt-30