XML过滤中缓存失效策略的性能分析数学模型

沈 洁 印桂生 王向辉
(哈尔滨工程大学计算机科学与技术学院 哈尔滨 150001)

摘 要 硬件缓存行为是内存驻留的数据密集型系统(例如XML过滤机制)的一个重要特征。目前对XML的过滤方式的主流研究都是用自动机来表达主存中长期运行的XML查询。现主要通过分析性的建模和系统度量来研究基于自动机的XML过滤的cache性能,将原本笼统地针对整个cache失效性的分析细化成建立cache圈内模型和跨圈模型来估计cache的失效率,并通过实验证明该评估机制具有较高的精确度。

关键词 XML,过滤,失效性,缓存
中图法分类号 TP393 **文献标识码** A

Evaluating Model of Cache Miss in the Filtering of XML Data

SHEN Jie YIN Gui-sheng WANG Xiang-hui
(Department of Computer Science and Technology, Harbin Engineering University, Harbin 150001, China)

Abstract One of the most important factor in the data-intensive system(ig. XML filtering engine) with memory resident is hardware cache behavior. Now, the popular researches in XML filtering are based on the automata to express the XML query. In this paper, we studied the cache performance of XML filtering based on the automata by the analytical modeling and system measurement, and estimated the cache miss in the intra-round model and inter-round model instead of the general estimation in the cache. Our results show that our estimation engine has more accuracy.

Keywords XML, Filtering, Miss, Cache

XML过滤是目前热门的查询处理方式,存在于主存中的XML查询将大量的XML文档进行过滤。XML可以应用于许多领域,例如选择性的信息发布或者基于内容的XML路由。由于XML的过滤机制需要内存驻留并且需要长期运行,因此需要对其缓存性能加以分析,确定能否通过缓存性能的改善来改善整个机制的性能。

1 相关的工作和本文的工作

为了研究基于自动机的XML过滤的缓存性能,将现有的XPath-NFA模型[3,4]建成一个YFilter的简化模型。它利用路径分享(共享查询中的公共表达式)通过XPath-NFA表达大量的XPath查询,但是这种模型没有提出其它的优化技术。进一步的研究在已有的对NFA向DFA的转换的基础上,通过构建子集来研究XPath-NFA向XPath-DFA的转化[6],并且通过实验讨论XPath-NFA和XPath-DFA在过滤性能上的差异。实验证明,在主存的运行能力允许的情况下构造子集来实现的XPath-DFA,在过滤性能上远胜于XPath-NFA。因此本文中也将采用XPath-DFA来进行cache性能的后续研究。

对于大量存储在缓存中的XML数据,在过滤的过程中不可避免地会产生缓存失效(cache miss),大量缓存失效的存在会导致过滤的效率低下,于是研究缓存的失效率对于改进XML数据的过滤是有重要意义的。我们将缓存中的失效分为强制失效、容量失效和冲突失效。本文就是要通过分析DFA对XML数据的过滤过程来评估其间缓存失效可能发生的情况,然后给出一个对于强制失效相关联的冲突失效的粗略估计方法。

2 缓存性能评估模型

首先基于XPath查询介绍一下XPath-DFA的过滤过程。它是由来自于SAX解析器的事件驱动。一般而言有以下4种类型的事件:start-of-document, end-of-document, start-of-element和end-of-element。一个start-of- document引发一个新的圈,而end-of-document标志着一个圈的结束。在一个圈周期中,当一个start-of-element事件到达时,就引发了自动机上的一次状态转移。当一个end-of-element事件发生时,自动机返回到上一个状态。可以建立一个运行时间栈来表示当前和先前访问过的状态。通过一个end-of-document事件,系统检查是否已经到达了任何的可接受状态,若有,则将这个文档发送给满足XPath查询要求的用户。

2.1 缓存建模

现代计算机中内存的层次通常包含一级缓存(L1 cache)、二级缓存(L2 cache)和主存。在缓存(L1或者L2)中存取数据或者命中(hit)或者失效(miss),给定cache失效的

到稿日期:2009-07-22 返修日期:2009-10-26 本文受863基金(No. 2007AA012401)资助。
沈 洁(1981-),女,博士生,主要研究方向为XML数据库,E-mail:shenjie@hrbeu.edu.cn;印桂生(1964-),男,博士,教授,博士生导师,主要研究方向为数据库与知识库、虚拟现实;王向辉(1980-),男,博士,讲师,主要研究方向为数据库、P2P网络。

数量 $\#miss$，以及 cache 命中的数量 $\#hit$，失效率就可以定义为 $m=\frac{\#miss}{\#miss+\#hit}$。一个内存中存取的平均代价 T_{avg} 计算如下：

$$T_{avg}=t_{L1}+m_{L1}\cdot t_{L2}+m_{L1}\cdot m_{L2}\cdot t_M \quad (1)$$

式中，t_{L1}，t_{L2}，t_M 分别是 L1，L2 和主存的存取时间，m_{L1}，m_{L2} 是 L1 和 L2 的失效率。由于存取时间是由硬件决定的，因此我们旨在减少缓存的失效率以及缓存中存取的总数。

我们定义缓存的配置是一个四元组〈C,B,N,A〉(如表 1 所列)。当 $A=1$ 时，为直接映像 cache；$A=N$ 时为全关联缓存；当 $A=n(1<n<N)$ 时为 n 关联缓存。除此以外，我们还要建立类似 LRU(Least Recently Used)的缓存替代策略模型。

表 1　Cache 配置

Parameters	Description
C	The cache capcity(bytes)
B	The cache line size(bytes)
N	The number of cache lines(C/B)
A	The degree of set-associativity

3　圈内过滤模型

首先从圈内开始过滤，此时只有一个文档被过滤，并且没有数据是从之前的圈中被重用的。由于在实际应用中通常一个文档都比较小，因此假定这儿不存在容量失效的情况，所以主要关注强制性的失效，并且讨论与基于强制失效产生的冲突失效。

定义 1(状态转移)　在自动机中的状态转移 i 用一个三元组〈Fi,Ri,Mi〉来定义。其中 Fi 代表 i 的足迹，足迹 F 是包含状态转移信息的 cache 行数的平均值，基于矩阵的自动机中 Fi 为常量 1，基于哈希表的自动机中 Fi 为常量 2，而基于映射的自动机中 Fi 为变量。为了简化过程，定义 Fi 为一个常量并且 Fi 和 F 之间是可交换的；Ri 代表 i 的重用，如果 $Ri=Fi$ 则说明所有的状态转移已经存在，称为最佳重用；而 Mi 则表示由 i 所带来的强制失效的数量。最后定义 ws 为 XML 文档在自动机上的工作集，这些集合中的状态都是在过滤过程中涉及到的，而工作集的大小就是工作集中状态的数量，表示为 $|ws|$。

3.1　评估圈内过滤中的强制失效性

根据状态转移的定义〈Fi,Ri,Mi〉以及工作集 ws，对于一个文档 d，可以通过两种方式来计算强制性失效 Com：(1) $Com=\sum_{i=1}^{p}M_i=\sum_{i=1}^{p}(F-R_i)$，其中 p 是过滤文档中所有的状态转移的总数；(2) $Com=|ws|\cdot F$。一般采用第二种计算方法。下面定义标签树来评估 XPath-DFA 中文档的工作集。

定义 2(标签树)　文件的标签树是满足以下性质的树：(1)每个节点都表示自动机中的一个状态并且拥有一个独立的 ID 号(根节点的 ID 号为 0)；(2)每条边代表了自动机中的一次状态转移并且标注文件的标签。如果对应的自动机的状态在 cache 中，就认为标签树中对应的节点也在 cache 中。通过借鉴 LRU 的替代策略，为每个节点增加一个时间戳，每个时间戳的初始化值为 -1，这就意味着该节点并没有被引入 cache 中。每当一个状态被存取时，相应节点的时间戳就增加 1，对应的数据表示发生状态转移的次数总数。一个带有正值时间戳的节点就表示该节点正处于 cache 中，并且越晚被存取的节点的时间戳值越大。

根据标签树的定义来构造标签树算法，并完成对文档中工作集的计算，统计强制性失效的数目，如算法 1。通过分析文件来建立标签树。最初，根节点(0 号节点)是一个主动节点。对每个 start-of-element 事件 e，使用标签树来模拟其状态转移。首先，检查主动节点是否存在通过 e 边连接的孩子节点(Line6)。如果没有，就为主动节点创造一个通过 e 边连接的孩子节点(Line7)。然后将这个孩子节点(不管是已有的还是后建的)压入运行时间栈(Line8)，接下来决定是否重用 Ri：如果该节点已经在 cache 中，就有一个理想的重用产生(Line9－10)。而(Line11)将活跃节点的时间戳更新。当一个 end-of-element 时间到来时，将节点出栈。最后，所有的节点都带有正值时间戳，这代表当过滤这个文档的时候所有的状态都被存取过(Line14)。在算法的最后估计出了强制性失效的数量(Line15)。根据这个算法，可以估算出图 1 中样本文件的工作集的大小为 4，强制性失效的数量也是 4(给定 $F=1$)。由于我们的算法假定在卷标树节点和自动机状态之间的关系是一对一的映像，因此当标签树节点和自动机的状态之间的关系是多对一的时候，评估的精确性就会受到影响。但是，在绝大多数大型过滤系统当中这种映像都是接近于一对一的关系。

算法 1　构造标签树

```
输入：文档事件数据流，足迹 F
输出：ws，Com
1：对于一个 start-of-document 事件，初始化 T 为只有 0 节点的标签
   树，栈 S，i=0，s. push(Node 0)；
2：while e is not an end-of-document event do
3：  if e is a start-of-element event then
4：  i++，Ri =0；//Processing the i^th state transition
5：    curNode=s. top()；
6：    if ChildOf(curNode，e)=null then
7：      NewChild(curNode，e)；
8：      s. push(ChildOf(curNode，e))；
9：      if curNode. timestamp ≥0 then
10：        Ri=F；
11：      curNode. timestamp=1；
12：    else
13：  s. pop()；// e is a end-of-element event
14：add all nodes of a positive timestamp to ws；
15：Com=|ws|·F
```

下面给出一个例子具体说明如何来构造一棵标签树。例 1：给定一个文件〈A〉〈B〉〈C〉〈/C〉〈D〉〈/D〉〈E〉〈F〉〈/F〉〈/E〉〈/B〉〈/A〉，根据算法 1 构造的标签树如图 1(a)所示，方框表示的是标签树的节点，节点当中的数字表示节点的 ID 号，方框外面标注的数字表示节点的时间戳。标签树的构造过程类似于过滤过程。图 1(b)所示的是运行时间栈，它被用来显示活跃的节点和已经被存取过的节点。栈中的“$”符号表示时间戳的更新。

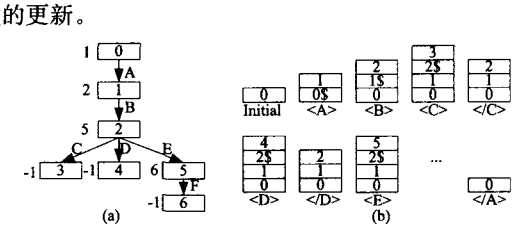

图 1　标签树的构造

4　跨圈过滤模型

前面已经讨论了圈内过滤中建立模型来评估缓存失效，

下面将建立两圈之间过滤中关于缓存行为的模型。圈内过滤和跨圈过滤有两个主要的区别：(1)圈内过滤当中的每个圈(第一圈除外)可以重用先前文档的工作集；(2)当面对多个文档时，过滤中涉及到的所有状态集的大小都可能会超过缓存的容量。因此，在跨圈过滤系统中应该着重考虑容量失效。

定义 3(工作集中的交集、并集和差运算) 给定一个文档序列($doc_1, doc_2, \cdots, doc_n$)，利用算法 1 可以得到相关的工作集($ws_1, ws_2, \cdots, ws_n$)。分别定义工作集序列上的交、并、差运算为 n 元集合运算：Intersection($ws_1 \cap ws_2 \cap \cdots \cap ws_n$)；Union($ws_1 \cup ws_2 \cup \cdots \cup ws_n$)和 Difference($ws_1 - ws_2 - ws_n$)。

定义 $ws_1, ws_2, \cdots, ws_n$ 的子集为 $ws_{i,1}, ws_{i,2}, \cdots, ws_{i,n}$，并在第 i 圈开始时进入缓冲。我们将 $rs_{i,j}$ 称为第 i 圈起始处文档 j 的缓冲集。而缓冲集的集合$\{rs_{i,j} \mid 1 \leqslant j \leqslant i\}$组成了从第 i 圈开始的缓冲内容。而第 $i+1$ 圈的缓冲内容是$\{rs_{i+1,j} \mid 1 \leqslant j \leqslant i+1\}$，这也是第 i 圈结束时的缓冲内容。由于前面假设的在圈内过滤中忽略容量失效，因此，在第 i 圈起始处 $rs_{i,j} = \emptyset$，而在 $i+1$ 圈起始处 $rs_{i,j} = ws_i$。

4.1 跨圈评估

对于给定的一个 n 个文档的序列($doc_1, doc_2, \cdots, doc_n$)以及相关的工作集($ws_1, ws_2, \cdots, ws_n$)，需要在每圈都检查缓存中内容是否改变。而改变除了受到文档工作集的影响以外，还与缓存的置换策略有关。在这里利用对标签树的交、并、补的操作对圈进行评估。

两个节点 n_i 和 n_j，来自两棵标签树 T_i 和 T_j，如果满足以下两个条件，就认为它们是重迭的：(1)存在一条从 T_i 的根节点到 n_i 的路径以及一条从 T_j 的根节点到 n_j 的路径，且两路径包含相同的卷标序列；(2)节点 n_i 和 n_j 都有正值的时间戳。通过这个定义，两个有正数值的根节点必定是重迭的。而负值时间戳的节点不予考虑，因为它们不在缓存中。

给定两个标签树 T_m 和 T_n($m<n$)，分别定义标签树的 3 个操作：(1) 交 $I(T_m, T_n)$的结果是一个新的标签树，其所有的节点来自两棵树中重迭的部分，而这些重迭节点的时间戳被设置成与 T_n 中相应的节点一致。(2) 差 $D(T_m, T_n)$的结果是一个源于 T_m 的新的标签树，且每个重迭节点的时间戳都被设置为-1。(3)并 $U(T_m, T_n)$产生一个新的包含所有时间戳被设置为 T_n 一致的重迭节点的标签树，以及来自两棵树的其它节点。下面举例来说明关于标签树的操作。给定两个文档，文档 1〈A〉〈B〉〈C〉〈/C〉〈D〉〈/D〉〈E〉〈F〉〈/F〉〈/E〉〈/B〉〈/A〉；文档 2〈A〉〈B〉〈C〉〈D〉〈/D〉〈/C〉〈E〉〈F〉〈/F〉〈G〉〈/G〉〈/E〉〈/B〉〈/A〉。图 2(a)，(b)分别表示依据文档 1 和文档 2 构建的标签树 T_1 和 T_2，阴影部分表示的是 T_1 和 T_2 的重迭部分。注意 T_1 和 T_2 中的标签为 3 的节点不是重迭节点，因为 T_1 中它的时间戳为负值。图 2(c)表示的是 T_1 和 T_2 的交，而 T_1 和 T_2 的差即 T_1 中所有节点，所有节点的时间戳为-1。

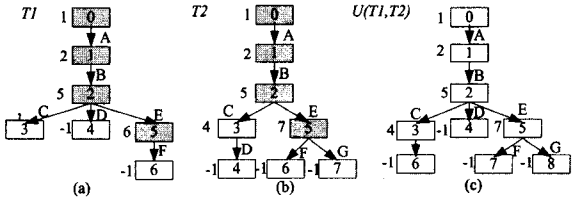

图 2 标签树及其操作

同理，可以将在缓存集上的操作转换成相应的标签树的操作。假设标签树 $rt_{i,j}$ 对应于相关的缓存集 $rs_{i,j}$，$rt_{i,j}$ $\{1 \leqslant j \leqslant i\}$组成的从第 i 圈开始的缓冲内容。不同的圈有不同的缓存内容，标签树需要反映出缓存内容的改变。因此，可以在标签树的每个节点中增加一个圈号，显示出最近使用该节点的圈。

在第 i 圈，给定标签树 T_i 相应的工作集为文档 i。假定与 T_i 中节点重迭的每个节点 $rt_{i,j}$ ($1 \leqslant j \leqslant i$)都有一个圈号 i 和一个与 T_i 中对应节点一致的时间戳。当缓存中的内容发生变化时，圈号较小的节点将被优先置换；而对于圈号一样的节点，时间戳值较小的节点被优先置换。当一个节点被置换出缓存后，时间戳将被设置为-1，在第 i 圈过滤完成后，$rt_{i+1,i} = T_i$。

通过对标签树的这 3 个操作，可以估算出第 i 圈的强制性失效为$|T_i - (rt_{i,1} \cup rt_{i,2} \cup \cdots \cup rt_{i,j})| \times F$，而容量失效的方法也可以同理推算出。

5 实验

本实验中的硬件环境为 IBM X61 计算机，操作系统是 Windows XP，CUP 为 Intel T7100，主频为 1.8GHz，内存为 2G。L1 缓存为 32k (16k 结构，16k 数据)，L2 缓存为 512k。cache 行大小(line size)为 64 字节。过滤系统用 C++并通过 g++3.2.2-5 编译。过滤实验全部采用内存驻留(memory-resident)技术并保证内存的使用率不超过 80%。实验中使用的两个文档集如下：

(1)圈内过滤中的文档集由 40 个大小不同的文档组成，每个文档 k($1 \leqslant k \leqslant 40$)包含$(40 * k + 1)$个带大约$(32 * k)$个工作集的 XML 元素。带最大工作集($32 \times 40 = 1280$ cache lines)的文档也远小于 L2 缓存大小(512kB/64B=8092 cache lines)。

(2)通过 80 个 XML 文档序列来验证跨圈模型的有效性，每个序列包含 40 个大小相同的文档。序列 s 中的一个文档包含$(5 * s + 1)$个元素，每个元素带有$(4 * s)$个工作集。

通过实验来证明前面提出的缓存过滤模式的精确性。过滤精确性的定义如下：

$$accuracy = \frac{\min(measurement, estimation)}{\max(measurement, estimation)}$$

5.1 圈内模型的实验数据

在外圈内过滤中，从每个文件集合的各个文件开始过滤。对于每个文档，首先度量缓存失效的总数，然后通过文档复制状态存取到邻近区域并且计算当连续存取这些状态时缓存失效的数量。由于在连续存取时发生的冲突性失效数量很少，且无容量失效，因此这个数值近似于过滤这个文档时发生的强制性失效。

图 3 给出的两条曲线分别代表圈内过滤时对各个文档进行过滤时发生的强制性失效的测量值(measurement)和评估值(estimation)。给定的 40 个文件的平均精确度为 91.2%，评估值总是略低于测量值的，原因可能包括为了简化而使用的一些假设。通过过滤单个文档，可以通过在总的失效数量中减去强制性失效的数量，从而计算出冲突失效的数量。通过实验知道，在整个文档中冲突失效和强制性失效的比率基本上是固定的，平均值为 0.84。在圈内模型中，一般用这个

平均值由强制性失效的值评估冲突失效。

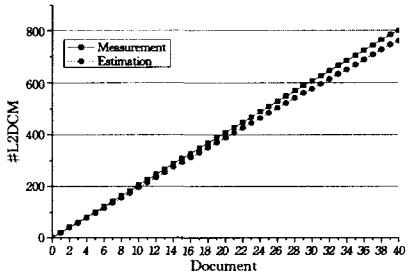

图 3　圈内模型的实验数据图

5.2　跨圈模型的实验数据

利用第 2 个文档集来实验跨圈过滤，得到的实验结果如图 4 所示。我们还是使用在圈内过滤时的冲突失效和强制性失效的平均比率来评估冲突失效值。

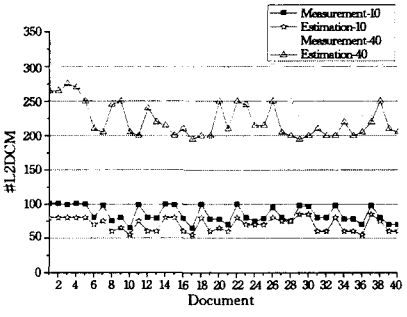

图 4　跨圈模型的实验数据图

结束语　本文主要提出在通过自动机对 XML 数据流进行过滤的情况下如何过滤的结果进行精确的评估。通过对缓存的分解，两段式地评估过滤的过程，从而得到与实际结果较为符合的评估结果。本文提出的圈内模型和跨圈模型经过实验证明存在较高的精确度。下一阶段的研究重点可以考虑，对于优化自动机的过滤方式，如何利用本文所建立的模型验证优化过的自动机能够既保证过滤效率的提高又能确保缓存查询的丢失。

参 考 文 献

[1] Altinel M, Franklin M J. Efficient filtering of XML documents for selective dissemination of information[C]// VLDB. 2000

[2] Chan C-Y, Felber P, Garofalakis M, et al. Efficient filtering of XML documents with XPath expressions[J]. VLDB Journal (Special Issue on XML), 2002, 11(4)

[3] Diao Y, Altinel M, Franklin M J, et al. Path sharing and predicate evaluation for high-performance XML filtering[C]// TODS. December 2003

[4] Diao Y, Fischer P, Franklin M J, et al. Yfilter: Efficient and scalable filtering of XML documents[C]// ICDE. 2002

[5] Green T J, Miklau G, Onizuka M, et al. Processing XML streams with deterministic automata[C]// ICDT. 2002

[6] Organization T S P. Sax: Simple API for XML[EB/OL]. http://www.saxproject.org

[7] 吴劲，卢显良，任立勇，等. 缓存失效策略的性能分析数学模型[J]. 电子科技大学学报，2005，34(4)：225-228

[8] 吴劲，卢显良，任立勇，等. 移动计算环境中缓存失效策略的归类研究法[J]. 计算机科学，2004，31(1)：39-41

(上接第 170 页)

结束语　本文给出了一种基于 ESB 的服务动态集成框架，业务操作在此框架中被分解为若干服务-数据流。同时提出了一种新的评价策略以衡量分解的有效性，给出了 ASDT，ASMT，ADDT，ADMT 和路由表的更新算法，以解决服务提供者和数据源更新时所引发的抽象描述与其实现模块之间的冲突。将该方法应用于“黑龙江省运政系统”的设计中，实现了省运管局各部门和单位之间异构服务和分布式数据的动态共享与集成。

参 考 文 献

[1] Kuppuraju S, Kumar A, Kumari G P. Case Study to Verify the Interoperability of a Service Oriented Architecture Stack[C]// Proc. of International Conference on Services Computing. 2007: 678-679

[2] Fujii K, Suda T. Semantics-Based Dynamic Service Composition [J]. IEEE Journal on Selected Areas in Communications, 2005, 23(12): 2361-2372

[3] Penta M, Esposito R, Villani M. WS Binder: A Framework to Enable Dynamic Binding of Composite Web Services[C]// Proc. of the 2006 International Workshop in Service-Oriented Software Engineering. 2006: 74-80

[4] Laliwala Z, Majumdar P, Chaudhary S. Semantic and Rules Based Event-Driven Dynamic Web Services Composition for Automation of Business Process[C]// Proc. of IEEE Services Computing Workshops. 2006: 175-182

[5] Chafle G, Dasgupta K, Kumar A. Adaptation in Web Service Composition and Execution[C]// Proc. of International Conference in Web Services. 2006: 549-557

[6] 李晓东，杨扬，郭文彩. 基于企业服务总线的数据共享与交换台[J]. 计算机工程，2006，32(21)：217-219

[7] 邵欢庆，康建初. 企业服务总线的研究与应用[J]. 计算机工程，2007，33(2)：220-222

[8] Maréchaux J-L. Combining Service - Oriented Architecture and Event-Driven Architecture Using an Enterprise Service Bus [M]. IBM Developworks, 2006

[9] Luo Min, Goldshlager B, Zhang Liang-jie. Designing and Implementing Enterprise Service Bus(ESB) and SOA Solutions[C]// Proc. of the International Conference on Services Computing. 2005: 14

[10] Bai Xiao-ying, Xie Ji-hui. DRESR: Dynamic Routing in Enterprise Service Bus[C]// Proc. of International Conference on E-Business Engineering. 2007: 528-531