基于有限自动机的 XML 过滤技术研究综述 *)

覃泳睿　孙未未　张卓瑶　余　平
(复旦大学计算机信息与技术系　上海 200433)

摘　要　介绍了基于有限自动机的 XML 过滤技术的研究现状,依据自动机的特点进行分类,分析了其中几种典型算法,并对 XML 过滤算法的优缺点进行了深入讨论和比较。

关键词　有限自动机,查询索引,XML 过滤技术

Survey of Finite Automaton Based XML Filtering Techniques

QIN Yong-rui　SUN Wei-wei　ZHANG Zhuo-yao　YU Ping
(Dept. of Computing and Information Technology, Fudan University, Shanghai 200433, China)

Abstract　Presented that the current state of research about XML filtering techniques based on finite automaton (FA). Classified the techniques according to the features of FA, and analyzed some typical algorithms. Also deeply discussed and compared the advantages and disadvantages of these algorithms.

Keywords　Finite automaton, Query index, XML filtering technique

1　引言

XML (eXtensible Markup Language)[1]即可扩展标记语言,自 1998 年出现以来发展迅速,目前已经成为国际互联网数据交换的标准格式。对 XML 格式数据的研究也是当前的研究热点[2]。

XML 文档是比较典型的半结构化数据(Semi-Structural Data)。半结构化数据是指介于完全结构化数据(如关系型数据库、面向对象数据库中的数据)和完全无结构的数据(如声音、图像文件等)之间的数据。它一般是自描述的,数据的结构和内容混在一起,没有明显的区分。XML 将结构信息编码到文档中,可以更加准确地描述用户感兴趣的数据,更好地体现了用户的需求。不过由此也产生了额外的代价:增加了 XML 文档与用户需求之间进行匹配即过滤 XML 文档的复杂度。在互联网环境下,存在海量的 XML 数据和对这些数据感兴趣的大量用户,因此对这些 XML 数据进行过滤的效率和可伸缩性的研究成为了热点问题。

XML 过滤技术研究中的主要问题有:1)如何构造高效的查询索引。和传统数据库系统中需要为所存储的大量数据建立索引一样,在 XML 过滤系统中,如何为大量的查询建立索引显得十分重要。同时,对于 XML 文档数据流还需要考虑如何建立合适的索引,以提高过滤引擎的效率。2)对于 XML 过滤系统,需要考虑如何有效地支持带谓词与带分支路径的查询。3)查询索引的更新问题。XML 作为国际互联网上数据交换的标准格式,XML 过滤系统也主要应用在国际互联网上,随时都可能会有新的用户提出新的查询请求,原有的用户也可能会提出新的查询请求或改变以前提出的查询请求,因此查询索引的更新效率也显得十分重要。

XML 过滤技术研究中考虑的主要因素:1)不同的 XML 文档的结构特征对过滤效率产生的影响。对于不同用途的 XML 文档,其文档结构的特征、复杂程度必然有很大的不同,从而可能对过滤系统的效率产生较大的影响。如 XML 文档的平均深度、所有 XML 元素的出现概率的分布等因素都可能对过滤效率产生影响。2)不同的用户查询集合对过滤效率的影响。用户的查询需求的多样性以及 XPath[3]查询表达式的灵活性使得用户查询集合中 XPath 查询表达式的特点将对 XML 过滤系统的效率有重要影响。如查询集合中查询表达式的数量、查询表达式中子孙运算符"//"和通配符"*"的出现概率、查询集合中所有查询表达式的平均深度等因素对过滤效率的影响。

近几年来,在 XML 过滤技术领域,运用有限自动机来对 XML 文档进行过滤已经有了大量的研究工作,并取得了大量的成果,它也是大规模分布式计算发布订阅系统中核心技术之一[4]。

本文主要对基于有限自动机的 XML 过滤技术的研究成果和研究现状进行分析讨论。首先介绍了 XML 过滤系统,并对当前基于有限自动机的 XML 过滤技术进行分类;接下来对当前主流的基于有限自动机的 XML 过滤技术进行分类阐述,并详细分析了各种有限自动机模型的优缺点;最后对现有技术进行了总结并对未来研究方向提出了展望。

2　背景介绍及分类比较

2.1　背景介绍

建立高性能、可伸缩的 XML 过滤系统有一个关键点,即查询和数据的角色与传统的数据库系统相比是调换过来的。在传统的数据库系统中,大量的数据是被存储的,并建立有各种相应的索引,查询则是独立地应用于系统上。在 XML 过滤系统中,查询是大量的、被长期存储的,并且系统对查询建

*)基金项目:国家自然科学基金项目(60503035),国家"八六三"高技术研究发展计划(2006AA01Z234),国家"九七三"重点基础研究发展规划基金项目(2006AA01Z234)。覃泳睿　硕士研究生,主要研究方向为移动数据管理;孙未未　博士,副教授,主要研究方向为移动计算;张卓瑶　硕士研究生,主要研究方向为无线数据广播;余　平　博士研究生,主要研究方向为移动数据管理。

立索引。而对于 XML 数据,通常为 XML 文档,通过以 XML 数据流的形式到达过滤系统并与已建立好索引的查询进行匹配,匹配的结果即为查询的结果。

图 1 是一个过滤系统的概要图示,这里简要说明一下过滤系统的各组成部分。首先是两大输入数据集:一个是文档流(Document streams),在 XML 过滤系统指的是 XML 数据流;另一个是用户需求描述(User profiles),在 XML 过滤系统中对应查询集合,通常是采用 XPath(准确来说是 XPath 的一个子集)描述的路径查询表达式集合,代表了不同的用户对数据的偏好。其次是过滤引擎系统,该引擎系统将会转换用户需求描述,使其变成可以进行有效存储和高效执行过滤操作的形式。当 XML 文档数据流到达时,过滤引擎将每个到来的文档与所有的用户查询进行匹配,以确定文档需要转发给哪些用户接收。文档转发的形式可以是整个文档的全部转发或者仅仅是实际匹配文档片段的转发。

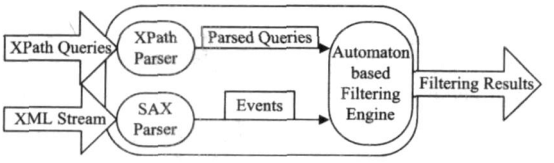

图 1　过滤系统

对于到来的文档数据流,首先需要转化为一系列的 XML 事件流,这项工作可以由基于 SAX 的 XML 解析器来完成。图 1 是一个简单 XML 文档经过 SAX 解析器解析后,转化为 XML 事件流的示意图。过滤引擎根据类似的事件流,将到达的 XML 文档与用户查询集合进行匹配。

在过滤 XML 文档时,一般情况下,其查询集合是采用面向结构的路径查询语言 XPath 描述的,与传统的关键字匹配技术相比较,存在以下的新问题:

1)需要准确匹配 XML 元素的顺序以及所处的位置;

2)对于 XPath 路径表达式中含有的通配符(*)以及子孙运算符(//)需要有效处理;

3)针对元素节点数据的过滤操作(指节点带谓词或带分支路径的情况)。

为此,研究者们在 XML 过滤技术的研究过程中引进了有限自动机的概念,用于高效地处理这些新问题。

2.2　分类比较

在已有的研究工作中,基于有限自动机的 XML 过滤技术主要分为两大类:

第 1 类是基于非确定型有限自动机(NFA,Nondeterministic Finite Automaton),主要特点是根据所有的 XPath 查询表达式构造相应的 DFA。非确定型有限自动机具有结构相对简单,状态空间比较小的优点。

第 2 类是基于确定型有限自动机(DFA,Deterministic Finite Automaton),主要特点是根据所有的 XPath 查询表达式构造相应的 DFA。确定型有限自动机具有状态转移效率很高因而过滤性能好的优点,而通常其状态空间也比较大,是一种典型的以空间换时间的方法。

表 1　2 类基于有限自动机的 XML 过滤技术的比较

类别	状态数目	内存要求	过滤性能	可伸缩性
基于 NFA	较小	较低	较差	较好
基于 DFA	较大,指数级	较高	较好	较差

3　基于非确定型有限自动机的 XML 过滤技术分析

基于非确定型有限自动机 XML 过滤技术的典型代表是 XFilter[5],YFilter[6,7]。

3.1　建立高效的查询索引

XFilter 是第一个采用有限自动机来进行 XML 数据流过滤的方法,它针对每一个路径表达式建立一个有限状态机 FSM(Finite State Mashine),同时采用在所有 FSMs 的状态转移符号上建立倒排索引的方法,加快过滤时的状态转移过程。

在 XFilter 中,每一个 XPath 查询都被 XPath 解析器分解成一个个的路径节点,并组成相应的节点集合。这些路径节点分别代表了原路径查询表达式中的元素节点并被看作该查询对应的 FSM 中的一个状态。对于查询表达式中的通配符"*"节点,则不需要产生一个相应的路径节点。

查询表达式被分解好了以后,还会被添加到一个查询索引中去,并构造哈希表,以该查询表达式中出现过的所有的元素名字作为键,而对应的值为两个列表:候选表(Candidate List)和等待表(Wait List)。这样的索引结构称为倒排索引(Inverted Index)。由于每个查询在同一时刻只能处于其相应的 FSM 中的某个状态,于是每个查询都利用其中的某个路径节点来表示当前的状态,这样的节点就称为当前节点,并且会被放置到以该节点对应的元素名称为入口的候选表中。所有表示了将来状态的路径节点都存放在以它们节点元素的名称为入口的等待表中,如图 2[5]所示。

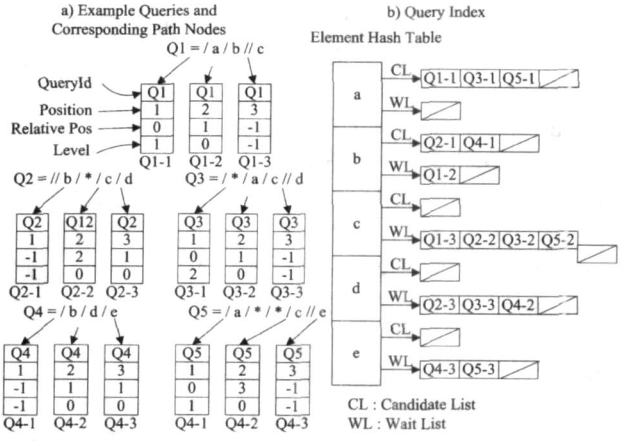

图 2　XFilter 查询索引

FSM 有限自动机的进行状态转移是一个事件驱动的过程,其事件流由基于 SAX 的 XML 解析器产生。这里仅讨论单一查询表达式的情况,即查询表达式不带分支路径的情况。

当遇到元素开始事件,过滤引擎会调用相应的元素开始事件处理模块,同时还会传入元素名字、元素所处的层次以及含有的属性值等值。模块根据元素名字作为哈希表入口在查询索引中查找并检查相应候选表中的所有路径节点,对于每个节点需要进行层次检查和属性检查。层次检查的目的是查看元素出现在文档中的层次是否满足相应查询表达式的要求;属性检查是查看元素的属性值是否满足查询表达式中的谓词条件。由于不存在其它的节点过滤器,所以如果这两种检查的结果都是满足要求的,则将当前路径节点所对应查询的下一个路径节点从等待表中复制到相应的候选表,表示转移到了下一个 FSM 状态;如果已经不存在下一个路径节点,则当前过滤文档是满足该查询的。

当遇到元素结束事件,过滤引擎将会从候选表中删除相应的路径节点,从而将各个查询表达式的 FSM 状态恢复至遇到该元素开始事件前的状态。

上面介绍的算法是比较直观的,XFilter 的作者给出了 2 种改进方法。

第 1 种是列表平衡方法(List Balancing)。其主要思想是对于层次值很小的元素,相应的文档选择性会比较小,因为此时对应的 XML 文档中的元素集合的基数相对比较小,这些元素在 XML 文档中出现的概率比较大。由这些元素引发的 FSM 状态转移过程对绝大多数的 XML 文档来说都是必然进行的,因此可以考虑将引发 FSM 状态转移的起始元素层次值增大。对小于该层次值的元素仅仅压入相应的栈中,直到出现了起始层次值以后的元素才开始驱动 FSM 的状态转移,并且只需要在最初的状态转移开始之前,检查一下堆栈中的元素是否都满足当前的查询表达式即可。如果满足,则进行正常的状态转移;如果不满足,则放弃状态转移。

第 2 种改进方法是进行预处理。其基本思想是对将要过滤的 XML 文档进行简单的预过滤,收集文档中所有出现过的元素并存放在一张称为出现表(Occurrence table)的哈希表中。预过滤结束后,那些含有某些节点的元素名字没有出现在出现表中的查询表达式就可以抛弃掉,从而在实际过滤的过程中不需要再检查这些查询表达式,提高了过滤的效率。

分析:XFilter 提出了一种有效的倒排索引结构来构造 XPath 的查询索引,同时提出了 2 种优化的方法,在文档结构较为简单(可以用文档平均深度值来衡量),查询的数量不是很多,并且查询的平均深度(这里指的查询表达式中路径节点的平均数目)也不是很深的时候,较好地解决了过滤 XML 文档与传统的关键字匹配技术相比较时存在的几个新问题。

缺点:1) 对于结构较为复杂的 XML 文档,XFilter 的性能较差,这主要是因为在进行预处理优化的时候,由于出现表变得很大,各元素出现的概率也增大了,因而可以提前抛弃的查询表达式数目减少了,而且随着文档平均深度的增加,处理带有子孙运算符"//"的查询表达式时需要进行状态转移的次数会大大增多,从而导致了过滤性能的下降。2) 在查询的数量增多,查询的平均深度增大的情况下,同时要处理的 FSMs 的数量增多了,过滤过程中需要进行状态转移的次数也大大增多,这种情况下 XFilter 的性能也显得比较差。3) XFilter 对每一个查询表达式建立一个 FSM,在对大量的 FSMs 构建索引时,没有很好地利用大量查询表达式之间具有的前缀相似的性质来简化索引结构和提高过滤的效率,导致了过滤效率不高。

YFilter 是在 FSM 基础上进行改进的,并提出了 NFA 索引结构,很好地解决了 XFilter 存在的问题。利用 NFA,可以将所有的查询表达式对应的 FSMs 全部合并到一个单一的有限自动机中,如图 3[7]所示。图中圆圈表示有限自动机的状态,箭头上的符号表示状态转移的触发条件。带阴影的状态是指被 2 个或 2 个以上的查询表达式共享的有限自动机状态。带双圆圈的状态表示一个接受状态,表示相应的查询表达式与当前过滤的 XML 文档相匹配。符号"*"表示通配符,即可以由任意的 XML 文档元素的元素开始事件驱动相应的状态转移;符号"ε"表示此处的状态转移的触发条件为空。可见,采用 NFA 索引结构,充分利用了大量 XPath 查询表达式之间存在的共享前缀的特性,可以显著地减少有限自动机的状态数目。

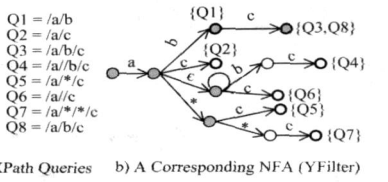

图 3 YFilter 查询索引

图 4[7]是 NFA 的一个执行的例子。左边是根据查询表达式集合构造出来的查询索引,即 NFA。每个 NFA 状态节点含有一个哈希表,节点左上方的数字表示 NFA 状态的 id,右边是 NFA 的运行栈,初始状态是 id 值为 1 的 NFA 状态。当遇到一个元素开始事件时,如元素〈a〉,根据运行栈里的处于栈顶的状态集,此时为{1},根据元素〈a〉、栈顶的状态集以及 NFA 查询索引可以计算得到新的状态集,并压入运行栈中。如果在新的状态集里含有接受状态,则说明存在某些查询与当前过滤的 XML 文档相匹配,此时还需要记录匹配结果。遇到新的元素开始事件时,采用同样的方法进行处理。而当遇到的是元素结束事件时,只需要简单地将运行栈中处于栈顶的状态集弹出即可,直到 XML 文档完成了过滤,运行栈重新变为空,等候下一个 XML 文档过滤操作的开始。

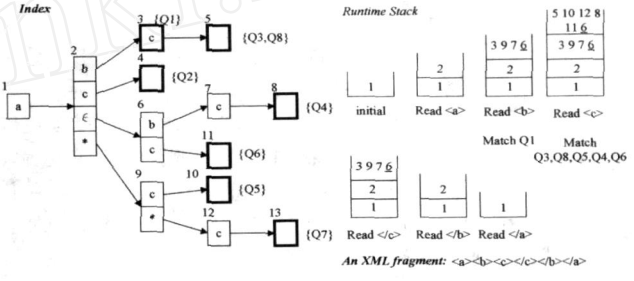

图 4 YFilter 执行例子

分析:该类型的有限自动机很好地利用了查询表达式之间前缀的相似性,即在构造查询索引的时候共享了查询表达式之间的相同的前缀,并将原本大量独立的查询索引全部合并成一个单一的 NFA,大大减少了有限自动机的状态数目和需要处理的有限自动机的数目(对于 NFA,需要处理的有限自动机的数目始终为 1 个),从而使得其过滤性能和第 1 类有限自动机相比有了较大的提高。此外,由于 NFA 的查询索引体积较小,在过滤过程中运行栈也并不记录运行栈中的历史状态集,所以所占用的内存也很小。

缺点:NFA 没有利用到 XML 文档数据流中各路径之间的相似性,所以文档结构不是很复杂。重复的 XML 元素和相似的 XML 路径比较多的情况下,必定会有大量相同的有限自动机状态转移的中间状态,即运行栈所压入的状态集被重复地进行计算了。特别是当 XML 文档的平均深度比较大的时候,其对应的运行栈所压入的状态集也会比较大,计算的代价也比较高,这在很大程度上影响了其过滤性能。

除了这两种典型方法之外,另有一些基于非确定型有限自动机的 XML 过滤技术。

XScan[8]和 XQRL[9]都是基于 FSMs 的过滤技术。和 XFilter 返回匹配的结果是真假值不同,XScan 中返回的匹配结果是相应的元组流。而 XQRL 主要是用于高效处理 XQuery[10]查询和由 XML Schema[11]定义的 XML 文档。

XTrie[12]是对基于 NFA 自动机的查询处理的一种扩展。

NFA 自动机每接受一个元素事件,都查找相关的转换,但是 XTrie 在接受元素事件序列之后,才选择相关的处理器进行响应。在对 XML 文档进行匹配时,不同序列中的共同部分只需要处理一次,从而可以提高效率。

苏[13]提出对输入的 XML 文档建立一个双索引结构来改进 YFilter 算法,优化 XML 文档过滤性能。利用此索引结构,该算法超前搜索元素节点在文档中的结构信息,预先排除不能保证得到任何匹配结果的元素节点,在处理祖先-后代关系("//")时可以避免大量不必要的查询处理。缺点是在过滤文档之前需要预先遍历文档以建立文档的索引,然后才可以进行过滤。

3.2 对带谓词和带分支路径查询表达式的支持

XFilter 中提出了节点过滤器的概念,处理带分支路径的查询表达式。该方法是在出现分支路径的主路径节点上添加节点过滤器,即根据分支路径构造相应的局部的 FSM,用于处理该分支路径的查询,并且对于分支路径再嵌套分支路径的情况也是适用的。这种局部的 FSM 只有在主路径满足一定的条件下才会触发其状态转移。在主路径上所有的节点过滤器返回值都为真,并且主路径也与当前过滤的 XML 文档相匹配时,该带分支路径的查询表达式才是确实与当前的 XML 文档相匹配的。

YFilter 中分别提出了 Inline 方法和 Selection Postponed (SP)方法来支持对带谓词查询表达式的处理,而对于带分支路径查询表达式则利用了 SP 方法来处理。

利用 Inline 方法处理带谓词的查询表达式时,直接在该查询表达式对应的所有 NFA 状态上检查所有的谓词表达式是否满足,并记录所有谓词检查的信息。当到达一个接受状态时,检查相应的查询表达式上所有的谓词表达式是否都满足。如果是,则认为该表达式是满足匹配的,否则丢弃该查询表达式。而利用 SP 方法处理带谓词的查询表达式时,是在最后到达接受状态的时候,才检查所有的谓词表达式是否都满足,因此需要在过滤的过程中,保留相应的历史的信息,用于批量检验谓词表达式。

利用 SP 方法对带分支路径查询表达式进行处理时,需要对分支路径建立相应的嵌套路径过滤器(Nested Path Filter),用于记录与分支路径相对应的 XML 元素节点信息。当查询表达式的主路径到达一个接受状态时,需要根据这些节点信息检查对应的分支路径是否都能正确匹配相应的 XML 文档路径,然后才能判断该带分支路径查询表达式是否确实与当前过滤的 XML 文档相匹配。

YFilter *[14]提出了一种聚集相似度函数,用于对 XPath 查询进行聚类,在降低匹配准确度的前提下,进一步增强了 YFilter 对分支查询的支持能力,过滤性能也有较大提高。

3.3 查询索引的更新

对于基于非确定型有限自动机的 XML 过滤技术,其查询索引的更新一般是一个增量维护的过程,因此查询索引的更新相对也比较简单。

3.4 分析比较表

表 2 基于非确定型有限自动机的 XML 过滤技术的比较

类别	过滤性能	分支查询的支持	查询索引更新	可伸缩性	预处理文档
XFilter[5]	低	×	中	差	×
YFilter[6,7]	中	√	简单	好	×
XScan[8]	高	√	中	中	×
SQRL[9]	高	√	中	中	×
XTrie[12]	中	√	中	中	×
苏[13]	高	√	简单	好	√
YFilter *[14]	高	√	中	好	×

4 基于确定型有限自动机的 XML 过滤技术分析

基于确定型有限自动机 XML 过滤技术的典型代表是 Lazy DFA[15]。

4.1 建立高效的查询索引

在 Lazy DFA 里,所有的查询表达式被转换到单一的 DFA 中,其步骤是:首先将所有的查询转换为一个单一的 NFA,然后根据 NFA 再转换为相应的 DFA。为了记录 DFA 状态转移的触发条件,每个 DFA 状态都维护一个状态转移哈希表。此外,还需要记录在该 DFA 状态下,有哪些查询表达式是和当前过滤的 XML 文档相匹配的,用于在过滤过程中进行结果收集。对于已经构造好的 DFA,用于过滤 XML 文档的时候,效率是非常高的。DFA 的状态转移和 NFA 以及 XFilter 的 FSMs 类似,也是由基于 SAX 的 XML 文档解析器产生的事件流来驱动的。在利用 DFA 过滤的过程中,仅需要维护一个运行栈和一个指向当前 DFA 状态的指针即可,其中运行栈中存放的是从初始状态到当前 DFA 状态之间、DFA 状态转移过程中的所有中间状态。当遇到一个元素开始事件时,过滤引擎就会根据当前过滤的元素节点的名字,查找当前 DFA 状态中的状态转移哈希表,得到目标状态。然后将当前的 DFA 状态压入运行栈中,并把目标状态置为当前 DFA 状态,最后根据需要进行结果收集。而在遇到一个元素结束事件时,仅需要将处于运行栈的栈顶的 DFA 状态弹出,并重新置为当前的 DFA 状态即可。可见,利用 DFA 进行 XML 文档的过滤,每次状态转移操作的代价和 NFA 相比是很小的,因此过滤效率非常高。

然而一般情况下,直接利用 DFA 进行 XML 文档的过滤是不现实的。因为在查询表达式集合很大的时候,直接转换成 DFA 时所产生 DFA 状态的数量呈指数级增长,因此很可能非常巨大,以至于无法将整个 DFA 都存放在内存中。解决的方法是在实际过滤的时候,采取根据需要计算并展开 DFA 状态的方法,减少了许多无用的 DFA 状态的展开,从而避免了 DFA 状态的爆炸式增长,只是对于新展开的 DFA 状态需要保留其相应的计算得到的 NFA 状态集,称为 NFA Table,用于根据已有的 DFA 状态来展开新的 DFA 状态。采用这个方法得到的 DFA 就称为 Lazy DFA。由于 DFA 状态是按需进行计算和展开的,在过滤过程中,Lazy DFA 可以分为预热阶段(Warmup Phase)和稳定阶段(Stable Phase)两个阶段。对于处在预热阶段的 Lazy DFA,过滤引擎的主要工作是计算并展开新的 DFA 状态,以及维护相应的 NFA Table,同时需要将相同的 DFA 状态合并,所以在这个阶段,其过滤性能往往比 NFA 要差。而在稳定阶段,绝对大部分需要用到的 DFA 状态都已经展开好了,Lazy DFA 过滤引擎的性能也接近或到达了极限,与 NFA 相比具有更高的过滤效率。

分析:基于确定型有限自动机的 XML 过滤技术的主要优点是按照实际需要计算和展开新的 DFA 状态并记录下来,对于大量已出现过的有限自动机状态不需要重新计算。因而在文档结构不是特别复杂以及查询集合也相对简单的情况下,重复利用到的 DFA 状态占了很大的比例,发挥了确定型有限自动机的优势,表现出了较高的过滤性能。

缺点:尽管 Lazy DFA 采用了按需展开 DFA 状态的方法,但仍然存在着状态空间暴涨的缺陷、在 XML 文档结构比较复杂或查询集合很大、很复杂的情况下,很容易出现内存耗尽的情况,严重影响了过滤性能;其次存在一个预热阶段,在这个阶段,Lazy DFA 的过滤性能会很差。

ONIZUKA[16]提出了两种优化 Lazy DFA 的方法。第 1 种是将查询表达式集合根据轴的类型("/"和"//")进行聚类,根据需要可以分成 $2n$ 个类,并在过滤过程中根据这些分类的集合分别构造相应的 Lazy DFA,得到 $2n$ 个 Lazy DFA,可以显著减少总的 DFA 状态数,避免了内存耗尽情况的出现,其代价是过滤引擎的效率只相当于原来仅有单一确定型有限自动机的过滤引擎的 $1/2n$ 倍。第 2 种优化的方法是共享已展开的 DFA 状态所对应的 NFA Table,从而达到减少内存使用量的目的。

CHEN[17]提出了增加一个预处理来减少需要展开的 DFA 状态的方法。作者假设在查询集合很大的情况下,直接应用相应的 DTD 来对查询集合进行预处理,可以抛弃掉一定比例的必然不可能满足的查询表达式,缩小查询集合的基数,从而达到减少最终需要展开的 DFA 状态数量的目的。

在 HE[18]中,作者提出了采用新的缓存策略来优化基于确定型有限自动机的 XML 过滤引擎的方法。基本思想是采用有效的缓存策略,使得在有限自动机进行状态转移时,可以提高 L2 Cache 的命中率。这是由于状态转移时需要查找新的有限自动机状态的入口,这时候很有可能会产生 L2 Cache 上的查找失效,而需要通过存取主存上的数据才能完成状态转移。相对于 L2 Cache 上的存取来说,其效率相对要差。因此提高 L2 Cache 上的查找命中率,可以有效地提高过滤引擎的性能。

4.2 对带谓词和带分支路径查询表达式的支持

XPush[19]中提出一种针对特殊类型的带谓词和带分支路径查询表达式的基于确定型有限自动机的处理方法,可以高效处理每个查询表达式平均带有 6～20 个谓词或分支路径表达式的情况。XPush 解决了 NFA 自动机的表达能力的问题。这一扩展主要利用了支持表达路径之间 AND/OR 关系的 AFA 自动机,AFA 自动机利用扩展的状态来保存不同路径的执行情况。XPush 执行器也是基于确定化自动机,提高了系统的查询效率,但是同样也面临着指数级别的确定化空间代价的问题。

类似于 XPush,高[20]提出了基于树自动机的 XEBT 机。XEBT 机基于表达能力丰富的树自动机,无须附加中间状态或保存中间结果,就能处理支持带谓词的 XPath;其次,XEBT 机支持多种优化策略,包括基于 DTD 的 XPath 查询自动机的构造;在空间代价有限增加的情况下采用局部确定化减少并发执行的状态;采用自上而下和自下而上相结合的查询处理策略。高[20]中的树自动机所占用的空间大多数情况下比 XPush 的小。

4.3 查询索引的更新

对于基于确定型有限自动机的 XML 过滤技术,其查询索引的更新是比较复杂的。

高[21]提出了基于转换累计自动机的查询执行器的增量维护方法,来完成查询集合的增量增加和删除,从而避免了执行器重构的昂贵代价。同时,利用 XML 文档和 DTD 结构约束,提高了查询执行器增量维护的效率,减少了增量维护所导致的空间冗余。

4.4 分析比较表

表 3 基于确定型有限自动机的 XML 过滤技术的比较

类别	过滤性能	分支查询的支持	查询索引更新	可伸缩性	预处理文档
Lazy DFA[15]	高	×	复杂	中	×
ONIZUKA[16]	高	×	中	高	×
CHEN[17]	中	×	复杂	高	×
HE[18]	高	×	复杂	高	×
XPush[19]	中	√	复杂	中	×
高[20]	高	√	复杂	中	×

结束语 基于有限自动机的 XML 过滤技术中,当前的研究以 NFA 和 DFA 为两大主流。NFA 具有结构简单,内存占用小的优点,而过滤效率则显得稍差;DFA 具有过滤效率很高的优点,但是内存占用往往过大,影响了过滤的性能。如何结合 NFA 和 DFA 的优点,在提高过滤效率的同时避免内存占用量的急剧上升,是未来的研究需要解决的问题。

目前,基于有限自动机的 XML 过滤技术通常仅支持由路径查询语言 XPath 的子集所描述的查询表达式集合。为了更加准确地描述用户的查询需求,增加对 XPath 的更为全面的支持是一个值得关注的研究方向。此外,由于互联网上信息日新月异,同时存在的用户数量巨大,因此用户对于信息检索和查询的需求的变化也会比较快。所以,在 XML 过滤技术研究中,根据用户的需求变化,如何实现高效的查询索引更新,是一个非常有意义的问题。

XML 过滤技术中,对于 XML 文档,以往的研究关注得比较多的是由 DTD 描述的 XML 文档,而 DTD 和 XML Schema 相比,其描述能力非常有限。研究由 XML Schema 描述的 XML 文档的过滤技术,有效支持结构更加复杂、数据类型更为丰富的 XML 文档的过滤,仍然是当前 XML 过滤技术研究领域的一项挑战。

参 考 文 献

[1] Bray T, Paoli J C, Sperberg-Mcqueen M, et al. Extensible Markup Language (XML) 1.0 (2nd Ed.) [S]. 2000 http://www.w3.org/TR/2000/REC-xml-20001006

[2] 孟小峰,周龙骧,王珊. 数据库技术发展趋势[J]. 软件学报,2004,15(12)

[3] Clark J, Derose S. XML Path Language (XPath) Version 1.0 [S], 1999. http://www.w3.org/TR/xpath

[4] 马建刚,黄涛,汪锦岭,等. 面向大规模分布式计算发布订阅系统核心技术[J]. 软件学报,2005,17(1)

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

[6] Diao Y, Fischer P, FranklinN M, et al. YFilter: Efficient and Scalable Filtering of XML Documents [C] //ICDE 2002

[7] Diao Y, Altinel M, Franklin M J, et al. Path sharing and predicate evaluation for high-performance XML filtering [J]. ACM Trans. on Database Systems (TODS), 2003, 28(4)

[8] Ives Z, Halevy A, Weld D. An XML query engine for network-bound data [J]. VLDB Journal, 2002, 11 (4)

[9] Florescu D, Hillery C. Kossmann D, et al. The BEA/XQRL streaming XQuery processor [C] //VLDB. 2003

[10] XML Query[S], 2002. http://www.w3.org/XML/Query

[11] XML Schema[S], 2001. http://www.w3.org/XML/Schema

[12] Chan C, Felber P, Garofalakis M N, et al. Efficient filtering of XML documents with XPath expressions [C] //ICDE. 2002

(下转第 47 页)

BA 性能的影响,随机选取三个无线传感器网络拓扑结构,每种拓扑结构都由 300 个传感器节点随机分布在 1200 × 1000m^2 大小的区域内,汇聚节点位于网络正中间。NNBA 中,第一隐层阈值 θ_1 和输出隐层的阈值 θ_2 依次选择 0,0.05,0.1,0.15,0.2,0.25 和 0.3。在 GROUP 协议中,网格宽度固定为 200m,簇头网格选举的周期为 20s。

图 7 显示了 NNBA 算法中主要参数第一隐层阈值 θ_1 和输出隐层的阈值 θ_2 对平均通信负载的影响。通过该曲线图,可以发现:(1) 随着第一隐层阈值 θ_1 和输出隐层的阈值 θ_2 的增加,平均通信负载逐渐减少,说明 NNBA 算法的数据融合效率越高;(2) 当输出隐层的阈值 $\theta_2 \geq 0.05$ 时,平均通信负载要明显低于 $\theta_2 \geq 0$ 时的平均通信负载;(3) 当 $\theta_2 \geq 0.05$ 时,θ_2 取不同值对平均通信负载影响不大,差异不明显,说明 θ_2 达到一定值后,对数据融合的效率影响不大;(4) 当 $\theta_1 \geq 0.2$ 且 $\theta_2 \geq 0.05$ 时,平均通信负载曲线近似平行于横轴,说明 θ_1 和 θ_2 取更大值对数据融合效率几乎已经没有影响。

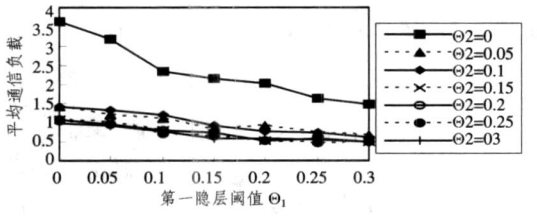

图 7 第一隐层阈值 θ_1 和输出隐层的阈值 θ_2 对平均通信负载的影响

虽然仿真测试结果表明,第一隐层阈值 θ_1 和输出隐层的阈值 θ_2 越大,可以获得越好的数据融合效率,但必须注意到,第一隐层阈值 θ_1 和输出隐层的阈值 θ_2 越大同时会对数据融合结果的真实性、有效性产生负面影响。另外,仿真测试结果还表明,太大的 θ_1 和 θ_2 对数据融合效率意义甚微。因此,在实际应用中,应该根据具体应用数据的特征,合理选择第一隐层阈值 θ_1 和输出隐层的阈值 θ_2 的取值,不仅要考虑到数据融合的效率,更要考虑到数据的有效性。

结束语 本文重点介绍了基于神经网络的数据融合模型(NNBA),该模型将多层次感知器模型与 GROUP 协议等分簇路由协议中簇结构进行了有机结合,可以利用各种神经网络算法实现对数据的融合处理。仿真测试的结果表明,NNBA 模型能够有效地节省传感器节点的能耗、延长网络寿命,并适用于多种无线传感器网络应用。

参 考 文 献

[1] Heidemann J, Silva F, Intanagonwiwat C, et al. Building efficient wireless sensor networks with low-level naming[C] // ACM SOSP. 2001

[2] Heinzelman W, Chandrakasan A, Balakrishnan H. Energy-efficient Communication Protocols for Wireless Microsensor Networks [C] // Hawaaian Int'l Conf. on Systems Science. 2000

[3] Yu L, Wang N, Zhang W, et al. GROUP: a Grid-clustering Routing Protocol for Wireless Sensor Networks [C] // IEEE International Conference on Wireless Communications, Networking and Mobile Computing. 2006

[4] Krishnamachari B, Estrin D, Wicker S. The Impact of Data Aggregation in Wireless Sensor Networks[C] // The 22nd International Conference on Distributed Computing Systems Workshops. 2002

[5] Intanagonwiwat C, Govindan R, Estrin D. Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks[C] // MOBICOM 2000. August 2000: 56-67

[6] Intanagonwiwat C, Estrin D, et al. Impact of Network Density on Data Aggregation in Wireless Sensor Networks [R]. Technical Report 01-750. University of Southern California, Computer Science Department, 2001

[7] Madden S, Franklin M J, et al. Tag: A Tiny Aggregation Service for Ad hoc Sensor Networks[C] // USENIX OSDI. 2002

[8] Arici T, Gedik B, Altunbasak Y, et al. PINCO: a Pipelined In-network Compression Scheme for Data Collection in Wireless Sensor Networks [C] // Proc. IEEE Int. Conf. Computer Communications and Networks. 2003

[9] Xu N, Rangwala S, Chintalapudi K K, et al. A wireless sensor network for structural monitoring[C] // Proc. of the 2nd Int'l Conf. on Embedded Networked Sensor Systems. New York: ACM Press, 2004

[10] Chou J, Petrovic D, Ramchandran K. Tracking and Exploiting Correlations in Dense Sensor Networks[C] // The Thirty-sixth Asilomar Conference on Systems and Computers. Nov. 2002

[11] Gao J, Guibas L, et al. Sparse Data Aggregation in Sensor Networks [C] // IPSN'07. April 2007

[12] Roedig U, Barroso A, Sreenan C J. Determination of Aggregation Points in Wireless Sensor Networks[C] // Proceedings of the 30th Euromicro Conference. 2004

[13] 尚峰,蒋国平,王芳. 应用 BP 网络构造复合型智能火灾探测器[J]. 自动化仪表,2003(03)

[14] 贾建华,王军峰,冯冬青. 人工神经网络在多传感器信息融合中的应用研究[J]. 微计算机信息,2006(07)

[15] NS-2 Network Simulator[EB/OL]. http://www.isi.edu/nsnam/ns/

(上接第 23 页)

[13] 苏明柿,张守志. XML 文档过滤算法 YFilter 的一种改进技术[J]. 计算机工程,2005,21(1)

[14] Zhang X, Yang L, Lee M, et al. Scaling SDI systems via query clustering and aggregation [C] // DASFAA 2004, LNCS 2973. 2004

[15] Green T, Gupta A, Miklau G, et al. Processing XML streams with deterministic automata and stream index [J]. ACM Trans. on Database Systems (TODS), 2004, 29(4)

[16] Onizuka M. Light-weight XPath processing of XML stream with deterministic automata [C] // CIKM. 2003

[17] Chen D, Wong R. Optimizing the lazy DFA approach for XML stream processing [C] // The Fifteenth Australasian Database Conference (ADC). 2004

[18] He B, Luo Q, Choi B. Cache-conscious automata for XML filtering [C] // ICDE. 2005

[19] Gupta A, Suciu D. Stream processing of XPath queries with predicates [C] // SIGMOD. 2003

[20] 高军,杨冬青,唐世渭,等. 基于树自动机的 XPath 在 XML 数据流上的高效执行[J]. 软件学报,2005,16(2)

[21] 高军,杨冬青,王腾蛟,等. 一种 XML 数据流之上持续查询执行器的增量维护方法[J]. 软件学报,2005,42(5)