基于分布式存储的正则表达式匹配算法设计与实现

李 璋[1] 杜慧敏[1] 张丽果[1,2]
(西安邮电大学电子工程学院微电子学系 西安 710061)[1] (西安电子科技大学微电子学院 西安 710071)[2]

摘 要 深度包检测(Deep Packet Inspection,DPI)是一种先进的包过滤技术,广泛应用于防火墙中。基于正则表达式的模式匹配算法已成为实现DPI的重要方法之一,传统的正则表达式匹配算法的时间复杂度较高,不能实时进行网络安全的防护。为了提高正则表达式匹配算法的实时性,提出一种基于分布式存储的正则表达式匹配并行算法,该算法通过对数据进行步长计数,采用分布式存储,实现了并行处理。实验分析表明,与传统的串行算法相比,该算法匹配速度至少提高5倍,能够有效降低算法时间复杂度,提高匹配效率。

关键词 正则表达式,模式匹配算法,确定型有限状态机,深度包检测,分布式存储
中图法分类号 TP338.6 **文献标识码** A

Design and Implement of Regular Expression Pattern Matching Algorithm Based on Distributed Storage

LI Zhang[1] DU Hui-min[1] ZHANG Li-guo[1,2]
(School of Electronic Engineering, Xi'an University of Posts & Telecommunications, Xi'an 710061, China)[1]
(School of Microelectronics, Xidian University, Xi'an 710071, China)[2]

Abstract Deep packet inspection is an advanced packet filter technology, widely used in the network firewall. Pattern matching algorithm based on regular expression has become one of the important methods in achieving DPI. Because the traditional regular expression matching algorithm has higher time complexity, it can not be used in real-time network security protection. In order to improve the real-time of regular expression matching algorithm, this paper proposed the parallel regular expression matches algorithm based on distributed storage. This algorithm counts step data, uses of the distributed storage, and realizes parallel processing. Experimental analysis shows that compared with the traditional serial algorithm, the proposed algorithm increases at least five times in matching speed, and can effectively reduce the time complexity of the algorithm, improve the matching efficiency.

Keywords Regular expression, Pattern matching algorithm, DFA (Deterministic finite automaton), DPI, Distributed storage

随着Internet的快速发展,网络安全面临着巨大挑战。DPI[1]是一种先进的包过滤技术,在分析包头的基础上,DPI对数据包的内容进行分析,可以发现、识别、分类、重新路由或阻止具有特殊数据或代码有效载荷的数据包。该技术越来越广泛地应用在防火墙中,可有效保护网络的安全。正则表达式RE(Regular Expression)[2,8,10]广泛地应用于文本编辑器或其他工具里,由于它具有强大而灵活的表达能力,因此基于正则表达式的模式匹配算法已经成为实现DPI的重要方法之一[3,9,10]。传统的正则表达式匹配方法是通过构建和运行一个确定的有限状态机DFA来实现的,其时间复杂度和空间复杂度很高[3-5]。针对该瓶颈,本文提出了一种易于硬件实现的基于分布式存储的正则表达式匹配算法。通过仿真可知,本文提出的算法与传统的串行算法相比,匹配速度至少提高了5倍,能够有效地降低算法的时间复杂度。

1 相关工作

传统的正则表达式匹配方法,主要集中在如何压缩DFA的存储空间。根据对现有算法的研究,按照压缩方式主要有转换压缩、状态压缩和字母表压缩[6]。文献[3]指出,传统的匹配方法,无论正则表达式如何复杂,都可以得到与每个输入字符有关的O(1)复杂度的执行时间,它采用的是串行的检测方式。然而,网络数据包的长度通常在40~1500字节(每个字符的ASCII码为一个字节)之间,那么按照每个字符O(1)复杂度的执行时间,传统的匹配算法完成一个具有n个字符数据包的执行时间复杂度为$O(n)$[9,10]。

从上面的工作可以看出,传统的正则表达式匹配方法只是研究如何压缩DFA的存储空间,而没有优化算法的执行时间。因此需要在传统的匹配算法的基础上,研究出能够解决时间复杂度的算法。

2 正则表达式并行匹配算法

2.1 相关概念

为了方便论述,本节介绍本文中用到的一些概念。

到稿日期:2012-12-29 返修日期:2013-01-02 本文受国家自然科学基金项目(60976020),陕西省教育厅科研计划项目(11JK1063,2010JK833)资助。

李 璋(1989-),男,硕士生,主要研究方向为通信系统与电路、网络安全,E-mail:lizhang_xy@126.com。

定义 1[7,10] 确定型有限自动机 DFA A 是一个五元组：$A=(\Sigma,SS,S_0,f,TS)$，其中：

1)Σ 是一个有穷字母表，其每个元素称为一个输入字符；

2)SS 是一个有穷集，它的每个元素称为一个状态；

3)$S_0\in SS$ 是唯一的一个初始状态；

4)f 是在 $SS\times\Sigma\to SS$ 上的转换函数；

5)$TS\subseteq SS$ 是一个终止状态集，又称为接受状态集。

一个 DFA 可以用一个有向的状态转移图表示[7]，在下面的论述中，不区分 DFA 和它的表示形式。

文献[7]指出，DFA 和正则表达式之间存在着如下的关系：一个正则表达式可以被一个具有 ε 转移的 NFA (Nondeterministic Finite Automata)所接受，而一个语言 L 如果可以被一个 NFA 所接受，那么必然可以被一个 DFA 所接受；如果 L 被一个 DFA 所接受，那么 L 可以用一个正则表达式表示，即每个 DFA 和一个正则表达式之间都存在一一对应的关系。

定义 2 $S=\{s_0,s_1,s_2,\cdots,s_k,1\leqslant k\leqslant n\}$ 是 DFA A 中状态集合 SS 的一个子集，s_0 为 A 的初始状态，s_k 为 A 的一个状态，称 S 为 s_0 到 s_k 的一条可达路径。$s_{k-1}\xrightarrow{i}s_k$ 表示 DFA 中 s_{k-1} 状态接收字符 i 转移到 s_k 状态。

在 DFA 的状态转移过程中，当前状态 s_{k-1} 转移到下一个状态 s_k 仅仅与当前所接收的字符有关，而与到达状态 s_{k-1} 前的历史状态无关。这个特性为设计匹配算法的并行化提供了基础。

定义 3 二元组$\langle k,i\rangle$定义为从 DFA 初始状态 s_0 开始，经过 $k-1$ 次转移并接收字符 i 后到达状态 s_k，即 $s_{k-1}\xrightarrow{i}s_k$，称二元组$\langle k,i\rangle$为 k 步接收字符 i。

定义 4 假设 A 是一个定义 1 给出的 DFA，$C_{\langle k,i\rangle}\in\Sigma$ 表示一个字符 i 的前趋字符集，s_k 是自动机经过 k 步运行后到达的状态，字符 $i\in\Sigma$ 是 A 经 $k-1$ 步后的输入字符，$C_{\langle k,i\rangle}$ 定义为：

$$C_{\langle k,i\rangle}=\begin{cases}\{c\mid s_l\xrightarrow{c}s_m\xrightarrow{i}s_k,\quad 0\leqslant l,m\leqslant n\}, & 2\leqslant k\leqslant n\\ \{\varepsilon\}, & k=1\end{cases}$$

下面通过一个例子来说明定义 4。图 1 表示了一个正则表达式. $*a[bce]+[0-9]d$ 与 DFA 之间的关系。

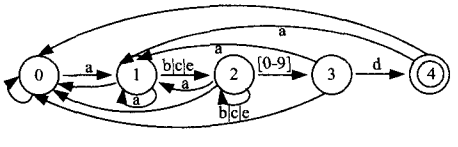

图 1 正则表达式. $*a[bce]+[0-9]d$ 的状态转移图

$C_{\langle 1,a\rangle}$ 表示字符 a 的前趋字符集，a 为 DFA 经过 1 步后接收的字符，由于 k 为 1，因此 $C_{\langle 1,a\rangle}=\{\varepsilon\}$。同理可得 $C_{\langle 1,b\rangle}=C_{\langle 1,c\rangle}=C_{\langle 1,d\rangle}=C_{\langle 1,e\rangle}=C_{\langle 1,0-9\rangle}=\{\varepsilon\}$；

$C_{\langle 2,a\rangle}$ 表示字符 a 的前趋字符集，a 为 DFA 经过 1 步运行后的输入字符，即 $C_{\langle 2,a\rangle}=\{c\mid s_0\xrightarrow{c}s_1\xrightarrow{a}s_2\}$，其转移路径有 $0\xrightarrow{a}1\xrightarrow{a}1,0\xrightarrow{\neg a}0\xrightarrow{a}1$，其中$\neg a$ 在该正则表达式中表示字符 b、c、e、d 以及$[0-9]$，因此 $C_{\langle 2,a\rangle}=\{a,\neg a\}$。

当 $k>2$ 时，各字符的前趋字符集以此类推。

定义 5 一个字符在字符串中的位置，定义为该字符在字符串中的步长。

例如，对于字符串 $abcdef$，字符 a 的步长为 1，字符 b 的步长为 2，字符 c 的步长为 3，字符 d、e 和 f 的步长以此类推。

2.2 分布式字符匹配算法

本文分布式匹配字符串的基本思想是：

(1)为 DFA 中每个字符 c 建立一个由三元组〈步长 k，前驱字符 p_c，状态转移 st〉表项构成的 T_c，$c\in\Sigma$，其中状态转移由〈当前状态→次态〉构成。

(2)将串行的输入字符进行串并转换，以便进行多个字符的并行处理。假设，一次可以并行处理 p 个字符。

(3)对 p 个并行字符，可以获知每个字符 c 的步长 k、前驱字符 p_c，将〈步长、前驱字符〉作为 c 所对应 T_c，$c\in\Sigma$ 的输入，进行查表。如果查到，则输出所有与〈步长、前驱字符〉相匹配的表项〈步长，前驱字符，状态转移〉。

(4)如果第 p 个字符的输出表项〈步长，前驱字符，状态转移〉中状态转移的次态有终态，则进入(5)；如果不是，则表示字符串不匹配，结束。

(5)对某个字符 c，如果查表结果中有多个表项输出，则需要确定字符 c 的唯一性。假设本次查找表的输入是〈k，p_c〉，获得的表项为〈k,p_c,st_1〉，〈k,p_c,st_2〉$\cdots$〈k,p_c,st_q〉。由于并行查找，因此可同时获取步长为 $k-1,k-2,\cdots1$ 所对应的所有表项，在这个表项中，完成下面的算法：

a)查步长为 $k-1$ 的字符的表项是否唯一，若唯一，则 goto b)，否则，继续查步长为 $k-2$ 的字符的表项是否唯一，直到查到步长为 $k_m(1\leqslant k_m\leqslant k)$的字符的表项唯一时，goto b)；

b)将得到唯一表项中状态转移的次态与下一步长的当前状态进行匹配，即上一个步长的次态与 该步长的当前状态相同，由此得到唯一的状态转移；

c)根据 b)，将所有不唯一的表项进行唯一化。

(6)检查经过唯一化后第 p 个字符的状态转移的次态是否为终态，若是，则匹配；否则，不匹配。

2.3 举例说明分布式字符匹配算法

2.3.1 存储

仍以图 1 为例来说明存储表的建立，该 DFA 中 $\Sigma=\{a,b,c,d,e,0-9\}$，$SS=\{0,1,2,3,4\}$，$S_0=\{0\}$，$TS=\{4\}$，首先为 Σ 中的每个字母建立存储表。由于字符 b、c、e，字符$[0-9]$具有相同的状态转移，故将字符 b、c、e，字符$[0-9]$当作一个整体进行存储，其所对应的存储表如表 1 所列。之所以 T_a 中步长为 3 的前趋字符 b 出现 2 次，是因为 T_b 中步长为 2 的状态转移的次态有 2 个，而 2 个不同的状态接收字符 a 后得到不同的状态。其他表项也是同样的道理。当步长大于 5 后，后续的表项与步长为 5 的表项一致，这是由于 DFA 状态是有限的，因此，在步长大于状态数目后，必然是重复运行的。这样，不必为每一个步长都进行存储，只需存储到最小重复步长 $k_{\min}$ 即可。此例中 $k_{\min}=5$。

表1　正则表达式. * a[bce]+[0−9]d 的存储表

T_a			$T_{b\|c\|e}$			T_d			$T_{[0-9]}$		
步长	前趋字符	状态转移	步长	前趋字符	状态转移	步长	前趋字符	状态转移	步长	前趋字符	状态转移
1	ε	0→1	1	ε	0→0	1	ε	0→0	1	ε	0→0
2	a	1→1	2	a	1→2	2	a	1→0	2	a	1→0
	→a	0→1		→a	0→0		→a	0→0		→a	0→0
3	a	1→1	3	a	1→2	3	a	1→0	3	a	1→0
	b|c|e	2→1		b|c|e	2→2		b|c|e	2→0		b|c|e	2→3
	b|c|e	0→1		b|c|e	0→0		b|c|e	0→0		b|c|e	0→0
	d	0→1		d	0→0		d	0→0		d	0→0
	[0−9]	0→1		[0−9]	0→0		[0−9]	0→0		[0−9]	0→0
4	a	1→1	4	a	1→2	4	a	1→0	4	a	1→0
	b|c|e	2→1		b|c|e	2→2		b|c|e	2→0		b|c|e	2→3
	b|c|e	0→1		b|c|e	0→0		b|c|e	0→0		b|c|e	0→0
	d	0→1		d	0→0		d	0→0		d	0→0
	[0−9]	3→1		[0−9]	3→0		[0−9]	3→4		[0−9]	3→0
	[0−9]	0→1		[0−9]	0→0		[0−9]	0→0		[0−9]	0→0
5	a	1→1	5	a	1→2	5	a	1→0	5	a	1→0
	b|c|e	2→1		b|c|e	2→2		b|c|e	2→0		b|c|e	2→3
	b|c|e	0→1		b|c|e	0→0		b|c|e	0→0		b|c|e	0→0
	d	0→1		d	0→0		d	0→0		d	0→0
	d	4→1		d	4→0		d	4→0		d	4→0
	[0−9]	3→1		[0−9]	3→0		[0−9]	3→4		[0−9]	3→0
	[0−9]	0→1		[0−9]	0→0		[0−9]	0→0		[0−9]	0→0

2.3.2　存储表的查找及匹配

下面以上述例子来说明该算法的匹配过程。假设待检测的并行数据为 *a1dabc2ae9d*，可以得到各个字符的步长以及前趋字符。比如，在第一个字符 *a* 时，其步长为 1，前趋字符为空字符 ε。对其中每个字符 *c*，将得到的〈步长、前驱字符〉作为 *c* 所对应表 T_c 的输入，查表 1，所得的结果见表 2，并将其状态转移按照步长的大小由小到大排列见表 3，再按照 2.2 节(5)进行每个字符 *c* 的状态转移的唯一化，见表 4。

表2　输入为 *a1dabc2ae9d* 时得到的表项

T_a			$T_{b\|c\|e}$			T_d			$T_{[0-9]}$		
步长	前趋字符	状态转移	步长	前趋字符	状态转移	步长	前趋字符	状态转移	步长	前趋字符	状态转移
1	ε	0→1	5	a	1→2	3	1	0→0	2	a	1→0
4	d	0→1	6	b	2→2 0→0	11	9	3→4 0→0	7	c	2→3 0→0
8	2	3→1 0→1	9	a	1→a				10	e	2→3 0→0

表3　按步长重排表 2

步长	前趋字符	状态转移
1	ε	0→1
2	a	1→0
3	1	0→0
4	d	0→1
5	a	1→2
6	b	2→2　0→0
7	c	2→3　0→0
8	2	3→1　0→1
9	a	1→2
10	e	2→3　0→0
11	9	3→4　0→0

在步长为 11 的字符 *d* 中，由于状态转移的次态是 4，而状态 4 是终态，因此字符串 *a1dabc2ae9d* 被正则表达式. * *a*[*bce*]+[0−9]*d* 所接受。

表4　表 3 的唯一化

步长	前趋字符	状态转移
1	ε	0→1
2	a	1→0
3	1	0→0
4	d	0→1
5	a	1→2
6	b	2→2
7	c	2→3
8	2	3→1
9	a	1→2
10	e	2→3
11	9	3→4

3　分布式字符匹配算法的硬件结构

分布式字符匹配算法的硬件结构如图 2 所示。

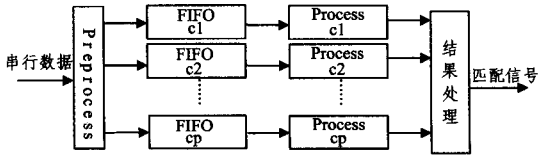

图2　分布式字符匹配算法的硬件结构

下面对各个模块的功能进行描述：

Preprocess 模块：首先将输入的串行数据转化为 *p* 位并行数据。然后 *p* 个字符按字符类别进行划分，得到每个字符的步长以及前趋字符。

FIFO 模块：将 Preprocess 模块输出字符的步长以及前趋字符进行缓存。在 2.3.2 节的例子中，从 Preprocess 模块得到字符 *a* 步长以及前趋字符有 3 个，分别是〈1,ε〉、〈4,d〉以及〈8,2〉，所以需要进行缓存。

Process 模块：把 FIFO 模块输出的数据输入至 Process 模块，从而得到相对应的状态转移。

结果处理模块：通过 Process 模块输出的状态转移过程，按照 2.2 节(5)判断该输入字符串是否匹配该正则表达式。

(下转第 99 页)

注这两个方面。此外，pR 的实现思路比较好，值得进一步关注。

参 考 文 献

[1] 赵毅，朱鹏，迟学斌，等. 浅析高性能计算应用的需求与发展[J]. 计算机研究与发展，2007，44(10)：1640-1646

[2] 迟学斌，赵毅. 高性能计算技术及其应用[J]. 中国科学院院刊，2007，22(4)：306-313

[3] Development R，Team C R. A language and environment for statistical computing[R]. R Foundation for Statistical Computing，Vienna，Austria，2012

[4] http://www.mathworks.com/products/matlab

[5] http://www.maplesoft.com

[6] http://www.sas.com

[7] http://www.ibm.com/software/analytics/spss/

[8] http://spotfire.tibco.com/Products/S-Plus-Overview.aspx

[9] http://stat.bell-labs.com/S/

[10] Vance A. R You Ready for R? [OL]. http://bits.blogs.nytimes.com/2009/01/08/r-you-ready-for-r)

[11] Vance A. Data Analysts Captivated By R's Power[OL]. http://www.nytimes.com/2009/01/07/technology/business-computing/07program.html

[12] http://www.schemers.org

[13] R CRAN[Z]. http://cran.r-project.org/

[14] http://www.Bioconductor.org

[15] Kepner J. HPC Productivity：An Overarching View[J]. International Journal of High Performance Computing Applications：Special Issue on HPC Productivity，2004，18(4)：393-397

[16] Cran R. Task View：High Performance and Parallel Computing with R[OL]. http://cran.r-project.org/web/views/HighPerformanceComputing.html

[17] Schmidberger M，et al. State of the Art in Parallel Computing with R[J]. Journal of Statistical Software，2009，31(1)：1-27

[18] 黄铠，徐志伟. 可扩展并行计算技术、结构与编程[M]. 北京：机械工业出版社，2000

[19] Breimyer P，et al. pR：Lightweight，Easy-to-Use Middleware to Plugin Parallel Analytical Computing with R[C]// IKE 2009. 2009：667-673

[20] Breimyer P，et al. pR：Automatic parallelization of data-parallel statistical computing codes for R in hybrid multi-node and multi-core environments[C]// IADIS AC(2). 2009：28-32

[21] Shah N，et al. pR：Enabling Automatic Parallelization of Data-Parallel Tasks and Interfacing Parallel Computing Libraries in R with Application to Fusion Reaction Simulations[C]// The !R User Conference 2010. Gaithersburg，Maryland，USA，2010

[22] Li Jiang-tian，et al. Transparent runtime parallelization of the R scripting language[J]. Journal of Parallel and Distributed Computing(JPDC)，2011，71(2)：157-168

(上接第 76 页)

由分布式字符匹配算法的硬件结构可知：与传统的串行匹配算法相比较，本文提出的分布式匹配算法可以实现并行处理，能够大大地降低算法的时间复杂度，使算法的匹配效率更高。

4 实验及分析

实验采用的硬件平台为 Xilinx ISE 13.2，基于 Xilinx 系列 Virtex5 型号的 FPGA 器件进行模拟仿真。

网络数据包的长度通常在 40～1500 字节之间，因此取不同长度的数据包进行验证。对于不同数据包长度下执行时间的测试，实验采用了 snort 系统 2012 年 3 月注册版的规则集[11]，从中提取 100 条正则表达式，每条正则表达式随机取 50 组数据包进行验证，得到在本文算法下执行时间的平均值，并与传统的算法进行比较，见表 5，表 5 中数据包长度即为 p 值。由表 5 可知，数据包长度越长，本算法的执行效率越高。

表 5 不同数据包长度执行时间

数据包长度（字节）	传统算法执行时间（时钟周期）	本文算法执行时间（时钟周期）	效率提高倍数
40	40	8	5.00
100	100	14	7.14
150	150	17	8.82
300	300	13	13.04
500	500	27	18.53
1000	1000	30	33.33
1500	1500	37	40.54

结束语 本文提出了一种基于分布式存储的正则表达式匹配算法，它能够有效地解决匹配过程中的时间复杂度问题，提高了匹配效率。实验表明，相比传统的匹配算法，该算法在处理时间上至少可以提高 5 倍以上，可以有效地实现在高速、大容量的 Internet 上对网络入侵的实时检测。目前国内外提出的算法都是针对如何压缩 DFA 空间的，尚未提出关于降低时间复杂度的算法，本文提出的算法可以实现并行处理，有效地降低了时间复杂度。下一步工作将考虑采用多核结构继续优化该算法，以进一步提高算法性能。

参 考 文 献

[1] 锐捷网络. DPI 技术白皮书[OL]. http://wenku.baidu.com/view/aa73eac66137ee06ef f91879.html，2009-03

[2] 邓凯元，姜磊. 正则表达式匹配引擎性能分析[J]. 计算机与现代化，2011，30(07)：105-107

[3] 张洁坤. 时空高效的正则表达式匹配算法研究[D]. 长沙：湖南大学，2010

[4] 刘俊超，赵国鸿，陈曙晖. 一种用于深度报文检测的 DFA 状态表压缩方法[J]. 计算机工程与应用，2008，44(22)：74-76

[5] 杨毅夫，刘燕兵，刘萍，等. 正则表达式的 DFA 压缩算法[J]. 通信学报，2009，30(10A)：36-42

[6] 姚远，刘鹏，单征，等. 面向存储的正则表达式匹配算法综述[J]. 计算机应用，2009，29(12)：3171-3173

[7] 刘胤. 深度包检测技术的研究与设计[D]. 贵阳：贵州大学，2008

[8] Sidhu R，Prasanna V K. Fast Regular Expression Matching using FPGAs[C]// The 9th Annual IEEE Symposium on Field-Programmable Custom Computing Machines. USA：Rohnert Park，California，2001：227-238

[9] Song T，Zhang W，Wang D S，et al. A memory efficient multiple pattern matching architecture for network[C]// Proceedings of the IEEE INFOCOM 2008. USA：Phoenix，2008：166-170

[10] Domenico F，Stefano G，GregorioP，et al. An improved DFA for fast regular expression matching[J]. ACM SIGCOMM Computer Communication Review，2008，38(5)：29-40

[11] Introduction to Snort [OL]. http://www.snort.org/docs/，2012-03-21