Journal of Computer Applications
计算机应用,2013,33(8):2370-2374,2382
ISSN 1001-9081
CODEN JYIIDU
2013-08-01
http://www.joca.cn

文章编号:1001-9081(2013)08-2370-05
doi:10.11772/j.issn.1001-9081.2013.08.2370

基于多字符 DFA 的高速正则表达式匹配算法

贺　炜*,郭云飞,莫　涵,扈红超
(国家数字交换系统工程技术研究中心,郑州 450002)
(*通信作者电子邮箱 hewxiaoyao@163.com)

摘　要:基于确定性有限自动机(DFA)的传统正则表达式匹配方法存在单周期处理单字符的速度瓶颈。为提升处理速率,提出一种单周期处理多字符的匹配算法 MC-DFA,该算法基于 DFA 实现,支持匹配位置的精确定位。MC-DFA 将传统 DFA 中的单字符跳转合并为多字符跳转,实现了单周期处理多个输入字符。通过状态转移矩阵二阶压缩算法,MC-DFA 分别对矩阵行内以及行间冗余进行消除,减少了内存使用。300 条规则下,单周期处理 8 字符时,MC-DFA 吞吐率能够达到 7.88 Gb/s,内存占用小于 6 MB,预处理时间为 19.24 s。实验结果表明,MC-DFA 能够有效提升系统吞吐率,并且保证内存占用在可接受范围之内,性能优于现有正则表达式匹配算法。

关键词:正则表达式;高速;多字符;精确定位;矩阵压缩
中图分类号:TP393　　**文献标志码**:A

Multi-character DFA-based high speed regular expression matching algorithm

HE Wei*, GUO Yunfei, MO Han, HU Hongchao
(*China National Digital Switching System Engineering and Technological R&D Center, Zhengzhou Henan* 450002, *China*)

Abstract: Traditional Deterministic Finite Automata (DFA) based regular expression matching can only process one character per cycle, which is the speed bottleneck. A new algorithm named Multi-Character DFA (MC-DFA) was proposed for high throughput matching and precise positioning. It combined the one character transition in traditional DFA together to handle multi-character processing per cycle. A new transition matrix compress algorithm was also proposed to reduce the redundancy introduced by MC-DFA. The result demonstrates that MC-DFA can improve the throughput efficiently while requiring acceptable memory. For a set of 300 regexes, 8C-DFA obtains a throughput of 7.88 Gb/s, memory usage less than 6 MB and 19.24 s preprocessing time, better than traditional methods.

Key words: regular expression; high throughput; multi-character; precise positioning; matrix compress

0　引言

近年来,作为网络信息过滤、文字处理等应用中的关键部分,正则表达式匹配技术得到了快速发展,作为实现模式匹配的主要手段,广泛应用于二进制序列分析、扩展标记语言处理及入侵检测系统[1-2]。

目前正则表达式匹配主要通过不确定性有限自动机(Nondeterministic Finite Automata,NFA)以及确定性有限自动机(Deterministic Finite Automata,DFA)实现[3]。NFA 的缺点在于时间复杂度比较高,对于有 n 个状态的 NFA,空间和时间复杂度均为 $O(n)$;DFA 处理一个字符则只需要访问一个状态,时间复杂度降低为 $O(1)$,但空间复杂度增加为 $O(2^n)$[4-5]。

由于理论模型限制,每个周期内,标准 NFA 和 DFA 最快只能处理一个字符,成为速度瓶颈[6-7]。为进一步提升匹配速率,当前众多研究都集中于如何构建支持单周期多字符处理的自动机结构。

本文提出一种基于 DFA 的单周期多字符处理算法 MC-DFA(Multi-Character DFA),利用了多字符处理的优势以及 DFA 时间复杂度低的特点,提升了系统处理速率,减少了内存访问次数。该算法支持每个周期处理 2^k 个字符(k 为任意自然数),并加入内存压缩技术,以减少内存消耗。通过现场可编程门阵列(Field Programmable Gate Array, FPGA)平台部署,算法的处理速率以及内存使用等性能得到了验证。

1　相关工作

传统的单字符匹配算法中,Liu 等[8]提出的 CSCA(Cluster-based Splitting Compression Algorithm)通过切分状态转移矩阵减少内存使用;Yang 等[9]提出的 SFA(Semi-deterministic Finite Automata)算法通过将部分 DFA 转化为 NFA 实现内存缩减;Qi 等[10]提出的 FEACAN(Front-End Acceleration for Content-Aware Network)算法通过压缩状态转移矩阵取得较好的内存性能。

Clark 等[11]给出了一种基于多字符偏移并行处理的 NFA 结构。对于 n 位的目标字符串,首先求得偏移 0 到 $n-1$ 位的 n 个字符串,将 n 个字符串送入 n 个相同的并行处理的 NFA 结构,得到匹配结果;该算法缺点在于多个 NFA 的复制导致内存消耗过大。

Sutton 等[12]给出了基于字符编码的多字符 NFA 匹配在 FPGA 中的实现方法,该算法在 Clark 等[11]提出的算法基础

收稿日期:2013-02-20;**修回日期**:2013-03-25。
基金项目:国家科技支撑计划项目(2012BAH02B03,2012BAH02B01);国家 863 计划项目(2011AA01A103,2011AA01A101,2011BAH19B04)。
作者简介:贺炜(1988-),男,山西吕梁人,硕士研究生,主要研究方向:网络安全中的特征匹配;　郭云飞(1963-),男,河南郑州人,教授,博士生导师,主要研究方向:三网融合;　莫涵(1988-),女,河南郑州人,硕士研究生,主要研究方向:可重构网络、组播传输;　扈红超(1983-),男,河南郑州人,研究员,博士,主要研究方向:网络协议分析。

《计算机应用》Journal of Computer Applications

上,将并行处理的 NFA 结构修改为串行处理的寄存器结构,避免了 NFA 的多次复制;但随着字符串长度的增加,系统处理延时增大,无法保证处理频率,影响吞吐率性能。

在 Sutton 等[12]的基础上, Yamagaki 等[13]给出了任意正则表达式到多字符 NFA 结构的构建方法,与之前的方法相比, Yamagaki 等[13]最大的突破在于实现了单个 NFA 结构处理多字符输入的功能,但是处理速率仍然受到 NFA 本身特性的影响,且算法依赖于硬件实现。

由于 NFA 时间复杂度较高,上述基于 NFA 的方法不能很好地改善系统吞吐率。为进一步提升处理速率,克服 NFA 的缺点,本文采用时间复杂度较低的 DFA 作为结构基础。

2　MC-DFA 算法

MC-DFA 算法基于 DFA 结构实现,该算法支持在单周期内处理多个输入字符(输入字符数为 2 的倍数),能够实现匹配位置的精确定位。

为便于算法描述,定义 DFA 数学表述如下:

定义 1　对于正则表达式 R, DFA 是确定性有限自动机 $M=(Q,\Sigma,\delta,q_0,F)$,其中: Q 是有限的 DFA 状态集合, Σ 是有限的字母表, δ 是确定性状态转移函数, q_0 是初始状态, F 是状态机结束状态集合。

实现 MC-DFA 之前,采用最小 DFA 构造法将正则表达式转换为支持单周期处理单字符的 DFA 结构(1 Character-DFA,1C-DFA),在此基础上,MC-DFA 的构建过程主要分为两个步骤:

1)将 1C-DFA 转换为单周期可处理 2^k 个字符的 DFA 结构(2^kC-DFA)。转换过程中,本文提出一种输入字符双倍处理算法(Input Character Double Algorithm,ICDA)。

2)压缩 2^kC-DFA 的状态转移矩阵,提升内存使用效率。为减少状态转移矩阵的内存占用,本文提出一种转移矩阵二阶压缩算法,可达到较高的压缩效率。

下面对 ICDA 以及转移矩阵二阶压缩算法进行详细介绍。

2.1　输入字符双倍处理算法

2.1.1　算法原理

ICDA 用于 2^kC-DFA 的构建。输入为单周期处理 $n(n=1,2,3,\cdots)$ 个字符的 DFA 结构 nC-DFA,输出为单周期处理 $2n$ 个字符的 DFA 结构 $2n$C-DFA。

ICDA 通过将 nC-DFA 中每个状态的入跳转和出跳转进行合并,使得入跳转直接到达下一个状态,达到处理字符数翻倍的目的。

2^kC-DFA 可通过对 1C-DFA 循环执行 k 次 ICDA 得到。ICDA 的伪代码如下:

算法 1　ICDA。

输入:DFA 结构, nC-DFA $(n=1,2,3,\cdots)$;

输出:DFA 结构, $2n$C-DFA。

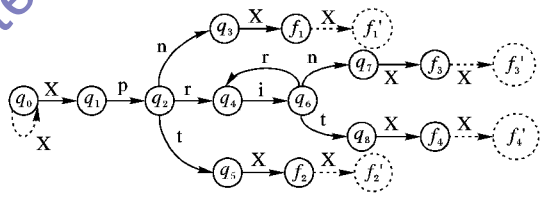
```
1)  for all states q ∈ Q
2)    if q = q0 then                          //初始状态添加自环转移
3)      add self loop transition (q,q), labeld "X"
4)    elseif q ∈ F then                       //增加结束状态
5)      add a new state q',q' ∈ F
6)      add transition from q to q', record as (q,q'), labeld "X"
7)    end if
8)  end for
9)  for all states q1 ∈ Q                     //合并入跳转和出跳转
10)   for all states q2 ∈ states have transition to q1
11)     for all states q3 ∈ states have transition from q1
12)       if transiton (q1,q3) ≠ (q,q)
13)         add new transition (q2,q3)
14)         combine labels of (q2,q1) (q1,q3) and resultin
            'NewLabel'
15)         add label 'NewLabel' to (q2,q3)
                                            //设立新转移标签
16) delete (q,q) and (q,q')                 //删除原始跳转
17) delete nC-DFA transitions
```

2.1.2　算法举例

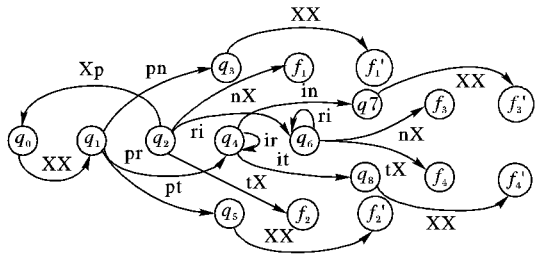
下面以正则表达式"p(ri)*(n|t)"为例,举例说明 ICDA 执行过程。

图 1 中,(a)给出执行伪代码 1)~8)行后的 DFA 结构图。其中, q_0 表示初始状态; f_1,f_2,f_3,f_4 表示 4 个结束状态;其余的 q_1 到 q_8 表示 8 个中间状态;箭头表示状态转移方向,箭头上方的字符表示接收的输入字符,"X"表示任意跳转。与原始 DFA 结构图相比,增加了两部分内容(图中虚线部分显示):一是初始状态的自环,跳转标记为"X";二是对每个结束状态各添加一个结束状态,记为 f_1', f_2', f_3', f_4',与 f_1, f_2, f_3, f_4 一一对应,相应的跳转标记为"X"。

图(b)给出执行一次 ICDA 后,DFA 结构的最终状态。该结构每个周期可以处理两个输入字符。如果完成匹配时结束状态是 f_1,f_2,f_3,f_4 中之一,则匹配位置在输入双字符的前一个字符位置;如果结束状态是 f_1',f_2',f_3',f_4' 中之一,则匹配位置在后一个字符处,实现了匹配位置的精确定位。



(a) 添加结束状态



(b) 算法最终结果

图 1　ICDA 构造双字符 DFA 结构示例

2.1.3　算法复杂度

观察图 1 可得,合并前后状态数有少许增长,状态跳转数量增长较多。下面给出 ICDA 对状态数以及状态跳转数量影响的三个定理以及相应证明。

为分析 DFA 结构的规模,定义相关参数如表 1 所示。

表 1　相关参数定义

符号	定义
ISN (Initial State Num)	初始状态数量
FSN (Final State Num)	结束状态数量
MSN (Middle State Num)	中间状态数量
TSN (Totle State Num)	DFA 总状态数
TN (Transition Num)	DFA 中状态跳转的数量

定理 1　ICDA 只改变 *FSN*。

Journal of Computer Applications 计算机应用

证明 根据算法伪代码表述,1)~8)行实现两个功能:a)增加了初始状态的自环;b)增加了一倍的 *FSN* 以及结束状态之间的跳转;这两个功能不改变 *ISN* 和 *MSN*。9)~15)修改各个状态间跳转关系,只改变跳转关系,不改变状态数。16)、17)行除原始跳转关系,不改变状态数。

综上所述,ICDA 不改变 *ISN* 和 *MSN*,但 *FSN* 增加一倍。

定理 2 对于常用规则集,执行 k 次 ICDA,*TSN* 增加不超过 1.07^k 倍。

证明 根据定理 1,分析 ICDA 带来的状态数改变,只需分析结束状态数的变化。

通过对大量正则表达式对应的 DFA 结构进行统计分析,结束状态在 DFA 总状态数中所占比例不超过 7%,即

$$FSN \leqslant 0.07 * TSN \tag{1}$$

数据来源在第 3 章仿真实验中给出,部分统计数据如表 2 所示。

设执行一次 ICDA 之前,各参数值为:TSN_1,FSN_1,ISN_1,MSN_1;执行一次 ICDA 之后,各参数值为:TSN_2,FSN_2。根据定理 1 以及式(1)可得:

$$TSN_2 = ISN_1 + MSN_1 + FSN_2 = ISN_1 + MSN_1 + 2 * FSN_1 = TSN_1 + FSN_1 \leqslant 1.07 * TSN_1 \tag{2}$$

执行 k 次 ICDA 后,$TSN_k \leqslant 1.07^k * TSN_1$。

表 2 DFA 数据统计表

规则名称	规则数	单条规则平均字符数	*TSN*	*FSN*	*TN*	$\frac{FSN}{TSN}$/%	*TN*/*TSN*
snort24	24	50	8 334	437	112 559	5.240	13.5
snort31	31	76	10 328	512	116 706	4.960	11.3
snort34	34	51	9 753	361	130 846	3.700	13.4
bro217	217	38	6 533	437	90 703	6.680	9.3
exact-math	300	55	111 438	447	1 203 530	0.410	10.8
ranges1	300	57	158 049	428	1 438 245	0.270	9.1
ranges5	300	56	165 786	496	1 558 388	0.299	9.4
tcp_homenet	733	47	241 587	956	3 116 472	0.390	12.9
dotstar0.3	300	56	145 239	578	1 220 007	0.390	8.4
dotstar0.6	300	59	164 172	602	1 510 382	0.600	9.2
dotstar0.9	300	59	183 327	594	1 631 610	0.320	8.9

定理 3 最坏情况下,执行 k 次 ICDA 后,$TN_{k+1} = \prod_{i=1}^{k}(TSN_i - 1) \cdot TN_1$。

证明 设执行 ICDA 之前,DFA 参数为:$TSN_1 = n$,$TN_1 = m$。最坏情况是指,任意两个状态间都有跳转关系,同时各个状态都存在自环跳转,则共有 n^2 条跳转关系,即 $TN_1 = m = n^2$。

ICDA 是通过将一个状态的入跳转和出跳转合并达到处理双倍字符的效果。对于 n 个状态,n^2 条跳转的 DFA,每个状态有 n 个入跳转,n 个出跳转,执行 ICDA 合并后共有 $n^3 - n^2$ 个跳转,变为原来的 $(n^3 - n^2)/n^2 = n - 1$ 倍。即执行一次 ICDA,$TN_2 = (n - 1) \cdot TN_1$;执行 k 次 ICDA 后,根据数学归纳法,$TN_{k+1} = \prod_{i=1}^{k}(TSN_i - 1) \cdot TN_1$,其中 $TSN_1 = n$。

引理 1 普通规则集下,执行 k 次 ICDA 后,$TN_k \leqslant 14^k \cdot TN_1$。

证明 如表 2 所示,根据统计结果,常见规则集构成的 DFA 中 $TN \leqslant 15TSN$。根据定理 3 中计算方式,执行 k 次 ICDA 后 $TN_k \leqslant 14^k \cdot TN_1$,远小于最坏情况下增长速率。

2.2 二阶跳转矩阵压缩

正则表达式转化为 DFA 后,状态转移存储于状态转移表(State Transition Table,STT)中,***STT*** 是二维矩阵,其有效利用率 *R_STT* 定义为:

$$R\_STT = \frac{\textbf{\textit{STT}}\text{ 内不同元素个数}}{\textbf{\textit{STT}}\text{ 大小}}$$

(3)根据 *R_STT* 定义及引理 1 可得:

引理 2 执行 ICDA 后,***STT*** 增大,*R_STT* 减小。

证明 每执行一次 ICDA,根据定理 2,状态数最多增加 1.07 倍,即不同元素个数增长 1.07 倍;根据引理 1,状态跳转数增加 14 倍,即 ***STT*** 增长 14 × 1.07 倍;根据式(3),*R_STT* 减小为原来的 1/14。

R_STT 的降低为压缩 ***STT*** 提供了理论依据。本文根据 *R_STT* 减小的特点,提出了跳转矩阵二阶压缩算法。

通过观察,***STT*** 是稀疏矩阵,每一行、每一列中都存在连续相同的数据。

对于行向量 $\boldsymbol{R}_1$,$\boldsymbol{R}_2$,定义行向量相似度(Row Relation Ratio,RRR)如下:

$$sim_{RRR} = \sum_{i=1}^{n}\delta(i)/n;$$
$$\delta(i) = \begin{cases}1, & R_1(i) - R_2(i) \neq 0\\ 0, & R_1(i) - R_2(i) = 0\end{cases} \tag{4}$$

其中 n 为行向量长度。sim_{RRR} 给出了两个行向量之间的相似程度,通过向量间不同元素的个数来度量。相似度越高,可压缩程度越高。

定义向量内冗余度(Row Inner Redundancy,RIR)如下:

$$sim_{RIR} = 1 - \sum_{i=2}^{n}\sigma(i)/n;$$
$$\sigma(i) = \begin{cases}1, & R_1(i) - R_1(i-1) \neq 0\\ 0, & R_1(i) - R_1(i-1) = 0\end{cases} \tag{5}$$

其中 n 为行向量长度。sim_{RIR} 给出了行向量内部的冗余程度,通过行向量内部相邻却不相同的元素所占比例来度量。冗余度越大,可压缩空间越大。

二阶压缩算法分两步执行:第一步利用向量相似度 sim_{RRR} 完成行向量间压缩;第二步利用向量内冗余度 sim_{RIR} 完成行向量内元素压缩。

2.2.1 行向量间压缩

1)计算各个行向量之间的相似度 sim_{RRR};

2)对所有满足条件 $sim_{RRR} > 0.25$ 的向量组进行合并存储。0.25 为实验测试得到的经验值。

Journal of Computer Applications 计算机应用

3)对于待合并向量 $\boldsymbol{R}_1$,$\boldsymbol{R}_2$,如图 2 所示,相同元素只保留一个,不同元素通过指针链接分别存储。

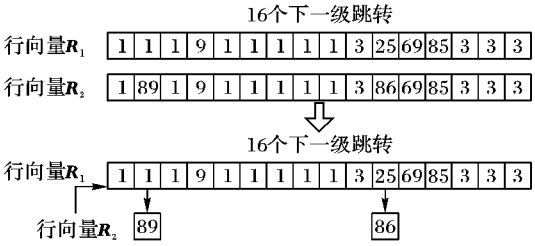


图 2　行向量间压缩示意图

2.2.2　行向量内冗余消除

1)计算各个行向量的冗余度 sim_{RIR};

2)选出满足条件 $sim_{\mathrm{RIR}} > 0.25$ 的向量消除行内冗余。

3)对于满足条件的向量 $\boldsymbol{R}_1$,建立 n 比特长的 1/0 数字串:NS。对于 $\boldsymbol{R}_1$ 中每一个元素,如果与前一个元素相同,则删除该元素,并将 NS 中对应位置为 0;如果与前一个元素不同,则保留该元素,将 NS 中对应位置为 1;$\boldsymbol{R}_1$ 中第一个元素需单独存储。得到新的行向量为 $\boldsymbol{R}_{\mathrm{new}}$。

向量内部冗余消除示意图如图 3 所示。经过行内冗余消除,每个行向量只需存储连续的不相同的元素以及一个 1/0 数字串,任意位置元素的读取方法为:

$$\begin{cases} R_1(i) = R_{\mathrm{new}}(k) \\ k = \sum_{j=1}^{i} NS(j) \end{cases} \tag{6}$$

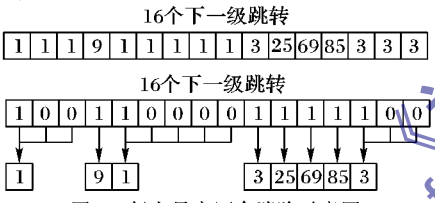


图 3　行向量内冗余消除示意图

3　性能仿真

算法采用 C 语言实现,系统为 Ubuntu 11.10,编译环境为 gcc4.2.4,内存 2 GB,CPU 为 Intel 双核,采用 Xilinx Vertex-6 LX565T 型号 FPGA,软件为 ISE13.3。

本文测试集合选自公开的 Snort 规则集[14]、Bro 规则集[15]等,包括:snort24、snort31、snort34、bro217、exact-math、ranges1、ranges5、tcp_homenet、dotstar0.3、dotstar0.6、dotstar0.9 等。在上述测试集合中,完成了 DFA 的数据统计,如表 2 所示;在测试集合中,选取部分规则集构建 MC-DFA(M = 1、2、4、8),即单字符 DFA、双字符 DFA(2C-DFA)、4 字符 DFA(4C-DFA)以及 8 字符 DFA(8C-DFA),对系统吞吐率、内存占有率、内存访问次数等性能进行测试验证,并与 SFA、FEACAN 以及 CSCA 进行对比。

3.1　单字符平均内存访问次数

单字符平均内存访问次数定义为各算法统计平均意义下处理每个字符需要访问内存的次数,用来衡量算法访问内存的频繁程度。

图 4 给出了部分规则集下 DFA、2C-DFA、4C-DFA 和 8C-DFA 的单字符平均内存访问次数,其中测试集 A ~ E 分别表示 snort34,bro217,ranges1,exact-math 和 dotstar0.6。可以看出,随着每周期处理字符数 M 的增大,单字符平均内存访问次数递减,这是由于每个周期能够处理更多的字符,而每次处理需要访问的内存次数是一定的,在平均意义下,每个字符所需的内存访问次数减少。

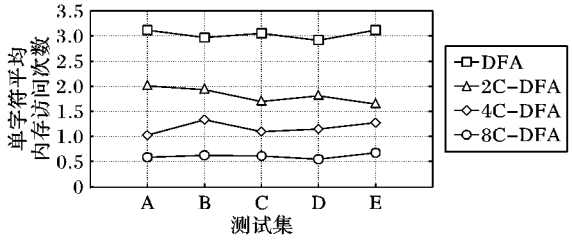


图 4　不同 M 值下平均内存访问次数

图 5 给出了部分规则集下 8C-DFA、SFA、FEACAN 和 CSCA 的单字符平均内存访问次数。SFA 算法没有存储压缩和多字符处理,所以单字符平均内存访问次数是 1;FEACAN 和 CSCA 使用了不同的压缩方法,降低了内存占用,但处理每个字符需要访问多次内存。四种算法中,8C-DFA 的单字符平均内存访问次数最少。

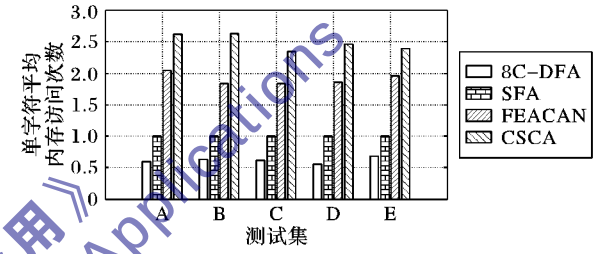


图 5　不同算法的平均内存访问次数

3.2　系统吞吐率

系统吞吐率是衡量算法匹配速率的关键指标,采用部署于 FPGA 后的吞吐率作为实验数据。

图 6 给出了部分规则集下 DFA、2C-DFA、4C-DFA 和 8C-DFA 的吞吐率对比情况。随着单周期处理字符数的增多,系统吞吐率相应增大;随着规则集的增大,生成的 DFA 结构更加复杂,导致处理速率降低。

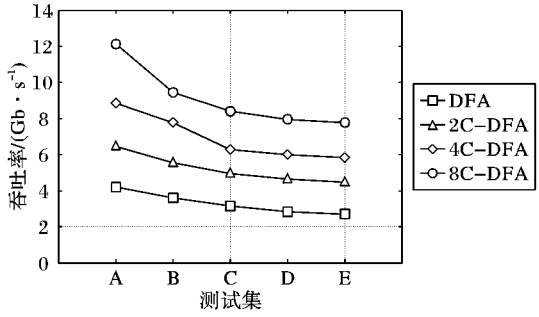


图 6　不同 M 值下的吞吐率对比

此外,吞吐率变化倍数不与单周期处理字符数成严格的比例,相对 DFA 而言,2C-DFA、4C-DFA 以及 8C-DFA 吞吐率增长分别约为 150%、250% 和 300%,这与理论值有一定的差距,原因在于每次读取多个字符时,需要消耗较多的内存读写时间,使得处理速率低于理论值。

图 7 给出了部分规则集下 8C-DFA、SFA、FEACAN 和 CSCA 的吞吐率对比情况。其中 SFA、FEACAN 和 CSCA 的吞吐率较为接近,8C-DFA 约为其他三种算法的两倍。

3.3　内存占用

内存占用是衡量算法空间性能的重要指标,采用部署于 FPGA 后的内存占用作为实验数据。

图 8 给出了部分规则集下 DFA、2C-DFA、4C-DFA 和 8C-

《计算机应用》Journal of Computer Applications

DFA 的内存占用对比情况。随着规则集的增大,MC-DFA 的内存占用逐渐增长;8C-DFA 的内存占用比 2C-DFA 增加约 4 倍。根据定理 3,不进行压缩的情况下,8C-DFA 的内存占用最大为 2C-DFA 的 196 倍,证明了 MC-DFA 采用压缩算法的有效性。

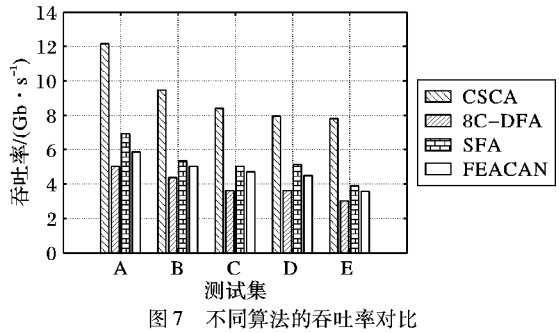


图 7 不同算法的吞吐率对比

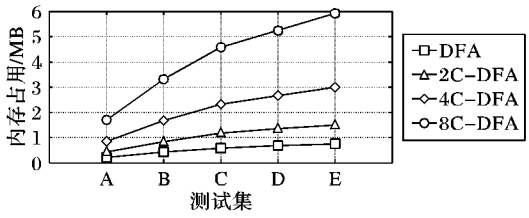


图 8 不同 M 值下内存占用

图 9 给出了部分规则集下 8C-DFA、SFA、FEACAN 和 CSCA 的内存占用对比情况。可以看出,8C-DFA 的内存占用为其他三种算法的 4 ~ 8 倍,在实验规则集下,内存占用最大为 6 MB,软硬件中都可以轻松满足内存需求,因此内存占用的增加在接受范围内。

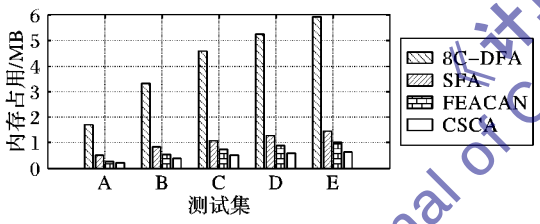


图 9 不同算法的内存占用

3.4 压缩率与预处理时间

表 3 给出了 MC-DFA 算法与 SFA、FEACAN 算法以及 CSCA 在压缩率、预处理时间的对比结果。压缩率是以传统 DFA 为标准,衡量各种算法对于内存缩减的程度;预处理时间是指算法将正则表达式规则集转化为 DFA 结构所需的时间。

表 3 其他性能对比

算法	压缩率/%	预处理时间/s
1C-DFA	78.60	3.39
2C-DFA	77.94	6.38
4C-DFA	81.23	11.43
8C-DFA	79.23	19.24
FEACAN	84.37	5.32
CSCA	88.34	6.27
SFA	82.54	1.28

从表 3 中可以看出,MC-DFA 算法的压缩率较为稳定,在 80% 左右,CSCA 的压缩率最好,可以达到 88%。预处理时间方面,随着 M 值的增大,MC-DFA 算法的预处理时间增长幅度较大,但数量级均为秒级,在可接受范围之内。各种算法中,SFA 的预处理时间最短,只需 1.28 s。

综合测试结果,8C-DFA 在吞吐率以及内存访问次数方面表现最好,但是在内存占用以及预处理时间方面比 FEACAN、CSCA 以及 SFA 差;压缩率方面,MC-DFA 接近 80%,与其他几种算法相差较小。CSCA 算法内存占用最好,压缩率最高,但是系统吞吐率以及单周期平均内存访问次数方面表现较差。FEACAN 算法的单周期平均内存访问次数最少,吞吐率要高于 2C-DFA,但小于 4C-DFA 和 8C-DFA。该对比数据说明实验表明,MC-DFA 算法以一定的内存占用和预处理时间为代价,显著地提高了系统吞吐率,并且内存占用以及预处理时间均在可接受范围之内,能够满足硬件及软件实现要求。

4 结语

本文提出的 MC-DFA 算法打破了传统的 DFA 算法单周期处理单字符的速度瓶颈,以一定的状态转移数量增长为代价,较好地提升了系统处理速率。MC-DFA 算法通过合并状态转移达到多字符处理目标,辅以相应的 SST 压缩算法,缓解了状态转移数量增长带来的内存损耗压力。通过 FPGA 部署实现,测试验证了算法的性能,给出了相应参数的选取标准,证明了算法的可行性和有效性。随着单周期处理字符数的增多,内存读写、内存损耗问题会带来较大影响,使得 MC-DFA 算法处理速率增长速度降低,不能成倍提升处理速率。

针对单周期多字符处理的研究并不仅限于文中提出的方法,其他的方法例如合并状态、并行处理等均据有一定的可行性。下一步的研究工作主要集中于算法在硬件中的优化以及算法的进一步性能提升。

参考文献:

[1] ANTONELLO R, FERNANDES S, SADOK D, *et al*. Deterministic finite automaton for scalable traffic identification: the power of compressing by range [C]// Proceedings of the 2012 IEEE Network Operations and Management Symposium. Piscataway: IEEE, 2012: 155 - 162.

[2] ZHENG K, ZHANG X, CAI Z, *et al*. Scalable NIDS via negative pattern matching and exclusive pattern matching [C]// INFOCOM 2010: Proceedings of 2010 IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies. Washington, DC: IEEE Computer and Communications Societies, 2010: 1 - 9.

[3] LIU T W, SUN Y, GUO L, *et al*. SDFA: series DFA for memory-efficient regular expression matching [C]// CIAA'12: Proceedings of the 17th International Conference on Implementation and Application of Automata, LNCS 7381. Berlin: Springer-Verlag, 2012: 337 - 344.

[4] WOODS L, TEUBNER J, ALONSO G. Complex event detection at wire speed with FPGAs [J]. Proceedings of the VLDB Endowment, 2010, 3(1/2): 660 - 669.

[5] YANG Y H E, JIANG W, PRASANNA V K. Compact architecture for high-throughput regular expression matching on FPGA [C]// Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems. New York: ACM, 2008: 30 - 39.

[6] MITRA A, NAJJAR W, BHUYAN L. Compiling PCRE to FPGA for accelerating snort IDS [C]// Proceedings of the 3rd ACM/IEEE Symposium on Architecture for Networking and Communications Systems. New York: ACM, 2007: 127 - 136.

(下转第 2382 页)

Journal of Computer Applications 计算机应用

表 1 shift 函数值

j	P	skip	shift(文献[7,15]算法)	shift(本文算法)
1	a	0	20	20
2	b	4	19	19
3	d	8	18	18
4	b	4	17	17
5	a	0	16	16
6	c	5	15	15
7	b	4	14	14
8	a	0	4	4
9	a	0	4	4
10	a	0	4	4
11	a	0	4	1

6 结语

本文首先分析了 BM 算法中函数 shift 的经典定义,给出了 shift 及其计算方法的严格、清晰的形式表述和数学证明;在此基础上,给出了 shift 的一个新的构造算法,证明了其时间复杂度与空间复杂度均为 $O(m)$。

值得注意的是,随着串匹配技术应用领域的不断扩大,基于软件的实现已经不能充分满足对匹配速度日益增长的需求,因此,基于硬件的实现逐渐出现,并成为当前的研究热点。理论分析与计算结果表明,本文给出的 shift 构造算法比已有的算法更简单,计算复杂度更低,从而更适合硬件实现。

BM 算法及其各种变形的平均时间复杂度仍然是一个尚未完全解决的问题。目前的研究结果主要集中在没有 shift 的 Horspool 算法,最近的结果有:在文本字符相互独立且同分布的假设下,Mahmoud 等[16]证明了 Horspool 算法的平均时间复杂度是渐进正态的,Smythe[17]应用 Markov 链理论也得到了这一结果。至于 BM 算法,shift 使得算法分析变得更加复杂,虽然 Boyer 与 Moore 证明了是亚线性的,但其概率模型有待商榷。因此,BM 类算法的平均时间复杂度有待进一步研究。

参考文献:

[1] BOYER R S, MOORE J S. A fast string searching algorithm [J]. Communications of the ACM, 1977, 20(10): 762 – 772.

[2] HORSPOOL R N. Practical fast searching in strings [J]. Software Practice and Experience, 1980, 10(6): 501 – 506.

[3] SUNDAY D M. A very fast substring search algorithm [J]. Communications of the ACM, 1990, 33(8): 132 – 142.

[4] COMMENTZ-WALTER B. A string matching algorithm fast on the average [C]// Proceedings of the 6th Colloquium, on Automata, Languages and Programming, LNCS 71. Berlin: Springer-Verlag, 1979: 118 – 132.

[5] WU S, MANBER U. A fast algorithm for multi-pattern searching, TR-94-17 [R]. Tucson: University of Arizona, 1994.

[6] KNUTH D E, MORRIS J H, PRATT V R. Fast pattern matching in strings [J]. SIAM Journal of Computing, 1977, 6(2): 323 – 350.

[7] RYTTER W. A correct preprocessing algorithm for Boyer-Moore string searching [J]. SIAM Journal of Computing, 1980, 9(3): 509 – 512.

[8] HUME A, SUNDAY D M. Fast string searching [J]. Software Practice and Experience, 1991, 21(11): 1221 – 1248.

[9] 张红梅, 范明钰. 模式匹配 BM 算法改进[J]. 计算机应用研究, 2009, 26(9): 3249 – 3252.

[10] 董明明, 巩青歌, 张琦. 入侵检测系统中模式匹配算法的改进[J]. 计算机应用与软件, 2011, 28(5): 272 – 274.

[11] 王浩, 张霖. 基于坏字符序检测的快速模式匹配算法[J]. 计算机应用与软件, 2012, 29(5): 114 – 116.

[12] GUIBAS L J, ODLYZKO A M. A new proof of the linearity of the Boyer-Moore string searching algorithm [J]. SIAM Journal of Computing, 1980, 9(4): 672 – 682.

[13] COLE R. Tight bounds on the complexity of the Boyer-Moore algorithm [J]. SIAM Journal of Computing, 1994, 23(5): 1075 – 1091.

[14] 刘萍, 刘燕兵, 郭莉, 等. 串匹配算法中模式串与文本之间关系的研究[J]. 软件学报, 2010, 21(7): 1503 – 1514.

[15] YANG W. On the shift-table in Boyer-Moore's string matching algorithm [J]. International Journal of Digital Content Technology and its Applications, 2009, 3(4): 10 – 20

[16] MAHMOUD H M, SMYTHE R T, REGNIER M. Analysis of Boyer-Moore-Horspool string-matching heuristic [J]. Random Structures Algorithms, 1997, 10(1/2): 169 – 186.

[17] SMYTHE R T. The Boyer-Moore-Horspool heuristic with Markovian input [J]. Random Structures Algorithms, 2001, 18(2): 153 – 163.

(上接第 2374 页)

[7] JIANG L, TAN J L, LIU Y B. ClusterFA: a memory-efficient DFA structure for network intrusion detection [C]// ASIACCS '12: Proceedings of the 7th ACM Symposium on Computer and Communications Security Information. New York: ACM, 2012: 65 – 66.

[8] LIU T W, YANG Y F, LIU Y B, *et al.* An efficient regular expressions compression algorithm from a new perspective [C]// INFOCOM 2011: Proceedings of 30th IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies. Washington, DC: IEEE Computer and Communications Societies, 2011: 2129 – 2137.

[9] YANG Y-H E, PRASANNA V K. Space-time tradeoff in regular expression matching with semi-deterministic finite automata [C]// INFOCOM 2011: Proceedings of 30th IEEE International Conference on Computer Communications. Washington, DC: IEEE Computer and Communications Societies, 2011: 1853 – 1861.

[10] QI Y X, WANG K, FONG J, *et al.* FEACAN: front-end acceleration for content-aware network processing [C]// INFOCOM 2011: Proceedings of 30th IEEE International Conference on Computer Communications. Washington, DC: IEEE Computer and Communications Societies, 2011: 2114 – 2122.

[11] CLARK C R, SCHIMMEL D E. Scalable pattern matching for high speed networks [C]// FCCM 2004: Proceedings of 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines. Piscataway: IEEE, 2004: 249 – 257.

[12] SUTTON P. Partial character decoding for improved regular expression matching in FPGAs [C]// FTP 2004: Proceedings of 2004 IEEE International Conference on Field-Programmable Technology. Piscataway: IEEE, 2004: 25 – 32.

[13] YAMAGAKI N, SIDHU R, KAMIYA S. High-speed regular expression matching engine using multi-character NFA [C]// FPL 2008: Proceedings of the 2008 International Conference on Field Programmable Logic and Applications. Piscataway: IEEE, 2008: 131 – 136.

[14] Sourcefire, SNORT [DB/OL]. (2012 – 10 – 09) [2012 – 12 – 18]. http://www.snort.org/snort-downloads.

[15] International Computer Science Institute, Bro Intrusion Detection System [DB/OL]. (2012 – 08 – 29) [2012 – 12 – 25]. http://bro-ids.org/download/index.html.

《计算机应用》Journal of Computer Applications