

## 基于状态约束的大规模正则表达式匹配算法

贺炜, 郭云飞, 扈红超

(国家数字交换系统工程技术研究中心, 河南 郑州 450002)

**摘要:** 通过观察不确定有限自动机 NFA 到确定性有限自动机 DFA 的转化过程, 分析内存增长的原因, 提出了一种基于状态间约束关系的正则表达式匹配算法 Group<sup>2</sup>-DFA。Group<sup>2</sup>-DFA 通过两级分组, 利用状态间的约束关系, 将原始 NFA 转化为 NFA 和 DFA 的混合结构。实验表明, 在保持一定处理速率的前提下, Group<sup>2</sup>-DFA 能够有效地减少内存占用。在 300 条规则下, Group<sup>2</sup>-DFA 吞吐率能够达到 1Gbps, 并且减少约 75% 的状态数。

**关键词:** 正则表达式; 状态爆炸; 自动机转化; 状态约束

中图分类号: TP 393

文献标识码: A

文章编号: 1000-436X(2013)10-0183-08

## States constrain-based algorithm for large scale regular expression matching

HE Wei, GUO Yun-fei, HU Hong-chao

(National Digital Switching System Engineering Technological R&D Center, Zhengzhou 450002, China)

**Abstract:** By analysis of state explosion in deterministic finite automata DFA, a novel algorithm Group<sup>2</sup>-DFA based on state constrains was proposed to reduce the memory usage. With the state constrains, states in NFA were classified into several groups. Group<sup>2</sup>-DFA introduces two-level classification and merges NFA and DFA together to a hybrid FA construction. The experiments show that Group<sup>2</sup>-DFA can reduce memory usage efficiently and keep high throughput with a small increase of memory reading time. With 300 regex rules, Group<sup>2</sup>-DFA can cut 75% states and achieve 1Gbps throughput.

**Key words:** regular expression; state explosion; automata convert; state constrains

### 1 引言

近年来, 作为网络信息过滤、文字处理等应用中的关键组成部分, 正则表达式匹配技术得到了快速发展。正则表达式匹配技术作为实现大规模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中<sup>[1,2]</sup>。目前正则表达式匹配主要通过不确定性有限自动机 (NFA, nondeterministic finite automata) 以及确定性有限自动机 (DFA, deterministic finite automata) 来实现。

假设需要对  $k$  ( $k > 1$ ) 条正则表达式完成匹配, 其中, 第  $i$  ( $i = 1, 2, \dots, k$ ) 条正则表达式中包含  $n_i$  个字符。将  $k$  条正则表达式通过 McNaughton-Yamada

构造法转化为 NFA 后, 会产生  $n = \sum_{i=1}^k n_i$  个 NFA 状态<sup>[3]</sup>, 每输入一个字符, 都需要对这些状态进行并行检测。上述特性导致基于 NFA 的正则表达式匹配在处理过程中的时间复杂度较高。虽然 NFA 能够通过硬件结构的加速来提高处理速率, 但硬件加速方案通常缺少灵活性并且很难做到实时更新<sup>[4]</sup>。

针对 NFA 的缺点, 另外一种解决方案是将 NFA 转化为 DFA。通过将 NFA 中各个状态之间所有的有效转移存储在状态转移表 (STT, state transition table) 中, DFA 可在硬件或软件平台中得到较好的实现。但是, 基于 DFA 的正则表达式匹配也存在一些问题<sup>[5]</sup>。

收稿日期: 2012-07-15; 修回日期: 2013-04-15

基金项目: 国家科技支撑计划基金资助项目 (2012BAH02B03, 2012BAH02B01); 国家高技术研究发展计划 (“863” 计划) 基金资助项目 (2011AA01A103, 2011AA01A101, 2011BAH19B04)

**Foundation Items:** The National Science & Technology Pillar Program (2012BAH02B03, 2012BAH02B01); The National High Technology Research and Development Program of China (2011AA01A103, 2011AA01A101, 2011BAH19B04)

1) 状态数爆炸:在最坏情况下,对于有  $n$  个状态的非确定性有限自动机 (NFA),其最小等价确定性有限自动机 (DFA) 的状态数将达到  $O(2^n)$  数量级。

2) 压缩时间代价:尽管 STT 可以通过消除相似状态冗余来压缩,但是现有压缩算法的时间复杂度至少为  $O(n^2)$ 。

NFA 的空间特性较好,时间特性较差,而 DFA 相反,两者都不能很好地满足实际网络应用的需求。因此,快速高效是大规模正则表达式匹配的首要目标。本文将 NFA、DFA 两者优势相结合,提出了一种新的自动机构建算法 Group<sup>2</sup>-DFA,在保持高速处理的前提下,缓解 DFA 状态爆炸的影响。本文首先通过分析状态爆炸现象,得出 DFA 状态爆炸的原因,然后根据状态间约束关系对 NFA 状态集合进行分组;第三步将每个 NFA 组转化为 DFA,得到若干 single-DFA,将各个 single-DFA 按照一定的方法组合,得到 Group-DFA。由于 Group-DFA 处理速率较低,对 Group-DFA 进行二级分组,最终得到 Group<sup>2</sup>-DFA。

## 2 相关工作

基于 NFA 的正则表达式匹配最初在集成电路中实现,随后 SIDHU 和 PRASANNA 将其部署在现场可编程门阵列 (FPGA) 中<sup>[6]</sup>。为了进一步提高系统的吞吐率以及资源利用率,FPGA 实现中加入了许多优化措施,例如流水线、前缀提取、多字符匹配、字符编码等<sup>[7,8]</sup>。但是由于电路结构本身的缺点,在 FPGA 平台中,很难对基于 NFA 的正则表达式匹配进行及时更新。

基于 DFA 的正则表达式匹配可通过处理器架构来实现,相对于 NFA,这种结构更加简单并易于操作。然而,状态爆炸问题导致算法对于内存的需求急剧增加,影响了内存性能和实际中部署的可行性。

为了解决 DFA 内存需求高的问题,目前常用的方法有 2 种:1) YU、PASETTO 等<sup>[9,10]</sup>利用 NFA 的不确定性来压缩 STT,以此达到较高的吞吐率性能。但是这类方法计算的复杂度较高,且大多依赖于特定的正则表达式集合特性。QI 等<sup>[11]</sup>利用 DFA 状态的相似性压缩 STT,提出 FEACAN (front-end acceleration for content-aware network) 算法,能够达到 80% 的压缩率。LIU 等<sup>[12]</sup>提出的 CSCA (cluster-based splitting compression algorithm) 通过

切分 STT 发现内在的冗余,能够达到 95% 的压缩率。但是 FEACAN 和 CSCA 算法都会降低系统的吞吐率。2) 将大规模正则表达式匹配问题分解为若干小问题分别解决。ANTONELLO 等<sup>[13]</sup>使用判决检验算法将  $m(m > 1)$  个正则表达式组合成  $g(m > g > 0)$  个小组;WOODS 等<sup>[14]</sup>将正则表达式切分为前缀和后缀 2 个组合,并且将所有的前缀构造为一个复合 DFA,当到达 DFA 边界状态时,触发后缀匹配;YANG 等<sup>[15]</sup>提出的 SFA (semi-deterministic finite automata) 算法采用 NFA 和 DFA 混合结构,在空间利用和吞吐率方面均有较好的效果。

## 3 理论基础

### 3.1 正则表达式匹配

定义 1 对于正则表达式  $R$ ,基于 NFA 的正则表达式匹配是在连续输入字符流的条件下,构造不确定性有限自动机  $M = (Q, S, d, q_0, F)$ ,其中,  $Q$  是有限的 NFA 状态集合;  $S$  是有限的字母表;  $d$  是不确定性状态转移函数,数量级为  $Q \times S$ , 大约为  $2^Q$ ;  $q_0$  是状态机的初始状态,  $q_0 \in Q$ ;  $F$  是状态机结束状态集合,  $F \subseteq Q$ 。

定义 2 对于正则表达式  $R$ ,基于 DFA 的正则表达式匹配是根据标准构造方法,将相应的不确定性有限自动机  $M$  转化为确定性有限自动机  $M' = (Q', S, d', q_0, F')$ ,其中,  $Q'$  是有限的 NFA 状态集合,  $Q' \subseteq 2^Q$ ;  $d'$  是确定性状态转移函数,数量级为  $Q \times S$ ;  $F'$  是状态机结束状态集合,  $F' \subseteq Q'$ ;  $q_0$  和  $S$  的定义与 NFA 中相同。

### 3.2 状态爆炸分析

YU 等<sup>[9]</sup>证明了对于实际的正则表达式,NFA 到 DFA 的转化会引起指数级别的状态爆炸,其等价最小 DFA 有  $2^n$  个状态。为了更好地说明 Group<sup>2</sup>-DFA 原理,简要分析发生状态爆炸的原因。

对于正则表达式  $r_1 = /. * [a | b] [c | d] \{n - 1\} /.$ , 构建 NFA,命名为 A。A 中有初始状态  $q_0$ 、结束状态  $q_n$  及  $n - 1$  个中间状态  $\{q_1, L, q_{n-1}\}$ ,各个状态间相互独立,互不影响。接收任意的输入,  $q_0$  到达状态  $q_1$ ;接收输入  $a$  或  $b$ ,  $q_1$  到达状态  $q_2$ ;接收输入  $c$  或  $d$ ,  $q_i$  到达状态  $q_{i+1}$ ,  $1 < i < n$ 。

构造输入字符流,第一个字符为任意字符,第二个字符为  $a$  或  $b$ ,之后的字符为  $c$  或  $d$ ,各个字符之间相互独立不相关。当正则表达式  $r_1$  接收到这一字符流时,对应的  $2^n$  个状态均可能出现。将 A

转化为 DFA 后,至少会有  $2^n$  个状态,与原有  $n+1$  个状态相比,呈指数级增长。

以上论述的基础是各个输入字符间相互独立,各个状态间互不影响,而在实际应用中,各个状态间存在相互约束,利用这种约束关系,可减小状态爆炸的规模。例如:状态  $q_1$ 、 $q_2$  相互独立时,状态间约束不确定,存在 4 种状态组合  $(q_1, q_2) = \{(0,0), (0,1), (1,0), (1,1)\}$  (标记 1 表示状态被激活)。若假设状态  $q_1$  被激活时,  $q_2$  一定被激活,则状态间约束关系确定,状态组合变为  $(q_1, q_2) = \{(0,0), (1,1)\}$  2 种,转化为 DFA 后,总状态数由  $2^{n+1}$  减少为  $2^n$ 。以此类推,当各个状态间均有确定的约束关系时,可消除 DFA 状态爆炸的现象。

### 3.3 状态约束关系

根据状态爆炸分析可得出,若能确定状态间的约束关系,减少转化过程中的不确定性,则可大幅地减少 DFA 中的状态数。为描述状态间约束关系,引入 3 种集合关系:相交、包含、分离,具体定义如下。

定义 3 记  $(q_0, s) = q$ , 其中,  $s$  为输入序列字符串;  $q_0$  为初始状态,  $q$  为  $q_0$  接收输入  $s$  后到达的目的状态。状态  $q_x, q_y \in Q$ , 记  $F_x$  是满足  $(q_0, s) = q_x$  的输入序列  $s$  的集合,  $F_y$  是满足  $(q_0, s) = q_y$  的输入序列  $s$  的集合。

若  $F_x \cap F_y = f$ , 则  $q_x, q_y$  互相分离, 记为  $q_x \perp q_y = f$ 。

若  $F_x \cap F_y = F_x$ , 则  $q_x$  包含于  $q_y$ , 记为  $q_x \subseteq q_y$ 。

若  $F_x \cap F_y \neq f$  且  $F_x \not\subseteq F_y, F_y \not\subseteq F_x$ , 则  $q_x, q_y$  相交。

当  $q_x, q_y$  分离时,约束关系确定,  $q_x, q_y$  不会被同时激活,在转化为 DFA 的过程中,不会引起状态爆炸。

当  $q_x$  包含于  $q_y$  时,  $q_x$  被激活,  $q_y$  一定被激活,不会引起状态增加;当  $q_y$  被激活,  $q_x$  不一定被激活,会引起一定的状态数增长。

当  $q_x, q_y$  相交时,2 种状态相关性不强,可能同时被激活,也可能只激活一个,约束关系不确定,会引起状态爆炸。

根据定义的 3 种约束关系,提出 NFA 状态约束关系分析 (NFA-SCA, NFA state constrain analysis) 算法,可得到 NFA 状态集中任意 2 个状态之间的约束关系。图 1 中 1)~8) 通过分析状态转移路径,得

到各个状态对应的输入序列集合  $F$ ; 9)~15) 比较各个状态的输入序列集合,决定相互间约束关系的种类。NFA-SCA 算法伪代码描述如图 1 所示。

```

输入: NFA M = (Q, S, d, q_0, F), q_x, q_y \in Q / q_0
输出: 状态间约束关系
1) for q_i \in Q, q_i \neq q_0 //遍历NFA状态集中状态
2)   for s_j \in \hat{a}^* //遍历相应的输入字符串
3)     //记录由q_0到达状态q_i的j种字符串s_j
4)     remember (s_j, q_i)
5)     //合并对应的j条字符串
6)     F_i = F_i \cup s_j
7)   end for
8) end for
9) for q_x, q_y \in Q; q_x \neq q_y
10) //对任意2个状态,得到状态间的约束关系
11)   if F_x \cap F_y = \emptyset then (q_x, q_y) = 0
12)   else if F_x \cap F_y = F_x then (q_x, q_y) = 1
13)   else if F_x \cap F_y = F_y then (q_x, q_y) = 2
14)   else (q_x, q_y) = 3
15) end for

```

图 1 NFA-SCA 算法

## 4 基于状态约束的 Group<sup>2</sup>-DFA 算法

### 4.1 NFA 状态集合分组

为了降低状态数爆炸式增长对正则表达式匹配的影响,在 NFA-SCA 算法得到的状态间约束关系的基础上,按照以下原则将状态集合划分为若干小组:

- 1) 同一组内状态两两不相交;
- 2) 相交的状态放到不同的组内;
- 3) 每个组内包含尽可能多的相离和相互包含的状态。

依照上述原则分组可保证同一组内不会有相交状态,确定了状态间的约束关系,避免了大规模的状态增长。

划分过程步骤如下。

1) 初始化:建立状态集合  $V$ , 包括所有 NFA 状态,置  $m = 0$ 。

2) 若  $V$  不为空,建立空集合  $V_m$ 。按状态序号,取  $V$  中一个状态  $q_i$ , 作以下判断:

若  $V_m$  为空,  $q_i$  加入  $V_m$ , 并从  $V$  中删除  $q_i$ ;

若  $V_m$  不为空,将  $q_i$  与  $V_m$  中所有状态相比,若  $q_i$  与这些状态都不相交,则  $q_i$  加入  $V_m$ , 并从  $V$  中删除  $q_i$ ; 否则,  $q_i$  放回  $V$  中,尝试  $V$  中下一个状态。

直到遍历  $V$  中的所有状态。记录  $V_m$  及其大小， $m = m + 1$ 。重复 2) 直到  $V$  为空。

状态集合划分 (SSD, state set division) 算法的伪代码描述如图 2 所示。

```

输入：NFA 状态集合
输出：k 个分组集合
1) while  $V \neq \emptyset$ 
2)   create  $V_m = \emptyset$ 
3)   for all  $q_i \in V$ 
4)     if  $V_m = \emptyset$  or  $(V_m, q_i) = 0/1/2$ 
5)       then  $V_m = V_m \cup q_i; V = V - q_i$ 
6)     end for
7)      $m = m + 1;$ 
8)   end while

```

图 2 SSD 算法

### 4.2 构建一级分组 DFA(Group-DFA)

SSD 算法可得到  $k$  个状态集合  $V_m, m = 0, 1, L, k$ ，每个  $V_m$  均有自身对应的状态集、字符表、转移函数、初始状态、结束状态，即对应于一个 NFA， $M_m = (Q_m, S, d_m, q_{0m}, F_m)$ ，其中， $Q_m$  是  $V_m$  中的状态集合； $S$  是有限的字母表； $d_m$  是  $V_m$  中的状态转移函数； $q_{0m}$  是 NFA 的初始状态； $F_m$  是 NFA 结束状态集合；

对于每一个  $M_m$ ，按照标准构建算法转化为相应的等价 DFA，即为  $M'_m$ ，由于每个  $M_m$  中不包含相交的状态，所有每个  $M'_m$  中状态数增长较少。这样可得到  $k$  个独立的 DFA，称为  $k$  个 single-DFA。

每个 single-DFA 中包括 2 种状态转移：内部转移和平行转移。内部转移实现 DFA 内部的状态跳转，每个平行转移作为其他 single-DFA 的输入，激活另一个 single-DFA 中的状态；因此，single-DFA 中的若干个状态都有可能同时被内部转移和其他 single-DFA 的平行转移同时激活，这种情况不满足 DFA 同一时刻只有一个激活状态的特性，为保证 DFA 的特性，添加多跳转关系综合判断逻辑，保留一个激活状态。

多跳转关系综合判断 (MTA, multi-transition analysis)：按照 single-DFA 编号的大小，即  $m$  值的大小定义优先级： $m$  值越小，single-DFA 输出的平行转移的优先级越高。当一个 single-DFA 中的几个状态同时被激活时，选择优先级高的转移所激活的状态，其余状态不被激活。

构造 Group-DFA 时，每个 single-DFA( $i$ ) 最多有  $k$  个进位输出状态，以  $k$  bit 长进位输出向量  $vector$  记录该输出状态。 $k$  个 single-DFA 最多同时有  $k$  个

进位输出向量： $vector(1), L, vector(k)$ ，记录此  $k$  个向量值，共需要  $k \times k$  大小的矩阵存储。

对于  $k$  个向量值，分析每个 single-DFA 会接收哪些平行转移，通过判断这些进位输出的优先级，即源 DFA 序号，确定保留哪一个平行转移作为输入。

MAT 算法代码描述如图 3 所示。

```

输入：k 个 single-DFA 输出的所有平行转移
输出：k 个 single-DFA 所需输入的平行转移
1) for  $i = 1:k$ 
2)   for  $j = 1:k$  //  $vector(1): vector(k)$ 
3)     //选择  $vector$  值最小的作为跳转输入
4)     if  $vector(j, i) = 1$ 
5)        $output(j, i);$  continue;
6)     end for
7)   end for

```

图 3 MTA 算法

$k$  个 single-DFA 间采用多跳转关系综合判断逻辑连接，都满足 DFA 的特性，组成的新 DFA 结构命名为 Group-DFA。Group-DFA 是 NFA 与 DFA 的混合结构，其中，各个 DFA 单独运行，DFA 之间为并行操作，由同一个初始状态触发，相当于 NFA 中的各个状态，呈现 NFA 特性。

在最好的情况下，原始 NFA 中，各个状态都不相交，则构造的 Group-DFA 是一个单独 DFA；在最坏的情况下，各个状态相互之间都相交，则构造的 Group-DFA 仍然是原始的 NFA。

### 4.3 构建二级分组 DFA(Group<sup>2</sup>-DFA)

Group-DFA 算法在实现过程中没有限制分组的数目。对于有  $N$  个状态的 NFA，假设存在  $k$  个相交状态对，则会产生  $k$  个 single-DFA。由于  $k$  个 single-DFA 之间呈 NFA 特性， $k$  值过大将导致处理速率急剧降低。

为降低分组数量对处理速率的影响，设计二级分组结构如下。

- 1) 将  $k$  个 single-DFA 看作一个 NFA 中的  $k$  个状态。
- 2) 对该  $k$  个状态执行 NFA-SCA 算法，得到状态间约束关系。
- 3) 根据 2) 中的结果，执行 SSD 算法，对状态集合进行划分，得到二级分组集合。不同之处在于划分集合的数目限定为  $p$  个，若存在多于  $p$  个的相交状态，将多余的状态划入已知的  $p$  个集合中，得到二级分组集合。
- 4) 对 3) 中得到的  $p$  个 NFA 状态集合按照标

准构建算法转化为  $p$  个 DFA，称为 Class2-DFA；并执行 MTA 算法，将  $p$  个 Class2-DFA 通过多跳转关系综合判断逻辑连接。

5) 在 4) 的结果中加入初始状态，二级分组 DFA——Group<sup>2</sup>-DFA 构建结束，Group<sup>2</sup>-DFA 中包含  $p$  个 Class2-DFA 结构，Class2-DFA 中的每个状态对应一个 single-DFA。

Group<sup>2</sup>-DFA 中包含 3 种转移关系：Class2-DFA 的内部转移、Class2-DFA 之间的平行转移和 single-DFA 的内部转移。Group<sup>2</sup>-DFA 中同样存在多个 DFA 状态被同时激活的问题，按照 single-DFA 中的方式，根据编号大小来定义优先级，然后使用多跳转关系综合判断逻辑处理。

#### 4.4 性能分析

有  $n$  个状态的 NFA，时间复杂度为  $O(l)$ ，空间复杂度为  $O(n)$ ；构建 Group-DFA 后，有  $k$  个 single-DFA，时间复杂度为  $O(k^2)$ ，空间复杂度为  $O(an)$ ，其中， $a$  为线性常数；构建 Group<sup>2</sup>-DFA 后，等价 NFA 结构有  $p$  个状态，且有两级的 DFA 逻辑结构，所以时间复杂度为  $O(2p^2)$ ，在  $p$  个二级集合中，每个集合最坏情况下有  $\lceil k/p \rceil$  个相交状态，每一个状态代表一个 single-DFA，定义  $x = \lceil k/p \rceil$  为二级分割参数；将  $p$  个二级集合转化为  $p$  个 DFA 后，相比于 Group-DFA，状态数会增加  $(2^x/x)$  倍，空间复杂度为  $O((2^x/x)an)$ ，随着  $\lceil k/p \rceil$  值的增加，Group<sup>2</sup>-DFA 的空间复杂度增大，但时间复杂度减小，为了权衡时间和空间的代价， $p$  值的选取需要根据实验结果和实际需求来确定。

### 5 实验仿真和性能验证

#### 5.1 模型建立

针对 Group-DFA 中 NFA、DFA 混合的特点，设计处理模型如图 4 所示。

图 4 中，Group-DFA 由若干个 single-DFA 单元以及多跳转关系综合判断模块（简称为综合模块）组成，single-DFA 单元如虚线框内所示，包括状态查找表 STT、状态合并、输入组合 3 个模块：输入组合以输入字符和当前状态作为输入，并将组合结果送入 STT 查询；STT 根据输入数据得到相应的转移结果，若是内部转移，直接送入状态合并模块，若是进位转移，则送入综合模块；状态合并模块接收 STT 的内部转移以及综合模块得到的进位转移作为输入，决定在 single-DFA 内部选择哪一个转移状态作为当前状态。

综合模块实现 MTA 算法中的功能，分析处理各个 single-DFA 得到的进位转移结果。

对于 Group<sup>2</sup>-DFA 二级分组结构包含  $p$  个 DFA 结构，每个 DFA 中的状态对应一个 single-DFA，处理模型如图 5 所示。

#### 5.2 性能仿真

性能分析采用的实验环境为：系统：Ubuntu 11.10；编译环境：gcc4.2.4；内存：2GB；CPU：Intel 双核。

实验数据采用 snort24、snort31、bro217、dotstar300 等 4 个规则集<sup>[16,17]</sup>。本节从空间存储消耗、系统吞吐率以及单字符平均内存访问次数 3 方面对算法性能进行分析，对比算法包括 Group<sup>2</sup>-DFA 算法、SFA 算法、FEACAN 算法以及 CSCA 算法。为了评价 4 种算法的性能，消除额外因素的影响，算法实现时，去掉了 FEACAN 算法中的硬件加速部分。

##### 5.2.1 正则表达式规模与 single-DFA 数量关系

表 1 中给出了正则表达式规模与生成的 single-DFA 数量之间的关系，规模大小以正则表达式数量表示；采用 NFA-SCA 算法和 SSD 算法生成相应的若干 single-DFA。

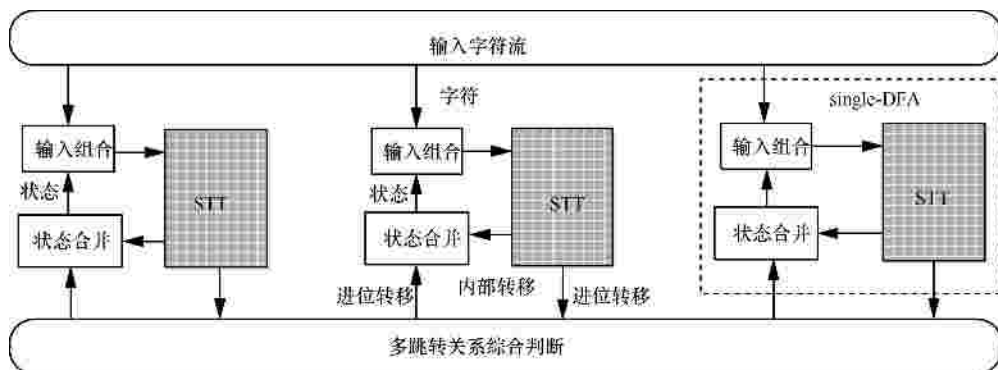


图 4 Group-DFA 处理模型

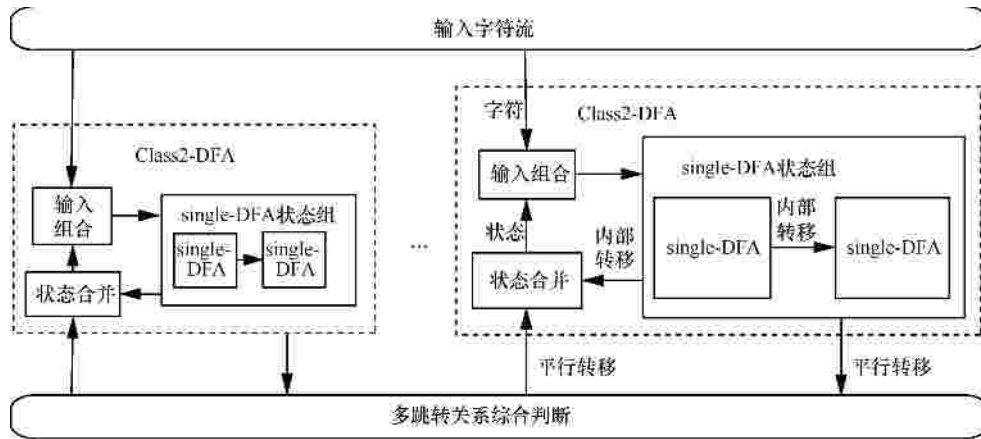


图 5 Group<sup>2</sup>-DFA 处理模型

表 1 正则表达式规模与 single-DFA 数量关系

正则表达式数目	NFA 状态数	single-DFA 数量
16	418	15
20	524	18
24	571	21
34	882	33
217	2 130	59
300	11 145	133

从表 1 中可看出 随着正则表达式规模的增大，NFA 状态数也增多，各个状态之间的关系也更加复杂，出现了更多的相交情形，single-DFA 的数量也随之增多。

### 5.2.2 二级分割参数 $\lceil k/p \rceil$ 的选取

为衡量二级分割参数  $x = \lceil k/p \rceil$  对 Group<sup>2</sup>-DFA 状态数和吞吐率性能的影响，定义性能增益函数  $g(x)$  为

$$g(x) = \frac{Thp(x)/Thp(1)}{SN(x)/SN(1)}$$

其中， $Thp(x)$  为吞吐率函数， $SN(x)$  为状态数函数，分别表示二级分割参数为  $x$  时的系统吞吐率和状态数。其中， $x = 1$  时，表示不进行分割，Group<sup>2</sup>-DFA 就是 Group-DFA。由性能增益函数  $g(x)$  的定义可知， $g(x)$  是以  $x = 1$  时的吞吐率  $Thp(1)$  和状态数  $SN(1)$  为标准，将  $Thp(x)$  和  $SN(x)$  归一化后的比值，表示随着  $x$  的变化，吞吐率和状态数增长的效果。

在 snort24、bro217 和 dotstar300 3 种规则集下，分别计算 Group<sup>2</sup>-DFA 的状态数以及吞吐率如图 6 所示。

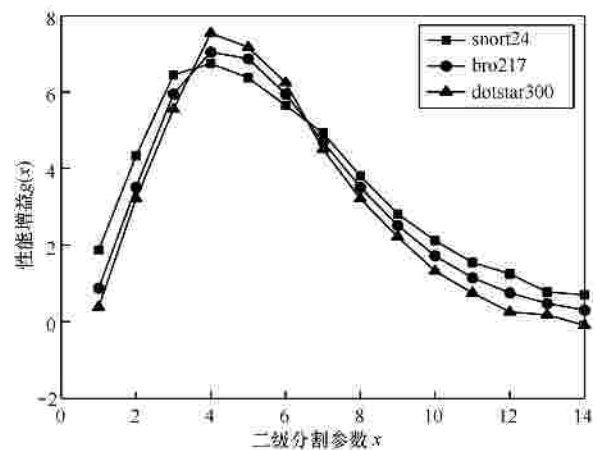


图 6 性能增益随分割参数的变化

从图 6 中可看出，对于不同的规则集， $x = 4$  时，性能增益均达到最大值，说明此时吞吐率增长较大，状态数增长相对较小，整体性能达到最佳。

面对不同的实际应用，对于空间性能要求较高时，可减小分割参数；对于吞吐率性能要求较高时，可增大分割参数，以达到合适的效果。

### 5.2.3 空间存储性能与系统吞吐率

空间存储占用与状态数量呈正比，因此分析算法生成的状态数可反映空间存储的性能。实验中实现了基于 SFA 算法、FEACAN 算法、CSCA 算法以及 Group<sup>2</sup>-DFA 算法的正则表达式匹配，根据 5.2.2 节的实验结果，Group<sup>2</sup>-DFA 算法的二级分割参数取  $x = 4$ 。正则表达式规则选用 snort24、bro217 和 dotstar300 的混合集合，规则数量从 24 开始逐渐增加，最多选取 300 条正则表达式。

4 种算法的状态数对比情况如图 7 所示。正则表达式规模较小时，4 种算法得到的状态数差别较小；随着正则表达式规模的增加，Group<sup>2</sup>-DFA 算法

的状态数增长最慢,近似呈线性增长,而 SFA 算法、FEACAN 算法以及 CSCA 算法的状态数增长较快,可以看出,Group<sup>2</sup>-DFA 算法在减少状态数方面的优势较为明显。

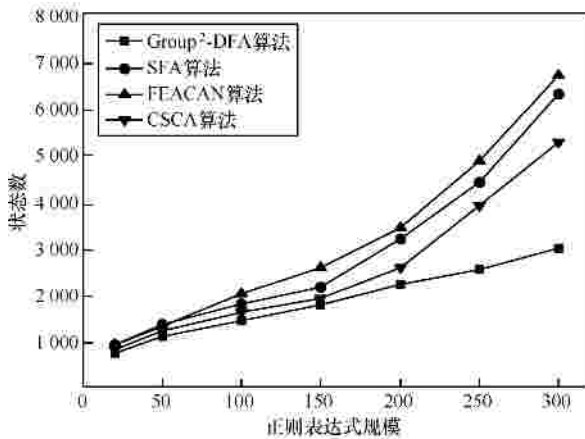


图 7 4 种算法状态数对比

表 2 给出了 4 种规则集合下,Group<sup>2</sup>-DFA 相对于 DFA 的状态减少率。

表 2 正则表达式集合下 Group<sup>2</sup>-DFA 的状态减少率

算法	snort24	snort34	bro217	dotstar300
NFA	571	882	2 130	11 145
DFA	8 334	9 753	6 532	>100 000
Group <sup>2</sup> -DFA	1 743	2 356	2 974	3 341

根据表 2 计算可得,对于 4 个不同的规则集合,Group<sup>2</sup>-DFA 的状态减少率分别为:79.1%、75.9%、54.4%以及大于 96.6%。

正则表达式集合 bro217 中,处于相交关系的状态较少,状态爆炸不明显,Group<sup>2</sup>-DFA 算法对于状态数的减少百分比较小;测试中,Group<sup>2</sup>-DFA 算法对于 snort 集合、dotstar\_300 集合的性能较好,能达到 75%以上。另外,由于 dotstar\_300 生成的 DFA 数目大于 1×10<sup>5</sup> 个,超出预设的内存占用,没有得到全部的 DFA 状态数。

图 8 给出了 4 种算法的吞吐率 (Gbit/s) 对比情况。从图 8 中可以看出,随着正则表达式规模的增加,Group<sup>2</sup>-DFA 算法的吞吐率较为稳定,这是由于 Group<sup>2</sup>-DFA 算法的吞吐率主要受到并行等价 NFA 数量的影响,与参数 x 的选取有关,与正则表达式规模无关。Group<sup>2</sup>-DFA 算法的吞吐率与 SFA 算法较为接近,相差较小,但 Group<sup>2</sup>-DFA 算法的浮动较小,比 SFA 算法稳定。

而 FEACAN 和 CSCA 的吞吐率受正则表达式规模的影响较大,虽然在正则表达式规模较小时,两者的吞吐率高于 Group<sup>2</sup>-DFA 算法,但是随着正则表达式规模的增加,吞吐率呈下降趋势,当正则表达式大于 250 条时,其吞吐率小于 Group<sup>2</sup>-DFA 算法。

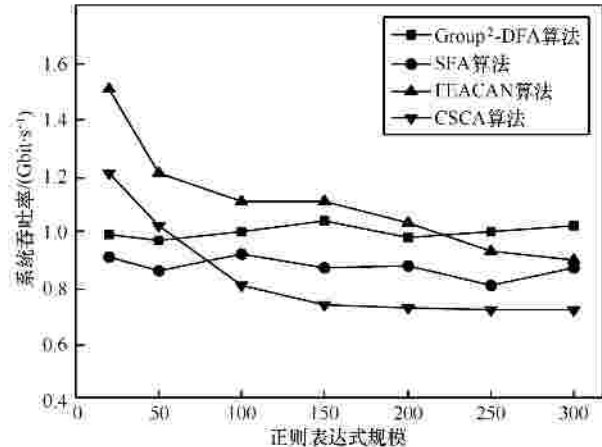


图 8 4 种算法的吞吐率对比

图 9 给出了 4 种算法处理一个字符时的平均内存访问次数。可以看出,在处理一个字符时,Group<sup>2</sup>-DFA 算法需要访问多次内存,时间复杂度有所增长,高于其他 3 种算法。这是由于 Group<sup>2</sup>-DFA 算法中包括两级分组结构,每一级中都包括 NFA 和 DFA 2 种形态,而 NFA 和 DFA 的状态转移表需要单独存储,导致处理字符时,可能需要读取多次内存。虽然 Group<sup>2</sup>-DFA 算法的时间复杂度有所增长,但由于采用二级分组结构,充分利用了 DFA 的快速处理特性,所以 Group<sup>2</sup>-DFA 算法仍然可以达到较高的吞吐率。

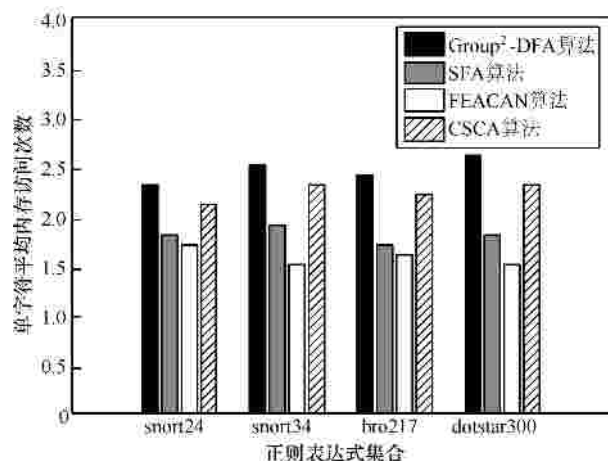


图 9 单字符平均内存访问次数

实验结果表明, Group<sup>2</sup>-DFA 算法的状态数小于 SFA、FEACAN、CSCA 算法; 当正则表达式规模增加时, Group<sup>2</sup>-DFA 算法的吞吐率稳定在较高水平; 对于不同的规则集合, Group<sup>2</sup>-DFA 算法的单字符平均内存访问次数高于其他 3 种算法。综上所述, 根据性能增益合理选取分割参数后, Group<sup>2</sup>-DFA 算法能够达到较好的状态减少率, 并且保持较高的系统吞吐率, 付出的代价是单字符平均内存访问次数有一定的增加。

## 6 结束语

本文提出的 Group<sup>2</sup>-DFA 算法结合 NFA 的空间和 DFA 的时间特性, 以一定的时间复杂度为代价, 减小了 DFA 的空间复杂度。算法通过分析 NFA 转化为 DFA 过程中产生状态爆炸的原因, 引入了一种状态分类方法, 构建二级分组模型, 在保证系统吞吐率的前提下, 有效地减轻了状态爆炸的影响。在系统结构上, Group<sup>2</sup>-DFA 算法呈现 NFA 的并行处理特性, 在内部实现中, 采用单独的 DFA 处理; 这种混合结构使得 Group<sup>2</sup>-DFA 算法能够通过多核技术实现并行加速, 有效地提升了系统的吞吐率。

对于大规模的正则表达式集合, 分组得到的 single-DFA 数量较多, 会降低系统处理的性能, 进一步的研究可考虑对分组的方法进行改进, 以提高处理速率。另外, 由于采用单字符输入的方法, 处理速率受到一定的限制, 为了达到更高的系统吞吐率, 可引入多字符处理机制来满足实际需求。

## 参考文献:

- [1] JIANG L, TAN J, LIU Y. ClusterFA: a memory-efficient DFA structure for network intrusion detection[A]. Proceedings of the 7th ACM Symposium on Computer and Communications Security Information[C]. New York, USA, 2012. 65-66.
- [2] BECCHI M, CROWLEY P. Efficient regular expression evaluation: theory to practice[A]. Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems[C]. Colorado: ACM, 2008. 50-59.
- [3] YANG Y H E, JIANG W, PRASANNA V K. Compact architecture for high-throughput regular expression matching on FPGA[A]. Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems[C]. San Jose, California, USA, 2008. 30-39.
- [4] ZHENG K, ZHANG X, CAI Z, *et al.* Scalable NIDS via negative pattern matching and exclusive pattern matching[A]. Proceedings IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies[C]. San Diego, 2010. 1-9.
- [5] 李奇越. 网络内容分析中基于硬件的字符串匹配算法的研究[D]. 合肥: 中国科技大学, 2008.
- [6] LI Q Y. String Matching Algorithm Research based on Hardware in Network, Content Analysis[D]. Hefei: University of Science and Technology of China, 2008.
- [7] LIN C H, HUANG C T, JIANG C P, *et al.* Optimization of regular expression pattern matching circuits on FPGA[A]. Proceedings of Design, Automation and Test in Europe[C]. Germany, 2006. 1-6.
- [8] LE H, PRASANNA V. A memory-efficient and modular approach for large-scale string pattern matching[J]. IEEE Transactions on Computers, 2013, 62(5):844-857.
- [9] BECCHI M, CROWLEY P. A hybrid finite automaton for practical deep packet inspection[A]. Proceedings of ACM CoNEXT Conference[C]. New York, USA, 2007. 1-12.
- [10] YU F, CHEN Z, DIAO Y, *et al.* Fast and memory-efficient regular expression matching for deep packet inspection[A]. ACM/IEEE Symposium on Architecture for Networking and Communications System[C]. San Jose, California, USA, 2006. 93-102.
- [11] PASETTO D, PETRINI F, AGARWAL V. Tools for very fast regular expression matching[J]. Computer, 2010, 43(3):50-58.
- [12] QI Y, WANG K, FONG J, *et al.* FEACAN: front-end acceleration for content-aware network processing[A]. Proceedings of 30th IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies[C]. Shanghai, China, 2011. 2114-2122.
- [13] LIU T, YANG Y, LIU Y, *et al.* An efficient regular expressions compression algorithm from a new perspective[A]. Proceedings of 30th IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies[C]. Shanghai, China, 2011. 2129-2137.
- [14] ANTONELLO R, FERNANDES S, SADOK D, *et al.* Deterministic finite automaton for scalable traffic identification: the power of compressing by range[A]. Network Operations and Management Symposium (NOMS)[C]. Piscataway: IEEE, 2012. 155-162.
- [15] WOODS L, TEUBNER J, ALONSO G. Complex event detection at wire speed with FPGAs[J]. The VLDB Endowment, 2010, 3(1-2):660-669.
- [16] YANG Y H E, PRASANNA V K. Space-time tradeoff in regular expression matching with semi-deterministic finite automata[A]. Proceedings of 30th IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies[C]. Shanghai, China, 2011. 1853-1861.
- [17] Sourcefire, SNORT[EB/OL]. <http://www.snort.org/snort-downloads>, 2012.
- [18] International computer science institute, bro intrusion detection system [EB/OL]. <http://bro-ids.org/download/index.html>, 2012.

## 作者简介:



贺炜 (1988-), 男, 山西吕梁人, 国家数字交换系统工程技术研究中心硕士生, 主要研究方向为高速模式匹配技术、网络特征检测技术。

郭云飞 (1963-), 男, 河南郑州人, 国家数字交换系统工程技术研究中心教授、博士生导师, 主要研究方向为网络安全、宽带信息网络和高速路由器关键技术。

扈红超 (1982-), 男, 河南商丘人, 博士, 国家数字交换系统工程技术研究中心讲师, 主要研究方向为高速路由器关键技术、网络流量均衡技术和业务管控技术等。