第36卷 第19期 计 算 机 工 程 2010年10月
Vol.36 No.19 Computer Engineering October 2010
·开发研究与设计技术· 文章编号：1000—3428(2010)19—0269—03 文献标识码：A 中图分类号：N945

高性能正则表达式匹配算法评估

金军航[a]，张大方[a,b]，黄 昆[b]
(湖南大学 a. 软件学院；b. 计算机与通信学院，长沙 410082)

摘 要：为对现有的高性能正则表达式匹配算法进行综合比较与分析，实现诸如DFA、D^2FA、CD^2FA、mDFA及XFA等最新算法，采用Snort规则集综合评估这些算法的存储空间和匹配时间。实验结果表明，在存储空间方面，与mDFA相比，XFA的存储空间减少84.9%~89.9%；在匹配效率方面，与mDFA相比，XFA的匹配时间增加了38.9%~174.6%；XFA在存储空间和匹配效率上具有良好的可伸缩性，即当规则数增加到8倍时，mDFA的存储空间增长了64倍，而XFA的存储空间仅增加了16倍，匹配时间仅增加了61.3%。

关键词：正则表达式匹配；确定有限自动机；扩展有限自动机；性能评估

Evaluation of High-performance Regular Expression Matching Algorithms

JIN Jun-hang[a], ZHANG Da-fang[a,b], HUANG Kun[b]
(a. School of Software; b. College of Computer and Communication, Hunan University, Changsha 410082, China)

【Abstract】 In order to analyze and evaluate the existed regular expression matching algorithms, it implements them such as DFA, D^2FA, CD^2FA, mDFA and XFA, and conducts extensive experiments on short rules to evaluate the performance of these algorithms. Experimental results show that compared with mDFA, XFA achieves 84.9%~89.9% memory reduction; XFA only increases 38.9%~174.6% matching times in comparison with mDFA; when the number of rules increases by 8 times, mDFA increases 64 times memory space, while XFA only increases 16 times memory space and increases 61.3% matching time, which demonstrates that XFA has good scalability in terms of memory requirements and matching efficiency.

【Key words】 regular expression matching; deterministic finite automaton; extended finite automaton; performance evaluation

1 概述

正则表达式匹配算法应用广泛，特别是深度数据包检测。但是，随着网络带宽和流量的迅猛增长，正则表达式匹配算法面临高性能挑战。一方面，为了达到数据包线速匹配，研究者利用现代嵌入式存储器技术，提出了面向硬件实现的正则表达式匹配算法。另一方面，基于硬件的正则表达式匹配算法实现面临嵌入式存储空间受限的挑战。但是，传统的正则表达式匹配算法难以满足这些高性能需求。确定有限自动机(Deterministic Finite Automaton, DFA)匹配效率高，但存在存储空间爆炸问题；非确定有限自动机(Nondeterministic Finite Automaton, NFA)的存储空间高效，但存在匹配效率低的问题。因此，高性能深度数据包检测的关键是设计一种快速且存储高效的正则表达式匹配算法。

为了达到低存储空间需求和高效匹配，研究者提出了许多高效的正则表达式匹配算法。文献[1]提出多DFA(Multiple DFA, mDFA)，通过合理划分规则集，产生多个DFA的方式来解决DFA状态空间爆炸的问题。文献[2-3]提出的扩展有限自动机(Extend Finite Automaton, XFA)是在DFA中加入某些变量包括比特位和计数器，来精简自动机的状态。文献[4]提出了输入延迟DFA(Delayed Input DFA, D^2FA)，D^2FA通过增加默认边来压缩状态的迁移表；文献[5]发现D^2FA存在过多的内存访问，匹配效率低下，提出了内容寻址D^2FA(Content Adrresing D^2FA, CD^2FA)，通过哈希来快速定位下一状态，从而提高匹配效率。

在这些正则表达式匹配算法中，研究者很少有对这些重要的算法进行全面的实验评估及性能分析。本文采用C++设计和实现了DFA、D^2FA、CD^2FA、mDFA和XFA等最新正则表示匹配算法，采用Snort规则集综合评估了这些算法的存储空间和匹配时间以及可伸缩性。综合分析表明，XFA是存储空间和匹配效率的最好折衷算法。

2 实验评估

2.1 实验方法

在Miehela Becchi的工作基础之上，本文采用C++设计和实现了DFA、mDFA、D^2FA、CD^2FA、XFA，并运行在CPU为Intel Core 2 Duo 2.13 GHz、内存为2 GB的计算机上。实验评估各个算法的存储空间和匹配效率，以及在这2个方面上的可伸缩性。实验所需的正则表达式规则集来源于Snort，从中提取了其中的FTP规则集和部分HTTP规则集作为实验对象，并且，通过抓取HTTP数据包和FTP数据包进行数据匹配。

使用Snort2Bro工具，把Snort格式的规则转换为Bro格式，产生了79条FTP规则和406条HTTP规则。对每条正则表达式产生单个DFA，然后用DFA合并算法来产生组合的

基金项目：国家自然科学基金资助项目(60673155, 90718008)
作者简介：金军航(1985－)，男，硕士研究生，主研方向：网络安全；张大方，教授、博士、博士生导师；黄 昆，博士研究生
收稿日期：2010-01-08 **E-mail：**jinjunhang@hotmail.com

DFA。对于 mDFA 算法，把状态数上限设为 28 000，一旦 DFA 状态数达到这个限制，一个新的 DFA 将重新开始被构造。对于无法构造出 1 个 DFA 情况下，D^2FA 和 CD^2FA 利用 mDFA 产生的多个 DFA，分别构造出多个 D^2FA 和 CD^2FA。如果产生了多个自动机，实验对多个自动机采用串行匹配方式评估匹配效率。对于 XFA 算法，需要对规则加上注释和变量映射等配置信息来生成单个 XFA，然后再利用 XFA 合并算法来产生组合的 XFA。

采用存储空间和匹配效率 2 个方面来评估正则表达式匹配算法的性能。在存储空间方面，除了对比各个算法构造的自动机的状态个数和迁移边条数之外，还将采用基于比特位图的实现来估计出自动机所需的最小存储空间。通常，自动机的存储空间主要取决于 2 个因素 :状态个数和迁移边条数，而 XFA 还需要存储状态上的指令。通过以下规则来估计存储空间：(1)DFA，mDFA，XFA 的迁移表采用比特位图过滤出失效边，每个状态对此需要 32 Byte 开销；(2)D^2FA，CD^2FA 除了要过滤出失效边，还要过滤出默认边，每个状态需要两个比特位图，每个状态需要 64 Byte 开销；(3)每条迁移边使用 4 Byte 来指向下一状态迁移表的起始地址 ;(4)XFA 的每条指令包含 1 Byte 操作类型，4 Byte 操作数地址，所属规则编号需要 $\lceil \log M / 8 \rceil$ Byte，M 表示规则数大小。在匹配效率方面，采用 1 GB 数据所需的匹配时间(s/GB)来评估各个算法的匹配效率。

为了评估各算法在存储空间和匹配效率上的可伸缩性，从 406 条 HTTP 规则集中随机抽取规则数比例为 1/8,1/4,1/2 的规则集，每个比例抽取 50 个规则集。然后，对各个比例的规则集利用各自算法产生自动机，评估出各个比例规则集所需的存储空间和匹配时间。

2.2 实验结果

2.2.1 存储空间

表 1 是各算法在 FTP 规则集的存储空间相关的指标对比。从表 1 中可以看出，所有算法都能构造出单个自动机，其中，DFA 付出了最大的存储开销(DFA 个数为 1，mDFA 等同于 DFA)。与 DFA 相比，D^2FA 的迁移边条数减少了 93.4%，存储空间大小减少了 58.8%；CD^2FA 的迁移边条数减少了 64.4%，存储空间大小减少了 35.1%；XFA 的状态个数减少了 91.2%，由于状态数的减少，XFA 的迁移边条数减少了 90.3%，存储空间大小减少了 89.9%。XFA 的存储空间包括迁移表大小、变量和指令个数等，其中，迁移表大小为 71 KB。

表 1 各算法在 FTP 规则集的存储相关指标对比

算法	自动机个数	状态个数	迁移边条数	存储/KB
DFA	1	4 148	152 002	741
mDFA	1	4 148	152 002	741
D^2FA	1	4148	9 980	305
CD^2FA	1	4 148	54 089	481
XFA	1	363	14 775	75

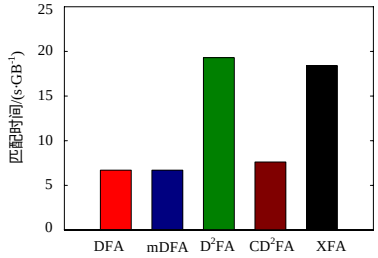
表 2 是各算法在 HTTP 规则集的存储空间相关的指标对比。由于 HTTP 规则集比较庞大，DFA 算法无法构造出单个 DFA，通过利用 mDFA 算法，探测出最大能够构造出 399 条规则的 DFA，表 2 所示的 DFA 是规则数为 399 的存储空间的情况。估计构造完成 406 条规则集的 DFA 大约需要 8 GB 的存储空间。mDFA 算法构造出 2 个 DFA，需要 10 600 KB 存储空间。表 2 指出：与 mDFA 相比，D^2FA 的迁移边条数减少了 94.9%，存储空间大小减少了 64.8%；CD^2FA 的迁移边条数减少了 79.3%，存储空间大小减少了 51.7%；XFA 的存储空间最小，与 mDFA 相比，状态个数减少了 89.1%，迁移边条数减少了 84.3%，存储空间大小减少了 84.9%。

表 2 各算法在 HTTP 规则集的存储相关指标对比

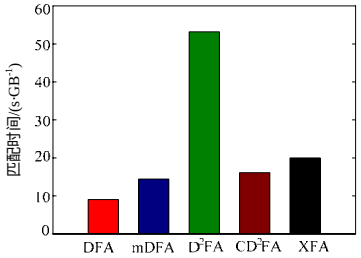
算法	自动机个数	状态个数	迁移边条数	存储/KB
DFA	1	>563 475	>33 099 580	>150 430
mDFA	2	51 250	2 227 867	10 600
D^2FA	2	51 250	113 011	3 730
CD^2FA	2	51 250	460 098	5 120
XFA	1	5 606	349 414	1 600

2.2.2 匹配效率

图 1 是各算法在不同规则集上的匹配时间对比。从图 1(a)中可以看出，对于 HTTP 规则集，DFA 拥有最好的匹配效率(DFA 个数为 1，mDFA 等同于 DFA)，1 GB 数据需要 6.7 s 的匹配时间。图 1(a)表明：与 DFA 相比，D^2FA 匹配时间增加了 188.1%，其原因是 D^2FA 需要进行大量默认边迁移；CD^2FA 改进了 D^2FA 匹配效率，但是需要付出哈希函数计算的开销，且与 DFA 相比，CD^2FA 匹配时间仅增加了 13.4%；由于 XFA 在状态迁移的同时执行状态上的指令操作，与 DFA 相比，XFA 的匹配时间增加了 174.6%。图 1(b)是各个算法在 HTTP 规则集上的匹配效率对比。由于 DFA 算法无法构造单个 DFA，图 1(b)所示的是笔者所能构造出的最大 DFA 的匹配效率，此时 DFA 完成 1 GB 数据匹配完成需要 9 s。mDFA 构造出 2 个 DFA，对 2 个 DFA 进行串行匹配，mDFA 匹配时间需要 14.4 s。图 1(b)表明，D^2FA 的匹配效率最差，与 mDFA 相比，其匹配时间增加了 269.4%；CD^2FA 匹配时间增加了 11.8%；由于指令的额外执行开销，与 mDFA 相比，XFA 匹配时间增加了 38.9%。



(a)FTP 规则集



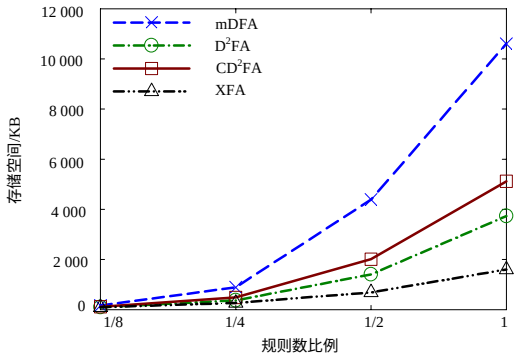
(b)HTTP 规则集

图 1 各算法匹配时间对比

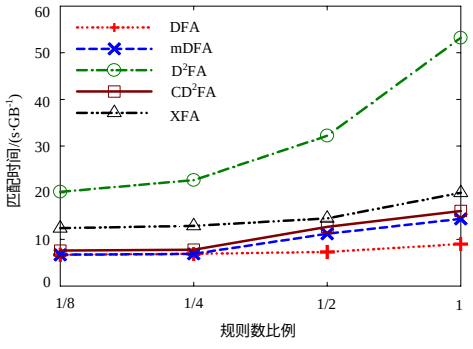
2.2.3 可伸缩性

图 2 为各算法的存储空间和匹配效率可伸缩性对比。由于 DFA 算法的存储空间增长太快，当规则数增加到 8 倍时，存储空间大约增长了 5 万倍，为了更清楚地映其他几个算法的比较，没有把 DFA 算法放在图 2(a)上比较。图 2(a)表明，当规则数增加到 8 倍时，mDFA 算法的存储空间增长最快，增长了 64 倍；D2FA 和 CD2FA 算法的存储空间增长较为平缓，D2FA 增长了 38 倍，而 CD2FA 增长了 44 倍；XFA 算法

的存储空间增长最为平缓，仅增长了 16 倍。图 2(b)为各算法的匹配效率可伸缩性对比。DFA 具有最好的匹配效率的可伸缩性，在规则集规模增长到 8 倍时，匹配时间只增长了 34.3%。DFA 的匹配时间增长的原因在于随着规模数增长，DFA 需要报告更多的匹配次数，以及 DFA 所需存储空间的增加，造成在状态迁移时引起 CPU 缓存命中率下降。XFA 算法的匹配时间较长，但是并没有随着规则集增大而出现急剧增加。当规则集增大到 8 倍时，XFA 匹配时间只增加了 61.3%。在规则集比例为 1/8 和 1/4 时，mDFA 算法都只产生 1 个 DFA，而在规则集比例为 1/2 时，有 40%的样本规则集 mDFA 产生了 2 个 DFA。因此，从图 2(b)中，发现在规则集规模比例从 1/4 增长到 1/2 和 1/2 增长到 1 时，匹配时间分别增长了 62.3%和 28.6%。由于 CD^{2}FA 和 D^{2}FA 是通过 mDFA 产生的多个 DFA 进行迁移边压缩，匹配时间也有类似的增加。CD^{2}FA 在规则集规模增长到 8 倍时，匹配时间增长了 111.8%，而 D^{2}FA 增长了 163.4%。由于随着规则集的增大，D^{2}FA 平均默认迁移边长增大，导致匹配时间增加的更多。



(a)存储空间



(b)匹配时间

图 2　各算法伸缩性对比

2.2.4　讨论

这些正则表达式匹配算法从不同角度来减少自动机所需存储空间，可以同时使用不同算法来减少更多存储空间。实验结果表明，XFA 提供了最好的存储空间和匹配效率的折衷，因此，可以选择以 XFA 为出发点，进一步减少它的存储空间和提高它的匹配性效率。XFA 极大地压缩了 DFA 的状态空间，但是并没有对减少迁移边数目。本文所提到的 2 种基于迁移表压缩的算法 D^{2}FA 和 CD^{2}FA，D^{2}FA 具有高效的存储空间，但是匹配效率下降太多。而 CD^{2}FA 的存储空间虽没有 D^{2}FA 高效，但是匹配效率要比 D^{2}FA 好很多。因此，可以采用 CD^{2}FA 来压缩 XFA 的迁移表，可以在匹配效率下降很少的情况下，进一步减小 XFA 的存储空间。另外，也可以从基于字母表压缩的角度来减少 XFA 的迁移表，但这种方法需要付出一次查询字母表的代价。

同时也发现并行化 XFA 可提高其匹配效率。XFA 匹配时需要进行状态迁移，然后执行状态上的指令，而变量值的修改以及正则表达式的接受都依赖于指令的执行。因此，可以确保在指令执行顺序不变的情况下，并行化 XFA 的状态迁移和指令执行，以提高其匹配效率。

3　结束语

本文实现并评估了 DFA、D^{2}FA、CD^{2}FA、mDFA 和 XFA 等正则表示匹配算法的存储空间、匹配时间以及可伸缩性。实现结果表明，XFA 是最高效的有限自动机，即与 mDFA 相比，XFA 在存储空间上减少了 84.9%~89.9%，在匹配时间上仅增加了 38.9%~174.6%；XFA 具有良好的可伸缩性，即当规则数增加到 8 倍时，mDFA 的存储空间增长了 64 倍，而 XFA 的存储空间仅增加了 16 倍，其匹配效率仅下降了 61.3%。综合分析表明，XFA 是存储空间和匹配效率的最好折衷算法。

本文的未来工作是进一步减少 XFA 的存储空间，并提高 XFA 的匹配效率。在减少存储空间方面，采用 CD^{2}FA 或者字母表压缩算法压缩 XFA 的迁移表；在提高匹配效率方面，并行化 XFA 的状态迁移和指令执行。此外，在 ASIC 或 FPGA 等专用硬件上实现 XFA 及其改进。

参考文献

[1] Yu Fang, Chen Zhifeng, Diao Yanlei, et al. Fast and Memory-efficient Regular Expression Matching for Deep Packet Inspection[C]//Proc. of the 2006 IEEE Symp. on Architectures for Networking and Communications Systems. San Jose, CA, USA: [s. n.], 2006: 93- 102.

[2] Randy S, Cristian E, Somesh J. XFA: Faster Signature Matching with Extended Automata[C]//Proc. of IEEE Symposium on Security and Privacy. Oakland, CA, USA: [s. n.], 2008: 187-201.

[3] Randy S, Cristian E, Somesh J. Deflating the Big Bang: Fast and Scalable Deep Packet Inspection with Extended Finite Automata[C]//Proc. of ACM SIGCOMM'08. Seattle, WA, USA: [s. n.], 2008: 207-218.

[4] Kumar S, Dharmapurikar S, Yu Fang, et al. Algorithms to Accelerate Multiple Regular Expressions Matching for Deep Packet Inspection[C]//Proc. of ACM SIGCOMM'06. Pisa, Italy: [s. n.], 2006: 339-350.

[5] Kumar S, Turner J, Williams J. Advanced Algorithms for Fast and Scalable Deep Packet Inspection[C]//Proc. of Architectures for Networking and Communications Systems. San Jose, CA, USA: [s. n.], 2006: 81-92.

编辑　陈　文