

基于进程通信语义的互联安全控制安全分析和优化

卢波, 陆月明

(北京邮电大学网络空间安全学院, 可信分布式计算与服务教育部重点实验室, 北京 100876)

摘要: 面对天地一体化信息网络多域安全控制需求, 针对单进程正则匹配的方式制约了互联安全网关的性能问题, 提出基于进程通信语义安全分析的互联安全控制高性能优化方法。该方法采用以进程概念为中心的并行程序设计语言描述主从进程的交互模型, 通过赋予状态转移系统语义, 实现主从进程的组语义精确的表达。通过形式化语义分析, 该方法具有运行无死锁、饥饿, 且等价于原有正则匹配的语义安全性。实验结果表明, 该方法能够充分发挥多核硬件优势, 使用进程绑定CPU方式, 减少操作系统调度器的上下文切换开销, 在复杂正则表达式情形下, 比单核运行提高一个数量级。

关键词: 形式化分析; 正则表达式匹配; 访问控制; 并行计算

中图分类号: TN918.91

文献标识码: A

doi: 10.11959/j.issn.2096-8930.2021029

Optimization and Security Analysis of Interconnected Security Control Based on Semantic of Process Communication

LU Bo, LU Yue ming

Key Laboratory of Trustworthy Distributed Computing and Service, Ministry of Education, School of Cyber Security, Beijing University of Posts and Telecommunications, Beijing 100876, China

Abstract: Facing the demand of multi-domain security control in the integrated network of air and ground, to solve the problem that single-process regular matching restricts the performance of interconnected security gateway, the method of high-performance optimization of interconnected security control based on semantic security analysis of process communication was proposed. The interaction model of master-slave processes was described in a parallel programming language in the method. The state transition system semantics were given to accurately express the combined semantics of the master-slave processes. The security of master-slave processes in combinatorial semantics was analyzed, which ensured that the parallel running of processes had no deadlock and starvation, and was equivalent to the original regular matching. Experimental results showed that the proposed method could give full play to the advantages of multi-core hardware and reduced the context switching overhead of the scheduler. In the case of complex regular expressions, the method was an order of magnitude better than single-core running.

Keywords: formal analysis, regular expression matching, access control, parallel computing

1 引言

天地一体化信息网络需要强大的并行运行机制支持, 具体表现为安全互联网大容量、高吞吐率、高度个性化定制。在多域互联安全控制框架中, 通过细粒度匹配时间和特征实现上述功能。进一步, 需要优化匹配

过程中复杂度较高的特征匹配过程, 即正则表达式匹配, 来提高并行运行效率。

采用并行的程序设计有利于充分利用硬件资源, 发挥计算潜力, 并能够实现弹性扩展, 使得安全互联网关可以发挥更加灵活的作用。并行程序计算打破了程序顺序执

收稿日期: 2021-06-03; 修回日期: 2021-08-24

通信作者: 陆月明, ymlu@bupt.edu.cn

基金项目: 国家重点研发计划资助 (No.2016YFB0800302)

Foundation Item: National Key R&D Program of China (No.2016YFB0800302)

行的结果封闭性。并行程序的执行结果受到了程序执行环境的影响。程序执行先后、运行快慢,在并行程序描述中,都是被允许的不同程序行为。为了确定程序运行的不同行为不会影响到并行程序关注的计算结果,需要采用并行程序的形式化语义方法来分析并行程序的语义性质。采用形式化语义的方法分析该方案可以在确保进程并行运行安全的基础上实现既定的安全控制功能。

针对进程并行加载正则表达式匹配数据包问题,提出一种高性能的并行匹配方法,并采用形式化语义的方法设计实现了这种方案。通过建立主从进程形式化模型,赋予标签转移系统语义,分析了主从进程方案的死锁、饥饿、计算等价性等一系列语义安全性,并通过实验验证了该方案的高效性。下面介绍该方案的具体内容。

2 背景分析

在多域网络互联安全控制的细粒度特征匹配过程中,举一个特征子树例子:

```
287485200-287488800,{"https://":accept; "http://":drop; "ftp://":drop; "sftp://":accept;}
```

其中, 287485200-287488800是时间特征,表示每天9:00-10:00^[1]。这里关注的是子树根下面的特征节点,该节点由正则表达式和过滤数据包动作两个部分组成。在互联网安全网中,表示的意思是如果数据包内容包含了特征"http://",那么就接受该数据包。同样接受的特征还有"sftp://".如果数据包内容包含了"http://",那么就丢弃该数据包。同样丢弃的数据包还包括特征"ftp://".

已有的互联安全控制方案是采用单进程顺序依次执行正则表达式"https://""http://""ftp://""sftp://"来匹配数据包内容,如果某一个正则表达式匹配数据包成功,则按照该正则表达式对应的动作执行数据包的接受或者丢弃动作。单进程只能利用一个CPU资源进行正则表达式匹配数据任务,限制了互联安全控制性能的进一步提高。如果多个进程并行执行子匹配树中的多个正则表达式匹配数据任务,可以减少正则表达式引擎匹配数据的时间。这就需把正则表达式匹配数据的任务从原来的互联安全控制主进程中分离出来,每一个正则表达式进程并行匹配数据包。主进程负责协调指派从进程资源到对应的正则表达式,并把每一个从进程的正则表达式匹配数据结果汇总,从而计算出数据包是接受还是丢弃。

2.1 相关工作

通过前面分析正则表达式匹配数据包内容可以看出,当前需要解决的问题有以下几个方面:在已有的互联安全控制方案中,单进程的正则表达式匹配数据包特

征限制了网关吞吐量的进一步提升;如何充分挖掘现有核硬件的潜力、选择最具有性价比的正则表达式引擎,成为进一步优化网络互联安全控制方案的关键问题;将顺序执行的方案优化为并行执行的方案,带来了同步问题;由于并行执行打破了执行结果的唯一性,需要谨慎地设计并行运行机制避免错误的执行;如何在并行进程环境下协调多个进程有序、高效地完成正则表达式匹配数据包,需要对并行执行过程进行跟踪分析,对所有的可能进行考虑,确保最坏的情况一定不会出现,以及乐观的情况一定会出现;根据正则表达式库的特点,结合实际情形,灵活选用最适合的方案,在效率、体积、功能之间实现平衡。

在并行程序设计中,涉及多个程序之间的交互问题。对于传统的计算式行为而言,图灵机和 λ 演算等概念已经做了非常经典的描述。与之对比,交互式行为的描述对计算机科学构成了挑战^[2]。最重要的问题之一是同步问题,并行意味着互相之间保持独立,在空间和时间上没有对应关系,但是计算的要求往往在某个状态和时刻,彼此又需要联系和沟通,这就需要握手通信。进程之间端口/港口通信和程序调用有类似的概念。程序调用是使用了递归和堆栈的方法描述这个过程。参考文献[3-4]提出了通信系统演算(Calculating of Communication Systems, CCS)的概念,提出了进程作为描述并行交互式行为的中心概念。通信系统演算中定义了进程的基本运算和相应的标签转移系统语义。

参考文献[5]对进程代数做了系统的研究,采用特有的逻辑和语义有力地刻画了进程代数的行为。把并行计算概念用来描述计算机网络中常用的协议行为,可以归纳出一般的协议语义分析方法。与通信系统演算类似,参考文献[6-7]提出了分布式程序设计语言通信顺序进程(Communicating Sequential Processes, CSP)。通信顺序进程也是对并行程序运行的一种刻画。自动机理论为程序设计的描述提供了一种机器形式的规范。在并行程序的设计需求下,自动机做了进一步扩充,一般化为标签转移系统(Labelled Transition System, LTS)。在后续发展中,基于这些语义概念一直做更多扩充。

KIM等^[8]提出了分层并发有限状态机模型。其本身支持并发性、分层分解和事件处理。此外,分层并发有限状态机是一个用户友好的图解表示模型。该模型的操作语义是在两个基本假设下定义的:同步假设和零时间假设。这个简单且理论上的假设适用于时间关键的任务,而不是通过检测事件和确定行为来实现高级行为协调(包括一些操作时延)。程序的语言层面涉及的多是运算性质、语言结构的语义分析。在更高层的应用设计

上,更多是面向实际功能需求的角色访问控制和功能单元对象。统一建模语言(Unified Modeling Language, UML)是使用广泛的语种。统一建模语言与对象约束语言(Object Constraint Language, OCL)一起,允许在系统实现之前对其进行精确的描述。同时,可以使用这些描述来检查一致性,从而检查给定UML/OCL模型的正确性。人们为此提出了许多(自动化的)方法^[9]。系统的行为通常通过序列图、状态机和活动图来考虑。随着契约式设计的日益流行,组合结构、类和操作也经常被用来描述UML/OCL中的行为。这些描述意味着目前还没有验证并发行为的解决方案。PRZIGODA等^[10]提出了一种解决方案,讨论了根据通用的UML/OCL解释可接受的对“并发”的可能解释;然后,提出了一种利用可满足模理论求解器来检查UML/OCL模型并发行为的方法。更进一步的,由于并行系统具有潜在的指数数量的执行交错序列,SHIROLE等^[11]提供了一个好的测试套件来揭示所有错误类型的测试场景。选择测试场景来发现上述错误需要对测试场景进行系统分析。每个执行交错序列都可以作为测试场景来评估系统的正确性。指数交错空间和非确定型执行使并发系统测试成为一项具有挑战性的任务。检查的并发错误包括同步、数据竞争、数据不一致、饥饿和死锁。

GADEA等^[12]利用有限状态机(Finite State Machine, FSM)理论,将客户—服务器交互过程建模为实时系统。数据的一致性由FSM的形式组来控制。分层有限状态机用于定义和模拟客户端和服务组件在处理和转换用户发起更改时的实时行为。基于FSM实现的随机输入测试方法被证明有助于组织和管理复杂分布式结构。分布式并行设计往往是多个进程单独设计,然后组合分析。PAZZI等^[13]讨论了基于状态的建模形式中的模块化和组合性问题,并介绍了相关的最新研究成果。部分一整体状态图为传统的状态图提供了模块化的构造,以便允许行为抽象的增量和完全可重用的组合,显式地执行协调的系统行为,并为后续的建模和实现阶段带来好处。部分一整体状态图具有可计算的语义,可以通过约束驱动的规范方法来指定。这种方法允许在设计时直接指定和验证状态的预期含义,从而避免使用不那么有效的验证技术,如穷尽性测试或模型检查。

并发控制是事务管理的重要组成部分,它保证了共享数据项的正确性。一方面,现有的并发控制技术大多采用锁定机制来实现并发控制,从而导致事务饥饿和死锁。另一方面,非锁定技术(乐观排序和时间戳排序)与高流产率和过多的事务重启有关^[14-15]。BAKURA等^[16]提出了一种无锁混合同步并发控制策略,该策略结合了乐观

策略和时间戳排序策略的特点。该技术允许移动设备自由读取数据项,并允许在断开模式下预提交,并在全局承诺的重连接期间传播预提交的数据。利用设计的混合并发模拟器,以事务流产率和响应时间为性能指标,对新方案的性能进行了评估。仿真结果表明,无锁混合策略在最小化事务流产率和响应时间方面优于乐观策略和时间戳策略。

在正则表达式匹配方面,正则表达式和正则语言以及自动机联系在一起,用确定型有限自动机(Deterministic Finite Automation, DFA)、非确定型有限自动机(Nondeterministic Finite Automation, NFA)以及 ϵ -非确定型有限自动机(ϵ -Nondeterministic Finite Automation, ϵ -NFA)赋予了正则表达式的语义,并且用子集构造法实现了转换对应。对正则语言的理解有了语义上的认识。RATHNAYAKE等^[17]改进了正则表达式的动态匹配过程,使用操作语义将正则表达式匹配的实现技术形式化。首先,在操作语义中添加一个延续,以描述当前表达式之后需要匹配的内容。接下来,使用指针将表达式表示为数据结构,这使得通过测试指针是否相等可以消除冗余搜索。ZENG等^[18-19]提出了一种基于正则表达式的测试建模理论来构建并发系统的测试模型,开发了一个轻量级的建模工具正则有限状态机来支持使用正则表达式的可视化建模。它支持生成7个描述软件行为的操作符,以及两种构造表达式模型的方法,包括正则有限状态机代码编辑器和工具栏编辑器。

2.2 贡献

本文主要的贡献在于,针对天地一体化信息网络中网间互联安全控制的部分环节性能较低的问题,提出了一种基于进程通信语义安全分析互联安全控制高性能优化方案。网间互联安全控制细粒度数据包过滤包括了五元组、时间、特征这几个维度。在匹配数据包特征的过程中,采用正则表达式引擎来匹配。由于存在多个正则表达式匹配数据包任务,已有的匹配过程采用顺序依次执行的单进程方式来完成。

为了充分发挥硬件潜力,优化多个正则表达式匹配数据包过程,本文具体采用了主从多进程共享内存通信的正则表达式引擎并行匹配数据包方法。为多进程并行的正则表达式匹配数据包过程建立形式化模型,并赋予相应的标签转移系统语义模型。通过形式化语义分析的方法确保设计无死锁、无饥饿、计算过程等价。采用CPU绑定的方式测试多进程并行正则表达式匹配数据包实验,验证了方案的安全高效。

参考文献[3-7]从不同的角度提出了进程通信语言,

没有用有限状态自动机作为多进程通信的语义,不能够直接转换成程序。参考文献[8-10]在进程通信中包含程序对象表达,能够直接转换为具体程序。其中,参考文献[8-9]对并行运行行为缺少正确性验证。参考文献[10-11]对并行运行常见并发错误提供了验证方法,但是只能对给定进程数量的并行运行做安全验证。参考文献[12-13]及参考文献[17-19]使用有限状态自动机作为进程运行语义,使用了不同的策略来优化模型,但是不支持进程并行运行。参考文献[14-16]对同步锁做了优化,能通过采用轻量化的设计提高进程间通信效率。这些方案没有用有限状态自动机作为多进程通信的语义,不能够直接转换成程序。和以上方案相比,在任意数量进程的情形下,本文采用有限状态自动机提供主从进程通信语义安全性验证,通过共享内存的方式实现主从进程之间的通信。

3 并行运行形式化

对进程并行运行的设计和分析验证需要对进程进行形式化,把进程之间的通信关系表达出来。采用通信系统演算的思想描述进程并行运行的状态,所有的元素都是基于进程的组构造的。进程由一系列动作描述进程关系。进程表述如公式(1)所示。其中0是一个空进程,它的特殊之处在于不接收任何动作和跳转。

动作Action是进程对应标签转移系统语义所接收的信号。转移关系体现在进程表达式中。从单个进程来讲,接收这个信号,就跳转到后面进程对应的状态。进程 P 可以用一个标识符 id 来代表。形式为 $P::id="P"$ 。为了和动作信号区分,进程一般用大写字母表示。动作信号一般用小写字母来表示。接收动作信号转移可以表示为 $Action \rightarrow$ 的形式,后面可以跟一个进程,意思是接收该动作信号后跳转到这个进程。结合前面的0进程,可以定义经过一些动作最终跳转到0的进程 $ZERO=a \rightarrow 0$ 。进程 $ZERO$ 意思是接收动作信号 a 就跳转到0进程状态。特别地,如果进程跳转的终点是进程自己,说明这个进程是一个循环跳转的进程,具体形式为 $P=a_1 \rightarrow a_2 \dots a_n \rightarrow P$ 。例如 $B=a.b \rightarrow B$ 意思就是 B 进程接收动作信号 $a.b$ 就跳转到 B 进程对应状态;而它又可以接收这个动作信号 $a.b$,反复循环。

除了上面的顺序执行进程以外,还需要表达并行执行进程的情形。首先是选择运算“+”,代表进程选择其中一个路径执行。例如 $CHOICE=A+B+C$,意思是进程 $CHOICE$ 执行的时候,会随机选择执行进程 A 、 B 或者 C 。其次是并行运算“||”,代表多个进程同时执行,没有先后关系,相对保持独立。例如 $PARALLEL=A \parallel B$,意思是进程 A 和 B 都在执行。

$$\begin{aligned}
 P ::= & \quad 0 & | & \quad id = "P" \\
 & \quad Action \rightarrow "P" & | & \quad id["id"] \\
 & \quad P + "P" & | & \quad P || "P" \\
 & \quad P["id"/"id"] & & \\
 & \quad P \setminus id & & \\
 & \quad Array : "P" & | & \quad id = "(P,)*P" \\
 & \quad Array = "when"("Expression")P & & \\
 & \quad Array = "when"("Expression")P & &
 \end{aligned} \tag{1}$$

并行执行运算会有一些复杂的组合作用。例如两个并行的进程存在相同的动作时,表明进程并行执行过程中有同步的约束。例如 $A=a \rightarrow b \rightarrow A$ 、 $B=c \rightarrow b \rightarrow B$,由于进程 A 和 B 都有动作信号 b ,因此在并行组合 $A_B=A \parallel B$ 以后,进程 A_B 不仅是并行执行,而且要求某一个进程在可以接收信号 b 状态时,要等待其他进程,一直到所有进程都到了接收该信号时,才能一起接收该信号并跳转到各自的相应状态。这里并行进程 A_B 就跳转到了 A 和 B 各自跳转后状态的组合状态,这里也是 A_B 初始状态。在初始状态, A 进程可以接收 a , B 进程可以接收 c ,因此 A_B 初始状态可以接收 a 或者 c 。如果初始状态接收到 a ,就跳转到 A 进程,接收信号 a 跳转、 B 进程保持的状态。在这个状态下,虽然 A 进程可以接收 b , B 进程可以接收 c ,但由于进程 B 也要在接收 b 的时候发生跳转,因此 A 进程必须等待 B 进程跳转到可以接收 b 的时候,才能接收 b 。所以当前状态只能让进程 B 接收 c 跳转、 A 保持。在接收 c 之后, A 进程和 B 进程都到了可以接收 b 的状态,这个时候就可以接收 b 跳转到初始组合状态。依次类推,可以得出其他剩余的组合状态跳转关系。

由于进程 A 和 B 存在相同的信号 b ,因此两个进程在并行运行的时候,需要做一个同步机制,以此来确保所有进程同步接收共有的信号。在实际的进程中,可能还有很多内部的动作转移细节,由于这些动作和其他并行进程没有同步交互关系,因此把这些动作统一为不可见动作。隐藏的不可见动作记为 τ 。这个动作用于简化形式模型,便于进一步的语义分析。

为了便于量化同一个进程模板下的进程实例,提供了数组量化Array。数组是通过数组名称以及范围来定义的。范围表示为 $[var:start..end]$, var 是一个遍历数组成员的变量,后面是由数组起始值和终止值表示的数组下标范围。数组 $s[i:1..5]$ 所表示的意思是数组 s 的下标为从1到5,遍历数组的变量为 i 。数组可以通过下标引用某一个成员,形式为 $id[id]$ 。例如 $s[3]$,表明引用了数组 s 中的3号元素。

数组表示不可避免地需要用区间范围来表达。进程数组涉及表达进程数量、资源数量问题。例如进程池中的进程数目,子进程作为相同行为的执行体集合,可以

使用模板实例化的方式创建多个相同的实例。涉及数量问题的形式化包括如下内容：

```
Constantdef::="const" id="Expression
Rangdef::="range" id="Expression"..Expression
Setdef::="set" id="{id(id)*}"
```

区间范围需要用常量表达起始和结束位置。常量Constantdef用于提供便利的进程数组范围定义，需要用到循环多个进程定义某种性质的时候，用Rangdef提供区间定义机制，一些离散方式定义的元素集合，用Setdef来确保可以被引用。为了计算方便，能够体现简单的量化关系，上述概念定义的时候，可以使用基于自然数的表达式计算，支持+、-以及括号运算。

```
Expression ::=          num
                |          id
                | Expression "+" Expression
                | Expression "-" Expression
                | ("Expression") (2)
```

数组可以修饰进程，用符号:修饰进程，表示以进程为模板，添加数组名称前缀变为进程数组。例如 $s[i:1..5]$ 表示5个以进程 B 为模板的进程数组已经创建，它们的信号用前缀“ $s[i].$ ”来做区分，例如 $s[1].a$ 、 $s[2].b$ 等信号。进程 B 的定义为 $B=c \rightarrow b \rightarrow B$ 。进程数组 $s[i:1..5]:B$ 中，第 i 个进程相当于 $s[i].B=s[i].c \rightarrow s[i].b \rightarrow s[i].B$ 。在数组修饰动作时，用符号“.”修饰动作，例如 $C=s[i:1..5].c \rightarrow C$ ，说明进程 C 可以接收5种信号，并跳转到初始状态。数组修饰进程时，数组中的进程成员之间不会有同步需求。如果有需求，还需要数组成员之间的同步关系做表达。因此，需要数组自己可以定义为进程，例如形式为 $T[b:1..5]$ ，表示数组中有进程 $T[1] \cdots T[5]$ 。进程数组成员之间的跳转关系可以用动作加下标引用表示，这样的定义用符号“,”分割。例如 $S=T[5], T[b:1..5]=\text{when}(b<5) a[b] \rightarrow T[b+1]$ ，意思是接收信号 $a[b]$ 会跳转到 $T[b+1]$ ，例如接收信号 $a[4]$ 会跳转到 $T[5]$ 。when关键字对适用的条件做了限定，要求 b 小于5，结合 b 的范围在1到5之间，得出 b 的可用范围是1到4。对于直接定义的数组进程，不能直接使用，需要指定某个数组进程成员作为一个起始状态，所以数组进程往往作为其他进程的内部构造。从进程 S 中可以看到起始状态是 $T[5]$ 。起始状态和数组定义用逗号“,”连接在一起作为进程 T 的定义。

```
Action::=id("id"*Array)."Action (3)
```

```
Array::=id "["id"."(id|id)"]" (4)
```

```
id::=[A-Z,a-z,_"[A-Z,a-z,0-9,_"* (5)
```

```
num::=[0-9]+ (6)
```

在规则匹配过程中，会有规则匹配树下的多个匹配正则表达式分支来匹配同一个数据包内容。这就涉及通过多个进程同时开始载入正则表达式和匹配数据段的过程。这里面涉及进程状态切换的表述，以及主进程和子进程调用的过程。主进程要等待子进程计算结束，汇总结果然后再继续后续工作。这个过程中间的子进程受数目影响，它和匹配树分支存在分配实体占用资源的问题。主进程在高效匹配正则表达式过程中需要事先建立进程池来实例化规则匹配树。进程池的好处在于不用频繁地创建和销毁进程。而子进程需要主进程主动地从进程池中申请使用，并在匹配结束之后归还资源供后续使用。

3.1 主从进程形式化建模

在有了数量支持之后，就可以进一步构建正则表达式的并行运行结构。正则表达式匹配在互联安全控制网中首先是接收到数据包，经过五元组匹配和时间匹配之后进入特征匹配树。由于子树根节点下面有多个正则表达式匹配被进程载入，且表达式之间互斥没有交集，因此可以并行处理。主进程从进程池中请求到进程载入正则表达式，匹配数据内容结束之后，按照匹配成功的条目执行数据包拦截动作，然后循环接收下一个数据包，重新开始进程加载正则表达式匹配数据过程。这个进程建模为如下形式

```
MASTER=packet→slave.start→τ→slave.join→MASTER (7)
```

主进程通过接收packet信号开始匹配数据包的正文内容，通过调用子进程开始匹配， τ 隐藏了中间匹配的详细过程。信号slave.join说明匹配已经结束，开始接收下一个数据包做匹配，如此循环反复。

相应的，子进程通过主进程申请资源实例化，加载正则表达式以匹配数据包。公式(8)模拟了实际的进程状态转换过程，在这里表达为如下形式

```
SLAVE=slave.start→slave.rotate→slave.join→SLAVE (8)
```

子进程通过和主进程共有的信号slave.start、slave.join实现协作运行。考虑到子进程作为一种资源被主进程所使用，还要模拟具体的子进程数量和正则表达式匹配数量，可将子进程合理地分配给正则表达式。所以有如下常量定义

```
const RN=3 \ 正则表达式匹配个数
range RRN=0..RN (9)
```

```
const TN=4 \ 子进程个数
range RTN=0..TN (10)
```

将子进程按照模板实例化,只需在前面加上范围区间前缀。形式为: $s[i:RTN]:SLAVE$ 。这样就有了 TN 个 $SLAVE$ 进程并行运行。为了表示正则表达式消耗可用子进程的关系,需要一个监视器来维持资源占用关系。当启动某个子进程 i 的时候,对应的资源消耗掉一个。用 $s[i:RTN]:slave.start \rightarrow REGEXP MATCH[b-1]$ 形式来表示。与此同时,用 $when(b==0)packet \rightarrow REGEXP MATCH[RN]$ 和 $s[i:RTN]:slave.join \rightarrow REGEND[b-1]$ 表示转移条件。当所有子进程正则表达式匹配数据包内容完成,才能继续匹配下一个数据包。具体形式如下

$$\begin{aligned}
 MASTER &= REGEXP MATCH[RN] \\
 REGEXP MATCH[b:RRN] &= (when(b>0)s[i:RTN].slave.start \\
 &\rightarrow REGEXP MATCH[b-1] \\
 &| when(b==0)arrive \rightarrow REGEND), \\
 REGEND &= REGEND[RN] \\
 REGEND[b:RRN] &= (when(b>0)s[i:RTN].slave. \\
 &join \rightarrow REGEND[b-1] \\
 &| when(b==0)packet \rightarrow \\
 ®EXP MATCH[RN])
 \end{aligned} \quad (11)$$

这样,采用“ $\parallel$ ”将两个进程组合并行运行,就得到了主从进程并行匹配数据包的过程,形式为 $MASTER_SLAVE=(MASTER \parallel s[i:RTN]:SLAVE)$ 。

3.2 正则匹配形式化建模

正则表达式匹配用正则语言表示,它能够和确定型有穷自动机以及非确定型有穷自动机等价。形式化的定义非常简单。在公式(12)~公式(13)中,符号 ϵ 表示空串,符号 a 表示字符,在计算机实际使用中涉及转义字符,因此会稍显复杂。具体表示为

$$\begin{aligned}
 \alpha &::= "" | [\text{charset} + "] \\
 \text{charset} &::= \text{char} - \text{char} \\
 &| \text{char}
 \end{aligned} \quad (12)$$

其中, char 在不同操作系统、程序设计语言中会有不同的表示方法。如果字符可以直接显示且不产生歧义,那么就无须转义,否则用“ $\backslash$ ”开头加前缀 $0x$ 或者 0 表示进制以及用具体的数字字母编码来表示这个字符。

通过建立主从进程并行载入正则表达式匹配数据包以及正则表达式的形式化模型以后,就可以进一步建立相应的语义为后续的性质分析做评估基础。

$$\begin{aligned}
 \text{RegExp} &::= \epsilon \\
 &| \alpha \\
 &| \text{RegExp} \text{ "?"} \\
 &| \text{RegExp} \text{ "+"}
 \end{aligned}$$

$$\begin{aligned}
 &| \text{RegExp} \text{ "*"} \\
 &| \text{RegExp} \text{ "."} \text{ RegExp} \\
 &| \text{RegExp} \text{ "|" } \text{RegExp} \\
 &| \text{"(" } \text{RegExp} \text{ ")" }
 \end{aligned} \quad (13)$$

公式(13)定义了正则表达式的语法规则。它支持数量描述运算、连接运算、合并运算以及括号运算。接下来把进程通信语言用具体的语义模型来解释。

4 赋予语义模型

通过标签转移系统可以直观地理解形式化模型的具体执行过程。其组成要素分为字母表 Σ 、状态集合 Q 、转移关系 $\rightarrow$ 。状态集合 Q 表示系统中所有的状态。转移关系表示某状态通过标签上的字符接收转移到另外一个状态。转移关系是一个状态关系集合。

主从单进程语义图如图1所示。

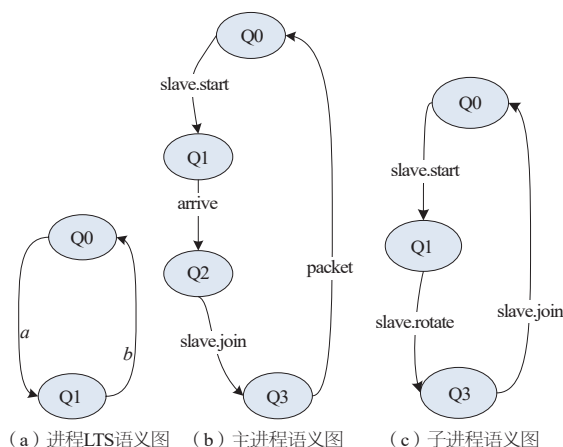


图1 主从单进程语义图

字母表、状态集合、转移关系三要素构成了标签转移系统。用这个抽象机器来赋予并行运行的主从进程语义。其中 0 进程对应为无任何动作的LTS,常见的进程定义形式为 $P=a \rightarrow Q, Q=b \rightarrow P$ 。这个进程对应的LTS为字母表 $\Sigma=a, b$,包含了 a 、 b 符号,接收符号 a 转移到 Q ,再接收 b 转移到 P 。详情如图1(a)所示,它构成一个循环运行的机器。

对于进程中的运算符语义,结合LTS表示可以理解运算符性质。其中选择运算“ $+$ ”,表示两个标签上的动作会执行其中的一个,根据的是当前系统接收到的信号,在LTS上表现为不同符号的两个分支。替换“ $/$ ”将对应的符号替换成另外一种表示方式,这为两个进程产生联系提供便利。约束运算“ $\backslash$ ”屏蔽掉相应的动作行为,在组合系统中不考虑这部分标签符号。并行运算

“||”表示的是两个进程并行运行，语义上表现为两个LTS的组合。其状态空间由原来的一维扩展为二维。组合LTS接收信号分别跳转到各自的状态。

在主进程中，对应的状态转移所表示的含义就是接收到packet信号开始转到正则表达式匹配；匹配结束之后，转移到初始状态继续等待packet信号循环执行。图1(b)、图1(c)说明了主从进程的状态转移关系。

通过并行运算将两个进程组合在一起，且子进程做了多个实例化进程并行运行。接下来把主从进程组合起来，先考虑最简单的情形。图2为只有一个正则表达式、一个子进程的情形，然后逐步增加正则表达式和进程资源数。图3是一个正则表达式的情况下，两个子进程的并行运行。

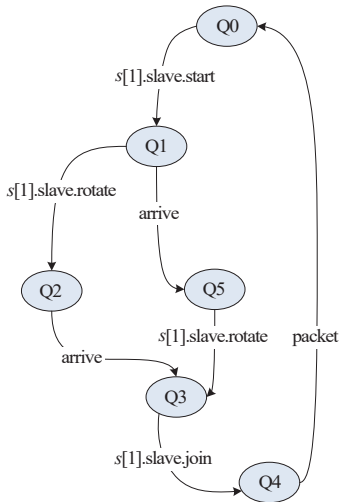


图2 单进程组合语义图

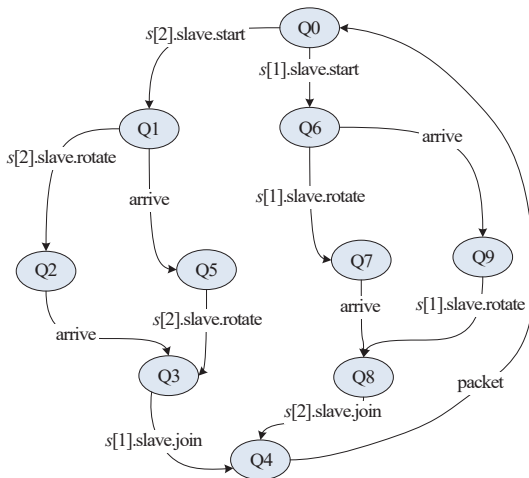


图3 两个子进程组合语义图

考虑到充分发挥多核CPU优势以尽可能优化性能，这里总是考虑进程资源池充足的情形，确保每个正则表达式几乎同时开始进行匹配，而不需要等待进程资源分配的情形。接下来图4、图5考虑两个正则表达式，进程资源数目分别是两个和三个的情况下，正则表达式如何分配进程资源匹配数据包内容。从图4、图5中可以看到状态的转移可能性空间呈现几何级数的增加，很快变得不容易跟踪分析。

将图5更改成更加紧凑的布局，因此有部分边和节点重合。可以直观看到这些图有一个相同的特点，那就是所有正则表达式在匹配完成之后，状态机通过标识packet符号的边转移到初始状态Q0。从这个观察中可以直观感受到该类型组合的LTS语义构成的系统满足一个性质。那就是只有所有的正则表达式匹配完成，然后才能响应packet符号，且只会跳转到初始状态进行新一轮的匹配，而不是无序地开启其他子进程。与此同时，其他开启的进程*i*一定是全部已经接收s[i].slave.join动作信号，完成状态转移。

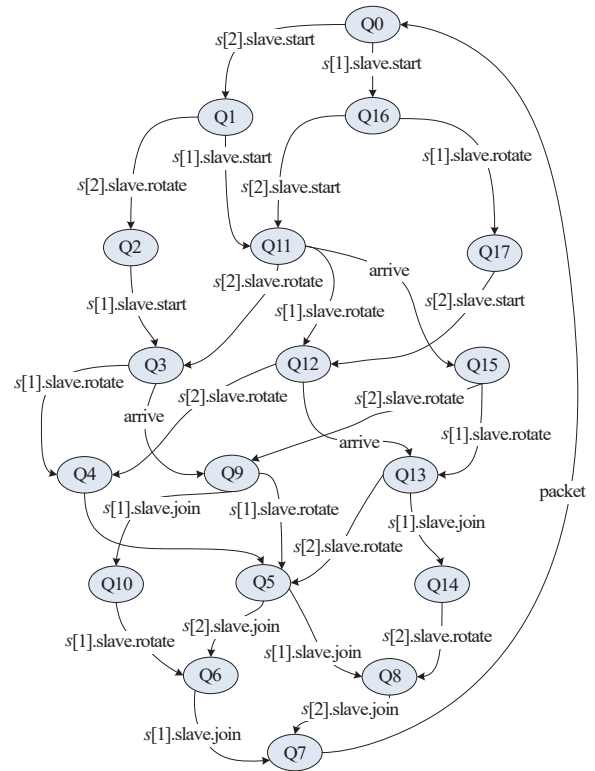


图4 两个正则表达式两个子进程组合的LTS语义图

在LTS中，迹tr的性质可以描述系统某种特性。迹是系统中接收信号序列在LTS中所形成的轨迹序列。这个序列由状态机中的状态构成。LTS系统是所能接收的转移轨迹的集合，即 $\forall tr \in M$ ，其中M是LTS系统所定义的抽象机器。

正则表达式的语义比较简单,各自对应的操作是LTS的结合。普通的字符就是从某种状态接收标签字符然后移动到另外一种状态。连接运算“.”就是通过首尾相连的方式将两个自动机串行组合。同样的,并行运算“||”是将两个自动机首状态和尾状态重合而成。闭包运算是在匹配完成之后通过一个 ϵ 标签从尾部状态跳转到开头状态构成。

经过这几种运算的反复组合就构成了正则表达式对应的复杂的LTS状态机,通过一系列的运算将正则语言和对应的LTS语义联系在一起。运行在安全互联网关上的多个进程加载正则语言匹配数据包内容的过程,可以

和单进程加载正则语言匹配数据包内容建立等价关系。这种等价关系确保采用并行结构计算得出的结果和串行计算得出的结果是一样的。下面分析主从进程的语义安全性。

5 语义安全性分析

通过语义分析,确保并行程序具备良好的性质。对于并程序,要避免使用临界资源导致的死锁问题、饥饿问题,以及程序运行等价性的问题。这样才能使得正则表达式并行载入匹配程序正确、稳定、安全、高效。

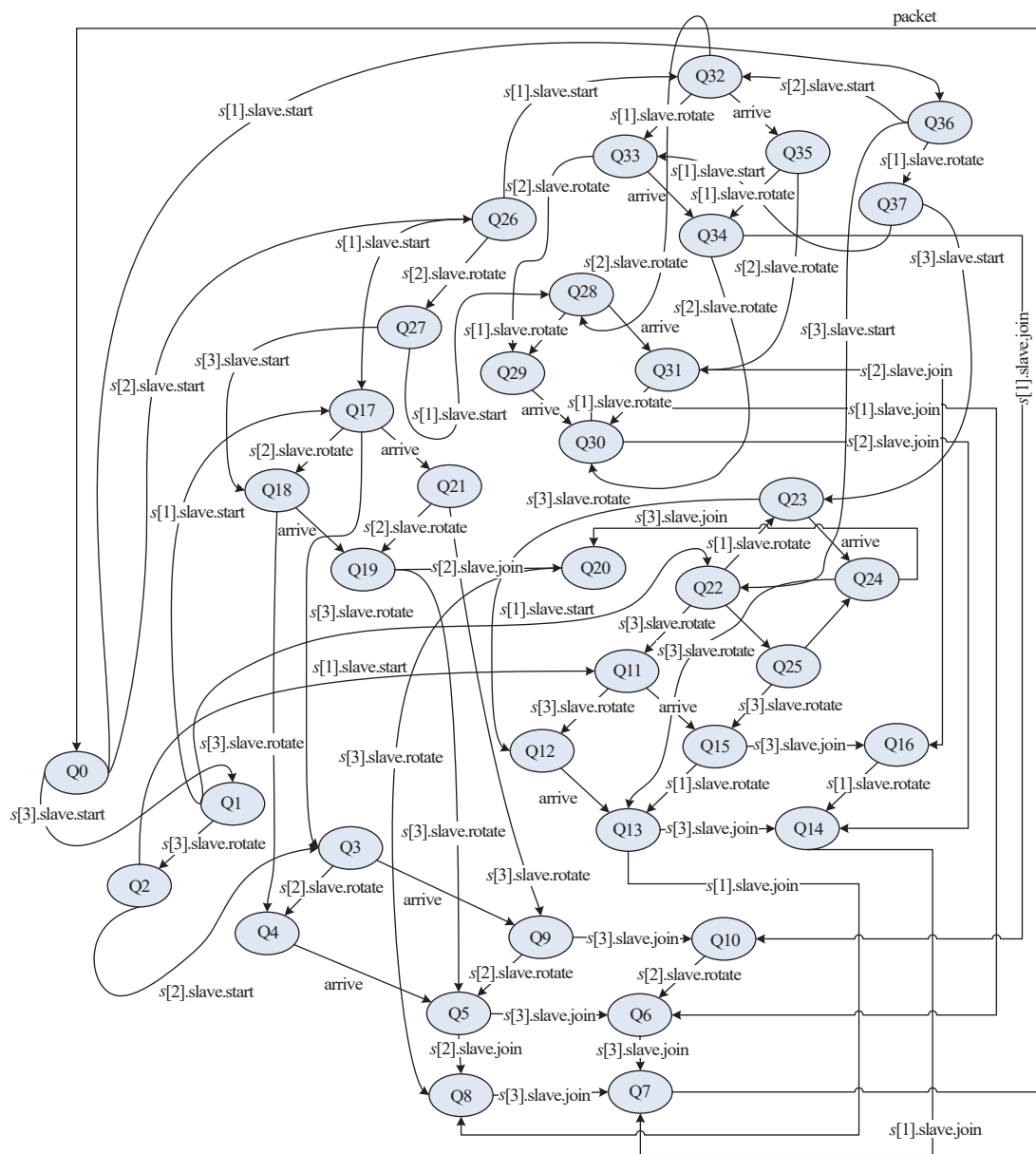


图5 两个正则表达式三个子进程组合的LTS语义图

5.1 无死锁状态性质

无死锁性质定义为

$$\forall tr \in (PaQ) \exists b \in \sum, R \in Q(QbR) \quad (14)$$

也就是说，每一个状态上，都存在信号使得状态可以跳转。通过直观地观察图2~图5，可以看到每一个状态都有信号可以跳转到其他状态，因此可以直观得出对于有限的这几种情形，系统是没有死锁的。这个问题可以推广到一般情形，它一共有两个参数RN、TN，即正则表达式个数、进程个数。在这个问题中，有以下几种符合实际情形的定义。

性质1

$$\forall RN, TN (RN \leq TN) \quad (15)$$

对于仅有一个进程和一个正则表达式的情形，对应的问题就是RN=1, TN=1情形，对于更复杂的情形，都是它的并行扩展。可以看到主从进程结合的方式是通过资源监视器联系在了一起。对于资源下标引用过界的担忧，需要通过约束来证明没有这种情形可能发生。

性质1从条件：

when($b > 0$) $s[i:RTN].slave.start \rightarrow REGEXPMATCH[b-1]$ 可以得出 $b-1 \in RRN$ 。且从条件：when($b > 0$) $s[i:RTN].slave.join \rightarrow REGEND[b-1]$ 中可以得出， $b-1 \in RRN$ 。

因为当跳转的条件是 $b=0$ 且 $b > 0$ 时，进行 $b=b-1$ 操作，因此 b 最小是1，由此得出 $1-1==0 \in RRN$ 。又因为当跳转条件为 $b=0$ 时，跳转到REGEND，且在REGEND中不同于前面的slave.start信号，状态机接收slave.join信号，因此 b 最多只减少到1，且没有增加的条件，最高为RN，所以它的范围是 $[0, RN] \in RRN$ 。与第二个命题同理。在REGEND中的 b 从RN降到0的时候，会跳转到REGEXPMATCH状态。在REGEXPMATCH中， b 为RN，这回到了状态机初始情形，确保了两个变量 b 都在0~RN范围之内。

性质2

从条件when($b > 0$) $s[i:RTN].slave.start \rightarrow REGEXPMATCH[b-1]$ 中可以得出： $\exists b(b-1==0)$ 。

性质2从条件：

when($b > 0$) $s[i:RTN].slave.join \rightarrow REGEND[b-1]$ 中可以得出 $\exists b(b-1==0)$ 。

根据性质1，以及MASTER_SLAVE=(MASTER $\parallel s[i:RTN]:SLAVE)$ ，可以得出可用的 $s[i]$ 数目大于等于RN，从而使得 $b-1$ 可以达到0。

性质3

$$\forall s[i].slave.start \exists s[i].slave.join$$

根据性质2，当有TN个进程接收slave.start信号以后， $b-1$ 就达到0，因此该状态只接收arrive信号从而状态机

到达了REGEND[RN]状态，然后开始只接收 $s[i].slave.join$ 信号，一直到RN个对应的信号接收操作执行完成。由于期间不接收slave.start信号，因此只能是前面已经启动的RN个进程运行到达slave.join状态，即正好是RN个对应的进程。并且等到RN个进程都接收了slave.join信号以后， $b-1$ 到达0。

性质4

$$\forall RN, TN (RN > 0 \wedge RN \leq TN) \rightarrow \text{机器} M \text{无死锁。}$$

根据性质3，所有进程 $[i]$ 在REGEXPMATCH[RN]启动之后，都必须要到REGEND[0]结束，因此每个SLAVE进程都完整地接收了 $[i:1..RN].slave.start/rotate/join$ 信号以及arrive信号。到达REGEND[0]状态后，接收packet信号到达REGEXPMATCH[RN]状态。反复循环，即所有状态都有转移信号。

5.2 无饥饿状态性质

相比死锁问题是最坏的事情一定不会发生，无饥饿问题是好的事情一定会发生。这里的无饥饿问题可以理解为，在无限次的运行过程中，进程的每一个状态都会运行到。这里面首先引入进度属性。进度属性是当进程定义的某个初始动作发生时，末尾动作一定发生。根据该进程定义，它的中间状态也一定会发生。

性质5

$$\forall [i:1..RN]. [i:1..RN].slave.start \rightarrow [i:1..RN].slave.join, \text{并且} REGEXPMATCH[RN] \rightarrow REGEND[0].$$

根据性质3可知，子进程SLAVE满足进度属性。又根据性质2可以知道， $b-1$ 一定可以达到0，因此可以实现REGEXPMATCH[RN]到REGEND[0]的互相跳转。综上所述，两个进程都满足进度属性。

性质6

无限次运行机器M，每个信号都会得到执行。

因为REGEXPMATCH和REGEND的局部变量 i 的循环范围都是 $[0, RTN]$ ，且 $RN > 0$ ，即至少有一个进程，因此每个SLAVE进程都有机会得到执行。又因为主从进程都有进度属性，因此每个进程上的信号都会得到执行，即机器M上的信号都会得到执行。

5.3 并行运行和串行等价

并行运行和串行等价性质用来回答单进程执行和多进程执行的结果是一致的，并行运行并不会导致结果计算的不正确。对于单进程依次载入N个正则表达式匹配，是模型中RN=TN=1的情形下循环运行n次。每次接收join信号后得出匹配结果，如果匹配成功，将 $re=i$ 保存下来。

性质7

并行运行和串行结果相等。这个过程和顺序无关。

根据多域网络互联安全控制中的定义, $\forall \text{reg}_1, \text{reg}_2 \in \pi_{\text{REG}}(R_{\text{mdr}}) (\mathcal{L}(\text{reg}_1) \cap \mathcal{L}(\text{reg}_2) = \emptyset)$, 正则表达式之间不存在交集, 因此结果只能是至多有一个正则表达式匹配成功。当顺序运行每个表达式的时候, 如果有且仅有一个正则表达式*i*匹配成功, $\text{re}=i$ 。对于并行执行时, *i*进程第*j*次匹配时, $\text{re}=i$, 跟次数*j*无关。因此两者相等。当不存在表达式匹配时, $\text{re}=0$, 所以结果一样。因此并行运行和串行运行结果等价。

按照形式化语义方法设计的并行程序可以确保主从进程良好的运行状态。如何使程序在实际环境中高效运行, 还需要考虑到额外的其他情况, 并对其进行设计、分析和验证。

6 实验验证

高性能主从进程并行载入正则表达式匹配数据包实验采用了8核i7处理器测试其运行效果。实验的对比方案有3种, 第一种是单核顺序执行没有任何多线程因素的常规匹配, 第二种是采用Linux内核自身调度机制的多进程匹配实验, 第三种是高性能方案, 采用CPU绑定的多进程匹配实验。操作系统是在64位CentOS 7上运行的。其中多进程运行需要信号量互斥以及进程间内存共享来传递数据。采用主进程调用子进程的方式来完成任务, 并最后在主进程汇总任务运行结果。互联安全控制高性能运行硬件环境见表1。

表1 互联安全控制高性能运行硬件环境

处理器	内存	操作系统	备注说明
Intel(R) Core i7-3770 3.40 GHz	8 GB	Windows 10	宿主主机
Intel(R) Core i7-3770 3.40 GHz	4 GB	CentOS7	虚拟主机

测试场景从CPU核数和正则表达式匹配规模两个维度展开。为了统一对比标准, 设定完成的任务次数为6 720次, 当*n*核的时候, 汇总的次数就是6 720/*n*次。正则表达式长度和数据包URL特征长度依次增加, 互联安全控制高

性能运行软件环境见表2。

通过这两个维度, 可以对CPU并行运行有一个较为直观的理解。通过图6可以看出, 在正则表达式非常简单的情形下, 常规的顺序执行所消耗的时间几乎为0。与之相对比, 多进程并行执行随着核数*n*的增加反而在变慢。通过分析, 造成这种现象的原因是信号量的加锁解锁操作带来的系统开销要明显大于正则表达式匹配所消耗的时间。采用操作系统提供的调度机制实现多线程并行运行的实验, 具体数据为: 1核450 000 ms, 2核910 000 ms, 3核1 420 000 ms, 4核1 890 000 ms, 5核4 320 000 ms, 6核5 180 000 ms, 7核6 050 000 ms, 8核6 960 000 ms。这种方案比进程绑定CPU的方式要大一个数量级, 所以不在图6中绘制。在较为复杂的正则表达式匹配过程中, 常规单进程运行的正则表达式匹配难度迅速增加。在regexp:63-text:6情形中, 耗时4 370 000 ms; 在regexp:63-text:50情形中, 耗时140 110 000 ms, 远远高于进程绑定CPU情形。与之相比, 进程绑定CPU的任务执行时间消耗只是微弱增加。

正则表达式和文本说明见表3。

从横向对比来看, 随着进程绑定CPU的数目增加, 1~4核和5~8核都是消耗时间在整体降低。在4核的时候, 达到最优位置; 然后从5核起时间消耗就猛然增加到2~3核消耗的时间区间中间; 最后随着核数增加时间消耗有所降低。从纵向对比来看, 随着正则表达式长度和URL文本长度增加, 4核领先8核的优势在降低, 在regexp:63-text:50情形时, 8核绑定进程载入正则表达式匹配URL文本的时间消耗要低于4核。

上述实验表明, 在正则表达式比较简单的情况下, 常规顺序执行的效率极其高效。采用操作系统调度策略的多线程运行会使得CPU忙于频繁的上下文切换, 从而大大降低匹配效率。采用进程绑定CPU的方式可以降低进程切换过程, 但是进程之间同步的开销仍然不可忽视, 只有在正则表达式比较复杂的情形下, 它的优势才

表2 互联安全控制高性能运行软件环境

软件	版本	测试单元
虚拟化支持	VMware Workstation 16 Pro	主从进程载入正则表达式匹配
C 标准库支持	GNU C Library -GLIBC 2.11	正则表达式匹配引擎
正则表达式匹配库	Perl Compatible Regular Expressions PCRE 8.32-17	正则表达式匹配引擎
正则表达式匹配库	Oniguruma 6.9.6	正则表达式匹配引擎
正则表达式匹配库	SubReg 1.0	正则表达式匹配引擎
正则表达式匹配库	Approximate Regex Matching Engine TRE 0.8.0	正则表达式匹配引擎
正则表达式匹配库	Hyperscan 5.4.0	正则表达式匹配引擎
正则表达式匹配库	International Components for Unicode ICU 66.1	正则表达式匹配引擎
编译器	GNU Compiler Collection 4.8.5 20150623	正则表达式匹配引擎模块编译
跨平台编译器	Cross Platform Make 2.8.12.2	正则表达式匹配引擎模块编译
C++库	Boost 1.63.0	正则表达式匹配引擎模块编译
有限状态机编译器	Ragel State Machine Compiler 6.9	正则表达式匹配引擎模块编译

表3 正则表达式和文本说明

代号	正则表达式	文本
regex: 3-text:18	ab+	asdfsdfdsfasfdasff
regex: 63-text:1	[alw]*[bmX]*[cny]*[doz]*[ep]*[fq]*[gr] *[hs]*[it]*[ju]*[kv]*[a]+	v
regex: 63-text:6	[alw]*[bmX]*[cny]*[doz]*[ep]*[fq]*[gr] *[hs]*[it]*[ju]*[kv]*[a]+	vkvvvv
regex: 63-text:50	[alw]*[bmX]*[cny]*[doz]*[ep]*[fq]*[gr] *[hs]*[it]*[ju]*[kv]*[a]+	vkvvvvvvvvvvvvvvvvvvvv vvvvvvvvvvvvvvvvvvvvvv vvvvvvvvv

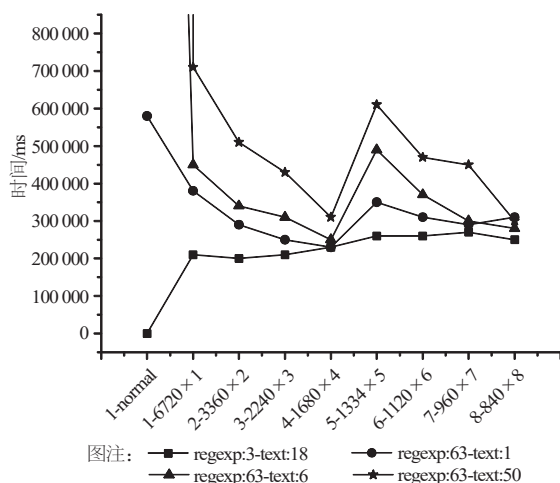


图6 高性能正则表达式匹配结果对比

会得以体现。在正则表达式长度没有复杂到更高的情形时，4核CPU绑定已经是最优的方案，和8核进程运行的效率差不多。只有在正则表达式长度更长、结构更加复杂的时候，才有采用8核CPU绑定进程方案的必要。这说明多个进程结果汇总协调也是重要的系统开销。

高性能优化方案采用在Windows主机上搭建宿主主机的方式运行程序。这样做的原因是，采用虚拟机的方式可以深度模拟CPU类型，从而构建符合编译正则表达式匹配程序的编译环境。

正则表达式匹配引擎有多种类型。匹配的方式大体可以分为确定型有限状态自动机和非确定型有限状态自动机两个大类。确定型有限状态自动机类型的正则表达式匹配引擎构造简单稳定，没有回溯的过程，但是在使用的过程当中，表述过程复杂。对于多次匹配相同字符串类型的问题，语法上不容易表达书写。非确定型有限状态自动机类型的正则表达式匹配引擎构造较为复杂，其核心思想是贪婪匹配回溯算法。这个算法是一个不稳定算法，存在匹配非常慢的情形。非确定型有限状态自动机类型的正则表达式匹配引擎可以支持更加灵活多样的正则表达式语法，使用者可以灵活地表达各种字符匹配情形。

GNU C Library (GLIBC) 是标准的C语言算法库，在CentOS等Linux操作系统上都有直接支持。Perl Compatible Regular Expressions (PCRE) 是开源的正则表达式匹配库，它支持的正则表达式语法灵活多样，使用较为广泛，Snort软件编译运行也依赖于这个库。Oniguruma是支持多种字符编码的正则表达式匹配库。International Components for Unicode (ICU) 是支持Unicode编码的正则表达式匹配库，分为Java语言版本和C语言版本。为了程序实现高效，这里采用了C语言版本ICUC66。SubReg是轻量级脚本扫描正则表达式匹配库，通常用于嵌入式设备等资源受限的通信设备中，SubReg支持的语法规则有限。Approximate Regex Matching Engine (TRE) 是一个轻量级的正则表达式匹配库，它采用了近似匹配的优化方法。TRE采用的正则匹配算法空间复杂度较高，但是和匹配字符长度无关。Hyperscan是基于英特尔运算处理器的正则表达式匹配库，在算法设计中，采用确定型有限状态自动机和非确定型有限状态自动机混合的方式构造。

图7是测试采用了不同正则表达式匹配引擎的URL特征处理过程所消耗时间的对比结果。通过图7可以看出，Hyperscan的效率最高，它基于自动机理论，采用有限状态自动机编译器ragel编码实现。另外Hyperscan提供了C语言调用接口，因此可以兼容多种类型的开发语言实现联合编译。在高性能的实现方面，可以优先考虑这种方案。除此之外，TRE性能也很高，但是它空间复杂度较高，在需要反复加载正则表达式的应用情形中，有可能运行性能不稳定。Oniguruma和ICU是非常优秀的设计方案，它们支持多种语言编码，尤其是国际编码Unicode以及UTF-8，可以支持汉字。如果匹配文本涉及多国语言，这两种是非常好

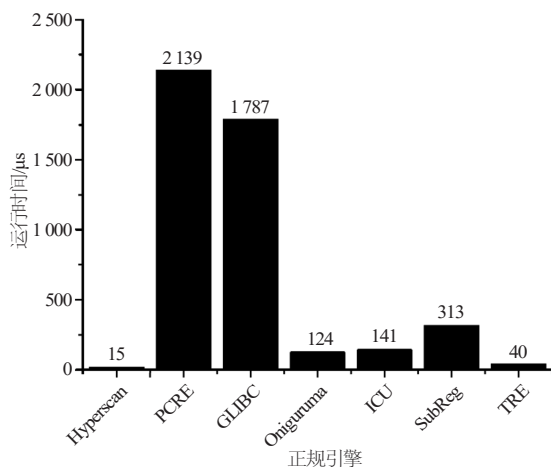


图7 采用了不同正则表达式匹配引擎的URL特征处理过程所消耗时间的对比

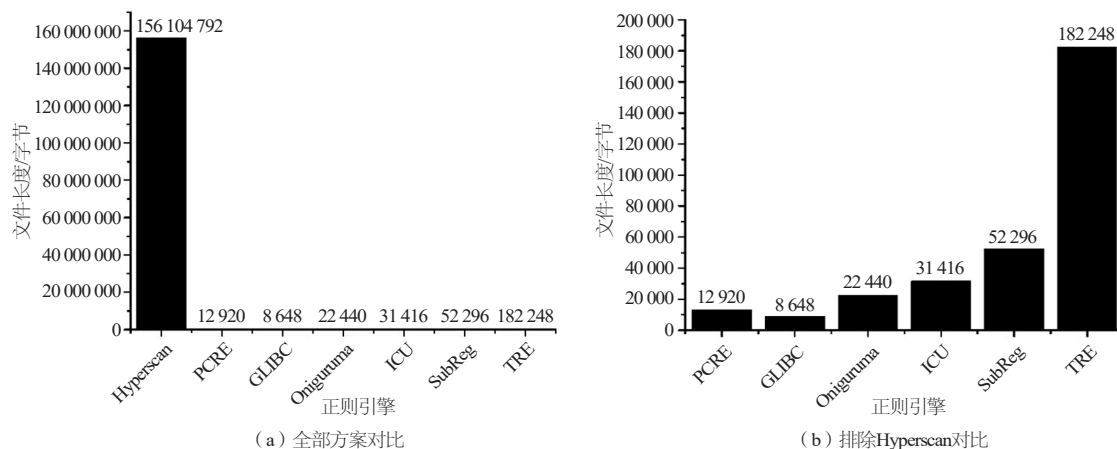


图8 正则表达式匹配空间复杂度

的参考方案,尤其是Oniguruma支持多种编码方式,它的兼容性更强。SubReg效率也还不错,但是考虑到它的语法较为简陋,只能处理一些简单的正则表达式匹配情形。使用广泛的PCRE和GLIBC库提供灵活标准的正则表达式匹配库,在不需要频繁运行程序的场合,使用这两种库能够快速开发,不用安装额外的一系列依赖开发包。

正则表达式匹配空间复杂度如图8所示。

图8(a)、图8(b)展示了程序占用内存、磁盘空间的对比。从图8(a)中可以看出,Hyperscan生成的程序非常巨大,有156 104 792字节。与之相对比,其他方案只有十几kB到百十来kB大小。如果是一些对内存占用有一定约束的场合,Hyperscan方案并不适合。图8(b)展示了排除Hyperscan方案之后的程序占用空间对比。可以看出,GLIBC生成的体积最小,TRE是除Hyperscan外最大的,但是也仅有182 248字节。还有一点需要考虑的是,SubReg是依赖ANSI C标准编写的,它没有任何C语言库依赖项。在嵌入式设备当中,可以用它来实现简易的正则表达式匹配引擎,完成一些简单的文本匹配任务。

7 结束语

本文利用形式化语义的方法设计了应用于网间互联控制的高性能并行优化方案。通过语义分析确保该设计具有良好的语义安全性,避免了死锁、饥饿、结果不一致的情况发生。通过进程绑定CPU的方式优化了进程调度效率,并通过实验对比分析得出优化方案适合的应用情形,为后续的进一步高性能优化提供了改进方向。

参考文献:

- [1] LU B, CAO R H, LU Y M, et al. Design and formal analysis of an authentication protocol, eWMDP on wearable devices[J]. IEEE Access, 2019, 7: 97771-97783.
- [2] ZHANG L H, LU B D, ZHAO T, et al. A novel decentralized blockchain networks model with high concurrency (blockchain networks model with high concurrency)[C]//Proceedings of 2019 IEEE 14th International Conference on Intelligent Systems and Knowledge Engineering (ISKE). Piscataway: IEEE Press, 2019: 800-804.
- [3] GORRIERI R, VERSARI C. Introduction to concurrency theory[M]. Cham: Springer International Publishing, 2015.
- [4] POLYVYANY A, SOLT A, WEIDLICH M, et al. Monotone precision and recall measures for comparing executions and specifications of dynamic systems[J]. ACM Transactions on Software Engineering and Methodology, 2020, 29(3): 1-41.
- [5] DIERTENS B. A process algebra software engineering environment[EB].
- [6] ZHANG F, ZHAO Y W, MA D F, et al. Formal verification of behavioral AADL models by stateful timed CSP[J]. IEEE Access, 2017, 5: 27421-27438.
- [7] MKAOUAR H, ZALILA B, HUGUES J, et al. A formal approach to AADL model-based software engineering[J]. International Journal on Software Tools for Technology Transfer, 2020, 22(2): 219-247.
- [8] KIM R, KWON H T, CHI S Y, et al. A coordination model for agent behaviors using hierarchical finite state machine with inter-level concurrency[C]//Proceedings of 2016 International Conference on Information and Communication Technology Convergence (ICTC). Piscataway: IEEE Press, 2016: 359-364.
- [9] PRIETO M, SUGZKY L, GONZÁLEZ N, et al. Evaluation of the uModel factory software used for the modeling of embedded

- systems with concurrent states[C]//Proceedings of 2017 IEEE 8th Latin American Symposium on Circuits & Systems (LASCAS). Piscataway: IEEE Press, 2017: 1-4.
- [10] PRZIGODA N, HILKEN C, WILLE R, et al. Checking concurrent behavior in UML/OCL models[C]//Proceedings of 2015 ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems (MODELS). Piscataway: IEEE Press, 2015: 176-185.
- [11] SHIROLE M, KUMAR R. Test scenario selection for concurrency testing from UML models[C]//Proceedings of 2015 Eighth International Conference on Contemporary Computing (IC3). Piscataway: IEEE Press, 2015: 531-536. .
- [12] GADEA C, IONESCU B, IONESCU D. Modeling and simulation of an operational transformation algorithm using finite state machines[C]//Proceedings of 2018 IEEE 12th International Symposium on Applied Computational Intelligence and Informatics (SACI). Piscataway: IEEE Press, 2018: 119-124.
- [13] PAZZI L, PRADELLI M. Modularity and part-whole compositionality for computing the state semantics of statecharts[C]//Proceedings of 2012 12th International Conference on Application of Concurrency to System Design. Piscataway: IEEE Press, 2012: 193-203.
- [14] RAVICHANDRAN K, PANDE S. F2C2-STM: flux-based feedback-driven concurrency control for STMs[C]//Proceedings of 2014 IEEE 28th International Parallel and Distributed Processing Symposium. Piscataway: IEEE Press, 2014: 927-938.
- [15] MEJIA-RICART L, OLMSTED A. Avoiding unnecessary deaths: drag-Back, a deadlock avoidance model[C]//Proceedings of 2017 12th International Conference for Internet Technology and Secured Transactions (ICITST). Piscataway: IEEE Press, 2017: 472-474.
- [16] BAKURA S A, MOHAMMED A. Lock-free hybrid concurrency control strategy for mobile environment[C]//Proceedings of 2014 IEEE 6th International Conference on Adaptive Science & Technology (ICAST). Piscataway: IEEE Press, 2014: 1-5.
- [17] RATHNAYAKE A, THIELECKE H. Regular expression matching and operational semantics[J]. Electronic Proceedings in Theoretical Computer Science, 2011, 62: 31-45.
- [18] ZENG M, LIU P, MIAO H K. The design and implementation of a modeling tool for regular expressions[C]//Proceedings of 2014 IIAI 3rd International Conference on Advanced Applied Informatics. Piscataway: IEEE Press, 2014: 726-731.
- [19] POURALI P. A user-centric approach to improve the quality of UML-like modeling tools and reduce the efforts of modeling[D]. Waterloo: University of Waterloo, 2020.

[作者简介]



卢波（1988-），男，北京邮电大学博士生，主要研究方向为协议安全分析、安全关键系统设计。



陆月明（1969-），男，北京邮电大学教授、博士生导师，可信分布式计算与服务教育部重点实验室学术委员会委员与学术带头人，主要研究方向为互联网安全等。