基于多槽分桶的快速规则冲突检测算法

罗 谦[1,2]，唐常杰[1]，郑皎凌[3]，胡 建[4]

(1. 四川大学计算机学院 成都 610065; 2. 中国民用航空总局第二研究所信息公司 成都 610041;
3. 成都信息工程学院软件工程系 成都 610225; 4. 中国西南电子设备研究所 成都 610036)

【摘要】为解决企业海量规则集合中产生的规则自我冲突问题，提出了基于多槽分桶的快速规则冲突检测算法MSSB。该算法利用同槽实桶之间规则两两必不冲突特性，将复杂的冲突规则求解转换为线性时间内的不冲突规则求解，从而在稳定的空间和时间复杂度下有效解决规则冲突发现问题。先形式化描述了通用规则冲突和不冲突，并在合理的假设条件下证明了3个规则间关系的命题和同槽不冲突定理；然后提出了基于哈夫曼树和三角矩阵结构的MSSB算法；最终在国内民航典型机场的规则集合上完成了对比实验，结果表明新算法的冲突检测性能比Policytree算法相提高了36.2%。

关 键 词 冲突检测; 多槽; 规则引擎; 分桶
中图分类号 TP301 文献标识码 A doi:10.3969/j.issn.1001-0548.2012.03.024

Fast Algorithm to Detect Conflict Rule Based on the Multi_Slot Sub_Bucket

LUO Qian[1,2], TANG Chang-jie[1], ZHENG Jiao-ling[3], and HU Jian[4]

(1. Department of Computer Science， Sichuan University Chengdu 610065;
2. Information Filiale, The Second Research Institute of CAAC Chengdu 610041;
3. Department of Software Engineering, Chengdu University of Infomation Technology Chengdu 610225;
4. Southwest China Research Institute of Electronic Equipment Chengdu 610036)

Abstract To resolve the conflict within the massive rules in enterprise, this paper proposes a fast rule conflict-detection algorithm named multi_slot sub_bucket conflict cetection (MSSB) based on multi_slot and sub_bucket. It turns rule's complexity conflict detection into result of non-conflict rules in lineartime by the theorem of non-conflict. First, this research proposed the concepts of general rule's conflict and non-conflict, and proves three propositions and the theorem of non-conflict. Then it proposes the MSSB algorithm by the structure of Huffman tree and Triangular matrix. Extensive experiments over real data of Hub Airport show the effectiveness of new proposed MSSB algorithm. The average space complexity is decreased 33.6% and matching time is decreased 36.2% compared with traditional linear detection and policytree.

Key words conflict detection; multi_slot; rule engines; sub_bucket

面向行业的软件系统设计越来越依赖于业务流程规范，同时多变的用户需求又要求系统能通过简单组合配置就能立即生效。该现状催生了规则引擎(rules engine)技术的高速发展。通过规则引擎将业务规范和用户商业需求转换为若干条规则记录加以管理，这样规则是否有效、是否匹配就能直接影响系统的操作行为，实现行业软件系统的高度灵活性。

文献[1-3]在防火墙过滤规则应用领域提出名为“PolicyTree”的树状结构，该文献借鉴Trie树的基本结构将过滤规则分解为记录属性名和记录属性值，并据此提出了单防火墙和分布式防火墙规则冲突算法。文献[4]基于动作数定理并结合上述研究成果提出了简化操作符的MSHtree算法；随之扩展操作类型提出了MSHTrie算法，使规则引擎有了普适性。规则匹配效率较PolicyTree算法有21.3%性能提升；文献[5]在规则匹配上选用Trie树结构，并利用二分查找实现了BLST算法，但规则冲突检测上却用一个简单的三重循环实现，显然无法适应海量规则以及多操作结果的普适场景；因此，文献[6-9]分别提出PATRICIA树和LE-Trie树结构改进该问题，但提出的

收稿日期：2011－07－15； 修回日期：2012－03－20
基金项目：国家自然科学基金(60773169)；中国民用航空局科研项目(MHRD200924)
作者简介：罗谦(1975－)，男，博士生，高级工程师，主要从事数据挖掘、进化计算、企业智能计算等方面的研究.

算法仅解决了规则中的源IP域和目的IP域的冲突检测，无法满足规则库的普适性需求；文献[10]率先在防火墙包过滤场景中证明了规则表达式交集下的冲突规则检测是一个SAT问题的变形，为NPC问题；文献[11]考虑通过压缩上述的算法的搜索路径来改进效率。

本文通过分析民航机场业务规则特点入手，深入研究了业务规则之间的三大关系，给出了业务规则之间冲突与不冲突的严格形式化描述，基于槽、实桶和虚桶等定义提出了同槽不冲突定理，并产生了用于规则快速冲突检测的MMSB算法。该算法在下三角矩阵和哈夫曼树(Huffman Tree)的支持下避免了任意规则对的所有规则点线性匹配，且对规则条数的增加不敏感。

1 基本概念和术语

1.1 民航机场规则库

行业通用的规则记录形式为“IF(规则条件表达式)THEN(业务操作结果)”，表1是一个民航机场资源分配规则库的简化说明。

表1 民航机场资源分配规则表

	航空公司	属性	经停站	VIP	制定者	结果
Rule1		国际	旧金山		A	机位1
Rule2	南航			是	A	机位2
Rule3		混合	北京	否	A	机位3
Rule4	上航	国内			A	机位4
Rule5	川航	国内	成都	是	A	机位5
Rule6	国航	国际			A	机位6
…	…	…	…	…	…	…
Rule119		国际			B	机位105
Rule120		混合			B	机位229

其中Rule1表示为：

IF(航班属性=国际and经停站=旧金山)THEN(占用机位1)，该规则记录的制定人员是“A”。上述规则的制定人员项是虚拟的，只用于说明将要讨论的问题。

在企业制定海量规则记录的过程中受不同操作人员和频繁变化的商务合约影响，容易出现表1的记录情况：制定人员A和制定人员B分别在不同时间段录入各自理解的规则记录。

观察1：如果一条业务数据X：四川航空公司从成都出发经停旧金山的3U488出港国际航班，经过表1的规则匹配操作，发现会同时命中Rule1(机位1)和Rule119(机位105)，显然操作结果是不一样的。如果算法采用线性匹配方式，则Rule119永远都不会命中。这与制定人员B制定Rule119的诉求不符。

从上述观察可以看出，企业海量规则表中可能存在观察1所示的规则间的潜在冲突(hidden conflict)，即多个规则的同时命中却只有一个规则生效。

1.2 相关概念和定义

行业的常见应用场景中，业务对象下的属性名对应一个变量名，业务对象的实例值对应变量名的值。形式化后为：

定义 1 规则R

1) 形如$R=\{(P_1\wedge P_2\wedge \cdots P_n)\text{->}A|P_i\in\{P\}$，$A\in\{\text{Action Set}\}\}$，称$R$为一条规则。

2) 其中，A是来自Action集合，表示$(P_1\wedge P_2\wedge\cdots P_n)$结果为True时所需执行的业务操作；

3) 规则点P_i，定义为满足如下的一个表达式：$P_i=\{(P_i.N=P_i.V)|\ P_i\in(\text{True，False})\}$，其中$P_i.N$是一个变量名，$P_i.V$为$P_i.N$对应值域的一个具体值，$n$表示规则点个数；表1中第二条规则Rule2可描述为：$R_2((P_1.N=\text{“航空公司”}，P_1.V=\text{“南航”})\wedge\cdots\text{->}(A_2=\text{“机位2”})$

为适应本文的应用场景，提出下列假设条件：

1) 规则点的不交叉性

设$P_i.N$和$P_j.N$表示任意一条规则R的第i个和第j个规则点属性名，则$\{P_i.\text{N}\}\cap\{P_j.N\}=\Phi$，其中$i\neq j$，$1\leqslant i，j\leqslant n$。

2) 规则动作结果的单一性

设r为规则库中的任意一条规则，则要求规则r的$|A|=1$，表示每一条规则有且仅有一个动作单值。

3) 规则点属性值的空值转换

设$P_i.V$表示任意一条规则R的第i个规则点属性值，当$P_i.V$不出现在规则表中时，则认为$P_i.V=\Omega$，Ω表示$P_i.V$的值域全集。

如表1中Rule1的$P_1.N=$“航空公司”，$P_1.V$对应内容为空，则根据假设3此时的$P_1.V=\{$“南航”，“上航”，“川航”，“国航”$\}$。这一假设是为冲突检测算法提出的。

透过观察1可以发现任意两条规则的对应规则点属性值所构成的集合会出现仿集合关系的包含、相交和相离关系。

定义 2(规则之间的包含关系) 设R_i和R_j为规则表中的任意两条规则，对于任意整数$k(1\leqslant k\leqslant m$，$m$为规则总数)，如果均出现$P_{[i,k]}.V=P_{[j,k]}.V$或者$P_{[i,k]}.V=\Omega$，则称规则$R_i$包含规则$R_j$。

表2　规则间包含关系

	航空公司	属性	经停站	VIP	制定者	结果
R_1		国际	旧金山		A	机位1
R_{119}		国际			B	机位105

定义 3(规则之间的相交关系)　设R_i和R_j为规则表中的任意两条规则，如果至少存在一个整数k($1\leqslant k\leqslant m$，m为规则总数)，使得$P_{[i,k]}.V=P_{[j,k]}.V$成立，同时$P_{[i,q]}.V=\Omega$ 或者$P_{[j,q]}.V=\Omega(k\neq q)$成立，则称规则$R_i$与规则$R_j$相交。

表3　规则间相交关系

	航空公司	属性	经停站	VIP	制定者	结果
R_1		国际	旧金山		A	机位1
R_6	国航	国际			A	机位6

定义 4(规则之间的相离关系)　设R_i和R_j为规则表中的任意两条规则，对于任意一个整数k($1\leqslant k\leqslant m$，m为规则总数)，如果均出现$P_{[i,k]}.V$或$P_{[j,k]}.V$不等于Ω时，必有对应的$P_{[i,k]}.V=\Omega$或$P_{[j,k]}.V=\Omega$成立，则称规则R_i与规则R_j相离。

表4　规则间相离关系

	航空公司	属性	经停站	VIP	制定者	结果
R_1		国际	旧金山		A	机位1
R_2	南航			是	A	机位2

定义 5(规则冲突)　设R_i和R_j为规则表中的任意两条规则，如果两规则的每一对规则点属性值的交集都不等于空集，且操作结果不同，则称这两条规则是冲突的，记conf(R_i, R_j)。形如：

IF $\{\forall k: P[i,k].V\cap P[j,k].V\neq\Phi|1\leqslant k\leqslant m\}$ and $(A_i\neq A_j)$

THEN conf(R_i, R_j)

表1中(Rule1，Rule119)、(Rule1，Rule6)和(Rule3，Rule120)均为冲突规则对。

下面将证明规则的3条重要命题，即规则的包含、相交和相离都必然导致冲突。

命题 1(包含必冲突性质)　设R_i和R_j为任意两条规则包含的规则对(R_i包含R_j)，当$A_i\neq A_j$时，则必有conf(R_i, R_j)成立。

证明：根据R_i包含R_j的定义，每一对规则点属性值要么相等，要么二者必有一个是Ω，故对应的交集必不等于空集，且操作结果不同，完全符合规则冲突的定义，故conf(R_i, R_j)。

命题 2(相交必冲突性质)　设R_i和R_j为任意两条规则相交的规则对，当$A_i\neq A_j$时，则必有conf(Ri, Rj)成立。

证明：根据R_i和R_j规则相交的定义，每一对规则点属性值可以相等，可以是属性值与全集Ω对应，故对应的交集必不等于空集，且操作结果不同，完全符合规则冲突的定义，故conf(R_i, R_j)。

命题 3(相离必冲突性质)　设R_i和R_j为任意两条规则相离的规则对，当$A_i\neq A_j$时，则必有conf(R_i, R_j)成立。

证明：根据R_i和R_j规则相离的定义，每一对规则点属性值都是值与全集的组合关系，故对应的交集必不等于空集，且操作结果不同，完全符合规则冲突的定义，故conf(R_i, R_j)。

显然，(Rule1, Rule119)、(Rule1, Rule6)和(Rule1, Rule2)分别说明了上述的3条性质。

通过定义5和3个命题，给出求解规则冲突的思路：任意给定的一个规则对，均需要遍历每一对规则点属性值之间的对应关系，当属于上述三者关系之一，则必属于冲突规则对。文献[5]就是用一个三重循环完成冲突的线性检测。当面临海量的规则集合时，时间复杂度无法满足实际需求。

本文通过对定义5的逆向思考，即转换寻求任意给定规模规则表的冲突对集合为寻找其对应的不冲突规则对。

定义 6(规则不冲突)　设R_i和R_j为规则表中的任意两条规则，只要存在一对规则点是互斥的，则称这两条规则是不冲突的，记compatible(R_i, R_j)。形如：

IF $\{\exists k: P_{[i,k]}.V\cap P_{[j,k]}.V=\Phi|1\leqslant k\leqslant m\}$

THEN (compatible(R_i, R_j))

表1中(Rule1, Rule4)、(Rule4, Rule5)和(Rule5, Rule16)均为不冲突规则对。为进一步说明如何求得规则库的不冲突对，需提出槽和桶的定义。

定义 7(槽S)　设$P_{[*,j]}$是所有规则R的第j个规则点，$P_{[*,j]}.N$是规则点上的变量名，则称$P_{[*,j]}.N$对应的$P_{[*,j]}.V$的值集合为槽S_j，形如：

$$S_j=\{P_{[*,j]}.V|\ 1\leqslant j\leqslant n\}$$

其中，n表示规则点总数。

如表1中$P_{[1,1]}.N$=“航空公司”，$P_{[1,1]}.V$的值集合形成的槽S_1=(南航，川航，上航，国航，…)

定义 8(桶Bucket)

1) 在槽S_k中根据每个$P_{[*,k]}$. V值对所有规则R进行分类，所形成的每一个规则分类集合分别对应一个实桶(Actuality Bucket)Act-B_s。形如:

Act-B_s=$\{\{R_k\}|\forall k: P_{[i,k]}.V=P_{[j,k]}.V,\ 1\leqslant k\leqslant m\}$

2) 在槽S_k中所有的规则集合R与所有实桶Act-B_s并集的差，形成一个槽S_k的虚拟桶(Virtuality Bucket)Vir-B_k。形如：

Vir-B_k=R-{Act-B_1 ∪ Act-B_2 ∪ … ∪ Act-B_s}

图1为表1规则集合按照定义7和定义8形成的图示。

引理 1(同槽规则单一分布) 设R为任意一个规则集合，r_i是R中任意一条规则($1\leqslant i\leqslant m$)，m为R中的规则数量。当规则点$P_{[i,j]}.V\neq\Phi$时，必存在规则r_i在槽S_j($1\leqslant j\leqslant q$，q为槽数)中有且仅出现一次。

证明：反证法。假设r_i在槽S_j出现过2次以上，则必然至少存在两个$P_{[i,k]}.V\neq P_{[i,o]}.V$，其中$k\neq o$；而根据$S_j$的定义可知$P_{[i,k]}.N=P_{[i,o]}.N$，从而推导出存在$\{P_{[i,k]}.N\}\cap\{P_{[i,o]}.N\}\neq\Omega$，其中$k\neq o$，$1\leqslant k$，$o\leqslant q$，和假设1冲突，故原命题成立。证毕。

定理 1(同槽不冲突定理) 设S_k是规则表R产生的一个槽，Act-B_i和Act-B_j为槽S_k的任意两个实桶，则Act-B_i和Act-B_j交叉组合形成的规则对必然是不冲突规则对。形式化表达为：

IF{<R_l,R_m>|$R_l\in$Act-$B_i\cap R_m\in$Act-$B_j\cap$Act-$B_i$in$S_k\cap$Act-B_j in S_k} THEN compatible(R_l, R_m)

证明：由引理1可知，规则R_l和R_m分别在槽S_k中有且仅出现一次；同时，规则R_l属于实桶Act-B_i，规则R_m属于实桶Act-B_j；由实桶的定义可推出$P_{[l,k]}.V\neq P_{[m,k]}.V$；故根据规则不冲突定义可知必有compatible(R_l, R_m)成立。证毕。

因此，基于单槽内的实桶交叉组合即可求得每个槽内的不冲突规则对集合，然后遍历所有槽产生的不冲突规则对求其并集，即可求得该规则库的所有不冲突规则对集合，最后利用任意规则对的组合全集与不冲突规则对集合求差，即可求得该规则库的冲突规则对集合。按此思想设计的MSSB算法通过避免任意两个规则对的规则点逐一匹配而获得了较高的冲突检测效率。

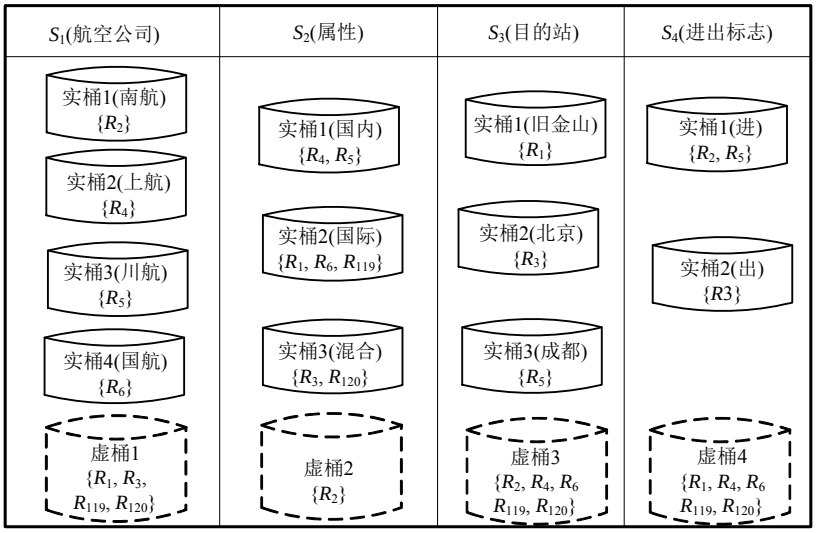

图 1 表 1 对应的槽和桶示意图

2 规则冲突检测算法(MSSB)

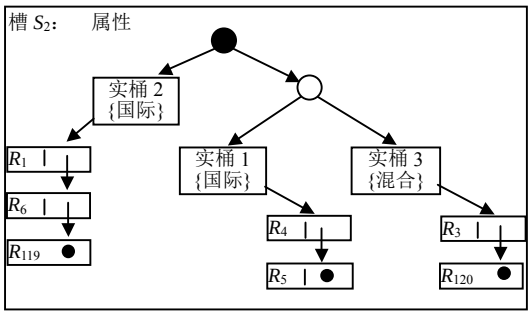

图 2 槽 S2 对应的哈夫曼树示意图

MSSB(multi_slot sub_bucket conflict detection)算法由3个层次的数据结构构成：1) 哈夫曼树集合，一颗哈夫曼树对应一个槽S；2) 哈夫曼树中的每个叶子节点对应一个实桶Act-B，实桶容纳一个单链表，依次存放具有相同规则点值的规则；3) 所有规则的规则点之间形成的一个下三角矩阵，表示冲突检测的结果。

利用MSSB算法构造出的槽S_2步骤为：

输入：m个规则数据

输出：包含所有不冲突规则对的冲突矩阵

//初始化后形成q个槽S。

1) Slots[1,2,…,q]=ConstructSlots(Rules[1,2,…,m])；

//初始化冲突矩阵(0代表不冲突; 1代表冲突)

2) ConflictMatrix[1, 2,…, (m^2-m)/2]=1；

3) for each Slots[i]

//对每个槽形成哈夫曼树的同时，完成冲突检测

4) HTree[i]=ConflicDec(Slots[i], ConflictMatrix[]);

5) end each

6) return ConflictMatrix[];

算法的三个主要函数：

//构造q个槽

Func ConstructSlots(Rules[1, 2,…, m])

1) for each Rules [i]

2) for RulePoints[j]

//属性值相同累加

3) CalculateNode(Slot[j]，RulePoints[j]);

//属性值相同插入链表

4) InsertNodeLink(Slot[j]， RulePoints[j]);

5) end for

6) end for

7) return Slots[1, 2,…, q];

Func ConflicDec(Slots[i]，ConflictMatrix[]) //对i个槽完成冲突检测

//经典的哈夫曼树构造算法

1) HTree[i]=Hufuman (Slots[i]);

2) CurLeaf=HTree[i].firstLeaf;

3) While (CurLeaf.next<>nil)

//获取两个实桶之间的不冲突规则对

4) GetNotConflicRule(CurLeaf,CurLeaf.next, ConflictMatrix[])

endWhile;

5) return HTree[i]

//求同槽的两个实桶所形成不冲突规则对集合

Proc GetNotConflicRule(CurLeaf, NextLeaf, ConflictMatrix[])

1) While (CurLeaf.nextNode<>nil)

2) While(NextLeaf.nextNode<>nil)

//更新冲突矩阵中的冲突结果

3) UpdateConflicMatrix(CurLeaf.nextNode, NextLeaf.nextNode, ConflictMatrix[]);

4) NextLeaf=NextLeaf.nextNode;

5) endWhile

6) CurLeaf=CurLeaf。nextNode;

7) endWhile

8) end

该算法的核心思想是在槽内利用传统的哈夫曼树形成规则匹配所需的数据结构，为利于冲突检测，将槽内实桶构造为树的叶子节点；在完成了单槽的哈夫曼树构造同时，立刻进行叶子节点之间的不冲突规则配对，并修改冲突矩阵结果；当完成所有槽的构造后，冲突矩阵的结果即为规则库不冲突规则集合。

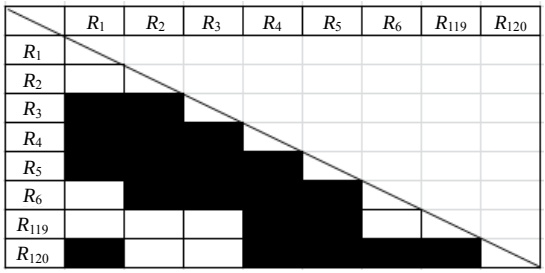

图 3　表 1 的冲突规则对分布图

图3为表1所示的规则集合在算法MSSB作用下的计算结果，横纵轴交互点的黑色部分表示两条规则之间不冲突，空白处对应的两条规则冲突。

3　实验及性能评估

本文的实验环境为：至强2×1.8 GHz CPU；8 GB Memory；2×143G DISK Size；1.5 Java Jdk；Red Linux ES版 OS；Eclipse3.5 IDE。

3.1　测试数据

实验中，为使上述算法的验证过程具有实用价值，分别选取3种类型(支线/干线/枢纽)的民航机场2010年生产数据为测试集。

表5　民航机场业务数据

	规则条数	槽数	单槽容量(平均)
支线机场	87	23	19
干线机场	2 376	23	42
枢纽机场	6 847	25	97

因规则冲突检测是规则引擎初始化阶段的一个工作环节，所以测试结果是在完整的规则引擎初始化构造所耗时间的总和度量算法性能的。

用文献[1-2]的PolicyTree结构及相应算法，与本文提出的MSSB算法在时间复杂度上逐一对比分析。

3.2　MSSB与其他方法的对比测试

实验依次用LinearDection (线性检测LD)算法、PolicyTree算法(PT)与MSSB算法就规则冲突检测的性能进行对比，最后统计多次执行结果平均时间为实验结果。表6为算法性能指标对比表；图5为冲突检测算法的时间性能；图6为算法的空间复杂度。

表 6 中 LDNode≈大小为 45 的字符串对象；PTNode≈大小为35的对象体；MSSBNode≈大小为20

的对象体。

表6 算法性能指标对比表

算 法		支线机场	干线机场	枢纽机场
Linear Dection	构造/s	0	0	0
	冲突检测/s	3.64	98.25	456.84
	空间/LDNode	2 001	54 648	171 175
Policy Tree	构造/s	4.34	13.69	40.07
	冲突检测/PTNode	3.57	70.43	239.11
	空间/s	4 264	163 995	543 525
MSSB	构造/s	4.97	9.54	28.61
	冲突检测/s	3.28	52.53	152.57
	空间/MSSBNode	4 278	65 313	159 450
Operator		“=”		

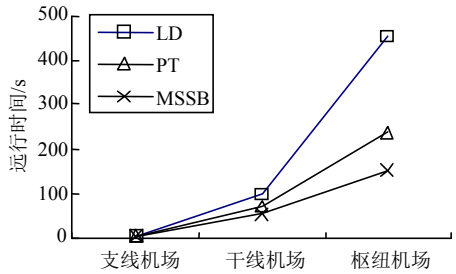

图 5 冲突检测算法的时间性能

表6和图5结果显示线性检测算法构造简单，构造算法的时间复杂度可忽略不计，但其冲突检测算法性能与规则数量有关，随着规则数的增加而增加；本文提出的MSSB算法基于TrieTree结构合并了规则的重复属性值，其构造算法时间复杂度与PolicyTree算法相比减少了近30.3%(干线机场)和28.6%(枢纽机场)；图5所示的冲突检测性能结果表明，MSSB算法较之PolicyTree算法提升了近25.4%(干线机场)和36.2%(枢纽机场)。

4 结 论

为解决民航机场业务规则集合制订中所产生的规则自我冲突问题，本文提出了基于多槽分桶的快速规则冲突检测算法MSSB。该算法利用同槽实桶之间规则对两两必不冲突特性，将复杂的冲突规则求解转换为线性时间内的不冲突规则求解，从而在更低的空间复杂度和时间复杂度下处理了规则冲突发现问题。实测结果表明，该算法在冲突检测性能上较PolicyTree算法提高36.2%。下一步的工作将重点针对规则冲突的分类识别机制以及规则冲突的自我摆脱等问题展开研究。

参 考 文 献

[1] AL-SHAER E S, HAMED H H. Conflict classificatinand analysis of distributed firewall policies[J]. IEEE Journal on Selected Areas in Communications-JSAC, 2005, 23(10): 2069-2084.

[2] AL-SHAER E S, HAMED H H. Discovery of policyanomalies in distributed firewalls[C]//INFOCOM 2004.Twenty-Third AnnualJoint Conference of the IEEE Computer and Communications Societies. [S.l.]: IEEE, 2004.

[3] 王卫平, 陈文惠, 朱卫未, 等. 防火墙规则配置错误快速检测算法[J]. 计算机工程, 2007, 33(11): 132-134.
WANG Wei-ping, CHEN Wen-hui, ZHU Wei-wei, et al. Algorithm for fast detecting firewall rule configuration mistakes[J]. Computer Engineering, 2007, 33(11): 132-134

[4] 罗谦, 唐常杰, 于磊, 等. 基于多槽哈夫曼Trie树的规则引擎快速匹配算法[J]. 四川大学学报(工程科学版), 2011, 43(5): 102-108.
LUO Qian, TANG Chang-jie, YU Lei, et al. A fast matching algorithm for rule engines based on multi-slot Huffman Trie tree[J]. Journal of SiChuan University(Engineering Science Edition), 2011, 43(5): 102-108.

[5] 田大新, 刘衍珩, 李永丽, 等. 数据包过滤规则的快速匹配算法和冲突检测[J]. 计算机研究与发展, 2005, 42(7): 108-116.
TIAN Da-xin, LIU Yan-heng, LI Yong-li, et al. A fast matching algorithm and conflict detection for packet filter rules[J]. Journal of Computer Research and Development, 2005, 42(7): 108-116.

[6] 李鑫, 季振洲, 刘韦辰, 等. 防火墙过滤规则集冲突检测算法[J]. 北京邮电大学学报, 2006, 29(4): 23-26.
LI Xin, JI Zhen-zhou, LIU Wei-chen, et al. An algorithm for detecting firewall filters conflicts[J]. Journal of Beijing University of Posts and Telecommunications, 2006, 29(4): 23-26.

[7] 梁建武, 龙晓梅, 刘军军. 基于LE-Trie的防火墙策略检测算法[J]. 计算机工程, 2010, 36(22): 134-136.
LIANG Jian-wu, LONG Xiao-mei, LIU Jun-jun. Detection algorithm for firewall policy based on LE-Trie[J]. Computer Engineering, 2010, 36(22): 134-136.

[8] CUPPENS F, CUPPENS B N, GARC’A A J. Detecting and removal of firewall misconfiguration[C]//Proceeding (499) Communication, Network, and Information Security. [S.l.]: [s.n.], 2005.

[9] ERONEN P, ZITTING J. An expert system for analyzing firewall rules[C]//Proceedings of the 6th Nordic Workshop on Secure. [S.l.]: [s.n.], 2001.

[10] WANG Dong, HAO Rui-bing, LEE D. Fault detection in rule-based software systems[J]. Information and Software Technology, 2003, 45(12): 865-871.

[11] QIU Li-li, VARGHESE G, SURI S. Fast firewall implementations for software and hardware-based routers[C]// Proceedings of the 2001 ACM SIGMETRICS International Conference on Measurement and modeling of Computer Systems. [S.l.]: ACM, 2001.

编 辑 税 红