# 基于冗余消除和属性数值化的XACML策略优化方法

戚 湧 陈 俊 李千目
(南京理工大学计算机科学与工程学院 南京 210094)

**摘 要** 可扩展的访问控制标记语言(eXtensible Access Control Markup Language,XACML)逐渐成为访问控制的标准之一。为了确保系统可用性,访问控制系统需要高效的XACML策略评估引擎。针对这一问题,从XACML策略本身潜在的不足出发,从冗余消除和属性数值化两个方面对XACML策略进行了优化。冗余消除在不影响策略评估结果的前提下去除策略库中的冗余规则,同时结合规则压缩消除规则间的冗余状态。属性数值化将文本的XACML策略属性转化为数值属性,使评估引擎匹配使用高效的数值匹配方式而不是低效的字符串匹配方式,同时使用Hash表结构存储数值属性与文本属性的映射关系有利于策略维护。仿真实验结果表明,提出的策略优化方法的性能与原始Sun XACML相比有较大提升。

**关键词** XACML,策略优化,冗余消除,属性数值化
**中图法分类号** TP309 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2016.2.036

## XACML Policy Optimization Method Based on Redundancy Elimination and Attribute Numericalization

QI Yong CHEN Jun LI Qian-mu
(School of Computer Science and Engineering,Nanjing University of Sicnece and Technology,Nanjing 210094,China)

**Abstract** XACML (eXtensible Access Control Markup Language) has become one of main access control standards. Access control systems need effective XACML evaluation engine to ensure system availability. To solve the problem above,this paper optimized XACML policy from two aspects: redundancy elimination and attribute numericalization, based on the potential shortcomings of XACML itself. Redundancy elimination removes the redundant rules in the policies and the redundant states between the rules by applying rule compression method. Attribute numericalization transforms textuary attributes of XACML policies into numerical attributes,to make evaluation engine use effective numerical match,instead of inefficient string match. In addition,it is beneficial for policy management that using Hash table to store the mappings between textuary attributes and numerical attributes. Simulation experimental results show that the policy engine using the policy optimization method proposed in this paper is much faster than Sun XACML.

**Keywords** XACML,Policy optimization,Redundancy elimination,Attribute numericalization

## 1 引言

如何确保信息的安全性以及对关键资源的安全访问已经成为重要的研究课题。在众多安全服务中,访问控制是确保系统安全的重要手段之一。为适应当前分布式系统架构以及多域访问控制等Web服务的特点,OASIS专门设计了一套针对Web服务访问控制的标准语言:可扩展的访问控制标记语言[1](eXtensible Access Control Markup Language,XACML)。XACML标准通过创建策略和规则并检查用户属性是否符合相关策略或规则要求来决定用户是否能够调用相应Web服务。安全管理员可以定义相应的策略和规则来细化访问控制权限。

由于XACML模型在设计之初并未考虑产生大量策略的情形,因此策略评估引擎的决策方法就是在策略库中强力搜索与之匹配的策略并做出决定。这在策略规模较小的情况下影响并不明显,然而一旦策略库规模增大,影响将十分显著。目前XACML相关研究工作[2-8]主要集中在策略建模、验证、分析、测试以及与其他模型相结合,却忽视了大规模策略库下策略评估的效率问题。随着访问控制对象数量的快速增长,策略和规则的数量也相应激增,策略评估性能已经成为制约系统可用性的瓶颈,所以亟需提升策略评估引擎性能以保证系统可用性。

Sun公司在OASIS组织制定XACML标准后推出了它的策略访问控制评估原型系统Sun XACML[9],定义了XACML的核心组件以及相应的检索组件接口,如属性检索组件接口、资源检索组件接口、策略检索组件接口,并且开发人员可以对上述检索组件接口的具体实现进行自定义。后来出现的访问控制评估系统大多都是该系统的扩展或改进。虽然Sun XACML按照XACML标准实现了完整的访问控制体系架构,但其没有进一步考虑策略检索和属性匹配的效率给策

到稿日期:2015-01-16 返修日期:2015-04-20 本文受国家自然科学基金项目(61272419)资助。
戚 湧(1970-),男,博士后,教授,博士生导师,CCF高级会员,主要研究方向为网络信息安全,E-mail:790815561@qq.com;陈 俊(1990-),男,硕士生,主要研究方向为网络信息安全;李千目(1979-),男,博士后,教授,博士生导师,主要研究方向为网络信息安全。

略评估引擎带来的性能方面的影响，比如当匹配访问请求时，需要暴力逐条检索策略库。在系统环境简单的小规模策略应用场景下，暴力逐条检索的匹配方式对策略检索的效率影响不大；但在系统环境复杂的大规模策略集成应用场景中，真正适用于访问请求的很小部分策略可能会随机分布在成千上万的策略中，暴力逐条检索模式下系统检索到有效策略的概率可能会相当低。目前，提高策略评估性能方面的工作主要围绕规则精化、缓存、索引等方面展开。Enterprise XACML[10]系统根据目标中包含的属性标记建立策略索引结构，缩小了策略检索空间。XEngine 系统[11]将 XACML 策略规则和请求转换为数字表示的形式，把嵌套递归式描述结构转换为扁平结构，采用首次适用合并算法避免规则的穷举匹配，但是其依据数值区间的数值映射，灵活性不强，不利于策略维护。牛德华等[12]提出的 HPEngine 使用统计分析动态精化策略，并将精化的策略由字符串形式转换为数值形式。但其文本属性至数字属性的映射结构依旧没有优化。MLOBEE 系统[13]综合考虑规则精化、缓存、索引因素，在判定评估前对策略进行预处理，通过规则精化缩减策略规模并调整规则顺序，运用多级缓存以及两级索引提高了策略匹配效率。但是其规则精化中的冗余消除只能到规则层次，即最多只能消除冗余规则，而无法消除规则之间存在的冗余状态。

XACML 是一种基于属性的访问控制标准语言，其基于细粒度的授权约束非常适用于如今互联网下这种分布式异构网络访问控制场景。然而，这也同时带来了策略和规则的繁杂，容易产生冲突和冗余规则等问题。同时，XACML 中很多属性值都是用 ASCII 字符串表示，策略评估引擎为访问请求进行决策时，需要使用低效的字符串匹配算法进行匹配，不适合规模庞大的策略库和用户访问非常频繁的分布式系统。本文从冗余消除和属性数值化两个方面对 XACML 策略进行优化。

## 2 背景知识

XACML 模型分为数据流模型和策略语言模型。

XACML 数据流模型由策略管理点(Policy Administration Point，PAP)、策略信息点(Policy Information Point，PIP)、策略决策点(Policy Decision Point，PDP)、策略执行点(Policy Enforcement Point，PEP)和上下文处理器(Context Handler)5 大部分组成。PAP 负责在系统中产生和维护安全策略，供 PDP 调用，主要由管理员操作；PIP 负责收集主体、资源和环境的属性信息，提供给 PDP；PDP 是系统中授权决策的实体，依据 XACML 描述的访问控制策略以及其他属性信息做出授权决策，并将结果返回给 PEP；典型的执行过程是：PEP 发送 XACML 请求给 PDP，PDP 导入访问控制策略，对访问控制策略进行组合计算并得出结论，PDP 发送 XACML 格式的响应信息；PEP 是在一个具体的应用环境下执行访问控制的实体。将具体应用环境下访问控制请求转换为一个基于主体、资源、动作和环境的决策请求发送给 PDP；然后根据 PDP 对决策请求的判决结果做出授权决策，如允许用户请求和拒绝用户请求等。典型的执行过程为：用户请求访问 Web 服务，系统的 PEP 收集用户和被访问的资源及其相关动作，PEP 将所收集的信息编成 XACML 格式的请求并发送给 PDP，PEP 等待 PDP 做出决策，PDP 做出决策并发送响应给 PEP，PEP 解析决策从而决定是否允许访问；Context Handler 将 PEP 发来的决策请求转换成 PDP 可以理解的 XACML 标准的决策请求，并将 PDP 返回的授权决策转换成为访问者能够理解的用户本地格式。

XACML 策略语言模型由 4 大基本部件组成：策略集(PolicySet)、策略(Policy)、规则(Rule)和合并算法(Combining Algorithm)。Rule 是策略树的叶子节点，也是执行评估的最小单位，必须分装到策略中，并且可以单独存在于 XACML 域的主要角色中。规则 Target 元素以请求属性的形式描述了规则适用的请求集，包括 4 个基本属性元素：资源(Resource)、主体(Subject)、动作(Action)和环境(Environment)。Condition 是一个布尔表达式，除可用 Target 中隐含的限制外，还可用 Condition 来明确规则的约束条件，因此 Condition 可以省略。Effect 表示当访问请求中的 Target(包括主体、资源、动作和环境)与规则中的 Target 匹配时该规则给出的决策结果，包括许可(Permit)和拒绝(Deny)。规则制定者还可以增加 Obligation Expressions 到规则中，且当 PDP 检测到一个规则中包含 Obligation Expressions 时，它将把 Obligation Expressions 转换成职责，并在响应内容中把转换得到的职责返回给 PEP。规则制定者也可以增加 Advice Expressions 到规则中，且当 PDP 检测到一个规则中包含 Advice Expressions 时，它将把 Advice Expressions 转换成建议，并在响应内容中把转换得到的建议返回给 PEP。与职责不同的是，建议可能会被 PEP 直接忽略掉。策略元素是一组规则和一些保护资源的条件信息。一条策略包含一个目标、一组规则、一个规则合并算法、职责表达式和建议表达式。策略 Target 元素与规则 Target 类似，表示策略适用的请求集。PolicySet、Policy、Rule都必须包含 Target 元素以用于给定请求。策略 Target 的属性也同样包含主体(Subject)、资源(Resource)和动作(Action)。与规则 Target 不同的是，策略和策略集 Target 有两种方法指定，一种是制定策略时在策略中明确设置，另一种是根据其内部的规则或策略通过合并算法计算归纳而出。如果使用第一种方法指定策略和策略集 Target，那么其内部组件的 Target 将会被忽略，全部使用其指定的 Target。请求中的属性值会与 Target 元素中的属性值做比较，如果所有属性值匹配，则说明该请求返回适用(Applicable)；如果请求与 Target 属性不匹配，则该请求返回不适用(NotApplicable)；如果匹配错误，则该请求返回不确定(Indeterminate)，具体情况由合并算法决定。如果一个请求满足一条策略的目标元素，则会继续验证该请求与该策略所有规则的目标元素是否匹配；反之，如果一个请求不满足一条策略，则会跳过策略，而不会继续验证该策略的规则。规则合并算法能够在规则冲突时给出决策结果。PolicySet 是 XACML 的顶层元素，可以包含若干策略或其他策略集，当对策略集进行评估时，策略合并算法给出一个明确评估标准来给出策略集的评估结果。其他部分与 Policy 类似。XACML 中合并算法根据适用的层次可分为策略合并算法和规则合并算法。具体来说，如果 Policy 内的多条 Rule 都适用于访问请求，则评估引擎根据规则合并算法确定该 Policy 的最终评估结果；如果 PolicySet 内的多条 Policy 都适用于访问请求，则评估引擎根据策略合并算法确定该 PolicySet 的最终评估结果。XACML 3.0 支持 12 种合并算法，并支持自定义合并算法[1]。常见的合并算法有以下几种：

1)许可优先(Permit-Overrides),只要有一条策略/规则的评估结果为 Permit,最终的评估结果就是 Permit。

2)有序许可优先(Ordered-Permit-Overrides),与许可优先相同,只是按照策略/规则在策略集/策略中的顺序评估相关策略/规则。

3)拒绝优先(Deny-Overrides),只要有一条策略/规则的评估为 Deny,最终的评估结果就是 Deny。

4)有序拒绝优先(Ordered-Permit-Overrides),与拒绝优先相同,只是按照策略/规则在策略集/策略中的顺序评估相关策略/规则。

5)首次适用(First-Applicable),在策略集/策略中遍历匹配策略/规则时,将遇到的第一条适用请求的策略/规则的评估结果作为最终的评估结果。

6)唯一适用(Only-One-Applicable),在策略集/策略中有且只有一条策略/规则适用请求时,将该策略/规则的评估结果作为最终的评估结果。

一个完整的 XACML 策略是一个 XML 格式文件,有组织地记录了策略标识符、规则、目标等。策略决策点通过查找相应的目标与主体并验证主体所在的策略是否符合访问控制要求,来做出访问控制决策。

## 3 策略优化方法

### 3.1 冗余消除

#### 3.1.1 相关定义及冗余分析

文献[14]根据规则状态覆盖进行冗余分析的方法中给出的相关性定义与请求的适用性略有偏差,因此本文以请求适用性为基础引入如下定义。

**定义 1**(规则状态) 令规则 Target 中单个 Subject 属性值、Resource 属性和 Action 属性组成的一个断言为规则的一个状态。规则 Target 的各属性都有可能是集合而非单个值,所以一个规则可能有多个状态。而对于那些只含有一个状态的规则,本文称之为单状态规则。

**定义 2**(规则状态空间) 规则 Target 包含 3 个属性集合(除 Environment 属性):Subject 集合、Resource 集合、Action 集合。这 3 个集合的笛卡尔积形成了该规则的状态空间。令 $SUB$、$RES$、$AC$ 分别表示规则 Target 的 Subject 集合、Resource 集合、Action 集合,那么规则 $r_i$ 的状态空间 $r_i.State\_Space=r_i.T.SUB\times r_i.T.RES\times r_i.T.AC$。

定义了规则状态空间之后,可以确定两个规则之间的状态空间必然存在某种相关关系。下面分别给出规则之间的状态空间覆盖、相交和无关的定义。

**定义 3**(规则状态空间覆盖) 对于两个规则 $r_i$ 和 $r_j$,若 $r_i.State\_Space\subseteq r_j.State\_Space$,则 $r_i$ 的状态空间覆盖 $r_j$ 的状态空间,记为 $r_i\leqslant r_j$,否则 $r_i\not\leqslant r_j$。

$r_i\leqslant r_j$ 表示 $r_i$ 中的状态在 $r_j$ 中都存在,在这种相关关系下若 $r_i$ 适用于某请求,那么 $r_j$ 必然也适用于该请求,后文的冗余规则判定将会以此为基础。另外,判断时无需计算出规则的状态空间,因为若 $(r_i.T.SUB\subseteq r_j.T.SUB)\wedge(r_i.T.RES\subseteq r_j.T.RES)\wedge(r_i.T.AC\subseteq r_j.T.AC)$,则必然 $r_i\leqslant r_j$。

**定义 4**(规则状态空间相交) 对于两个规则 $r_i$ 和 $r_j$,若 $(r_j.State\_Space\cap r_j.State\_Space\neq\varnothing)\wedge(r_i\not\leqslant r_j)\wedge(r_j\not\leqslant r_i)$,则 $r_i$ 的状态空间与 $r_j$ 的状态空间相交,记为 $r_i\Leftrightarrow r_j$,否则 $r_i\not\Leftrightarrow r_j$。

$r_i\Leftrightarrow r_j$ 表示 $r_i$ 和 $r_j$ 中有共同的状态,并且 $r_i$ 和 $r_j$ 的状态空间之间没有覆盖关系,这种相关关系下,$r_i$ 和 $r_j$ 可能在某些情况下同时适用于同一请求。同样地,这里也无需计算规则的状态空间,因为 $r_i.State\_Space\cap r_j.State\_Space\neq\varnothing$ 等价于 $(r_i.T.SUB\cap r_j.T.SUB\neq\varnothing)\wedge(r_i.T.RES\cap r_j.T.RES\neq\varnothing)\wedge(r_i.T.AC\cap r_j.T.AC\neq\varnothing)$。

**定义 5**(规则状态空间无关) 对于两个规则 $r_i$ 和 $r_j$,若 $r_i.State\_Space\cap r_j.State\_Space=\varnothing$,则 $r_i$ 的状态空间与 $r_j$ 的状态空间无关,记为 $r_i\parallel r_j$。

$r_i\parallel r_j$ 表示 $r_i$ 和 $r_j$ 中无共同的状态,完全无关,这种相关关系下 $r_i$ 和 $r_j$ 不可能同时适用于同一请求。该相关关系一般不直接判断,而是通过判断既非覆盖关系又非相交关系来得到,因为规则状态空间的相关关系只有覆盖、相关、无关 3 种。

冗余的判断通过规则状态空间的相关性分析得出。下面分析各相关性下的冗余情况。

规则状态空间无关情况下,两个规则无共同状态,不可能同时适用于同一请求,所以不存在冗余情况。

规则状态空间相交情况下,两个规则有共同状态且不存在覆盖关系,可能在某些情况下同时适用于同一请求,存在状态冗余。然而有些情况下只有一个规则适用于请求,所以不能判定两个规则之一为冗余规则。

规则状态空间覆盖情况下,若 $r_i\leqslant r_j$ 则 $r_j$ 中的状态在 $r_j$ 中都存在,若 $r_i$ 适用于某请求。那么 $r_j$ 必然也适用于该请求。若 $r_i$ 和 $r_j$ 的 Effect 相同,则 $r_i$ 的存在对评估结果不会有影响,$r_i$ 为冗余规则;若 $r_i$ 和 $r_j$ 的 Effect 不相同,则需根据具体的合并算法决定。

#### 3.1.2 冗余消除规则

虽然文献[14]给出的规则状态空间的概念与本文的略有差别,但是与在状态覆盖下的冗余判定方法类似,所以本文将借鉴文献[13]的冗余规则判定定理给出冗余规则判定定理,证明过程不再赘述。

**定义 6** 访问请求 $req(sub,res,ac)$,表示发送至 PDP 的 XACML 格式请求,其中 $sub$、$res$ 和 $ac$ 分别表示访问请求的主体、资源和动作。若规则 $r$ 适用于 $req(sub,res,ac)$,则记为 $r\vdash req(sub,res,ac)$。

**定理 1** 在 Permit-Overrides 规则合并算法下,若 $r_i\leqslant r_j$ 且 $r_j.E=Permit$,则 $r_i$ 是冗余规则。

**定理 2** 在 Permit-Overrides 规则合并算法下,若 $r_i\leqslant r_j$ 且 $r_j.E=Deny$,则当 $r_i.E=Deny$ 时,$r_i$ 是冗余规则;当 $r_i.E=Permit$ 时,$r_i$ 不是冗余规则。

**定理 3** 在 Deny-Overrides 规则合并算法下,若 $r_i\leqslant r_j$ 且 $r_j.E=Deny$,则 $r_i$ 是冗余规则。

**定理 4** 在 Deny-Overrides 规则合并算法下,若 $r_i\leqslant r_j$ 且 $r_j.E=Permit$,则当 $r_i.E=Permit$ 时,$r_i$ 是冗余规则;当 $r_i.E=Deny$ 时,$r_i$ 不是冗余规则。

**定理 5** 在 First-Applicable 规则合并算法下,若 $r_i\leqslant r_j$ 且 $r_i.E=r_j.E$,则 $r_i$ 是冗余规则。

**定理 6** 在 First-Applicable 规则合并算法下,令 $seq(r)$ 表示规则 $r$ 在策略中的位置顺序。若 $r_i\leqslant r_j$ 且 $r_i.E\neq r_j.E$,则当 $seq(r_j)<seq(r_i)$ 时,$r_i$ 是冗余规则;当 $seq(r_j)>seq(r_i)$ 时,$r_i$ 不是冗余规则。

在 Only-One-Applicable 规则合并算法的策略中,若规则

$r_i$ 和 $r_j$ 同时适用于某一请求，评估引擎将返回“Indeterminate”。因此，该算法不适宜分析规则冗余的情况，此处不再分析。

依据上述规则能够在不影响策略评估结果的前提下去除策略库中的冗余规则，缩小策略库规模，提高评估效率。

3.1.3 冗余状态消除

在经过上述冗余规则消除之后，不能继续消除状态空间相交的规则之间的冗余状态。因为规则是一个整体，要么删除，要么保留，目前还没有方法通过修改规则来避免这种冗余。总之，造成这种困境的最根本原因是冗余消除的粒度太大，所以需要一种方法来减小粒度。文献[15]中规则压缩的思想是将简单规则集成为复杂规则，受其启发，本文进行逆向思考，首先将复杂规则分解为简单规则。例如对于规则 $r_1$：

$Subject \in \{Sub1, Sub2\} \wedge Resource$ is $Res1 \wedge Action \in \{Ac1, Ac2\} \Rightarrow$Permit

可以将 $r_i$ 分解为如下单状态规则：

$r_{11}$：$Subject$ is $Sub1 \wedge Resource$ is $Res1 \wedge Action$ is $Ac1 \Rightarrow$ Permit

$r_{12}$：$Subject$ is $Sub1 \wedge Resource$ is $Res1 \wedge Action$ is $Ac2 \Rightarrow$ Permit

$r_{13}$：$Subject$ is $Sub2 \wedge Resource$ is $Res1 \wedge Action$ is $Ac1 \Rightarrow$ Permit

$r_{14}$：$Subject$ is $Sub2 \wedge Resource$ is $Res1 \wedge Action$ is $Ac2 \Rightarrow$ Permit

通过这种方式，可以将规则的粒度降至最低，这样就可以根据 3.1.2 节冗余规则判定定理找出冗余规则并将其消除。对于规则 $r_2$：

$Subject \in \{Sub1, Sub3\} \wedge Resource$ is $Res1 \wedge Action \in \{Ac1, Ac2\} \Rightarrow$Deny

可以将 $r_2$ 分解为如下单状态规则：

$r_{21}$：$Subject$ is $Sub1 \wedge Resource$ is $Res1 \wedge Action$ is $Ac1 \Rightarrow$ Deny

$r_{22}$：$Subject$ is $Sub1 \wedge Resource$ is $Res1 \wedge Action$ is $Ac2 \Rightarrow$ Deny

$r_{23}$：$Subject$ is $Sub3 \wedge Resource$ is $Res1 \wedge Action$ is $Ac1 \Rightarrow$ Deny

$r_{24}$：$Subject$ is $Sub3 \wedge Resource$ is $Res1 \wedge Action$ is $Ac2 \Rightarrow$ Deny

假设 $r_1$ 和 $r_2$ 属于策略 $P$，$P$ 的规则合并算法为 Deny-Overrides。根据规则状态空间的定义，$r_1 \Leftrightarrow r_2$，将 $r_1$ 和 $r_2$ 分解为上述的 8 个单状态规则后使用 3.1.2 节的冗余规则判定定理可以得出 $r_{11}$ 和 $r_{12}$ 为冗余规则，将其消除后还剩 $r_{13}$、$r_{14}$、$r_{21}$、$r_{22}$、$r_{23}$ 和 $r_{24}$ 这 6 个单状态规则。使用文献[15]的规则压缩算法将这 6 个单状态按 Effect 进行压缩可以得出 $r_1'$ 和 $r_2'$：

$Subject$ is $Sub2 \wedge Resource$ is $Res1 \wedge Action \in \{Ac1, Ac2\} \Rightarrow$ Permit

$Subject \in \{Sub1, Sub3\} \wedge Resource$ is $Res1 \wedge Action \in \{Ac1, Ac2\} \Rightarrow$Deny

通过以上方法，本文将状态空间相交的规则间的冗余状态消除，更细粒度地消除了冗余。主要分为以下几个步骤：

1）若两个规则状态空间相交，则将两个规则分解为多个单状态规则；

2）使用冗余规则判定定理对产生的单状态规则进行冗余规则消除；

3）使用规则压缩算法将剩余的单状态规则按 Effect 分别压缩。

### 3.2 属性数值化

文献[11]提出策略属性的数值化，但受其系统限制，策略的同一属性必须是连续的整数区间，这增加了策略维护的成本，增加或删除属性可能会对后续规则属性的整数映射产生影响。本文通过采用开放地址法处理 Hash 冲突的 Hash 表结构存储的整数属性值到 ASCII 字符串属性值的映射关系。

在策略库预处理以及策略管理中为主体、资源、操作属性中所有的字符串属性建立散列表，使用散列函数计算 XACML 策略中出现的每个字符串属性的散列值，并将其作为映射的整数，将文本的 XACML 策略属性转化为数值的 XACML 策略属性，同时当请求到达时，使用散列函数将请求中的字符串属性替换为整数属性，使评估引擎使用高效的整数而不是低效的字符串匹配算法进行比较。策略属性值 Hash 结果示例如表 1 所列。

表 1 策略属性值 Hash 结果

| Subject | Resource | Action |
|---|---|---|
| Sub1:2588177 | Res1:2543537 | Ac1:65583 |
| Sub2:2588178 | Res2:2543538 | Ac2:65584 |
| | Res3:2543539 | |

在预处理策略库以及所有的请求到达 PDP 时，如果策略库每条 Target 或请求中包含的属性类型为字符串（http://www.w3.org/2001/XMLSchema#string）且匹配函数为 XACML 标准中的字符串匹配函数（urn:oasis:names:tc:xacml:1.0:function:string-equal），则根据 Hash 函数将字符串型属性转成对应整数（http://www.w3.org/2001/XMLSchema#integer），同时将匹配函数变为整数相等（urn:oasis:names:tc:xacml:1.0:function:integer-equal）。

这种方式下，虽然在接收到请求属性映射时需要进行一次字符串比较，从而会占用一些时间资源，但相对于在整个策略库中进行大量的字符串比较，其时间损耗几乎可以忽略不计。同时，由于内存中一直需要维持一个字符串属性映射至整数属性值的 Hash 表，需要耗费内存空间资源，因此本质上这是一种以空间换取时间的方法。但是随着计算机内存成本的降低以及系统对策略评估性能的强烈需求，这种以空间换取时间的方法非常符合系统的发展以及需求。

安全管理员维护策略时需要直观地看到字符串属性以帮助制定策略，这时需要将内存中数值化的策略还原。还原过程只需根据整数去相应的空间取出字符串属性进行替换即可，所以使用散列表能够高速执行属性还原过程，方便策略维护。由时间复杂度分析可知，每条属性的数值化和还原过程的时间复杂度都为 O(1)，几乎不影响策略评估性能。

本文使用开放地址法来避免因两个不同的字符串属性得到相同的散列值而造成的冲突。具体的方法可在线性探查法（Linear Probing）、二次探查法（Quadratic Probing）、双重散列法（Double Hashing）这几种方法中选择。这几种方法各有利弊，计算成本也不尽相同，可根据系统环境以及环境分析利弊进行选择。

## 4 仿真实验与分析

本文开展仿真实验以验证提出方法的有效性。首先将两

种方法与 Sun XACML PDP 进行评估性能比较，然后进行融合，再与 Sun XACMLPDP 进行性能比较，得出量化结果。实验环境为：2.6GHz Intel Core i5 CPU，16GB DDR3 内存，OS X Yosemite 操作系统，语言环境为 Java Runtime Environment 1.8.0_25，开发环境为 Eclipse Luna Release(4.4.0)。

为了实验的科学性和有效性，首先给出测量 Sun XACML 原始模型及各优化方法的实验步骤和结果。因为 Permit-Overrides 和 Deny-Overrides 是最为常用的两种策略/规则合并算法，且其他很多种策略/规则合并算法都是在这两种算法基础上衍生而出的，同时为了方便比较，本文实验中 Request 请求都基于这两种策略/规则合并算法。因为实验总的运行周期较长，占用主机资源较高，不可避免地要受到网络和主机等相关因素的影响，在具体实验中由于实验室网络和主机运行程序较多等原因也会产生较大的数据偏差，因此在以下的实验里将每组数据测量 3 次求其平均值，这样就可以尽量减小因网络传输、主机资源占用等对数据偏差造成的影响。虽然有可能有些数据仍然存在偏差，但是基本能够反映出数据的本质和趋势。

实验数据如表 2 所列，测试都是基于 1000 个请求产生的，其中策略库的数量从 500 逐步涨至 10000。为了能尽量减小误差，本次实验的步骤重复 3 次，总的平均时间由 3 次平均时间再求平均得到，以防止产生离群数据，尽量缩小影响。从表 2 中数据可以看出，两种优化方法的运行时间相对于 Sun XACML 来说都有较大减少，效率得到较大提升，且将两种优化方法进行融合后，效果更好。其中，冗余消除将性能提升了大约 3～3.6 倍；属性数值化将性能提升了大约 7～8 倍，优化方法融合后性能提升了约 12～14 倍。

表 2　实验数据(单位：ms)

| 规则数量 | Sun XACML 运行时间 | 冗余消除运行时间 | 属性数值化运行时间 | 优化方法融合运行时间 |
|---|---|---|---|---|
| 500 | 25.23 | 7.796 | 3.102 | 1.716 |
| 1000 | 50.675 | 16.544 | 6.168 | 3.605 |
| 1500 | 71.303 | 19.528 | 9.767 | 4.789 |
| 2000 | 93.803 | 29.61 | 12.834 | 7.252 |
| 2500 | 119.272 | 39.733 | 16.993 | 10.132 |
| 3000 | 136.284 | 37.268 | 18.354 | 8.984 |
| 3500 | 164.113 | 45.751 | 23.455 | 11.704 |
| 4000 | 184.778 | 59.255 | 24.78 | 14.224 |
| 4500 | 206.621 | 62.177 | 27.137 | 14.618 |
| 5000 | 242.047 | 74.557 | 31.651 | 17.451 |
| 5500 | 261.038 | 72.295 | 34.926 | 17.314 |
| 6000 | 275.167 | 86.906 | 36.774 | 20.79 |
| 6500 | 298.815 | 97.295 | 40.907 | 23.842 |
| 7000 | 321.791 | 96.203 | 42.06 | 22.508 |
| 7500 | 336.365 | 106.626 | 47.352 | 26.869 |
| 8000 | 361.196 | 108.919 | 52.895 | 28.552 |
| 8500 | 392.057 | 117.233 | 53.461 | 28.614 |
| 9000 | 410.941 | 121.411 | 56.028 | 29.63 |
| 9500 | 427.64 | 120.102 | 61.482 | 30.908 |
| 10000 | 457.729 | 131.099 | 64.927 | 33.286 |

仿真实验数据折线图如图 1 所示，可以看出冗余消除的平均评估时间随着规则数量的增大而增加。然而该曲线并不像 Sun XACML 那么平滑，它频繁出现抖动，在规则数为 3000、5500、7000 以及 9500 时平均运行时间甚至出现了下降。出现这种情况的主要原因是策略库中的冗余数量具有随机性，在不同规则数的策略中冗余规则比例不尽相同，所以消除冗余规则后的剩余规则数并不按初始的规则数规律增加。属性数值化的平均评估时间随着规则数量的增大而相对平稳增加，没有出现非常大的波动，这是因为优化效果由数值匹配替代字符串匹配产生，随着规则数量的增大，其性能提升不会产生较大的浮动，大体上是数字匹配与字符串匹配的效率比；在少数几个点出现的小波动应该是字符串长度等随机原因导致的；而将这两种方法融合后，性能可以进一步得到提升。

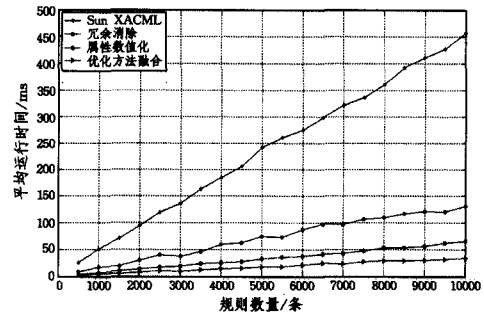


图 1　仿真实验数据折线图

**结束语**　本文从 XACML 策略本身潜在的不足出发，从两个方面对 XACML 策略进行优化。首先提出规则状态、规则状态空间及其相关性等概念，以集合为基础形式化表达了规则之间在请求匹配层面的相关关系；并结合规则压缩算法，对规则进行更深层次的冗余消除，不仅能够消除冗余规则，还能够消除规则之间存在的冗余状态。在属性数值化优化方法中使用 Hash 表存储数值到字符串的映射关系，增强了灵活性，方便维护。仿真实验结果表明两种优化方法与 Sun XACML 相比，对性能都有不同程度提升，且将两种优化方法进行融合后策略评估引擎性能进一步得到提升。

## 参 考 文 献


[1] Standard OASIS. eXtensible Access Control Markup Language (XACML) Version 3.0. [S/OL]. 2013. http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html

[2] Ramli C D P K, Nielson H R, Nielson F. The logic of XACML [J]. Science of Computer Programming, 2014, 83: 80-105

[3] Bertolino A, Daoudagh S, Lonetti F, et al. Xacmut: Xacml 2.0 mutants generator[C]// 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops (ICSTW), 2013. IEEE, 2013: 28-33

[4] El Kateb D, Elrakaiby Y, Mouelhi T, et al. Towards a Full Support of Obligations In XACML[C]// 9th International Conference on Risks and Security of Internet and Systems. 2014

[5] Lunardelli A, Matteucci I, Mori P, et al. A prototype for solving conflicts in XACML-based e-Health policies[C]// 2013 IEEE 26th International Symposium on Computer-Based Medical Systems (CBMS), 2013. IEEE, 2013: 449-452

[6] Le T T K, Van H D S, Dang A T, et al. Towards a Flexible Framework to Support a Generalized Extension of XACML for Spatio-temporal RBAC Model with Reasoning Ability[J]. International Journal of Web Information Systems, 2014, 10(2): 437-451

[7] Ryba G, Jung M, Kastner W. Authorization as a service in smart grids: Evaluating the PaaS paradigm for XACML policy decision points[C]// 2013 IEEE 18th Conference on Emerging Technologies & Factory Automation (ETFA), 2013. IEEE, 2013: 1-4

[8] De la Rosa Algarín A, Ziminski T B, Demurjian S A, et al. Generating XACML Enforcement Policies for Role-Based Access Control of XML Documents[C]// Web Information Systems and

Technologies, 2014. Springer Berlin Heidelberg, 2014: 21-36
[9] Sun XACML[OL]. 2006. http://sunxacml.sourceforge.net
[10] Enterprise XACML[OL]. 2008. http://code.google.com/p/enterprise-java-xacml
[11] Liu A X, Chen Fei, Hwang J H, et al. Designing fast and scalable XACML policy evaluation engines[J]. IEEE Trans on Computers, 2011, 60(12): 1802-1817
[12] Niu De-hua, Ma Jian-feng, Ma Zhuo, et al. HPEngine: high performance XACML policy evaluation engine based on statistical analysis[J]. Journal on Communications, 2014, 35(8): 206-215 (in Chinese)
牛德华, 马建峰, 马卓, 等. 基于统计分析优化的高性能 XACML 策略评估引擎[J]. 通信学报, 2014, 35(8): 206-215
[13] Wang Ya-zhe, Feng Deng-guo, Zhang Li-wu, et al. XACML Policy Evaluation Engine Based on Multi-Level Optimization Technology[J]. Journal of Software, 2011, 22(2): 323-338 (in Chinese)
王雅哲, 冯登国, 张立武, 等. 基于多层次优化技术的 XACML 策略评估引擎[J]. Journal of Software, 2011, 22(2): 323-338
[14] Wang Ya-zhe, Feng Deng-guo. A Conflict and Redundancy Analysis Method for XACML Rules[J]. Chinese Journal of Computers, 2009(3): 516-530 (in Chinese)
王雅哲, 冯登国. 一种 XACML 规则冲突及冗余分析方法[J]. 计算机学报, 2009(3): 516-530
[15] Stepien B, Matwin S, Felty A. An Algorithm for Compression of XACML Access Control Policy Sets by Recursive Subsumption [C]// 2012 Seventh International Conference on Availability, Reliability and Security (ARES), 2012. IEEE, 2012: 161-167

(上接第 147 页)

时延高于本算法(LDBDC)在送达率为 0.94 时的平均时延，且大部分时候高于本算法在送达率为 0.96 时的平均时延，甚至随着 $N$ 的增大会超过本算法在送达率为 0.98 时的时延。主要原因是 GBCA 算法采取信道分配来提高送达率，节点数目越多，信道干扰越大，收敛迭代越多，所以使得数据包的送达率越低，时延越大。而前 3 种算法都是将监测区域网格化，每个单元格选择一个簇头，这使得簇头节点的分布更加均匀，相对位置变动不大。

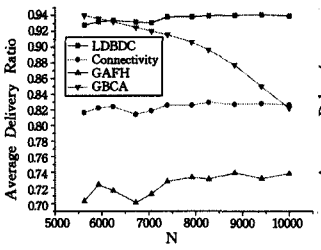


图 9 不同算法的平均送达率比较数据

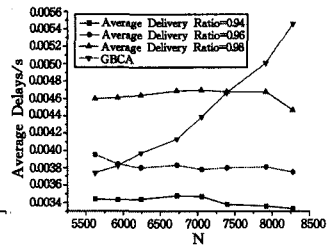


图 10 不同算法的平均时延比较数据

**结束语** 本文针对地震、火灾救援这种特殊的应用场景提出了一种满足送达率约束的低时延的拓扑控制算法，该算法首先根据网络的送达率要求将区域网格化；然后提出了简单高效的簇头选举机制和相应的路由选择算法，降低了算法的时间复杂度和报文复杂度。数学理论推导和仿真实验分析证明，本算法的实验数据和理论数据的误差很小，能够在满足送达率的约束下近似最小化网络的平均时延。

## 参 考 文 献

[1] Geethapriya S, Jawahar A. Performance evaluation of hybrid topology control in WSN[C]// IEEE International Conference on Communications and Signal Processing (ICCSP). 2013: 9-13
[2] Zhao J, Govindan R. Understanding packet delivery performance in dense wireless sensor networks[C]// Proceedings of the 1st ACM International Conference on Embedded Networked Sensor Systems. 2003: 1-13
[3] Xu H, Huang L, Liu W, et al. Topology control for delay-constraint data collection in wireless sensor networks[J]. Computer Communications, 2009, 32(17): 1820-1828
[4] Toscano E, Lo Bello L. A topology management protocol with bounded delay for wireless sensor networks[C]// IEEE International Conference on Emerging Technologies and Factory Automation (ETFA). 2008: 942-951
[5] Ammari H M, Das S K. A trade-off between energy and delay in data dissemination for wireless sensor networks using transmission range slicing[J]. Computer Communications, 2008, 31(9): 1687-1704
[6] Tsirigos A, Haas Z J. Analysis of multipath Routing-Part I: the effect on the packet delivery ratio[J]. IEEE Transactions on Wireless Communications, 2004, 3(1): 138-146
[7] Qin L, Kunz T. Increasing packet delivery ratio in DSR by link prediction[C]// Proceedings of the 36th Annual Hawaii International Conference on System Sciences. 2003
[8] Ma X, Yin X, Butron G, et al. Packet Delivery Ratio in k-Dimensional Broadcast Ad Hoc Networks[J]. IEEE Communications Letters, 2013, 17(12): 2252-2255
[9] He L. Delay-minimum energy-aware routing protocol (DERP) for wireless sensor networks[C]// Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/ Distributed Computing. 2007, 3: 155-160
[10] Yu Q, Chen J, Fan Y, et al. Multi-channel assignment in wireless sensor networks: A game theoretic approach[C]// IEEE INFOCOM. 2010: 1-9
[11] Wang X, Liao X, Huang H, et al. Topology control in lossy wireless sensor networks with delay constraint[C]// IEEE Wireless Communications and Networking Conference (WCNC). 2013: 958-963
[12] Eslamlu G B, Sabaei M, Fereydooni M. A new delay constraint topology control algorithm in WSN[C]// IEEE International Conference on Innovations in Information Technology (IIT). 2012: 189-193
[13] Santi P. Silence is golden with high probability: Maintaining a connected backbone in wireless sensor networks[M]// 1st European Workshop on Wireless Sensor Networks. 2004: 106-121
[14] Li Chong. Research on Topology Control and Routing Protocol Based on Honeycomb Mesh in Wireless Sensor Network[D]. Shenyang: Northeastern University, 2011 (in Chinese)
李充. 基于蜂窝网格的无线传感器拓扑控制及路由协议研究[D]. 沈阳: 东北大学, 2011
[15] Kröller A, Fekete S P, Pfisterer D, et al. Deterministic boundary recognition and topology extraction for large sensor networks [C]// Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm. 2006: 1000-1009