doi:10.3969/j.issn.1000-436x.2014.08.025

基于统计分析优化的高性能 XACML 策略评估引擎

牛德华，马建峰，马卓，李辰楠，王蕾
(西安电子科技大学 计算机学院，陕西 西安 710071)

摘 要：为提高分布式环境下 XACML 策略评估引擎的效率，提出了新的 XACML 策略评估引擎 HPEngine。该引擎利用基于统计分析的策略优化机制动态精化策略，并将精化的策略由文本形式转化为数值形式；同时采用基于统计分析的多级缓存机制存储频繁调用的请求结果对、属性和策略信息。仿真结果表明，HPEngine 所采用的基于统计分析的多级优化机制缩减了策略规模，降低了引擎和其他功能部件的通信损耗，减少了匹配运算量，提高了匹配速度，整体评估性能优于其他同类系统。

关键词：可扩展的访问控制标记语言；策略评估引擎；统计分析；策略优化
中图分类号：TP393 **文献标识码**：A **文章编号**：1000-436X(2014)08-0206-10

HPEngine: high performance XACML policy evaluation engine based on statistical analysis

NIU De-hua, MA Jian-feng, MA Zhuo, LI Chen-nan, WANG Lei
(School of Computer Science and Technology, Xidian University, Xi'an 710071, China)

Abstract: To improve the efficiency of the XACML(eXtensible access control markup language) policy evaluation engine under distributed environment, a novel XACML policy evaluation engine, HPEngine was proposed. The HPEngine dynamically refined policies based on statistical analysis of the policy optimization mechanism first and transformed text form of policy into numerical afterward. Moreover, the engine adopted the multi-level caching mechanism based on the statistical analysis to store frequently called request-results, attributes and policy information. Emulation results show that multi-level optimization mechanisms based on the statistical analysis applied in HPEngine significantly reduce the size of policies, decrease the communication cost between the engine and other components, lessen the amount of matching operation and improve the speed of matching. Comparative analysis demonstrates that HPEngine is obviously better in performance than other similar systems.

Key words: XACML; policy evaluation engine; statistical analysis; policy optimization

1 引言

可扩展的访问控制标记语言（XACML）[1]是在 2003 年 2 月由 OASIS（organization for the advancement of structured information standards）组织制定的用于决定用户访问请求的通用访问控制策略描述语言。与其他访问控制策略描述语言相比，XACML 具有通用性、可扩展性和功能强大的特点，已成为许多大型企业应用和商业化产品实现安全授权功能的实际标准。XACML 策略由一组保证网络资源不被非法使用的规则组成，当用户申请访问资源时，访问控制模块通过评估访问控制策略对用户进行授权。云计算、社交网络、Web service 等业务需要指定大量的 XACML 策略对资源进行细粒度访问控制。但随着系统在线用户和资源数量的不断增加（如截止到 2012 年 12 月底，新浪微博的用户数量已超过 5 亿，平均每天活跃用户达到 4 620 万），访问控制策略包含的规则数越来越多，结构越来越

收稿日期：2013-03-27；**修回日期**：2013-07-11
基金项目：长江学者和创新团队发展计划基金资助项目(IRT1078)
Foundation Item: The Program for Changjiang Scholars and Innovative Research Team (IRT1078)

复杂，策略评估效率已成为制约系统可用性的关键瓶颈，亟需一种高效的策略评估引擎对海量用户的请求及时做出正确授权。

然而，现有开源商业 XACML 引擎仍然不能解决上述评估效率问题。Sun 公司的 XACML 评估引擎[2]采用遍历匹配方式，在当前具有大规模 XACML 策略的分布式环境中，策略评估效率很低。XACMLLight[3]专注于评估引擎的远程服务调用，并没有对策略匹配过程进行优化。AXESCON XACML[4]提供了策略载入和策略缓存功能，但其仍然按照 XACML 嵌套结构逐层匹配，没有对匹配逻辑优化。Enterprise XACML[5]提供了策略索引功能，在一定程度上缩减了策略检索范围，但其索引结构没有考虑策略中规则目标的匹配优化问题。

近年来，XACML 策略评估引擎的效率问题也引起了学术界的广泛关注[6~14]。Liu 等人[7,8]把字符串 XACML 策略转化为数值化的 XACML 策略，把字符串比较变为数值比较，从而提供策略评估效率，但是没有考虑对策略进行精简操作，冗余策略仍然存在。文献[9]提出使用统计分析方法对策略和规则进行重排序，将频繁调用的策略和规则放到最前面，在一定程度上提高了评估效率，但依然是遍历匹配。文献[10]提出的多层次优化技术明显提高了策略评估效率，但该系统不能保证缓存内容一定是调用最频繁的，其策略匹配仍然是字符串比较，策略评估效率有待提高。

针对上述问题，本文提出基于统计分析优化的高性能 XACML 策略评估引擎 HPEngine。首先利用基于统计分析的策略优化机制动态精化策略，并将精化后的 XACML 策略由文本形式转化为数值形式；然后利用统计分析机制为频繁调用的属性、策略以及请求结果对建立缓存，达到既缩减策略规模又优化匹配方式的目的。

2　预备知识

XACML 是一种基于属性访问控制模型的策略描述语言，由 PolicySet、Policy、Target、Rule 和合并算法组成，采用树状层次嵌套结构定义访问权限，如图 1 所示。

定义 1　策略集(PolicySet)。PolicySet 是 XACML 策略的根元素，可用元组表示 $PS=(id,t,P,PC)$。其中，id 为策略集编号，t 为策略集的 Target 元素，$P=\{p_1,\cdots,p_n\}$由一系列 Policy 构成，PC 是 Policy 的合并算法（combine algorithm）。

```
<PolicySet PolicySetId="1" CombiningAlgId="first-application">
 <Policy PolicyId="1" CombiningAlgId="permit-overrides">
  <Rule RuleId="1" Effect="Deny">
   <Target>
    <Subjects><Subject>student</Subject></Subjects>
    <Resources><Resource>grades/excel</Resource></Resources>
    <Actions><Action>write</Action></Actions>
   </Target>
  </Rule>
  <Rule RuleId="2" Effect="Deny">
   <Target>
    <Subjects><Subject>teacher</Subject></Subjects>
    <Resources><Resource>grades/</Resource></Resources>
    <Actions><Action>read</Action></Actions>
   </Target>
  </Rule>
  <Rule RuleId="3" Effect="Permit">
   <Target>
    <Subjects><Subject>student</Subject><Subject>teacher
</Subject> </Subjects>
    <Resources><Resource>grades/</Resource></Resources>
    <Actions><Action>read</Action><Action>write</Action>
</Actions>
   </Target>
  </Rule>
 </Policy>
 <Policy PolicyId="2" CombiningAlgId="first-application">
  <Rule RuleId="4" Effect="Permit">
   <Target>
    <Subjects><Subject>admin</Subject><Subject>president
</Subject></Subjects>
    <Resources><Resource>grades/</Resource><Resource>grades/
excel</Resource><Resource>wages/excel</Resource><Resource>
wages/ </Resource></Resources>
    <Actions><Action>read</Action><Action>write</Action>
</Actions>
   </Target>
  </Rule>
  <Rule RuleId="5" Effect="Deny">
   <Target>
    <Subjects><Subject>admin</Subject></Subjects>
    <Resources><Resource>wages/excel</Resource></Resources>
    <Actions><Action>write</Action></Actions>
   </Target>
  </Rule>
 </Policy>
 <Policy PolicyId="3" CombiningAlgId="deny-overrides">
  <Rule RuleId="6" Effect="Deny">
   <Target>
    <Subjects><Subject>admin</Subject><Subject>president
</Subject></Subjects>
    <Resources><Resource>wages/</Resource></Resources>
    <Actions><Action>read</Action><Action>write</Action>
</Actions>
   </Target>
  </Rule>
  <Rule RuleId="7" Effect="Permit">
   <Target>
    <Subjects><Subject>president</Subject></Subjects>
    <Resources><Resource>wages/excel</Resource></Resources>
    <Actions><Action>write</Action></Actions>
   </Target>
  </Rule>
 </Policy>
<PolicySet>
```

图 1　XACML 策略示例

定义 2　策略(Policy)。Policy 用来保护资源，可用元组表示 $P=(id,t,R,RC)$。其中，id 为策略编号，t 为策略的 Target 元素，$R=\{r_1,\cdots,r_n\}$由一系列 rule

构成，RC 是 Rule 的合并算法。

定义 3 规则(Rule)。Rule 是最小单位的策略原语，可用元组表示 $R=(id,t,e,c)$。其中，id 为规则编号，t 为规则的 Target 元素，e 是 effect 元素 $e\in\{Permit,Deny\}$，c 是对请求的限制条件。

定义 4 目标(Target)。目标元素由主体属性集($Sub.t$)、资源属性集($Res.t$)、动作属性集($Ac.t$)和环境属性集($En.t$)构成，表示在满足环境属性约束的条件下，允许或拒绝主体属性所有者对资源属性指定的资源执行动作属性指定的具体操作。由于环境属性与具体应用相关，所以下文不再考虑环境属性。

定义 5 合并算法。当 Policy 中的多条 Rule 或 PolicySet 中的多条 Policy 都匹配访问请求时，XACML 利用规则/策略的合并算法计算最终的评估结果。合并算法包括肯定优先(permit-override)、否定优先(deny-override)、首次适用(first-applicable)和唯一适用(only-one-applicable)。

定义 6 访问请求(access request)。用户的访问请求用元组表示为 $req=(s,o,a)$。其中，s 是访问请求者，o 为要访问的资源，a 是对资源执行的操作。

3 HPEngine 引擎平台

本节详细阐述 HPEngine 评估引擎。先给出 HPEngine 引擎平台的体系结构和功能部件组成；然后介绍 HPEngine 评估引擎采用的关键优化技术：基于统计分析的策略优化和基于统计分析的多级缓存机制；最后说明 HPEngine 评估引擎对用户访问请求的完整评估过程。

3.1 HPEngine 引擎平台体系架构

根据 XACML 标准的访问控制架构可知，一个完整的访问控制系统通常包括策略执行点、策略决策点、策略信息点、策略管理点等功能部件。因此，HPEngine 引擎不但实现了策略评估相关的核心功能，而且提供了完整的访问授权支撑平台。如图 2 所示，其主要由审计服务(AS, audit service) 、策略管理服务(PMS, policy managerment service)、策略决策服务(PDS, policy decision service)、策略持久化服务(PPS, policy persistence service)和属性断言服务(AAS, attribute assertion service)功能部件组成。

AS 记录系统的请求、响应、策略集及属性调用信息，为统计分析提供最准确的原始数据，保证缓存内容均为评估引擎频繁调用的，以提高策略的决策速度。

PMS 提供一个集中式的图形化策略管理平台，其主要包括策略的基本操作和基于统计分析的策略优化。策略基本操作包括创建、修改、删除和更新策略。基于统计分析的策略优化包括去除冗余规则、调整规则顺序和策略数值化，用来实现策略匹配的前期优化。

PDS 是 HPEngine 引擎的关键部件，统计分析机制、基于统计分析的多级缓存机制和策略缓存中的两级策略索引都包含在该模块内。上下文处理器将原始的用户访问请求解析为 XACML 请求发送给评估引擎，并将返回的判断结果封装为 XACML 响应；评估引擎利用多级缓存对当前请求进行策略匹配和访问决策；统计分析对审计服务的日志信息进行分析，实时、动态地更新缓存内容，调整规则顺序，

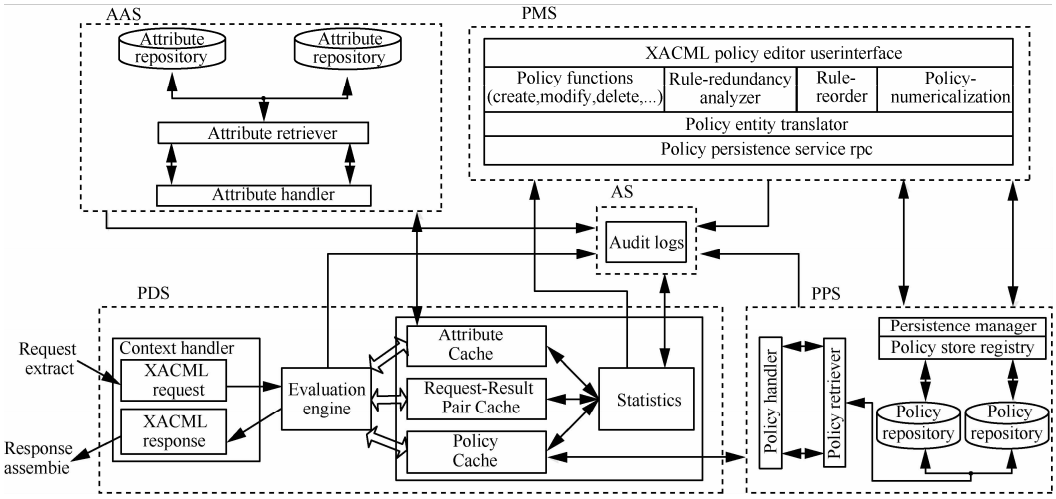

图 2 HPEngine 引擎架构

保证多级缓存机制中存储的内容都是频繁使用的。

PPS 实现策略持久化存储，支持多种策略存储方式及策略库的动态添加和注销；并对 PDS 提供策略检索服务。

AAS 提供属性断言存储和发布服务，为 PDS 的策略匹配提供属性检索功能。属性请求处理器响应来自 PDS 的属性检索请求，并将查找到的属性以 SAML 断言格式返回。

3.2 统计分析

统计模块定期分析审计服务记录的系统运行日志，并根据分析结果更新缓存内容以及调整策略中规则的排列顺序。

统计分析模块主要记录在一定时间内策略、规则的调用频率，属性的调用频率，访问请求的访问频率；单位均为次/周。以图 1 XACML 策略示例为例，策略统计表的格式如表 1 和表 2 所示，其他统计格类似。

表 1　　策略统计分析

策略 *id*	允许率/%	拒绝率/%	调用频率/(次·周)$^{-1}$
1	66	34	47 806
2	89	11	6 312
3	0	0	0

表 2　　规则统计分析

规则 *id*	策略	允许率/%	拒绝率/%	调用频率/(次·周)$^{-1}$
1	1	0.4	1.6	826
2	1	0	0	0
3	1	78	20	46 980
4	2	55	45	6 312
5	2	0	0	0
6	3	0	0	0
7	3	0	0	0

3.3 基于统计分析的策略优化机制

由 XACML 策略评估引擎的匹配方式可知，策略的规则数目和规则顺序决定了匹配运算规模和评估效率。数以千计的策略中可能存在部分对访问请求不产生实际决策影响的规则，策略规则的排列顺序也与引擎的实际运算量直接相关。此外，评估引擎采用的策略匹配算法也影响评估引擎的效率。基于统计分析的策略优化机制，从以上 3 个方面借助统计分析得到的信息对策略库进行优化，达到提高评估引擎效率的目的。

3.3.1 去除冗余规则

本节采用文献[11]提出的状态覆盖法去除策略中的冗余规则。

若规则 R_i 限定主体 *Sub* 在环境 *En* 下可对资源 *Res* 进行访问操作 *Ac*，则属性状态向量(*Sub*, *Res*, *Ac*, *En*)满足规则状态 $State_i$，记为 $State_i \models (Sub, Res, Ac, En)^{effect}$。若

$$(\forall Sub_{i_n}, \exists Sub_{j_m} : Sub_{i_n} \in \bigcup_{m=1}^{m_1} Sub_{j_m} \vee Sub_{j_m} \lhd Sub_{i_n}) \wedge$$

$$(\forall \mathrm{Re}s_{i_n}, \exists \mathrm{Re}s_{j_m} : \mathrm{Re}s_{i_n} \in \bigcup_{m=1}^{m_2} \mathrm{Re}s_{j_m} \vee$$

$$\mathrm{Re}s_{i_n} \lhd \mathrm{Re}s_{j_m}) \wedge (\forall Ac_{i_n} : Ac_{i_n} \in \bigcup_{m=1}^{m_3} Ac_{j_m}) \wedge$$

$$(\forall En_{i_n} : En_{i_n} \in \bigcup_{m=1}^{m_4} En_{j_m})$$

成立，则称 $State_j$ 覆盖 $State_i$，记为 $State_i \prec State_j$。

定义 7　冗余规则。如果 R_i 和 R_j 存在 $State_i \prec State_j$，那么在特定的合并算法下，R_i 可能被 R_j 覆盖对决策结果不产生任何影响，称此类规则为冗余规则。

根据策略合并算法和规则合并算法判断策略间是否存在冗余规则，根据规则合并算法判断策略内部是否存在冗余规则。具体冗余规则判定定理如下。

定理 1[11]　permit-override 算法：R_i·*effect* 为任意类型，R_j·*effect*= *permit* 且 $State_{R_i} \prec State_{R_j}$ 成立；若 $Res_i \lhd Res_j$ 不成立，则 R_i 是冗余规则；否则 R_i 不是冗余规则。

定理 2[11]　deny-override 算法：若 R_i·*effect* 为任意类型，R_j·*effect*=*deny* 和 $State_{R_i} \prec State_{R_j}$ 成立，则 R_i 是冗余规则。

定理 3[11]　规则 R_i 在策略中的位置顺序记为 $seq(R_i)$。在 first-applicable 算法下：$seq(R_i) \prec seq(R_j)$，$State_{R_i} \prec State_{R_j}$ 均成立。1) 若 R_j·*effect*=*deny*，则 R_i 是冗余规则；2) 若 R_j·*effect*=*permit* 和 $Res_i \lhd Res_j$ 不成立，则 R_i 是冗余规则。

利用上述定理对图 1 XACML 策略示例进行冗余分析，可删除冗余规则 *R*2、*R*5、*R*6、*R*7。分析过程及结果如表 3 所示。

3.3.2 动态重排序策略

文献[10]重排序了策略规则，但其将所有 permit 类型规则放到 deny 类型规则的前面，当 deny 类型

表 3　　**规则冗余分析过程及结果**

策略内规则冗余分析		策略间规则冗余分析	
P_1	(R_1,R_2)不存在状态覆盖关系 (R_1,R_3) $State_{R_1} \prec State_{R_3}$，但 R_1 不是冗余 (R_2,R_3) $State_{R_1} \prec State_{R_3}$，$R_2$ 冗余	(P_1,P_2)	(R_1,R_4)不存在状态覆盖关系 (R_3,R_4)不存在状态覆盖关系
P_2	(R_4,R_5) $State_{R_5} \prec State_{R_4}$，$R_5$ 冗余	(P_1,P_3)	(R_1,R_6)不存在状态覆盖关系 (R_3,R_6)不存在状态覆盖关系
P_3	(R_6,R_7) $State_{R_7} \prec State_{R_6}$，$R_7$ 冗余	(P_2,P_3)	(R_4,R_6) $State_{R_6} \prec State_{R_4}$，$R_7$ 冗余

的规则经常调用，而 permit 类型的规则很少调用时，系统性能很低；此外，文献[10]没有对 permit 类型和 deny 类型的规则按调用频率由高到低的顺序进行内部排序，引擎匹配运算量仍然很大。本文根据统计分析得到的策略和规则的调用频率按由高到低的顺序重排序去除冗余后的策略和规则，让调用最频繁的策略和规则排在最前面，从而减少引擎的匹配运算量。

规则 1　合并算法是 first-applicable 时，根据算法逻辑不能调整策略和规则的排列顺序。PolicySet 的合并算法是 first-applicable 时，不可调整策略间的排列顺序；Policy 的合并算法是 first-applicable 时，不可调整策略内规则间的排列顺序。

规则 2　其他合并算法，利用统计分析结果重排序去除冗余规则后的策略文件中策略和规则的位置。

规则 3　不同策略按策略调用频率由高到低排序，同一策略中不同规则按规则调用频率由高到低排序。

利用上述规则和表 1 统计分析的结果，对去除冗余后的图 1 所示的 XACML 策略示例进行动态重排序。排序结果如图 3 所示。由于 PolicySet 合并算法为 first-applicable，所以策略间不能调整顺序；对于策略 1，由于规则 3 的调用频率高于规则 1，所以二者调整顺序。

3.3.3　精化策略数值化

文本的 XACML 策略文件中每个属性的制约因素都用 ASCII 字符串表示，策略评估引擎为用户访问请求进行决策时，需要使用低效的字符串匹配算法，不适合用户访问非常频繁的分布式系统（如云计算、社交网络等）。文献[7]虽然采用了策略数值化机制，但不是对精化后的策略数值化，冗余规则仍然存在，数值化运算量大。本文对去除冗余规则和动态重排序规则处理后的精化策略进行数值化，减少了数值化的运算量。

本文分别为主体、资源、操作属性建立属性值索引表，为 XACML 策略中出现的每个属性都映射一个唯一的整数，将文本的 XACML 策略转化为数值的 XACML 策略，使评估引擎使用高效的整数比较，而不是低效的字符串匹配算法。

图 3 所示 XACML 策略示例索引如表 4 所示，数值化得到的策略文件如图 4 所示。

```
<PolicySet PolicySetId="1" CombiningAlgId="first-application">
 <Policy PolicyId="1" CombiningAlgId="permit-overrides">
  <Rule RuleId="3" Effect="Permit">
   <Target>
    <Subjects><Subject>student</Subject><Subject>teacher
</Subject></Subjects>
    <Resources><Resource>grades/</Resource></Resources>
    <Actions><Action>read</Action><Action>write</Action>
</Actions>
   </Target>
  </Rule>
  <Rule RuleId="1" Effect="Deny">
   <Target>
    <Subjects><Subject>student</Subject></Subjects>
    <Resources><Resource>grades/excel</Resource></Resources>
    <Actions><Action>write</Action></Actions>
   </Target>
  </Rule>
 </Policy>
 <Policy PolicyId="2" CombiningAlgId="first-application">
  <Rule RuleId="4" Effect="Permit">
   <Target>
    <Subjects><Subject>admin</Subject><Subject>president
</Subject></Subjects>
    <Resources><Resource>grades/</Resource><Resource>
grades/excel</Resource><Resource>wages/excel</Resource>
<Resource>wages/ </Resource></Resources>
    <Actions><Action>read</Action><Action>write</Action>
</Actions>
   </Target>
  </Rule>
 </Policy>
<PolicySet>
```

图 3　动态重排序后 XACML 策略示例

表 4　　属性值索引

Subject	Resource	Action
admin:0	grades/:0	read:0
president:1	grades/excel:1	write:1
teacher:2	wages/:2	—
student:3	wages/excel:3	—

```
<PolicySet PolicySetId="1" CombiningAlgId="first-application">
 <Policy PolicyId="1" CombiningAlgId="permit-overrides">
  <Rule RuleId="3" Effect="Permit">
   <Target>
    <Subjects><Subject>3</Subject><Subject>2</Subject>
</Subjects>
    <Resources><Resource>0</Resource></Resources>
    <Actions><Action>0</Action><Action>1</Action></Actions>
   </Target>
  </Rule>
  <Rule RuleId="1" Effect="Deny">
   <Target>
    <Subjects><Subject>3</Subject></Subjects>
    <Resources><Resource>1</Resource></Resources>
    <Actions><Action>1</Action></Actions>
   </Target>
  </Rule>
 </Policy>
 <Policy PolicyId="2" CombiningAlgId="first-application">
  <Rule RuleId="4" Effect="Permit">
   <Target>
   <Subjects><Subject>0</Subject><Subject>1</Subject></Subjects>
    <Resources><Resource>0</Resource><Resource>1</Resource>
     <Resource>3</Resource><Resource>2</Resource></Resources>
    <Actions><Action>0</Action><Action>1</Action></Actions>
   </Target>
  </Rule>
 </Policy>
<PolicySet>
```

图 4　数值化后 XACML 策略示例

3.4　基于统计分析的多级缓存机制

策略评估引擎完成一次请求决策既要从 AAS 服务检索属性信息，又要从 PPS 服务中检索策略信息，然后集成处理；因此，信息检索是造成系统延迟的一个重要原因。有效的缓存机制可以减少系统功能部件间的频繁交互，从而降低检索操作的代价。文献[10]采用了多级缓存机制，但该系统不能保证缓存内容一定是调用最频繁的。

本节介绍的 HPEngine 引擎采用基于统计分析的多级缓存机制将调用最频繁的信息存储在缓存中。基于统计分析的多级缓存机制包括：请求结果对缓存、属性缓存以及策略缓存；策略缓存中采用的两级策略索引技术进一步提高了 XACML 策略匹配效率。

3.4.1　请求结果对缓存

请求结果对缓存是加速评估过程最有效的缓存优化机制，即把用户之前的访问请求结果进行保存，当再次访问时，不必触发属性检索、策略检索及策略匹配等导致系统响应延迟的复杂流程。在访问请求期内用户可能激活多个访问会话，在同一会话内可访问多种资源并有不同的决策结果。因此请求结果对缓存应按照请求标识↔会话标识、会话标识↔决策结果两层映射模式构建，如图 5 所示。请求标识 *req* 由请求者 *s*、资源 *o*、操作 *a* 和请求频率 *f* 组成；每个请求标识对应一个 SessionID 列表，保存该请求激活的所有访问会话；每个 SessionID 对应一个访问列表，保存请求在该会话内访问的具体资源 ResID 和对应的决策结果 Result。为每个请求附加一个访问频率 *f*，统计分析部件会定期比较该请求 *f* 与其他访问请求 *f* 值的大小，保证请求结果对缓存内容是频繁访问的请求及对应决策结果。

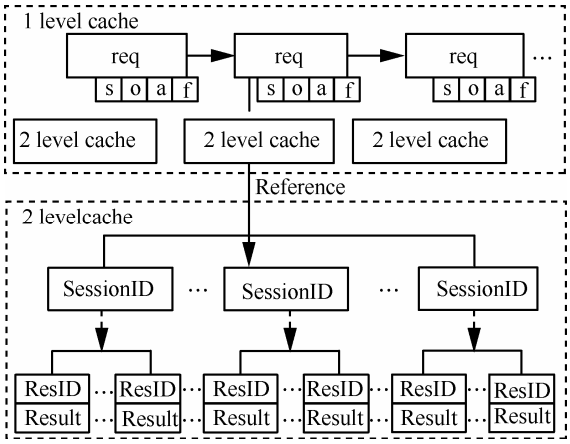

图 5　请求结果对缓存结构示意

3.4.2　属性缓存

属性信息用来辅助评估引擎对请求进行决策，但是频繁的属性检索会导致系统响应延迟，影响评估引擎的效率。为了减少属性检索操作，HPEngine 引擎提供了如图 6 所示的属性缓存机制来提高评估引擎的效率。属性缓存同样采用两层映射模式，第一层映射由主体标识 Identity 和属性标识 AttrName 列表构成，每个 AttrName 对应一个属性值列表 ValueList 构成第二层映射。类似请求结果对缓存，本文为每个 AttrName 附加一个调用频率 *f* 保证属性缓存内容是频繁调用的主体对应的属性值列表。

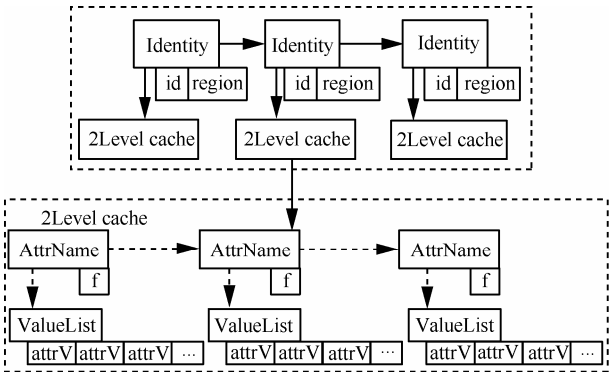

图 6　属性缓存结构

3.4.3　策略缓存

由于缺乏高效策略索引和匹配方式，评估引擎对请求决策时需要遍历策略文件从根节点到叶子节点路径上的所有 Target，导致每个访问请求都要与大量对决策结果没有实际影响的策略和规则进行匹配运算，引擎评估效率非常低。针对上述问题 HPEngine 引擎通过对策略文件组成结构分析，提出使用两级索引技术实现策略缓存机制，提高策略匹配效率。

一个策略文件由若干策略组成，一个策略由若干规则构成，而策略文件用来保护资源信息。因而，本文将资源 res 标识作为第一级索引的主键，每个主键指向一系列用来保护该资源的策略列表 policyList；第二级索引针对策略内多个规则的目标元素，依次从目标元素中提取资源属性和动作属性，并计算二者的笛卡尔乘积 $perSets=(t{\cdot}res)\otimes(t{\cdot}ac)$作为二级索引的主键，主体属性列表作为键值。通过两级策略缓存机制将层次结构的策略映射为扁平结构。两级策略缓存结构如图 7 所示，res 标识由资源 *id* 和访问频率 *f* 组成，*f* 的值由统计部件不断更新，保证频繁访问的资源

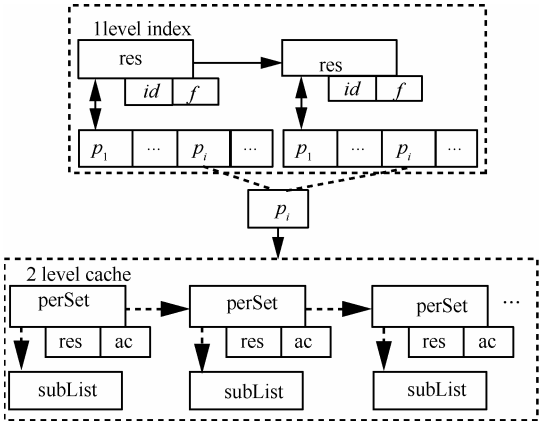

图 7　两级策略缓存结构

都存储在缓存中；$perSet\in perSets$，subList 是 perSet 对应的主体属性列表。图 4 XACML 策略文件对应的两级策略缓存示意如图 8 所示。

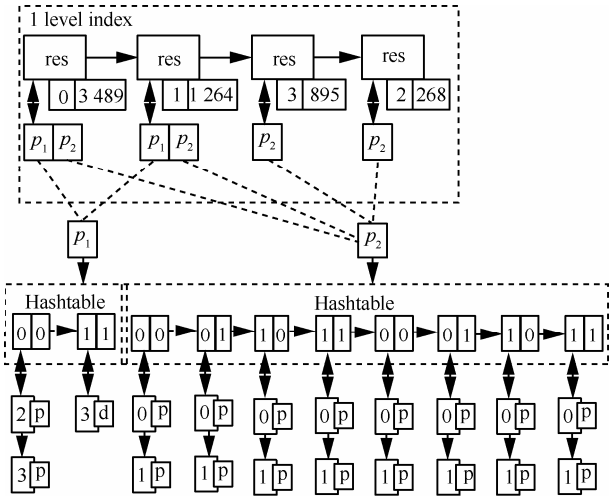

图 8　XACML 策略文件的两级策略缓存

4　技术比较和性能分析

XACML 策略规模庞大和遍历式匹配策略的授权方式是影响 XACML 策略评估引擎效率的两大主要因素，本节先从这两大主要因素的角度分析各种评估引擎技术原理的差异，然后通过多种类型的仿真实验验证 HPEngine 基于统计分析的多级优化机制的有效性及其整体评估性能优势。

4.1　策略加载方式及策略匹配模式比较

通过分析各种开源引擎系统的具体实现可知，评估引擎评估效率的差异主要在于策略加载方式和策略匹配方式。

策略加载方式包括静态加载和动态加载，静态加载是在引擎系统初始化时一次性将所有可用策略加载到内存中；动态加载是评估引擎收到访问请求后从策略仓库中检索适用的策略并加载到内存中。动态加载的优点是内存空间开销小，缺点是需要额外的系统通信开销；静态加载的缺点是内存开销大，优点是匹配速度快。

策略匹配是指请求中包含的实体信息与策略目标匹配或策略内的规则目标匹配，并按照指定的合并算法遍历策略对象的过程。

缓存机制各种评估引擎采用的技术细节比较如表 5 所示。

4.2　仿真实验分析

实验环境：Inter(R) Core(TM) 3.10 GHz CPU、

表 5　　**主流评估引擎技术比较**

评估引擎	策略加载模式	策略存储模式	策略匹配模式	特殊优化机制
Sun XACML	静态	策略列表	两次匹配	无
Enterprise XACML	静态	一级散列表	一次查找	索引结构
XEngine	静态	树结构	一次匹配	数值化、标准化
MLOBEE	静态	两级散列表	3 次查找	规则精简、多级缓存，两级索引
HPEngine	静态	两级散列表	一次匹配	统计分析、规则重排序、数值化、多级缓存、两级索引

4 GB DDR3 内存，操作系统为 Windows XP SP3，JAVA JRE1.6。

仿真实验涉及的评估引擎包括 Sun XACML、Enterprise XACML、XEngine 和 HPEngine。测试用例使用 XACML 官方测试包[15]和文献[16]使用的真实应用系统的 XACML 策略，并根据具体实验场景进行修改和扩充。

4.2.1　基于统计分析的策略优化机制性能分析

为了体现该优化机制的通用性，该实验分别比较 4 种评估引擎对策略优化前后性能的差异。测试用例包含 3 组样本：1）由 1 000 条原始策略组成，其中有 360 个冗余规则，去除冗余规则后，根据规则调用频率需要对规则进行重排序的策略有 450 条；2）由 5 000 条原始策略组成，其中有 2 700 个冗余规则，去除冗余规则后，根据规则调用频率需要对规则进行重排序的策略有 1 500 条；3）由 10 000 条原始策略组成，其中有 5 000 个冗余规则，去除冗余规则后，根据规则调用频率需要对规则进行重排序的策略有 3 000 条。

上述测试用例平均每条策略包含 3 条规则，根据策略中包含的属性信息，分别随机生成 500 次不同的访问请求（请求中已包含完整属性信息），各评估引擎先使用原始策略评估访问请求并计算完成一次响应的平均时间，然后使用去除冗余规则和动态重排序机制精化后的策略对访问请求进行评估，并计算完成一次响应的平均时间；最后使用精化策略数值化机制处理后的策略对数值化后的访问请求进行评估，并计算完成一次响应的平均时间。

由图 9(a)和图 9(b)可知，去除冗余规则和动态重排序策略机制明显提高了引擎的效率，这是由于按规则调用频率由高到低的顺序重排序去除冗余后的规则，减少了策略的匹配运算量，提高了策略的匹配速度。由图 9(b)和图 9(c)可知，精化策略数值化机制明显提高了引擎的效率，这是由于数值化机制将文本的 XACML 策略转化为数值的 XACML 策略，使评估引擎使用高效的整数比较，而不是低效的字符串匹配算法，提高了策略的匹配速度。

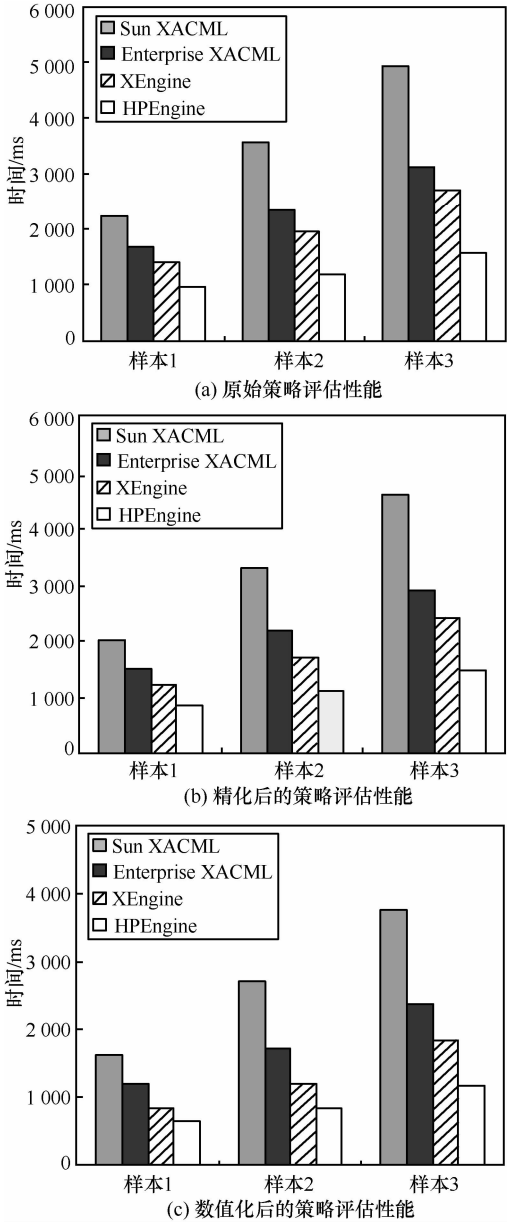

图 9　基于统计分析的策略优化性能比较

综上所述，基于统计分析的策略优化机制明显提高了引擎的效率，各评估引擎间评估性能的差异主要是由策略加载方式和匹配方式不同而导致的。

4.2.2 基于统计分析的多级缓存机制性能分析

缓存机制可以减少引擎与其他功能部件的频繁交互次数，由于篇幅限制，本文只分析请求结果对缓存对评估性能的影响，属性和策略缓存类似。

对相同请求的首次评估各引擎都需要进行属性检索、策略检索和策略匹配操作，但对于以后的重复请求，采用了请求结果缓存类似机制的引擎评估效率会明显提升。

本实验使用 4.2.1 节精化后的 3 个策略样本作为测试用例，分别由 1 000、3 000、5 000 规则组成，对应 3 组请求（包含 100、300、500 个不同请求），每个请求发送 50 次，不考虑每个请求的首次响应时间，计算各评估引擎对每组请求完成一次响应的平均时间。

由图 10 可知，在处理重复请求时，采用了结果缓存机制相关技术的 Enterprise XACML 和 HPEngine 引擎的评估效率显著提高；而 HPEngine 引擎由于采用两级索引机制存储结果缓存，提高了结果缓存检索速度，因而评估效率优于 Enterprise XACML 引擎。

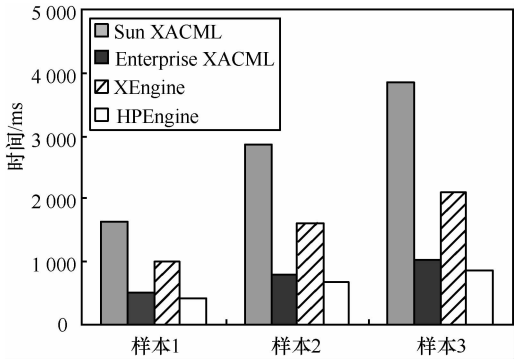

图 10 请求结果对缓存机制的性能比较

4.2.3 引擎综合评估性能分析

为了验证 HPEngine 评估引擎的可用性，本节测试用例样本 1 采用文献[16]使用的实际应用系统中的 XACML 策略；为了验证该系统在拥有大规模策略应用系统中的评估性能，本节对 XACML 官方测试他组进行扩展，得到两组合成策略：样本 2 和 3。其中，样本 2 由 10 000 条规则构成单一策略，并包含 2 000 条冗余规则，根据规则调用的频率需要对 3 000 条规则调整顺序；样本 3 由 4 000 条策略组成，平均每条策略包含 3 条规则，整个样本有 4 000 条冗余规则，根据规则调用的频率需要对 2 000 条规则调整顺序。

为 100 个用户分别构造 30 个不同的访问请求，前 10 个请求随机发送 15 次，中间 10 个请求随机发送 5 次，最后 10 个请求随机发送 1 次；计算各评估引擎在不同测试策略样本下对上述访问请求完成一次响应的平均时间。

由图 11 可知，各评估引擎对样本 1 的评估速率明显快于样本 2 和样本 3，这主要是由于样本 1 使用的策略文件包含的规则数比样本 2 和样本 3 少很多，引擎在策略解析匹配时花费的时间比较少。

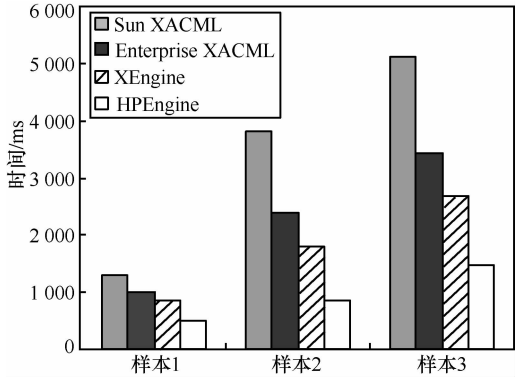

图 11 评估引擎整体性能比较

由图 11 可知，虽然样本 2 和样本 3 的规则总体规模相等，但各种引擎对多规则组合样本的评估速率普遍快于多策略组合样本，这主要由于后者的策略结构比前者的策略结构复杂，引擎在策略解析匹配时需要更多的处理时间。

由图 11 可以看出，HPEngine 引擎无论对实际系统的策略，还是根据测试需要合成的策略都能做出正确的授权决策；其采用的基于统计分析的多级优化机制提高了评估引擎的效率，整体评估性能明显优于其他同类系统。

5 结束语

XACML 策略规模庞大和遍历式匹配策略的授权方式是导致分布式环境下 XACML 策略评估引擎效率低的两大主要因素，本文从这两大主要因素的角度分析了现有开源引擎系统的具体实现过程，提出了新的 XACML 策略评估引擎 HPEngine。该引擎利用基于统计分析的策略优化机制动态精化策略，并将精化的策略由文本的形式转化为数值形式，缩减了策略规模，减少了匹配运算量，提高

了匹配速度；采用基于统计分析的多缓存机制为频繁调用的请求结果对、属性和策略信息建立缓存，有效降低了评估引擎和其他功能部件的通信损耗，提高了匹配速度。仿真结果表明，HPEngine 所采用的基于统计分析的多级优化机制提高了策略评估引擎的评估效率，整体评估性能明显优于其他同类系统。

参考文献：

[1] Brief Introduction to XACML[EB/OL]. https://www.oasis-open.org/committees/download.php/2713/Brief_Introduction_to_XACML.html.

[2] Sun XACML[EB/OL]. http://sunxacml.sourceforge.net/.

[3] XACMLight[EB/OL]. http://sourceforge.net/projects/xacmllight/.

[4] AXESCON XACML[EB/OL]. http://axescon.com/ax2e/.

[5] Enterprise XACML[EB/OL]. http://code.google.com/p/enterprise-java-xacml/.

[6] BUTLER B, JENNINGS B, FAME D B. XACML policy perfor mance evaluation using a flexible load testing framework[A]. Proceedings of the 17th ACM conference on Computer and communications security (CCS)[C]. New York, USA, 2010.978-980.

[7] LIU A X, CHEN F, HWANG J H. Designing fast and scalable XACML policy evaluation engines[J]. IEEE Transactions on Computers, 2011,60(12):1802-1817.

[8] LIU A X, CHEN F, HWANG J H. XEngine: a fast and scalable XACML policy evaluation engine[A]. Proceedings of the 2008 ACM-SIGMETRICS International Conference on Measurement and Modeling of Computer Systems[C]. New York, USA, 2008.265-276.

[9] MAROUF S, SHEHAB M, SQUICCIARINI A. Adaptive reordering and clustering-based framework for efficient XACML policy evaluation[J]. IEEE Transactions on Services Computing, 2011,10(4): 300-313.

[10] 王雅哲, 冯登国, 张立武. 基于多层次优化技术的 XACML 策略评估引擎[J]. 软件学报, 2011,22(2),323-338.
WANG Y Z, FENG D G, ZHANG L W. XACML policy evaluation engine based on multi-level optimization technology[J]. Journal of Software, 2011,2,22(2),323-338.

[11] 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,2009,32(3): 516-530.
WANG Y Z, FENG D G. A conflict and redundancy analysis method for XACML rules[J]. Chinese Journal of Computers, 2009, 32(3): 516-530.

[12] STEPIEN B, MATWIN S, FELTY A. An algorithm for compression of XACML access control policy sets by recursive subsumption[A]. 2012 Seventh International Conference on Availability, Reliability and Security[C]. Ottawa, Canada, 2012.161-167.

[13] PHILIP L, MISELDINE S A, KARLSRUHE. Automated xacml policy reconfiguration for evaluation optimization[A]. Proceedings of the Fourth International Workshop on Software Engineering for Secure Systems[C]. New York, USA, 2008.1-8.

[14] TURKMEN F, CRISPO B. Performance evaluation of XACML PDP implementations[A]. Proceedings of the 2008 ACM Workshop on Secure web services[C]. New York, USA, 2008.37-44.

[15] XACML 2.0 conformance tests[EB/OL]. http://www.oasis-open.org/committees/download.php/14846/xacml2.0-ct-v.0.4.zip.

[16] FISLER K, KRISHNAMURTHI S, MEYEROVICH L. Verification and change impact analysis of access-control policies[A]. Proceedings of the 27th International Conference on Software Engineering[C]. New York, USA, 2005.196-205.

作者简介：

牛德华（1989-），男，山西吕梁人，西安电子科技大学硕士生，主要研究方向为云存储访问控制、云存储安全、移动终端安全等。

马建峰（1963-），男，陕西西安人，博士，西安电子科技大学计算机学院院长、教授、博士生导师，主要研究方向为密码学、计算机网络与信息安全。

马卓（1980-），男，陕西延安人，博士，西安电子科技大学副教授，主要研究方向为无线网络安全、安全协议的形式化分析与设计理论与方法、可信计算理论与技术等。

李辰楠（1988-），男，陕西西安人，西安电子科技大学硕士生，主要研究方向为网络安全、Web 访问控制、云计算安全存储等。

王蕾（1988-），女，陕西西安人，西安电子科技大学硕士生，主要研究方向为网络安全、密钥管理、云计算安全存储等。