Cluster Computing (2021) 24:1505–1524
https://doi.org/10.1007/s10586-020-03204-0

An efficient policy evaluation engine with locomotive algorithm

Fan Deng[1] · Zhenhua Yu[1] · Houbing Song[2] · Rongyi Zhao[3] · Qi Zheng[3] · Zhenyu Li[3] · Huansheng He[3] · Yixin Zhang[3] · Fangzhi Guo[3]

Received: 27 March 2020 / Revised: 20 October 2020 / Accepted: 30 October 2020 / Published online: 26 November 2020

Abstract
The evaluation performance of PDP (policy decision point), especially in large-scale policy sets, is one of the most significant challenges in XACML (eXtensible Access Control Markup Language). With high time-consuming and extensive storage policies, large-scale policy sets are becoming more complicated when their evaluation performance need to be improved. Based on numericalization and batch processing, a new locomotive algorithm is proposed to design and implement a novel policy evaluation engine called XDPNBE that can efficiently deal with large-scale policy sets and make authorization decisions in multiple circumstances. XDPNBE enables efficient decisions within an attributed-based access control framework that has a strong promotion of evaluation performance. By simulating requests, XDPNBE is compared with the Sun PDP, XEngine, HPEngine and SBA-XACML. Experimental results show that if the number of requests reaches 10,000, the evaluation time of XDPNBE on the large-scale policy set with 120,000 rules is approximately 0.21%, 4.69%, 5.67% and 9.66% of that of the Sun PDP, XEngine, HPEngine and SBA-XACML, respectively.

Keywords Evaluation performance · Locomotive algorithm · Policy decision point (PDP) · XACML

1 Introduction

Attribute-based access control is a significant security part in a SOA (Service Oriented Architecture) software system that defines an access control paradigm whereby access rights are granted to users who combine attributes together. Access control is an important security mechanism for the protection of sensitive information and authorization system resources [1, 2]. The operating efficiency of an authorization service is determined by the evaluation performance of PDP (policy decision point) that is a vital component in an access control model. PDP needs to load a policy set composed of a large number of policies, whose evaluation performance will fall into a serious degradation with the scale of a policy set growing larger and larger. This problem leads authorization service systems to a challenging position [3]. A large-scale policy set is a major bottleneck of improving PDP evaluation performance in an authorization system because of its flexible construction, uneasy description and containing massive polices [4].

XACML stands for "eXtensible Access Control Markup Language", which not only suits the specific environment, resource and application system, but also can fit the actual requirements with versatility. XACML is one of the standard implements of attribute-based access control, and it has great adaptability, compatibility and expressive ability [5]. However, XACML cannot effectively detect conflicts in a policy. Deng et al. solve this problem by presenting a conflict detecting and eliminating engine termed XDPCE, which not only can detect and eliminate conflicts within a policy, but also has the same ability as a PDP [6]. In this paper, a creative locomotive algorithm is designed based on XDPCE. In a further way, to meet the demanding evaluation of a web-based information system, the rising of XACML's scale and complexity is becoming more notable.

✉ Zhenhua Yu
zhenhuayu@xust.edu.cn

✉ Houbing Song
songh4@erau.edu

[1] Institute of Systems Security and Control, School of Computer Science and Technology, Xi'an University of Science and Technology, Xi'an 710054, China

[2] Department of Electrical, Computer, Software, and Systems Engineering, Embry-Riddle Aeronautical University, Daytona Beach, FL 32114, USA

[3] School of Computer Science and Technology, Xidian University, Xi'an 710071, China

The major measurements to solve this problem include optimizing a policy set, formulating a policy evaluation engine, and so on. International and domestic academics and reporters have proposed extensive researches, and the findings include formulating decision diagrams, policy reordering, clustering strategies, numericalization, caching mechanism, and so on [7]. Researchers have optimized an XACML policy set from different perspectives, but the related studies available mainly have the following two problems [4, 7].

1. The status quo that requires a lot of storage space to statically preprocess large-scale complicated policy sets does not change.
2. The high time consumption of evaluating requests to dynamically adjust large-scale complicated policy sets is not alleviated.

Ngo et al. [8] present a solution called MIDD (Multi-data-type Interval Decision Diagram) using data interval partition aggregation together with new decision diagram combinations. After that, they improve it by using the data interval partition aggregation that can parse and transform complex logical expressions in policies into decision tree structures [9]. Niu et al. [10] adopt a multi-level caching mechanism based on a statistical analysis to store frequency called request-results, attributes and policy information. Policy recording and clustering strategies are adopted to optimize policy procession in [11]. Marouf et al. [12] describe an adaptive approach for XACML policy optimization by applying a clustering technique to policy sets based on the K-means algorithm. A new approach called "Satisfiability Modulo Theories (SMT)" can easily lend itself to verification at run-time during authorization query answering [13]. Qi et al. [14] optimize an XACML policy by numericalization. Attribute numericalization transforms textuary attributes of XACML policies into numerical attributes. Azzam et al. [15] elaborate on a set-based language that covers all the XACML components and establish an intermediate layer to which policies are automatically converted.

This paper makes the following contributions.

1. A novel policy evaluation engine is presented based on numericalization, batch processing and secondary classification.
2. A creative algorithm called locomotive algorithm is studied that can efficiently improve the PDP evaluation performance especially in large-scale complicated sets.

The rest of the paper is organized as follows. Related works are reviewed in Sect. 2. Section 3 introduces the framework of our proposed policy evaluation engine called XDPNBE. Section 4 outlines our propositions including numericalization, hash coding and a locomotive algorithm. In Sect. 5, a numericalization method of policy sets is described. We present a creative locomotive algorithm combined with XDPCE, secondary classification and batch processing in Sect. 6. Section 7 implements XDPNBE and compares its performance with Sun PDP, XEngine, HPEngine and SBA-XACML, respectively. Finally, we conclude this paper in Sect. 8.

2 Related work

In this section, we review related works addressing PDP evaluation performance, including numericalization, secondary classification and batch processing for XACML policies. Their limitations are discussed to emphasize our contributions.

Qi et al. [14] establish a hash table for all strings in an XACML policy, and map strings to numbers that efficiently convert strings to more easily processed numbers. However, this method consumes a lot of memory space. In this paper, we numeralize different rules according to the characteristics of different attributes of an XACML policy set. For example, attributes are counted and assigned with less changes, hash functions are used for attributes with more changes, and strings are converted into a fixed-length number for processing.

The SBA-XACML [15] reduces the complexity of policy sets and the overhead of real-time policy evaluation. However, when a policy set is extremely large and complicated, conflicts and redundancies will occur. A novel XACML policy evaluation engine, namely XEngine [16], converts all strings in an XACML policy to numerical values and the hierarchical structure of an XACML policy into a flat structure. Niu et al. [10] improve the XEngine and propose a novel XACML policy evaluation engine called HPEngine. Compared to the direct numericalization in the XEngine, the HPEngine dynamically refines policies based on a statistical analysis of a policy optimization mechanism and transforms the text form of a policy into a numerical form afterward. The HPEngine is better in performance than the XEngine, though the problem of "converting all strings into integers by the same function" still exists.

Meng et al. [17] introduce a feature selection based dual-graph sparse non-negative matrix factorization for the local discriminative clustering (DSNMF-LDC). Redundant and irrelevant features are eliminated through dimensionality reduction, and discriminative features are selected by a dual-graph model. The local discriminative clustering is utilized to divide the selected features to several categories. Blockchain technology is exploited to define an access control system that guarantees the auditability of policy evaluation [18]. Policies and attributes are managed by

smart contracts deployed on the blockchain. However, its evaluation performance is limited by choosing the blockchain protocol. Deng et al. [19] explore a novel distributed PDP model based on a combination of two-stage clustering and reordering to reduce the limitation of computational performance of a single PDP.

Methods of numericalization proposed in [20] and [21] only support one-to-one matching methods. For example, resource 1 is represented by integer 1. These methods are too simple to deal with complicated policy sets. JBoss XACML [22] implements a package based on the Sun XACML, but still uses the inefficient strategy matching pattern. AXESCON XACML [23] amplifies the load and cache functions in an evaluation engine, and presents the policy reference and multi-policies matching. Nevertheless, AXESCON XACML needs more improvements on the logic of matching and indexing. In the aspect of indexing, Enterprise XACML [24] is fruitful for its high-efficiency indexing structure, and reduces the scope of policies to be retrieved, which may cause duplicate index and multiple matching.

A locomotive algorithm combined with numericalization, secondary classification and batch processing is designed to avoid the problems above.

3 Policy decision engine XDPNBE

In this section, we present the overall architecture of our approach. A policy evaluation engine called XDPNBE (XiDian Policy Numericalization & Batch processing Engine) is proposed. By loading policies, the XDPNBE can evaluate access requests and return authorization results to context processors and the PEP (Policy Evaluation Point).

The evaluation process of the XDPNBE includes two parts:

1. Policy sets and requests are preprocessed with numericalization;
2. A locomotive algorithm is designed to make authorization decisions by employing secondary classification and batch processing.

Numericalization of policy sets and requests simplifies the matching procedure and is a significant foundation for the locomotive algorithm. When requests arrive, the locomotive algorithm eliminates conflicts in policy sets by using XDPCE [6] as preprocessing, and makes authorization decisions efficiently by using secondary classification and batch processing. The output of the XDPNBE can be correctly generated owing to the locomotive algorithm with massive requests. It can dramatically economize the time of processing large-scale complicated policy sets. Our propositions include numericalization, hash coding and a locomotive algorithm. The structure of the XDPNBE is described in Fig. 1.

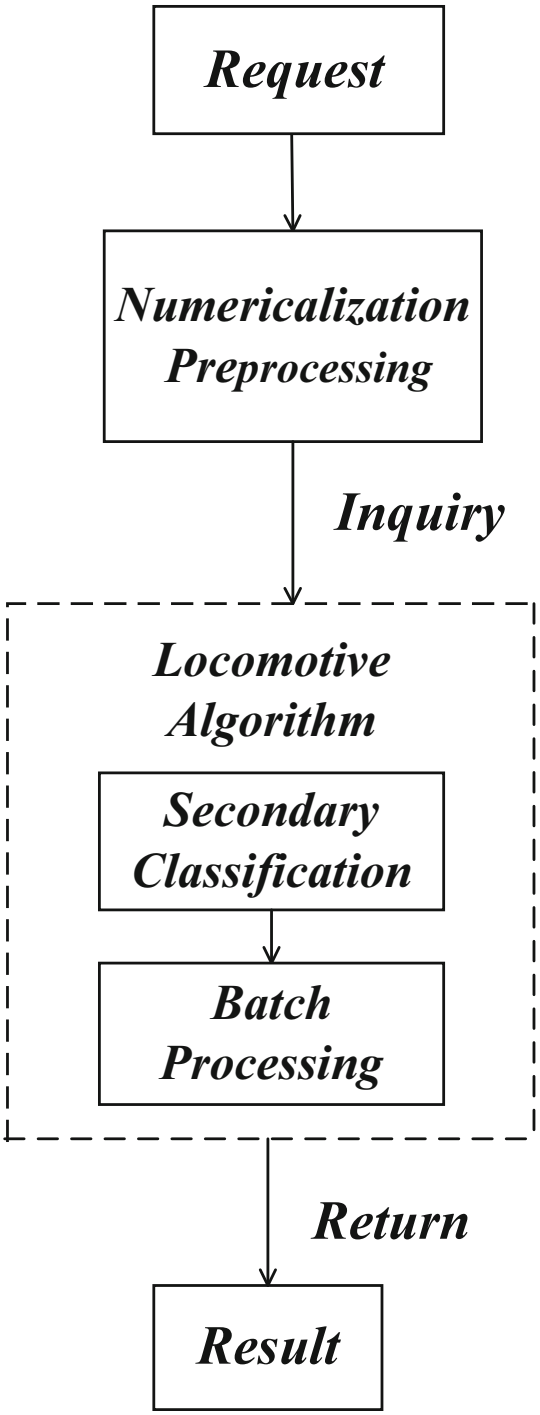


Fig. 1 Structure of XDPNBE

4 Approach overview

Considering that most of attributes in rules are made of character strings, there are two distinct methods to analyze and optimize rules. One of them is dividing long strings into shorter ones to improve analysis performance; the other is a numericalization method of transforming

character strings to numbers. We address attribute numericalization on the basis of hash coding and hash function. Among all the universal hash functions, the BKDRHash function [25] has the most effective consequence in both coding and application, and also has high evaluation performance when character strings are getting longer. Under such a circumstance, subject attributes are labeled simply with integers on the account of its fixed limits of authority. Resource and action attributes in large-scale complicated policy sets are set to numbers using the BKDRHash function.

We also propose a locomotive algorithm that takes every sub table of a secondary classification policy set as a train and each rule as a compartment. Based on the secondary classification and batch processing, this algorithm can successfully optimize rules and policies. In order to improve efficiency by downsizing matching, we classify policies according to their subject attributes and then classify them according to their action attributes. Processing requests in batches makes the locomotive algorithm suitable for concurrent processing.

This paper aims at reforming policy sets and simplifying the policy decision procedure, so that the evaluation performance in large-scale policy sets can be improved. An innovative locomotive algorithm is proposed to implement an effective measurement.

5 Numericalization of policy sets

In [26], the numericalization values of attributes in a policy is restricted by systems. The same attributes in the policy must be successive integers, which not only increases the cost of maintaining a policy, but also may affect the attributes of the subsequent sequence when one attribute needs to be added or deleted. Numericalization has implications and does not consider security issues. In [14], the numericalization method uses hash functions to digitize attributes of subject, action, resource and condition. However, there are many types of hash function with different effects. There is no indication of which hash function to use, and a uniform hash function is used to handle all attributes, without considering that the hash function does not work well when processing attributes with different string length. Similarly, security issues are not considered.

In this paper, a hash function is adopted to process attributes with long string length, and attributes with short string length is numbered. A zipper method is designed to resolve conflicts. When a policy set is preprocessed, a hash table is created for all the string attributes of subject, resource, and action. In the actual situation, the string length of a subject is short, while the string length of a resource and action is relatively long. Therefore, a numericalization method is used for subject attributes, and a hash function is adopted for resource and action attributes to map them to numbers, hence converting inefficient character matching into efficient digital comparisons.

If policies have dynamic natures, we can also use numericalization flexibly. For example, given "age > 21" or "has been with the company for more than 6 months", we can define that "age > 21" is number 1 and "has been with the company for more than 6 months" is number 2. We can present a range by using numbers, giving a concrete analysis to concrete problems.

A batch processing is used to batch requests and then they are handled in batches. The idea of pipelined processor architecture are adopted. After the previous batch of requests is numericalized, it will be matched, and the next batch of requests is numericalized during the matching process of the previous batch of requests, hence an improvement of matching speed of requests in each batch.

5.1 Comparison and selection of numericalization methods

A hash function is used to digitize strings and a zipper method to resolve conflicts. The commonly used string hash functions are BKDRHash, APHash, DJBHash, JSHash, RSHash, SDBMHash, PJWHash, ELFHash, etc. The results of evaluating these hash functions [25] are shown in Table 1 [27].

Table 1 Comparisons of hash function performance

Hash function	Data1	Data2	Data3	Data4
BKDRHash	2	0	4774	481
APHash	2	3	4754	493
DJBHash	2	2	4975	474
JSHash	1	4	4761	506
RSHash	1	0	4861	505
SDBMHash	3	2	4849	504
PJWHash	30	26	4878	513
ELFHash	30	26	4878	513

Hash function	Data1 score	Data2 score	Data3 score	Data4 score
BKDRHash	96.55	100	90.95	82.05
APHash	96.55	88.46	100	51.28
DJBHash	96.55	92.31	0	100
JSHash	100	86.42	96.83	17.95
RSHash	100	100	51.58	20.51
SDBMHash	93.1	92.31	57.01	23.08
PJWHash	0	0	43.89	0
ELFHash	0	0	43.89	0

In Table 1, data 1 is the number of 100,000 random letters and numbers hash collision. Data 2 is the number of 100,000 meaningful English sentence hash conflicts. Data 3 is the number of collisions between the hash value of data 1 and 1,000,003 (a large prime number) stored in the linear table. Data 4 is the number of conflicts stored in the linear table after the hash value of data 1 and 10,000,019 (a larger prime number). After the evaluation, the scores of the four data can be obtained. Quadratic mean is used to compute the average score as shown in Eq. (1).

$$Q_n = \sqrt{\frac{\sum_{i=1}^{n} x_i^2}{n}} = \sqrt{\frac{x_1^2 + x_2^2 + \ldots + x_n^2}{n}} \quad (1)$$

According to the Eq. (1), the average score of the four data are 92.64 for BKDRHash, 86.28 for APHash, 83.43 for DJBHash, 81.94 for JSHash, 75.96 for RSHash, 72.41 for SDBMHash, 21.95 for PJWHash, and 21.95 for ELFHash by means of squared average.

It can be found that the effect of BKDRHash is the most prominent in both the actual effect and the coding implementation. Therefore, the BKDRHash function is selected for numericalization and used to solve the hash value of a string, where a is a string and s is a seed, as shown in Eq. (2).

$$hash(a) = (\sum_n a[n] * s^n) mod 2^{31} \quad (2)$$

We consider the size of a policy set for the test, where seed is 31. The hash function of Eq. (2) is used to create a hash table for resource and action attributes. Subject attributes are numbered from zero. If a different subject attribute comes, its number is added to the previous number. A rule is quantified as $< N$, *HashCode1, HashCode2, environment attribute, 0|1* $>$ form, and a request is quantified as $< N$, *HashCode1, HashCode2, environment attribute* $>$ form.

5.2 Conflict handling

For resolving conflicts, a classic zipper approach is adopted. For some hash addresses that are shared by multiple key values, a single-linked list is established for them. When a conflict occurs, a single-linked list corresponding to its hashed address is searched for to handle the conflict, as shown in Fig. 2.

For example, rules in the three commonly used XACML policy sets LMS [28], VMS [29], and ASMS [30] are quantified. Tables 2, 3, and 4 show the partial results of quantifying rules in the policy sets of LMS, VMS and ASMS.

In Tables 2, 3, and 4, the mapping relationships after the quantification of all the attributes are stored in hash tables, so the time complexity of their digitization and restoration is O(1). Although it makes a string comparison to receive a request attribute map and consumes some time resources, its time loss is negligible compared with a large number of string comparisons in the entire policy set. At the same time, because memory always needs to maintain a hash table that maps a string attribute to an integer value, it consumes memory space resources, so in essence this is a method of trading time for space. However, with the reduction of the cost of computer memory and the strong demand of a system for improving PDP evaluating performance, this method of exchanging space for time is particularly in line with the development of a system.

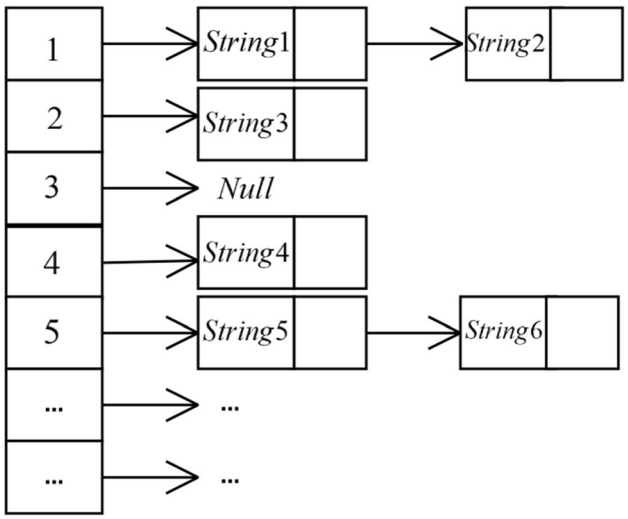


Fig. 2 Zipper method

Table 2 Numericalization results of LMS policy set

Subject	Resource	Action
0	322,933	65,113
4	281,754	978,696
5	738,000	144,155

Table 3 Numericalization results of VMS policy set

Subject	Resource	Action
0	44,643	41,224
1	198,123	940,356
2	738,000	441,544

Table 4 Numericalization results of ASMS policy set

Subject	Resource	Action
0	73,800	537,537
1	0	915,229
2	738,000	537,537

6 Policy decision method based on a locomotive algorithm

On the basis of numericalization, a locomotive algorithm is designed to make policy decisions. The algorithm combines XDPCE, secondary classification and batch processing to optimize PDPs. XDPCE is used as preprocessing to eliminate conflicts. Secondary classification determines the required matching rules by the method of indexing. Considering the reality of large data traffic, we select batch processing to increase throughput by finding multiple requests in batches that helps achieving the goal of improving the average efficiency.

6.1 Locomotive algorithm

In order to improve the speed of making decisions, every sub table of the secondary classification is regarded as a train and each rule as a compartment. The first request for each category matches the locomotive, that is, the subject and action attribute class. A new node is created as a mark to make sure that the same type of passenger (request) belongs to the same train (the sub table) by using the characteristics of the vertical classification (if there is no match in each sub table, it shows that there is no corresponding rule in the policy set). After that, the requests of the same category (in batches) will be retrieved directly from the train, which means that the search area will be greatly reduced from the front to the rear of the vehicle. On average, the time to find the locomotive is equal to the matching time of the subject attribute class plus that of the action attribute class. The total matching process is shown in Fig. 3.

A secondary table is used to store category information after a policy set is secondary classified, and a request table is employed to store the category information after requests are classified. The locomotive algorithm is shown in Algorithm 2.

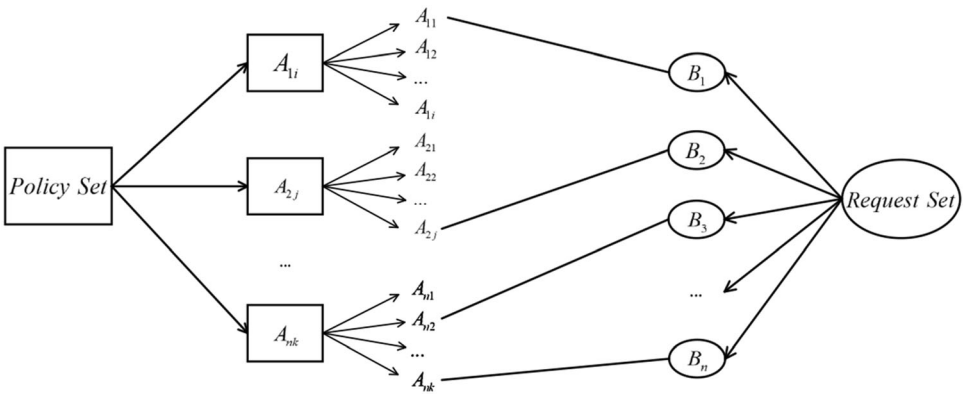


Fig. 3 Matching process of requests based on locomotive algorithm

6.2 Classification

We improve the PDP evaluation performance by matching the irrelevant policies as few as possible to reduce the scope of the required traversal of a policy set in the process of matching requests one by one. Our method is based on the classification of subject attributes and divide a policy set into several subclasses according to subject attributes of each policy. Thus, a request is distributed to one of subclasses, avoiding a traversal of other unrelated subclasses.

6.2.1 Secondary classification based on action attributes

If a matching object is composed of two attributes randomly, its possible number of class is the product of two attributes. But if we determine one of the attributes first and then determine the other one, the number of the object is the sum of the number of two attributes only. Correspondingly, if there are multiple attributes, we first consider the attribute combinations except the first attribute as the second attribute, and then process requests on the second attribute. The common attributes of both policies and requests are subject, action, resource and condition. Among them, action attributes are the most complicated; subject attributes are followed by a fewer number of resource attributes; condition attributes sometimes have no value and the total number is the least. Therefore, policies are classified according to their subject and action attributes in order. It is difficult to merge multiple action attributes into one class because the values of action attributes are not regular and discontinuous. In different rules, subject attributes are often different if the corresponding action attributes are distinct. In order to avoid redundancies, each action attribute is regarded as a classification foundation. To sum up, the result of secondary classification in this paper is that subject and action attributes are both classification bases, and the classification results are shown in Fig. 4.

Algorithm 2. Locomotive Algorithm

Input: A numerical policy set and real-time requests

Output: Authorization decisions for requests

Preprocessing: Using XDPCE to eliminate conflicts in a policy set

```
/* The value in the secondary table is empty before initialization */
for i = 0 to Subject_amount
    first_table[i] = NULL
    for j = 0 to Action_amount
        secondary_table[i][j] = NULL
/* Initialize the secondary table according to the policy set */
i = 0, j = 0, k = temp
/* i and j record the number of subjects and actions, respectively */
/* k is a temporary variable */
for each rule in policy set
    if rule.Subject not in the secondary table
        secondary_table[i] ← rule.Subject
        secondary_table[i][j[i++]++] ← rule.Action
if rule.Action not in the secondary table
    find k ensure secondary_table[k] = rule.Subject
    secondary_table[k][j[k]++] ← rule.Action
/* Classify real-time requests */
/* The number of requests is a fixed value, such as 1000 */
for each request
    if request.Subject & request.Action not in the request table
        request_table ← request.Subject & request.Action
/* Match requests in batches */
for i = 0 to sizeof(request_table)
    find corresponding subject attribute class in secondary table
    traversal rules in the class and make decisions
return authorization decisions
```

6.2.2 Comparisons on classifications basis of action attributes and resources attributes

Suppose that the total number of subject attributes is X and the number of subject attribute classes is x, the total number of action attributes is Y, and the total number of resource attributes and condition attributes is Z. After subject attributes are divided into x classes, the number of subject attributes contained by each subject attribute class is X/x, and the time that matching the specific subject attribute is changed from $O(X)$ to $O(X/x) + O(x)$. Since the product of X/x and x is constant, the difference ($|X/x - x|$) is smaller and the sum ($X/x + x$) is bigger. Therefore, we make $x = X/x$, that is, we suppose that x and X/x are both approximately equal to $\sqrt{X}$. It can also be deduced that the time complexities of matching the subject attribute class and the action attributes are $O(x)$ and $O(Y)$, respectively, the time complexity of specifying subject in the subject attribute class is $O(X/x)$, and the time complexity of matching resource attributes and condition attributes is $O(Z)$. The total matching time is the sum of $O(x)$, $O(Y)$, $O(X/x)$ and $O(Z)$.

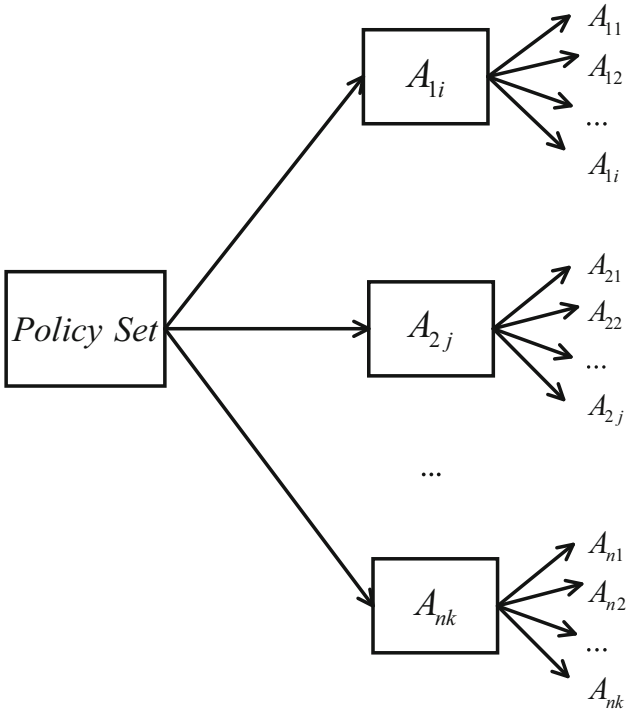


Fig. 4 Secondary classification of decision sets

6.2.3 Efficiency of secondary classification

The main index used in this paper is to classify the rules of policy sets according to the subject attributes. For example, the rules with E0001 $\sim$ E0010 subject attributes belong to the same subject attribute class. When a request is received, it is necessary to identify the subject attribute class, determine the subject attributes, and match it one by one until the same rule is matched. The matching time of subject attribute is T_0. We have

$$T_0 = x + X/x + YZ \tag{3}$$

where x is the time used to determine the subject attribute class of the request, X/x is the time used to determine the specific subject in a subject attribute class, and YZ is the time used to match the remaining attributes (action,

resource and condition). Secondary classification is to match the action attributes after the subject attribute class is fixed, determine the subject in the subject attribute class, and match the resource attributes and the condition attributes. The total secondary classification index time is T_n, as shown in Eq. (4).

$$T_n = x + X/x + Y + Z \tag{4}$$

Equation (4) shows the matching sequence used in secondary classification. After the subject attribute class and the action attributes are successfully matched, the search scope is reduced to the sub table, whose attributes include subject, resource and condition of the corresponding subject attribute class. Compared with the subject index, the time that secondary classification index saves is shown in Eq. (5).

$$T_0 - T_n = YZ - Y - Z \tag{5}$$

6.3 Batch processing for requests

In the optimization of PDPs, it is common to improve the efficiency of making decision by improving the speed of single decision processing. Considering the large network coverage in the information age, people's demand for Internet operation is much higher than before, which leads to many concurrency problems. We envisage that when requests are dense and similar enough, the decision making method can be improved by batch processing, that is, when multiple regular requests arrives concurrently, they can be processed in batch according to their connections. This method bypasses the single processing time optimization, and takes the unit time processing number as the working efficiency reference standard. Assuming that the speed of single processing is the same, if the throughput is increased, the decision time can be optimized. In order to process multiple associated requests, requests need to be categorized.

6.3.1 Criteria for requests classification

In this paper, requests are classified by finding the correlation or similarity of attributes. We have considered the subject and resource attributes as classification bases (using the condition attribute is not significant), but for an actual system, it is difficult to send multiple requests for the same subject in a short time (not only more than one here, but a certain base number). As to resource attributes, they are difficult to repeat as a single resource (not suitable for batch processing), and the base number is small (the batch processing space is very limited). On the contrary, the action attributes are often repeated in a system's request, and the secondary classification of the previous text has

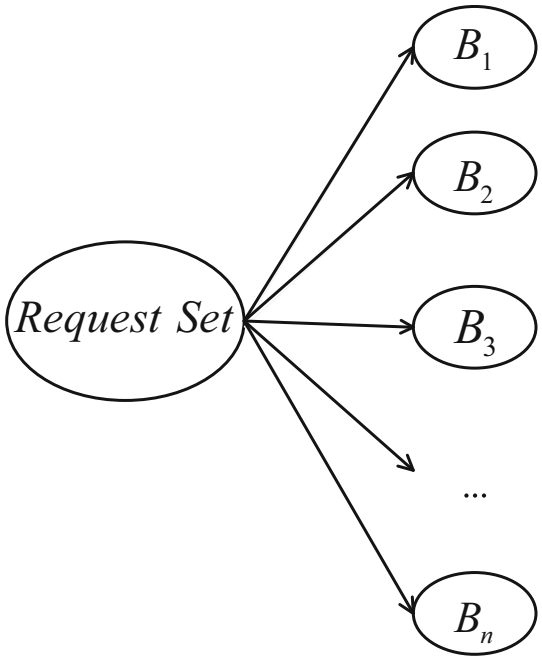


Fig. 5 Vertical classification of requests

classified the policy sets according to action attributes, so action attributes are finally chosen as criteria for the classification of requests and the basis for batch processing. The classification of rules follows the classification method of a policy set. If and only if the subject attributes belong to the same class (avoiding that requests in the same group are divided into different classes of policy sets and speeding up on the basis of the subject attribute index) and the action attributes are completely equal (since the numericalization attribute values are used), two policies can belong to the same class. Using the vertical classification of policy sets, the result of classification is shown in Fig. 5.

6.3.2 Time analysis of related algorithms

We assume that the number of pending requests per unit time is B, and the number of categories after the request classification is b.

6.3.2.1 Subject attribute index The time used to match the subject attribute class under average case is $x/2$, and the time used to specify the subject is $X/(2*x) = x/2$, and the time used to match other attributes is $Y/2*Z/4 = YZ/8$. The total matching time is $x + YZ/8$.

6.3.2.2 Secondary classification based on action attributes The time used to find the locomotive under average case is T_s, as shown in Eq. (6).

$$T_s = Bx/2 + BY/2 \tag{6}$$

The time used to match the specific rules is T_m, as shown in Eq. (7).

$$T_m = B*(X/x)/2 + BZ/4 = Bx/2 + BZ/4 \quad (7)$$

The total matching time is T_1, as shown in Eq. (8).

$$T_1 = BY/2 + Bx + BZ/4 \quad (8)$$

6.3.2.3 Batch processing in request classification In order to show the conditions, advantages and disadvantages of batch processing more directly, we discuss the average time, maximum time and minimum time of its usage. Next, the average time is used to illustrate the influence of the similarity of requests on the batch processing algorithm.

Average case The classification time of requests T_c is $b^2/2$. The time used to find the locomotive T_s is $bx/2 + bY/2$. The index time in the table after the secondary classification T_m is $Bx/2 + BZ/4$. The total matching time T_2-(*average*) is the sum of T_c, T_s and T_m, as shown in Eq. (9).

$$T_2(average) = b^2/2 + b(x + Y)/2 + B(x/2 + Z/4) \quad (9)$$

Worst case ($b = B$) The time used to match the action attributions is $B^2/2$, and other time is the same as the time in method 1. Adding to the matching time of the action attribute, the total matching time is $T_2(worse)$, as shown in Eq. (10).

$$T_2(worse) = B^2/2 + BY/2 + Bx/2 + BZ/4 \quad (10)$$

Best case The best case is the case when $b = 1$. Ignoring the locomotive matching time (there is only one locomotive), $T_2(best)$ is as shown in Eq. (11).

$$T_2(best) = B + Bx + BY/2 + BZ/4 \quad (11)$$

6.3.2.4 Mathematical proof of effectiveness of batch processing In order to test the effectiveness of the batch processing method and find out the applicable conditions, a lemma is introduced and its proof is given.

Lemma 1 *Classify requests using batch processing can enhance the efficiency of policy decision when* $b < (B/b - 1)(Y + x)$.

Proof To analyze the effect of classifying requests using batch processing, let r denote $T_1 - T_2$.

If and only if $r > 0$, batch processing is effective.

Next, we prove that $r > 0$ when $b < (B/b - 1)(Y + x)$.

Since $T_1 = BY/2 + Bx + BZ/4$ and $T_2 = b^2/2 + b(x + Y)/2 + B(x/2 + Z/4)$, we have

$$r = T_1 - T_2 = BY/2 + Bx + BZ/4 - b^2/2 - bx/2 - bY/2 - Bx/2 - BZ/4 = [(B/b - 1)(Y + x) - b]b/2$$

Since $b < (B/b - 1)(Y + x)$, we have.

$(B/b - 1)(Y + x) - b > 0$.

According to the definition of b, we have $b > 0$.

Thus, $r > 0$ when $b < (B/b - 1)(Y + x)$.

Similarly, $b < (B/b - 1)(Y + x)$ when $r > 0$.

If and only if $r > 0$, we see that the batch processing is effective. Among them, Y is the number of action attribute types, B/b is the average number of requests per category.

6.4 Cache and real-time matching

When dealing with requests in batch processing, we inevitably have to face a waiting time problem. Assuming that a group of requests are accepted in one second, a system should wait for one second before processing the first request, and if the time we actually save is less than the waiting time, it shows that the method of batch processing is a failure.

6.4.1 Shortening waiting time and caching instead of waiting

The first request does not need to wait for all subsequent requests within the limited time, but when the request is matched with the new request, the request of the waiting area is read into the policy set and then changed to a new request. This can make the waiting time more averaging and avoid some requests waiting a long time. This method caches the values of subject and action attributes in a system, and the request is directly matched with the cache, without waiting. The significance of batch processing is that it can improve efficiency in the case of high data repeatability and large amount of data. Batch processing should only be used when conditions are met. In real-time matching, each new action class needs to be compared more than once; The cache saves the waiting time and takes up a part of space. From a macro point of view, when the action attributes in requests are repeated a lot, it is suitable for real-time matching. And when the action attributes are not concentrated, it is suitable for cache matching.

6.4.2 Efficiency comparison between caching and real-time matching

The difference between the caching and real-time matching is mainly reflected in the time of matching action attributes. The average time of matching action attributes in caching is $Y^2/2$, and the average time of action attribute matching in real time is $b^2/2$ (This value may have errors due to the distinct type density in the requests). Real-time matching requires the previous request to wait for the next request.

Definition 1 Waiting time T_w..refers to the average interval between two consecutive requests received by a system.

The average time of action attribute matching in real time is $b^2/2 + T_w$. When the condition is known, the matching method with shorter average time is more efficient, and the time difference is T_b is shown in Eq. (12).

$$T_b = Y^2/2 - b^2/2 - T_w \tag{12}$$

When T_b is greater than zero, the real-time matching is adopted, otherwise the caching mechanism (secondary classification) is adopted.

6.5 Summary

In reality, the batch processing method can be suitably used to improve the PDP evaluation performance in practical systems, and can obtain preferable performance (i.e. improving the speed of decision-making to a great extent) in the case that a large number of requests arrive simultaneously, or that the arriving requests have the same subject attributes and action attributes.

7 Experimental results and analysis

In order to verify that the method based on a locomotive algorithm can improve the PDP evaluation performance, we introduce the policies adopted in experiments, the method of generating test policies and the experiments in which comparisons of the evaluation performance of the policy decision engine XDPNBE with that of the Sun PDP [31], XEngine [16], HPEngine [32], and SBA-XACML [15] are made in this section.

The experiments are carried out on a laptop computer running Windows 10, with Intel (R) Core (TM) i7-6700HQ 2.6 GHz processor and 16 GB of RAM. To eliminate the performance factors of implementation languages, the XDPNBE, XEngine, HPEngine and SBA-XACML are implemented in Java since the Sun PDP is written in Java.

7.1 Experimental policies

In order to simulate practical application scenarios, we select four policy sets from practical systems to test. Library Management System (LMS) [28] provides access control policies by which a public library can use Web to manage books. Virtual Meeting System (VMS) [29] provides access control policies by which Web conference services can be managed. Auction Sale Management System (ASMS) [30] provides access control policies by which items can be bought or sold online. The Continue-a policy is taken and converted from [33].

The numbers of rules in the LMS, VMS, ASMS and Continue-a are 3000, 6000, 9000 and 12,000, respectively. We consider these four policy sets as the small-scale policy sets. We expand the numbers of rules in the LMS, VMS, ASMS and Continue-a to 30,000, 60,000, 90,000 and 120,000, respectively. The new obtained policy sets are referred to as the ELMS, EVMS, EASMS and EContinue-a. We consider the new four policy sets as the large-scale policy sets.

7.2 Generation of test requests

In order to improve the coverage of the test, Martin and Xie [34] analyzed policies by Change-Impact to automatically generate access requests conforming to Change-Impact. They put forward that conflicting policies or rules can be obtained by conflict detection tools according to the fact that different policies or different rules in the same policy could make inconsistent results of evaluation for the same request. Besides, correlative access requests can be constructed for testing according to the conflicting policies or rules.

According to Wei et al. [35], access requests can be automatically generated to test the correctness of the PDP as well as the configured policies. They proposed that the Context Shema that is defined by the XML Schema of the XACML describes all the structures of the access requests that might be accepted by the PDP, or all the valid input requests. They put forward that their developed X-CREATE can generate possible structures of access requests according to the Context Shema of the XACML. The policy analyzer obtains possible input values of every attribute from a policy. The policy manager adopts the method for random allocation to distribute the obtained input values into structures of access requests. Simple Combinatorial is another test scheme generating access requests according to all the possible combinations of attribute values of subject, action and resource in the XACML policies.

Inspired by the above, we use composite transformation and context Shema to simulate actual access requests according to the actual requirements of performance testing.

7.3 Policy evaluation engines

Nowadays, the PDP [36] termed Sun PDP [31] has been widely used that can evaluate access requests selectively according to the internal rule matching mechanism [37]. We adopt the Sun PDP to evaluate requests as a decision engine in our following experiments. There are two main

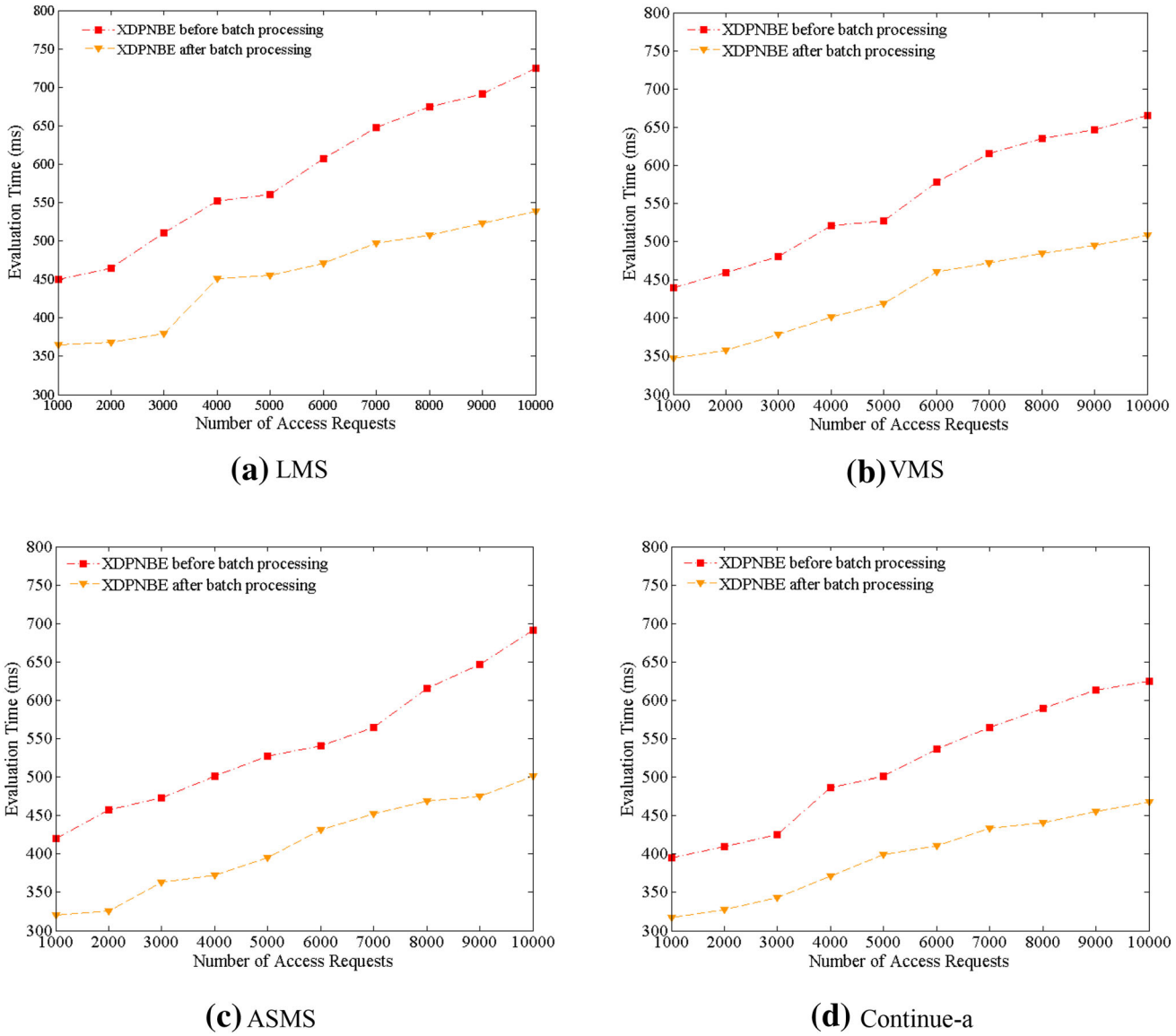


Fig. 6 Variation of evaluation time of XDPNBE before and after batch processing

reasons for our choosing the Sun PDP. One is that the Sun PDP is the first and the most widely deployed implementation of XACML evaluation engines. It has become the industrial standard. The other is that the Sun PDP is open source that provides convenience.

XEngine [16] can convert text XACML policies into numerical policies, transform numerical policies from complex structures to canonical structures, convert numerical policies into tree-type data structures and efficiently process requests.

HPEngine [32] uses a statistical analysis mechanism to create caches for attributes, policies, and request results which are frequently invoked to achieve the goal of reducing the size of the policy and optimizing the matching method.

SBA-XACML [15] is a semantic-based language that establishes an intermediate layer for a policy automatic transformation. It contains formal semantics and algorithms that use mathematical operations to provide an effective policy evaluation.

In practical applications, reducing the time spent in matching policy sets is a principal issue to improve the evaluation performance of PDPs. In our experiments, we compare the evaluation performance of our proposed XDPNBE with that of the Sun PDP, XEngine, HPEngine and SBA-XACML.

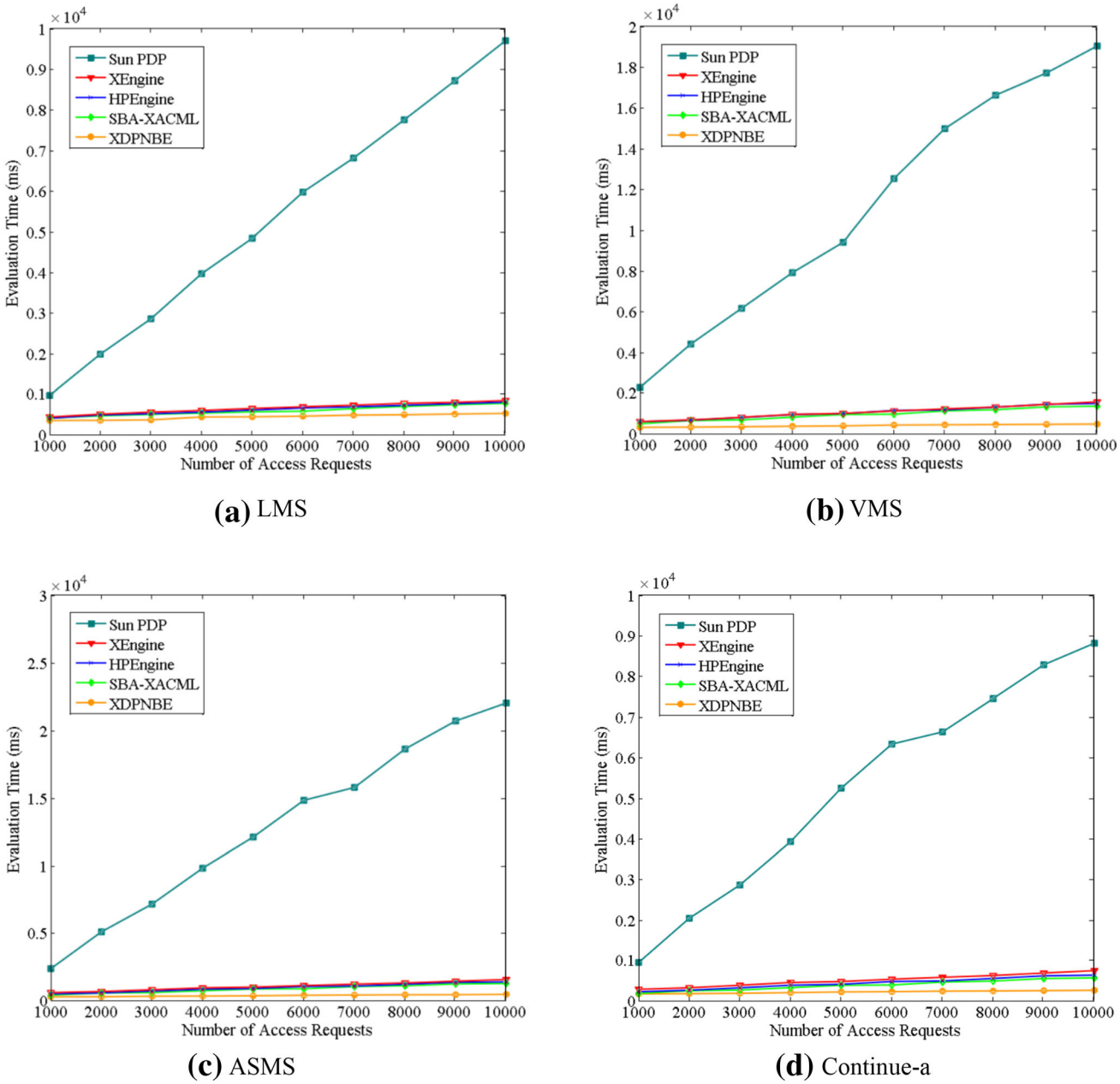


Fig. 7 Comparisons of evaluation time on small-scale policy sets

7.4 Performance tests and comparisons

In order to make the experimental results more convincing, we conduct four experiments as follows.

1. Performance evaluation of XDPNBE before and after batch processing.
2. Comparisons of evaluation performance on the small-scale policy sets.
3. Comparisons of evaluation performance on the large-scale policy sets.
4. Self-comparisons of evaluation performance of XDPNBE on the small-scale and large-scale policy sets.

7.4.1 Performance evaluation of XDPNBE before and after batch processing

In order to highlight the efficiency after batch processing of requests, we evaluate the performance with or without batch processing of requests in the access control process of XDPNBE by contrast. We generate 1000, 2000, …,

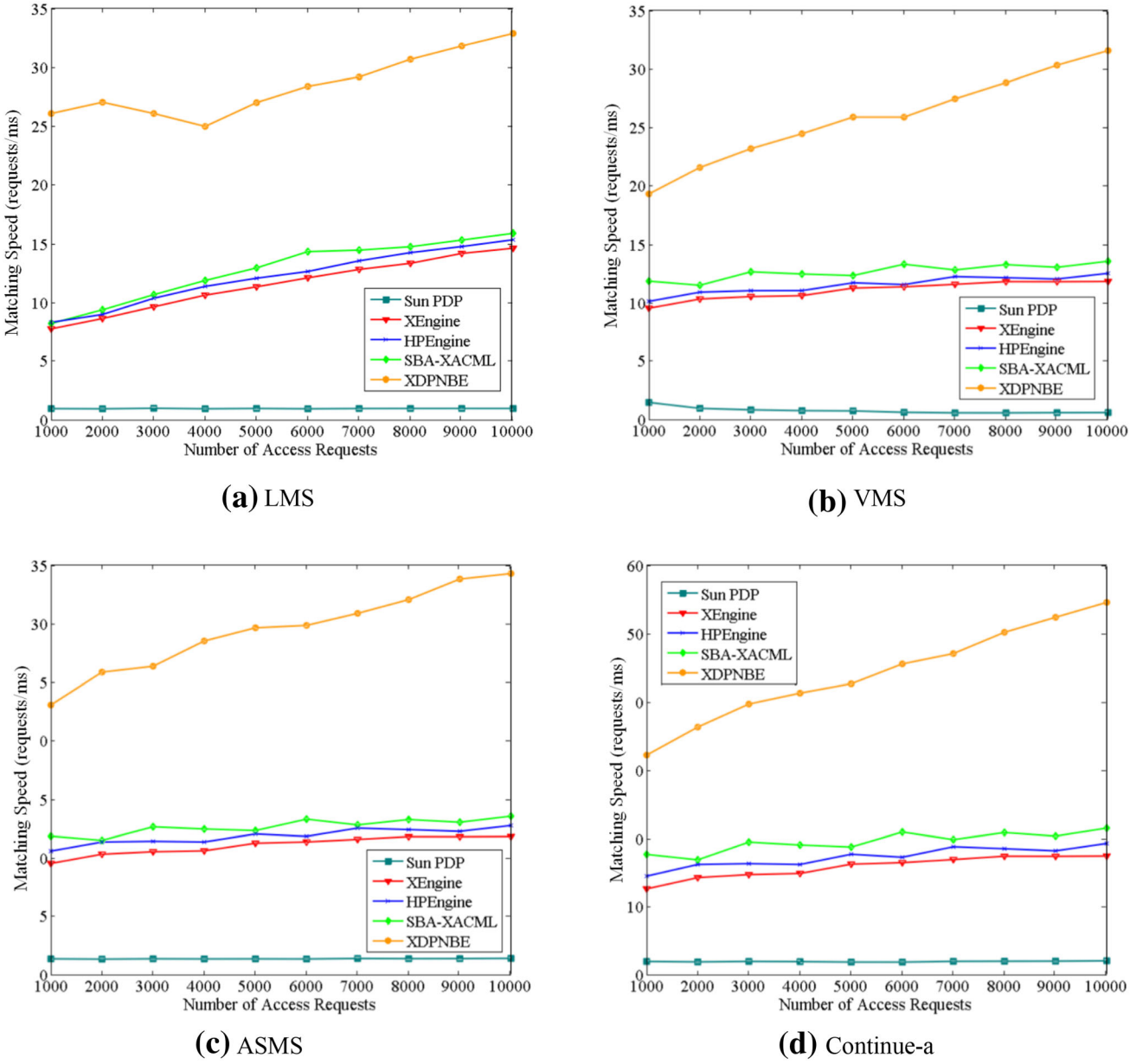


Fig. 8 Comparisons of matching speed for small-scale policy sets

10,000 access requests randomly to measure the evaluation time of the PDP. For the four policy sets of the LMS, VMS, ASMS and Continue-a, the variation of the evaluation time of XDPNBE before and after batch processing with the number of access requests is shown in Fig. 6.

From Fig. 6, we observe that

Whether requests are batch processed or not, the evaluation time of the XDPNBE increases when the number of access requests grows.

The growth rate of XDPNBE after batch processing is less than that of XDPNBE before the requests are batch processed.

For the policy sets of LMS, VMS, ASMS and Continue-a, the XDPNBE with batch processing of requests can effectively reduce evaluation time.

7.4.2 Comparisons of evaluation performance on small-scale policy sets

We compare our proposed XDPNBE with the Sun PDP, XEngine, HPEngine and SBA-XACML in terms of evaluation time for the four small-scale policy sets LMS, VMS, ASMS, and Continue-a, respectively. We randomly generate 1000, 2000, …, 10,000 access requests to record the

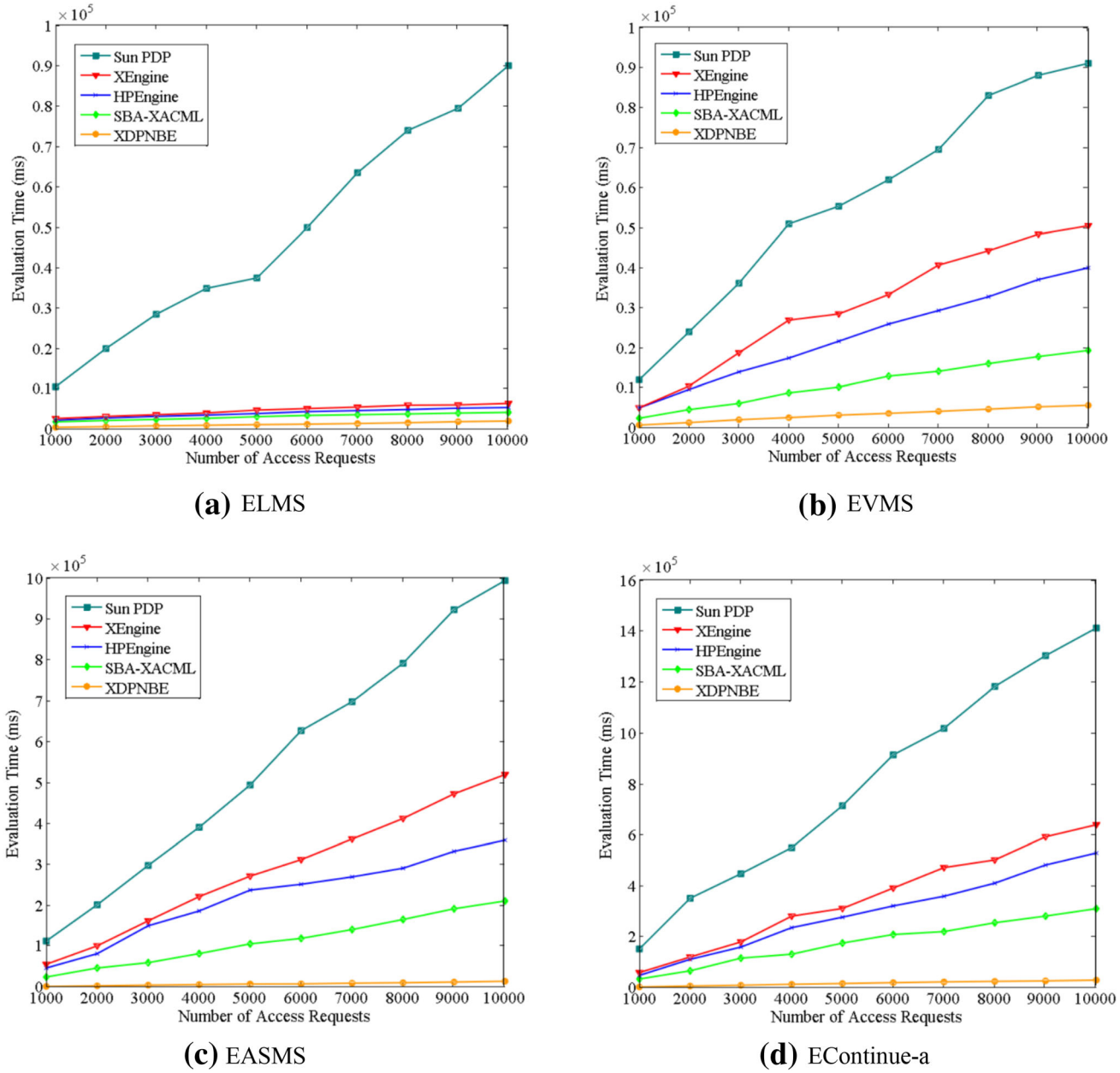


Fig. 9 Comparisons of evaluation time on large-scale policy sets

evaluation time and matching speed of these policy evaluation engines. The variations of the evaluation time of each policy evaluation engine with the amount of requests are shown in Fig. 7.

From Fig. 7, we conclude that

Compared with the Sun PDP, XEngine, HPEngine and SBA-XACML, XDPNBE is the most efficient engine in all experimental conditions.

For the same policy set, as the size of the policy set increases, the growth rate of evaluation time of XDPNBE grows much more slowly than that of the Sun PDP, XEngine, HPEngine and SBA-XACML.

In view of the policy set Continue-a containing 12,000 rules, when the amount of requests reaches to 10,000, the evaluation time of XDPNBE is approximately 3.12%, 27.25%, 31.47% and 33.34% of that of the Sun PDP, XEngine, HPEngine and SBA-XACML, respectively.

For the small-scale policy sets, the variations of the matching speed of each policy evaluation engine with the amount of requests are shown in Fig. 8.

From Fig. 8, we conclude that

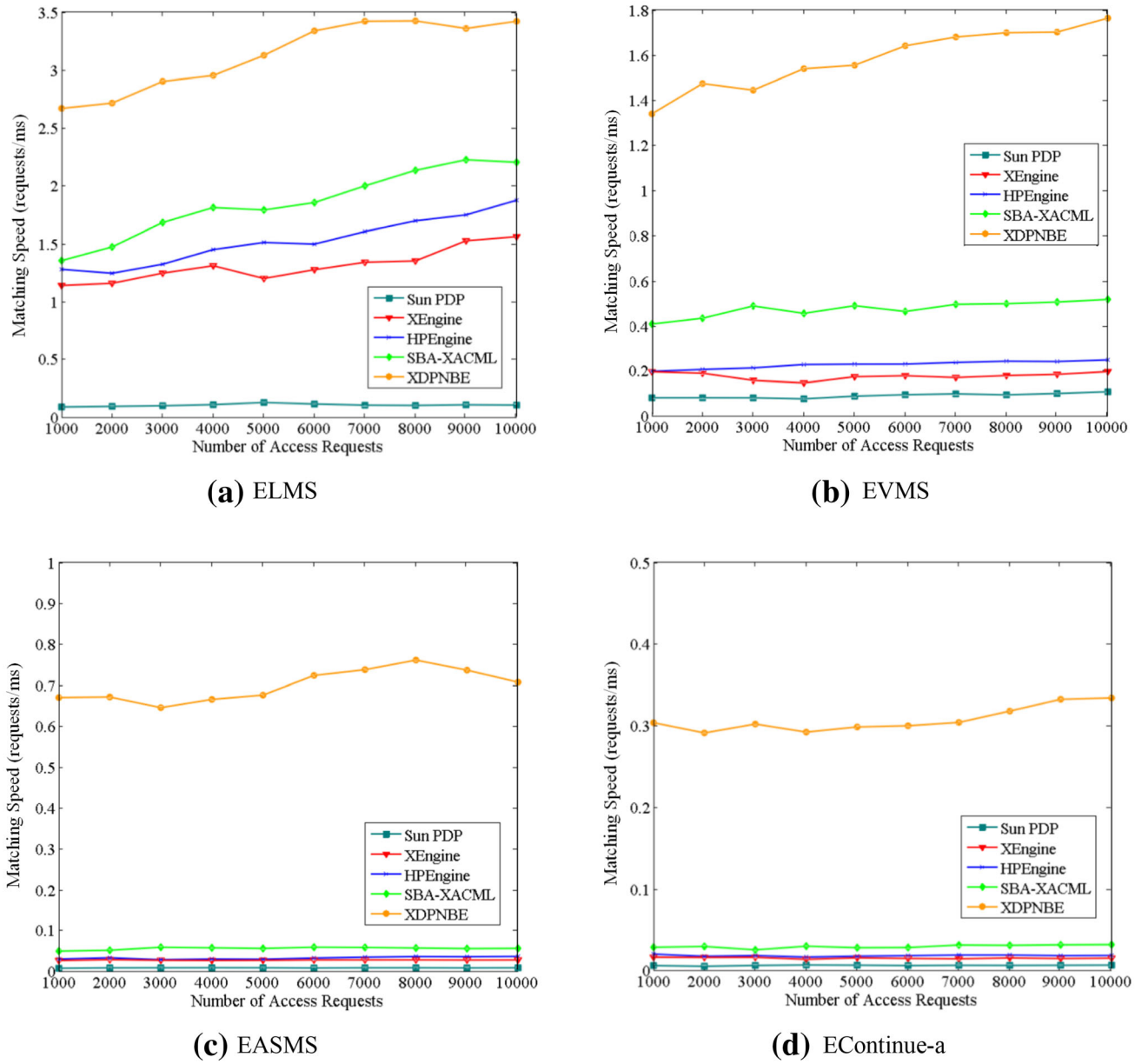


Fig. 10 Comparisons of matching speed for large-scale policy sets

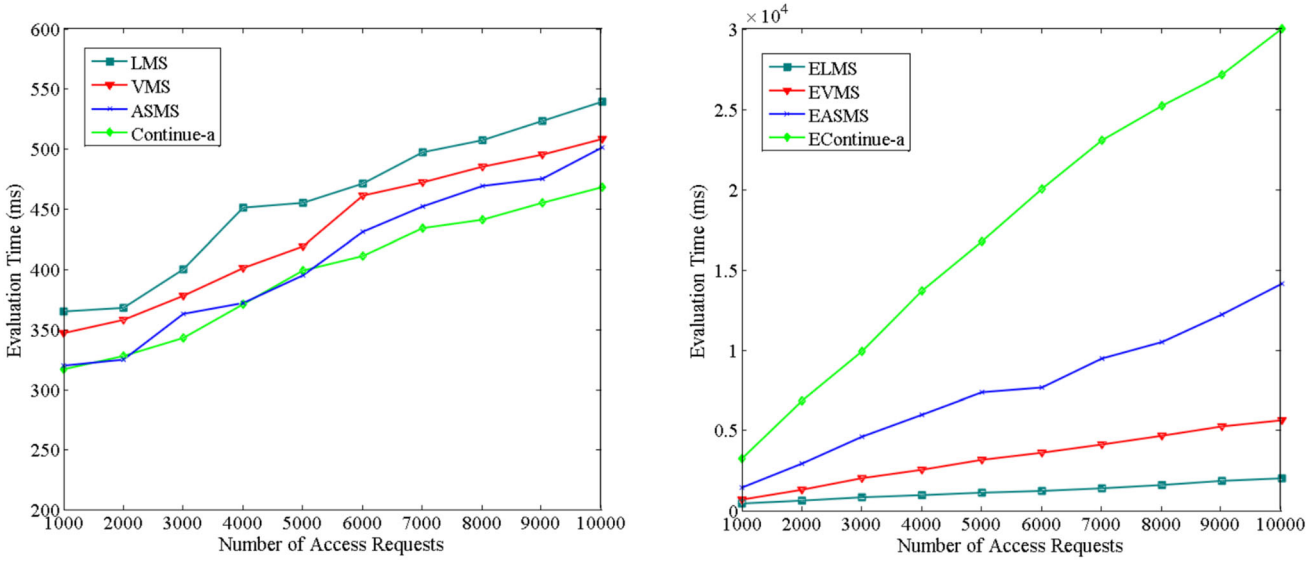
Compared with the Sun PDP, XEngine, HPEngine and SBA-XACML, XDPNBE is the fastest engine in the matching speed of rules.

For the same policy set, with the increase of the number of requests, the matching speed of the XEngine, HPEngine and SBA-XACML does not change significantly, while XDPNBE has a significant improvement on the matching speed of rules.

7.4.3 Comparisons of evaluation performance on large-scale policy sets

We compare the proposed XDPNBE with the Sun PDP, XEngine, HPEngine and SBA-XACML in terms of evaluation time for the four large-scale policy sets ELMS, EVMS, EASMS and EContinue-a, respectively. We randomly generate 1000, 2000, …, 10,000 access requests to record the evaluation time and matching speed of these policy evaluation engines. The variations of the evaluation time of each policy evaluation engine with the amount of requests are depicted in Fig. 9.

From Fig. 9, we conclude that



(a) Small-scale Policy Sets

(b) Large-scale Policy Sets

Fig. 11 Self-comparisons of evaluation time

1. As the amount of requests rises, XDPNBE spends less evaluation time than the other three policy evaluation engines.
2. For the same policy set, as the size of the policy set increases, the growth rate of evaluation time of XDPNBE grows much slower than that of the Sun PDP, XEngine, HPEngine and SBA-XACML.
3. In view of the policy set EContinue-a containing 120,000 rules, if the number of requests reaches to 10,000, the evaluation time of XDPNBE is approximately 0.21%, 4.69%, 5.67% and 9.66% of that of the Sun PDP, XEngine, HPEngine and SBA-XACML, respectively.

For the large-scale policy sets, the variations of the matching speed of each policy evaluation engine with the amount of requests are depicted in Fig. 10.

From Fig. 10, we conclude that

1. As the amount of requests rises, XDPNBE is faster in the matching speed of rules than the other three policy evaluation engines.
2. For the policy set ELMS, with the increase in the number of requests, the matching speed of Sun PDP, XEngine, HPEngine and SBA-XACML is stable and lower than that of XDPNBE, while the matching speed of XDPNBE is gradually decreasing and tends to be stable. In the policy sets EVMS, EASMS and EContinue-a, the matching speed of these four engines is basically stable, and XDPNBE has obvious advantages.

7.4.4 Self-comparisons of evaluation performance of XDPNBE on small-scale and large-scale policy sets

We conduct self-comparison experiments of XDPNBE in terms of evaluation time for both the three small-scale policy sets and three large-scale policy sets. We randomly generate 1000, 2000, …, 10,000 access requests to record the evaluation time and matching speed of XDPNBE. The variations of the evaluation time of XDPNBE with the amount of requests for both the small-scale and large-scale policy sets are visualized in Fig. 11.

From Fig. 11, we conclude that

For the small-scale and large-scale policy sets, the evaluation time of XDPNBE grows approximately linearly as the amount of requests rises.

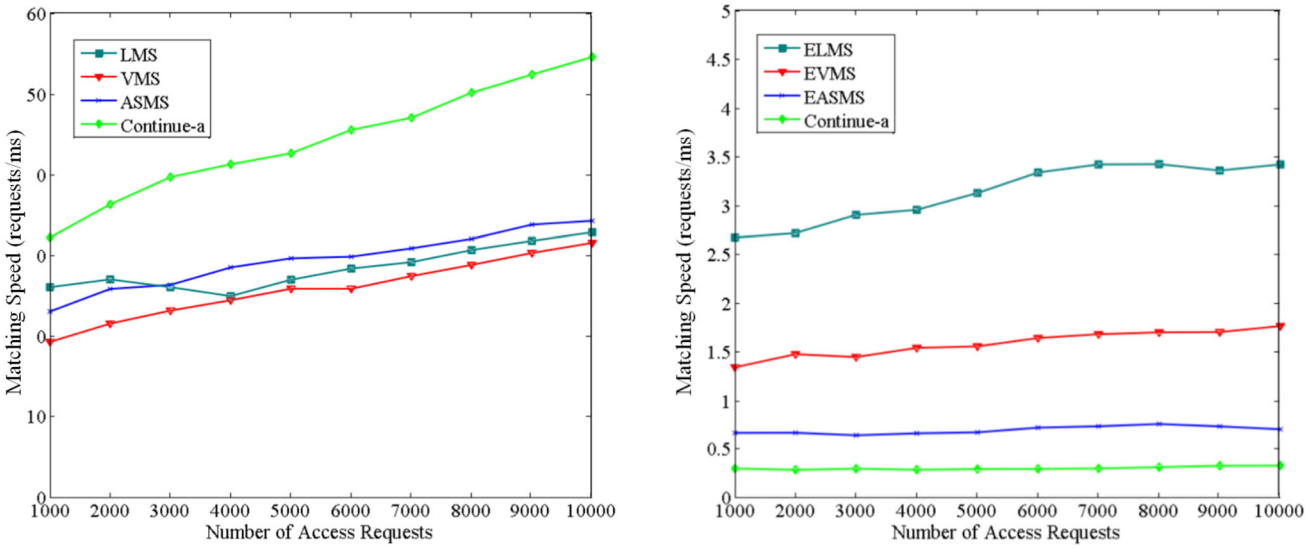
For the three small-scale policy sets, the variations of the evaluation time of XDPNBE are nearly the same as the amount of requests rises.

For the large-scale policy sets, as the size of the policy set rises, the growth rate of evaluation time of XDPNBE gets faster.

The variations of the matching speed of XDPNBE with the amount of requests for both the small-scale and large-scale policy sets are visualized in Fig. 12.

From Fig. 12, we conclude that

1. XDPNBE runs faster in the matching speed of large-scale policy sets than that of small-scale policy sets.
2. For the three small-scale policy sets, the more requests arrive, the faster the matching speed of XDPNBE will



(a) Small-scale Policy Sets

(b) Large-scale Policy Sets

Fig. 12 Self-comparisons of matching speed

be. However, for the large-scale policy sets, the matching speed of XDPNBE gradually tends to be stable. It means that when a large-scale policy set reaches a certain size, the increasing rate of matching speed of XDPNBE reaches a bottleneck and maintains stability.

8 Conclusions

A policy evaluation engine termed XDPNBE is presented in this paper, which not only can improve the PDP evaluation performance in a safer way, but also can be used for concurrent processing. XDPNBE is based on a locomotive algorithm that combines ideas of XDPCE, secondary classification and batch processing.

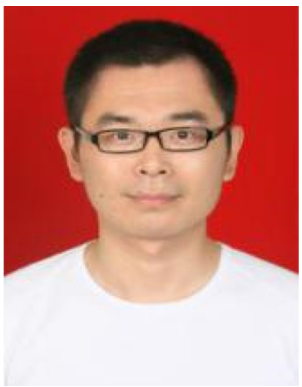
Before making policy decisions based on the locomotive algorithm, a numericalization method is employed to preprocess policy sets and real-time requests. In the numericalization part, we adopt the BKDRHash function to process data with long string length and a zipper method to resolve conflicts. The locomotive algorithm takes every sub table of the secondary classification policy set as a train and each rule as a compartment, which classifies policies according to their subject and action attributes successively, then increases throughput by finding multiple requests in batches and helps to improve the average efficiency especially at concurrency.

Acknowledgements This work was supported in part by the National Natural Science Foundation of China under Grant No. 61873277 and 61702408, in part by the Natural Science Foundation of Shaanxi Province in China under Grants Nos. 2019JM–020, 2019JM–162 and 2020JM–526, in part by the Science Research Plan Project of Education Department of Shaanxi Province under Grant 18JK0507, and in part by the Innovation Group for Interdisciplinary Computing Technologies, School of Computer Science and Technology, Xi'an University of Science and Technology.

References

1. Yaseen, Q., Jararweh, Y., Panda, B., Althebyan, Q.: An insider threat aware access control for cloud relational databases. Clust. Comput. **20**(1), 2669–2685 (2017)
2. Zhu, N.-F., Cai, F.-B., He, J.-S., Zhang, Y.-X., Li, W.-X., Li, Z.: Management of access privileges for dynamic access control. Clust. Comput. **22**(1), 8899–8917 (2019)
3. Habib, M.A., Ahmad, M., Jabbar, S., Khalid, S., Chaudhry, J., Saleem, K., Rodrigues, J., Khalil, M.S.: Security and privacy based access control model for internet of connected vehicles. Fut. Gener. Comput. Syst. **97**(1), 687–696 (2019)
4. Cheminod, M., Durante, L., Seno, L., Valenza, F., Valenzano, A.: A comprehensive approach to the automatic refinement and verification of access control policies. Comput. Security **80**(1), 186–199 (2019)
5. Gadouche, H., Farah, Z., Tari, A.: A correct-by-construction model for attribute-based access control. Clust. Comput. **23**(1), 1517–1528 (2020)
6. Deng, F., Zhang, L.-Y.: Elimination of policy conflict to improve the PDP evaluation performance. J. Netw. Comput. Appl. **80**(4), 45–57 (2017)
7. Butler, B., Jennings, B.: Measurement and prediction of access control policy evaluation performance. IEEE Trans. Netw. Serv. Manag. **12**(4), 526–539 (2015)
8. Ngo, C., Makkes, M.X., Demchenko, Y., Laat, C.D.: Multi-data-types interval decision diagrams for XACML evaluation engine. In: Proc. 11th IEEE Int. Conf. Privacy Security Trust, pp. 257–266 (2013)

9. Ngo, C., Demchenko, Y., Laat, C.D.: Decision diagrams for XACML policy evaluation and management. Comput. Secur. **49**(5), 1–16 (2015)
10. Niu, D.-H., Ma, J.-F., Ma, Z., Li, C.-N., Wang, L.: HPEngine: hign performance XACML policy evaluation engine based on statistical analysis. J. Commun. **35**(8), 206–215 (2017)
11. Sun, P.-J.: XACML policy evaluation optimization research based on attribute weighted clustering and statistics recording. In: Proc. 14th IEEE Int. Conf. Inf Autom., pp. 1190–1195 (2017)
12. Marouf, S., Shehab, M., Squicciarini, A., Sundareswaran, S.: Adaptive reordering and clustering-based framework for efficient XACML policy evaluation. IEEE Trans. Serv. Comput. **4**(4), 303–313 (2011)
13. Turkmen, F., Demchenko, Y.: On the use of SMT solving for XACML policy evaluation. In: Proc. 2016 IEEE Int. Conf. Cloud Computing Technology Science (CloudCom), pp. 539–544 (2017)
14. Qi, Y., Chen, J., Li, Q.-M.: XACML policy evaluation optimization method based on redundancy elimination and attribute numericalization. Comput. Sci. **43**(2), 163–168 (2016)
15. Mourad, A., Jebbaoui, H.: SBA-XACML: set-based approach providing efficient policy decision process for accessing web services. Expert. Syst. Appl. **42**(1), 165–178 (2015)
16. Liu, X., Chen, F., Hwang, J.H., Xie, T.: Designing fast and scalable XACML policy evaluation engines. IEEE Trans. Comput. **60**(12), 1802–1817 (2011)
17. Meng, Y., Shang, R., Jiao, L., Zhang, W., Yuan, Y., Yang, S.: Feature selection based dual-graph sparse non-negative matrix factorization for local discriminative clustering. Neurocomputing **290**(1), 87–99 (2018)
18. Francesco Maesa, D.D., Mori, P., Ricci, L.: A blockchain based approach for the definition of auditable Access Control systems. Comput. Secur. **84**(1), 93-119‘ (2019)
19. Deng, F., Lu, J., Wang, S.Y., Pan, J., Zhang, L.Y.: A distributed PDP model based on spectral clustering for improving evaluation performance. World Wide Web **22**(1), 1555–1576 (2019)
20. Deng, F., Zhang, L.-Y., Zhang, C., Ban, H., Wan, C., Shi, M., Chen, C., Zhang, E.: Establishment of rule dictionary for efficient XACML policy management. Knowl. Based Syst. **175**(1), 26–35 (2019)
21. Ma, X., Kang, K., Lu, W.-S., Xu, L., Chen, C.: Management of access privileges for dynamic access control. Clust. Comput. **22**(1), 12539–12550 (2019)
22. JBoss XACML [Online]. https://jboss.org/jbosssecurity/download/index.html
23. AXESCON XACML [Online]. https://axescon.com/ax2e/
24. Enterprise XACML [Online]. https://code.google.com/p/enterprise-java-xacml/
25. Introduction and performance comparison of classical string hash function [Online]. Available: https://blog.csdn.net/djinglan/article/details/8812934
26. Wang, X., Liao, X., Huang, H., Guo, S.: Topology control in lossy wireless sensor networks with delay constraint. In: Proc. IEEE Wireless Communications Net. Conf., pp. 958–963 (2013).
27. Available: https://blog.csdn.net/djinglan/article/details/8812934
28. Traon, Y.L., Mouelhi, T., Pretschner, A., Baudry, B.: Test-driven assessment of access control in legacy applications. In: Proc. 2008 Int. Conf. Software Testing Verification and Validation, pp. 238–247 (2008).
29. Mouelhi, T., Fleurey, F., Baudry, B., Traon, Y.: A model-based framework for security policy specification, deployment and testing. In: Proc. 11th Int. Conf. Model Driven Engineering Languages and Syst., pp. 537–552 (2008).
30. Mouelhi, T., Traon, Y.L., Baudry, B.: Transforming and selecting functional test cases for security policy testing. In: Proc. 2009 Int. Conf. Software, pp. 171–180 (2009).
31. Sun’s XACML implementation [Online]. https://sunxacml.sourceforge.net/
32. Niu, H., Ma, J.F., Li, C.N., Wang, L.: HPEngine: high performance XACML policy evaluation engine based on statistical analysis. J. Commun. **35**(8), 205–215 (2014)
33. Fisler, K., Krishnamurthi, S., Meyerovich, L.A., Tschantz, M.C.: Verification and change-impact analysis of access-control policies. ICSE **1**(1), 196–205 (2005)
34. Martin, E., Xie, T.: Automated test generation for access control policies via Change-Impact Analysis. In: Proc. Int. Workshop. Software Engineering for Secure Syst., pp. 5–6 (2007).
35. Wei, S., Yen, I.L., Bastani, F., Bao, T., Thuraisingham, B.: Role-based integrated access control and data provenance for SOA based net-centric systems. IEEE J. Magazines **9**(6), 940–953 (2016)
36. Kateb, D.E., Mouelhi, T., Traon, Y.L., Hwang, J.H., Xie, T.: Refactoring access control policies for performance improvement. In: Proc. Int. Conf. Performance Engineering, pp. 323–334 (2012).
37. Ramli, D.P.K., Nielson, H.R., Nielson, F.: The logic of XACML. Formal Aspects Compon. Software **7253**(2), 205–222 (2012)



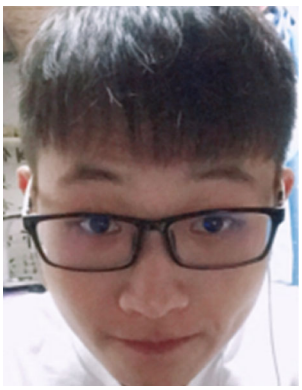
Fan Deng received the B.S., M.S., and Ph.D. degrees from Xidian University, Xi’an, China. He is a lecturer at the School of Computer Science and Technology, Xi’an University of Science and Technology, Xi’an, China. His research interests are information security, software security, and compiler infrastructure.

Zhenhua Yu received the B.S. degree and M.S. degree from Xidian University, Xi’an, China, in 1999 and 2003, respectively, and the Ph.D. degree from Xi’an Jiaotong University, Xi’an, China, in 2006. He is currently a Professor with the Institute of System Security and Control, School of Computer Science and Technology, Xi’an University of Science and Technology, Xi’an, China. He has authored more than 20 technical papers for conferences and journals, and holds two invention patents. His research mainly focuses on cyber-physical systems and system security.

Houbing Song (M'12–SM'14) received the M.S. degree in civil engineering from the University of Texas, El Paso, TX, USA, in 2006, and the Ph.D. degree in electrical engineering from the University of Virginia, Charlottesville, VA, USA, in 2012. In 2017, he joined the Department of Electrical, Computer, Software, and Systems engineering, Embry-Riddle Aeronautical University, Daytona Beach, FL, USA, where he is currently an Assistant Professor and the Director of the Security and Optimization for Networked Globe Laboratory (SONG Lab). He has served on the Faculty of West Virginia University, from 2012 to 2017. In 2007, he was an Engineering Research Associate with the Texas A&M Transportation Institute. He is the editor of six books, including Big Data Analytics for Cyber-Physical Systems: Machine Learning for the Internet of Things (Elsevier, 2019), Smart Cities: Foundations, Principles, and Applications (Hoboken, NJ: Wiley, 2017), Security and Privacy in Cyber-Physical Systems: Foundations, Principles, and Applications (Chichester, U.K.: Wiley-IEEE Press, 2017), Cyber-Physical Systems: Foundations, Principles and Applications (Boston, MA: Academic Press, 2016), and Industrial Internet of Things: Cybermanufacturing Systems (Cham, Switzerland: Springer, 2016). He is the author of more than 100 articles. His research interests include cyber–physical systems, cybersecurity and privacy, the Internet of Things, edge computing, big data analytics, unmanned aircraft systems, connected vehicle, smart and connected health, and wireless communications and networking. Dr. Song is a Senior Member of the ACM. He was a recipient of the Navigation and Surveillance Technologies (ICNS) Conference, the very first recipient of the Golden Bear Scholar Award, the highest campus-wide recognition for research excellence at the West Virginia University Institute of Technology (WVU Tech), in 2016, the prestigious Air Force Research Laboratory's Information Directorate (AFRL/RI) Visiting Faculty Research Fellowship, in 2018, and the Best Paper Award from 2019 Integrated Communication. He has served as an Associate Technical Editor for the IEEE Communications Magazine and a Guest Editor for the IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS (J-SAC), IEEE INTERNET OF THINGS JOURNAL, IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS, and IEEE NETWORK.

Rongyi Zhao is a senior at the School of Computer Science and Technology, Xidian University, Xi'an, China. He won the third class scholarship. His research interests are authentication & authorization and software security.

Qi Zheng is a senior at the School of Computer Science and Technology, Xidian University, Xi'an, China. She won National Encouragement Scholarship, the first class scholarship in college of software, second prize of Mathematical Modeling Contest of Xidian University, second prize of creative group in the second Internet plus innovation and entrepreneurship competition and second prize of the fifteenth HUAWEI cup ACM/ICPC programming competition of Xidian University. Her research interests are information security and software security.



Zhenyu Li is a senior at the School of Computer Science and Technology, Xidian University, Xi'an, China. He won the first and second class scholarship and the third prize of 2018 "Huawei Cup" Programming Competition. He was awarded the outstanding student and outstanding graduate. His research interests are information security and access control.

Huansheng He is a senior at the School of Computer Science and Technology, Xidian University, Xi'an, China. His research interests are information security and access control.

Yixin Zhang is a senior at the School of Computer Science and Technology, Xidian University, Xi'an, China. She won the second and third class scholarship. Her research interests are authentication & authorization and access control.

Fangzhi Guo is a senior at the School of Computer Science and Technology, Xidian University, Xi'an, China. His research interests are access control and authentication & authorization.