Soft Computing (2022) 26:3075–3089
https://doi.org/10.1007/s00500-021-06447-0

APPLICATION OF SOFT COMPUTING

Improvement on PDP Evaluation Performance Based on Neural Networks and SGDK-means Algorithm

Fan Deng[1] · Zhenhua Yu[1] · Houbing Song[2] · Liyong Zhang[3] · Xi Song[3] · Min Zhang[3] · Zhenyu Zhang[3] · Yu Mei[3]

Accepted: 15 October 2021 / Published online: 2 November 2021

Abstract
With the purpose of improving the PDP (policy decision point) evaluation performance, a novel and efficient evaluation engine, namely XDNNEngine, based on neural networks and an SGDK-means (stochastic gradient descent K-means) algorithm is proposed. We divide a policy set into different clusters, distinguish different rules based on their own features and label them for the training of neural networks by using the K-means algorithm and an asynchronous SGDK-means algorithm. Then, we utilize neural networks to search for the applicable rule. A quantitative neural network is introduced to reduce a server's computational cost. By simulating the arrival of requests, XDNNEngine is compared with the Sun PDP, XEngine and SBA-XACML. Experimental results show that 1) if the number of requests reaches 10,000, the evaluation time of XDNNEngine on the large-scale policy set with 10,000 rules is approximately 2.5 ms, and 2) in the same condition as 1), the evaluation time of XDNNEngine is reduced by 98.27%, 90.36% and 84.69%, respectively, over that of the Sun PDP, XEngine and SBA-XACML.

Keywords Access control · Evaluation performance · Neural network · Policy decision point · SGDK-means algorithm

1 Introduction

Access control, when a user performs access operation to a network or information system, refers to the access to distributed resources which the system controls or protects through authentication, dynamic authorization and other technologies. In recent years, as the service-oriented architecture (SOA) (Angulo et al. 2017), Web services, cloud computing and other emerging network application technologies rapidly develop, authorized access control has now become an important protection mechanism in network and information security.

In an access control model, permissions on resources to which users can access are generally described by policies (Atlam et al. 2018, Wang et al. 2016, Han and Lei 2012). Nowadays, one of the best languages for describing policies is the eXtensible Access Control Markup Language (XACML) (OASIS 2007) that as an open standard language based on XML, was approved by the Organization for the Advancement of Structured Information Standards (OASIS) to develop access control for standardizing XML in February 2003.

Policy decision point (PDP) is an important part of an access control model. The PDP stores an access control policy set that is written by XACML. When a new request is received, the PDP checks whether the request is permitted or not. However, as information systems become increasingly complex, the number and complexity of XACML policies grow rapidly (Petersen and Voigtlaender 2018). Existing XACML policy evaluation engines, such as Sun XACML PDP (Sun's XACML implementation 2015),

✉ 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

compare a new request to all policies in the XACML policy set by brute-force search, which leads to a great deal of time to deal with the new request and the poorer performance of PDP. Therefore, deciding how to improve the evaluation performance of policy decision point, in the field of authorization services, becomes an important and critical problem to be solved.

At present, the principal problems in the evaluation performance of PDP include:

(1) The conflict and redundancy of policies in a policy set cause extra comparison times and lower the matching speed.
(2) The matching method of brute-force search affects the matching speed of requests.

As known, many studies have addressed the problem of eliminating conflicts and redundancies in a policy set (Jebbaouia et al. 2015), achieving satisfactory results. Kuang et al. (2018) put forward a modality conflict detection model to recognize the applicable policies during policy evaluation. Ngo et al. (2015) propose an XACML logical model and a decision diagram by the data interval partition aggregation. We have done some research about it in Deng and Zhang (2015); Deng et al. (2016). In addition, Deng et al. (2019a) propose the XACML policy evaluation engine XDPMOE based on bitmap storage and HashMap to improve the evaluation efficiency of the XACML policy. Liu et al. (2008) report XEngine that converts a policy set to a numerical policy and a normalized policy to tree data structures. This paper aims to improve the performance of PDP by changing the matching method.

With the advent of the era of big data, the explosive growth of data has gone beyond the scope of human's abilities to observe and calculate, and thus, a machine learning combining with statistics, database science and computer science has received much attention (Jiang 2018; Zerari et al. 2019). Machine learning is the research of how a computer imitates human's learning behavior, acquires new knowledge or experience, and reorganizes existing knowledge structure to improve its performance. This paper applies machine learning to the matching method of access control.

The contributions of this paper are stated as follows.

(1) The traditional K-means (Peña et al. 1999) algorithm can be used to complete the classification of a policy set. Considering the repeated calculation and the large-scale policy sets, we propose an asynchronous K-means algorithm to accelerate the classification process. That is to say, multi-thread and mini-batch are utile to improve the PDP evaluation performance.
(2) With a view to the particularity of a policy set, distinguishing different clusters is not simple. The proposed SGDK-means (stochastic gradient descent K-means) algorithm makes centroids of different clusters more distinguishable, which can render the following training of neural networks more effective.
(3) Due to the higher classification performance of neural networks, we utilize them to search for the applicable rule for every coming request, which makes it possible to dynamically insert new policies into the policy set without restarting its preprocessing. In addition, a quantitative neural network is proposed to reduce the requirement of a server's computational cost in classifying new requests.

The remainder of this paper is organized as follows. Section 2 reviews the related works and shows the notations used in the paper. The framework of the XDNNEngine is addressed and its two-stage evaluation process is outlined in Sec 3. Section 4 reports the SGDK-means algorithm and elaborates on the training of neural networks. In Sec 5, experimental results on the PDP performance improvement are shown and analyzed. Finally, we conclude this paper in Sec 6.

2 Related work and related notations

In the area of access control, scholars are trying to improving the PDP evaluation performance in a variety of ways. Several efforts have been mainly devoted to three directions, including elimination of conflicts and redundancies, numericalization of policy sets, and clustering and distributed models.

2.1 Elimination of conflicts and redundancies

Because of the derivation of authorizations according to the hierarchical structure policies, redundancies between rules may exist (Lin et al. 2013). This kind of authorization propagation can also result in unexpected conflicts (Singh and Singh 2010). It is useful to find an effective way to detect and reduce the conflicts and redundancies.

A number of works focus on the modality conflict detection (Kamoda et al. 2005, Toe et al. 2013 Mohan et al. 2011). An XACML logic model is presented by Ngo et al. (2015), which achieves a decision diagram using the data interval partition aggregation. By this model, the logic expressions in a policy can be parsed and transformed into a decision tree structure that can improve evaluation performance and detect redundancies between policies. Wang

et al. (2011) prove that redundant policies can be removed without changing the result of evaluation and propose a framework named MLOBEE (multi-level optimization-based evaluation engine). However, they do not formulate a general method of refinement. Moreover, Deng and Zhang (2015) argue that conflicts can be eliminated by constructing a special data structure called resource index tree.

These works perform well in eliminating conflicts and redundancies. Yet it is possible that the scale of policy remains large even after conflicts and redundancies are eliminated. In fact, the evaluation process can be considered as a simple procedure that compares attributes between the coming request and each rule in policies. The prevalent Sun PDP uses brute force to match rules, which is not elegant enough and leaves a great space for progress. Since the comparison with strings is much slower than numbers, with the pursuit of better efficiency, approaches have been studied to improve efficiency in this evaluation process.

2.2 Numericalization of policy sets

The comparison of strings is slower than that of integer. Guided by this idea, approaches that convert rules in a numeric format and organize these rules in structures designed for easily finding applicable rules are developed by more and more researchers.

Alex and Liu (2008) present a fast and scalable XACML policy evaluation engine, called XEngine. Firstly, this engine converts strings into integers. Then policies with a hierarchical structure in the numerical format are organized into a flat structure with the possibility for changing the combining algorithm. Deng et al. 2019a) address a rule dictionary designed for boosting evaluation using a multi-dimensional array to store rules in a numeric format. When a request is coming, it can be converted into a numeric format by checking the map that has already been obtained. Then, the rule dictionary calculates the exact location where the applicable policy should be. Mourad and Jebbaoui (2015) report an SBA-XACML framework by elaborating on a set-based language and create an intermediate layer that can automatically convert policies. A semantics-based policy evaluation method is also introduced in Mourad and Jebbaoui (2015) to accelerate the PDP evaluation process.

However, the evaluation performance of methods mentioned above is not satisfied when a large-scale policy set is evaluated. If rules reach more than 10,000 in a policy set, the evaluation time of XEngine and SBA-XACML cannot be reduced dramatically. Moreover, the rule dictionary requires a great deal of space for the dictionary. Benefited by the previous works aiming for eliminating the conflicts and redundancies between policies and the numericalization of policies, a lot of approaches using clustering and distributed PDP techniques reveal an unexpected performance in this area.

2.3 Clustering and distributed models

Using clustering techniques, we can divide rules into different groups. Accordingly, when a request is coming, we can locate which group contains the applicable rule that we are searching for, then we can easily find the very rule. Since a policy set contains a large number of rules, the searching process can be handled by a distributed system. That is to say, a number of PDPs search for the applicable rule simultaneously to speed up the evaluation.

Marouf and Shehab (2009) propose an adaptive approach for policy optimization using a clustering technique to categorize policies and rules by the subject attribute. They also elaborate on a framework to dynamically reorder policies according to access request statistics. (Liu and Wang 2015) come up with a new evaluation framework based on clustering methods to cut down policy scale, focusing on the processing of policies and rules. There is a two-stage clustering method in Liu and Wang (2015). The first stage is coarse-grained clustering, and the second stage is fine-grained clustering. This method reduces the number of comparisons by assigning the preprocessed policies to evaluate a coming request. Deng et al. (2019b) address a distributed PDP model based on spectral clustering. According to the subject attribute, clustering of policies and rules can divide a policy set into different groups, which can be assigned to different PDPs. Among these PDPs, similar rules are applicable to a certain request, which means that we can calculate the similarity between different rules and divide the policy set into many more fine subsets, and pick centroids of different subsets.

Previous clustering techniques focus on the subject attribute of requests, while there are several evident clusters of subjects in the policy set, which makes it much more effective to cluster policies. In this way, clustering cannot improve the performance substantially compared with brute-force evaluation, while these methods depend on the classification between subjects. Therefore, clustering can be difficult when there is no explicit classification of subjects between policies.

Besides the previously mentioned methods, there are still a number of noticeable works. An optimization technique for policy evaluation using two trees, a matching tree for quick finding of applicable rule and a combining tree for finding the matching tree, is proposed by Pina et al. (Santiago et al. 2012). This work performs well in the evaluation process, but does not address the problem of a frequent insertion of new rules into the original policy set.

2.4 Our novel work

In this paper, an asynchronous K-means cluster algorithm is proposed and neural networks are adopted in the evaluation process to improve the PDP evaluation performance. Euclidean distance is used in the cluster algorithm. We also compare the different performances between an asynchronous K-means algorithm and a synchronous one. Neural networks are also utilized to speed up the PDP evaluation process.

2.5 Related notations

For the convenience of discussion, we show some related notations used in this paper as follows.

Notation 1. Rule and attribute: $x_i = (s_i, r_i, a_i, c_i, e_i)$ represents the ith rule, where s_i, r_i, a_i, c_i, and e_i represent its five attributes of *source*, *resource*, *action*, *condition* and *effect*, respectively.

Notation 2. Parameters of K-means algorithm: K is the total number of clusters. $Center_i$ is the ith centroid of cluster and Center(x_i) represents the Center of x_i. T represents the max times of iteration, and D is the distance between Center$_i$ and x_i. ∇_{x_i}(Cluster) represents the gradient estimator, as shown in Eq. (1).

$$\nabla_{x_i}(\text{Cluster}) = x_i - \text{Center} \tag{1}$$

Notation 3. Terms of neural network: There are different weights in different edges. W_i is introduced to describe the weight vector of the ith layer and b_i is the bias of the ith layer. The parameter of the ith layer is Θ_i, $\Theta_i = [W_i, b_i]$. A_i, Z_i and g represent the activation vector of the ith layer, output vector of the ith layer and the activation function, respectively. a_i represents the ith element in the vector A_i, z_i stands for the ith element in the vector Z_i, Y represents the correct label vector, η is used to denote the learning rate, J is the cost function, and E_i is the error, also known as, false estimation of the ith layer.

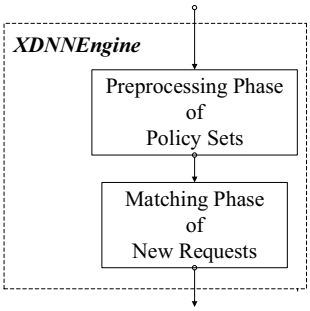

Fig. 1 Framework of XDNNEngine

3 Approach overview

Based on neural networks, we propose a policy evaluation engine to improve the PDP evaluation performance. We call the engine XDNNEngine (XiDian neural networks engine), as shown in Fig. 1. The XDNNEngine has the function of the PDP. By loading policies, the XDNNEngine can evaluate access requests and return the authorization results to context processors and the PEP (policy execution point).

The evaluation process of the XDNNEngine includes two phases:

(1) the preprocessing phase of policy sets;
(2) the matching phase of new requests.

3.1 Preprocessing phase of policy sets

The preprocessing phase of policy sets is shown in Fig. 2, which includes four steps: 1) the numeralization of policy sets, 2) the classification of policies by SGDK-means, 3) the construction of neural networks and 4) the training of neural networks.

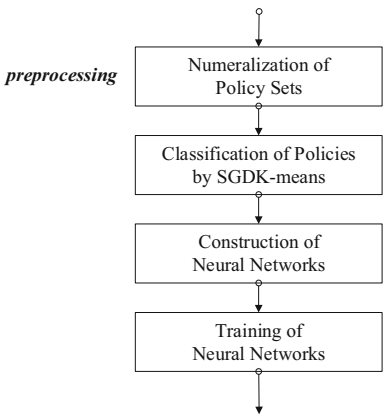

Fig. 2 Preprocessing phase of policy sets

The first and most important step of preprocessing is the numeralization of policy sets. In this step, we extract the attributes of each policy from an input policy set, and then encode the attributes with data and output numerical policies.

The second step of preprocessing is the classification of policies. The K-means clustering algorithm is adopted in this part to cluster the input numerical policies, which labels the policies on the basis of some sorts of criterion.

In the last two steps of preprocessing, we construct and train a neural network classifier. Firstly, we define the structure of the network and initialize its parameters. Then, we extract a part of the policies from the policy set as the training set, which is used to update and adjust the parameters by the calculation. We can train a neural network classifier with high accuracy.

3.2 Matching phase of new requests

This phase includes the numeralization of new requests and their matching by a network classifier, as shown in Fig. 3. For a new coming request, we make it numerical. Then, we obtain the confidence of request for each class through the network, which means the probability that the request exists in each class. The matching process starts with the class that has the highest confidence. Then, the engine traverses each class as the confidence goes from high to low until we find it, or fail to find it after searching all classes.

4 Clustering and neural networks

For the purpose of training neural networks, there should be a labeled policy set. With the pursuit of efficient, the K-means clustering algorithm can divide rules into different clusters in high speed.

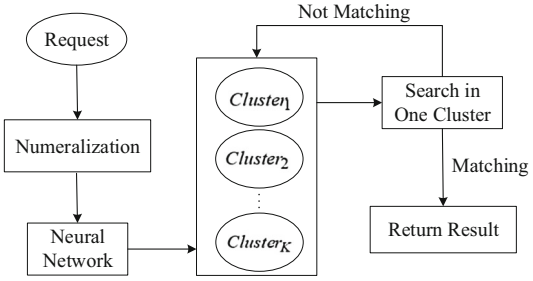

Fig. 3 Matching phase of new requests

Table 1 Extracted rules

Subject0	Resource0	Action0	Condition0	Deny
Subject1	Resource0	Action1	Condition0	Deny
Subject2	Resource1	Action2	Condition0	Permit
Subject3	Resource2	Action3	Condition1	Permit

Table 2 Rules in numeric format

Subject	Resource	Action	Condition	Effect
0	0	0	0	0
1	0	1	0	0
2	1	2	0	1
3	2	3	1	1

4.1 Clustering

We first convert a policy into a numeric format by the converting algorithm proposed in Deng et al. (2019a). This algorithm takes a policy as an input and converts the four main attributes of each rule in the policy to numerical values in order. For example, after having extracted rules from a sample policy, as shown in Table 1, we can obtain rules in numerical format, as shown in Table 2.

In order to divide policies into different distinct subsets, we use the K-means clustering algorithm, serving as an unsupervised method that can help us obtain K clusters of rules and K has to be given in advance (Hartigan and Wong 1979). Compared with the previous clustering works that mainly consider subjects as a main factor to divide rules, the K-means clustering algorithm takes all attributes into consideration. The result of K-means clustering will initially perform better in classification process, and the synchronous K-means algorithm is shown in Algorithm 1. Notations used in Algorithm 1 are detailed in Sect. 2. T is the max time of iterations and x_i denotes the ith numeric rule. $Center(x_i)$ represents the centroid of rule x_i, and $Cluster_i$ represents ith cluster.

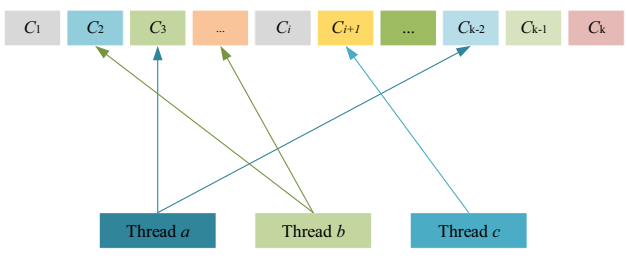

Fig. 4 Multi-thread clustering

Algorithm 1: Synchronous K-means cluster and training of neural networks

Input: Policy set in a numerical format

Output: K clusters of rules and a neural network classifier

1: **for** $i = 1, 2, \ldots, T$ **do**

2: **for** every x_i **do**

3: Calculate Euclidean distance between centroids and other data points

4: Assign x_i into the cluster which has the nearest centroid

5: **end for**

6: Update centroids using

$$Center(x_i) = \frac{1}{|Cluster_i|} \sum_{c \in Cluster_i} c$$

7: Calculate the difference between two iterations

8: **end for**

9: Train neural networks with these K clusters

10: **return** K clusters and the parameters of neural networks

As shown in Algorithm 1, the K-means algorithm consists of two phases. The first phase is to pick up K centroids and assigns each point into a cluster to make the distance between the rule points and the centroid as small as possible. Therefore, we need a function to compute the distance between two objects. Here, Euclidean distance is used. After obtaining an initial group of clusters, we recalculate the new centroids and assign each point again based on the new centroids. This process is repeated until the distance is minimum. Thus, the K-means algorithm is an iterative procedure that minimizes the sum distance between data points and their cluster centroids.

However, K must be given in advance and usually it is not easy to find the most appropriate K to obtain high performance. Intuitively, K is expected to be as big as possible. The more clusters we have, the less rules to be searched afterwards. However, too many clusters, which means less rules in a cluster, also increase the difficulty to find which cluster has the applicable rule. Therefore, clusters with fewer rules do not have capability for generalization, which means a coming request may not find the applicable rule easily.

There are many irrelevant calculations in the above procedures. It is natural to think of using an asynchronous method to make calculations faster. The process of this method is shown in Fig. 4.

Algorithm 1 can be considered as a single-thread procedure. Therefore, it is natural to improve Algorithm 1 by using a multi-thread technique. The distance between different data points in a policy set is not obviously distinguishable. In other words, the different centroids can be very close. To this end, a new improved approach, called an asynchronous SGDK-means (stochastic gradient descent K-means) algorithm, making centroids more distinct, is shown in Algorithm 2. Notations used in Algorithm 2 are the same as Algorithm 1. m_s is a parameter to control the times of increasing distances between two closed centroids.

It should be emphasized that Algorithms 1 and 2 have the same inputs and outputs, but Algorithm 2 delivers a significant improvement on the time of clustering and training of neural networks.

After applying the K-means clustering algorithm to large-scale policy sets, we can obtain K centroids that can

be used to help us search for the applicable rule by comparing the similarity between the coming request and these centroids. The details of this process will be exposed in the next subsection.

Algorithm 2: Asynchronous SGDK-means cluster and training of neural networks

Input: Policy set in a numerical format

Output: K clusters of rules and a neural network classifier

1: **for** $i = 1, 2, \ldots, T$ **do**

/* Parallel computing lines 2–8 */

2: **for** every x_i **do**

3: Calculate Euclidean distance between centroids and other data points

4: Assign x_i into the cluster that has the nearest centroid

5: **end for**

6: Save all $\nabla_{x_i}(Center)$ for all x_i

7: Update centroids using

$$Center(x_i) = \frac{1}{|Cluster_i|} \sum_{c \in Cluster_i} c$$

8: Calculate the difference between two iterations

/* Synchronous Barrier for all threads */

/* Parallel computing lines 9–14 */

9: **for** $t = 1, 2, \ldots, m_s$ **do**

10: Randomly select x_t, $x_t \in X$

11: Calculate $Center_{new}$ of x_t, and get $\nabla_{x_t}(Center_{new})$

12: **if** $Center_{new} \neq Center(x_t)$

13: $Center_{new} = Center_{new} - \nabla_{x_t}(Center_{new})$

14: **end if**

/* Synchronous Barrier for all threads */

15: **end for**

16: **end for**

17: Train neural networks with these K clusters

18: **return** K clusters and the parameters of neural network

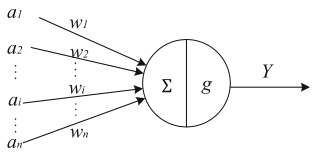

Fig. 5 Calculation in a neuron

4.2 Construction and training of neural networks

After having divided a policy set into K clusters, when a request is coming, we should determine which cluster has the most possibly applicable rule. To make this process more efficient, we train a two-layer full-connected neural network and obtain an estimation model.

A neural network is a model to classify an input by calculating how possible the input belongs to a class. It utilizes a unit called neuron to construct a whole network to simulate the working method of human brain. After feeding the network a policy set already classified, it can train itself to fit the policy set. Then, when an unknown sample comes, the network can estimate to which class it most possibly belongs. In fact, a neural network draws a decision boundary by formatting a complex function to calculate the possibility. Every neuron uses a simple function called activation function to calculate the activation of the input from last layer and the output of this layer is propagated to the next layer after multiplying weights.

As shown in Fig. 5, where $\sum$ denotes the summation and g is the activation function, a neuron takes the output from last layer as input, sums them up, uses an activation function to calculate an activation and propagates it to the next layer after multiplying a weight. All the weights are parameters of this network and they can be adjusted automatically in order to learn the features of a policy set. A neural network is composed of many neurons and formats a complex function by combining all the simple functions in every neuron.

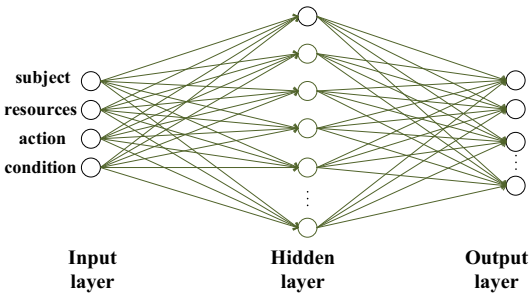

Fig. 6 Architecture of neural networks

After learning the features from a policy set, a neural network can estimate to which class a new sample belongs by using a complex function formatted during the learning. There are several widely used activation functions, such as sigmoid function, and rectified linear unit (ReLU), as shown in Eqs. (2) and (3). The whole process of forward propagation is shown in Eqs. (4), (5) and (9), where x is the sum of outputs from last layer.

$$S(x) = \frac{1}{1+e^{-x}} \tag{2}$$

$$f(x) = max(0, x) \tag{3}$$

In the process of forward propagation, the activation of every neuron A_i is propagated to the next layer after multiplying a weight W_i and plus a bias b_i, i.e.,

$$z_i = W_{i-1}A_{i-1} + b_{i-1} \tag{4}$$

Every neuron takes the output from the last layer z_{i-1} and generates new activation a_i to be propagated, i.e.,

$$a_i = g(z_{i-1}) \tag{5}$$

After having obtained predictions, we can compare them with the correct label vector Y, which is already labeled by the K-means algorithm. According to false estimation provided by a network and the cost function J (calculated by Eq. (6)), the backup propagation algorithm can make accuracy higher by using back propagation in order to compute the partial derivative of the cost function (noted as $\delta^{(i)}$) (David et al. 1986). We have

$$J(W_i) = -Y\log(A_i) - (1-Y)\log(1-A_i) \tag{6}$$

Then by using an optimization algorithm, such as gradient descent, we can find a prediction model with high performance. The gradient descent is a widely used method to find a solution in an optimization problem, as shown in Eq. (7), where α is the learning rate in gradient descent.

$$W_i^{\text{new}} = W_i^{\text{old}} - \alpha \frac{\partial}{\partial W_i^{\text{old}}}(W_i^{\text{old}}) \tag{7}$$

The whole process of back propagation is shown in Eq. (8). The new parameter of the ith layer Θ_i^{new} can be obtained by updating the old parameter Θ_i^{old} plus learning rate η times $\delta^{(i)}$.

$$\Theta_i^{\text{new}} = \Theta_i^{\text{old}} + \eta\delta^{(i)} \tag{8}$$

To calculate $\delta^{(i)}$, a chain rule is used (shown in Eq. (9)). The partial derivative of the cost function E_i with respect to the parameters of the last layer W_{i-1} can be calculated by multiplication of the partial derivative of cost function E_i with respect to activation A_i, the partial derivative of A_i with respect to the output from the last layer Z_{i-1} and the

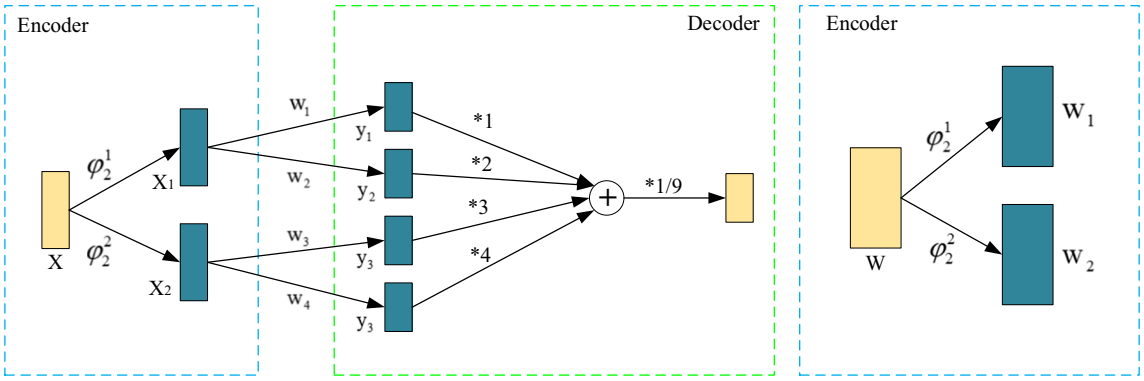

Fig. 7 Structure of quantitative neural network

partial derivative of Z_{i-1} with respect to parameters of the last layer.

$$\delta^{(i)} = \frac{\partial E_i}{\partial W_{i-1}} = \frac{\partial E_i}{\partial A_i}\frac{\partial A_i}{\partial Z_{i-1}}\frac{\partial Z_{i-1}}{\partial W_{i-1}} \tag{9}$$

Therefore, we need to calculate $\frac{\partial E_i}{\partial A_i}$, $\frac{\partial A_i}{\partial Z_{i-1}}$ and $\frac{\partial Z_{i-1}}{\partial W_{i-1}}$ at first (shown in Eqs. (10), (11) and (12)).

The partial derivative of the output from the last layer Z_{i-1} with respect to W_{i-1} can be obtained from Eq. (4).

$$\frac{\partial Z_{i-1}}{\partial W_{i-1}} = A_{i-1} \tag{10}$$

The partial derivative of the ReLU function (shown in Eq. (3)) with respect to Z_{i-1} is shown in Eq. (11).

$$\frac{\partial A_i}{\partial Z_{i-1}} = \begin{cases} 0, z_{i-1} < 0 \\ 1, z_{i-1} \geq 0 \end{cases} \tag{11}$$

We can derive cost function E_i with respect to activation A_i from Eq. (6).

$$\frac{\partial E_i}{\partial A_i} = \frac{A_i - Y}{A_i(1 - A_i)} \tag{12}$$

Then, when an unknown sample comes, the network can determine to which class it most possibly belongs. In fact, a neural network draws a decision boundary to distinguish different classes (Cybenko 1989). After learning the features from a policy set, a neural network can estimate to which class a new sample belongs by using a complex function formatted.

As shown in Fig. 6, in our network model, there are four nodes in the input layer, corresponding to the object, resource, action and condition, respectively. The number of hidden layer nodes is uncertain, since it can be adjusted in order to improve the performance of evaluation on the Internet. We can even increase the number of the layers to improve the evaluation performance of the network. The number of output layer nodes should be the same as K that we assign to the K-means algorithm before. Therefore, the numbers of nodes in the input layer and the output layer in our network model are limited, and the hidden layer can be adjusted for the purpose to achieve the best training effect. The learning rate is set as 10^{-6}. In the early stage of the training network, the learning rate can be slightly higher. In the process of back propagation, if the initial learning rate is high, the value of the loss function will decrease rapidly. However, in the later stage, the learning rate should be adjusted to catch the optimal position of the loss function, which can be achieved by a programming method. During the training of a network, we pass data (object, resource, action, condition) into the network and obtain the output by forward propagation. In the output layer, the label vector is used to calculate the loss function. For example, we input a vector as done in Deng and Zhang (2015); Singh and Singh (2010); Kamoda et al. (2005); Wang et al. (2011) into a network and it is classified into the 60th cluster. Specially, if there are 100 nodes in the output layer, the 60th position of label vector should be 1 while the other positions should be 0. By the difference between the value of prediction and correct value, the value of loss can be calculated and the gradient can be propagated layer by layer using back propagation. At the same time, the parameters of the network are updated to achieve the goal of training the network.

At the last layer of network, we use the Softmax function (Blorasso and Sanguineti 1995). A_i is the output of the i th node in the last layer. All output values are converted to a probability distribution (shown in Eq. (13), m is the number of nodes in the last layer) with positive values whose sum is 1. What we obtain is a prediction of probability distribution for classification. As shown in Algorithm 2, we will search for the applicable rule in the most possible cluster. If we cannot find the applicable rule, a new cluster with the highest possibility will be searched until it is found.

$$\widehat{y} = \frac{e^{A_i}}{\sum_{i=1}^{m} e^{A_i}} \tag{13}$$

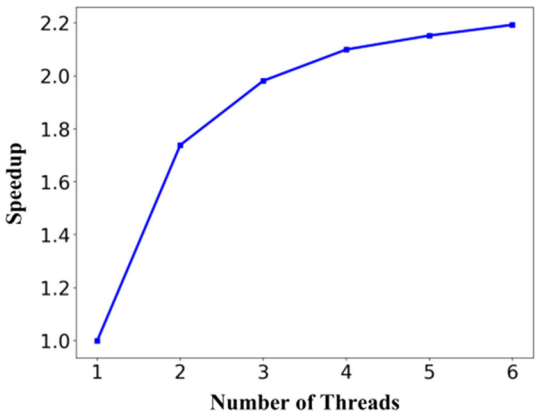

Fig. 8 Comparison in clustering using different number of threads of LMS

However, the algorithms above require expensive computational cost of a server. We can distribute the computation in different embedded devices to lessen a server's computational cost. To optimize the algorithm, we evolve a simple neural network into a quantitative neural network, as shown in Fig. 7.

$$\text{Encoder}(x) = \begin{cases} \varphi\frac{1}{2} : sign\left(\sin\left(\frac{3}{4}\pi x\right)\right) \\ \varphi\frac{2}{2} : sign\left(-\sin\left(\frac{3}{2}\pi x\right)\right) \end{cases} \tag{14}$$

In the later calculations, the following activation functions are used to process the input from last layer and calculate the activation.

$$\text{HTanh}(x) = \begin{cases} +1, & x > 1 \\ x, & -1 \le x \le 1 \\ -1, & x < -1 \end{cases} \tag{15}$$

$$\text{HReLU}(x) = \begin{cases} +1, & x > 1 \\ x, & 0 \le x \le 1 \\ -1, & x < 0 \end{cases} \tag{16}$$

This type of neural networks can perform well in an embedded device, like FPGA, and thus, the requirement of a server's computational cost is reduced, since the calculation can be finished in the embedded devices.

After the network is trained, the model can be used to find the application rule, as shown in Algorithm 3.

Algorithm 3: Search for the applicable rule

Input: A request

Output: The applicable rule

1: Transform Req into a numerical format Req_{num}

2: Feed Req_{num} to the network and obtain K confidences in an ascending order

3: **while** not found applicable rule **do**

4: Search in the cluster

5: Next cluster

6: **end while**

7: **return** the applicable rule

The activation function in this type of neuron network should be adjusted to adapt to the new network, calculated by the following formula.

In the part of encoder, we use Eq. (14) to process the input, where x is a numrical input request.

5 Experimental results and analyses

In this section, we demonstrate experimental policies, the method to generate our test policies and comparisons of evaluation performance among XDNNEngine, the Sun PDP, XEngine and SBA-XACML. Our experiments are

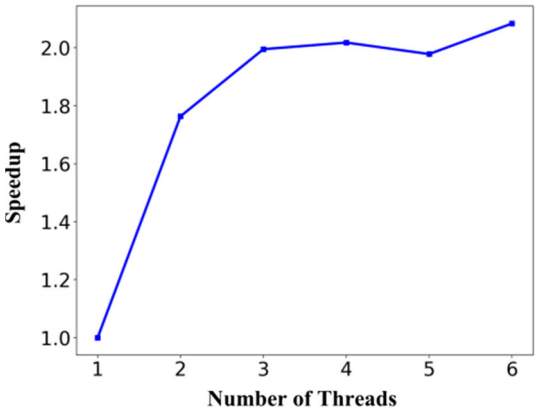

Fig. 9 Comparison in clustering using different number of threads of VMS

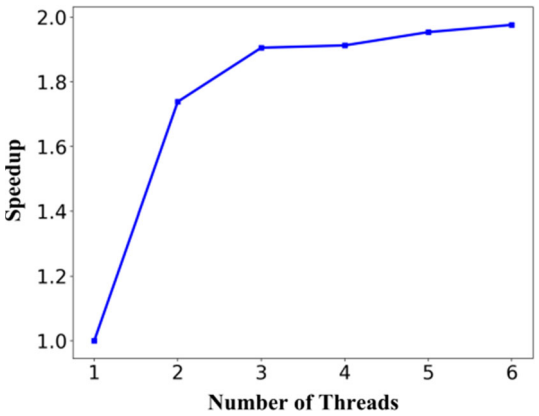

Fig. 10 Comparison in clustering using different number of threads of ASMS

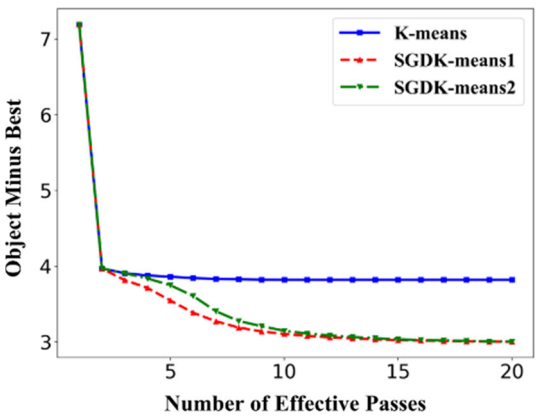

Fig. 11 Comparison between K-means and SGDK-means of LMS

conducted on a laptop running windows 10 with 8 GB of memory and quad-core 2.50 GHz Intel processor.

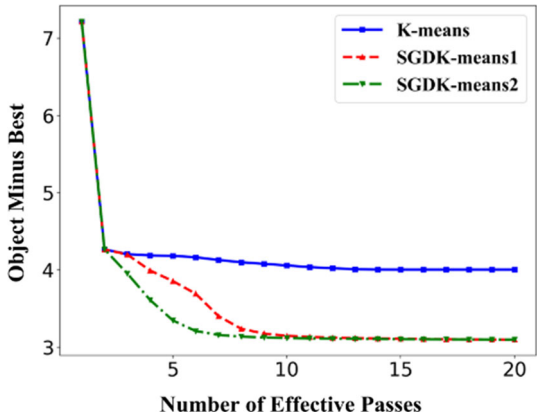

Fig. 12 Comparison between K-means and SGDK-means of VMS

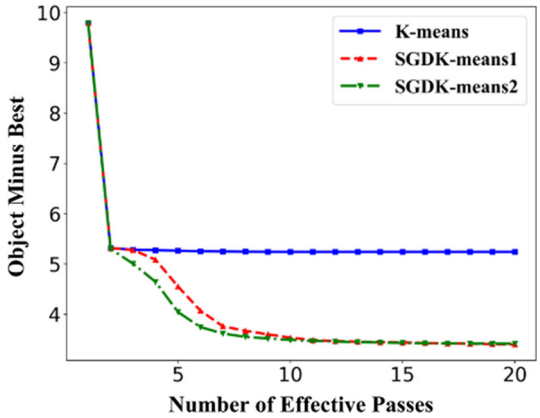

Fig. 13 Comparison between K-means and SGDK-means of ASMS

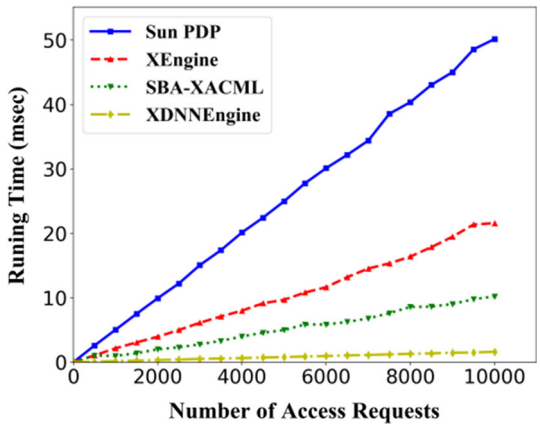

Fig. 14 Comparison of evaluation performance of LMS

5.1 Experimental policies

For the purpose of simulating practical application scenarios, three XACML access control policies are chosen as follows.

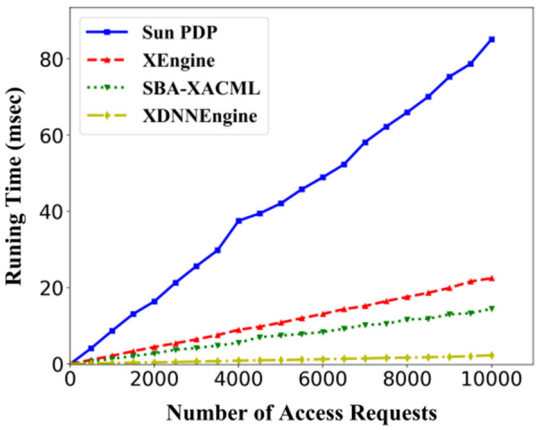

Fig. 15 Comparison of evaluation performance of VMS

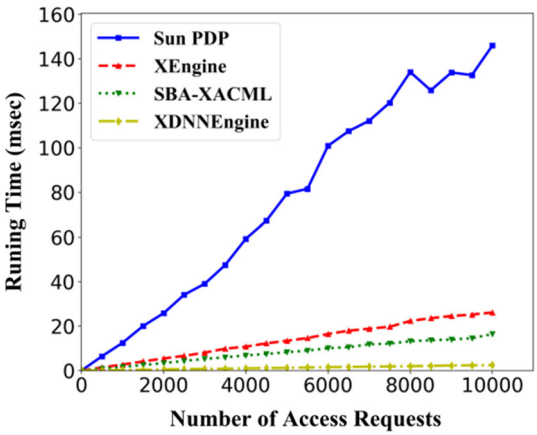

Fig. 16 Comparison of evaluation performance of ASMS

(1) Library management system (LMS) (Pretschner and Baudry 2008) provides access control policies such that a public library can use web services to manage books.
(2) Virtual meeting system (VMS) (Mouelhi et al. 2008) provides access control policies such that web conference services can be managed.
(3) Auction sale management system (ASMS) (Mouelhi et al. 2009) provides access control policies such that buying and selling items online is available.

The policies in the LMS, VMS and ASMS contain 720 rules, 945 rules and 1760 rules, respectively.

In order to satisfy the actual requirement, we expand the three policy sets by a random combination method to make comparisons of a result more clearly. According to Cartesian Products under Subject, Resource, Action and Condition, we create new rules and insert them into the original policy. Finally, the numbers of policy rules contained in the LMS, VMS and ASMS are expanded to 10,000, 20,000 and 30,000, respectively.

This experiment is only based on the simplest policy sets. We do not consider the factors such as scattered predicates, multi-valued rules.

5.2 Generation of test requests

Policies presented in Martin (2006) are utilized for maximizing the coverage of the test, which can automatically generate access requests meeting *Change-Impact*. The main idea is that conflicting rules can be obtained by conflict detection tools according to the fact that different policies or different rules in the same policy can make inconsistent results of evaluation for the same request, and that correlative access requests can be constructed for testing according to the conflicting rules.

The study in Wei et al. (2011) suggests that access requests can be generated automatically to test the correctness of the PDP and the configured policies, which indicates that Context Schema, defined by the XML Schema of the XACML, describes all the structures of the access requests that might be accepted by the PDP. This paper shows that the developed *X-CREATE* can generate possible structures of access requests according to the Context Schema of the XACML. The policy analyzer obtains probable 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. Another test scheme is Simple Combinatorial that can generate access requests according to all the possible combinations of attribute values of *Subject*, *Action*, *Resource* and *Condition* in the XACML policies.

In order to make our experiment meet the practical requirement, the combination of *Change-Impact*, *Context Schema* and *Simple Combinatorial* is adopted to simulate the actual access requests in this paper.

5.3 Clustering and distributed methods

In our experiments, we compare our implementation with the Sun PDP, XEngine and SBA-XACML, respectively.

The Sun PDP, as a universally applied policy decision point (Kateb et al. 2012), is able to evaluate access requests based on the internal rule matching mechanism (Ramli et al. 2014). The Sun PDP has become an industry standard and the most widely deployed implementation of XACML evaluation engine.

SBA-XACML (Mourad and Jebbaoui 2015) is a novel set-based algebra (i.e., SBA) scheme that provides efficient evaluation of XACML policies. SBA-XACML contains formal semantics and algorithms that take advantage of the mathematical operations to provide efficient policy evaluation.

XEngine (Liu et al. 2008) can not only convert a textual XACML policy to a numerical policy, but also convert a numerical policy from complex structures to normalized ones. Moreover, as a policy evaluation engine, XEngine can translate numerical policies into tree data structures and deal with requests efficiently.

In our experiment, we divide the policy sets into arbitrary classes by the K-means algorithm. There are less rules in each class when the number of classes increases. As a result, we divide the LMS, VMS and ASMS into 30, 40 and 50 classes, respectively. In this way, the time cost in searching has dramatically reduced.

5.3.1 Clustering time

In this experiment, we use multi-thread to increase the speedup of clustering such that it can efficiently make calculations faster. Figures 8, 9 and 10 show how speedup varies in different thread numbers in three given policy sets.

From Fig. 8, 9 and 10, we observe that.

(1) Speedup gets higher when the number of threads increases in the LMS, VMS and ASMS. When the number reaches three, it remains steady.
(2) The asynchronous method is obviously better than the single thread method in terms of clustering time.

5.3.2 Object minus best

To make the distance between different data points in the policy set more distinguishable, an SGDK-means algorithm is utilized in our experiment. We compare SGDK-means and K-means in this section. In Figs. 11, 12 and 13, SGDK-means1 picks up 5 centroids and SGDK-means2 picks up 10 centroids.

From Figs. 11, 12 and 13, we observe that.

(1) Object minus best declines more significantly in SGDK-means than in K-means when the number of effective passes grows.
(2) When the number of effective passes is more than 1, object minus best of K-means stays at a high level, whereas SGDK-means decreases when effective passes number increases.
(3) A higher rate of descent can be seen in SGDK-means2 compared with SGDK-means1 as the more clusters we have, and the less rules should be searched afterward.

5.3.3 Evaluation time

In this part, we compare our method with Sun PDP, XEngine and SBA-XACML in terms of evaluation time. We randomly generate 500, 1000, …, 10,000 requests to record the PDP evaluation time. The results can be seen in Figs.14, 15 and 16.

From Figs.14, 15 and 16, we observe that.

(1) Given the three policy sets, when the number of access requests grows, the evaluation time of Sun PDP, XEngine, SBA-XACML and XDNNEngine increases.
(2) The rate of increase in evaluation time of XDNNEngine is less than that of the rest three methods in the LMS, VMS and ASMS.

6 Conclusions and future work

In this paper, an approach based on neural networks is implemented to improve the PDP evaluation performance. After converting the rules into a numerical format, we use an SGDK-means algorithm to divide a policy set into K clusters. Based on the obtained clusters, we train a two-layer full-connected neural network. When a request is coming, this network can calculate possibilities that the applicable rule belongs to each cluster. According to the confidence given by the network, we search for the applicable rule among the clusters on the basis of possibility from high to low until we find it or the applicable rule does not exist. If we want to insert a new rule into the policy set, after converting it into a numerical format, we can put it into the most appropriate cluster by the distances between this rule and each centroid that has been obtained during the application of an SGDK-means algorithm.

Compared with the Sun PDP, XEngine and SBA-XACML, the proposed approach reduces the PDP evaluation time dramatically under the condition of loading a large-scale policy set. Moreover, the application of K-means algorithm makes it much easier to insert a new rule into the original policy set. What is more valuable, this approach can find the applicable rule faster than other methods. We believe that the training of neural networks will be easier when the capability of computation is improved. The proposed method can be employed to study big data access control in social networks (Yu et al. 2021a; Yang et al. 2020) and CoVID-19 (Yu et al. 2021b, c, d).

Funding This study was funded by (1) The Natural Science Foundation of Shaanxi Province in China (Grant numbers: 2019JM–020, 2019JM–162 and 2020JM-526). (2) The Science Research Plan Project of Education Department of Shaanxi Province (Grant number:

18JK0507). (3) The National Natural Science Foundation of China (Grant numbers: 61873277, 71571190, and 61702408). (4) The Innovation Group for Interdisciplinary Computing Technologies, School of Computer Science and Technology, Xi'an University of Science and Technology.

Declarations

Conflict of interest All authors declare that they have no financial and personal relationships with other people or organizations that can inappropriately influence their work; there is no professional or other personal interest of any nature or kind in any product, service and/or company that could be construed as influencing the position presented in, or the review of, the manuscript entitled, "Improvement on PDP Evaluation Performance Based on Neural Network and SGDK-means Algorithm."

Ethical approval This article does not contain any studies with human participants or animals performed by any of the authors.

Informed consent Informed consent was obtained from all individual participants included in the study.

References

Alex FC, Liu X (2008) Xengine: a fast and scalable XACML policy evaluation engine, In: Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, pp. 265–276

Angulo P, Guzmán CC, Jiménez G, Romero D (2017) A service-oriented architecture and its ICT-infrastructure to support eco-efficiency performance monitoring in manufacturing enterprises. Int J Comput Integr Manuf 30:202–214

Atlam FH, Alassafi OM, Alenezi A, Walters JR and Wills BG (2018) XACML for building access control policies in internet of things, In: Proceedings of the International Conference on Internet of Things, Big Data and Security, pp. 253–260.

Blorasso P, Sanguineti V (1995) Self-organizing body schema for motor planning. J Motor Behavior 27:52–66

Cybenko G (1989) Approximation by superpositions of a sigmoidal function. Math Control Sig Syst 2:303–314

David ER, Geoffrey EH, Ronald JW (1986) Learning representations by back-propagation errors, Letters to. Nature 323:533–536

Deng F, Zhang LY (2015) Elimination of policy conflict to improve the PDP evaluation performance. J Netw Comput Appl 80:45–57

Deng F, Zhang LY, Zhou BY, Zhang JW, Cao HY (2016) Elimination of the redundancy related to combining algorithms to improve the PDP evaluation performance. Math Probl Eng 7608408:1–18

Deng F, Zhang LY, Zhang CY, Ban H, Wan C, Shi MH, Chen C, Zhang ET (2019a) Establishment of rule dictionary for efficient XACML policy management. Knowl-Based Syst 175:26–35

Deng F, Lu J, Wang SY, Pan J, Zhang Y (2019b) A distributed PDP model based on spectral clustering for improving evaluation performance. World Wide Web 22:1555–1576

Han WL, Lei C (2012) A survey on policy languages in network and security management. Comput Netw 56:477–489

Hartigan JA, Wong MA (1979) A K-means clustering algorithm. J R Stat Soc 28:100–108

Jebbaouia H, Mourada A, Otrokb H, Haratya R (2015) Semantics-based approach for detecting flaws, conflicts and redundancies in XACML policies. Comput Electr Eng 44:91–103

Jiang S (2018) Machine learning research in big data environment, In: Proceedings of the Fifth International Conference on Electrical & Electronics Engineering and Computer Science, pp. 227–231

Kamoda H, Yamaoka M, Matsuda S, Broda K, Sloman M (2005) Policy conflict analysis using free variable tableaux for access control in Web services environments, In: Proceedings of the Fourteenth International World Wide Web Conference, pp. 121–126.

Kateb DE, Mouelhi T, Traon YL, Hwang JH, Xie T (2012) Refactoring access control policies for performance improvement, In: Proceedings of the International Conference on Performance Engineering, pp. 323–334

Kuang TP, Ibrahim H, Sidi F, Udzir NI (2018) An effective modality conflict model for identifying applicable policies during policy evaluation. J Adv Comp Eng Technol 4:255–266

Lin D, Rao P, Ferrini R, Bertino E, Lobo J (2013) A similarity measure for comparing XACML policies. IEEE Trans Knowl Data Eng 25:1946–1959

Liu T, Wang Y (2015) Beyond scale: an efficient framework for evaluating web access control policies in the era of big data, In: Proceedings of the Advances in Information and Computer Security, pp. 361–334

Liu AX, Chen F, Hwang JH, Xie T (2008) XEngine: a fast and scalable XACML policy evaluation engine, In: Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, pp. 265–276

Marouf S, Shehab M (2009) Statistics & clustering based framework for efficient XACML policy evaluation, In: Proceedings of the IEEE International Symposium on Policy for Distributed Systems and Networks, pp.118–125.

Martin E (2006) Automated test generation for access control policies, In: Proceedings of the International Workshop on Software Engineering for Secure Systems, IEEE Computer Society, pp.752–753

Mohan A, Blough DM, Kurc T, Post A, Saltz J (2011) Detection of conflicts and inconsistencies in taxonomy-based authorization policies, In: Proceedings of the IEEE International Conference on Bioinformatics and Biomedicine, pp. 590–594

Mouelhi T, Fleurey F, Baudry B, Traon YL (2008) A model-based framework for security policy specification, deployment and testing, In: Proceedings of the Eleventh International Conference on Model Driven Engineering Languages and Systems, Berlin, pp. 537–552

Mouelhi T, Traon YL, Baudry B (2009) Transforming and selecting functional test cases for security policy testing, In: Proceedings of the International Conference on Software, pp. 171–180

Mourad A, Jebbaoui H (2015) SBA-XACML: Set-based approach providing efficient policy decision process for accessing Web services. Expert Syst Appl 42:165–178

Ngo C, Demchenko T, Laat CD (2015) Decision diagrams for XACML policy evaluation and management. Comput Secur 49:1–16

OASIS. eXtensible Access Control Markup Language (XACML) V2.0 specification set. URL http://www.oasis-open.org/committees/xacml/, 2007.

Peña JM, Lozano JA, Larrañaga P (1999) An empirical comparison of four initialization methods for the K-Means algorithm. Pattern Recogn Lett 20:1027–1040

Petersen P, Voigtlaender F (2018) Optimal approximation of piecewise smooth functions using deep ReLU neural networks. Neural Netw 108:196–330

Pretschner A, Baudry B (2008) Test-driven assessment of access control in legacy applications, In: Proceedings of the International Conference on Software Testing, Verification, and Validation. Lillehammer, pp. 238–247

Ramli CDPK, Nielson HR, Nielson F (2014) The logic of XACML. Sci Comput Program 83:80–105

Santiago PR, Mario L, Félix GM (2012) Graph-based XACML evaluation, In: Proceedings of the Seventeenth ACM Symposium on Access Control Models and Technologies, pp. 83–92

Singh K, Singh S (2010) Design and evaluation of XACML conflict policies detection. Int J Comp Sci Inf Technol 2:65–74

Sun's XACML implementation (2015). URL http://sunxacml.sourceforge.net/.

Teo PK, Ibrahim H, Udzir NI, Sidi F (2013) Heterogeneity XACML policy evaluation engine, In: Proceedings of the Second International Conference on Digital Enterprise and Information Systems, pp. 230–238

Wang YZ, Feng DG, Zhang LW, Zhang M (2011) XACML policy evaluation engine based on multi-level optimization technology. J Softw 22:323–338

Wang X, Shi W, Xiang Y, Li J (2016) Efficient network security policy enforcement with policy space analysis. IEEE/ACM Trans Network 24:2926–2938

Wei S, Yen IL, Bastani F, Bao T, Thuraisingham B (2011) Role-based integrated access control and data provenance for SOA based net-centric systems, In: Proceedings of the Sixth IEEE International Symposium on Service Oriented System Engineering, pp. 225–234

Yang L, Yu Z, El-Meligy MA, El-Sherbeeny AM, Wu N (2020) On multiplexity-aware influence spread in social networks. IEEE Access 8:106705–106713

Yu Z, Lu S, Wang D, Li W (2021a) Modeling and analysis of rumor propagation in social networks. Inf Sci 580:857–873

Yu Z, Ellahi R, Nutini A, Sohail A, Sait SM (2021b) Modeling and simulations of CoViD-19 molecular mechanism induced by cytokines storm during SARS-CoV2 infection. J Mol Liq 327:114863

Yu Z, Arif R, Fahmy MA, Sohail A (2021c) Self organizing maps for the parametric analysis of COVID-19 SEIRS delayed model. Chaos Soliton Fract 150:111202

Yu Z, Abdel-Salam ASG, Sohail A, Alam F (2021d) Forecasting the impact of environmental stresses on the frequent waves of COVID19. Nonlinear Dyn 106:1509–1523

Zerari N, Chemachema M, Essounbouli N (2019) Neural network based adaptive tracking control for a class of pure feedback nonlinear systems with input saturation. IEEE/CAA J Autom Sinica 6:278–290