World Wide Web (2019) 22:1555–1576
https://doi.org/10.1007/s11280-018-0588-8

A distributed PDP model based on spectral clustering for improving evaluation performance

Fan Deng[1] · **Jie Lu**[2] · **Shi-Yu Wang**[2] · **Jie Pan**[3] · **Li-Yong Zhang**[2]

Received: 27 September 2017 / Revised: 4 February 2018 / Accepted: 15 May 2018 /
Published online: 22 May 2018

Abstract In modern access control systems, the Policy Decision Point (PDP) needs to be more efficient to meet the ever-growing demands of Web access authorization. Present XACML implementations of access control systems follow the same architecture based on ABAC, but varies in the design of PDP and other components. As a critical process in PDP, evaluation of attributes is often implemented in a simple and inefficient way in real applications. In order to improve the PDP evaluation performance, we propose a novel distributed PDP model, called XPDP, based on the combination of two-stage clustering and reordering to eliminate the limitation of computational performance of a single PDP. Firstly, we cluster rules based on *subject* and use spectral clustering method to perform further clustering. Secondly, the clusters of rules are reordered before evaluation for every inbound request based on similarity. Finally, we introduce a distributed PDP architecture for distributed deployment, providing with a brand new perspective of designing access control systems. A comparison in evaluation performance between the XPDP and the Sun PDP, as well as SBA-XACML, is made. In the experiment of using 10,000 synthetic access requests with three practical policy sets, the XPDP is 3.26 times faster than Sun PDP, and is 1.85 times faster than SBA-XACML. Experimental results show that the PDP evaluation performance can be prominently improved.

Keywords Policy decision point(PDP) · Evaluation performance · Clustering and reordering · Spectral clustering · Distributed deployment

✉ Fan Deng
deng_enya@126.com

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

[2] School of Software, Xidian University, Xi'an 710071, China

[3] School of Engineering, Hong Kong University of Science and Technology, Hong Kong, China

1 Introduction

Access control mechanism can be found everywhere in modern internet circumstances. The most significant purpose of applying access control is that it protects private information [7]. More specifically, it limits internet users to access resources within the boundaries determined by certain rules. The application of access control has been consistently enlarging in recent years thanks to the fast advancing technologies such as cloud computing [23], ubiquitous computing [29] and e-business [28], which all involve access control as an important part to support resource sharing, privacy protecting, etc. The access control efficiency influences the time delay of most behavior involving communicating with a traditional Web server. In the application of cloud computing, the flexible and efficient handling of authorizations is particularly important to exploit the full benefits of cloud computing where cross-domain collaborations occur on-demand [27]. Therefore, it is a significantly meaningful issue to enhance the efficiency of access control.

Nowadays, the most widely adopted model based on the evaluation of policies is XACML (eXtensible Access Control Mark-up Language). It is an authorization policy language in XML format based on the Attribute-Based Access Control (ABAC) model [22]. It specifies the form of rules and policies, their hierarchy order and the details in the evaluation process. A set of rules constitutes a policy, which is a key element in the whole access control process. Requests are evaluated by the evaluation module, known as PDP (Policy Decision Point), based on rules. By authenticating a user and then using their locally assigned attributes or rights to make access decisions according to locally defined policies [14], the response to that request is made in form of saying *Permit* or *Deny*, which means whether the request is granted.

Despite its importance, high performance policy evaluation has received little attention. Most prior work on policies focuses on correctness issues [13] and security issues. It has been a pressing issue to find a best solution in PDP evaluation, or at least achieve better efficiency to meet the needs of growing demands. The efficiency of present policy evaluation suffers mainly because (of)

- The expansion of policies [5].
- The present algorithms for evaluation are primitive. In order to find the applicable rules, the worst case requires going through on the whole set of rules, which is extremely time costing when there is huge number of rules. There must be a new algorithm, or even a new structure of policies to boost this process.
- The situation of processing highly concurrent requests. Emerging applications, such as real-time enforcement of privacy policies in a sensor network or location-aware computing environment, require high throughput [1]. The idea of keeping the whole set of policies in a single PDP assures the robustness of evaluation, but this is only practical for the era when there are not too many requests. Also, relying on high performance computers to deal with large number of requests at the same time is neither economic nor helpful. This conservative solution would finally reach its limit when there is not faster computer. To scale beyond the capacity of a single server, distributed policy evaluation algorithms are needed, to coordinate concurrent processing of requests on multiple servers [2].

In actual applications, traditional systems fail to make use of some obvious features in a policy set, therefore are far from efficient in terms of evaluation delay. Consider a course selection system of universities. During the time before or at the beginning of a semester, all

students from different grades shall pick up some courses. Some courses are designed exclusively for students in a certain grade or major. For example, an art major freshman requesting to select a course in advanced computer programming is usually prohibited. Other limitations also exist. At the same time a professor is requesting to see how many students have already selected his course, and he has the permission to access such data as a teacher. The subject of requests decides many behaviors a student can't have. The common problem seen in this access control application is when a large number of requests from teachers and students are sent to the server simultaneously, the efficiency of access control process is being so strictly tested that many systems yields high response delay. While the evaluation engine is dealing with a queue of requests, it doesn't have any precedent awareness of the features in those requests, such as the identity of the requester and what action is being requested in general (selecting a course or checking the number of student), despite those features are rather evident, which makes the sequent compares of attributes in requests and rules a necessity. It could be more efficient if the evaluation engine has some precedent knowledge of the features of policies or requests.

The clustering method to avoid inapplicable matches is now regarded by many scholars as an effective way to improve evaluation efficiency. The difficult part of this method is that the clustering of policy set has to be precise and effective so that the process of evaluation knows which cluster is to be matched with requests. In fact, there are many algorithms that perform almost perfect clustering on policy set, many algorithms from the field of machine learning have been experimented to cluster policies, but the bottleneck is to find an approach effective in both clustering and evaluation.

This paper makes the following contributions.

- A preprocess for policy sets, the two stages of clustering, is presented, which precisely divides the policy set into several groups. This method overcomes the weakness of previous clustering approaches, since specifically the second phase clustering based on the similarity of rules, does not need to consider the intrinsic features of policy set and can directly and effortlessly perform the clustering, while also allowing to later retrieval according to the distinct features of access requests. This process ensures the applicable rules to be evaluated as early as possible, thus reduces the number of rules needed to process an access request, yet introduces little extra computation.
- A novel PDP model, called XPDP, is designed, which takes the advantage of clustered rules, also introduces reordering for clusters, and is capable of reducing the number of rules needed to evaluate an access request significantly. Comparing to the Sun PDP and SBA-XACML, our XPDP only needs 13.8 times less of rules and takes 3.6 times less of time to process a request.
- A distributed PDP architecture is presented. This eliminates the present limitation of computational performance of a single server. This is also an efficient distributed architecture for XACML access control system, partly because by performing clustering it enables distributed PDPs to work independently, overcoming the problem of communication between PDPs.

This paper is organized in the following order. Section 2 reviews the most pervasive methods to improve the efficiency of evaluation. Section 3 concludes our proposed approach, including clustering and distributed PDPs. Section 4 describes in detail about clustering to divide a policy set. In Section 5, we propose a framework of

distributed PDPs based on the existing clusters of policy set. Section 6 makes a comparison in evaluation performance between our proposed distributed PDP model and the Sun PDP, as well as the SBA-XACML, and evaluates the actual improvement on PDP evaluation performance. Finally, Section 7 presents some conclusions and directions for our future work.

2 Related work

The pursuit of better efficiency is always a strong desire to computer scientists. People achieve better performance by constantly modifying present algorithms, proposing new framework, etc. In the field of policy evaluation efficiency, scholars are researching from different perspectives. Keeping still the evaluation process but trying to modify the policy set, either by reducing the redundancy or changing the order of rules, has been researched recently. Others started from proposing brand new evaluation processes without modifying policy set.

2.1 Reordering and clustering

The basic manipulation in evaluation process is comparing attributes between requests and policies. To diminish the times of comparison is a simple yet effective method to improve efficiency, given that the prevailing Sun PDP is still using brute force to match rules. Guided by this idea, many techniques have been applied, including reordering and clustering of policies, etc.

S. Marouf [16, 17] proposed an adaptive framework for reordering and clustering. The reordering approach changes the priority of rules based on the frequency of successful hit of rules according to evaluation history. The more frequently used rule has a greater possibility to be used by subsequent requests as well, so by increasing its priority and reordering, the expected number of evaluations decreases. However, the simple reordering method still lacks an analysis of overall layout of policies, and the past request data in a particular period of time may not be representative enough to reflect all situations in any time. The assumption that requests are distributed in a consistent pattern is also dubious.

Liu et al. [11] proposed a new evaluation framework based on clustering methods to cut down policy-scale. They mainly focus on the preprocessing on policies and rules, which specifically contains two stage clustering, the first stage coarse-grained clustering and the second stage fine-grained clustering. The pre-clustered and approximate-applicable policies are assigned to evaluate a particular request, which greatly reduces the number of comparisons.

Previous work using clustering techniques is mainly based on different subjects of requests. It is very effective when there are several evident clusters of subjects, therefore accurate clustering of policies could be possible. For example, in course selection system, it is obvious that although there are many subjects, the clusters of subjects are students and teachers. Comparing with brute force evaluation, clustering certainly do improve the efficiency by rearranging the layout of rules and policies, yet heavily rely on explicit classification between subjects and is also difficult to implement. That is to say, where there is not an effective classification of subjects, clustering could be difficult and meaningless.

2.2 Decision diagram

By constructing a decision diagram (or matching tree) to facilitate the evaluation process is widely researched (e.g. [12, 13, 22, 24]). A matching tree is a tree structure, which has one root node and multiple terminal nodes. It converts the process of comparing each rule to walking through a decision path from the root to one of the terminal nodes. Specifically, for a request, extract its attributes and compare them with nodes step by step from root to terminal nodes. When it reaches a terminal node, the decision would be made. The critical intuition behind this method is to avoid the evaluation of every rules in the policy, thus improving the efficiency.

Liu proposed a new scheme for evaluation called the XEngine [13]. It first performs normalization and canonical representation on policies as preprocessing. The idea of policy normalization is to convert a policy with a complex logical structure to an equivalent policy with a simple logical structure [13], i.e., a numeric value produced by enumeration. Then the normalized policies are fitted into a decision diagram (matching tree). Besides the improvement of efficiency by applying matching tree, the evaluation within a PDP based on numerical form of policies is more efficient than processing textual strings. The experimental results show that its efficiency is orders of magnitude faster than that of Sun PDP [13]. But its weakness is that they do not support obligations due to the conversion of all combining algorithms to first-applicable [20]. Also, experiments by Azzam Mourad show that the main overhead reduction is achieved when all the requests (i.e. up to 100,000 requests) are received, converted and loaded in the memory at the same time [20], which is not practical in real-life application.

A modified XEngine is proposed by Pina Ros, which combines the *Matching Tree* with the *Combination Tree* to accommodate the XEngine to policy sets with different combination algorithms, and it also support obligations [24]. To be specific, they use the *Matching Tree* to find all applicable rules, then use those applicable rules to consist the *Combination Tree*, based on which the decision is made. The *Combination Tree* preserves the original hierarchy of combining algorithms [24], therefore it supports all combining algorithms. However, just like XEngine, storing all policies in the memory and expecting all requests received at the same time is not practical, also the efficiency comparing to the original XEngine is sacrificed for generality.

2.3 Others

Mourad et al. in 2015 proposed an SBA-XACML framework [20]. It transforms intermediate representation of XACML constructs into readable mathematical syntax based on set theory, and then converts it to SBA-XACML. Policies are compiled to the policies based on SBA-XACML and are stored. In real-life application, a coming request is preprocessed by compiler at the beginning, presented in SBA-XACML. Then it is evaluated by those compiled policies. Taking advantage of the mathematical operations provided by the proposed semantics, the performance of SBA-XACML surpasses that of XEngine. Another improvement is that it supports all combination algorithms. Based on this framework, Azzam Mourad introduces a novel set and semantics based scheme that allows to detect flaws, conflicts and redundancies between rules by offering new mechanisms to analyze the meaning of policy rules through semantics verification by inference rule structure and deductive logic [6].

Kolovski et al. [8] presented a formalization of XACML using description logic (DL), which is a decidable fragment of First-Order logic. Mapping XACML to description logic

allows to use off-the-shelf DL reasoners for analysis tasks and consequently make it capable of detecting redundant rules. Removing redundant rules from XACML policies may potentially improve the performance of XACML policy evaluation. However, this hypothesis is yet to be validated [12]. Also as is pointed out by Azzam Mourad et al. [6], they do not support multi-subject requests, complex attribute functions, rule conditions and Only-One-Applicable combining algorithm.

The idea to optimize the police set by removing all the redundant policies is discussed by many scholars. Wang et al. [30] proposed a framework called MLOBEE (multi-level optimization based evaluation engine). They prove that redundant policies can be deleted since doing this would not change the result of evaluation. They implement rule refinement to lessen scale policies and adjust the sequence of the rule. But the general method of refinement is not elaborated. Deng et al. [4] proposed an algorithm to automatically remove all the redundant rules. Although experiments explore that the efficiency does improve, and applying this method to present access control system doesn't require structural modification on PDP, the refinement of policies does not make a significant difference comparing to other frameworks, such as the XEngine.

3 Approach overview

Our main work includes the preprocessing of a policy set by clustering and the design of a distributed PDP system. The clustering of a policy set has the following two phases of manipulation on rules.

- Clustering based on the attribute subject. Rules with the same subject are put together as the first phase clustered sets $\{S_1, \ldots, S_n\}$, where S_i is a set of rules having the same subject.
- Spectral clustering on each S_i. This phase has many details including the definition and numerization of rules similarity, and using a graph to describe the relationship among rules. The spectral clustering algorithm is applied to perform a cut for the graph and the consequent subgraphs consist the clusters we need.

When constructing a distributed PDP system, the basic idea is to have many PDPs connected to internet, processing requests that are only applicable to the policy set in a particular individual PDP, therefore greatly shrink the set of rules to be evaluated. Each S_i in $\{S_1, \ldots, S_n\}$ is placed in a single PDP. Within an individual PDP, a request is preprocessed for the PDP to obtain basic features of the request, and then evaluated by a relatively small range of rules, which are nevertheless very likely to be applicable to the request. Specifically, the most likely applicable cluster is taken ahead of all other clusters to evaluate the request, and if the evaluation turns out to be inapplicable, then the second most likely applicable cluster of rules is used, and so forth. The evaluation process based on the second phase clustering are shown in Figure 1.

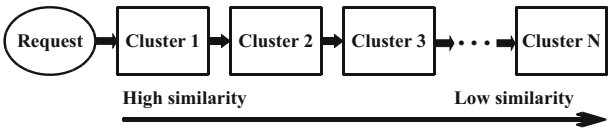

Figure 1 Evaluation process based on second phase clustering

In most cases, if the clustering of rules is effective enough, so that the applicable rule is found in the very beginning of evaluation, this process shall be much more efficient than brute-force search.

4 Clustering

Clustering of policies and rules based on subjects makes it possible to divide policy set into different groups, therefore those groups can be easily assigned to different PDP. It makes a significant difference in increasing efficiency by reducing the volume of policies in a single PDP. However, if the whole policy set is divided and distributed in many PDPs, each of which contains the only copy of policies and rules that other PDPs don't have, it becomes very difficult to accurately decide which PDP is to process the incoming request. More details about dispatching requests are discussed later. Having realized the afterward problems in distributed PDP application, we have to come up with a method to precisely divide a policy set.

Certainly any way of clustering is precise as long as the intersection of any two clusters is empty (Imaging cutting meat from a roasted mutton leg, to cut it precisely means there is no meat left after it is cut by a knife). In our case of evaluation system, the most precise clustering is to divide policy set based on a particular attribute in policies or rules. For example, if the attribute A of a rule equals X_i, where $X_i \in X$ and X is the collection of all values for attribute A, then the rule goes to cluster i. It is not only precise in clustering, but also straightforward in evaluation. We can just find the corresponding cluster that has its attribute. Another advantage is that the straightforward way of finding the corresponding policies is extremely essential in the distributed PDPs. The first phase clustering mentioned in section 3 is based on this idea.

The volume of a clustered subset of policy set might still be too large to handle. Reviewing the specific attributes in a policy set, there are many rules applicable to a single request. Therefore, more careful clustering is necessary to reduce the volume of the final applicable policies. The next step of clustering needs to dig deeper to divide the subset based on its intern features.

This two-phase clustering is expected to divide the whole policy set into two-layer subsets, which is precise in division, straightforward in finding corresponding subset, and very swift in evaluation because of its relatively small volume of individual subset of rules.

4.1 Clustering based on subjects

In XACML policies, a rule is the elementary unit of a policy, and the atomic element in PDP evaluation process. That is, a collection of rules can be used to evaluate, but components within a rule can't be evaluated separately. When a rule is evaluated, its content is the specific element being evaluated. Therefore, the similarity of rules is based on the similarity of the components within rules.

The main components of a rule can be described as a tuple shown in Formula 1.

$$Rule = (target, condition, effect) \tag{1}$$

The *effect* of a rule is either *Permit* or *Deny*. This attribute indicates the consequence of the evaluation for the rule. The *condition* of a rule represents a Boolean expression, the value of which is determined by checking certain attributes in a request, and it has to be *True* in order to be applicable to requests. The *target* is defined as a tuple shown in Formula 2.

$$target = (subjects, actions, resources) \tag{2}$$

Besides *condition*, the *target* of the rule also determines whether this *rule* is applicable. The three attributes denote what subjects it serves, what actions it may permit or deny, and what resources it authorizes. Only when all three attributes are applicable to the request can we say the *rule* to be applicable. The difference between *target* and *condition* is that the *condition* further refines the applicability established by *target*. That is, *target* is the more fundamental element in analyzing the applicability.

According to Formula 1 and 2, a rule is simply presented in Formula 3.

$$Rule = \{subjects, actions, resources, condition, effect\} \tag{3}$$

The rule evaluation basically relies on comparing attributes in rules with requests. The XACML defines the rule as a tuple, composed by target, effect and condition. As what has been introduced, the target is composed by three attributes: subject, resource and action. Classification based on attribute *subject* is ideal for performing the first phase division of rules, because it denotes the object who issues the request, and statistically in many applications it is an efficient way to create relatively small number of categories comparing to the number of rules. The result of this step is groups of rules which have the same *subject*. More than one group should be assigned to a PDP. In view of the independency of every distributed PDP, the group of rules are literally a policy set. For example, ten rules in a policy set are shown in Figure 2.

In this case, $Rule_1$ and$Rule_3$ have the same subject *Tester*, therefore they are clustered into the same group.

$Rule_4$ and $Rule_7$ also have the same subject, so finally there are four rules clustered in this group with *subject* = *Tester*. The first phase clustering on these ten rules based on *subject* has the output as shown in Figure 3.

4.2 Clustering based on similarity

In section 4, the rules with the same *subject* has been grouped and assigned to different PDPs. However, in many XACML policies, the condition that many rules share the same *subject* is common. Consider the application where a *subject* is allowed to access many *resources*, even after work in section 3.1, it is still necessary to have further clustering on a single PDP.

$Rule_1$={Tester,Write,D:/coder,11:00<Time<18:00,permit}
$Rule_2$={Coder,Open,D:/coder,09:00<Time<11:00,deny}
$Rule_3$={Tester,Write,D:/text,07:00<Time<08:00,permit}
$Rule_4$={Tester,Read,D:/coder,13:00<Time<18:00,deny}
$Rule_5$={Coder,Write,D:/document,15:00<Time<18:00,permit}
$Rule_6$={Developer,Read,D:/text,11:00<Time<12:00,deny}
$Rule_7$={Tester,Open,D:/coder,10:00<Time<13:00,permit}
$Rule_8$={Administrator,Read,D:/text,11:00<Time<15:00,deny}
$Rule_9$={Coder,Read,D:/coder,01:00<Time<07:00,deny}
$Rule_{10}$={Developer,Open,D:/document,00:00<Time<01:00,permit}

Figure 2 Example of rules

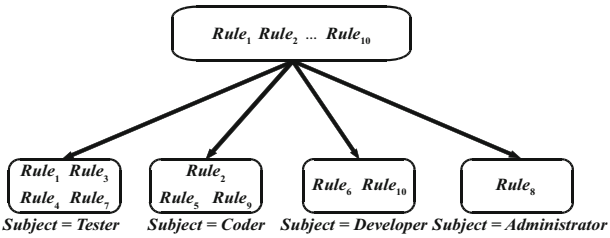

Figure 3 First phase clustering

Similar rules usually are all applicable to a certain request. To be specific, two rules are similar which have the same subject, resource or action. When the resource attribute of a request is *book,* other rules whose resource is not *book* are not applicable at all. So, when similar rules are clustered, requests just need to be evaluated by its corresponding cluster. So far, we have depicted the basic features of this new concept, that is, the similarity of rules. Previous researches in XACML policy also have this concept. Pranthima Rao et al. [9, 10] used the notion of *policy similarity measure*. Specifically, if the similarity score of policies P_1 and P_2 is higher than that of policies P_1 and P_3, it means P_1 and P_2 may yield the same decisions to a larger common request set than P_1 and P_3 will do. This is similar to our original goal of proposing similarity of rules.

4.2.1 Definition of similarity

The value domains of attributes in a rule are generally discretely distributed. Reviewing the ten rules listed in Figure 2, for instance, the value of *subject* is assigned from the set {Tester, Coder, Developer, Administrator}. If the similarity degree is defined from 0 to 1, considering $Rule_1$ and $Rule_3$, because the value of *subject* in these two rules is exactly identical, the similarity degree should be the maximum, or 1. This gives an idea of measuring the similarity of all attributes in two rules. Note that this definition of similarity is based on the rules having the same *subject*, therefore the similarity of the *subject* is neglected. To compare the similarity of two rules which are not included in the same subset produced in section 4.1 is meaningless.

The *Attribute Similarity Degree* (*ASD* for short) of corresponding attributes from two rules $Rule_1$ and $Rule_2$ is defined in Formula 4.

$$ASD_{Atrribute} = \begin{cases} 1 & (Atrribute_{Rule_1} = Attribute_{Rule_2}) \\ 0 & (Atrribute_{Rule_1} \neq Attribute_{Rule_2}) \end{cases} \tag{4}$$

Based on the definition of *ASD*, we define *Rule Similarity Degree* (*RSD* for short) to calculate the similarity of two rules. The first phase clustering has produced subsets of rules which all have the same *subject*, thus the similarity of rules is determined by the *action, resource* and *condition* attributes. The *RSD* of two rules is defined in Formula 5, where *w*1, *w*2, *w*3are three weights.

$$RSD(Rule_1, Rule_2) = w_1 * ASD_{action} + w_2 * ASD_{resource} + w_3 * ASD_{condition} \tag{5}$$

Nevertheless, the similarity of rules should not simply be the sum of the three *ASD*. Reviewing the process of matching a request and a rule, the component attributes are compared in sequential order, that is, *subjects* in requests and rules are first compared, then the *actions*, *resources* and *conditions* are compared. Consider that attributes at different

sequential position in rules require different cost when evaluating, for example, if *actions* in requests and rules are different, then the *resources* and *conditions* will not be compared because the rule is not applicable at all. With this regard, the weight of earlier compared attribute should be bigger than that of the later compared attribute. Using these weights, a numeric similarity measurement can be calculated.

In the definition above, assume that $w_1 = 0.6$, $w_2 = 0.3$, $w_3 = 0.1$. Take the three rules in Figure 4 as an example.

The *RSD* can be calculated in Formulas 6, 7, and 8.

$$RSD(Rule_1, Rule_2) = 0.6 \times 0 + 0.3 \times 1 + 0.1 \times 0 = 0.3 \tag{6}$$

$$RSD(Rule_1, Rule_3) = 0.6 \times 1 + 0.3 \times 0 + 0.1 \times 0 = 0.6 \tag{7}$$

$$RSD(Rule_2, Rule_3) = 0.6 \times 0 + 0.3 \times 0 + 0.1 \times 0 = 0 \tag{8}$$

4.2.2 Spectral clustering

The definition of *RSD* provides a useful lightweight approach to pre-compile a set of policies and returns the most similar policies for further exploration [10]. Given that the similarity of any two rules is already available in numerical form, the second phase clustering turns to be a mathematical problem. The optimized solution to this problem entails the detailed information about the features of data.

To have a clear sight of this problem, take the rules in Figure 2 as an example, after calculating the *RSD* of any two rules, the relationship of rules is described as a weighted undirected graph, shown in Figure 5. Note that Rules with $RSD = 0$, or say edges with $weight = 0$ are neglected.

The weighted graph consists of all rules and edges that link rules and shows the overall similarity. The basis of clustering in this case is to divide the graph ensuring that the rules within a cluster are tightly linked, or say the edge have large weights, while the rules in different clusters are loosely linked. Therefore, the clustering problem is converted to a graph partitioning problem based on the numeric format of similarity.

In fact, the task of finding good clusters has been the focus of considerable research in machine learning and pattern recognition [21], and have already been researched by many scholars. K-means and spectral clustering are among the algorithms which are most commonly used to solve this problem, and they are largely adopted in many applications such as computer vision. Spectral clustering has some significant advantages, it is reasonably fast especially for sparse data sets up to several thousands, and it is empirically very successful in graph partitioning.

$$Rule_1 = \{1, Tester, Write, D:/coder, 11:00 < Time < 18:00, permit\}$$
$$Rule_2 = \{2, Tester, Open, D:/coder, 09:00 < Time < 11:00, deny\}$$
$$Rule_3 = \{3, Tester, Write, D:/text, 07:00 < Time < 08:00, permit\}$$

Figure 4 Rules used to calculate RSD

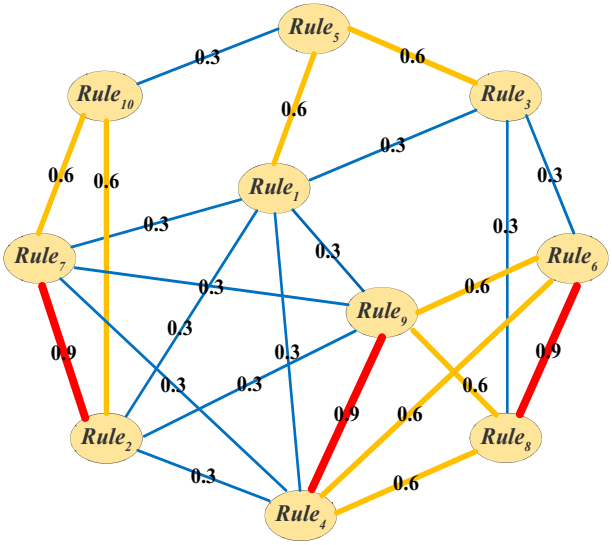

Figure 5 Weighted graph of rules

Here the spectral clustering method is adopted, which is effective to the circumstance where the graph is sparse in edge and plentiful in number of nodes. Spectral clustering helps to produce a cut for the graph which ensures the cut divides the policy set into many policies. *A Tutorial on Spectral Clustering* [15] gives the specific steps and details about the implement of spectral clustering. We use the weighted adjacency matrix W to describe the similarity graph. The steps of applying spectral clustering are described in Algorithm 1, shown in Figure 6.

This algorithm is effective in performing a good cut to the similarity graph and resulting clusters of rules. To understand what this algorithm really does besides all these matrices computations, and why these computations are rational to perform clusters, Formula 9 explains the reason.

Algorithm 1: ***Spectral Clustering***

Input: Weighted graph of rules W.

Output: Clusters of rules produced by spectral clustering $C_1,...,C_N$.

Step 1: Obtain the Laplacian matrix L. It is defined as $L = D - W$, where D is the degree matrix of W.

Step 2: Compute the eigenvalue and eigenvector of L.

Step 3: Get the smallest k eigenvalues and the corresponding eigenvectors, denoted as $u_1,...,u_k$.

Step 4: Let M be the matrix containing the vectors $u_1,...,u_k$ as columns. In this case of policy set, the ith row in M indicates the feature vector of the ith rule. The numerical values in a feature vector contains the information of features of a rule, which means the cluster of rules can be done based on these features.

Step 5: Cluster the rules with K-means algorithm based on M. The output clusters $C_1,...,C_N$ are the final result of the second phase clustering.

Figure 6 Algorithm for Spectral clustering

$$RatioCut(C_1, \ldots, C_k) = \frac{1}{2} \sum_{i=1}^{k} \frac{W(C_i, \bar{C}_i)}{|C_i|} \tag{9}$$

For clusters of rules C_1, …, C_k produced by a particular clustering algorithm, the quality or reliability of its clustering output is measured by function *RatioCut*. A good clustering plan has a relatively big value of *RatioCut*. This function is straightforwardly defined to evaluate a clustering plan, yet there is still difficult to understand the definition. The expression $W(C_i, \bar{C}_i)$ is defined as the sum of weights on edges which across C_i (the ith cluster) and all other clusters $\bar{C}_i$.

For example, in Figure 7, to compute the sum of weights on edges across cluster 2 and the other two clusters, we have the computation which is shown in Formula 10.

$$W(C_2, \bar{C}_2) = w[A, C] + w[B, C] + w[F, G] + w[F, H] \tag{10}$$

This example illustrates how to calculate $W(C_i, \bar{C}_i)$. Starting from Cluster 2, there are four edges highlighted in red, linking to nodes in other clusters. $W(C_i, \bar{C}_i)$ equals the sum of the weight on these edges.

As is already discussed, weight on edges linking two clusters is supposed to be as small as possible, which means the sum of weight on such edges, as numerator in *RatioCut*, is expected to be small.

To ensure every cluster has a reasonable number of rules, which is also the goal of clustering rules, for if too many clusters are produced with few rules in each one of clusters, the efficiency will drop to the case of no clustering at all. So the *RatioCut* has a denominator defined as the number of rules in a cluster C_i.

The problem amounts to find a possible clustering plan with the smallest *RatioCut*. But this is still an extremely difficult task if a method of simply traversing all possible clustering cases to find the smallest *RatioCut* is used. The spectral clustering algorithm describes that the function optimization problem is strongly connected to the eigenvalue of Laplacian matrix. By decomposing the Laplacian matrix and computing eigenvalues, the problem of finding optimizing clusters is transformed to cluster a few eigenvectors of the Laplacian matrix.

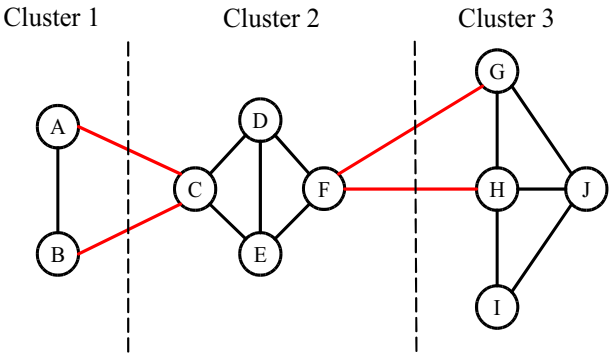

Figure 7 An example of calculating $W(C_i, \bar{C}_i)$

4.2.3 Centroid of cluster

To identify the most likely applicable cluster of rules for a coming request, we compare the similarity of the request with all clusters to find the most similar cluster. But doing this is not practical since the cost of comparing all clusters is as much as the cost of evaluation process in Sun PDP. It is ideal to compare only one or few rules in every cluster to determine the applicable cluster, which is the centroid of a cluster. The centroid of a cluster is a particular rule which has the maximum similarity to every rule in that cluster. The centroid of a cluster shall be as representative as possible. Given a cluster of rules produced by spectral clustering, which all have the same *subject*, we can find the value of each attribute that most frequently appears in all rules of the cluster. For example, in a cluster, the value that appears for the most times in *action* is *Read*, so that the centroid of this cluster has value *Read* for *action*. Other attributes in the centroid can be obtained in the same way.

5 Distributed PDPs

To further improve the evaluation efficiency, we propose a distributed PDP model to support the new approach by using clustered policies. Requests are dispatched by the *Request Dispatch Center* to the applicable PDP according to their *subject*. Note that for the rules whose *subject* appears for only few times in the whole policy set, they are gathered as a singleton and are assigned to a singleton PDP. The distributed PDP model is described in Figure 8.

When a sub PDP receives its request, it calculates the request and determine which policy is most possibly applicable to the request. This is accomplished by calculating the similarity between requests and centroids. The two-phase dispatch finally has refined the rules to a relatively small subset for evaluation. Within a sub PDP, for most cases, the coming request is only compared with most probably applicable set of rules, not the whole policy set until the *effect* is determined. Comparing to the traditional PDP, where the most time delay is spent on

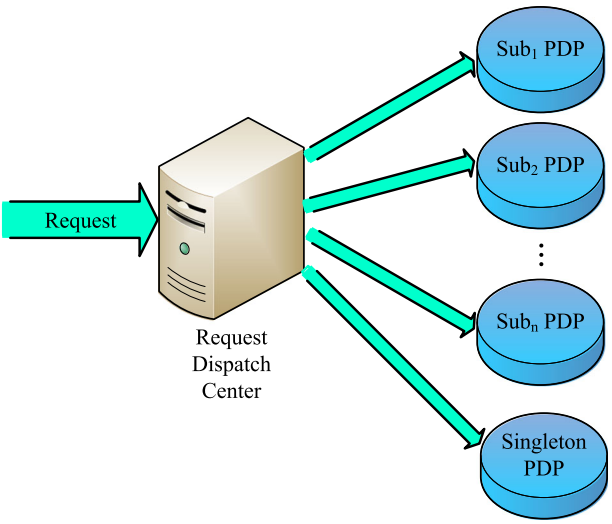

Figure 8 Distributed PDP model

sequentially compare of rules, the sub PDP benefits from clustering to reduce the size of rules that are to be compared.

It is worthwhile to mention that every Sub PDP still holds a policy with complete three-level of elements (policy set, policy and rule), therefore is compatible with XACML standards. Specifically, when policy is assigned to a Sub PDP, the high-level element (policy set and policy) are taken directly from the original policy, with a portion of rules partitioned by clustering. This uniformity with standards ensures optimizing evaluation without violating other mechanism for protecting security, privacy, etc. Figure 9 shows an example of constructing two Sub PDP from a single PDP in view of their policy.

We propose a new framework for request evaluation of individual sub PDPs as shown in Figure 9.

In Figure 10, the *Similarity Degree Calculator* computes the similarity of the request and all centroids. The *Request Transponder* then, according to these similarity degrees, sends the request to a particular *Rule Matcher*, whose rules correspond to the centroid with the highest similarity degree with the request. Some details are introduced below.

- The *Similarity Degree Calculator* is the component which picks up the most likely cluster of rules for the request. The decision is made based on the features of request and the set of centroids of clusters, which is representative enough to indicate the most obvious features of rules within its cluster. Centroids are calculated in section 4.2.3. Actually, it is a specific node in the graph which is the most tightly linked by the nodes around it, therefore such nodes among all the nodes in the same cluster are the most representative of their features, in this case, the value of attributes. The highest similarity degree denotes that this cluster of rules is the most likely set of rules that are applicable to the present request. Reviewing the definition of *RSD*, the similarity degree between the request and centroids of clusters of rules is similarly defined, whereas the attributes to be compared come from centroids (a rule) and a request instead of another rule.
- The *Request Transponder* uses the similarity degree from *Similarity Degree Calculator* to determine a certain *Rule Matcher* to evaluate. The *Rule Matcher* whose set of rules most likely applicable to the request is chosen.
- The *Rule Matcher* is the only component in the whole system that does the traditional evaluation. The policy on which the *Rule Matcher* is based is relatively small in volume therefore fast in comparing attributes.

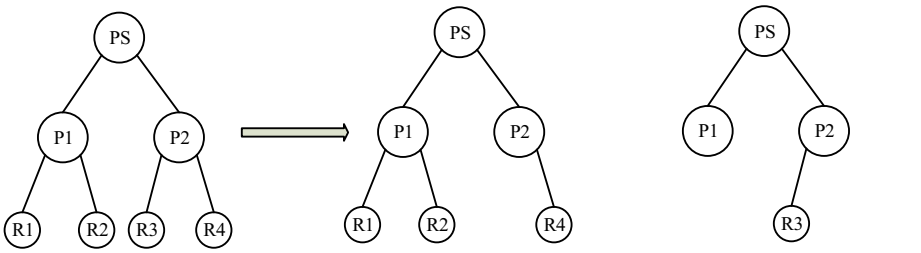

Figure 9 Construct two Sub PDPs from a single PDP

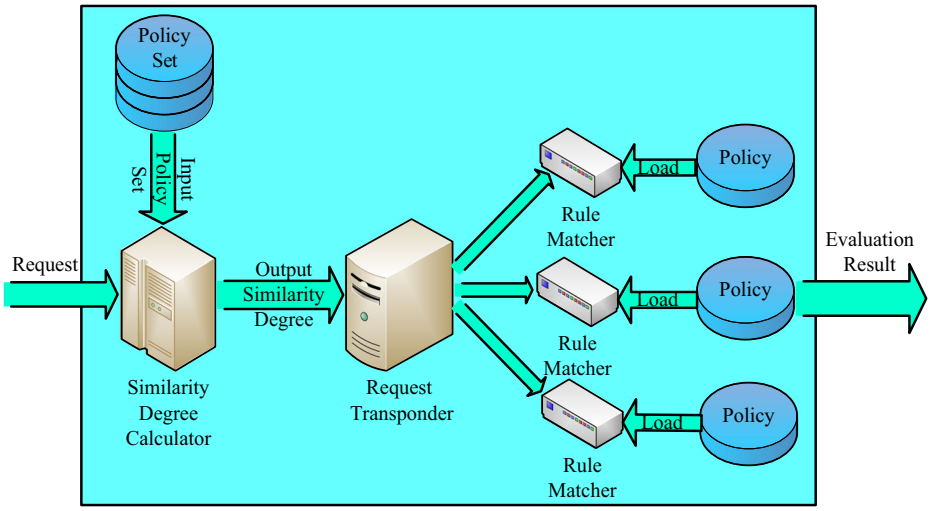

Figure 10 Process of evaluating one request within Sub PDPs

The first phase clustering based on the attribute *subject* greatly decreases the evaluation of inapplicable rules. However, the efficiency of this evaluation engine suffers in the second phase evaluation because the second phase clustering is not as precise as the formal clustering. The slight deviation of evaluation is brought by two operations:

1) *Spectral clustering*. Consider that a node is linked by two clusters which are symmetrical in layout and the weight of such a node on both sides is equal. The partitioning may divide this node to the left or right cluster. However, in evaluation, this will possibly intrigue an absence of rules because some rules are too uncertain to make a precise cut.
2) *Similarity Degree Calculator*. Consider that a request is ambiguous in determining which *Rule Matcher* is applicable, that is, the similarity degree of the request and more than one set of rules are too close to accurately choose a *Rule Matcher*.

These problems are resulted by the roughness of clustering itself. In many applications, this slight deviation might be neglectable, for example the image identification. But the PDP evaluation will suffer because the rearranged order of clusters might make it worse than no sorting condition. For instance, the only applicable rule might be in the last cluster to evaluate after the rearrangement. We need experiments to find out the exact improvement of PDP evaluation performance and the chance for such a situation to happen.

6 Experimental results and analysis

By implementing our distributed PDP model, called XPDP, and doing experiments on it as well as other models, we manage to find out the actual improvement of evaluation efficiency, and also the actual outcome of spectral clustering and its impact on evaluation. We design the following three experiments.

1) A comparison of the evaluation time of processing different number of requests ranging from 2000 to 10,000 on the Sun PDP, SBA-XACML and XPDP, using policy sets *VMS* [26], *LMS* [18] and *ASMS* [19] respectively.
2) A comparison of the average number of rules evaluated for one request using three policy sets on the Sun PDP, XPDP (with one-stage clustering) and XPDP (with two-stage clustering), using 10,000 synthetic requests.
3) In order to test whether the applicable rules are effectively gathered and evaluated early, we present the statistical feature by counting the number of requests which stop at each cluster that is sequentially evaluated, using 10,000 synthetic requests.

The experiments are carried out on a laptop computer running Ubuntu 17.04, with Intel Core i5-4200H 2.8GHz processor and 8GB of RAM. Our proposed XPDP has been implemented purely in C++. The performance of XPDP is compared and analyzed with the Sun PDP [25] and SBA-XACML [20]. Although there is an available open source implementation of the Sun PDP based on Java, we still constructed a C++ version of the Sun PDP, in order to eliminate the extraneous effects of programming languages themselves as many as possible. The reason why we think it is better to experiment in C++ lies in its controllability with respect to the manual predictable garbage collection mechanism. Programmers have the ability of controlling such activities by using C++.

Since we concentrate mainly on the comparison of algorithm efficiency, we do not implement the process of XML analysis in the experiment, instead we simply use textual policy set and file read manipulation. Every request is also presented as pure strings in textual file.

6.1 Test policies

In order to simulate practical application scenarios, we select the following three XACML access control policies from practical systems:

- *Library Management System* (*LMS*) [26]: The *LMS* provides access control policies by which a public library can use Web services to manage books.
- *Virtual Meeting System* (*VMS*) [18]: The *VMS* provides access control policies by which Web conference services can be managed. The *VMS* allows users to organize online meetings in a distributed platform. When a user connects to the server, he/she can enter or exit a meeting, make a statement and ask questions at the meeting, etc. Every meeting has an administrator, whose responsibilities are initializing the meeting information and setting some parameters (such as the meeting's title and organization). The administrator can also assign to every meeting a host, who is capable of selecting a user to make a statement.
- *Auction Sale Management System* (*ASMS*) [19]: The *ASMS* provides access control policies by which items can be bought or sold online. A seller initializes the lowest price of and the description of an item which are allowed to be submitted when at auction. A User can participate in the bidding process by bidding the item. The restriction to a user is that there must be enough money in his/her account before bidding.

The policy of the *LMS* contains 720 rules, that of the *VMS* 945 rules, and that of the *ASMS* 1760 rules. In the light of actual requirement, the policies of the *LMS*, *VMS*, and *ASMS* need to

be expanded to contain more rules. What we do is that according to the Cartesian product of different subjects, actions, resources, and conditions in all rules of a policy [4], we construct new rules and add them to the original policy. The number of rules in the policies of the *LMS*, *VMS*, and *ASMS* is expanded separately to 3000, 6000, and 9000.

6.2 Generation of test requests

We analyze the policy set and obtained all distinct values ever appeared in all rules. These values consist a domain for every particular attribute in a rule. Then we randomly pick one value from each attribute domain and combine the values to form every single request. Another detail here in experiment is to avoid the timing of reading from disk, which is very slow comparing to the time used in evaluation and could severely interfere the results, so we first read all these requests into main memory as a *vector* < *request* > .

6.3 Performance tests and comparisons

We define the similarity of rules in section 4.2.1. In the second phase clustering, we must decide how many clusters of rules to be produced from spectrum clustering. Such clusters are used to sequentially evaluate the request. If there are only a few number of them, the efficiency of evaluation would suffer because the request may go through a great portion of rules. Say there are three clusters, then every rule must at least go through a third of all rules regardless of the distribution of rules within a cluster. Too many clusters are also not ideal, because every cluster has a centroid to evaluate aiming to decide the order of clusters to later evaluation. An extreme example would be to have the same number of clusters as rules. Obviously, the process of computing request-centroid similarity and the clusters sorting is too time-consuming, even no improvement comparing to brute force search. In our example of the definition of similarity in section 4.2.1, any pair of rules has a similarity degree from the domain consisted by totally eight distinct numbers. We consider this as a hint to produce eight clusters in our experiment. Viewing the policy sets in our experiment, after the first phase clustering, we have an average of 220 rules. These rules are put into eight clusters, with an average of 28 rules in every cluster.

6.3.1 Comparisons of evaluation time

We make experiments on our proposed XPDP, as well as two popular PDP models, the Sun PDP and SBA-XACML. We want to find out the exact improvement of our XPDP comparing to other PDPs. Since the time of processing requests is a critical measure of performance, and it has a significant impact on user experience, we choose the evaluation time as a target to compare. The experiments are carried out on three PDP models, using different policy sets and different number of requests. The experimental results are shown in the Figure 11. In the implementation of the XPDP, we use eight clusters in the second phase clustering, as is already discussed above, and we follow the approach above to obtain requests.

We learn from the Figure 11 that the XPDP has 8*N times of better time efficiency than the Sun PDP (N for the number of all distinct subjects) regardless of computing eight similarities between a rule and centroids. It is only achieved when always the first cluster is precise enough for evaluation, that is to say, there is at least one applicable rule in the first cluster so that the XPDP does not need to check the following clusters. The experiment shows less improvement

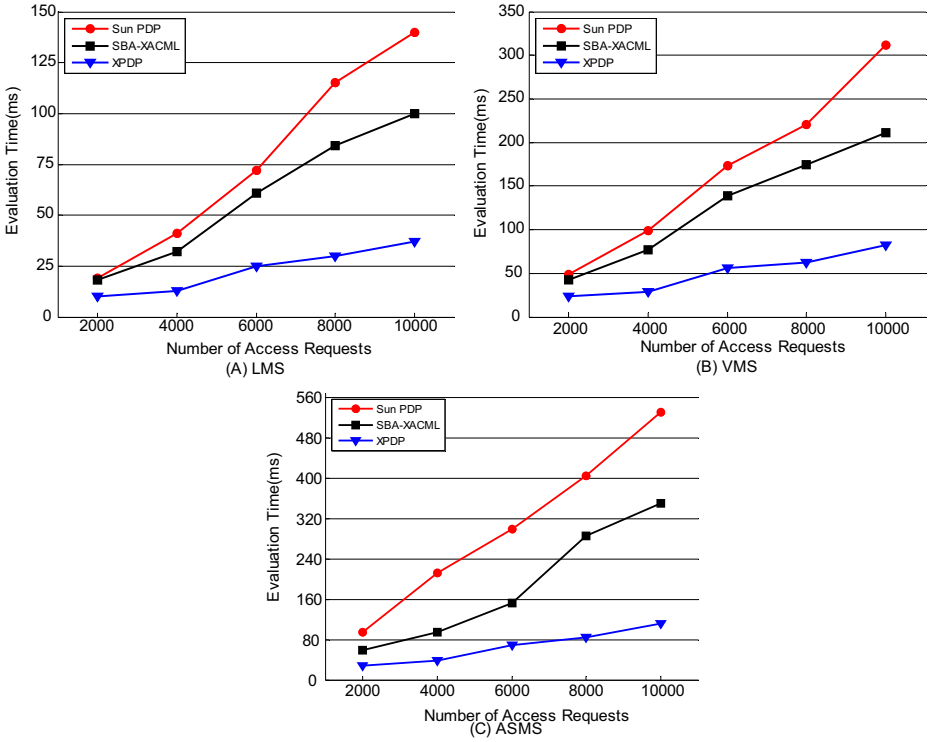

Figure 11 Total evaluation for different number of access requests

of efficiency. The XPDP model achieves 3.26 times better time efficiency comparing to the Sun PDP, and 1.85 times better time efficiency comparing to the SBA-XACML(see Table 1).

6.3.2 Comparisons of required rules to evaluate one request

By analyzing the internal processing, we obtain a hint to further improve the XPDP. Since the XPDP is based on the clustering and reordering to reduce the number of matched rules, it is straightforward to see the exact number of rules that is reduced by using the XPDP. We firstly make an experiment on the XPDP only with the first phase clustering, then we add the second phase clustering to it. This allows to illustrate the impact of two clustering steps respectively. We use 10,000 synthetic requests to test the XPDP.

Table 1 Evaluation time and improvement ratio on 10,000 access requests

Policy	# of Rules	Processing time (ms)			XPDP's improvement ratio (%)	
		Sun PDP	SBA-XACML	XPDP	Sun PDP	SBA-XACML
LMS	3000	139	100	34	309	194
VMS	6000	314	202	81	288	149
ASMS	9000	536	347	111	383	213

Average Improvement Ratio: 326% (compare Sun PDP), 185% (compare SBA-XACML)

Using clustering significantly reduces the number of rules evaluated for an access request. The Figure 12 shows that by adopting the first phase clustering, the XPDP needs 4.6 times less of rules and 13.8 times less of rules when using the second phase clustering. For example, In the *ASMS* policy set, the Sun PDP needs to evaluate 4099 rules, but in the XPDP with two-stage clustering, it only needs 296 rules. The improvement from the XPDP with one-stage clustering to two stage-clustering is not as significant as from the Sun PDP to the XPDP with one-stage clustering. Theoretically, if all requests stop at the very beginning of all eight clusters, the improvement should be near 8 times less of rules, in contrast with only 3 times less of rules. The experiment in 6.3.3 shows the number of requests which stop at each cluster of the total eight clusters, and it gives explains to this discrepancy.

After experimenting on three policy sets, we drew solid lines between the data points on Figure 12. Through the relatively smooth line, it shows that with the increasing volume of test policies (expanded *LMS*, *VMS* and *ASMS* has 3000, 6000 and 9000 rules respectively), the average number of rules evaluated for one request approximately satisfies linear growth. This result supports our PDP model to efficiently work in other conditions with even larger volume of policies.

6.3.3 Statistical features of evaluation process based on spectral clustering and reordering

By reordering the groups of rules produced in the second phase clustering, the XPDP arranges the particular group that contains the applicable rule to be evaluated ahead of other groups. We make an experiment to find out how effective it works by arranging the groups queue. We test the XPDP with 10,000 requests and record how many requests stop at the first cluster, the second cluster, and so on. By saying "stop at a cluster", it means this cluster contains the applicable rules to the request and the evaluation decision is made at this cluster. The policy sets and the approach to obtain requests are identical with the previous experiments (Figure 13).

It is clear to see that most requests stop at the few front clusters of rules. The first and second clusters are enough for half of the requests, making the following six clusters

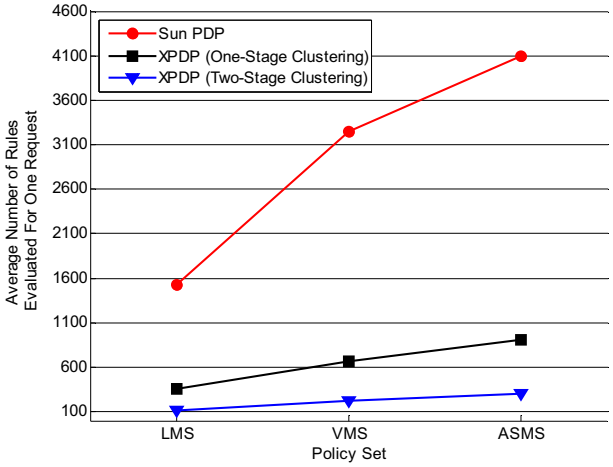

Figure 12 Average number of rules evaluated for one request

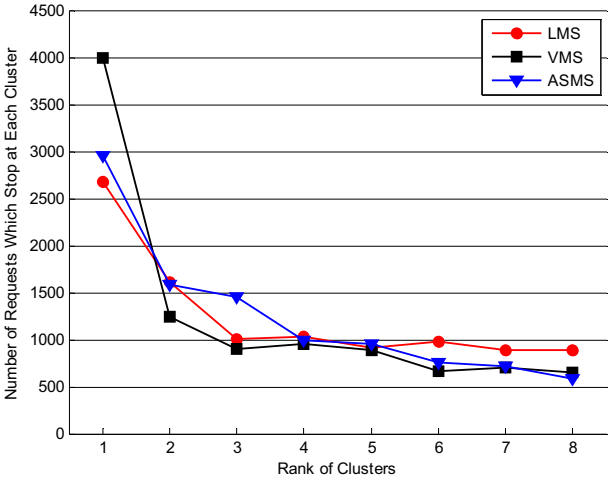

Figure 13 Number of requests which stop at each cluster

unnecessary to be evaluated. This experiment proves that the second phase clustering indeed improves the efficiency by gathering the most likely applicable rules and let these rules to be evaluated ahead of other rules.

7 Conclusions and future work

We introduce the present situation of XACML policy evaluation, which is heavily used in many access control applications, and we come to realize that the present solution faces serious challenge in terms of processing speed and efficiency. In order to alleviate present bottleneck, we propose a new PDP framework, which consists of two stages of rule clustering and a distributed PDP architecture. It is intuitive to think that by clustering rules, the number of comparison between rules and requests is reduced, hence improves the evaluation performance. The two-phase clustering aims to effectively divide the policy set into small groups. The second phase clustering is based on similarity of rules, and after clustering it also supports ordering of such groups of rules for evaluation. It combines the advantages of both ordering and clustering, while in the same time it requires no cost of time on clustering (it is preprocessed and the clustering results are fixed) and little cost of time on ordering. Experiments show that our framework obtains significant improvement over the widely used Sun PDP.

In the future, we are going to find a better measurement of similarity degree, establishing a more accurate relationship-graph for rules. The present definition of similarity degree is mainly based on string comparison, which is too simple to handle complicated situations, such as multi-value attributes and conditions of a numeric range. For example, assume that a rule which has *condition* as $18 < age < 30$, and the other two rules which has *condition* as $10 < age < 20$ and $18 < age < 40$ respectively, the current definition of similarity degree would regard these two rules with the first rule as equally dissimilar. The definition of similarity degree influences the ultimate improvement of efficiency, so it shall be more carefully defined to satisfy more complicated situations with

different features of policy set. From another perspective, we could preprocess the policy set and eliminate both redundant and conflict rules from the policy set, which could further improve the PDP evaluation performance [3].

Acknowledgments This work is supported by the scientific research cultivation fund of Xi'an University of Science and Technology in China (201635), the PhD research startup foundation of Xi'an University of Science and Technology in China (2015QDJ072), the natural science foundation of Shaanxi province in China (2017JQ6053), and the national natural science foundation of China (61702408). This work was also supported by the Innovation Group for Interdisciplinary Computing Technologies, College of Computer Science and Technology, Xi'an University of Science and Technology.

References

1. Borders, K., Zhao, X., Prakash, A.: CPOL: high-performance policy evaluation. In: Proceedings of International Conference on Computer and Communications Security, 147–157, ACM (2005)
2. Bui, T., Stoller S.D., Sharma, S.: Fast distributed evaluation of stateful attribute-based access control policies. In: Proceedings of International Conference on Data and Applications Security and Privacy, 101–119, IFIP (2017)
3. Deng, F., Zhang, L.Y.: Elimination of policy conflict to improve the PDP evaluation performance. J. Netw. Comput. Appl. **80**(4), 45–57 (2017)
4. Deng, F., Zhang, L.Y., Zhou, B.Y., Zhang, J.W., Cao, H.Y.: Elimination of the redundancy related to combining algorithms to improve the PDP evaluation performance. Math. Probl. Eng. **2016**(4), 1–18 (2016)
5. Hughes, G., Bultan, T.: Automated verification of access control policies using a SAT solver. Int. J. Softw. Tools Technol. Transfer. **10**(6), 503–520 (2008)
6. Jebbaoui, H., Mourad, A., Otrok, H., Haraty, R.: Semantics-based approach for detecting flaws, conflicts and redundancies in XACML policies. Comput. Electr. Eng. **44(C)**, 91–103 (2015)
7. Kabir, M.E., Wang, H., Bertino, E.: A role-involved purpose-based access control model. Inf. Syst. Front. **14**(3), 809–822 (2012)
8. Kolovski, V., Hendler, J., Parsia, B.: Analyzing Web access control policies. In: Proceedings of International Conference on World Wide Web, 677–686, ACM (2007)
9. Lin, D., Rao, P., Bertino, E., Lobo, J.: An approach to evaluate policy similarity. In: Proceedings of ACM Symposium on Access Control Models and Technologies, 1–10, ACM (2007)
10. Lin, D., Rao, P., Ferrini, R., Bertino, E., Lobo, J.: A similarity measure for comparing XACML policies. IEEE Trans. Knowl. Data Eng. **25**(9), 1946–1959 (2013)
11. Liu, T., Wang, Y.: Beyond scale: an efficient framework for evaluating Web access control policies in the era of big data. In: Proceedings of International Workshop on Security, 316–334, (2015)
12. Liu, A.X., Chen, F., Hwang, J.H., Xie, T.: Xengine: a fast and scalable XACML policy evaluation engine. In: Proceedings of ACM SIGMETRICS Performance Evaluation Review, 265–276, ACM (2008)
13. Liu, A.X., Chen, F., Hwang, J.H., Xie, T.: Designing fast and scalable XACML policy evaluation engines. IEEE Trans. Comput. **60**(12), 1802–1817 (2011)
14. Lorch, M., Proctor, S., Lepro, R., Kafura, D., Shah, S.: First experiences using XACML for access control in distributed systems. In: Proceedings of ACM Workshop on XML Security, 25–37, ACM (2003)
15. Luxburg, U.V.: A tutorial on spectral clustering. Stat. Comput. **17**(4), 395–416 (2007)
16. Marouf, S., Shehab, M., Squicciarini, A., Sundareswaran, S.: Statistics & clustering based framework for efficient XACML policy evaluation. In: Proceedings of International Conference on Policies for Distributed Systems and Networks, 118–125, IEEE (2009)
17. 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), 300–313 (2011)
18. Mouelhi, T., Fleurey, F., Baudry, B., Traon, Y.: A model-based framework for security policy specification, deployment and testing. In: Proceedings of International Conference on Model Driven Engineering Languages and Systems, 537–552, (2008)
19. Mouelhi, T., Traon, Y.L., Baudry, B.: Transforming and selecting functional test cases for security policy testing. In: proceedings of international conference on software testing, verification, and validation, 171–180, IEEE (2009)
20. 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)

21. Ng, A.Y., Jordan, M.I., Weiss, Y.: On spectral clustering: analysis and an algorithm. Proc. NIPS. **14**(2001), 849–856 (2001)
22. Ngo, C., Demchenko, Y., Laat, C.D.: Decision diagrams for XACML policy evaluation and management. Comput. Secur. **49**(5), 1–16 (2015)
23. Pei, X., Yu, H., Fan, G.: Achieving efficient access control via XACML policy in cloud computing. In: Proceedings of International Conference on Software Engineering and Knowledge Engineering, 110–115 (2015)
24. Ros, S.P., Lischka, M.: Graph-based XACML evaluation. In: Proceedings of ACM Symposium on Access Control Models and Technologies, 83–92, ACM (2012)
25. Sun's XACML implementation: http://sunxacml.sourceforge.net/
26. Traon, Y.L., Mouelhi, T., Pretschner, A., Baudry, B.: Test-driven assessment of access control in legacy applications. In: proceedings of international conference on software testing, verification, and validation, 238–247, IEEE (2008)
27. Turkmen, F., Demchenko Y.: On the use of SMT solving for XACML policy evaluation. In: Proceedings of International Conference on Cloud Computing Technology and Science, 539–544, IEEE (2016)
28. Wang, H., Cao, J., Zhang, Y.: A flexible payment scheme and its role-based access control. IEEE Trans. Knowl. Data Eng. **17**(3), 425–436 (2005)
29. Wang, H., Zhang, Y., Cao, J.: Access control management for ubiquitous computing. Futur. Gener. Comput. Syst. **24**(8), 870–878 (2008)
30. Wang, Y.Z., Feng, D.G., Zhang, L.W., Zhang, M.: XACML policy evaluation engine based on multi-level optimization technology. J. Softw. **22**(2), 323–338 (2011)