Int. J. Fuzzy Syst. (2017) 19(1):62–77
DOI 10.1007/s40815-016-0147-3

An Efficient Fuzzy-Based Hybrid System to Cloud Intrusion Detection

Sivakami Raja[1] · **Saravanan Ramaiah**[2]

Received: 20 August 2015 / Revised: 18 November 2015 / Accepted: 16 January 2016 / Published online: 5 February 2016

Abstract Cloud is a collection of resources such as hardware, networks, servers, storage, applications, and interfaces to provide on-demand services to customers. Since access to cloud is through internet, data stored in clouds are vulnerable to attacks from external as well as internal intruders. In order to preserve privacy of the data in cloud, several intrusion detection approaches, authentication techniques, and access control policies are being used. The common intrusion detection systems are predominantly incompetent to be used in cloud environments. In this paper, the usage of type-2 fuzzy neural network based on genetic algorithm is discussed to incorporate intrusion detection techniques into cloud. These systems are intelligent to gain knowledge of fuzzy sets and fuzzy rules from data to detect intrusions in a cloud environment. Using a standard benchmark data from a cloud intrusion detection dataset experiments are done, tested, and compared with other existing approaches in terms of detection rate accuracy, precision, recall, MSE, and scalability.

Keywords Clustering algorithms · Fuzzy neural networks · Genetic algorithm · Hybrid intelligent systems · Intrusion detection

✉ Sivakami Raja
rsivakami@psnacet.edu.in

Saravanan Ramaiah
directorrvsetgi@rvsgroup.com

[1] Department of Information Technology, PSNA College of Engineering and Technology, Dindigul, Tamilnadu, India

[2] Department of Computer Science and Engineering, RVS Educational Trust's Group of Institutions, Dindigul, Tamilnadu, India

1 Introduction

In this present era of computing, information is a major asset for every organization. From the local area network to the currently available highly connected internet, the world is being benefited from the easiness of data storage and access. With the introduction of cloud computing, maintenance of data has also become a simple task. However, this flexibility introduces a problem of data security as a major issue. This is due to the fact that intruders and hackers are also enjoying technologies for their security-threatening activities. And also, by subscribing to the cloud, cloud consumers permit external sources to hold control of their data. This creates room for hurdles. So data security and privacy are very serious problems in the victory of an industry process. Hence, organizations are using different solutions to achieve data security.

An efficient intrusion detection system should be fast, self-monitored, fault tolerant, easily configurable, difficult to cheat, available without interruption, and free from false errors with an overhead as minimum as possible. Its aim is to evaluate information systems and to perform early detection of malicious activity for reducing the security risk to an acceptably low level. High false-positive alarm rate may disrupt information availability, whereas high false-negative alarm rate may result in serious damage to the protected systems in the form of inappropriate access to sensitive information and data damaging. The performance of IDS is based on the amount of sufficient log data, its continuous updates on them, and the correct and quick detection of intrusion from the comparison between current activity of the user and the historical data.

The rest of this paper is organized as follows: Sect. 2 outlines IDS in cloud. The concept of genetic algorithm-based fuzzy neural architecture in intrusion detection is

described in Sect. 3 and our proposed method is described in Sect. 4. Results of experiments are presented and analyzed in Sect. 5. At last, Sect. 6 concludes our work.

2 Intrusion Detection in Cloud

For the purpose of achieving a green world and also due to the invention of new technologies, paper-work is vanishing from existence and the concept of digital data is introduced and is gaining importance. But these digital data may require hardware, software, and networks for their storage and access which may lead to practical complexities. A potential solution to these issues is a cloud storage service. This service offers several benefits including user-friendly access, maintenance and sharing of large volume of data, synchronization of various devices, and more importantly, cost efficiency. Large amount of sensitive information can be stored on computers. This information stored on the cloud may be precious to others with malicious motive. So cloud providers follow various methodologies to solve these issues. But, any unauthorized element including a provider should not gain access to cloud storage, since a small damage or loss of data may be a potential harm to its owner. Firewalls and encryption are the most commonly used technologies for providing security and privacy. But they may not be of enough efficiency to protect data against intrusions. However, they can be used at the earlier part of the intrusion detection process.

Machine learning is one of the methods adopted to guide the system for detecting intrusions. A grid and cloud computing intrusion detection system is proposed in [1] to detect intrusions by employing an artificial neural network for teaching the system. An IDS with the central management approach [2] has been developed to address heterogeneity and virtualization features of cloud computing. In [3], a model based on hypervisor has been proposed for protecting the system from diverse attacks in the infrastructure layer which improves the system. An individual IDS is suggested in [4] where a single controller manages the instances of IDS, exploiting the knowledge-base and ANN pattern matching techniques. The fully distributed intrusion detection system developed in [5] implements a hybrid approach for detecting intrusions using host and network-based audit information for cloud computing. Numerous cloud intrusion detection techniques function on different layers of clouds independently [6].

Autonomic clouds have emerged as an outcome of integrating autonomic computing and cloud computing, yielding in stable and friendly cloud architectures and deployments. So they have contemporarily fascinated researchers to compose cloud intrusion detection engines with negligible human intervention. Network virtualization concept is handled in [7] for a cloud-based intrusion detection and prevention system without any control over the host. Table 1 portrays the drawbacks of current intrusion detection solutions, whereas our proposed method accomplishes the following benefits:

2.1 Clustering and Concept Learning

Based on mentioned criteria, grouping of elements is done through which the system can be trained to classify intrusion detection dataset into five groups (one group of normal

Table 1 Existing intrusion detection approaches

Approach	Drawback
Snort	Slow in processing packets
Suricata	Increased resource consumption and false-positive rate
Bro IDS	Requires expertise technical knowledge
Security Onion	Possesses the disadvantages of all its constituent technologies
Next-generation intrusion detection expert system (NIDES)	Unsatisfactory performance in observing collaborative and long-duration penetration attacks
Distributed intrusion detection system (DIDS)	If Resource access is accomplished via an unsupervised domain, two related sessions of the same user cannot be mapped
State transition analysis tool (STAT) and USTAT	Weak in detecting DoS and masquerading attacks. Moreover, it requires an additional rule-based misuse/anomaly intrusion detection system
Tripwire	If used alone, it cannot provide complete security. So it has to be used along with other security-related tools
Graph-based intrusion detection system (GrIDS)	If used alone, it cannot provide complete security
Thumbprinting	Utilization of various encryption methodologies is restricted

patterns and four groups of intrusive patterns). This is achieved through data reduction, data analysis, and data classification.

2.2 Predictive Learning

From the temporal intrusion detection dataset and several sequences of resource access events, the system learns intrusive and non-intrusive behaviors of users. This is done by understanding even small changes that occur in the behavior of users, which could be impossible in expert systems.

2.3 Modularity

We have used the structured methodology for intrusion detection with four phases, where each phase gathers its input from its previous phase. However, the processing of each phase is completely isolated from the remaining phases which could avoid unexpected operational and configuration errors. Its modular nature makes system development, system debugging, and system administration simple and dynamic.

3 Fuzzy Neural Network Based on Genetic Algorithm

An artificial neural network (ANN) is a way of reasoning inspired by principles of neurons in the central nervous system of living things. It is a set of one or more layers of nodes interconnected by directed, weighted links, and trained with desired input–output patterns through which it can learn weights on the links that provide the correct outputs for the training and similar inputs. So it can be used to create a decision support system (DSS) when sufficient training examples exist. Fuzzy logic is a technique for the representation and handling of imprecise and vague information through a fuzzy rule-based system. In spite of this benefit, it typically consumes much time for designing and tuning the membership functions. Moreover, it does not contain a competent learning algorithm to minimize output errors. So neural network can be used for this same task to get benefited from learning, adaptation, fault tolerance, parallelism, and generalization. But it suffers from relatively slow learning process. To complement each of the above individual systems with their advantages, fuzzy neural systems or neuro-fuzzy models were developed as intelligent hybrid systems which are successful in several real-world applications.

Neural fuzzy systems may take up type-1 fuzzy sets, which characterize uncertainties using numbers in the range [0, 1]. But, it needs each and every element of the universal set to be associated with a specific real number. On the other hand, type-2 fuzzy sets model uncertainties by expressing their membership functions as type-1 fuzzy sets. They are of three dimensions in which the last dimension is the membership degree of each point on its two-dimensional domain. A type-2 fuzzy neural network (T2FNN) incorporates a type-2 fuzzy process as its antecedent and a two-layer neural network as its consequent [8]. A general T2FNN is complex due to the complication of type reduction process and they are not easy to follow for a variety of reasons. An algorithm for computing the centroid of an interval type-2 fuzzy set is developed in [9]. Here, the third dimension does not contribute any new information since its value is same in all points. Hence, the third dimension is disregarded. Further, making inference from interval type-2 fuzzy set is simple. Therefore, the interval T2FNN (IT2FNN) [10] can be adopted to simplify the computational process.

In the field of intrusion detection, the use of FNN alone is satisfactory for detecting repeated or more-frequent attacks, such as DoS and probing attacks. But it cannot effectively achieve high detection rates for less-frequent attacks, such as R2L and U2R attacks. When K-NN is used alone for anomaly intrusion detection, only 91 % accuracy is achieved in [11]. This inconsistency occurs due to the difference in number of records available in an intrusion detection dataset, related to the more-frequent and less-frequent attacks. Since more-frequent attacks have loads of records in an intrusion detection dataset, FNN learns them accurately and produces exact results. But due to the small number of records corresponding to the less-frequent attacks, FNN cannot be trained accurately for their detection. So it is essential to combine FNN with another methodology which can compensate this crisis.

Genetic algorithm (GA) is a global optimal search approach that makes use of operations found in natural evolution. They work sound on mixed-value, combinatorial problems and offer a suitable technique to problems that require an efficient searching. They are less vulnerable to getting caught at local optima than gradient search methods. But, if genetic algorithm is used independently in intrusion detection, it too does not provide satisfactory results. But in [12], GA produces improved detection rate for U2R attacks which motivated us to combine GA with NN. So GA can be combined with fuzzy neural networks for achieving satisfactory detection rates in both more- and less-frequent attacks by refining the structure and/or parameters of the underlying FNN. The fundamental concept is to preserve a population of chromosomes for symbolizing candidate solutions to the problem being studied. This population matures by time in the course of selection, crossover, and mutation to find the best one(s). In [13], an iterative approach is suggested where each individual produces one rule.

There are various GA-based algorithms to evolve the parameters of membership functions and the number of fuzzy rules. For a GA-based algorithm employed in fuzzy neural networks, the initial network structure is more often given initially. Then GA is adopted for enhancing the systems topology through membership functions and/or fuzzy rules. A fuzzy neural network is used in [14] to extract Mamdani-type fuzzy rules [15] from the input–output space in three stages. But this approach requires fuzzy partitions of input and output spaces in advance. The parameters of membership functions and the number of fuzzy rules are developed in [16] by encoding them to form a complex and long chromosome. But, the number of fuzzy sets for each variable must be provided as earlier information and needs the maximum acceptable number of fuzzy rules.

4 Proposed Intrusion Detection System

In our work, a cloud intrusion detection system (CIDS) is designed for infrastructure-as-a-service (IaaS) model for achieving cloud user security. This system is modeled as a Fuzzy Neural Network based on Genetic Algorithm. Figure 1 shows our proposed system of intrusion detection in cloud environment. This proposed approach consists of four phases, namely Clusters formation, Extraction of initial fuzzy rules, Fuzzy rulebase optimization, and Parameters refinement. A method based on GA is applied for fuzzy rulebase optimization and a neural fuzzy system with dynamical optimal learning algorithm is used for parameters refinement. The completion of phase IV results in the formation of rulebase which can be used for detecting intrusions in real time. So, whenever the cloud user tries to access the service, his/her pattern of activity is audited across fuzzy rulebase. Inference from this comparison is used in making decisions regarding intrusions and normal activities.

4.1 Clusters Formation

Fuzzy rule extraction methods depend on clustering in the dataset. Each cluster should effectively classify a region in the input data space that can enclose a sufficient amount of data to hold the presence of a fuzzy input/output relationship. The Fuzzy C means algorithm [17] and Kohonen's Self-Organizing Map [18] are familiar clustering algorithms. But their goodness relies upon the initial values of the number of cluster centers and their locations. Mountain method [19] is proposed for their estimation. In spite of its simplicity and effectiveness, it has exponential growth of computation with the size of the problem. In subtractive clustering [20], without any initial guess, all data points are regarded as cluster centers. But the trouble is that at times the real cluster center may not be pinpointed to one of the data points. And also, if the neighborhood radius is too small, the effect of neighboring data points will be ignored. If it is large, it will include all data points in its neighborhood. Still, this method provides a high-quality approximation with shortened computations. K-means clustering [21] achieves better accuracy than other clustering techniques. FCM clustering also achieves accuracy close to K-means clustering. But it is computationally slower due to the complications involved in its fuzzy measures calculations. In our work, for phase I of Fig. 1, we use modified K-means algorithm [22] to separate the training dataset into several clusters through symmetry distance between patterns. The similarity between each pattern and each existing cluster is computed to determine whether the pattern has to be associated with an already existing cluster or with a new cluster. To create new clusters, Minkowski distance [23] is used rather than Euclidean distance used in [22].

Let the training dataset contain k number of records $a_1, a_2, \ldots, a_k$ where each record a_j has r patterns $[p_{1j}, p_{2j}, \ldots, p_{rj}]$. Each record a_j, $1 \le j \le k$, indicates an attack type and is associated with an output d_j. This d_j denotes the type of an attack that should be detected successfully for an input record a_j during training and testing. Let n be the number of clusters, where clusters are represented by $C_1, C_2, \ldots, C_n$. Initially no clusters exist. Through the process of training only, clusters are formed based on the similarity between records. Each C_i has mean $m_i = [m_{1i}, m_{2i}, \ldots, m_{ki}]$, deviation $\rho_i = [\rho_{1i}, \rho_{2i}, \ldots, \rho_{ki}]$, and a height h_i that is the mean of expected outputs of patterns enclosed in C_i. Let S_i be the number of training patterns available in C_i. The following is an algorithm used in our work for cluster formation:

1. Select K patterns at random and initialize their cluster centroids as $C_1, C_2, \ldots, C_k$.
2. Use traditional K-means clustering for initializing centroids of all clusters.
3. When they converge or a prespecified terminating condition is fulfilled, goto next step.
4. For pattern x, find the cluster centroid by $k^* = \arg\min_{i=1}^{K} d_S(x, C_i)$, where $d_S(x, C_i)$ is the symmetry distance between a pattern x and C_i.
5. If this calculated distance is less than a prespecified distance, assign x to C_{k^*}.
6. Else assign x to the cluster C_{k^*}, by calculating $k^* = \arg\min_{i=1}^{K} d_M(x, C_i)$ where $d_M(x, C_i)$ is the Minkowski distance between x and C_i.

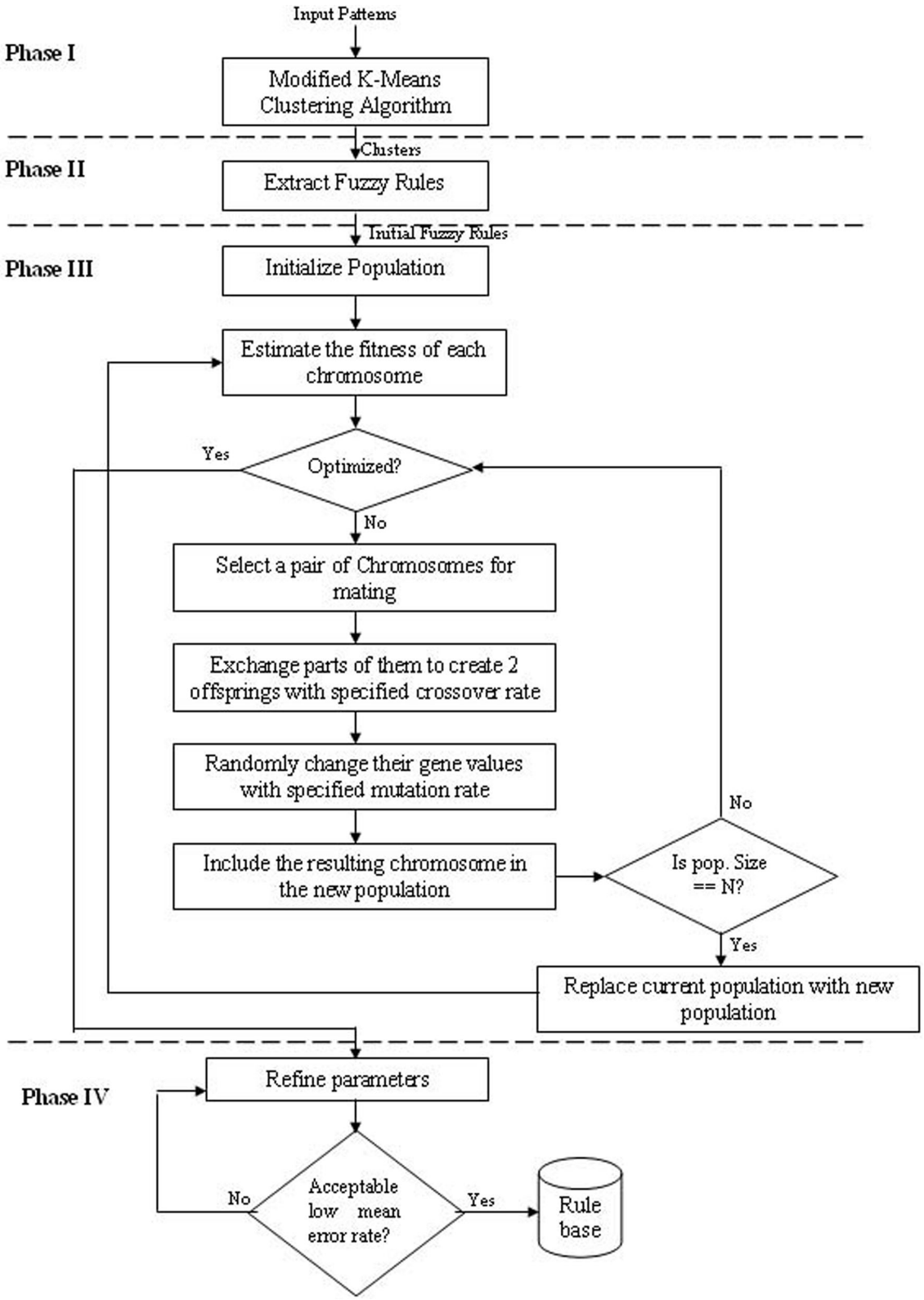

Fig. 1 Proposed model for cloud intrusion detection system

7. Calculate the updated centroid of K clusters by $C_k(t+1) = \frac{1}{N_k} \sum_{i \in U_k(t)} x_i$. Here, at time t, $U_k(t)$ is a collection of patterns in kth cluster and N_k is the cardinality of U_k.
8. If the clusters are modified or the current iteration number is less than the predefined limit, goto (3). Else, stop.

The above algorithm results in a collection of clusters after the completion of processing all training patterns. Type-2 fuzzy TSK rules are then constructed from these clusters.

4.2 Fuzzy Rules Extraction

For extracting fuzzy rules from a dataset (for phase II shown in Fig. 1), several methods were proposed. A fuzzy partitioning method is proposed in [24] to create a set of fuzzy rules. This method segregates the input space into a group of subspaces. But it has increased complexity in terms of speed. In [25], a combination of unsupervised clustering and gradient descent approach is explained for the same purpose. But, for enormous volume of patterns, it is also a time-consuming method. Another method [26] extracts fuzzy rules by first splitting the output space into several clusters. Then, based on the expected output, training patterns are attached to clusters. In this way, partitioning of input space is done. But this method suffers from clusters overlapping. A genetic algorithm-based approach is given in [27] to generate rules from datasets.

At first all data are normalized into the range [0, 1] and then each cluster C_i is converted into a type-2 fuzzy rule of the form

If a_1 is $\tilde{A}_{1i}$ and a_2 is $\tilde{A}_{2i}$ and . . .and a_k is $\tilde{A}_{ki}$ Then b is $c_i = \omega_{0i} + \omega_{1i}a_1 + \cdots + \omega_{ki}a_k$, where $\tilde{A}_{1i}, \tilde{A}_{2i}, \cdots \tilde{A}_{ki}$ are type-2 fuzzy sets for $a_1, a_2, \ldots, a_k$ and each ω is a rule weight of real type. The membership function of $\tilde{A}_{ij}, 1 \le i \le k$, is $\mu_{\tilde{A}_{ij}}(a_i) = \text{gauss}\left(u; \text{gauss}\left(a_i; m_{ij}^p, \sigma_{ij}^p\right), \sigma_{ij}^s\right)$, where $u \in [0, 1]$. Each rule includes mean, deviation, and the scaled deviation as its antecedent parameters and the rule weight as its consequent parameters. The weight is tied to a rule to determine a degree of fulfillment by multiplying with antecedents.

4.3 Fuzzy Rulebase Optimization

The generated T2FNN is based on neurons. Each neuron possesses a center vector and a width vector, thus symbolizes one cluster in the input space. This initial structure of a T2FNN may be redundant and may lead to unnecessary rules in the extracted set of fuzzy rules. Hence, for optimizing the topology of initial network structure, GA is recommended. This optimizes fuzzy rulebase (for phase III of Fig. 1) by identifying the least important fuzzy rules. GAs represent data as chromosomes. The fitness value of a chromosome should be a function of both the number of fuzzy rules and the network performance [28] for selecting the best individuals. The process is rerun for several generations until reaching the individual(s) that firmly accommodate the desired condition [29].

1. Initialize population by selecting random individuals.
2. Repeat
   a. For the size of the population
      i. Select two individuals as $parent_1 = w_1, w_2, \ldots, w_{len}$ and $parent_2 = x_1, x_2, \ldots, x_{len}$ by standard roulette wheel selection, where len is the number of fuzzy rules in the initial rulebase.
      ii. Apply the following crossover to produce a child as $child = o_1, o_2, \ldots, o_{len}$, for $i = 1, 2, \ldots, len$:

$$o_i = \begin{cases} p_i, & \text{if } w_i = x_i \\ o_i, & \text{if } w_i \neq x_i \text{ and } \tau \geq 0.5 \\ \bar{o}_i, & \text{if } w_i \neq x_i \text{ and } \tau < 0.5 \end{cases}.$$

      Here, τ is a random number.
      iii. Apply mutation to the $child$ with mutation rate as 0.1.
      iv. Calculate the fitness values for $child(f_C)$, $parent_1(f_{P1})$ and $parent_2(f_{P2})$ as $f = \frac{1}{RMSD^2 + \frac{1}{(1+len-ctlen)^2}}$, where $ctlen$ is the number of fuzzy rules in the current rulebase and RMSD is the root-mean-square deviation.
      v. Calculate d_{P1} as the Minkowski distance between $parent_1$ and $child$.
      vi. Similarly, calculate d_{P2} between $parent_2$ and $child$.
      vii. If $d_{P2} > d_{P1}$, and $f_C > f_{P1}$, replace $parent_1$ with $child$. Else if $d_{P2} \leq d_{P1}$ and $f_C > f_{P2}$, replace $parent_2$ with $child$.
   b. End For
3. Until the maximum number of generations is reached.
4. Extract the best individuals as a final solution.

This phase is concluded with the compact set of type-2 fuzzy rules by eliminating least significant fuzzy rules.

4.4 Refinement of Fuzzy Rules

This phase is included in our proposed system to refine the parameters of generated fuzzy rules. A self-evolving IT2FNN is proposed in [30] which combines online clustering and rule-ordered Kalman filter algorithm. Gradient descent backpropagation and gradient descent with adaptive learning rate backpropagation are suggested in [31] to refine parameters. We proceed to improve the accuracy of these fuzzy rules through dynamical optimal learning algorithm. In all iterations of training, we begin to improve antecedent parameters by keeping consequent parameters as constant. After that, consequent parameters are refined by holding antecedent parameters as constant. This refinement technique is repeated until this phase results in preferred approximation precision. As mentioned earlier, consequent parameters are kept as fixed initially and IT2FNN is used with gradient descent learning to optimize antecedent parameters. By using BP method, for k input–output training patterns $[p_j : d_j]$, $j = 1, 2, \ldots, r$, the following function of mean square error (MSE) is considered:

$$e_j = \frac{1}{2}\left[y(p_j) - d_j\right]^2. \tag{1}$$

Equation (1) is our objective function that is to be minimized (optimized). Here, $y(p_j)$ is the actual output for jth pattern. We use Eq. (2) to refine mean and a similar type of equation is used to refine standard deviation by keeping the weight as fixed.

$$m_{j1}^i(l+1) = m_{j1}^i(l) - \left(\frac{\alpha(y(p_l) - d_l)\left(p_{jl} - m_{j1}^i\right)N\left(m_{j1}^i, \sigma_j^i; p_{jl}\right)}{2\sigma_j^{i^2}} \times \frac{\left(\prod_{\substack{q=1 \\ q\neq j}}^{t} \bar{u}_{\tilde{A}_q^i}\right)(\omega_{1i} - y_1)}{\left(\sum_{i=1}^{L} \bar{g}^i + \sum_{i=L+1}^{M} \underline{g}^i\right)}\right). \tag{2}$$

Equations (3) and (4) are used to tune consequent parameters by keeping antecedent parameters fixed. If $i > L$,

$$\omega_{1i}(l+1) = \omega_{1i}(l) - \alpha\left(\frac{(y(p_l) - d_l)}{2}\right) \times \left(\frac{\prod_{q=1}^{t} \bar{u}_{\tilde{A}_q^i}}{\sum_{i=1}^{L} \bar{g}^i + \sum_{i=L+1}^{M} \underline{g}^i}\right). \tag{3}$$

If $i > L$,

$$\omega_{1i}(l+1) = \omega_{1i}(l) - \alpha\left(\frac{(y(p_l) - d_l)}{2}\right) \times \left(\frac{\prod_{q=1}^{t} \underline{u}_{\tilde{A}_q^i}}{\sum_{i=1}^{L} \bar{g}^i + \sum_{i=L+1}^{M} \underline{g}^i}\right), \tag{4}$$

where α is the learning rate parameter, M is the total number of rules in the rule base, t is the number of inputs to rule i, $\underline{u}_{\tilde{A}_q^i}(p_j)$ is the lower bound membership value of fuzzy sets $\tilde{A} - q^i$, $\bar{u}_{\tilde{A}_q^i}(p_j)$ is the upper membership value of fuzzy sets $\tilde{A} - q^i$, and $*$ is the product t norm. Here, iterative Karnik–Mendel method is used to acquire L and R values. Here,

$$N\left(m_j^i, \sigma_j^i; p_j\right) = \exp\left(-\frac{1}{2}\left(\frac{p_j - m_j^i}{\sigma_j^i}\right)^2\right) \tag{5}$$

$$\underline{g}^i = \underline{u}_{A_{i1}}(p_1) * \underline{u}_{A_{i2}}(p_2) * \cdots * \underline{u}_{A_{in}}(p_n) = \prod_{q=1}^{t} \underline{u}_{A_q^i}(p_q) \tag{6}$$

and

$$\bar{g}^i = \bar{u}_{A_{i1}}(p_1) * \bar{u}_{A_{i2}}(p_2) * \cdots * \bar{u}_{A_{in}}(p_n) = \prod_{q=1}^{t} \bar{u}_{A_q^i}(p_q). \tag{7}$$

Similarly consequent parameters are optimized by fixing antecedent parameters as constant. After the system is trained, it can be used for detecting intrusions in real time. When the cloud user requires access to the service, his/her pattern of activity is checked against fuzzy rulebase. Inference from this comparison is used in making decisions regarding intrusions, after type reduction and defuzzification.

5 Experimental Results

Our CIDS consists of two entities, namely *Cloud* and *Consumer*. We define *Cloud* as an entity which hosts user's data and acts as Infrastructure-as-a-service (IaaS) provider. *Consumers* are defined as entities which request services from the *Cloud*. We have created the *Cloud* environment with the machine configured with Intel core 2 Duo CPU, 2.5 GHz, 4 GB memory, and GNU = Linux kernel 2.6. 32. This Cloud is designed using Eucalyptus [32] for facilitating *Consumer* to avail IaaS services. We conducted experiments using Cloud Intrusion Detection Dataset (CIDD) [5] in three steps labeled, training, validation, and testing. In training, fuzzy rules are extracted and optimized by our proposed IDS architecture using the training data to achieve maximum accuracy on the dark data. The validation step is to assess how precisely a system will act upon in practice. This method observes the error on validation dataset and terminates system training when this error starts to increase. In the testing phase, the test data are passed through the saved trained model to detect intrusions.

The cloud intrusion detection dataset consists of connection records for describing normal and intrusive accesses. Each such record has 41 features for describing a

total of 24 attack types. These attack types are grouped into four main categories of attacks: denial-of-service (DoS), probe, remote-to-local (R2L), and user-to-root (U2R), as shown in Table 2. The original dataset is of size 744 MB with 4,940,000 records, where each record defines a specific attack. Using all 41 features could result in a big IDS model with increased computation time, which could be an overhead for online detection. Moreover, if the dataset contains unrelated features, analysis will be difficult to detect suspicious behaviors. CIDS must therefore reduce the amount of data to be processed. By using an intelligent input feature selection [33], the number of features is reduced to 12 and shown in Table 3. Features in the datasets are of different forms such as symbolic, discrete, and continuous. Most pattern classification methods are not capable to process data in such a format. So each feature is linearly scaled to the range [0, 1] by preprocessing. We selected 20,000 records made at random in which 12,000 are normal and 8000 are intrusive. Training, validation, and testing enclose 11,982, 4810. and 3208 samples, respectively. Since the dataset has five diverse classes, we carried out a five-class binary classification.

Experiments are conducted for the threshold values of 0.05, 0.10, 0.15, and 0.20. These thresholds are considered as mutation rates. To get away from local minima, we have used these di_erent error threshold values randomly in our experiments. The results are observed at the end of phases III and IV which are given in figures from 2 to 5. Figure 2 explains the accuracy obtained at the end of phase III for training dataset and Fig. 3 explains the same for the test dataset.

Using the same training and test datasets, results are obtained at the completion of phase IV. These results are explained in Figs. 4 and 5 and we found significant improvement in detection accuracy which shows the importance of parameter refinement phase in our proposed intrusion detection process. The average detection rates for each category of pattern during testing and training are shown in Table 4. It is inferred that the proposed approach achieves the accuracy of 98.70 % during training phase and 98.49 % during testing phase. Moreover, we obtained 0.99 and 0.98 classification rates during training and testing, respectively.

Since we found that our cloud intrusion detection system achieves better results with parameters refinement phase,

Table 2 Types of attacks

Attack	Description
Denial-of-Service (DoS)	An attacker forges some resource overloaded and refuses to handle genuine user requests
Probing	An attacker examines a network of computers to gather vulnerability information for abuse
Remote-to-local (R2L)	An attacker first sends packets to a machine over a network, then takes benefit of the machines weakness to illegitimately gain access as a legitimate user
User-to-Root (U2R)	An attacker initially accesses resources as a normal user and then exploits vulnerability to expand to root access to the system

Table 3 List of features considered in the proposed method

Feature	Type	Description
service	Discrete/integer/nominal	Service on destination (e.g., TCP, UDP, telnet, ftp)
src_bytes	Continuous/real/numeric	Number of bytes sent from source to destination
dst_bytes	Continuous/real/numeric	Number of bytes sent from destination to source
logged_in	Discrete/integer/binary	1, if logged in successfully; 0, if not
count	Continuous/real/numeric	Number of connections to the same host as the current connection in the past two seconds
srv_count	Continuous/real/numeric	Number of connections to the same service as the current connection in the past two seconds
serror_rate	Continuous/real/numeric	Percentage of connections with SYN errors
srv_rerror_rate	Continuous/real/numeric	Percentage of connections with REJ errors
srv_diff_host_rate	Continuous/real/numeric	Percentage of connections to different hosts
dst_host_count	Continuous/real/numeric	Count of connections that hold the same destination host
dst_host_srv_count	Continuous/real/numeric	Count of connections that hold the same destination host and the same service
dst_host_diff_ srv_ rate	Continuous/real/numeric	Percentage of different services on the current destination host

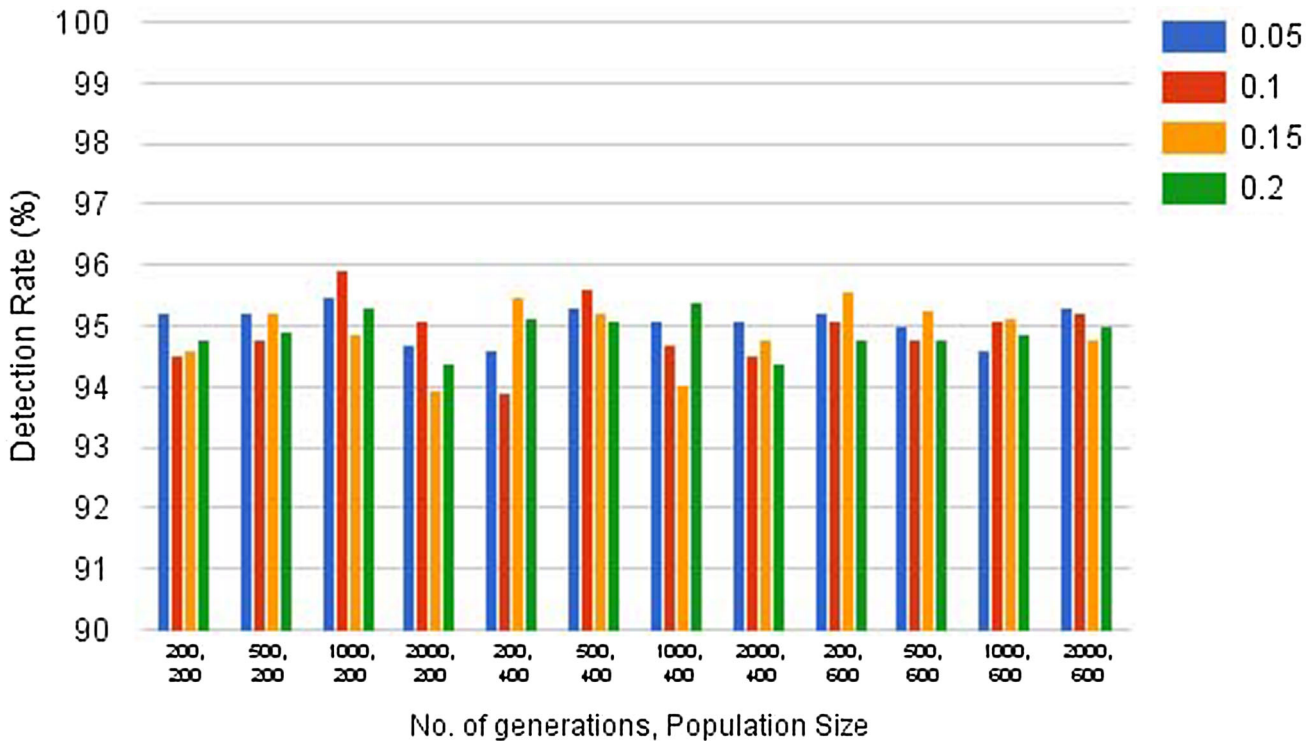

Fig. 2 Training set detection rate—without parameters refinement

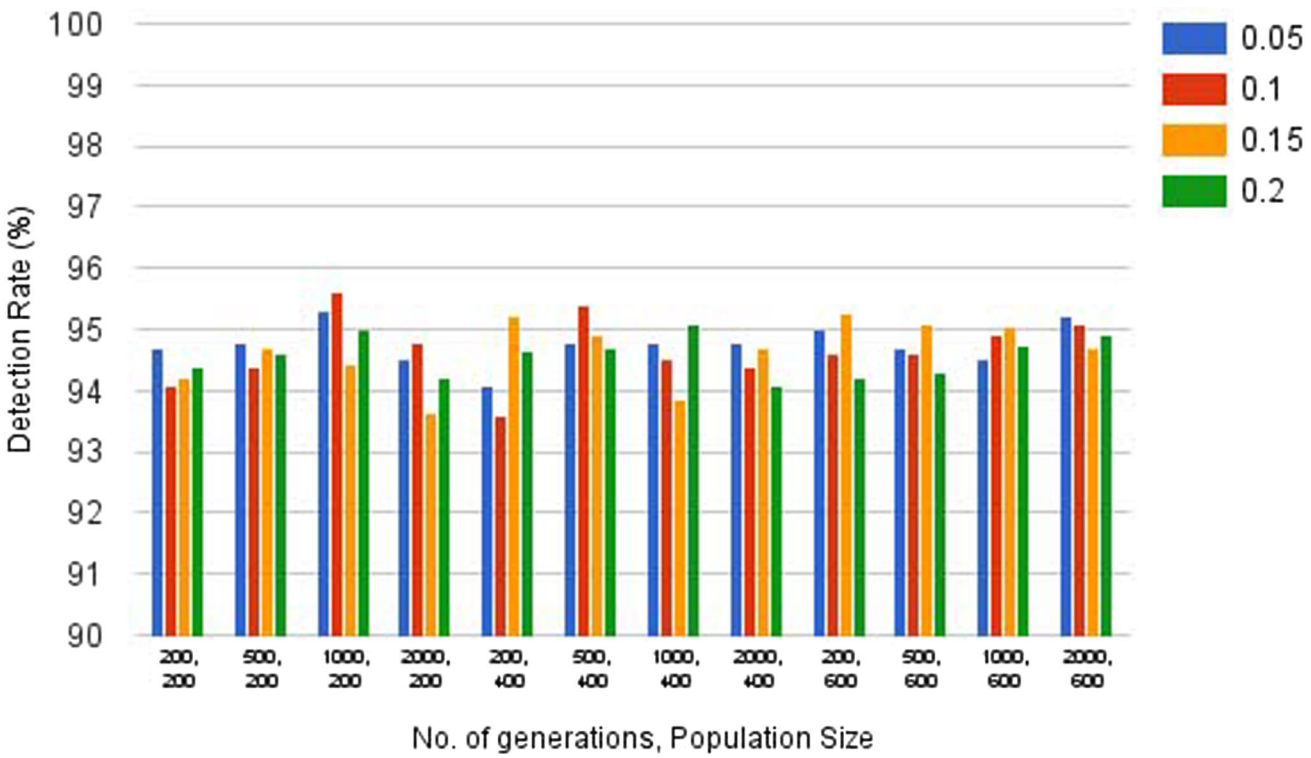

Fig. 3 Test set detection rate—without parameters refinement

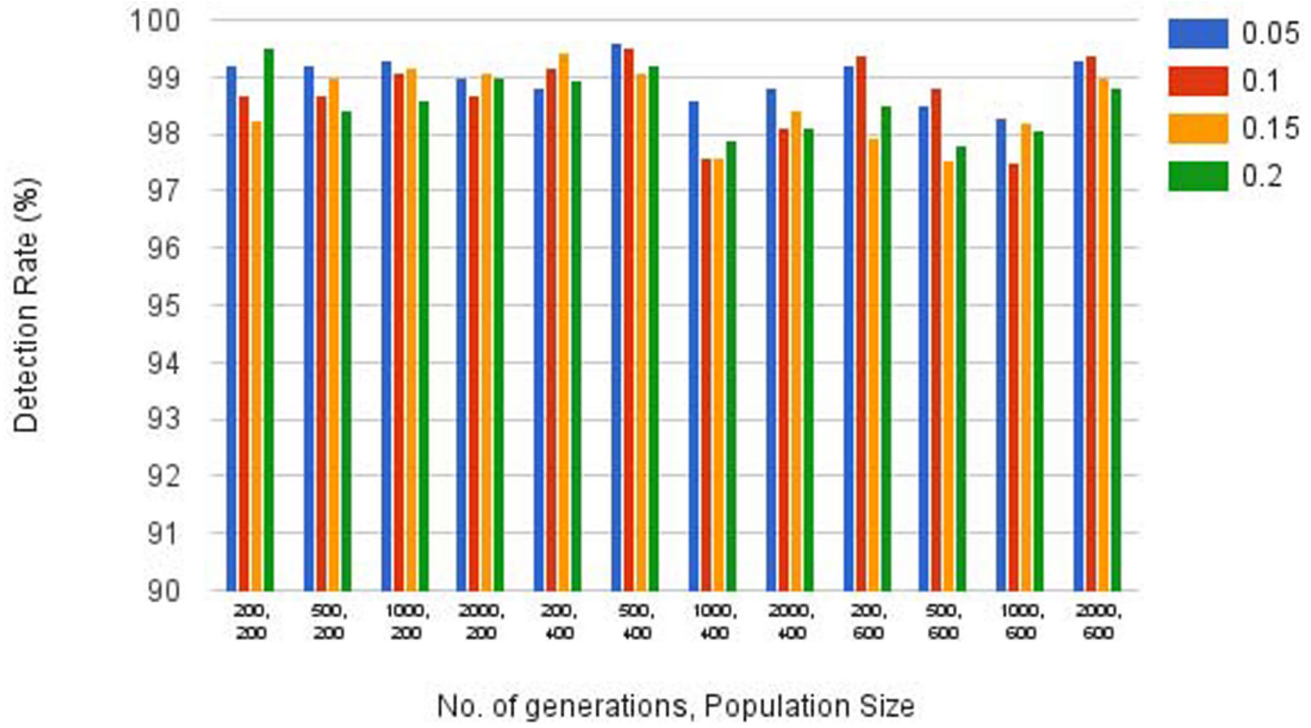

Fig. 4 Training set detection rate—with parameters refinement

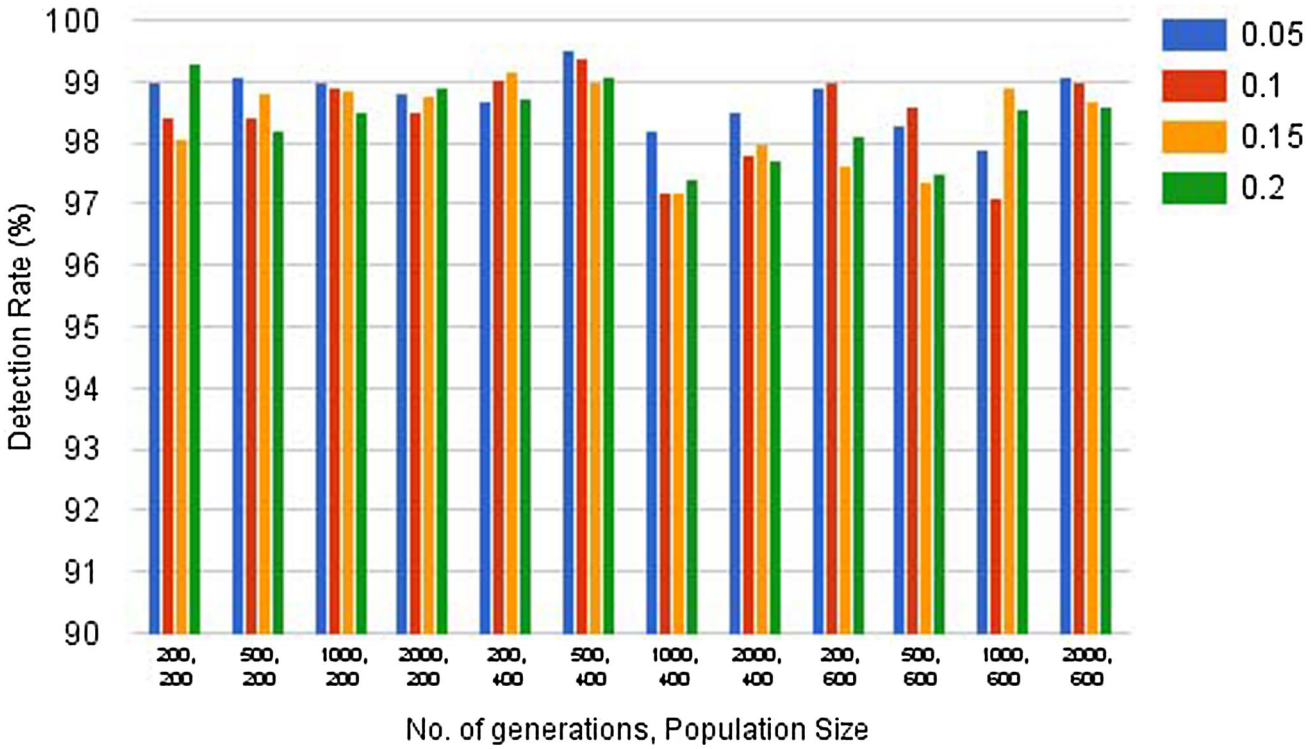

Fig. 5 Test set detection rate—with parameters refinement

the following tables were given by considering measurements obtained after parameters refinement phase.

We conducted experiments for various population sizes (200, 400, and 600). Table 5 gives detection accuracies during training for every combination of threshold value and number of generations. Similar types of experiments were conducted during testing and the results are shown in Table 6.

From these measurements, we infer that lower populations present good accuracies with lower threshold values, whereas higher populations achieve good results with moderately higher threshold values. Additionally, lower populations exhibit likely equal performance irrespective of the number of generations. Analogous experiments are conducted for different number of generations (200, 500, 1000, and 2000). Results are shown in Tables 7 and 8. Their observations imply that lower number of generations produces good accuracies with lower populations and higher thresholds. On the other hand, higher number of

Table 4 Detection accuracy

Pattern	Detection rate (%)	
	Training	Testing
Normal	97.22	98.16
Probe	99.80	99.88
DoS	97.77	96.17
U2R	99.65	99.03
R2L	99.08	99.22

Table 5 Training accuracy based on population size

No.of gen	Population size = 200				Population size = 400				Population size = 600			
	0.05	0.10	0.15	0.20	0.05	0.10	0.15	0.20	0.05	0.10	0.15	0.20
200	99.2	98.7	98.3	99.5	98.8	99.2	99.5	98.9	99.2	99.4	97.9	98.5
500	99.2	98.7	99.0	98.4	99.6	99.5	99.1	99.2	98.5	98.8	97.6	97.8
1000	99.3	99.1	99.2	98.6	98.6	97.6	97.6	97.9	98.3	97.5	98.2	98.1
2000	99.0	98.7	99.1	99.0	98.8	98.1	98.4	98.1	99.3	99.4	99.0	98.8

Table 6 Testing accuracy based on population size

No.of gen	Population size = 200				Population size = 400				Population size = 600			
	0.05	0.10	0.15	0.20	0.05	0.10	0.15	0.20	0.05	0.10	0.15	0.20
200	99.0	98.4	98.1	99.3	98.7	99.2	99.2	98.8	98.9	99.0	97.7	98.1
500	99.1	98.4	98.8	98.2	99.5	99.4	99.0	99.1	98.3	98.6	97.4	97.5
1000	99.0	98.9	98.9	98.5	98.2	97.2	97.2	97.4	97.9	97.1	98.9	98.6
2000	98.8	98.5	98.8	98.9	98.5	97.8	98.0	97.7	99.1	99.0	98.7	98.6

Table 7 Training accuracy based on number of generations

Pop size	No. of gen = 200				No. of gen = 500				No. of gen = 1000				No. of gen = 2000			
	0.05	0.10	0.15	0.20	0.05	0.10	0.15	0.20	0.05	0.10	0.15	0.20	0.05	0.10	0.15	0.20
200	99.2	98.7	98.3	99.5	99.2	98.7	99.0	98.4	99.3	99.1	99.2	98.6	99.0	98.7	99.1	99.0
400	98.8	99.2	99.5	98.9	99.6	99.5	99.1	99.2	98.6	97.6	97.6	97.9	98.8	98.1	98.4	98.1
600	99.2	99.4	97.9	98.5	98.5	98.8	97.6	97.8	98.3	97.5	98.2	98.1	99.3	99.4	99.0	98.8

Table 8 Testing accuracy based on number of generations

Pop size	No. of gen = 200				No. of gen = 500				No. of gen = 1000				No. of gen = 2000			
	0.05	0.10	0.15	0.20	0.05	0.10	0.15	0.20	0.05	0.10	0.15	0.20	0.05	0.10	0.15	0.20
200	99.0	98.4	98.1	99.3	99.1	98.4	98.8	98.2	99.0	98.9	98.9	98.5	98.8	98.5	98.8	98.9
400	98.7	99.1	99.2	98.8	99.5	99.4	99.0	99.1	98.2	97.2	97.2	97.4	98.5	97.8	98.0	97.7
600	98.9	99.0	97.7	98.1	98.3	98.6	97.4	97.5	97.9	97.1	98.9	98.6	99.1	99.0	98.7	98.6

Table 9 Training and testing MSE based on threshold θ

θ	Training MSE	Testing MSE
0.05	1.17	1.75
0.10	2.05	2.91
0.15	2.46	3.09
0.20	2.32	2.96

Table 10 Confusion matrix—training

Actual class	Detection accuracy of predicted class (%)				
	Normal	Probe	DoS	U2R	R2L
Normal	97.2446	0.6359	1.5684	0.2543	0.2967
Probe	0	100	0	0	0
DoS	1.0750	0.6639	97.7658	0.1581	0.3372
U2R	0	0	0	100	0
R2L	0	0	0	0	100

Table 11 Confusion matrix—testing

Actual class	Detection accuracy of predicted class (%)				
	Normal	Probe	DoS	U2R	R2L
Normal	98.08	0.64	0.96	0.16	0.16
Probe	0	100	0	0	0
DoS	2.193167	1.180936	96.16196	0.84353	0.379587
U2R	0	0	0	100	0
R2L	0	0	0.60241	0	99.39759

Table 12 Comparison with other approaches

Rate	Proposed method		As in [34]	As in [35]	As in [12]
	Training	Testing			
Recall	0.9894	0.9799	0.8610	0.9499	0.9199
Precision	0.9932	0.9953	0.8861	0.9280	0.9903
Accuracy	0.9861	0.9801	0.7989	0.9004	0.9282
F_1 score	0.9883	0.9875	0.8734	0.9388	0.9538

generations brings fine accuracies for lower populations and lower thresholds.

Table 9 shows MSE values obtained from training phase and testing phase for various values of θ, which gives the average MSE of 2.001 and 2.6768 during training and testing, respectively.

Tables 10 and 11 show the confusion matrix during training and testing, respectively. From these tables, it is inferred that, though we have some detection degradation for DoS and R2L attacks, our system behaves perfectly in identifying probe and U2R attacks. Further its true negative prediction is increased by 0.8354 % during testing.

We analyzed the rate of recall, precision, accuracy, and F_1 score for training and testing and the results are given in Table 12. Further, we compared them with that of other methods. This comparison shows that the proposed system can achieve good results in the field of cloud intrusion detection.

Results of using NN and GA individually in identifying different types of attacks in intrusion detection are shown in Table 13 along with the results of our proposed method which shows that our proposed method behaves well in identifying all four types of attacks with an average detection rate of 98.598 %.

To show the effectiveness of our proposed cloud intrusion detection system, we conducted similar experiments using UCS (supervised learning classifier system) [37] and NICE (Network Intrusion detection and Countermeasure sElection in virtual network systems) [39] for detecting intrusions in a cloud environment. The results of comparison are made by varying number of jobs and number of nodes (scalability) and they are explained as follows:

Further, Table 14 gives the average success rate of intrusion detection when a single framework is adopted in various researches. In all these cases, the achieved detection rate is less than the average detection rate (98.598 %) of our proposed method which shows the importance of our hybrid technique.

Rates of error in detecting intrusions (false negatives and false positives) during training are shown in Fig. 6 for the three intrusion detection approaches (UCS, NICE, and Proposed method). From this figure, it is inferred that all the three methods achieve approximately similar error rate for less number of jobs. But our proposed method gives lower error rate than that of remaining two methods and this accuracy improvement is achieved through the application of our hybrid method, whereas UCS and NICE employ single methodology.

As shown in Fig. 7, the percentages of CPU (resource) utilization for different number of jobs under the three different methods are compared. In all three methods, CPU usage is almost identical for less number of jobs. As the number of jobs tends to increase, UCS, NICE, and proposed methods exhibit different performances in CPU utilization and our proposed hybrid intrusion detection

Table 13 Results of using NN or GA alone in intrusion detection

Actual class	Detection accuracy of predicted class (%)					
	FNN [36]	ANN [37]	ANN [38]	GA [35]	GA [12]	Proposed NN + GA
Normal	95.89	99.38	96.4	69.5	96.14	97.69
Probe	81.61	75.36	85.7	71.1	85.77	99.84
DoS	97.00	96.77	97.5	99.4	96.68	96.97
U2R	14.91	21.52	48.0	18.9	75.71	99.34
R2L	6.90	3.10	95.0	5.4	30.3	99.15
Average	59.262	59.226	84.52	52.86	76.92	98.598

Table 14 Comparison of average intrusion detection rates under various non-hybrid frameworks

Method	Average detection rate (%)
Multilayer perception	77.41
ART-1	97.42
ART-2	97.19
SOM	95.74
GA classifier	92.94
Proposed method	98.598

approach consumes less CPU time than that of other two methods. It saves 28 % of CPU time than NICE and 33 % of CPU time than UCS in the field of cloud intrusion detection during training phase.

To show the speed of computation, we varied the number of nodes in a cloud for same number of jobs. For each of the three frameworks, average execution time during training is measured. Figure 8 shows the results of these experiments incorporating intrusion detection in a cloud environment. The experimental results prove that the speed of the proposed method is directly proportional to the size of the cloud and when we double the number of nodes for similar set of jobs, its speed up ratio is also increased by a factor of 2.15 approximately. Further, it has improved speed up ratio than that of NICE for larger number of nodes in a cloud. This shows the commendable scaling tendency of our proposed method toward large networks. But for small number of nodes, NICE produces better speed up ratio. But this deviation is only about 0.07. But still, this issue of our proposed method has to be addressed in future.

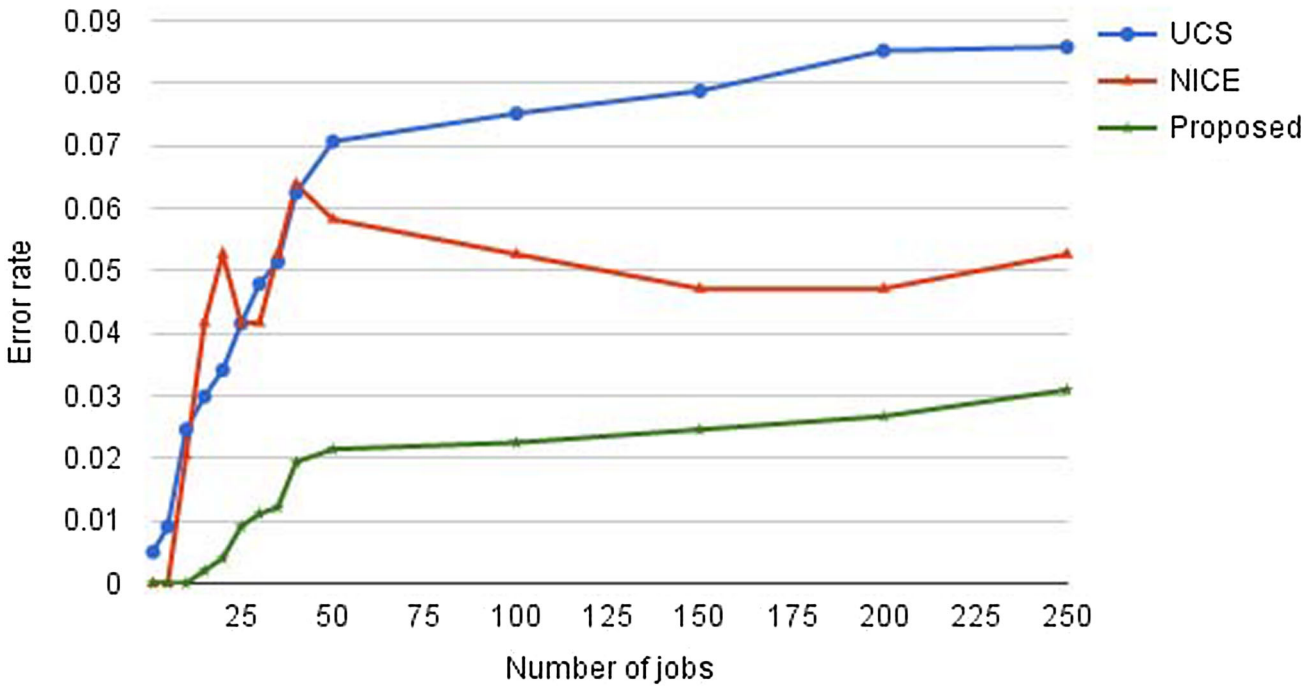

Fig. 6 Comparison of error rate under different number of jobs

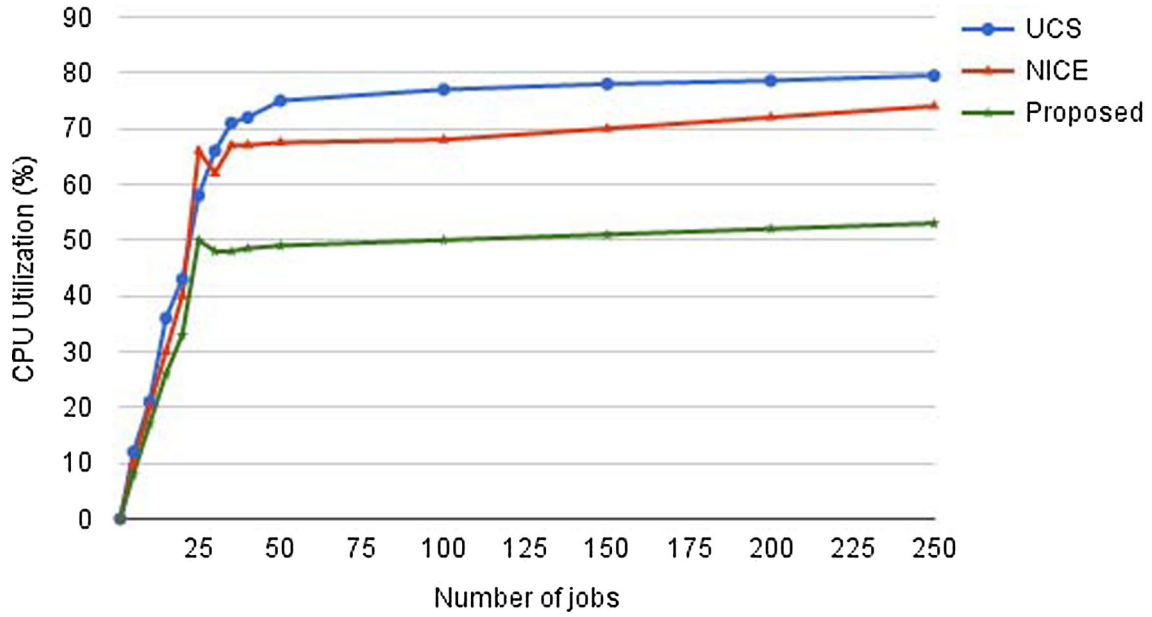

Fig. 7 Comparison of CPU utilization under different number of jobs

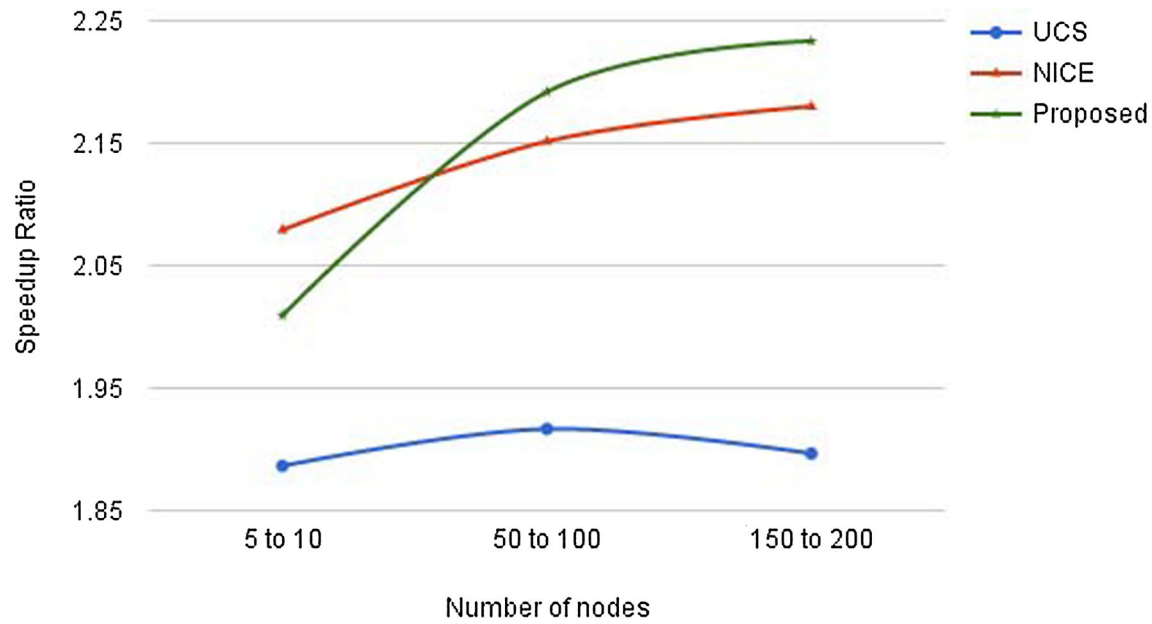

Fig. 8 Comparison of speedup ratio under different number of nodes

6 Conclusion

In this paper, we have proposed a methodology for modeling IDS in a cloud. The intrusion detection training dataset is partitioned into several clusters with similar patterns belonging to the same cluster using modified K-Means algorithm with Minkowski distance. Each of the resulting clusters is defined with the membership function by statistical mean and deviation which results in a type-2 fuzzy TSK-rule. A fuzzy neural network is established correspondingly and the compact fuzzy rulebase is constructed by applying genetic algorithm. Then the associated parameters are refined by dynamical optimal learning. Later, for every new input from the test dataset, a crisp

output is obtained by integrating the inferred results of all the rules. This result is then compared with the desired result based on which detection rates are calculated. The proposed scheme enables cloud users to avail cloud services without worrying about intrusions when they wish to switch to cloud data centers. Since the achieved results are optimum as compared to other approaches in the field of attack detection, we trust that our proposed model provides a convincing method.

References

1. Vieira, K., et al.: Intrusion detection for grid and cloud computing. IT Prof. **12**(4), 38–43 (2010)
2. Xin, W., Ting-lei, H., Xiao-yu, L.: Research on the Intrusion detection mechanism based on cloud computing. In: International conference on intelligent computing and integrated systems (ICISS), Guilin, pp. 125–128 (2010)
3. Tupakula, U., Varadharajan, V., Akku N.: Intrusion detection techniques for infrastructure as a service cloud. In: IEEE international conference on dependable, autonomic and secure computing, pp. 744–751 (2011)
4. Dhage, S. et al: Intrusion detection system in cloud computing environment. In: International Conference and Workshop on Emerging Trends in Technology, New York, NY, USA, pp. 235–239 (2011)
5. Kholidy, H.A., Baiardi, F.: CIDD: A cloud intrusion detection dataset for cloud computing and Masquerade Attacks, Ninth international conference on information technology—New Generations, Las Vegas, Nevada USA, ISBN. 978-0-7695-4654-4 (2012);
6. Subashini, S., Kavitha, V.: A survey on security issues in service delivery models of cloud computing. J. Netw. Comput. Appl. **34**(1), 1–11 (2011)
7. Jin, H. et al: A VMM-based intrusion prevention system in cloud computing environment. J. Supercomput., pp. 1–19 (2011)
8. Zadeh, L.A.: The concept of a linguistic variable and its application to approximate reasoning I. Inf. Sci. **8**(3), 199–249 (1975)
9. Karnik, N.N., Mendel, J.M.: Centroid of a type-2 fuzzy set. Inf. Sci. **132**, 195–220 (2001)
10. Lin, Faa-Jeng, Chou, Po-Huan, Shieh, Po-Huang, Chen, Syuan-Yi: Robust control of an LUSM-Based XY θ motion control stage using an adaptive interval Type-2 fuzzy neural network. IEEE Trans. Fuzzy Syst. **17**(1), 24–38 (2009)
11. Bhat, A.H., Patra, S., Jena, D.: Machine learning approach for intrusion detection on cloud virtual machines. Int. J. Appl. Innov. Eng. Manag. **2**(6), 56–66 (2013)
12. Shirazi, H.M., Kalaji, Y.: An intelligent intrusion detection system using genetic algorithms and features selection. Majlesi J. Electr. Eng. **4**(1), 33–43 (2010)
13. Gonzalez, A., Herrera, F.: Multi-stage genetic fuzzy systems based on the iterative rule learning approach. Mathw. Soft Comput. **4**, 233–249 (1997)
14. Farag, W.A., Quintana, V.H., Lambert-Torres, G.: A genetic-based neuro-fuzzy approach for modeling and control of dynamical systems. IEEE Trans. Neural Netw. **9**(5), 756–767 (1998)
15. Mamdani, E.H., Assilian, S.: An experiment in linguistic synthesis with a fuzzy logic controller. Int. J. Man Mach. Stud. **7**, 1–13 (1975)
16. Shi, Y., Eberhart, R., Chen, Y.: Implementation of evolutionary fuzzy systems. IEEE Trans. Fuzzy Syst. **7**(1), 109–119 (1999)
17. Bezdek, J.: Pattern Recognition with Fuzzy Objective Function Algorithms. Plenum Press, New York (1981)
18. Kohonen, T.: Self-Organizing Maps. Springer, New York (1997)
19. Yager, R., Filev, D.: Generation of fuzzy rules by mountain clustering. J. Intell. Fuzzy Syst. **2**(3), 209–219 (1994)
20. Chiu, S.: Fuzzy model identification based on cluster estimation. J. Intell. Fuzzy Syst. **2**, 267–278 (1994)
21. Kanungo, T., et al.: An efficient k-means clustering algorithm: analysis and implementation. IEEE Trans. Pattern Anal. Mach. Intell **24**(7), 881–892 (2002)
22. Su, M.-C., Chou, C.-H.: A modified version of the K-means algorithm with a distance based on cluster symmetry. IEEE Trans. Pattern Anal. Mach. Intell. **23**(6), 674–680 (2001)
23. Aziz, A.S.A., et al.: Artificial immune system inspired intrusion detection system using genetic algorithm. Informatica **36**, 347–357 (2012)
24. Lin, Y., Cunningham, I.I.I.G.A., Coggeshall, S.V.: Using fuzzy partitions to create fuzzy systems from input-output data and set the initial weights in a fuzzy neural network. IEEE Trans. Fuzzy Syst. **5**(4), 614–621 (1997)
25. Wong, C.-C., Chen, C.-C.: A hybrid clustering and gradient descent approach for fuzzy modeling. IEEE Trans. Syst. Man Cybern. Part B **29**(6), 686–693 (1999)
26. Thawonmas, R., Abe, S.: Function approximation based on fuzzy rules extracted from partitioned numerical data. IEEE Trans. Syst. Man Cybern. **29**(4), 525–534 (1999)
27. Vivekanandan, P., Rajalakshmi, M., Nedunchezhian, R.: An intelligent genetic algorithm for mining classification rules in large datasets. Comput. Inform. **32**(1), 1–22 (2013)
28. Leng, G., McGinnity, T.M., Prasad, G.: Design for self-organizing fuzzy neural networks based on genetic algorithms. IEEE Trans. Fuzzy Syst. **14**(6), 755–766 (2006)
29. Uncu, O., Turksen, I.B.: Discrete interval type 2 fuzzy system models using uncertainty in learning parameters. IEEE Trans. Fuzzy Syst. **15**(1), 90–106 (2007)
30. Juang, C.-F., Tsao, Y.-W.: A self-evolving interval type-2 fuzzy neural network with online structure and parameter learning. IEEE Trans. Fuzzy Syst. **16**(6), 1411–1424 (2008)
31. Castro, J.R., et al.: A hybrid learning algorithm for a class of interval type-2 fuzzy neural networks. Inf. Sci. **179**(13), 2175–2193 (2009)
32. Eucalyptus. http://www.eucalyptus.com
33. Abraham A., Jain R.: Soft computing models for network intrusion detection systems. In: Classification and Clustering for Knowledge Discovery. Studies in Computational Intelligence, vol. 4, pp. 191–207. Springer, Berlin (2005)
34. Mostaque Md and Morshedur Hassan (2013); Network intrusion detection system using genetic algorithm and fuzzy logic. Int. J. Innov. Res. Comput. Commun. Eng. 1(7)
35. Mohammad Sazzadul Hoque, Md. Abdul Mukit and Md. Abu Naser Bikas: An implementation of intrusion detection system using genetic algorithm. Int. J. Netw. Secur. Appl., vol. 4, no. 2 (2012)
36. Tsang, S. K., Wang, H.: Anomaly intrusion detection using multi-objective genetic fuzzy system and agent-based evolutionary computation framework. In: Proceedings of the Fifth IEEE international conference on data mining (2005)
37. Shafi, K., Abbass, H.A.: An adaptive genetic-based signature learning system for intrusion detection. Expert Syst. Appl. **36**, 12036–12043 (2009)
38. Mukkamala, S., Sung, A.H., Abraham, A.: Intrusion detection using ensemble of soft computing paradigms, advances in soft computing. Intell. Syst. Des. Appl. **23**, 239–248 (2003)

39. Chung, C.-J., Khatkar, P., Xing, T., Lee, J., Huang, D.: NICE: network intrusion detection and countermeasure selection in virtual network systems. IEEE Trans. Dependable Secure Comput. **10**(4), 198–211 (2013)

Sivakami Raja received her B.E. degree from the Department of Computer Science and Engineering, Madurai Kamaraj University, India, in 2002 and M.E. degree from the Department of Information and Communication Engineering, Anna University, Chennai, in 2006. Currently, she is an Assistant Professor in the Department of Information Technology, PSNA College of Engineering and Technology, Dindigul, Tamilnadu, India. Her research interest includes data mining, information security, cloud computing, and bigdata analytics.

Saravanan Ramaiah received his B.E. degree in Electrical and Electronics Engineering from Thiagarajar College of Engineering, India, in 1994, M.E. in Computer Science and Engineering from Madurai Kamaraj University, India, in 2000, and Ph.D. in Distributed Computing from Anna University, in 2010. Currently, he is a Director in RVS Educational Trust's Group of Institutions, Dindigul, Tamilnadu, India. His research interest includes distributed computing, information security, and mobile computing.