2013 IEEE Congress on Evolutionary Computation
June 20-23, Cancún, México

MapReduce Intrusion Detection System based on a Particle Swarm Optimization Clustering Algorithm

Ibrahim Aljarah and Simone A. Ludwig
Department of Computer Science
North Dakota State University
Fargo, ND, USA
{ibrahim.aljarah,simone.ludwig}@ndsu.edu

***Abstract*—The increasing volume of data in large networks to be analyzed imposes new challenges to an intrusion detection system. Since data in computer networks is growing rapidly, the analysis of these large amounts of data to discover anomaly fragments has to be done within a reasonable amount of time. Some of the past and current intrusion detection systems are based on a clustering approach. However, in order to cope with the increasing amount of data, new parallel methods need to be developed in order to make the algorithms scalable. In this paper, we propose an intrusion detection system based on a parallel particle swarm optimization clustering algorithm using the MapReduce methodology. The use of particle swarm optimization for the clustering task is a very efficient way since particle swarm optimization avoids the sensitivity problem of initial cluster centroids as well as premature convergence. The proposed intrusion detection system processes large data sets on commodity hardware. The experimental results on a real intrusion data set demonstrate that the proposed intrusion detection system scales very well with increasing data set sizes. Moreover, it achieves close to the linear speedup by improving the intrusion detection and false alarm rates.**

I. Introduction

Network intrusion detection has been identified as one of the most challenging needs of the network security community in recent years. This is because of the inflated number of users and the amount of data exchanged which makes it difficult to distinguish the normal data connections from others that contain attacks. This requires the development of intrusion detection systems (IDSs) that can analyze large amounts of data in a reasonable time in order to take appropriate actions against the attacks.

IDSs are classified based on their analysis model and placement approach. In the analysis approach, IDSs are categorized into two classes: misuse and anomaly detection. In the misuse-based class, the IDS checks the network and system activity for a known misuse pattern that was identified beforehand through a pattern matching algorithm.

The anomaly-detection based IDS works differently whereby the decisions are made based on a profile of a normal network or system behavior, often constructed using statistical or machine learning techniques. Each of these approaches offer its strengths and weaknesses. Misuse-based systems generally have very low false positive rates that indicate error rates of mistakenly detected non-intrusion cases. Therefore, this approach is seen in the majority of commercial systems. In addition, the misuse-based systems are unable to identify novel or obfuscated attacks.

On the other hand, anomaly-based IDSs are able to detect new attacks that have not been seen before. However, this model produces a large number of false positives. The reason for this is the inability of current anomaly-based techniques to cope adequately with the fact that in the real world, normal, legitimate computer networks, and system usage changes over time. This implies that any profile of normal behavior needs to be dynamic. Thus, with the exponential growth in the different types of attacks, a pattern-matching algorithm is not trust-worthy in the misuse approach, and therefore, it is recommended to use both in an IDS [1].

The placement approach is usually divided into host-based and network-based systems. An IDS which operates on a computer to detect malicious activity on that host, is called a host-based IDS; whereas an IDS that tries to detect events of interest by analyzing and monitoring network traffic data is called a network-based IDS [2]. Network-based IDSs detect attacks by analyzing network packet traffic along a network segment or switch, enabling the monitoring and protection of multiple hosts by a separate machine. Host-based IDS systems have the ability to determine if an attempted attack was successful or not in a local machine. Network-based systems are able to monitor a large number of hosts with relatively low deployment costs in comparison to host-based systems, and are able to identify attacks to and from multiple hosts.

There are several different data mining techniques that have been used for IDSs in the past. Clustering [3] is one of the data mining techniques that is used to explore data. Clustering uses an unsupervised learning mechanism to find distinct patterns in a group of data without prior knowledge about data labels. Through the learning process, the similarities between the data objects are measured to divide the data objects into different subsets called clusters. High quality clusters denote that the similarities within the same cluster and the dissimilarities between different clusters should be maximized. In many clustering algorithms, the similarity between different data objects is based on distance calculations such as the closer objects are placed together. The distance is considered the best similarity measure especially when the shape of the clusters

978-1-4799-0454-9/13/$31.00 ©2013 IEEE

is formed to be spherical.

There are many applications in which large data sets need to be explored such as image analysis, pattern recognition analysis, social networks, large-scale network traffic analysis and many others. These applications are too large to be processed by sequential methods. Traditional intrusion detection methods based on clustering do not scale well with larger sizes of network traffic and are computationally expensive in terms of memory. Furthermore, large-scale network traffic analysis provides a challenge in terms of performance while identifying the anomalies connections, and that is why a parallel algorithm is needed for detecting intrusions.

In general, developing traditional parallel algorithms using the Message Passing Interface (MPI) methodology [4] faces a wide range of difficulties such as handling the network communication in an efficient manner and balancing the distribution of the processing load between different processors. Also, parallel algorithms suffer from node failure, thus reducing the algorithm's scalability. As a result, the development of an efficient parallel intrusion detection algorithm should be scalable and obtain high intrusion detection rates.

The MapReduce programming model [5] has become an alternative parallel processing model for MPI [4] especially for data intensive applications. Many advantages make the MapReduce methodology to be a good choice for parallelizing the data mining tasks such as easy implementation without having to know too many parallel programming details. In addition, MapReduce provides many solutions for node failure and load balancing.

MapReduce usually divides the input data set into independent splits which depend on the size of the data set and the number of computer nodes used. MapReduce consists of two main functions: Map and Reduce functions. The Map function processes the input data records as (key, value) data pairs to generate intermediate output as (key, values list) data pairs, and then the Reduce function merges and aggregates all intermediate (values list) output coming from the Map function having the same intermediate key.

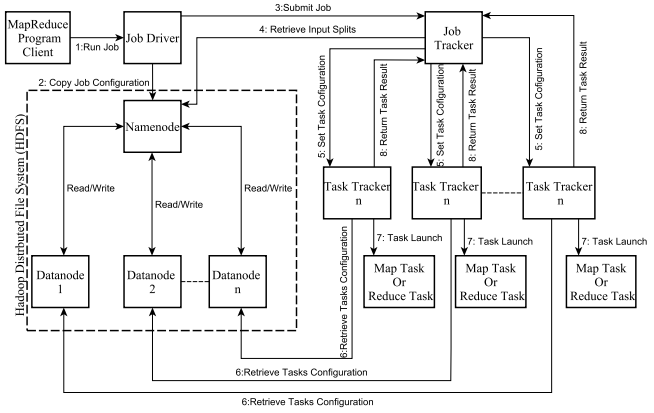


Fig. 1. Hadoop Architecture Diagram [7]

Hadoop [6] is an open source framework, introduced by Apache that uses the MapReduce methodology, which was built in order to deal with data-intensive applications. The Hadoop framework has its own distributed file system called the Hadoop Distributed File System (HDFS) that is used to support the management and processing of large scale data sets. Furthermore, the MapReduce in Hadoop is designed to work efficiently with HDFS by moving the computation process to the data and not the other way around to allow Hadoop to achieve high data locality. Figure 1 shows the Hadoop architecture diagram with interaction between its components. Interested readers may refer to [6] for more details.

This paper presents a parallel intrusion detection system (IDS-MRCPSO) based on the MapReduce framework since it has been confirmed as a good parallelization methodology for many applications. In addition, the proposed system incorporates clustering analysis to build the detection model by formulating the intrusion detection problem as an optimization problem. Furthermore, the proposed system has been tested on a real large-scale intrusion data set with different training subset sizes to show its speedup and scalability, and to present its detection quality.

The rest of this paper is organized as follows: Section II presents the related work in the area of anomaly-detection algorithms based on clustering. In Section III, a brief introduction to the Particle Swarm Optimization (PSO) technique is given and then our proposed IDS-MRCPSO system is introduced. Section IV presents the experimental evaluation, and Section V presents our conclusions.

II. Related Work

Anomaly-detection based intrusion detection systems work based on a profile of a normal network or system behavior using statistical or machine learning techniques. Anomaly-detection based on machine learning techniques can be categorized as either supervised or unsupervised depending on whether the class labels are known during the learning process or not. Several techniques have been proposed to tackle the intrusion detection problem using unsupervised algorithms like clustering-based algorithms.

We focus only on closely related work of unsupervised algorithms that were developed to solve anomaly detection. Also, we discuss one unsupervised parallel algorithm that was applied on anomaly detection systems.

Leung et al. in [7] proposed a density-based clustering algorithm by applying the frequent pattern tree on a high dimensional data set. Their algorithm was applied on one million records and achieved good detection rates, but it suffered from high false positive rates.

However in [8], the same authors proposed an anomaly detection technique based on a K-means clustering algorithm. A technique to enhance the initial centers was proposed to avoid the shortcoming of sensitivity of the initial clusters in K-means clustering to enhance the clusters quality. The experiments were applied on 0.4% of the whole data set.

A fuzzy C-means intrusion detection algorithm was proposed in [9], where a weighting means for the degree of record

membership with specific clusters was used. The algorithm was tested with five samples of random data, each sample having ten thousand records. The results showed high false positives rates with satisfactory detection rates.

Li et al. in [10] combined the K-means algorithm with PSO to build an intrusion detection system. The algorithm tried to benefit from the PSO characteristics to avoid premature convergence that K-means suffers from. The algorithm achieved relatively better results than the K-means algorithm.

Mazel et al. in [11] introduced an unsupervised approach to detect the network anomalies by combining the subspace clustering with inter-clustering result associations to mark the anomalies from the network traffic flow. The authors build an autonomous intrusion detection system to enhance the network protections against the intrusions. The system was tested with real network traffic and verified that the anomalies can be detected in the distributed network.

Gao et al. in [12] proposed a parallel clustering ensemble algorithm to speed the detection of intrusions in massive network traffic. Their algorithm was applied on a sample of data and achieved improved detection time while having satisfactory detection rate.

The technique proposed in this paper is different from the techniques explained above. All algorithms were examined with small sample sizes that were selected randomly from the complete training KDD intrusion data set [13], whereas our proposed technique is applied on the complete training data set for building the learning model.

In particular, we used the whole training data set because the testing data used does not come from the same distribution as the training data. Thus, random sampling affects the intrusion detection system because random sampling only reflects the distribution of a training data sample which leads to possible significant regions of the testing data being left out. For these reasons, using a larger training sample is likely to cover more significant regions and build a stronger detection model. Furthermore, the proposed system enables the use of larger amounts of data to build the detection model due to the parallelization, and this leads to higher detection rates.

As far as we know, our proposed system is the first work on the parallelization of intrusion detection systems using the MapReduce methodology, which is considered an alternative model for parallel processing over the MPI methodology [4].

III. Proposed Approach

Given that our proposed intrusion detection system is based on PSO clustering using MapReduce methodology, we first briefly introduce PSO, introduce PSO clustering using MapReduce (MR-CPSO) [14], and then outline the details of our proposed intrusion detection system.

A. Background to Particle Swarm Optimization

Particle Swarm Optimization (PSO) is a swarm intelligence method developed by Kennedy and Eberhart in 1995 [15]. The behavior of PSO mimics the social interaction between individuals such as interactions between the birds in flocks trying to locate an optimal food source. The direction of the movement of each bird is controlled by its current location, the best food location it ever found, and the best food location any bird in the flock ever found.

In PSO, a number of simple entities called particles are placed in the search space of the target problem, and each particle evaluates its position based on a given objective function calculating the fitness value. Each particle then determines its movement (velocity) through the problem search space by taking the history of its own current position and best position achieved by the whole swarm. Furthermore, the movement of a particle is affected by its inertia, and other constants. However, the whole swarm after several iterations is likely to move close to the global best solution.

PSO updates the particles positions inside the problem search space using the following equations:

$$X_i(t+1) = X_i(t) + V_i(t+1) \tag{1}$$

where X_i is the position of particle i, t is the iteration number, and V_i is the velocity of particle i given by the following equation:

$$\begin{aligned} V_i(t+1) = W \cdot V_i(t) + r_1 \cdot c_1 \cdot [XP_i - X_i(t)] \\ + r_2 \cdot c_2 \cdot [XG - X_i(t)] \end{aligned} \tag{2}$$

where W is inertia weight, r_1 and r_2 are randomly generated numbers uniformly distributed between 0 and 1, c_1, c_2 are constant coefficients, XP_i is the current best position of particle i, and XG is the current global best position of the whole swarm.

B. Proposed Intrusion Detection System (IDS-MRCPSO)

Analyzing large network traffic data to detect intrusions takes a long time, thus, our proposed system uses the data clustering concept based on the PSO approach. PSO is parallelized using the MapReduce model as to scale with large-scale network traffic.

The proposed intrusion detection system consists of three main components: preprocessing component, detector model construction component, and validation component. Figure 2 shows the proposed IDS architecture diagram. The preprocessing component follows three consequent steps: missing value record elimination, categorical feature elimination, and data normalization. First, we discard the records that have missing values because we use the records in the distance equation of the clustering technique, and therefore, a record with a missing value is not usable in the equation.

Then, the elimination of the categorical features is done by removing any feature with categorical data. The purpose of this process is to use only numerical data in our distance calculation, because for the categorical data the distance calculations are difficult and depend on the data itself. We could use the matching technique, but it would not help to distinguish the records in terms of the total distance.

At the end of the preprocessing stage, the normalization process normalizes the data set to avoid the bias problem some

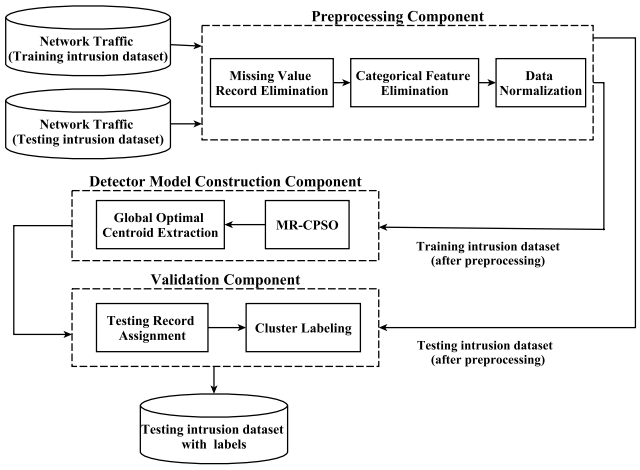


Fig. 2. Proposed IDS-MRCPSO Architecture Diagram

larger features values can cause. Furthermore, the normalization process is applied on the training and testing data sets at the same time, because applying normalization on training and testing data sets separately will create two different normalized data sets, since the minimum and maximum are based on the input data. The normalization process is done using the following equation:

$$X_{ji_{new}} = \frac{X_{ji} - X_{i_{min}}}{X_{i_{max}} - X_{i_{min}}} \tag{3}$$

where X_{ji} is the value of record j for feature i; $X_{i_{min}}$ is the minimum value of feature i; $X_{i_{max}}$ is the maximum value of feature i.

The detector model construction stage starts by applying the MR-CPSO [14] algorithm to the data results from the preprocessing stage, where only training data is used.

In [14], the authors proposed a parallel particle swarm optimization clustering (MR-CPSO) algorithm that is based on the MapReduce programming model. The experimental evaluation using large-scale data sets proved that MR-CPSO scales very well with increasing data set sizes and achieved reasonable clustering quality. The MR-CPSO is a partitioning clustering algorithm which uses individual centroids to represent different clusters. The initial centroids are selected randomly from the data set instances, then the centroids are updated iteratively based on the swarm particles' velocities until convergence to the global best centroid vector is achieved, which then is used in the validation stage. The best centroid vector is evaluated based on the average minimum distances between the data instances and the selected cluster centroids.

The computational complexity of an intrusion detection system depends on the MR-CPSO algorithm used to generate optimal centroids from the training data. This algorithm has quadratic complexity because it requires computation of pairwise distances for all data set instances. In addition, the validation stage of the detection model is relatively fast, whereby it involves comparing testing data records with a small number of generated centroids.

In MR-CPSO, each particle keeps some information which is used in the clustering task such as: centroid vector, velocity vector, fitness value, etc. The particle information is updated in each iteration using the information from the previous iteration. MR-CPSO is divided into three main modules: the first module is responsible to update the particle's centroid vector, the second module is responsible to evaluate the fitness function, and the third module is to merge the outputs of the first and second modules in order to update the swarm. Figure 3 shows the MR-CPSO architecture diagram.

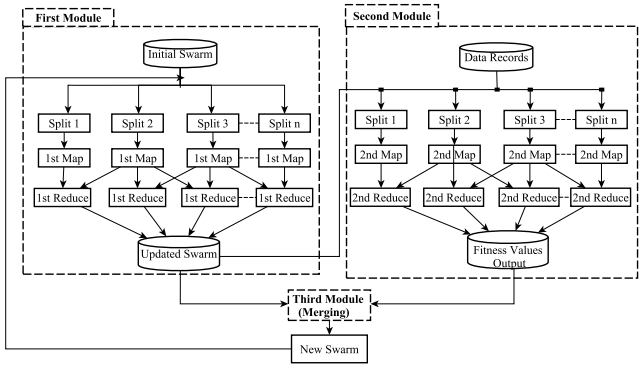


Fig. 3. MR-CPSO Algorithm Architecture Diagram [7]

In the first module, each particle gets an updated centroid vector in each iteration based on PSO movement equations. In this module an individual MapReduce job is triggered, where the *Map* function treats each particle as a Value and particle identification number as a Key. The *Map* function is started by retrieving the *Map* Value which contains all information about the particle. After that, the centroid vector is updated using the previous information based on the PSO equations. At the end, the *Map* function emits the particle with updated centroid vector to the *Reduce* function. The *Reduce* function sorts the *Map* intermediate output according to the Key and merges them into one output intermediate file to be the input for the next module.

The second module of MR-CPSO presents another MapReduce job that evaluates the fitness function using the updated swarm particles. The fitness evaluation depends on measuring the distances between all data records and particle centroid vector. The fitness evaluation is calculated using the total sum of squares errors as follows:

$$Fitness = \frac{\sum_{j=1}^{k} \sum_{i=1}^{n_j} Distance(R_i, C_j)}{k} \tag{4}$$

where n_j denotes the number of records that belong to cluster j; R_i is the i^{th} record; k is the number of available centroids; $Distance(R_i, C_j)$ is the Euclidean distance between record R_i and the centroid C_j.

The *Map* function in this module treats each record as a Value and each record's identification number as a Key. For each particle in the retrieved swarm, the *Map* function extracts the centroid vector from the particle and calculates the distance value between the *Map* Value (record) and the centroid vector,

and then returns the centroid id, which contains the minimum distance, and the particle id to the *Reduce* function.

The *Reduce* function task is to group the values with the same Key together and then to calculate the summation of the minimum distances for each particle centroid. After that, the *Reduce* function emits the Key with total distances to use them as new centroid fitness values.

In the third module of MR-CPSO, the fitness value is calculated for each particle by summing centroids fitness values generated by the second module. Then, the previous swarm fitness values are updated by the new fitness values. After that, the best personal fitness and its centroids are modified based on a comparison between these values and the new fitness values. In addition, if there is any particle that has a fitness value smaller than the current global best fitness value, the global best fitness is assigned the smaller fitness value as well as its centroid vector is updated. At the end, the updated swarm is used for the next iteration. For more details about the MR-CPSO algorithm, the readers can refer to [14].

After the detector model construction stage ends, we extract the global best centroid vector to use them as the detection model in the validation stage. In the validation stage, we used different record subsets called testing data set to evaluate the detection model by calculating the distances between the testing records and the global best centroids vector (detection model). After that, we assigned the testing records to the closed clusters based on the minimum distances. The pseudo-code of the testing records assignment procedure is shown in Algorithm 1.

Algorithm 1 Testing Records Assignment

```
procedure CREATEASSIGNMENT(model, testData, clustersNo)
    featuresNo=extractNoOfFeatures(testingData)
    recordsNo=extractNoOfRecords(testingData)
    assignmentVector= new Array(recordsNo)
    for i = 1 to recordsNo do
        dist = calcEuclidean(model[1], testData[i])
        minDist=dist
        cID=1
        for j = 2 to clustersNo do
            dist=calcEuclidean(model[j], testData[i])
            if dist ≤ minDist then
                minDist=dist
                cID=j
            end if
        end for
        assignmentCluster[i]=cID
    end for
    return assignmentVector
end procedure
```

Finally, the cluster labeling process is triggered to find the correct labels for output clusters generated from the testing record assignment step.

The assignment of cluster labels is accomplished by the maximum percentage of intersections between the original clusters of the testing data, and the clusters that are generated by applying the testing record assignment.

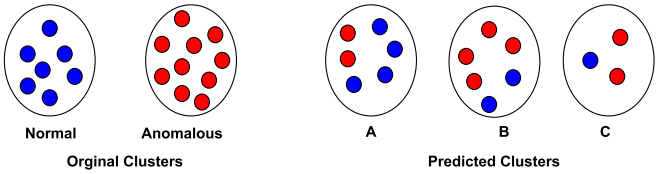


Fig. 4. Clusters labeling process example.

Figure 4 illustrates the cluster labeling process on an example, where the percentage of the normal records in A is PN_A = $\frac{Normal \cap A}{size(A)}$ = $\frac{4}{6}$, and the percentage of anomalous records in A is PA_A = $\frac{Anomalous \cap A}{size(A)}$ = $\frac{2}{6}$; the maximum between these values is max(PN_A, PA_A) = $\frac{4}{6}$; thus, cluster A is a normal cluster. Similarly, for cluster B, the percentage of normal records is PN_B = $\frac{Normal \cap B}{size(B)}$ = $\frac{2}{6}$, PA_B = $\frac{Anomalous \cap B}{size(B)}$ = $\frac{4}{6}$ is the percentage of anomalous records, and max(PN_B, PA_B)=$\frac{4}{6}$; therefore, cluster B is an anomalous cluster. For cluster C, PN_C = $\frac{1}{3}$, PA_C = $\frac{2}{3}$, and max(PN_C, PA_C) = $\frac{2}{3}$; hence C is an anomalous cluster.

IV. Experiments and Results

In this section, we describe the evaluation of the proposed intrusion detection system in terms of detection quality. Furthermore, we discuss the running time and the system speedup of our proposed system.

A. Environment

We ran the experiments on the Longhorn Hadoop cluster hosted by the Texas Advanced Computing Center (TACC)[1]. The cluster contains 384 compute cores and 2.304 TB of aggregated memory and has 48 nodes containing 48GB of RAM, 8 Intel Nehalem cores (2.5GHz each). For our experiments, we used Hadoop version 0.20 for the MapReduce framework, and Java runtime 1.6 for the system implementation. Furthermore, in order to guarantee a constant level of parallelization, the maximum number of mapper and reducer tasks were set to 8 per node since each node contains 8 cores.

B. Data Set Description

To evaluate our proposed system, we used a big intrusion detection data set [13] that has never has been fully analyzed by any standard data mining algorithms. It was used in 1999 as the benchmark at the Knowledge Discovery and Data Mining (KDD99)[2] competition, however, only portions of the data set were analyzed at a time. This data set contains a standard set of data to be audited that includes a wide variety of intrusions simulated in a military network environment.

Each record in the data set represents a connection between two IP addresses, starting and ending at defined times and

[1]https://portal.longhorn.tacc.utexas.edu/
[2]http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

protocol. Every record is represented by 41 features, and each record represents a separate connection and is considered to be independent of any other record. The data is either identified as normal or as one of the 24 different types of attacks. The 24 attacks are grouped into four main types: probing, denial of service (DoS), unauthorized access from a remote machine (R2L), and unauthorized access to root (U2R).

The training data set contains 4,898,431 connection records, which are collected during seven weeks of network traffic. Two weeks are used to produce approximately two million connection records as the testing data set, which is reduced to 311,029 records that are used to evaluate the learning model. Furthermore, it is worth to note that the testing data set contains 7.5% unknown attack types, which are not in the training data set.

A preprocessing process is applied on the training and testing data sets by discarding the records that have missing values and reducing the number of features to 38 by discarding the 3 categorical features such as the protocol and service features. Furthermore, the normalization process is applied on the training and testing data sets.

1) Training Data Set Samples: In order to evaluate the impact of the training data set size in the detector model construction stage, we extracted 5 different samples from the whole training data set. In this paper, we used the stratified sampling by randomly selecting the records from the original training data set by keeping the same ratio between the different classes.

In order to simplify the names of the training data set samples, the sample name consist of the specific format based on the percentage of the whole training data set. For example, the TRAIN20 sample consists of 20% of the whole training data set. The 5 samples are described in Table I.

TABLE I
DATA SET SAMPLES

Sample ID	Percent. (%)	Normal	Anomalies	Total
TRAIN20	20%	194,556	785,130	979,686
TRAIN40	40%	389,112	1,570,260	1,959,372
TRAIN60	60%	583,669	2,355,390	2,939,059
TRAIN80	80%	778,225	3,140,520	3,918,745
TRAIN100	100%	972,781	3,925,650	4,898,431

C. Evaluation Measures

We used the parallel Speedup [16, 17] measure calculated using Equation 5 to evaluate the performance of our proposed system. Speedup is measured by fixing the data set by increasing the number of cluster nodes. The speedup measure is calculated as:

$$Speedup = \frac{T_2}{T_n} \quad (5)$$

where T_2 is the running time using 2 nodes, and T_n is the running time using n nodes, whereby n is a multiple of 2.

For the intrusion detection quality, we used the True Positive Rate (TPR) or Detection Rate (DR) measure which is the ratio between the number of correctly detected attacks and the total number of attacks. Another measure used to evaluate intrusion detection systems is the False Positives Rate (FPR) or False Alarm Rate (FAR) which falsely identifies an intrusion detected that is not an intrusion. FPR is a ratio between the number of false positives and the total number of false positives plus the false negatives.

In addition, we evaluate the IDS's effectiveness by the Receiver Operating Characteristic (ROC) [18] curve which is a plot of the TPR against FPR. Therefore, we used the Area Under Curve (AUC) measure [18] as the ROC curve evaluation to combine the TPR and FPR which is considered a good indicator of their relationship. The AUC is calculated by the following equation:

$$AUC = \frac{(1 - FPR) \times (1 + TPR)}{2} + \frac{FPR \times TPR}{2} \quad (6)$$

We used the PSO settings that are recommended by [19, 20]. We used a swarm size of 100 particles, and an adaptive inertia weight with maximum value W of 0.9. Also, we set the acceleration coefficient constants c_1 and c_2 to 1.49.

D. Results

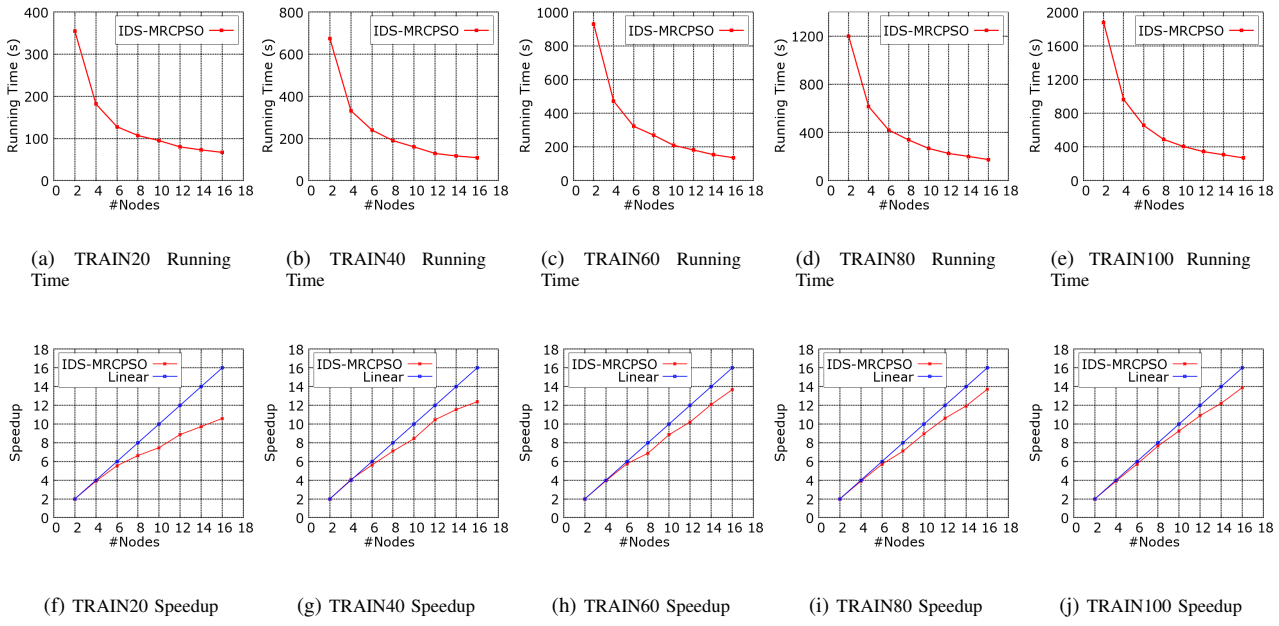
To evaluate the effectiveness of the IDS-MRCPSO system, we run multiple experiments using different sizes of training data that are given in Table I. In Table II, we report the results of the proposed system based on TPR, FPR, and AUC for different training data set sizes. For this experiment, we set the number of PSO iterations to 50, and the number of clusters to 5 which were empirically determined. We observed that the TPR value of IDS-MRCPSO using the complete training data set (TRAIN100) achieves the best TPR and AUC value compared to other smaller training data sets. In addition, TRAIN100 obtains the lowest FPR results of all training data sets. For example, the IDS-MRCPSO system has a high TPR of 0.939 for TRAIN100, while it has a TPR of 0.903 for TRAIN20. For TRAIN100, the FPR value is 0.013, while for TRAIN20 the FRP is 0.038. The AUC value for TRAIN100 is 0.963 while the AUC value for TRAIN20 is 0.933. Hence, the results show our system can distinguish between the normal data records and anomaly records effectively. The results demonstrate that using larger training data, better results can be achieved.

TABLE II
PROPOSED IDS-MRCPSO SYSTEM RESULTS

Sample ID	TPR	FPR	AUC
TRAIN20	0.903	0.038	0.933
TRAIN40	0.911	0.021	0.945
TRAIN60	0.927	0.015	0.956
TRAIN80	0.935	0.013	0.961
TRAIN100	0.939	0.013	0.963

Figure 5 shows the ROC curve using the proposed IDS-MRCPSO system. The figure shows that the best performance with a high AUC value is achieved when using TRAIN-100 compared to the other curves.

To investigate the scalability of the proposed system, we ran multiple experiments with different number of nodes. In each



(a) TRAIN20 Running Time (b) TRAIN40 Running Time (c) TRAIN60 Running Time (d) TRAIN80 Running Time (e) TRAIN100 Running Time

(f) TRAIN20 Speedup (g) TRAIN40 Speedup (h) TRAIN60 Speedup (i) TRAIN80 Speedup (j) TRAIN100 Speedup

Fig. 6. Running time and speedup results on the KDD data set samples. 6(a)-6(e) Running time for KDD data set samples from 20% to 100% sizes. 6(f)-6(j) Speedup measure for KDD data set samples from 20% to 100% sizes.

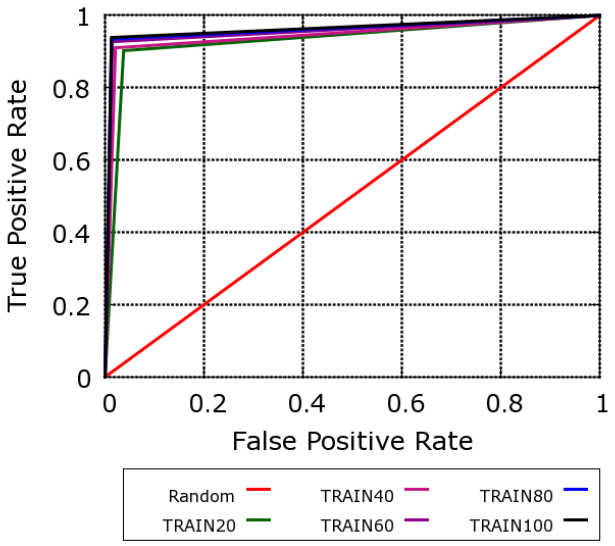


Fig. 5. ROC Results

experiment, we report the running time and speedup taking the average of 25 PSO iterations. The running times and speedup measures are shown in Figure 6. In Figures 6(a)-6(e), the running time results are reported for the 5 training data sets for different number of Hadoop cluster nodes. All subfigures show that the running time improves faster for 2 nodes and 4 nodes than at the end when the number of nodes is 16 nodes. Furthermore, the impact of the training data set on the running time is well observed. The running time on 2 nodes takes 355, 675, 930, 1200 and 1875 seconds for TRAIN20, TRAIN40, TRAIN60, TRAIN80, and TRAIN100, respectively, while the running time on 16 nodes takes 67, 109, 136, 175 and 250 seconds for the same samples, respectively. As can be seen, the improvement factor of the running times for 16 nodes compared to the running time with 2 nodes are 5.30, 6.19, 6.83, 6.86, and 6.94, respectively.

In Figures 6(f)-6(j), the speedup results using different training data set sizes with different numbers of nodes are shown. As can be observed from these figures, the speedup for TRAIN20 is very close to the linear speedup using 4, and 6 nodes. It begins to diverge from the linear speedup around 8 nodes that can be attributed to the overhead of the Hadoop framework such as starting MapReduce jobs and storing intermediate outputs to the distributed file system.

The same trend is observed for TRAIN40, TRAIN60, TRAIN80 and TRAIN100. For TRAIN40, the speedup is very close to the linear one using 2 to 8 nodes, but it starts to diverge from the linear line with little difference compared to TRAIN20. For TRAIN60 and TRAIN80, the speedup is close to the linear one with 10 and 12 nodes, and it then starts to have a larger difference for larger numbers of nodes. We can summarize that the overhead of the Hadoop framework is reduced for larger data sets and the speedup is closer to the linear one. Furthermore, the speedup scales close to linear for most training data sets samples. The proposed system with TRAIN100 achieves a significant speedup getting very close to the linear speedup. The speedup results showed reasonable scalability for the proposed system.

V. Conclusion

In this paper, we proposed an IDS-MRCPSO system for intrusion detection using the MapReduce methodology to solve the management of large-scale network traffic. We have

shown that the intrusion detection system can be parallelized efficiently with the MapReduce methodology. Experiments were performed on a real intrusion data set in order to measure the system speedup. The experimental results reveal that IDS-MRCPSO is efficient with increasing training data set sizes, and scales very close to the optimal speedup by improving the detection results. Furthermore, we used the whole training data to build the detection model to avoid the random sampling effects, thus, this technique covers more significant regions of the training data set, and builds a stronger detection model. The results validate that using larger training data leads to better detection rates by keeping the false alarm very low.

Our future plan is to include experiments with new intrusion data sets. Furthermore, we will expand the system such as to distinguish between the different types of intrusions and not only whether an intrusion has occurred or not.

ACKNOWLEDGMENT

This work used the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by National Science Foundation grant number OCI-1053575. The authors acknowledge the support of the NDSU Advance FORWARD program sponsored by NSF HRD-0811239 and ND EPSCoR through NSF grant EPS-0814442.

REFERENCES

[1] A. Lazarevic, V. Kumar, and J. Srivastava, "Intrusion detection: A survey," *Managing Cyber Treats*, pp. 19–78, 2005.

[2] H. Debar and J. Viinikka, "Intrusion detection: Introduction to intrusion detection and security information management," *Foundation of Security Analysis and Design III*, vol. 3655, pp. 207–236, 2005.

[3] J. Han, *Data Mining: Concepts and Techniques*. Morgan Kaufmann, San Francisco, CA, USA, 2005.

[4] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra, *MPI: The Complete Reference*. MIT Press Cambridge, MA, USA, 1995.

[5] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," in *Proceedings of the OSDI '04*, 2004, pp. 137–150.

[6] T. White, *Hadoop: The Definitive Guide*, original ed. O'Reilly Media, Jun. 2009.

[7] K. Leung and C. Leckie, "Unsupervised anomaly detection in network intrusion detection using clusters," in *Proceedings of the Twenty-eighth Australasian conference on Computer Science*. Newcastle, Australia: Australian Computer Society, 2005, pp. 333–342.

[8] M. Jianliang, S. Haikun, and B. Ling, "The application on intrusion detection based on k-means cluster algorithm," in *Proceedings of the 2009 International Forum on Information Technology and Applications*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 150–152.

[9] W. Jiang, M. Yao, and J. Yan, "Intrusion detection based on improved fuzzy c-means algorithm," in *Proceedings of the 2008 International Symposium on Information Science and Engieering*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 326–329.

[10] Z. Li, Y. Li, and L. Xu, "Anomaly intrusion detection method based on k-means clustering algorithm with particle swarm optimization," in *Proceedings of the 2011 International Conference of Information Technology, Computer Engineering and Management Sciences*. Washington, DC, USA: IEEE Computer Society, 2011, pp. 157–161.

[11] J. Mazel, P. Casas, Y. Labit, and P. Owezarski, "Subspace clustering, inter-clustering results association & anomaly correlation for unsupervised network anomaly detection," in *Proceedings of the 7th International Conference on Network and Services Management*, Paris, France, 2011, pp. 73–80.

[12] H. Gao, D. Zhu, and X. Wang, "A parallel clustering ensemble algorithm for intrusion detection system," in *Proceedings of the DCABES'10 Conference*. Washington, DC, USA: IEEE Computer Society, 2010, pp. 450–453.

[13] S. D. Bay, D. Kibler, M. J. Pazzani, and P. Smyth, "The uci kdd archive of large data sets for data mining research and experimentation," *SIGKDD Explor. Newsl.*, vol. 2, pp. 81–85, December 2000.

[14] I. Aljarah and S. A. Ludwig, "Parallel particle swarm optimization clustering algorithm based on mapreduce methodology," in *Proceedings of the Fourth World Congress on Nature and Biologically Inspired Computing (NaBIC'12)*, Mexico City, Mexico, November 2012, pp. 104–111.

[15] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of the 1995 IEEE International Conference on Neural Networks*. Brisbane, Australia, 1995, pp. 1942–1948.

[16] A. Grama, A. Gupta, G. Karypis, and V. Kumar, *Introduction to Parallel Computing*. Addison-Wesley, USA, 2003.

[17] Z. Gao, T. Li, J. Zhang, C. Zhao, and Z. Wang, "A parallel method for unpacking original high speed rail data based on mapreduce," *Springer Berlin Heidelberg*, vol. 124, pp. 59–68, 2012.

[18] W. Zhu, N. Zeng, and N. Wang, "Sensitivity, specificity, accuracy associated confidence interval and roc analysis with practical sas implementations," in *In Proceedings of the NorthEast SAS Users Group Conference NESUG10*, 2010.

[19] H. Shi and R. Eberhart, "Parameter selection in particle swarm optimization," in *Proc. 7th Annual Conference on Evolutionary Programming*, San Diego, CA, 1998, pp. 201–208.

[20] I. Trelea, "The particle swarm optimization algorithm: convergence analysis and parameter selection," in *Information Processing Letters*, 2003.