Int J Parallel Prog (2017) 45:1194–1213
DOI 10.1007/s10766-016-0456-z

Hadoop Based Parallel Binary Bat Algorithm for Network Intrusion Detection

P. Natesan[1] · R. R. Rajalaxmi[1] · G. Gowrison[2] · P. Balasubramanie[1]

Received: 28 February 2016 / Accepted: 22 September 2016 / Published online: 30 September 2016

Abstract In Internet applications, due to the growth of big data with more features, intrusion detection has become a difficult process in terms of computational complexity, storage efficiency and getting optimized solutions of classification through existing sequential computing environment. Using a parallel computing model and a nature inspired feature selection technique, a Hadoop Based Parallel Binary Bat Algorithm method is proposed for efficient feature selection and classification in order to obtain optimized detection rate. The MapReduce programming model of Hadoop improves computational complexity, the Parallel Binary Bat algorithm optimizes the prominent features selection and parallel Naïve Bayes provide cost-effective classification. The experimental results show that the proposed methodologies perform competently better than sequential computing approaches on massive data and the computational complexity is significantly reduced for feature selection as well as classification in big data applications.

✉ P. Natesan
natesanp@gmail.com

R. R. Rajalaxmi
rrr_kec@yahoo.co.in

G. Gowrison
gowrison@rediffmail.com

P. Balasubramanie
pbalu_20032001@yahoo.co.in

[1] Department of Computer Science and Engineering, Kongu Engineering College, Perundurai, Erode, Tamil Nadu 638 052, India

[2] Department of Electronics and Communication Engineering, Institute of Road and Transport Technology, Erode, Tamil Nadu 638 316, India

Keywords Hadoop · Parallel Binary Bat · MapReduce · Feature selection · Classification

1 Introduction and Related Work

Due to the expansion of computer networks, the number of intrusion incidents is increasing gradually. This may challenge the network administrators to prevent unauthorized access to confidential and privileged information and this had led many researchers to focus on constructing a system called intrusion detection system (IDS). Various sorts of IDS approaches can be emphasized on the network access events in the context of system security management. The two common approaches are misuse detection and anomaly detection [1,12,31,34]. In the former the detection is about observed behaviors that match a predefined pattern of events described as known attacks. The later approach aims at detecting user's behavior that move away from the normal profile [5]. The drawback in misuse detection is that it cannot detect undefined attacks. To overcome this problem, anomaly intrusion detection is used which can predict an undefined attack by detecting any deviation from the learned user's normal profile.

In order to resolve the disadvantages of these two intrusion detection approaches, several hybrid IDSs have been proposed recently by combining different machine learning techniques. Pfahringer [23], the winner of the KDDCup99, developed a cost-effective bagged boosting algorithm by integrating C4.5 decision trees, while Levin [16] built an optimal decision forest using Kernel Miner technique. Both have acceptable detection rates in detecting Normal, DoS and Probe attacks but undesirable rate in detecting R2L and U2R attacks. Depren et al. [5] suggested a hybrid IDS consisting of an anomaly detection component, a misuse detection component and a decision support system. To improve the detection rate further, Peddabachigari et al. [22] designed an IDS by fusing decision trees and support vector machines, while Xiang et al. [35] constructed a multiple-level tree classifier which contains three-levels of decision tree classification. However, these two methods suffer from low detection rates for unknown attacks.

Adaboost based intrusion detection systems are developed to improve the attacks detection rate in which the output of a classifier is boosted further by adjusting the learning weights [7,11,13,21,33]. These approaches promised a higher detection rate with a minimum false alarm rate, but suffered from higher resource consumption and slow training and testing processes. To overcome these issues many researchers have proposed feature selection techniques.

Feature selection (FS) is defined as the process of choosing an optimal prominent subset of features that represents the entire dataset [27,28]. Principal Component Analysis (PCA) is one of the most widely used feature reduction technique for data analysis and compression. Venkatachalam and Selvan [29] applied the three feature reduction techniques of Principal Component Analysis (PCA), Linear Discriminant Analysis (LDA) and Independent Component Analysis (ICA) to the KDDCup99 dataset. These techniques perform a linear mapping of the data to a lower dimensional space in such a way that the variance of the data in the low dimensional depiction gets

maximized. The dataset with reduced attributes are applied to Binary tree, ART and LAMPSTAR Neural Network classifiers with improved results.

Parallel computing may be a good solution to feature selection and classification, in which computations are carried out concurrently. Task parallelism tries to run different tasks in parallel while data parallelism targets to execute the same process on multiple data sets. These two parallel mechanisms are utilized to parallelize the feature selection process. Mohammad et al. [20] distributed the entire dataset into multiple nodes in the cluster and applied the feature selection algorithm to get sufficiently good subsets of attributes in a high dimensional search space through parallel computing for synthetic datasets only. Deng et al. [4] decomposed the whole high-dimensional dataset into many sub-tables and suggested an approach for parallel feature selection from a series of decision subsystems.

Recently, a simple and parallel computation approach called MapReduce [3], has been used by many researchers, which is a highly scalable parallel programming model for data-intensive and computation-intensive applications on a cluster of commodity machines. MapReduce has been successfully applied in data mining [10,26,30] and machine learning [2,9,14,17,18,24,37].

The salient features of MapReduce model are as follows. It hides many system-level details such as hardware heterogeneity and platform differences from the user and without human intervention parallelize the computation across large-scale clusters of machines. This model also handles machine failures in the cluster and schedules communications among the cluster of machines to make an efficient use of networks and storage media.

The main contributions of this paper can be enumerated as follows.

1. Parallel version of Binary Bat Algorithm (PBBA) using the iterative MapReduce programming model to select the prominent subset of features.
2. Distributed classification model using Naive Bayes algorithm to improve the attacks classification accuracy and attacks detection time.

The subsequent sections of the paper are organized as follows. Section 2 presents sequential Bat and sequential Binary Bat algorithms. The MapReduce based Parallel Binary Bat algorithm is presented in Sect. 3. The sequential and parallel Naïve Bayes classification algorithms are presented in Sect. 4. Section 5 presents the experimental results and discussion, while Sect. 6 gives the conclusions of the proposed work.

2 Overview of the Bat Algorithms

2.1 Bat Algorithm

Bat Algorithm (BA) has been developed based on the echolocation behavior of bats. Typically, bats emit a short pulse of sound and receive the echo of the sound after a fraction of time which is used to determine their remoteness from an object. Also, they have the capability to distinguish between an obstacle and a prey which helps them to hunt in a complete dark environment [36]. In BA, an artificial bat has a position, velocity and frequency vectors which are updated during the course of iterations. The

artificial bats move around the search space utilizing the position and velocity vectors or updated position vectors within the continuous real domain.

For each bat (b_i), there is a position (x_i), frequency (f_i) and velocity (v_i). At each step of t, the bats move to the next position with new velocities which is

$$v_i\,(t+1) = v_i\,(t) + (x_i\,(t) - gbest)\,f_i \tag{1}$$

where gbest is the best solution obtained so far. After computing the velocity, the position of the bat is updated as follows.

$$x_i\,(t+1) = x_i\,(t) + v_i\,(t+1) \tag{2}$$

The frequency of ith bat is computed in each iteration as follows.

$$f_i = f_{min} + (f_{max} - f_{min})\,\beta \tag{3}$$

where β is a random number uniformly distributed in the range [0,1]. The exploitation capability of BA is improved by using a random walk method which is given by

$$x_{new} = x_{old} + \varepsilon A' \tag{4}$$

where ε is a random number between $[-1,1]$ and A' is the loudness of the emitted sound. The loudness and pulse emission rate are adjusted in each iteration, as follows.

$$A_i\,(t+1) = \alpha\,A_i \tag{5}$$

$$r_i\,(t+1) = r_i\,(t) + \left[1 - \exp\left(-\gamma t\right)\right] \tag{6}$$

where α and γ are constants.

2.2 Binary Bat Algorithm

In the feature selection problem, each feature subset is coded as a binary string of 1s and 0s where 1 denotes the presence and 0 represents the absence of a feature. To map the problem with BA, each bats position is considered as a binary value which creates a binary search space and the bats can move to a new position by flipping various numbers of bits. In the continuous version of BA, the artificial bats can move around the search space by utilizing the position and velocity vectors within the continuous real domain. Consequently, the concept of position updating can be easily implemented for bats by adding velocities to positions and however, the meaning of position updating is different in a discrete binary space. The position updating process cannot be performed like the BA which has a continuous version and the movement of the bat in the search domain has only the binary values '0' and '1', known as binary BA. Hence, the binary version of BA employs a different strategy to update its velocity and position. Therefore, a transfer function is used to change the bats' positions from "0" to "1" or

vice versa [15,19,25]. The transfer function for updating the bat position in the binary space is

$$V\left(v_i^k\right) = \left|\frac{2}{\pi}\arctan\left(\frac{\pi}{2}\right)v_i^k\right| \tag{7}$$

where v_i^k is the velocity of the bat 'i' in kth dimension. Now the position of the bat is modified as

$$x_i^k\,(t+1) = \begin{cases} 0\ if\ \ V(v_i^k\,(t+1)) > \delta \\ 1\ if\ \ V(v_i^k\,(t+1)) < \delta \end{cases} \tag{8}$$

where δ is uniformly distributed between [0,1]. After updating the position, the fitness of each bat is evaluated.

2.3 Fitness Function for BBA

Feature selection (FS) is defined as the process of choosing an optimal prominent subset of features that represents the entire dataset. All of these feature selection methods may be grouped into filter based and wrapper based approaches. The filter approach relies on general characteristics of high-dimensional datasets with fast evaluation and feature selection in subsets and it does not take into account mining algorithms. But, the wrapper approach is an optimizing algorithm that adds or removes features for construction of various subset features and then employs a mining algorithm to evaluate the subset of features.

The present work deals with unsupervised feature selection using the k-means clustering algorithm with wrapper approach in order to classify a given dataset through a certain number of clusters. The goal of the k-means algorithm is to find k points of a dataset where point 'k' is the cluster center or centroid of each cluster. In particular, k-means clustering algorithm is used to cluster or group 'n' data points into 'k' disjoint subsets s_j, containing n_j data points so as to minimize the mean square error (MSE) and is given as,

$$\text{Mean Square Error (MSE)} = \sum\nolimits_{j=1}^{k}\sum\nolimits_{n\in s_j}\left|x_n - \mu_j\right| \tag{9}$$

where x_n is a feature vector representing the nth data point and μ_j is the geometric centroid of the data point in s_j. Thus, fitness is evaluated based on the cluster quality measured using Mean Square Error.

3 Hadoop Based Parallel BBA (HPBBA)

Hadoop is a framework proposed by Google which provides a fault tolerant and reliable environment for parallel algorithms and the programmers can place various components of the distributed application in the networked computers resourcefully. Furthermore, to handle large volume of data, this framework supports a scalable file system called Hadoop Distributed File System (HDFS) which stores very large files in blocks across various machines in a large cluster of commodity hardwares.

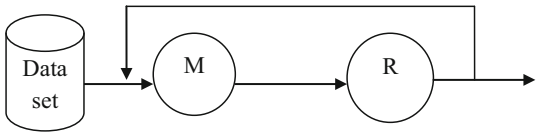

Fig. 1 Iterative MapReduce model

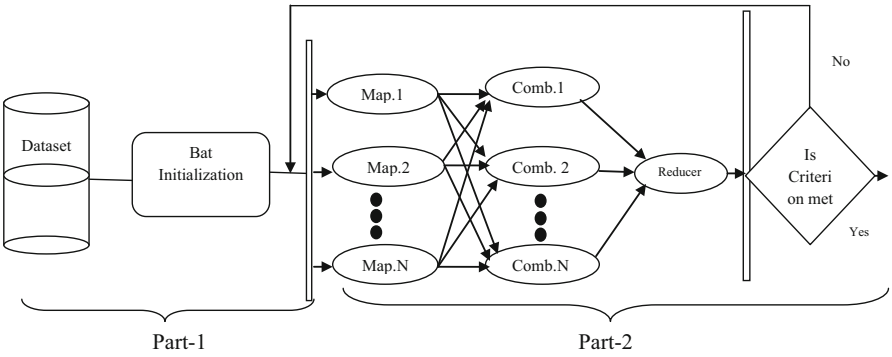

Fig. 2 Mapper–Chainer model for Parallel Binary Bat algorithm

In general, nature inspired computing algorithms need high computational resources to find an optimal solution within exhaustive search spaces. To reduce computation time and memory requirement, Hadoop parallel implementation is employed with binary BA. Hence, nature inspired computing algorithms can be articulated as iterative procedures to obtain optimal solutions with minimum computing resources and storage requirements. The iterative operations can be expressed as fitness evaluation, finding local optimum solution and global solution over the distributed data. We establish an extension of the MapReduce programming model, known as Iterative MapReduce method, which supports iteration as a primary construct. The proposed iterative MapReduce model is used for feature selection of KDDcup99 dataset which is illustrated in Fig. 1.

The different observations about the iterative MapReduce programming model are: (1) there is a one-to-one correspondence between the map task and the reduce task, since the execution unit in both the map task and reduce task is the processor core (2) Each iteration is articulated as only one MapReduce job (3) the locality optimization supported by the Hadoop framework reduces the loading effort of DFS data in to a map for each iteration.

The proposed feature selection methodology shown in Fig. 2 adopts the Mapper–Chainer model with two parts. The first part consists of a single MapReduce job responsible for bat initialization and the second part is an iterative MapReduce job to determine the best optimal solution.

Part-1: Bat Initialization: The bat search space is assumed as an area that contains many food/prey sources on it. Since, the best of food/prey source locations are not known ahead, it is required that an initial population is randomly generated from an integer-valued vector with a number of bats 'n_b' and dimension 'd'. The bats are initialized as per Algorithm 1 with the initial value for the minimum frequency set

as 0 ($f_{min} = 0$) and the maximum frequency set as 1 ($f_{max} = 1$). The pulse rate r_i and loudness A_i for each bat are also initialized. After the initialization of various bat parameters, the initial fitness value for each bat is calculated and finally the best current bat is obtained.

Algorithm 1. HPBBA – Bat Initialization

input : partitioned dataset ; output : initialized population

*/ * initialize the bat population */*

***FOR** i ← 1 to n_b*

***FOR** j ← 1 to d*

$x_j \leftarrow$ *rand(0 or 1) ;*

ENDFOR

/ initialize the bat parameters */*

$f_{min} \leftarrow 0$; $f_{max} \leftarrow 1$; $r_i \leftarrow 0.5$; $A_i \leftarrow 0.25$;

/ compute the fitness for each bat 'b_i' as per equation (9) */*

$f(b_i) \leftarrow \sum_{j=1}^{k} \sum_{n \in s_j} |x_n - \mu_j|$;

current gbest ← best [$f(b_i)$];

ENDFOR

The dataset 'D' is partitioned into 64 MB size multiple input files, stored by the HDFS across data nodes of a Hadoop cluster. After the bat populations are randomly initialized throughout the search space and after obtaining the best bat, (composite key, values) pairs are obtained as inputs for the iterative MapReduce part. In complex operations, where several parameters are used, the basic (key, value) pair is not sufficient. Hence, a composite key is created using Writable Comparable interface object of Hadoop framework.

Part-2: Iterative MapReduce: The map tasks in the iteration will be started after completion of bat initialization. Each map function sequentially reads each partitioned dataset from its local input split as per the format (composite key, values). Now, multiple map functions runs in parallel mode and the fitness value ($f_p(b_i)$) is computed for each bat with the given partitioned dataset. The pseudo code of HPBBA-Map is given in Algorithm 2.

Algorithm 2. HPBBA-Map(key, value)

input : key<input file offset, bat id> ; value < partitioned instances, fitness of each bat>

output : key'<partition id, bat id> ; value'< new fitness value of each bat for its partition>

***FOR** i ← 1 to n_b*

/ update velocity and frequency parameters */*

$v_i(t+1) \leftarrow v_i(t) + [x_i(t) - gbest] f_i$;

$f_i \leftarrow f_{min} + (f_{max} - f_{min})\,\beta$;

/ evaluate current fitness value */*

$f_p(b_i) \leftarrow \sum_{j=1}^{k} \sum_{n \in s_j} |x_n - \mu_j|$;

key' ← (partition id, b_i) ;

value' ← $f_p(b_i)$;

emit <key', value'>

ENDFOR

Next, these fitness values emitted by different map functions of the 'm' different partitions with dataset are summed up in the combiner for 'n_b' bats. The average fitness value for each bat of the entire dataset ($f_D(b_i)$) is obtained as per Algorithm 3.

Algorithm 3.HPBBA-Combine(key,value)

input: key<partition id, b_i> ; value< $f_p(b_i)$>
output: key'<partition id, bat id> ; value'< fitness of each bat for dataset D>
***FOR** i ← 1 to n_b*
  ***FOR** j← 1 to m*
    $tf_D(b_i) \leftarrow f_D(b_i) + f_{pj}(b_i)$;
  ENDFOR
  $f_D(b_i) \leftarrow tf_D(b_i)/m$;
  key' ← (partition id, b_i);
  value' ← $f_D(b_i)$;
  emit < key', value'>
ENDFOR

Finally, the combiner emitted values in the form of (composite key, values) are obtained by Algorithm 4. The next position of the bat is updated and a new solution is generated. The best solution among the solutions from the bats is stored in Gbest state variable and the updated bat with its fitness is passed to the next iteration.

Algorithm 4.HPBBA-Reduce(key, value)

input : key<partition id, b_i> ; value<$f_D(b_i)$>
output: key'<partition id, bat id> ; value' <best fitness>
***FOR** i ← 1 to n_b*
  update x_{new} ;
  /* choose gbest randomly */
  if (rand() > r_i)
    gbest ← rand(best solution);
  end if
  /*generate a new solution by flying randomly */
  if [(rand < A_i) and ($f(b_i)$ < gbest)]
    $A_i(t+1) \leftarrow \alpha A_i$;
    $r_i(t+1) \leftarrow r_i(t) + [1 - exp(-\gamma t)]$;
  end if
  /* rank the bats and find the current gbest */
  gbest ← $f_{gbest}(b_i)$;
  key'<partition id, b_i> ;
  value'<$f_{gbest}(b_i)$> ;
  emit< key', value'> ;
ENDFOR

Part-2 of Fig. 2 is executed until the termination criteria are met. The pseudo-code for the proposed feature methodology is given in algorithm 5.

Algorithm 5.HPBBA

input : dataset D

output : reduced dataset with selected features

```
        /*  initialize feature set fs */
        fs ←φ ;
         /*  bat initialization */
          CALL (algorithm 1)
WHILE (t <= maximum_number_of_iterations)
        block ←partition(D) ;
        /* create m instances of map */
        map tasks ← m;
        CALL (algorithm 2 )
        /* fitness computation */
        CALL (algorithm 3 )
        /* gbest computation */
        CALL (algorithm 4 )
END WHILE
          fs← features(best_bat)
```

It can be seen here that the various stages of the Binary Bat algorithm can be combined with iterative MapReduce in a trouble-free approach.

3.1 Time Complexity of HPBBA

The running time of the HPBBA is the total time required to initialize the bat (Algorithm 1) and to perform the iterative MapReduce operation. The time complexity to initialize the bat population is $T_{init} = t_k * m * (2nkd + nk + nd) / n_c$, where n is the number of instances, d is the number of attributes, k is the number of clusters, t_k is the number of iterations required for k-means to converge, n_c is the number of cores in the cluster and m is the number of partitions of the dataset.

The time complexity of the map function in part-2 of iterative MapReduce is $T_{map} = n_b * t_k * (2nkd + nk + nd) / n_c$, where n_b is the number of bats. The combiner function requires $T_{com} = n_b * c$ time and the reducer takes a time of $T_{red} = n_b * r$. Here c is the number of combiners and r is the number of reducers. The total running time 'T' of HPBBA is $T_{init} + T_{map} + T_{com} + T_{red}$.

In the total running time 'T', the map function which uses k-means clustering to cluster the instances of the dataset requires the highest computation time T_{map}. This T_{map} increases linearly with the number of attributes as well as instances of the dataset. Also, it is assumed that the time for node communication, which depends on quality of the network, is constant. Hence, $T_{init} + T_{com} + T_{red}$ may not be considered for running time execution and the resultant running time is based only on T_{map} and the time complexity of HPBBA is $O(n_b * t_k * (2nkd + nk + nd) / n_c)$.

4 Naive Bayesian Classification

A Naive Bayes classifier is a simple probabilistic classifier based on Bayes theorem with effective independence assumptions. It assumes that the presence of a particular attribute of a class label such as normal or attack categories in a dataset is unrelated to the presence of any other attribute, for the class condition variable.

The Naive Bayes classification model is trained with a known training data set. The computation of maximum likelihood value for the occurrences of events is used to estimate the required parameters of Naive Bayes model. This characteristic of Naive Bayes enables the user to work with the Naive bayes model without considering the Bayesian probability or applying any Bayesian approaches in practical applications [6]. The following features of the Naive Bayes classifier make it very useful in many research domains.

- Each distribution can be independently estimated as a one dimensional distribution
- Minimum computational cost is ensured
- Provides efficiency to handle missing values and noise in the dataset
- The variance is very low due to a lesser amount of searching.
- Naive Bayes follows Incremental learning and due to this characteristic the new training samples are learnt quickly.

The Parallel Naïve Bayesian Classifier (PNBC) is implemented as a combination of map and reduce functions. The pseudo code of the map function is shown in Algorithm 6. Each map function sequentially reads each vertically partitioned dataset from its local input split as per the format (Composite Key, Values). Multiple map functions run in parallel and compute the conditional probability values for the given vertically partitioned dataset.

Algorithm 6. PNBC-Map (key, value)

input : key<input file offset, attributes> ; value < partitioned instances>
output : key'<attributes and classes> ; value'< conditional probability>
/ for attributes $A_1,A_2,\ldots,A_{dp}$ and class label $C_1,C_2,\ldots,C_l$ of vertically partitioned dataset D_p */*
***FOR** j ← 1 to l*
  ***FOR** i ← 1 to dp*
    $p(C_j|A_i) \leftarrow p(A_i|C_j) * p(C_j)$;
  ENDFOR
ENDFOR
  key'<attributes, classes> ;
  value'<conditional probability for all classes> ;
  emit < key', value'>

Next, these conditional probability values emitted by different map functions for the 'm' different partitions and 'd' attributes are multiplied in the reducer for the same classes of the entire dataset. The pseudo code of the reducer function is shown in Algorithm 7. The class with the maximum obtained conditional probability is chosen as the class label for the given input X.

Algorithm 7. PNBC-Reduce (key, value)

```
input: key<attributes, classes> ; value <conditional probability for various classes>
output: key'< class> ;   value' <highest conditional probability>
/*  highest conditional probability for given condition variable X */
FOR j ←1 to l
  FOR  i ←1 to m
          calculate  p(Cj|Xi) from all map functions ;
  ENDFOR
       label(Cj) ←highest [p(Cj|Xi)]
ENDFOR
        key'<classes> ;
         value'<highest conditional probability> ;
       emit < key', value'>
```

The implementation of the parallel Naive Bayesian classifier is shown in the above algorithms, where evaluation of the conditional probabilities $p(C_j|X_i)$ is done in parallel, by distributing the attributes and instances among different cores in the cluster of machines.

4.1 Time Complexity

Let n be the number of instances in the dataset and d is the number of attributes. The dataset is distributed among n_c cores in the cluster. The computational complexity of PNBC is $O(n * d/n_c)$ and it shows that the computational time reduces gradually when the number of cores in the cluster increases.

5 Experiments and Results Analysis

In this section, the datasets, software and cluster configuration used in the experiment are described. The feature selection results using the proposed HPBBA and the classification results using the proposed PNBC are presented. The detection results from the proposed parallel classification approach as well as those of the winners of KDDCup99 competition and other published work in available literature are compared.

5.1 Descriptions of Datasets, Software and Cluster Configuration

The proposed method for intrusion detection was evaluated using the KDDCup99 dataset which is widely used as a benchmark dataset by many researchers for IDS evaluation. The three independent sets of KDDCup99 are the full KDDCup99 training set, 10 % KDDCup99 training set and 10 % KDDCup99 test set. Each record represents a network connection described by 41 features and a class label specifying the type of this record as either normal or one of 39 specific attack types. The attack types in KDDCup99 can be categorized as either a normal class or one of four attack classes, namely Denial of Service (DoS), Probe, Remote to Local (R2L) and User to Root

Table 1 The number of instances in the training and testing datasets

Attack category	Number of instances in		
	Full KDDCup99 training data	10 % KDDCup99 training data	10 % KDDCup99 test data
Normal	972,778	97,277	60,593
Dos	3,914,580	391,458	229,853
Probe	41,102	4107	4166
R2L	1126	1126	16,189
U2R	52	52	228
Total	4,929,638	494,020	311,029

Table 2 List of features selected using PBBA

Total number of features selected	Names of the feature
24	duration, service, flag, src_bytes, dst_bytes, hot, num_failed_logins, logged_in, num_compromised, root_shell, num_root, num_file_creations, num_access_files, is_guest_login, srv_count, dst_host_count, dst_host_srv_count, dst_host_same_srv_rate, dst_host_diff_srv_rate, dst_host_same_src_port_rate, dst_host_srv_diff_host_rate, dst_host_srv_serror_rate, dst_host_rerror_rate and dst_host_srv_rerror_rate

(U2R). Table 1 lists the number instances present in the KDDCup99 training and test datasets.

The performance of our algorithms are evaluated with an in-house Hadoop cluster equipped with 16 nodes. Each node has an Intel core-i5 3.2 GHz quad core processor, 8 GB main memory and runs on the Ubuntu 12.1 operating system, on which Java JDK 1.6, Hadoop 2.6 and Apache Mahout Library are installed. All the nodes in the cluster are equipped with Gigabit Ethernet network interface cards and connected to the Gigabit ports on the high speed manageable switch. The nodes in the cluster use the secure shell protocol to communicate with one another. The default Hadoop parameter configurations are used to set the replication factor as 3 and the number of Map tasks per core is set as one.

5.2 Results for Feature Selection Experiment

The feature selection experiment using PBBA has been performed on full KDD-Cup99 dataset with 41 features and the most significant features that contribute to the improvement of classification accuracy are listed in Table. 2.

In the following sections, the various other characteristics of the proposed work such as speedup, scaleup and sizeup are observed by employing different sizes of KDDCup99 training dataset on a cluster of different sizes.

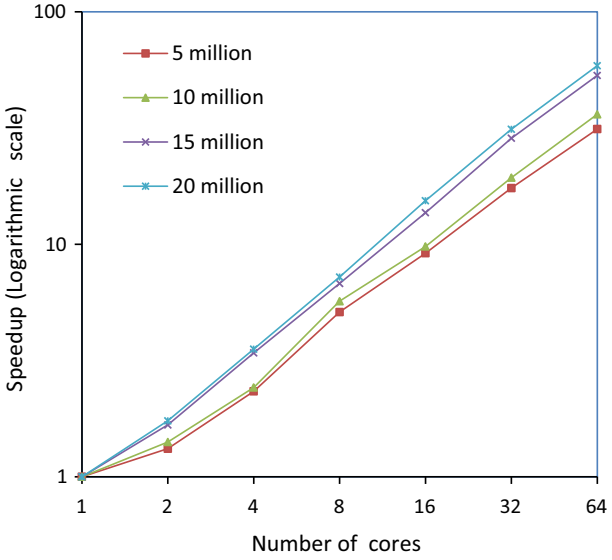

Fig. 3 Speedup of HPBBA

5.3 Speedup

Speedup or strong scalability evaluates the capability of the parallelism and is used to enhance the execution time. It is defined as the ratio of the sequential execution time to the parallel execution time. Speedup can be expressed as

$$Speedup\,(m, D) = \frac{ET\,(1)}{ET\,(m)} \tag{10}$$

where m is the number of cores in the computing cluster, ET(1) is the execution time of the tasks on one core of the computing node, ET(m) is the execution time of the parallel tasks with m cores [24,32].

To measure the speedup performance of our algorithm, the full KDDCup99 dataset of 5 million instances is used in a single core and the number of cores has been increased in a step by step manner. In particular, HPBBA algorithm is first applied in a system consisting of 1 core, and then gradually increased. The number of cores in the experiment varies from 2 to 64 in the order of 2, 4, 8, 16, 32 and 64. The speedup evaluation experiment on datasets has been repeated with different sizes by duplicating the full KDDCup99 dataset into 2, 3 and 4 times. Figure 3 illustrates the experimental results.

The speedup of the PBBA algorithm becomes approximately linear when the size of the dataset increases, especially when the dataset contains 15 and 20 million instances. It is observed that the performance of 64-core system on very large dataset is much better than the performance of the same with a smaller dataset which is due the fact that the data processing takes more time than the communication time owing to network

latency among the nodes in the cluster and the fault-tolerance time, which gives a good speedup performance.

The ideal parallelism exhibits a constant speedup with increasing number of cores in the computing cluster. However, in practice, it is very difficult to achieve the linear speedup because of the communication cost among the nodes due to network latency and the skew of the slaves. Though, the dataset is partitioned uniformly among the map tasks in the cluster, the read/write operations varies among the disk drives in the nodes and the slowest slave node decides the total time needed to complete the task.

5.4 Scaleup

Scaleup or the weak scalability defines the ability of the parallel computation algorithm to grow both the number of processing cores in the cluster and the dataset size. It is also defined as the ability of an m-times larger system in the cluster to perform an m-times larger job in the same run-time as the original system and this can be expressed as

$$Scaleup\,(m, D) = \frac{ET\,(1, D)}{ET\,(m, mD)} \tag{11}$$

where m is the number of cores in the computing cluster, ET(1,D) is the execution time of the tasks on one core with data size of D, ET(m, mD) is the execution time of the parallel tasks with m cores in the computing cluster with data size m times of D. An ideal parallelism shows a constant scale up with increasing number of computing cores in the cluster and dataset size [24,32].

To demonstrate how well the PBBA scales up, scalability experiments were performed where the size of the dataset was increased in proportion to the number of cores. The experiments with datasets size of 1.25, 2.5, 5, 10, 20 and 40 million instances were performed on 2, 4, 8, 16, 32 and 64 cores respectively and Fig. 4 shows the performance results on these datasets. It is observed that the scalability of HPBBA decreases slowly when the size of the dataset increases and it maintains a value of scale up higher than 81 %. The HPBBA algorithm is seen to scale very well.

5.5 Sizeup

Sizeup measures the ability of the parallelism to handle the data growth. It calculates how much longer it takes to complete the parallel tasks, when the dataset size is n-times larger than the original dataset. Sizeup analysis is performed by keeping the number of computing core in the cluster as constant and increasing the size of the datasets by the factor 'n'. It can be expressed as follows

$$Sizeup\,(m) = \frac{ET\,(m, nD)}{ET\,(m, D)} \tag{12}$$

where m is the number of computing cores and n is the incremental factor of the data size. T(m,D) is the execution time of the parallel tasks with m computing cores for the

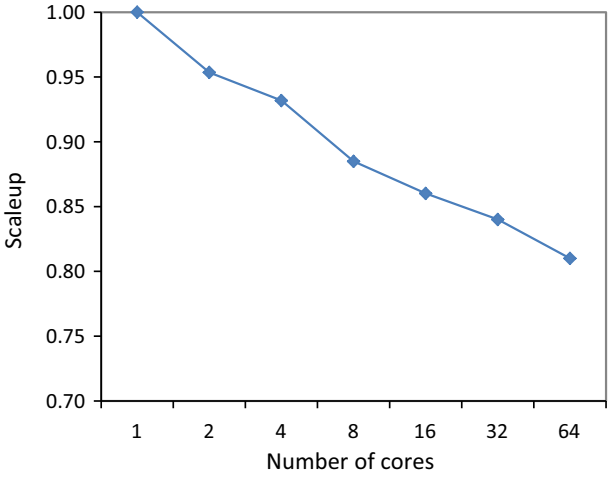

Fig. 4 Sclaeup of HPBBA

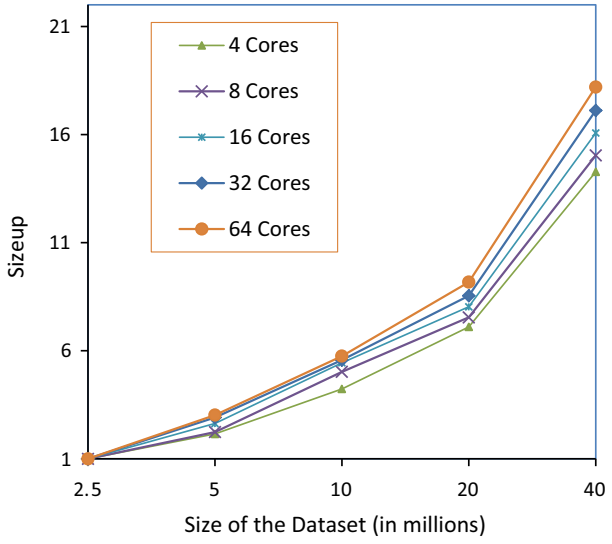

Fig. 5 Sizeup of HPBBA

dataset size D and T(m,nD) is the execution time of parallel tasks with m computing cores and with dataset size n times of D [24,32].

To measure the performance of sizeup, the number of cores were chosen as 4, 8, 16, 32 and 64 with the dataset sizes of 2.5, 5, 10, 20 and 40 million instances. Figure 5 shows the sizeup results on different cores. It is observed that when the number of cores is small such as 4, 8 and 16 the sizeup performances vary modestly. However, the value of sizeup on larger number of cores such as 32 and 64 decreases significantly when compared to smaller size cores of 4, 8 and 16 on the same datasets.

Table 3 Confusion matrix

Class	Predicted negative class	Predicted positive class
Actual negative class	TN	FP
Actual positive class	FN	TP

5.6 Results for Classification Experiments

The selected features in the above experiment are used as the inputs of the classifier. The parallel Naive Bayes classification algorithm is used for evaluating the classification accuracy of the proposed feature selection approach. The trained model of the distributed classifier is constructed by employing the full KDDCup99 training dataset. To test the detection accuracy of the trained classifier model and performance of parallelism, the 10 % KDDCup99 test dataset is duplicated 10 times.

The output from the detection model is known as the confusion matrix. Table 3 lists the confusion matrix and consists of four classes: False Positive (FP), and False Negative (FN), True Positive (TP) and True Negative (TN).

The classification performances of the proposed algorithms are measured using two metrics, namely Detection Rate (DR) and False Positive Rate (FPR). These can be calculated from the confusion matrix, and are defined as,

$$\text{Detection Rate (DR)} = \text{TP}/(\text{TP}+\text{FN}) \tag{13}$$

$$\text{False Positive Rate (FPR)} = \text{FP}/(\text{FP}+\text{TN}) \tag{14}$$

Looking at the results of Table 4, it is observed that the overall performance in detection rates of five classes have been improved while considering the 24 features selected from the feature selection approach PBBA. It can be seen that the PNBC performs better in detecting the Normal, DoS and Probe attacks with a lowest false positive rate of 1.57 %. The PNBC is enhanced in detecting DoS and Probe attacks and the performance is not exceptional in R2L and U2R category of attacks due to the following reason.

- There is an unequal distribution of the attack class categories in full KDDCup99 training dataset that could significantly affect the classifier learning and detecting of R2L and U2R attacks.
- In the complete KDDCup99 dataset, 79.4 and 19.7 % of the instances belong to DoS and Normal categories respectively and the remaining 0.9 % of the training dataset instances belong to Probe, R2L and U2R categories of attacks.
- However, the number of instances for Probe attacks is 41,102, which is sufficient for training the classifier.
- Also, more number of new attacks in the R2L and U2R test dataset which are not present in the training dataset affects its detection rate.

In the detection of attacks, PNBC outperformed the other classifiers in detecting the Probe, R2L and U2R attacks. As shown in Table 5, PNBC achieved the highest probe

Table 4 Detection rates and false positive rate of PNBC

No. of features	(%) of detection rate					% of false positive rate
	Normal	DoS	Probe	R2L	U2R	
41	97.81	96.34	92.93	45.56	72.51	2.18
24	**98.42**	**97.53**	**93.54**	**47.41**	**74.13**	**1.57**

Bold values indicate the highest detection rate

Table 5 Comparison of overall detection rates

Category	KDDCup99 winners [23]	KDDCup99 runners [16]	DSSVM [8]	MLHC [35]	PNBC
Normal	**99.50**	99.4	98.4	96.80	98.42
DoS	97.10	97.5	97.2	**98.66**	97.53
Probe	83.30	84.5	87.5	93.40	**93.54**
R2L	8.40	7.3	6.3	46.97	**47.41**
U2R	13.20	11.8	3.1	71.43	**74.13**

Bold values indicate the highest detection rate

Table 6 Running time of PNBC on different sizes of clusters

No. of cores in the cluster	Training time (s)	Testing time (s)
2	2623	1327
4	1467	792
8	764	431
16	312	176

attacks detection rate of 93.54 % and the R2L attacks detection rate is at 47.41 % and the U2R attacks detection rate is at 74.13 % in classifying the KDDCup99 test dataset.

Speed is always an important metric as the use of intrusion detection system is for real time applications. Table 6 shows the running time recorded for the training and testing of the classifier PNBC on different sizes of the cluster. The number of cores were chosen as 2, 4, 8 and 16. It is observed that the training takes longer time than testing and when the number of cores is increased the training and testing time decreases significantly. The total running time for classifying all the connection records of 100 % KDDCup99 test dataset on the cluster of 16 cores is 176 s.

From the literature survey, it has been found that mostly the classification and feature selection of IDS have been experimented in sequential algorithm with single node and the results in terms of complexity, detection rate and alarm rate have been presented. But, the proposed work was experimented with parallel algorithm in which multiple nodes were used for execution. So, speedup, sizeup and scaleup parameters of the present algorithm are more productive compared to existing sequential algorithm methods. Our work has also been compared with some existing approaches of

sequential algorithm and the detection rate and alarm rate are more in the proposed parallel algorithm approach.

6 Conclusion

This work proposes methods for the fast and scalable IDS for large volume data and aims to improve upon existing works from two perspectives. Firstly, the parallel version of Binary Bat Algorithm has been developed which is employed as a wrapper based feature selection tool and has greater impact on minimizing the memory requirement for IDS and computational complexity of the classifier. Next, a parallel Naive Bayes model has been employed as the classification engine in a multi node cluster which improves the attack detection rate of Probe and U2R attack categories and also reduces the detection time drastically which is proportional to the number of nodes in the cluster.

The main findings of the present work can be summarized as follows:

1. The proposed system is more suitable for Big data analysis.
2. The detection rate of Probe (93.54 %) and R2L (74.13 %) attack categories of the proposed system is better as compared to existing works discussed in the literature.
3. The performance of the parallel execution system in terms of scale up, speedup and size up has been measured in a cluster with different sizes and it is found that the performance is proportional to the number of nodes in the cluster.

Acknowledgements The authors would like to thank all anonymous reviewers for their constructive and insightful suggestions to improve this paper.

References

1. Abadeh, M.S., Habibi, J.: A hybridization of evolutionary fuzzy systems and ant colony optimization for intrusion detection. ISC Int. J. Inf. Secur. **2**(1), 33–46 (2010)
2. Chu, C.T., Kim, S., Lin, Y.A.: MapReduce for machine learning on multicore. In: Proceedings of the 20th Conference on Advances in Neural Information Processing Systems, NIPS, pp. 281–288 (2006)
3. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. Commun. ACM **51**(1), 107–113 (2008)
4. Deng, D.Y., Yan, D.X., Wang, J.Y.: Parallel reducts based on attribute significance. In: Yu, J., Greco, S., Lingras, P., et al. (eds.) Rough Set and Knowledge Technology. Lecture Notes in Computer Science, vol. 6401, pp. 336–343. Springer, Berlin (2010)
5. Depren, O., Topllar, M., Anarim, E., Ciliz, M.K.: An intelligent intrusion detection system (IDS) for anomaly and misuse detection in computer networks. Expert Syst. Appl. **29**, 713–722 (2005)
6. Domingos, P., Pazzani, M.: On the optimality of the simple Bayesian classifier under zero-one loss. Mach. Learn. **29**, 103–130 (1997)
7. Gowrison, G., Ramar, K., Muneeswaran, K., Revathi, T.: Minimal complexity attack classification intrusion detection system. Appl. Soft Comput. **13**, 921–927 (2013)
8. Guo, C., Zhou, Y., Ping, Y., Zhang, Z., Liu, G., Yang, Y.: A distance sum-based hybrid method for intrusion detection. Appl. Intell. **40**, 178–188 (2014). doi:10.1007/s10489-013-0452-6
9. Hadoop MapReduce. http://hadoop.apache.org/ (2015)
10. Han, L.X., Liew, C.C., Hemert, J.V., Atkinson, M.: A generic parallel processing model for facilitating data mining and integration. Parallel Comput. **37**, 157–171 (2011)
11. Harb, H.M., Desuky, A.S.: Adaboost ensemble with genetic algorithm post optimization for intrusion detection. Int. J. Comput. Sci. Issues **8**(5), 28–33 (2011)

12. Horng, S.-J., Ming-Yang, S., Chen, Y.-H., Kao, T.-W., Chen, R.-J., Lai, J.-L., Perkasa, C.D.: A novel intrusion detection system based on hierarchical clustering and support vector machines. Expert Syst. Appl. **38**(1), 306–313 (2011)
13. Hu, W., Hu, W.: Network-based intrusion detection using Adaboost algorithm. In: Proceedings of the 2005 IEEE/WIC/ACM International Conference on Web Intelligence (WI'05) (2005)
14. Jain, A.K., Murty, M.N., Flynn, P.J.: Data clustering: A review. ACM Comput. Surv. **31**(3), 264–323 (1999)
15. Kennedy, J., Eberhart, R.C.: A discrete binary version of the particle swarm algorithm. In: IEEE International Conference on Computational Cybernetics and Simulation, pp 4104–4108 (1997)
16. Levin, I.: KDD-99 classifier learning contest LLSoft's results overview. SIGKDD Explore. ACM SIGKDD (2000)
17. Mahmud, W.M., Agiza, H.N., Radwan, E.: Intrusion detection using rough sets based parallel genetic algorithm hybrid model. In: Proceedings of the World Congress on Engineering and Computer Science (WCECS-2009), USA
18. McNabb, A.W., Monson, C.K., Seppi, K.D.: Parallel PSO Using MapReduce. In: Proceedings of 2007 IEEE Congress on Evolutionary Computation, CEC, IEEE Computer Society, pp. 7–16 (2007)
19. Mirjalili, S., Mohd Hashim, S.Z.: BMOA: binary magnetic optimization algorithm. In: 2011 3rd International Conference on Machine Learning and Computing (ICMLC 2011), Singapore, 2011, pp. 201–206 (2011)
20. Mohammad, M.R., Dominik, S., Wróblewski, J.: Parallel island model for attribute reduction. In: Pal, S.K., et al. (eds.) PReMI 2005. LNCS 3776, pp. 714–719, Springer (2005)
21. Natesan, P., Balasubramanie, P., Gowrison, G.: Improving attack detection rate in network intrusion detection using adaboost algorithm with multiple weak classifiers. J. Inf. Comput. Sci. **8**(8), 2239–2251 (2012)
22. Peddabachigari, S., Abraham, A., Grosan, C., Thomas, J.: Modelling intrusion detection system using hybrid systems. J. Netw. Comput. Appl. **30**, 114–132 (2007)
23. Pfahringer, B.: Winning the KDD99 classification cup: bagged boosting. SIGKDD Explor. **1**(2), 67–75 (2000)
24. Qian, J., Miao, D., Zhang, Z., Yue, X.: Parallel attribute reduction algorithms using MapReduce. J. Inf. Sci. **279**, 671–690 (2014)
25. Rashedi, E., Nezamabadi-pour, H., Saryazdi, S.: BGSA: binary gravitational search algorithm. Nat. Comput. **9**, 727–745 (2009)
26. Srinivasan, A., Faruquie, T.A., Sachindra, J.: Data and task parallelism in ILP using MapReduce. Mach. Learn. **86**(1), 141–168 (2012)
27. Sung, A.H., Mukkamala, S.: The feature selection and intrusion detection problems. In Proceedings of advances in computer science—ASIAN 2004: higher-level decision making. In: 9th Asian Computing Science Conference, vol. **3321**, pp. 468-482 (2004)
28. Tsang, C.H., Kwong, S.: Multi-agent intrusion detection system in industrial network using ant colony clustering approach and unsupervised feature extraction. In: Proceedings of the IEEE International Conference on Industrial Technology **2005**(ICIT2005), pp. 51–56 (2005)
29. Venkatachalam, V., Selvan, S.: Performance comparison of intrusion detection system classifiers using various feature reduction techniques. Int. J. Simul. **9**(1), 30–39 (2008)
30. Verma, A., Llora, X., Goldberg, D.E., Campbell, R.H.: Scaling genetic algorithms using MapReduce. In: Proceedings of the 2009 Ninth International Conference on Intelligent Systems Design and Applications, IEEE Computer Society, pp. 13–18 (2009)
31. Wang, G., Hao, J., Ma, J., Huang, L.: A new approach to intrusion detection using artificial neural networks and fuzzy clustering. Expert Syst. Appl. **37**(9), 6225–6232 (2010)
32. Weaver, J.: A scalability metric for parallel computations on large, growing datasets (like the web). In: Proceedings of the Joint Workshop on Scalable and High-Performance Semantic Web Systems (2012)
33. Weiming, H., Wei, H., Maybank, S.: AdaBoost-based algorithm for network intrusion detection. IEEE Trans. Syst. Man Cybern. Part B Cybern. **38**(2), 577–583 (2008)
34. Xiang, C., Chong, M.Y., Zhu, H.L.: Design of multiple-level tree classifiers for intrusion detection system. In: Proceedings of the 2004 IEEE Conference on Cybernetics and Intelligent Systems, December, Singapore, pp. 872–877 (2004)
35. Xiang, C., Yong, P.C., Meng, L.S.: Design of multiple-level hybrid classifier for intrusion detection system using Bayesian clustering and decision trees. Pattern Recognit. Lett. **29**, 918–924 (2008)

36. Yang, X.S.: A new metaheuristic bat-inspired algorithm. In: Gonzalez, J.R., et al. (eds.) Nature Inspired Cooperative Strategies for Optimization (NICSO 2010), vol. 284, pp. 65–74. Springer, Berlin (2010)
37. Zhao, W.Z., Ma, H.F., He, Q.: Parallel K-means clustering based on MapReduce. In: Jaatun, M.G., Zhao, G., Rong, C. (eds.) Cloud Computing, CloudCom2009, pp. 674–679. Springer, Berlin (2009)