Received July 29, 2019, accepted August 11, 2019, date of publication August 15, 2019, date of current version August 29, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2935691

A Resource Aware Parallelized Back Propagation Neural Network in Enabling Efficient Large-Scale Digital Health Data Processing

YANG LIU[1], **XIANBANG CHEN**[1], **LIXIONG XU**[1], **HUAQIANG LI**[1], **AND MAOZHEN LI**[2]
[1]College of Electrical Engineering, Sichuan University, Chengdu 610065, China
[2]Department of Electronic and Computer Engineering, Brunel University London, Uxbridge UB8 3PH, U.K.

Corresponding author: Huaqiang Li (lihuaqiang@scu.edu.cn)

ABSTRACT Along with the development of digital health, efficient machine learning is anxiously needed to handle the growing health data. Among various machine learning algorithms, back propagation neural network (BPNN) shows great effectiveness in both academia and industrial fields. However, it is frequently reported that the conventional BPNN algorithm encounters low efficiency issue in dealing with large-scale digital health data. Therefore this paper presents a Hadoop based parallelized BPNN algorithm which is able to process the large-scale data efficiently. In order to complement the potential accuracy loss issue for the parallelized data processing, ensemble learning techniques are also involved. Additionally although Hadoop supplies a number of default schedulers, the heterogeneous distributed computing environment may still impact the efficiency of the parallelized BPNN. Consequently, this paper also presents a gene expression programming (GEP) algorithm based load balancing approach, which enables the computing resource awareness and the optimal scheduling of the parallelized BPNN. The experiments employ the classification task as the underlying testing basis. Two types of the experiments are carried out, in which the first one focuses on evaluating the accuracy of the presented algorithm with classifying the benchmark dataset; the second one focuses on evaluating the efficiency of the presented algorithm with classifying the large-scale dataset. The experimental results show the effectiveness of the presented resource aware parallelized BPNN algorithm.

INDEX TERMS Back propagation neural network, parallelization, Hadoop, load balancing, gene expression programming.

I. INTRODUCTION

At present, machine learning technologies have been significantly applied in the digital health researches to achieve the classification, optimization, function approximation, pattern recognition, and so on. Among a number of machine learning algorithms, artificial neural network (ANN) has been proved to be one of the most effective algorithms to solve the practical problems [1]–[6], [30]–[33]. In various types of ANN algorithms, back propagation neural network (BPNN) has been widely studied due to its remarkable function approximation abilities. For example, Almaadeed *et al.* [1] employed multimodal neural networks including BPNN for voice identification. Fan *et al.* [2] also successfully employed BPNN to implement the respiratory monitoring which is an important tool for clinical monitoring.

The associate editor coordinating the review of this article and approving it for publication was Yongqiang Cheng.

However, in the recent years due to the development of the data collection and the storage technologies, the volume of data has been increasingly enlarged. In current big data era, researchers adopted BPNN to carry out the large-scale data analysis [3], [4]. However, the authors pointed out that BPNN encounters low efficiency issue for handling the large-scale data. Therefore, quite a number of efforts have been done to solve the issue. Li *et al.* [5] presented a BPNN algorithm with a self-adaptive learning speed in the training phase. Based on the experimental results, their algorithm performs efficiently compared to the constant learning speed based BPNN. Liu *et al.* [6] also designed a hierarchical neural network for processing the large-scale geometry information. Their work indicates that the neural network has great potential and effectiveness to process the large-scale data. However, the processing efficiency should be carefully considered. In terms of large-scale data processing using the neural network, Gu *et al.* [7] pointed out that the most time-consuming phase is the training phase. Therefore the authors presented cNeural which is an in-memory computing based distributed neural network aiming at improving the training efficiency. Although the experimental results indicate that the algorithm is effective, their simple in-memory distributed computing platform has been significantly limited by the participated volume of memory, the lack of the fault tolerance, and the lack of the load balancing. As a result, the well-developed distributed computing infrastructures or frameworks with the advanced distributed computing oriented characteristics become effective tools in enabling the parallelization of the neural network.

Huqqani *et al.* [8] established an OpenMP (Open Multi-Processing) based distributed computing environment for parallelizing the neural network. Their parallelization is based on the separation of the structural and topological data. However, the simple data separation may result in insufficient training of the network, which further leads to the accuracy loss issue. And also although OpenMP can improve the utilization of the multi-core system, it is still limited by the participated memory to process the extreme large-scale tasks. Research [9] presented a parallel artificial neural network using MPI (Message Passing Interface). However, the impact of the heterogeneity of the cluster has not been discussed. As a result, Hadoop [10] which is a MapReduce computing model [11] based distributing computing framework has been widely employed by quite a number of researchers. Liu *et al.* [12] presented a MapReduce based distributed BPNN for processing large-scale mobile data. They also separated the training dataset into a number of data chunks and further employed adaboosting to complement the algorithm accuracy loss caused by the data separation. However, the simple sampling technique employed by adaboosting may enlarge the weight of wrongly classified data, which would deteriorate the algorithm accuracy. Liu *et al.* [13], [29] employed bagging technology instead of adaboosting to overcome the accuracy loss issue caused by the data separation in the parallelization. Based on the experimental results, their Hadoop based parallelized BPNN can maintain accuracy with satisfied efficiency for classifying large-scale data.

The aforementioned researches indicate that the parallelized BPNN benefits from the advantages of the Hadoop framework. However, several researches pointed out that the load imbalance issue of the distributed training significantly impacts the data processing efficiency in a heterogeneous Hadoop cluster [17], [19], [29]. Although Hadoop has a number of default schedulers such as FIFO, fair scheduler, and capacity scheduler, these universal schedulers cannot adapt to various kinds of Hadoop jobs [14]–[16]. Additionally scheduler designed for a specific type of Hadoop job [14]–[16] may not serve the other types of the jobs well [17]. Research [17] also presented a load balancing algorithm which can serve multiple types of Hadoop jobs. However, the authors admitted that their algorithm only considers the load balancing among mappers so that if the imbalance occurs in reducers the performance of the algorithm may deteriorate. Therefore, a proper scheduler specially serving the parallelized BPNN training in the heterogeneous Hadoop cluster is quite necessary.

In order to process the large-scale data using BPNN in a heterogeneous Hadoop cluster efficiently and accurately, this paper presents a resource aware parallelized BPNN algorithm. Based on the data separation, the standalone BPNN can be parallelized into a number of parallel sub-BPNNs, each of which inputs one separated data chunk to process. In order to handle the insufficient training and the accuracy loss issues due to the data separation, ensemble techniques including bootstrapping and majority voting are employed. By aggregating the outputs of the sub-BPNNs, the accuracy of the presented algorithm can be improved. Additionally, this paper further employs the gene expression programming algorithm to reveal the relations between the processing capacity of the cluster and the Hadoop parameters. And then the genetic algorithm is adopted to optimize the parameters according to the divisible load theory. Based on the optimized parameters, the load balancing can be achieved and therefore the efficiency of the presented parallelized BPNN running in the Hadoop cluster is able to be improved.

The rest of the paper is organized as: section II briefly presents the principles of BPNN and Hadoop; section III gives the details of parallelizing BPNN in Hadoop; section IV introduces the scheduler design focusing on improving the efficiency of the distributed training; section V shows and discusses the experimental results; section VI concludes the paper.

II. BRIEF INTRODUCTION OF BPNN AND HADOOP FRAMEWORK

A. BACK PROPAGATION NEURAL NETWORK (BPNN)

Fig. 1 shows a typical three-layer BPNN with one input layer, one hidden layer, and one output layer as an example. The input layer contains a number of n inputs; the hidden layer contains a number of p neurons; the output layer contains a number of l outputs.

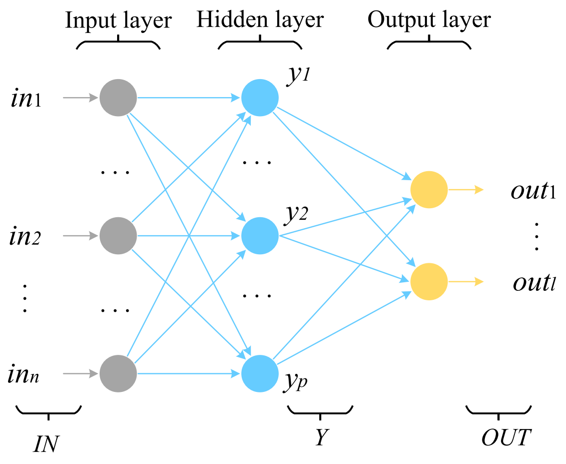


FIGURE 1. The structure of a typical BPNN.

Let $\boldsymbol{IN} = (in_1, in_2, \ldots, in_n)^{\mathrm{T}}$ denote one input data instance; $\boldsymbol{Y} = (y_1, y_2, \ldots, y_p)^{\mathrm{T}}$ denote the output of the hidden layer; $\boldsymbol{OUT} = (out_1, out_2, \ldots, out_l)^{\mathrm{T}}$ denote the output of the output layer; $D = (d_1, d_2, \ldots, d_l)^{\mathrm{T}}$ denote the actual value; w_{ij} and b_{ij} $(i = 1, 2, \ldots, n; j = 1, 2, \ldots, p)$ denote the weights and biases between the input layer and the hidden layer; w_{jk} and $b_{jk}(j = 1, 2, \ldots, p; k = 1, 2, \ldots, l)$ denote the weights and biases between the hidden layer and the output layer. *Sigmoid* function [34] indicated by (1) is selected as the activation function.

$$f(x) = 1/(1 + e^{-x}) \tag{1}$$

Therefore, in the feed forward phase:

$$y_j = f[\sum_{i=1}^{n}(w_{ij}in_i + b_{ij})], \quad j = 1, 2, \ldots, p \tag{2}$$

$$out_k = f[\sum_{j=1}^{p}(w_{jk}y_j + b_{jk})], \quad k = 1, 2, \ldots, l \tag{3}$$

The feed forward phase completes.

In the back propagation phase, let E denote the discrepancy between $\boldsymbol{OUT}$ and $\boldsymbol{D}$. Therefore E can be represented by (4).

$$E = \frac{1}{2}(\boldsymbol{D} - \boldsymbol{OUT})^2 = \frac{1}{2}\sum_{k=1}^{l}(d_k - out_k)^2 \tag{4}$$

Further, E can be derived from the back layer to the front layer. Therefore, for the output layer and the hidden layer, E can be represented by (5) and (6) respectively.

$$E = \frac{1}{2}\sum_{k=1}^{l}\{d_k - f[\sum_{j=1}^{p}(w_{jk}y_j + b_{jk})]\}^2 \tag{5}$$

$$E = \frac{1}{2}\sum_{k=1}^{l}\{d_k - f[\sum_{j=1}^{p}(w_{jk}f[\sum_{i=1}^{n}(w_{ij}in_i + b_{ij})] + b_{jk})]\}^2 \tag{6}$$

According to (4) to (6), the back propagation phase is able to tune the weights w and biases b of the output layer and the hidden layer based on the gradient descent. Let α denote the learning speed, $\alpha \in (0, 1)$. Equations (7) and (8) indicate the details of the tuning.

$$\Delta w = -\alpha\frac{\partial E}{\partial w}, \quad w = w + \Delta w \tag{7}$$

$$\Delta b = -\alpha\frac{\partial E}{\partial b}, \quad b = b + \Delta b \tag{8}$$

The back propagation phase completes.

After a number of epochs, the training terminates. The trained network can be employed for the testing using the testing data instances by executing only feed forward.

B. HADOOP FRAMEWORK

Hadoop framework is a famous implementation of MapReduce computing model which provides the parallel computing mainly based on the Map and Reduce functions. As an important component of Hadoop framework, Hadoop Distributed File System (HDFS) supplies remarkable data storage and IO abilities which significantly facilitate the big data processing. In HDFS, data instances are stored in data chunks in the form of key-value pairs for example {k1, v1}. Hadoop framework also provides another two important components including mapper and reducer which are the implementations of the Map and Reduce functions of MapReduce computing model. Mainly the mappers serve the parallel data processing and the reducers collect the intermediate outputs from the mappers and generate the final output. When a Hadoop cluster starts a job, a number of mappers will be started in parallel, each of which reads one data chunk containing several data instances from HDFS and starts processing according to k1. The mappers keep generating the intermediate outputs in the form of {k2, v2}. The reducers collect and further process the intermediate outputs according to k2. At last, the reducers generate the ultimate outputs which are finally saved into HDFS again. The computing resources of Hadoop are managed by YARN [10] which supplies a number of schedulers for example capacity scheduler, fair scheduler, and FIFO scheduler. The main architecture of the Hadoop framework is shown in Fig. 2.

III. THE PARALLELIZATION OF BPNN

The parallelization of BPNN is based on the data separation. However, the researches [13], [18], [29] pointed out that the simple separation of the training data leads to accuracy loss in the testing phase. Therefore this paper employs bootstrapping and majority voting to improve the accuracy for the parallelized BPNN.

Bootstrapping is a sampling algorithm which has ability to simulate the original sample distribution in the sampled data [27]. The basic concept of bootstrapping is to control the appearance time of the training instances in the sampled samples. In order to implement bootstrapping, replacement method can be employed. Based on certain bootstrapping number, each training instance appears exactly the same time. The original sample distribution can be maintained in the separated sample data chunks using bootstrapping. Majority voting [28] is proved to be an effective way of combining a number of weak classifiers into a strong classifier. Based on the voted result from each participated weak classifiers, majority voting has a higher chance to achieve the correct result.

In the training phase, our work firstly adopts bootstrapping to generate a number of m bootstrapped data chunks based on the original training dataset, where m is the number of mappers employed to run the algorithm. Let T denote the original training dataset; T_b $b = 1, \ldots, m$ denote a bootstrapped data chunk. Therefore T and T_b satisfy $\bigcup_{b=1}^{m} T_b = T$. In each data chunk, a specially designed format for each training instance is:

$$\{instance_i, target_i, instancetype\}$$

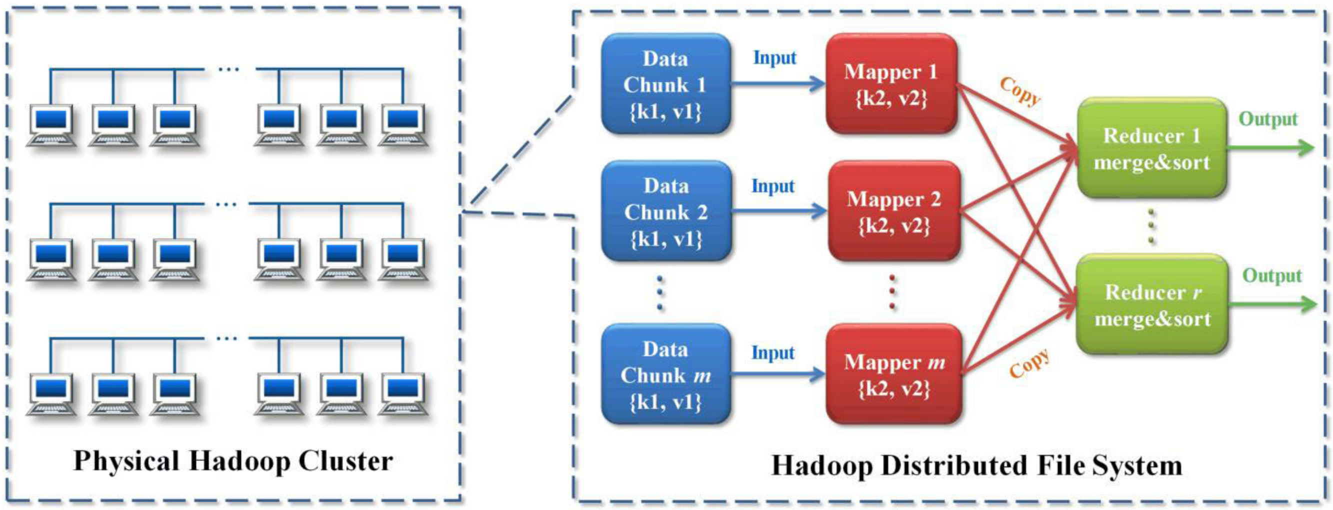


FIGURE 2. The architecture of the Hadoop framework.

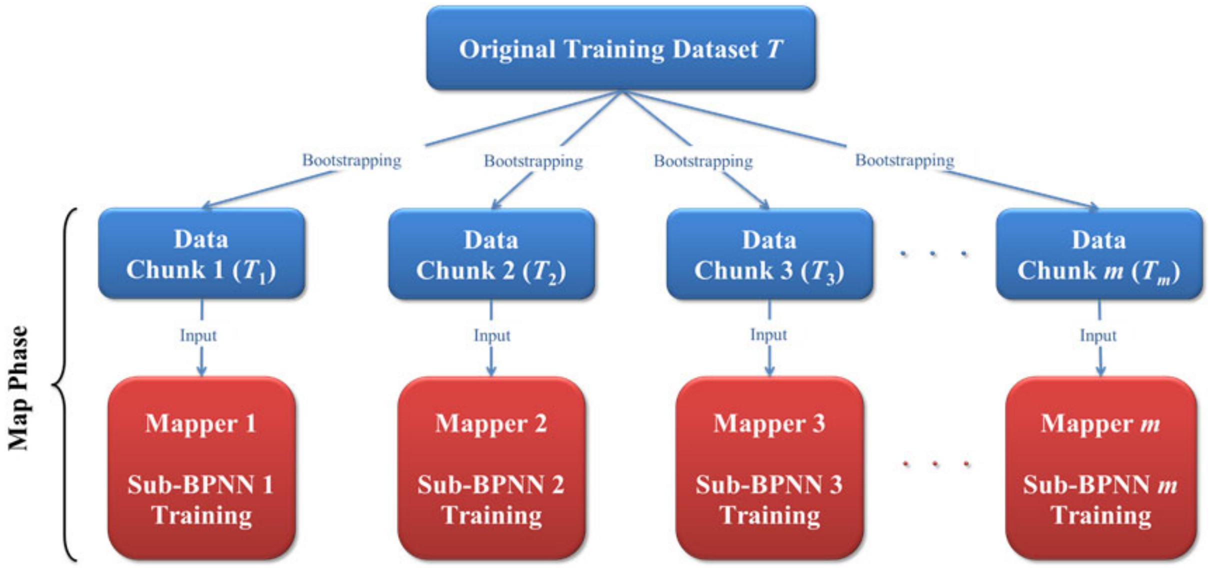

Original Training Dataset T
Bootstrapping
Data Chunk 1 (T1)
Data Chunk 2 (T2)
Data Chunk 3 (T3)
Data Chunk m (Tm)
Input
Map Phase
Mapper 1 Sub-BPNN 1 Training
Mapper 2 Sub-BPNN 2 Training
Mapper 3 Sub-BPNN 3 Training
Mapper m Sub-BPNN m Training

FIGURE 3. The training phase of the parallelized BPNN.

where $instance_i$ represents the normalized i^{th} instance in a data chunk; $target_i$ represents the encoded output of $instance_i$; The *instancetype* field is filled a string "training" to explicitly indicate that $instance_i$ is a training instance.

Secondly, each mapper initializes one BPNN (sub-BPNN) inside itself with the random initial parameters. And then each mapper inputs one data chunk from HDFS respectively. As a result, the training instances saved in the data chunk can be finally input into the mapper moreover into the sub-BPNN one by one. If the *instancetype* is "training", the sub-BPNN in the mapper executes the training operation. The $instance_i$ is processed by the feed forward using (2) to (3) whilst the encoded $target_i$ is employed to tune the network parameters by the back propagation using (4) to (8). After a number of epochs, the training of the sub-BPNN terminates. As a result, for the number of m mappers, a number of m trained classifiers are created in the Hadoop cluster. The training phase of the parallelized BPNN is shown in Fig. 3.

In the testing phase, let T_e denote the testing dataset; $instance_t$ denote one normalized testing instance in T_e, $instance \in T_e$. As long as the testing phase starts, each testing $instance_t$ is input into all the number of m previously trained mappers. In each mapper, $instance_t$ is identified by the sub-BPNN using only feed forward according to (2) to (3). And then the mapper outputs an intermediate output in the {key-value} form:

$$\{instance_t, output_z\}$$

where $output_z$ represents the result for $instance_t$ processed by the mapper so that a number of m mappers finally output

$$\left\{ instance_t \;,\; \begin{matrix} output_1 \text{ of Mapper } 1 \\ output_2 \text{ of Mapper } 2 \\ output_3 \text{ of Mapper } 3 \\ output_3 \text{ of Mapper } 4 \\ output_4 \text{ of Mapper } 5 \\ \vdots \\ output_z \text{ of Mapper } m \end{matrix} \right\}$$

FIGURE 4. **The collection formed by the reducer.**

a number of m intermediate outputs for the testing instance $instance_t$. Afterwards the parallelized BPNN algorithm starts one reducer to collect all the number of m intermediate outputs. Finally, a collection of the key $instance_t$ with a number of m results is aggregated. Fig. 4 shows an example of the collection.

Inside the collection, majority voting votes the final result *result* for $instance_t$. The result with the maximum number of occurrence in the collection is voted as the final result. At last the final *result* for $instance_t$ is saved into HDFS in the form of:

$$\{instance_t, result\}$$

The testing phase keeps running until all the testing instances in T_e are identified. And then the parallelized algorithm terminates. Fig. 5 shows an example that the testing phase of the parallelized BPNN processes a testing $instance_t$. The algorithm finally decides that $output_1$ is the final result.

IV. SCHEDULER OPTIMIZATION OF THE HADOOP FRAMEWORK

A. HADOOP PARAMETERS IMPACTING DATANODE PROCESSING TIME

Although Hadoop provides a number of universal schedulers to balance the load for a heterogeneous cluster, a number of researches for example [17], [20] pointed out that the default schedulers may not work well for arbitrary types of jobs. Therefore a number of specially designed load balancing algorithms which are frequently based on the data locality or the modeling of the Hadoop job processing are presented. However, it has been admitted that the scheduler optimization based on the data locality cannot fully utilize the cluster resources [19]. Moreover, the scheduler optimizations using the modeling of Hadoop [17], [20] claimed that the modeling is extremely complicated because of the complex workflows and intertwined parameters of the Hadoop framework. As a

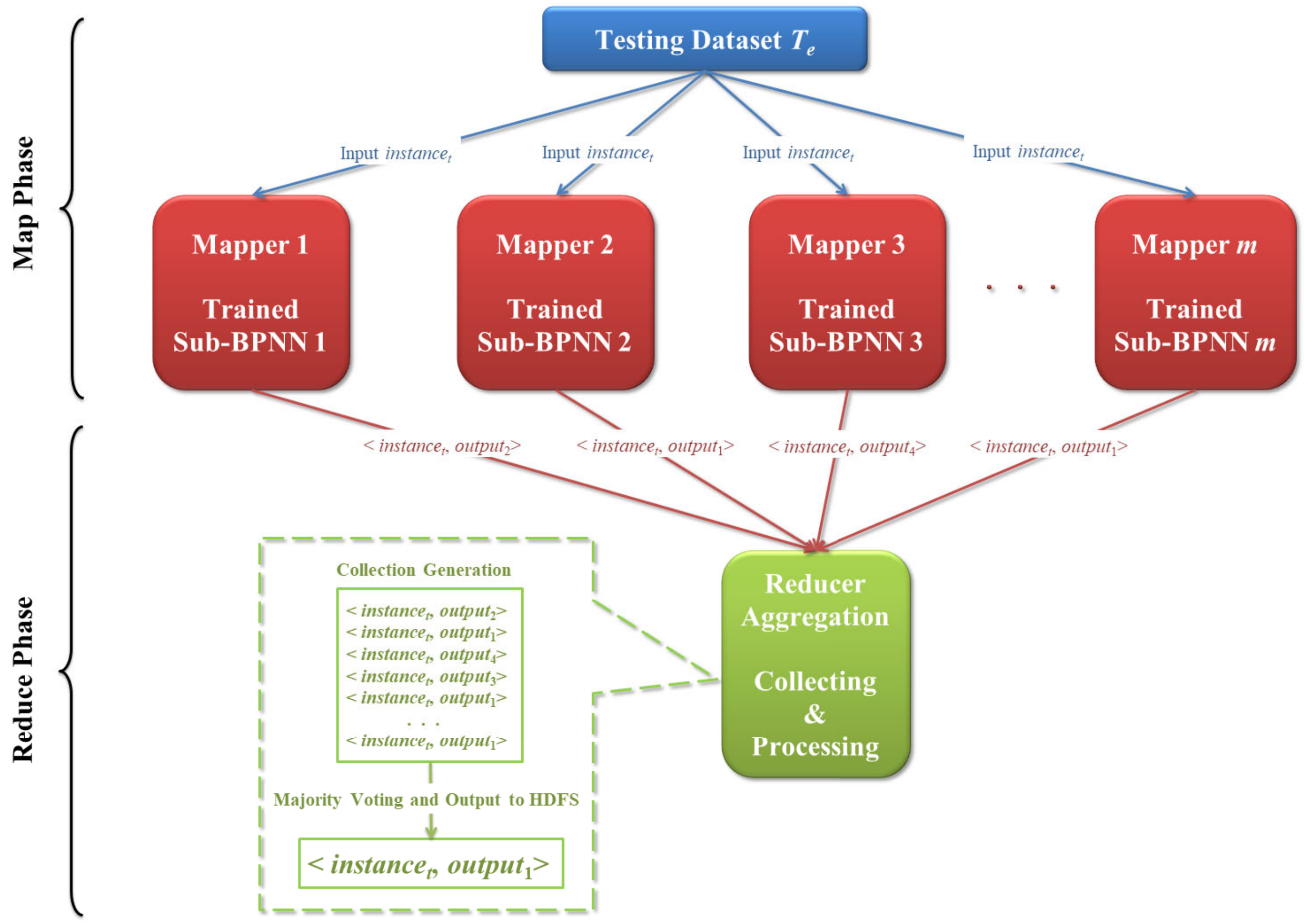

Map Phase
Testing Dataset T_e
Input $instance_t$
Input $instance_t$
Input $instance_t$
Input $instance_t$
Mapper 1 Trained Sub-BPNN 1
Mapper 2 Trained Sub-BPNN 2
Mapper 3 Trained Sub-BPNN 3
Mapper m Trained Sub-BPNN m
< $instance_t$, $output_2$>
< $instance_t$, $output_1$>
< $instance_t$, $output_4$>
< $instance_t$, $output_1$>
Reduce Phase
Collection Generation
< $instance_t$, $output_2$>
< $instance_t$, $output_1$>
< $instance_t$, $output_4$>
< $instance_t$, $output_3$>
< $instance_t$, $output_1$>
< $instance_t$, $output_1$>
Majority Voting and Output to HDFS
< $instance_t$, $output_1$>
Reducer Aggregation
Collecting & Processing

FIGURE 5. **The testing phase of the parallelized BPNN.**

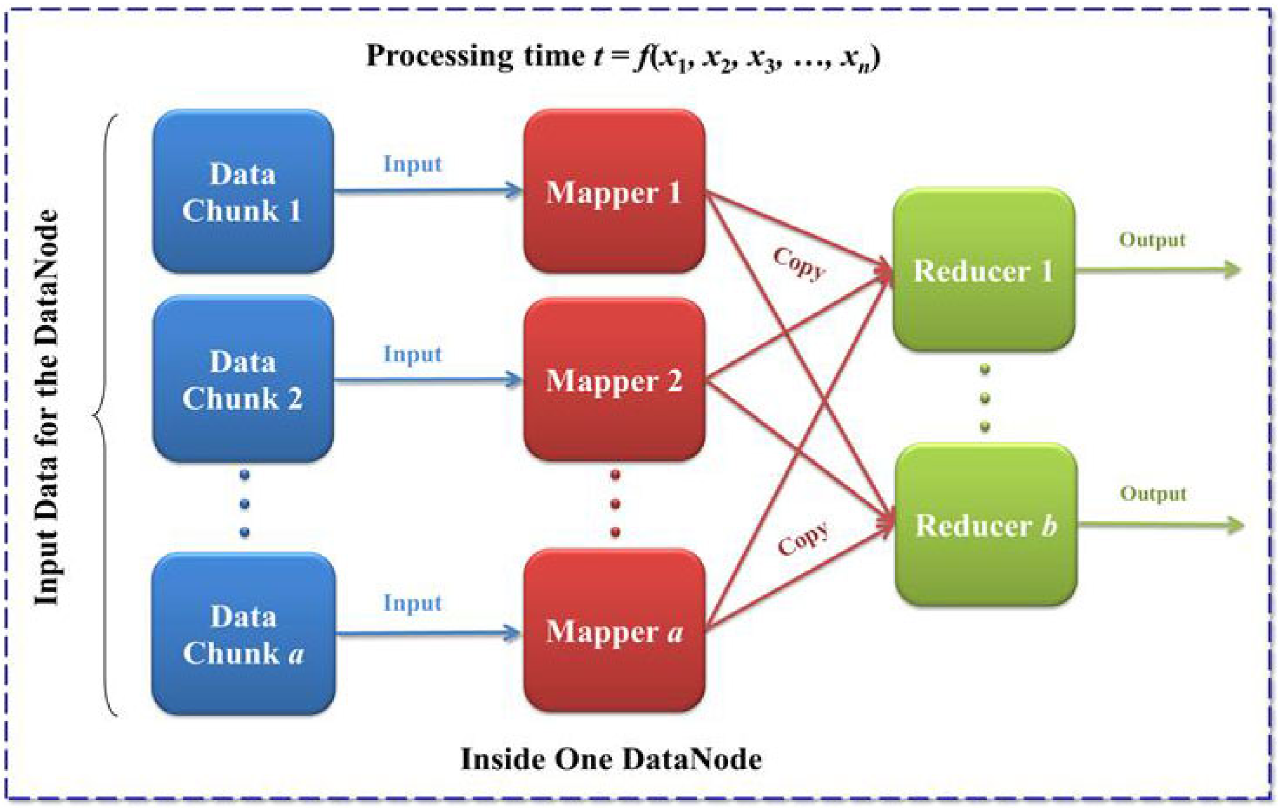


FIGURE 6. **Identification of the processing time *t* of a DataNode.**

result the authors can only concentrate the modeling of the mapper, whilst the modeling of the reducer is neglected. Research [21] presented a GEP based Hadoop parameter optimization in enabling the improvement of the cluster efficiency. However, the authors mined the correlations of the processing time and the Hadoop parameters based on the entire cluster, which may lose accuracy of measuring the processing capacity of individual DataNodes in the cluster.

The aforementioned difficulties motivate us that if a pattern (processing capacity) of a working DataNode can be recognized, the complicated mathematical modeling of the node could be avoided. Therefore, the awareness of the processing capacity of one DataNode can be great helpful of balancing load among multiple DataNodes. This paper evaluates the processing capacity mainly represented by the processing time of each individual DataNode in the cluster, which eases the identification of the correlations between the nodal processing time and the Hadoop parameters.

Fig. 6 shows a working DataNode contributing a number of a mappers and a number of b reducers to the cluster. The processing time t of the DataNode can be determined by the dependent variables $\boldsymbol{x}$ which represent a number of n parameters (including input data size and Hadoop framework parameters controlling the data processing) and a mathematical equation f. Therefore, based on the history experimental t and $\boldsymbol{x}$, a proper equation f could be mined.

However, it should be realized that there are a great number of the parameters which may affect the processing time of the Hadoop framework. The great number of the parameters may significantly impact the function mining and the solution of the optimized scheduler. Therefore in order to mine the equation f and facilitate the solution of the optimized scheduler, firstly a proper number of Hadoop parameters which impact the processing time mostly [21] of one DataNode are listed in Table 1. The values of the parameters employed in our experiments are listed in Table 2. We also abbreviate the names of the parameters in Table 2.

Based on the parameters and values shown in Table 1 and Table 2, a number of experiments were carried out. The processing time t of each DataNode and its corresponding values of the parameters are recorded. Afterwards this paper employs GEP algorithm [22] to mine the equation f using the recorded history processing time t and the values of the parameters.

B. MINING THE EQUATION REPRESENTING THE TRAINING TIME OF A DATANODE USING GEP

GEP is a function mining algorithm which is able to mine a mathematical equation f representing the correlation $\boldsymbol{y} = f(\boldsymbol{x})$ based on the values of the dependent variables $\boldsymbol{y}$ and the independent variables $\boldsymbol{x}$. Similar to the genetic algorithm, GEP simulates the biological evolution to mine an equation

TABLE 1. The parameters impacting the processing time of a DataNode.

Name of the parameter	Default value	Data Type	Brief description
input data size of one node	size of the input data	Float	total size of the data chunks input into a DataNode
mapreduce.task.io.sort.factor	10	Integer	the number of streams that can be merged while sorting
mapreduce.task.io.sort.mb	100	Float	the size of the buffer for sorting
mapreduce.map.sort. spill.percent	0.8	Float	the threshold in percentage for spilling the data saved in buffer to hard disk
mapreduce.job.reduces	1	Integer	the number of the reducers employed by a submitted job
mapreduce.tasktracker. map.tasks.maximum	2	Integer	the number of the map slots (mappers) configured in each DataNode
mapreduce.tasktracker. reduce.tasks.maximum	2	Integer	the number of the reduce slots (reducers) configured in each DataNode
mapred.child.java.opts	200MB	Float	the maximum size of the memory for Java virtual machine of each task
mapreduce.reduce.shuffle. merge.percent	0.66	Float	the threshold in percentage for starting merging the data in the buffer
mapreduce.reduce.shuffle. input.buffer.percent	0.7	Float	the amount of memory in percentage assigned to a reducer to store map intermediate output during the shuffle process
mapreduce.reduce.shuffle. parallelcopies	5	Integer	the number of the copy threads of a reducer

TABLE 2. The values of the parameters employed in the experiments.

Name of the parameter	Abbreviation	Range of value
input data size of one node	X_0	{60.28MB, 120.56MB, 180.84MB, 241.12MB, 361.68MB}
mapreduce.task.io.sort.factor	X_1	[2, 20]
mapreduce.task.io.sort.mb	X_2	[10MB, 200MB]
mapreduce.map.sort. spill.percent	X_3	[0.1, 0.9]
mapreduce.job.reduces	X_4	[1, 4]
mapreduce.tasktracker. map.tasks.maximum	X_5	4
mapreduce.tasktracker. reduce.tasks.maximum	X_6	4
mapred.child.java.opts	X_7	[400MB, 500MB]
mapreduce.reduce.shuffle. merge.percent	X_8	[0.1, 0.9]
mapreduce.reduce.shuffle. input.buffer.percent	X_9	[0.1, 0.9]
mapreduce.reduce.shuffle. parallelcopies	X_{10}	[2, 10]

that has the best fitness for representing the correlation between $\boldsymbol{y}$ and $\boldsymbol{x}$. However, a novel component named as function set including a number of mathematical operators has been employed by GEP so that the algorithm is able to evolve and output the mined equation f explicitly. In the evolution, GEP generates offspring mainly using the operations including selection, crossover, and mutation. Each individual of the offspring is assessed by a fitness function so that the best fitted individuals have higher chances to be selected to produce next generation. The evolution keeps evolving until a satisfied equation is discovered.

Let $\boldsymbol{t}$ represent the recorded history processing times of one DataNode; t_c represent one record in $\boldsymbol{t}$; $\boldsymbol{parameters}$ represent the recorded history values of the Hadoop parameters; $parameters_c$ represent one record in $\boldsymbol{parameters}$, which is related to t_c. Table 3 shows the steps of mining the equation f which describes $\boldsymbol{t} = f(\boldsymbol{parameters})$ for one DataNode using GEP algorithm.

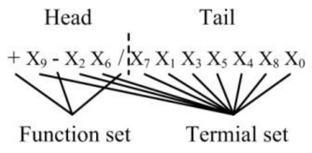
As long as the equation f representing $\boldsymbol{t} = f(\boldsymbol{parameters})$ for an individual DataNode is mined, it can be further employed to execute the load balancing for the heterogeneous Hadoop cluster.

TABLE 3. The function mining based on history experimental data.

Input dataset: $\boldsymbol{t}$, $\boldsymbol{parameters}$, $t_c \in \boldsymbol{t}$, $parameters_c \in \boldsymbol{parameters}$

Output: f for representing $\boldsymbol{t} = f(\boldsymbol{parameters})$

1. Initiate GEP components and parameters including function set, link function, selection, mutation, and crossover. A chromosome is composed by 4 genes using the link function, each of which is consisted by function set and terminal set, for example:



2. Set fitness function max(num_r/num_t), where num_t represents the total sample number; num_r represents the correctly fitted samples using the currently mined f; a correctly fitted sample satisfies $|value\text{-}target|<0.15$; *target* represents the input t_c; *value* represents the calculated time using the current mined equation f and the input $parameters_c$.

3. While fitness is NOT satisfied OR iteration number is NOT reached

   GEP keeps evolving to mine the equation f.

4. Output the finally mined equation f.

5. Algorithm terminates.

C. LOAD BALANCING FOR A HETEROGENEOUS HADOOP CLUSTER

Let N denote the number of DataNodes participated to process a job; *TotalDataSize* denote the total training data size of the job; $InputDataSize_j$ ($X_{0\,j}$) represent the training data size input into the j^{th} DataNode, $j = 1, 2, \ldots, N$; t_j represent the estimated processing time for $InputDataSize_j$ of the j^{th} DataNode using the mined equation f_j; $parameters_{cluster}$ (X_1 to X_{10}) represent the optimized parameters to be configured in the physical Hadoop cluster. Therefore, according to the divisible load theory [23]–[25], the efficiency of the job processing can be improved if the number of N DataNodes finish the processing at the same time:

$$t_1 = t_2 = t_3 \cdots = t_N \tag{9}$$

Moreover according to the mined f, (9) can be represented by (10):

$$\begin{aligned} &f_1(InputDataSize_1, parameters_{cluster}) \\ &\quad = f_2(InputDataSize_2, parameters_{cluster}) \\ &\quad = f_3(InputDataSize_3, parameters_{cluster}) \\ &\quad = \cdots = f_N(InputDataSize_N, parameters_{cluster}) \end{aligned} \tag{10}$$

As a result, the optimized parameters for the Hadoop cluster and the input training data size for each DataNode can be achieved by solving (10). As long as the equation is solved, the training data size assigned to the j^{th} DataNode is determined by $InputDataSize_j$, whilst $parameters_{cluster}$ are written into the configuration file of the physical Hadoop cluster.

However, (10) is difficult to be solved. Therefore this paper adopts genetic algorithm (GA) to solve the equation. In order to implement GA, the detailed settings are listed below.

- Fitness function: To measure the similarity of the processing times for the participated DataNodes, Mean Square Error (MSE) is employed as the fitness function *Fitness* which can be represented by (11).

$$Fitness = \sqrt{\sum_{j=1}^{N} (\bar{t} - t_j)}, \quad \bar{t} = \frac{\sum_{j=1}^{N} t_j}{N} \tag{11}$$

- Crossover, mutation, population, and selection: Crossover is the single point crossover. The mutation is the single point mutation based on a probability $p_m = 0.1$. The population is 100. In each generation the best fitted chromosome individuals are selected to generate next generation.
- Constraint: During the evolution, the values of the parameters in a chromosome individual should satisfy (12) to (14).

$$\sum_{j=1}^{N} InputDataSize_j = TotalDataSize \tag{12}$$

$$mapreduce.task.io.sort.mb < mapred.child.java.opts \tag{13}$$

$$\begin{aligned} mapreduce.job.reduces \leq\ & mapreduce.tasktracker \\ & .reduce.tasks.maximum \end{aligned} \tag{14}$$

- Chromosome and gene: The coding of the chromosome is decimal coding. An example of a chromosome and its genes (4 nodes in one cluster) is shown in Fig. 7.

V. PERFORMANCE EVALUATION

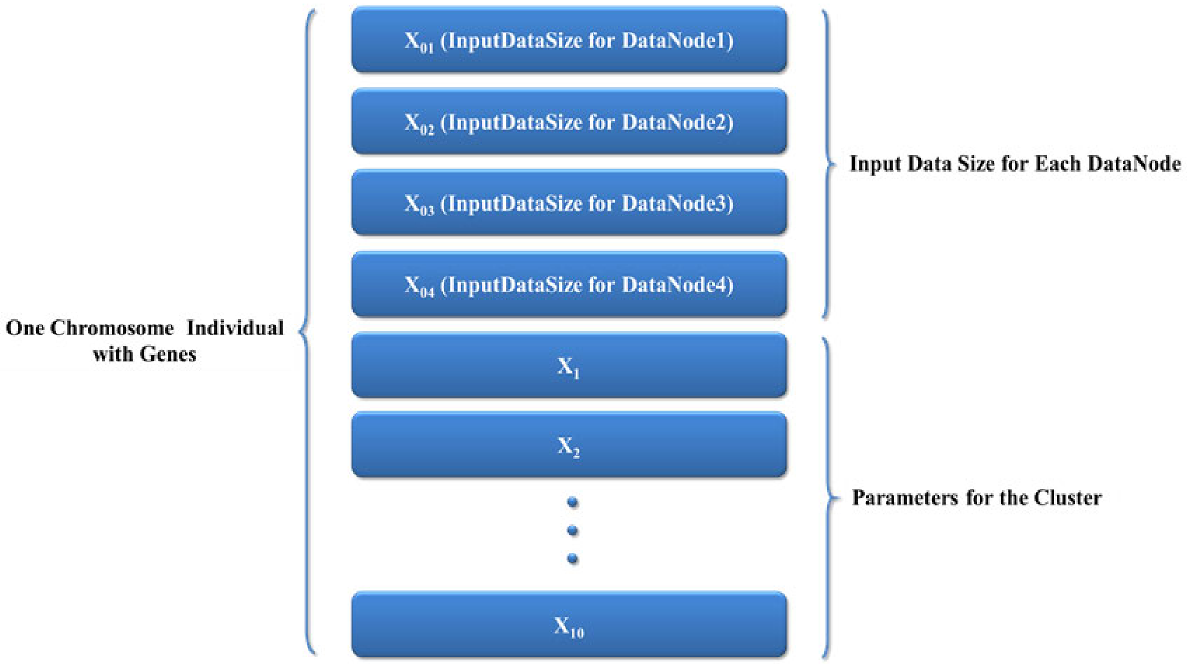
A. EXPERIMENTAL ENVIRONMENT

To evaluate the performance of the resource aware parallelized BPNN algorithm, a physical cluster with one NameNode and four heterogeneous DataNodes is established. The details of the cluster are shown in Table 4.

The cluster maximally supplies 16 mappers and 4 reducers. The dataset employed in the following experiments is a standard benchmark dataset Iris dataset [26], of which the details are shown in Table 5. The algorithm efficiency and accuracy are evaluated based on the classification task using the Iris dataset. In the algorithm accuracy evaluations, the number of the training instances is from 10 to 120 whilst the rest instances are the testing instances. The bootstrapping number is initially set to 4. In the algorithm efficiency evaluations, the data size is increased from 4MB to 1024MB.

B. EVALUATION OF ALGORITHM ACCURACY

The experiments in this section focus on the algorithm accuracy. In terms of comparison, the standalone BPNN is also implemented. The experimental results are shown in Fig. 8 and Fig. 9.



X_{01} (InputDataSize for DataNode1)
X_{02} (InputDataSize for DataNode2)
X_{03} (InputDataSize for DataNode3)
X_{04} (InputDataSize for DataNode4)
X_1
X_2
X_{10}
Input Data Size for Each DataNode
Parameters for the Cluster
One Chromosome Individual with Genes

FIGURE 7. One chromosome and its genes of the genetic algorithm.

TABLE 4. The details of the cluster.

Node type	CPU	Memory	Hard Disk	Operating system
NameNode	Core i7@3.0GHz	8GB	750GB SSD	Ubuntu 14.04 LTS with Hadoop 2.6.3
DataNode1	Core i7@3.8GHz	32GB	240GB SSD	Ubuntu 14.04 LTS with Hadoop 2.6.3
DataNode2	Core i5@3.2GHz	8GB	1TB HDD	Ubuntu 14.04 LTS with Hadoop 2.6.3
DataNode3	Core i7@2.8GHz	16GB	1TB SSD	Ubuntu 14.04 LTS with Hadoop 2.6.3
DataNode4	Core i5@1.9GHz	8GB	256GB SSD	Ubuntu 14.04 LTS with Hadoop 2.6.3

TABLE 5. The details of the dataset.

Type	Iris
Dataset Characteristics	Multivariate
Instance Number	150
Attribute Number	4
Class Number	3

Fig. 8 shows the classification accuracies of the standalone BPNN and the parallelized BPNN without the presented load balancing algorithm (with the default capacity scheduler supplied by YARN). The results indicate that the bootstrapping and majority voting can help to improve the algorithm precision when the number of the training instances is small. However, when the number of the training instances becomes larger, the performances of the two algorithms become similar. Fig. 8 also shows that with bootstrapping and majority voting the curve of the parallelized BPNN increases stably, which indicates that the employed ensemble techniques can improve the accuracy stability.

TABLE 6. The parameters employed of the GEP algorithm.

Number of genes	4
Link function	+
Head length	6
Function set	+-*/ cos sin tan exp log sqrt abs
Fitness	max(num_r/num_t)
Terminal set	X_0 X_1 X_2 X_3 X_4 X_5 X_6 X_7 X_8 X_9 X_{10}
Population size	1000
Mutation rate	0.044
IS transposition rate	0.1
RIS transposition rate	0.1
Gene transposition rate	0.1
One-point recombination rate	0.4
Two-point recombination rate	0.2
Gene recombination rater	0.1

Fig. 9 shows the classification accuracy of the parallelized BPNN with the presented load balancing algorithm and with the default capacity scheduler. The figure indicates that the

TABLE 7. An example of the optimized parameters for processing 1GB data of the cluster.

Parameter	X_{01}	X_{02}	X_{03}	X_{04}	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9	X_{10}
Value	337.4	375.8	160.1	150.7	-	54	0.57	-	4	1	-	0.15	0.51	2

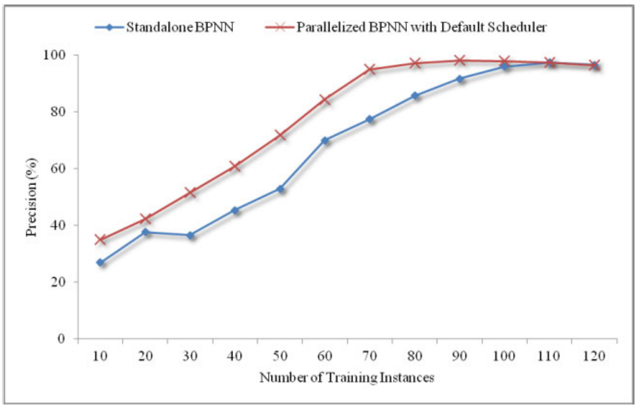


FIGURE 8. The precision comparison of standalone BPNN and parallelized BPNN.

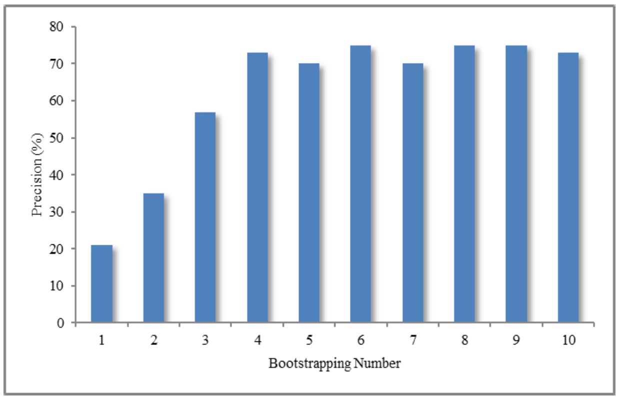


FIGURE 10. The precision affected by the bootstrapping number.

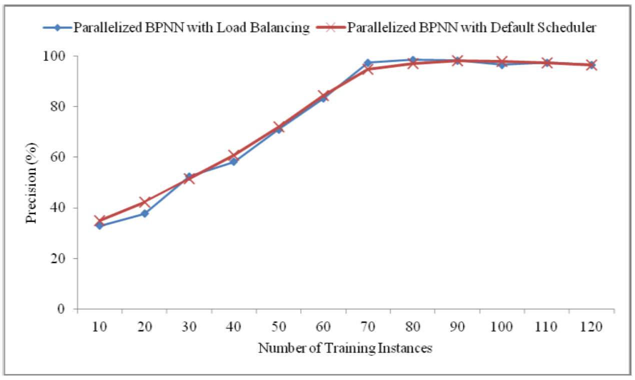


FIGURE 9. The precision comparison of the parallelized BPNN.

performances of both the parallelized BPNNs are quite similar. The slight gap between the two curves is mainly because of the differences of the parameters in the two networks.

Fig. 10 shows the accuracy of the parallelized BPNN affected by the bootstrapping number. In the experiments, a number of 50 training instances and 100 testing instances are employed. Fig. 10 shows that initially increasing the bootstrapping number can improve the algorithm accuracy. However, when a certain bootstrapped value is reached, keeping enlarging the bootstrapping number cannot significantly improve the accuracy.

C. EVALUATION OF ALGORITHM EFFICIENCY

The mined equations representing the processing times of the four DataNodes are listed in the appendix. Based on the mined equations, the load balancing is applied in the physical Hadoop cluster. The optimized Hadoop parameters are configured in the cluster, whilst the optimized size of the input training data for each DataNode is uploaded into HDFS. The performances of the load balancing algorithm with an increasing number of DataNodes in the cluster are shown in Fig. 11, 12, 13, 14 and Table 7. In terms of comparisons, the parallelized BPNN with the default capacity scheduler and the standalone BPNN are also implemented.

Fig. 11 shows the estimated performances and the real performances of the resource aware parallelized BPNN with two participated DataNodes (DataNode1 and 2). Fig. 11a suggests that theoretically the parallelized BPNN based on the presented load balancing algorithm significantly outperforms the one with the default scheduler. The experimental results based on the physical cluster shown in Fig. 11b further prove that the presented resource aware parallelized BPNN shows better efficiency due to its computing resource dispatching ability especially for an individual Hadoop job.

Similar to Fig. 11, Fig. 12a and Fig. 12b shows the algorithm performances with three participated DataNodes (DataNode1, 2 and 3). The presented resource aware parallelized BPNN performs the best.

Fig. 13 shows the algorithm performances with four DataNodes (DataNode1, 2, 3 and 4) in the cluster. The results also indicate that the efficiency of the resource aware parallelized BPNN benefits from the presented load balancing algorithm significantly. In the experiments, the parallelized BPNN with the default capacity scheduler shows worse performance even than that of the clusters with only two and three DataNodes. The reason is that DataNode4 in the cluster has the smallest processing capacity. The evenly distributed data for the DataNode costs longer processing time which greatly deteriorates the performance of the entire cluster.

Additionally the details of the optimized parameters for processing 1024MB data using the four DataNodes are listed in Table 7. The values marked by "−" indicate that the

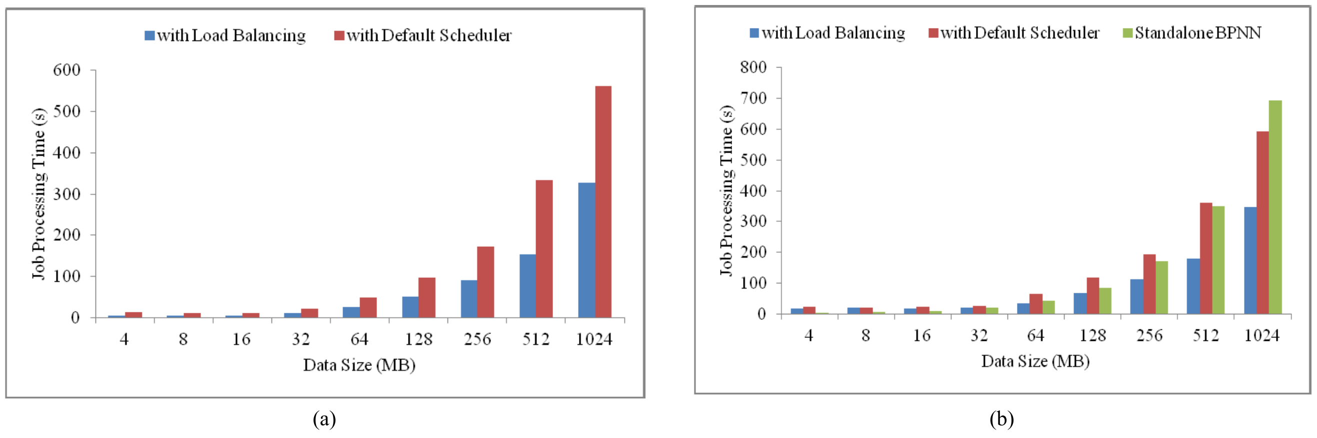


FIGURE 11. (a) The comparisons of the estimated job processing times for two DataNodes. (b) The comparisons of the real job processing times for two DataNodes.

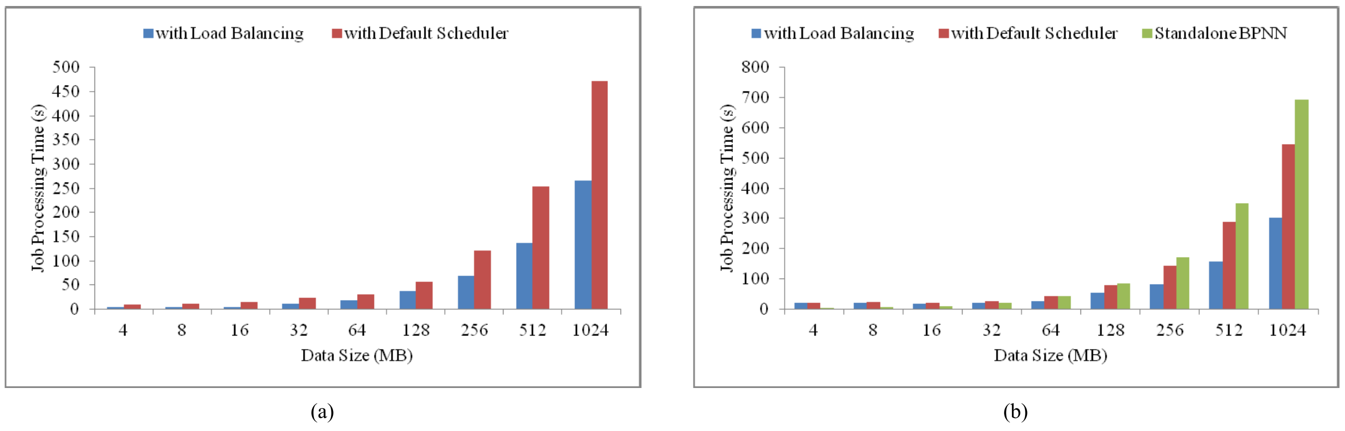


FIGURE 12. (a) The comparisons of the estimated job processing times for three DataNodes. (b) The comparisons of the real job processing times for three DataNodes.

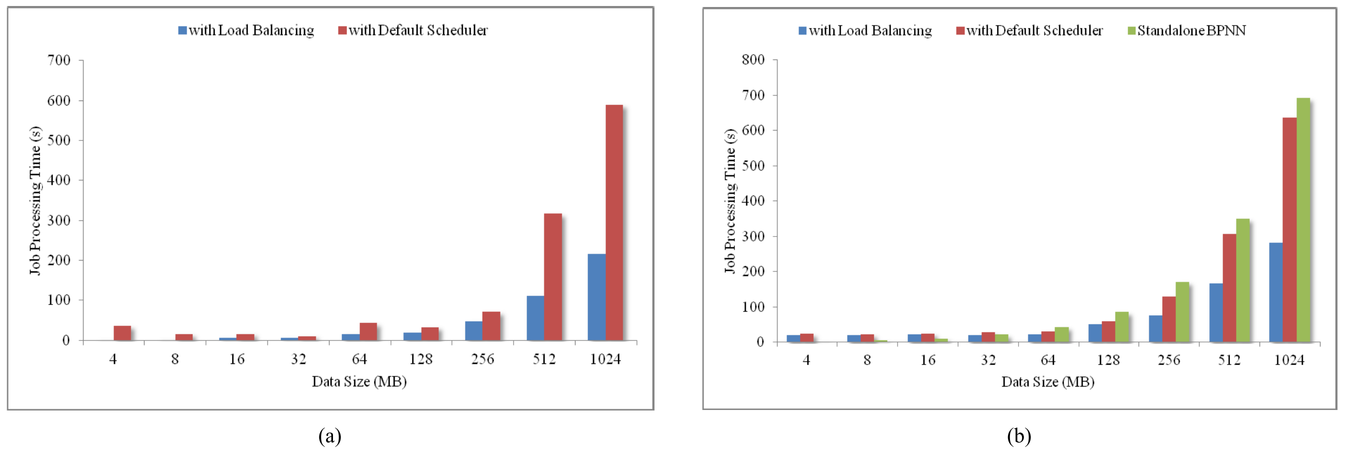


FIGURE 13. (a) The comparisons of the estimated job processing times for four DataNodes. (b) The comparisons of the real job processing times for four DataNodes.

parameters are phased out during the GEP evolution. Result shown in Fig. 13 proves the effectiveness of the optimized parameters.

Fig. 14 shows the convergence of GA for mining the optimized scheduler dealing with the data size of 1024MB. Because of the large number of parameters in a chromosome,

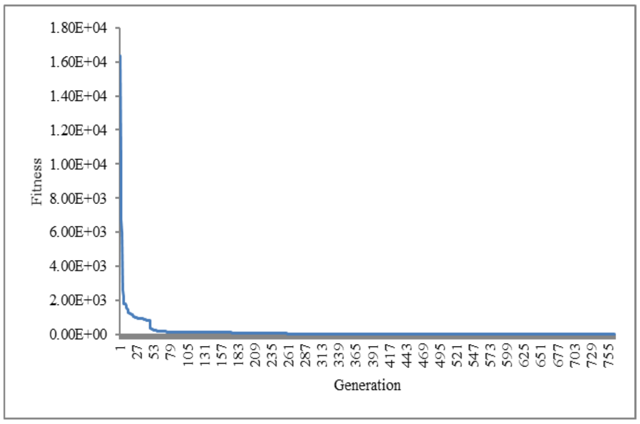


FIGURE 14. The convergence of GA for mining the optimized scheduler dealing with the data size of 1024MB.

300 generations are consumed to achieve the optimized parameters.

VI. CONCLUSION

This paper presents a resource aware parallelized BPNN algorithm to serve the large-scale digital health data processing efficiently. Based on the data separation, the traditional standalone BPNN can be parallelized using the Hadoop framework. Further in order to solve the accuracy loss issue, bootstrapping and majority voting techniques have been employed. Therefore, the distributed weak classifiers are able to compose a strong classifier to execute the accurate data processing. In terms of the load imbalance in the heterogeneous cluster, this paper also presents a load balancing algorithm using the GEP and GA algorithms. Based on the experimental results, the presented algorithm is able to handle the large-scale data processing in terms of efficiency and accuracy.

However, there are still two issues which should be addressed in our future work. The first one is that the class imbalance issue frequently exists in the practical digital health datasets. The data separation and sampling may aggravate the issue. The second one is that only a limited number of parameters are selected to achieve the optimal scheduler. However, it is known that a large number of parameters may affect the processing time of the Hadoop framework. Therefore, to develop an optimal parameter selection approach and a scheduler solution approach with better convergence ability will be great helpful to further explorer the potential of the Hadoop framework in terms of the efficiency improvement.

APPENDIX

This section shows the experimental results of measuring the processing abilities of the DataNodes. In each figure, two curves are generated. One is based on the real processing time of the DataNode, whilst the other one is the estimated processing time based on the mined equation f.

The results shown in Fig. 15 is based on the experiments of DataNode1. The mined equation is shown by (15).

$$t = (((abs(((sqrt(X_9)) + (\tan(X_6))) * ((X_0)/(X_5)))) + (\exp(X_3))) + ((((X_{10})/(X_2)) + ((X_2) + (X_5)))/(X_2))) + (\sin(X_3)) \tag{15}$$

where t denotes the data processing time of the DataNode.

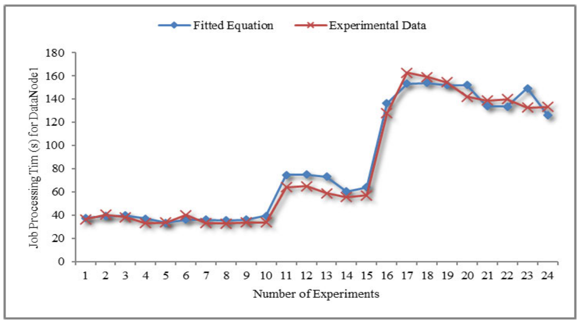


FIGURE 15. The real processing time and the processing time computed using the equation mined by GEP for DataNode1.

The results shown in Fig. 16 is based on the experiments of DataNode2. The mined equation is shown by (16).

$$t = (((\tan(\exp(\tan(X_0))))) + (\tan(\exp(\tan(\tan(X_0)))))) + ((X_0)/((X_6) + (X_3)))) + (abs(\tan(\tan(\tan(X_0))))) \tag{16}$$

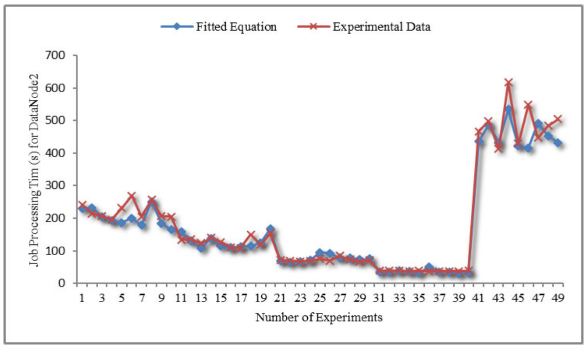


FIGURE 16. The real processing time and the processing time computed using the equation mined by GEP for DataNode2.

The results shown in Fig. 17 is based on the experiments of DataNode3. The mined equation is shown by (17).

$$t = (((-X_8)) + (\cos(X_2))) + ((\cos(X_5))/(\tan((X_9) * (X_6))))) + ((X_0)/(\cos(\cos((\cos(X_9)) - (X_8))))) \tag{17}$$

The results shown in Fig. 18 is based on the experiments of DataNode4. The mined equation is shown by (18).

$$t = ((((\cos(\log(X_0))) * (X_2)) + (X_0)) + (((X_2) * (\log(X_8))) * (\cos(X_0)))) + (X_0) \tag{18}$$

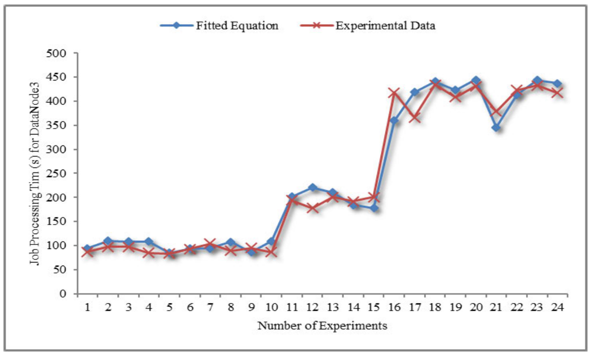


FIGURE 17. The real processing time and the processing time computed using the equation mined by GEP for DataNode3.

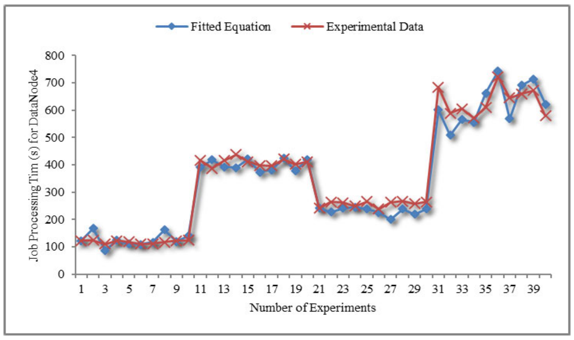


FIGURE 18. The real processing time and the processing time computed using the equation mined by GEP for DataNode4.

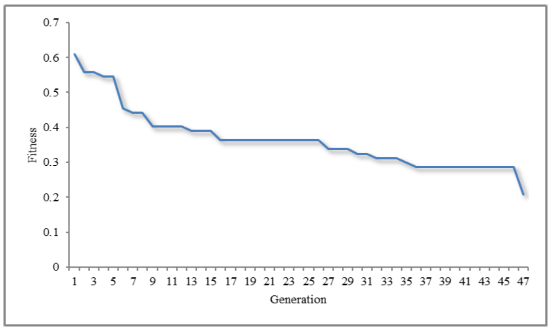


FIGURE 19. The convergence of GEP for mining the equation for DataNode2.

Fig. 19 shows the convergence of the GEP algorithm for mining the equation for DataNode2 as an example. It can be observed that to reach the target fitness, only around 50 generations are needed.

REFERENCES

[1] N. Almaadeed, A. Aggoun, and A. Amira, "Speaker identification using multimodal neural networks and wavelet analysis," *IET Biometrics*, vol. 4, no. 1, pp. 18–28, Mar. 2015.

[2] D. Fan, J. Yang, J. Zhang, Z. Lv, H. Huang, J. Qi, and P. Yang, "Effectively measuring respiratory flow with portable pressure data using back propagation neural network," *IEEE J. Transl. Eng. Health Med.*, vol. 6, 2018, Art. no. 1600112. doi: 10.1109/JTEHM.2017.2688458.

[3] K. O. Stanley, D. B. D'Ambrosio, and J. Gauci, "A hypercube-based encoding for evolving large-scale neural networks," *Artif. Life*, vol. 15, no. 2, pp. 185–212, Apr. 2009.

[4] F. Jin and G. Shu, "Back propagation neural network based on artificial bee colony algorithm," in *Proc. IFOST*, Sep. 2012, pp. 1–4.

[5] Y. Li, Y. Fu, H. Li, and S.-W. Zhang, "The improved training algorithm of back propagation neural network with self-adaptive learning rate," in *Proc. CINC*, Jun. 2009, pp. 73–76.

[6] Y.-H. Liu, S.-W. Luo, A.-J. Li, H. Huang, and J.-W. Wen, "Information geometry on extendable hierarchical large scale neural network model," in *Proc. ICMLC*, vol. 3, Nov. 2003, pp. 1380–1384.

[7] R. Gu, F. Shen, and Y. Huang, "A parallel computing platform for training large scale neural networks," in *Proc. BigData*, Oct. 2013, pp. 376–384.

[8] A. A. Huqqani, E. Schikuta, and E. Mann, "Parallelized neural networks as a service," in *Proc. IJCNN*, Jul. 2014, pp. 2282–2289.

[9] L. N. Long and A. Gupta, "Scalable massively parallel artificial neural networks," *J. Aerosp. Comput., Inf., Commun.*, vol. 5, no. 1, pp. 3–15, Jan. 2008.

[10] *Apache Hadoop*. Accessed: Mar. 2, 2019. [Online]. Available: http://hadoop.apache.org

[11] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[12] Z. Liu, H. Li, and G. Miao, "MapReduce-based backpropagation neural network over large scale mobile data," in *Proc. ICNC*, Aug. 2010, pp. 1726–1730.

[13] Y. Liu, J. Yang, Y. Huang, L. Xu, S. Li, and M. Qi, "MapReduce based parallel neural networks in enabling large scale machine learning," *Comput. Intell. Neurosci.*, vol. 2015, p. 1, Jan. 2015.

[14] H.-C. Lin and C. S. Raghavendra, "A dynamic load-balancing policy with a central job dispatcher (LBC)," *IEEE Trans. Softw. Eng.*, vol. 18, no. 2, pp. 148–158, Feb. 1992.

[15] J. Xie, S. Yin, X. Ruan, Z. Ding, Y. Tian, J. Majors, A. Manzanares, and X. Qin, "Improving MapReduce performance through data placement in heterogeneous Hadoop clusters," in *Proc. IPDPSW*, Atlanta, GA, USA, Apr. 2010, pp. 1–9.

[16] S. Dhakal, M. M. Hayat, J. E. Pezoa, C. Yang, and D. A. Bader, "Dynamic load balancing in distributed systems in the presence of delays: A regeneration-theory approach," *IEEE Trans. Parallel Distrib. Syst.*, vol. 18, no. 4, pp. 485–497, Apr. 2007.

[17] Y. Liu, W. Jing, Y. Liu, L. Lv, M. Qi, and Y. Xiang, "A sliding window-based dynamic load balancing for heterogeneous Hadoop clusters," *Concurrency Comput., Pract. Exper.*, vol. 29, no. 3, 2016, Art. no. e3763. doi: 10.1002/cpe.3763.

[18] Y. Liu, W. Jing, and L. Xu, "Parallelizing backpropagation neural network using MapReduce and cascading model," *Comput. Intell. Neurosci.*, vol. 2016, no. 2, 2016, Art. no. 2842780.

[19] Y. Liu, M. Li, M. Khan, and M. Qi, "A MapReduce based distributed LSI for scalable information retrieval," *Comput. Inform.*, vol. 33, no. 2, pp. 259–280, Jun. 2014.

[20] Y. Liu, M. Li, N. K. Alham, S. Hammoud, and M. Ponraj, "Load balancing in MapReduce environments for data intensive applications," in *Proc. FSKD*, Jul. 2011, pp. 2675–2678.

[21] M. Khan, Z. Huang, M. Li, G. A. Taylor, and M. Khan, "Optimizing Hadoop parameter settings with gene expression programming guided PSO," *Concurrency Comput., Pract. Exper.*, vol. 29, Feb. 2016, Art. no. e3786. doi: 10.1002/cpe.3786.

[22] C. Ferreira, "Gene expression programming in problem solving," in *Soft Computing and Industry*. London, U.K.: Springer, 2002, pp. 635–653. doi: 10.1007/978-1-4471-0123-9_54.

[23] A. Shokripour and M. Othman, "Survey on divisible load theory and its applications," in *Proc. ICIME*, Apr. 2009, pp. 300–304.

[24] T. G. Robertazzi, "Ten reasons to use divisible load theory," *Computer*, vol. 36, no. 5, pp. 63–68, May 2003.

[25] X. Li, X. Liu, and H. Kang, "Sensing workload scheduling in sensor networks using divisible load theory," in *Proc. IEEE GLOBECOM* Nov. 2007, pp. 785–789.

[26] *Iris Dataset*. Accessed: Feb. 26, 2019. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/Iris

[27] N. K. Alham, "Parallelizing support vector machines for scalable image annotation," Ph.D. dissertation, School Eng. Des., Brunel Univ. London, London, U.K., 2011.

[28] N. K. Alham, M. Li, Y. Liu, and M. Qi, "A MapReduce-based distributed SVM ensemble for scalable image classification and annotation," *Comput. Math. Appl.*, vol. 66, no. 10, pp. 1920–1934, Dec. 2013.

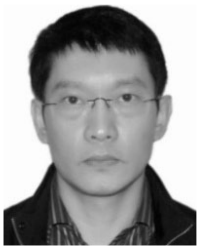
[29] Y. Liu, L. Xu, and M. Li, "The parallelization of back propagation neural network in mapreduce and spark," *Int. J. Parallel Prog.*, vol. 45, no. 4, pp. 760–779, Aug. 2017.

[30] O. Noureldeen and I. Hamdan, "A novel controllable crowbar based on fault type protection technique for DFIG wind energy conversion system using adaptive neuro-fuzzy inference system," *Protection Control Mod. Power Syst.*, vol. 3, no. 3, pp. 328–339, 2018. doi: 10.1186/s41601-018-0106-0.
[31] A. A. Majd, H. Samet, and T. Ghanbari, "k-NN based fault detection and classification methods for power transmission systems," *Protection Control Mod. Power Syst.*, vol. 2, no. 2, pp. 359–369, 2017. doi: 10.1186/s41601-017-0063-z.
[32] J. Zhang, Y. Xia, Y. Xie, M. Fulham, and D. D. Feng, "Classification of medical images in the biomedical literature by jointly using deep and handcrafted visual features," *IEEE J. Biomed. Health Inform.*, vol. 22, no. 5, pp. 1521–1530, Sep. 2018.
[33] B. Lv, Y. Chen, H. Dai, S. Su, and M. Lin, "PKBPNN-based tracking range extending approach for TMR magnetic tracking system," *IEEE Access*, vol. 7, pp. 63123–63132, 2019.
[34] Y. Qin, X. Wang, and J. Zou, "The optimized deep belief networks with improved logistic Sigmoid units and their application in fault diagnosis for planetary gearboxes of wind turbines," *IEEE Trans. Ind. Electron.*, vol. 66, no. 5, pp. 3814–3824, May 2019.

YANG LIU received the Ph.D. degree from the School of Engineering and Design, Brunel University, London, U.K., in 2011. He is currently an Associate Professor with the College of Electrical Engineering, Sichuan University, Chengdu, China. His research interests include big data analytics, power system planning and dispatching, renewable energy, and integrated energy systems.

XIANBANG CHEN is currently pursuing the master's degree with the College of Electrical Engineering, Sichuan University, Chengdu, China. His research interests include data-driven technologies, economic dispatch for power systems, and integrated energy systems.

LIXIONG XU received the Ph.D. degree from the School of Electrical Engineering and Information, Sichuan University, Chengdu, China, in 2014, where he is currently working with the College of Electrical Engineering. His research interests include machine learning in power systems, power system stability and control, and integrated energy systems.



HUAQIANG LI received the Ph.D. degree from Hiroshima University, Hiroshima, Japan, in 2004. He is currently a Professor with the College of Electrical Engineering, Sichuan University, Chengdu, China. His research interests include renewable energy, power system stability and control, and active distribution networks.

MAOZHEN LI received the Ph.D. degree from the Institute of Software, Chinese Academy of Sciences, Beijing, China, in 1997. He was a Postdoctoral Research Fellow with the School of Computer Science and Informatics, Cardiff University, U.K., from 1999 to 2002. He is currently a Professor with the Department of Electronic and Computer Engineering, Brunel University London, Uxbridge, U.K. His research interests include the areas of high performance computing, big data analytics, and intelligent systems. He is on the Editorial Boards of a number of journals. He has more than 150 research publications in these areas. He is a Fellow of the British Computer Society and the Institute of Engineering and Technology.