Journal of Network Intelligence ©2024 ISSN 2414-8105
Taiwan Ubiquitous Information Volume **9**, Number **3**, August **2024**

Optimization of Distributed Storage System of Legal Electronic Data Based on Machine Learning

Rong-He Han
Law School
Fujian University of Technology, Fuzhou 350118, P. R. China
hanronghe@163.com

Yi-Ping Han
The University of Manchester
Manchester M139PL, The United Kingdom
yipinghan395@gmail.com

*Corresponding author: Rong-He Han

Received November 25, 2023, revised January 26, 2024, accepted March 7, 2024.

Abstract. *With the advent of the big data era, the volume of electronic data in the legal service system grows dramatically, while the amount of data access gradually increases, resulting in slower and slower response times for system functions. Therefore, to solve the response time problem of legal electronic data, this work proposes a performance tuning method based on distributed storage system. Firstly, the distributed storage features are screened twice to obtain the set of parameters that affect the storage performance. In the process of performing training sample generation, orthogonal experimental design method is used to select representative feature samples for experiments to obtain the experimental values of throughput and latency. Then, the training samples are preprocessed, and the parameter samples and experimental values are combined as the sample data for the prediction model. An integrated neural network model based on Kernel Density Estimation (KDE) is proposed for the problem of sample data containing unbiased noise. The integrated neural network model is used to train and learn from the sample data, and further feature selection is performed to obtain throughput and delay prediction models. Finally, according to the characteristics of the throughput and delay prediction models, multiple swarms co-evolutionary approach is used to improve the position updating method of the traditional Fruit fly optimisation algorithm. The performance of the distributed storage system is optimised using the hybrid Fruit fly optimisation algorithm, and the optimal solution as well as the corresponding optimal parameter configurations are obtained. Experiments are conducted using four typical workloads to verify the accuracy of the proposed prediction model. The results show that the proposed tuning method can effectively improve the performance of the distributed storage system for legal electronic data.*

Keywords: legal service system; big data; feature selection; integrated neural network; fruit fly optimisation algorithm

1. **Introduction.** With the growing legal awareness of citizens, there is an increasing demand for various legal services. Usually, most of the legal services are repetitive, shared and generic [1, 2]. In the current real life, the realisation of a legal service information system has a greater practical application value. However, with the arrival of the big data era, the amount of electronic data in the legal service system [3] has increased dramatically, while the amount of data access has also increased gradually, resulting in

slower and slower response times for system functions. In the distributed storage system, the response time of some functions becomes larger with the increase of data size, and the response time of data insertion and retrieval is also getting larger.

In recent years, with the rapid development of cloud computing and the Internet, leading to the arrival of the big data era, the volume of data has grown rapidly, while the types of data have also increased dramatically, of which more than 80% is unstructured data. Since the data model of relational database is fixed, it cannot process and store new types of unstructured data and can only be used by some specific application scenarios. In order to solve the problem of processing and storing massive unstructured data, HDFS [4, 5] and MapReduce [6, 7] technologies have emerged one after another, and Google Inc. has designed and implemented a new data processing and storing scheme by combining Google File System (GFS) [8] and MapReduce framework. Meanwhile, Google designed and developed a distributed data storage system, which is used to process massive data. Distributed data storage system [9, 10] is a distributed, open-source, scalable non-relational storage system that is a powerful tool for processing and storing semi-structured and unstructured data. Database applications are able to run properly and efficiently and have become a major concern for developers. Storing more and more data in distributed databases leads to slower and slower response times for operations such as data insertion, updating, and retrieval, and the performance of distributed storage systems has become a key issue.

Therefore, under the existing equipment resources, how to improve the performance of the distributed storage system for legal electronic data by modifying the parameter configuration has become a popular issue. There is a default parameter configuration in distributed clusters, but due to the different cluster sizes and resources of each system, the default parameter configuration of distributed clusters can no longer meet the system performance requirements in many real-life situations [11]. Therefore, how to design reasonable parameter configurations to optimise system performance has also become an urgent problem to be solved. Aiming at the problems described above, the research objective of this work is to improve the service performance by adjusting the parameter configurations of the distributed storage system under the condition that the computer hardware resources are determined.

1.1. **Related Work.** the problem of performance modelling and optimization of storage systems has gained extensive attention from researchers both at home and abroad.

Zarras [12] proposed an empirical scheme that allows determining the performance characteristics of component-based applications through benchmarking and analysis, then obtaining a performance prediction model about this class of applications, and finally verifying the validity of the model using an application implemented in two base components, CORBA and J2EE. Zhao et al. [13] developed an algorithm for analysing the performance of a black-box distributed system by acquiring message traces of system activities, two different algorithms were developed to analyse the node-specific causal relationships of a black-box distributed system from these traces, thus enabling developers to overcome performance bottlenecks.

Currently, the performance analysis of configurable software systems is also attracting a lot of attention. Pham et al. [14] proposed a methodology to derive a performance impact model for configurable systems. This methodology trained a performance impact model by describing the impact of all relevant configuration options and their interactions, and combining machine learning and heuristic sampling using stepwise linear regression. This model describes the relationship between configuration options and their interactions with the performance of a configurable system. Yu et al. [15] proposed a method to automatically detect the interactions of performance related features as a way to improve the accuracy of prediction, and the results showed an average prediction accuracy of 95%. Betta et al. [16] proposed a Fourier transform based algorithm to predict the performance of any configurable system and mapped the relationship between all possible combinations of configurations of n features of a system and its performance value into a Boolean function, i.e., a performance model. Vieira and Eisner [17] proposed a method for exact parameter space pruning, and the results revealed that some of the candidate parameters were filtered out through a priori knowledge in optimisation procedures with evolutionary algorithms.

Machine learning techniques have shown greater advantages in the performance modelling and optimisation problems of storage systems. Jiang and Hassan [18] proposed a linear regression model which is used to evaluate the interaction of loaded concurrent queries in a distributed database system. The results show that two important factors affecting query performance are network latency and local processing overhead. Gao et al. [19] proposed a machine learning model based performance optimisation approach for distributed systems. The authors used machine learning techniques to construct an integrated performance prediction model that can predict the performance of a distributed system and based on this, resource allocation and load balancing is performed. The authors also propose a new resource scheduling strategy that combines machine learning and deep reinforcement learning techniques, which can allocate resources based on system load and demand to further improve system performance.

Based on the above analysis, it can be seen that distributed storage systems have become the current mainstream solution. However, there are a large number of nonlinear relationships between target variables and process variables in distributed storage systems, which arise from the fact that many programmes are composed of multiple processes in series or parallel, which gives rise to nonlinear problems between many variables, which leads to defective problems in the output of traditional machine learning models when they are applied to distributed storage systems, such as the partial least squares (PLS) [20, 21] and the Support Vector Regression (SVR) [22]

1.2. **Motivation and contribution.** The unique multi-layer architecture of neural networks and the activation function at the connection of each layer make it possible to derive a wide variety of network forms, thus enabling neural networks to extract features suitable for the task through multi-layer non-linear mapping. In addition, the kernel function method maps the projection of low-dimensional raw data to a high-dimensional space, i.e., the raw data is upgraded by a nonlinear function, which is conducive to solving the nonlinear problem between variables. Therefore, in order to solve the response time problem of legal electronic data, this work proposes a performance tuning method based on distributed storage systems in order to obtain the optimal parameter configuration of distributed clusters for a given device resource.

The main innovations and contributions of this work include:

(1) Construct a performance prediction model. The training samples are preprocessed, and the parameter samples and experimental values are merged as the sample data for the prediction model. An integrated neural network model based on Kernel Density Estimation (KDE) [23] is proposed to address the problem of sample data containing unbiased noise. The integrated neural network model is used to train and learn from the sample data, and further feature selection is performed to obtain throughput and delay prediction models.

(2) Performance optimisation based on predictive models. According to the characteristics of throughput and latency prediction models, multiple swarm co-evolution methods

are used to improve the position updating method of the traditional fruit fly optimisation algorithm. The performance of the distributed storage system is optimised using the hybrid Fruit fly optimisation algorithm, and the optimal solution as well as the corresponding optimal parameter configuration are obtained.

2. **Feature selection and training sample generation.**

2.1. **Performance feature selection.**

The distributed storage is parameterised using the file hbase-site.xml. There are as many as 197 features in distributed storage [24, 25], which mainly contain six aspects such as Client, CallQueue, MemStore, BlockCache, HStoreFile and WAL, but among them, the features that affect the performance of distributed storage need to be further filtered.

Since there are so many features of distributed storage, only some of them are given here. This partial list of features allows you to view detailed information about each feature. Many features just declare a configuration path, a port number or define a field that does not have an impact on performance, for example, the feature hbase.rootdir is the shared directory of the RegionServer. The only effect of the feature hbase.cluster.distributed is to distinguish whether the cluster is fully distributed or pseudo-distributed. The feature hbase.zookeeper.quorum primarily indicates the nodes running Zookeeper. The feature hbase.master.info.port indicates the port number of the interface that is open to the public, which is usually configured as 60020. So when you do the first feature filtering, you can eliminate those features that have no impact on the performance according to the feature descriptions.

In the first filtering step, parameter features that have no impact on storage performance at all need to be filtered out. After the first screening, it is found that the remaining parameter features are still relatively large, at this time, the parameter feature set also contains many parameters that have a very small impact on the performance, so in the next step of feature filtering, it is necessary to filter out this part of the features as well.

Table 1. Selected Parameter Characteristics Affecting Distributed Storage Performance.

Feature name	Eigenvalue type	Feature Defaults	Degree of impact
hbase.rootdir	double	3000	High
hbase.cluster.distributed	double	0.1	High
hbase.zookeeper.quorum	int	32	High
hbase.local.dir	double	1.0	High
hbase.master.port	int	5	High
hbase.master.info.port	int	5	High
Hbase.region.server.handler.count	int	30	High
hbase.ipc.server.callqueue.handler.factor	double	0	High
hbase.ipc.server.callqueue.read.ratio	double	0	High
...	...	...	...

Some parameters have a low impact on performance, for example, hbase.master.logcleaner.ttl, hbase.rpc.timeout, etc. The impact of these features on performance can be ignored, and therefore, these features can be filtered out during the second feature filtering. There is also a part of parameter features whose impact is defined as medium, for example, hbase.snapshot.master.timeout.millis, etc. Although the impact of these features on the performance is defined as medium, according to the application experience in the actual project, it is found that some of these features have negligible impact on the performance in the actual system, so this part of the parameter features is filtered out, and thus these features can be filtered out during the second feature filtering. Therefore, this part of the parameter features is filtered out, so that the parameter features that affect the performance are selected, and a total of 24 parameter features are selected. In this paper, the information of these parameter features will be given in the form of table, which mainly includes the feature name, feature value type, feature default value, and the influence of the feature on the performance, as shown in Table 1.

2.2. **Training sample generation.** After two steps of feature screening, 24 parameters in the distributed cluster are screened. In this paper, YCSB tool is used to simulate the client to experiment on distributed storage performance. Therefore, two performance-related parameters in YCSB tool, client threads (ycsb.client.threads) and data requests (operationcount), which simulate the number of client threads and user data requests in the actual system, are screened. When an orthogonal experimental design is used to conduct the experiment, the number of levels for each parameter feature and the values of the levels to be taken need to be determined first.

Algorithm 1 Method for solving orthogonal matrices

```
1: Input: Level of features $S$, orthogonal matrix $[a_{i,j}]_{N\times M}$
2: Output: Orthogonal matrix corresponding to the number of feature levels $[a_{i,j}]_{N\times M}$
3: Initialise the matrix $[a_{i,j}]_{N\times M}$ to all zeros.
4: for $k = 1$ to $H$ do
5:     $j = \text{Math.pow}(S, k-1)/(S-1)+1$
6:     for $j = 1$ to $N$ do
7:         $temp = (i-1)/\text{Math.pow}(S, H-k)$
8:         $a_{i,j} = \text{Math.floor}(temp)\%S$
9:     end for
10: end for
11: for $k = 2$ to $H$ do
12:     $j = \text{Math.pow}(S, k-1)/(S-1)+1$
13:     for $s = 1$ to $j-1$ do
14:         for $t = 1$ to $S-1$ do
15:             for $i = 1$ to $M$ do
16:                 $a(i, j+(s-1)\times(S-1)+t) = (a(i,s)+a(i,j))\%S$
17:             end for
18:         end for
19:     end for
20: end for
21: Add 1 to each element of the orthogonal matrix.
22: return $[a_{i,j}]_{N\times M}$
```

Each parameter feature takes 5 values within the range of values. If a parameter feature does not have 5 values in the range of values, you can use the proposed level method, which repeats some levels of this feature several times to make up to 5 levels. For example, the parameter feature hbase.hstore.compactionThreshold is an integer parameter with a default value of 3, a value range of [3,6], and only 4 level values. We can repeat this parameter feature once for each of 5 and 6 to make up to 5 levels (e.g. 3,4,5,6,5,6).

In this paper, a 23-factor 5-level orthogonal test was used to select a representative sample set of parameters for the experiment. The number of orthogonal table trials is calculated as shown in Equation (1).

$$N = \sum_{i=1}^{M}(S-1)+1 \tag{1}$$

$$M = \frac{S^H - 1}{S-1} \tag{2}$$

where M denotes the number of factors, S denotes the number of levels for each factor, N denotes the number of experiments, and H is a positive integer.

In this experiment, the value of S is taken as 5 and the value of M is taken as 23, which gives the value of H as 3. Therefore, in this paper, an orthogonal matrix with 125 rows and 23 columns is designed. The orthogonal table solving procedure for this experiment is shown in Algorithm 1 with time complexity $O(SHM^2)$ and space complexity $O(1)$.

3. **Performance prediction model based on integrated neural network.**

3.1. **Data preprocessing.**

Data preprocessing is the first step in the whole process of making throughput and latency predictions, which involves merging the experimental values of throughput as well as latency with the corresponding parameter feature samples and converting them into a data format in which the integrated neural network can be trained as a model.

Data preprocessing is mainly done by reading the log files of the experiments to obtain the experimental values of throughput as well as delay and combining them with the parameter feature list to generate a CSV file which is used as an input to the integrated neural network. Since the integrated neural network is stochastic and insensitive to the possible outliers in the experiments, there is no need to process the outliers of throughput and delay when constructing the prediction model using the integrated neural network. For data preprocessing, the log file and parameter feature list need to be read and then all the sample data are loaded into the programme.

Firstly read the parameter configuration sample data from the file directory and store the data into the two dimensional array conf_data[rowNum][columnNum], then read the experimental values of throughput and latency from the log files. Finally, these two arrays are deposited into two CSV files throughput_csv and latency_csv respectively, which are used as input files for the integrated neural network training model.

3.2. **Integrated neural network model based on KDE.**

In this paper, throughput prediction model and delay prediction model are obtained by training sample data with integrated neural network model. Generating the prediction models consists of a total of three steps, the first step is to read the training sample file and differentiate between the features and the two target values, i.e., throughput and latency. The second step is to train the sample data using the integrated neural network based on the features and the two target values respectively and get the importance of each feature in the model and again perform the feature selection to get the final prediction model.

Various data in distributed storage processes often face superimposed interference from serial and parallel processes, etc., which manifests itself as interference from unbiased noise and tends to produce a large amount of weakly labelled data, which leads to a large number of nonlinear relationships between the target variables and the process variables. However, traditional machine learning is difficult to cope with such problems. To address the problem of large number of weak labels in distributed storage processes, an integrated neural network model based on KDE is proposed.

KDE is a nonparametric density estimation method [26], which mainly utilises a method based on experimental data to estimate the probability density function of a particular random variable. Two types of methods are included in probability density estimation: parametric density estimation and nonparametric density estimation. Unlike parametric density estimation, nonparametric density estimation produces more accurate results, and its applicability is enhanced by the fact that no prior assumptions are made about the distribution. The probability density estimation used subsequently in this paper is the nonparametric estimation method.

Histograms are a common form of nonparametric density estimation, and kernel density estimation is also a commonly used method of nonparametric density estimation, which is based on nonparametric estimation of the calculated probability density function is shown below:

$$p(x) = \frac{1}{Nh} \sum_{i=1}^{N} K\left(\frac{x - x_i}{h}\right) \tag{3}$$

where $p(x)$ denotes the calculated probability density function, N is the number of samples, h denotes the width of the computational window, and K denotes the selected kernel function. The commonly used kernel functions are triangular kernel, Gaussian kernel and linear kernel.

The value of the window size h affects the results of the final kernel density estimation. When h is too large, the data tends to flatten out, and when h is chosen too small, it makes the data samples too dispersed to collect the characteristics of the sample data itself, leading to too much randomness. Therefore the common practice is to choose the window width empirically.

$$h = 1.06\hat{\delta}N^{-\frac{1}{5}} \tag{4}$$

where $\hat{\delta}$ is the sample standard deviation.

As shown in Figure 1, a Multi-Layer Perceptron (MLP) [27] is a fully-connected layer, where each layer is connected using an activation function to ensure the existence of a nonlinear mapping. Because of the activation function, the MLP has the ability to learn nonlinearly, and the number of layers and neurons in the hidden layer of the MLP is generally designed according to the requirements, and the MLP is trained by Backpropagation (BP) algorithm to obtain the corresponding parameters of the fully-connected network.

The BP algorithm uses Jacobi matrices to multiply with vectors. The mapping of vectors to a function of quantities is assumed to have a gradient given the loss function $l = f(y)$, where y is the output of the neural network:

$$\nabla l = \left(\frac{\partial l}{\partial y_1}, \ldots, \frac{\partial l}{\partial y_m}\right)^T \tag{5}$$

As a variety of traditional machine learning algorithms, the main idea of Random Forests (RF) algorithm [28] is to integrate multiple decision trees in parallel through the integration learning Bagging idea. Usually, RF is used in classification problems, for classification problems, the results of the model are voted by a number of decision trees within the model, and the final result is to select the classification with the most votes. For regression tasks, the results of the model are selected as the predicted mean of all decision trees.

Assuming that the input feature (independent variable) under a particular dataset is X and the target variable (to be predicted target label) is y, and (X, y) conforms to an independent distribution, then randomly generating the training set θ at (X, y), the

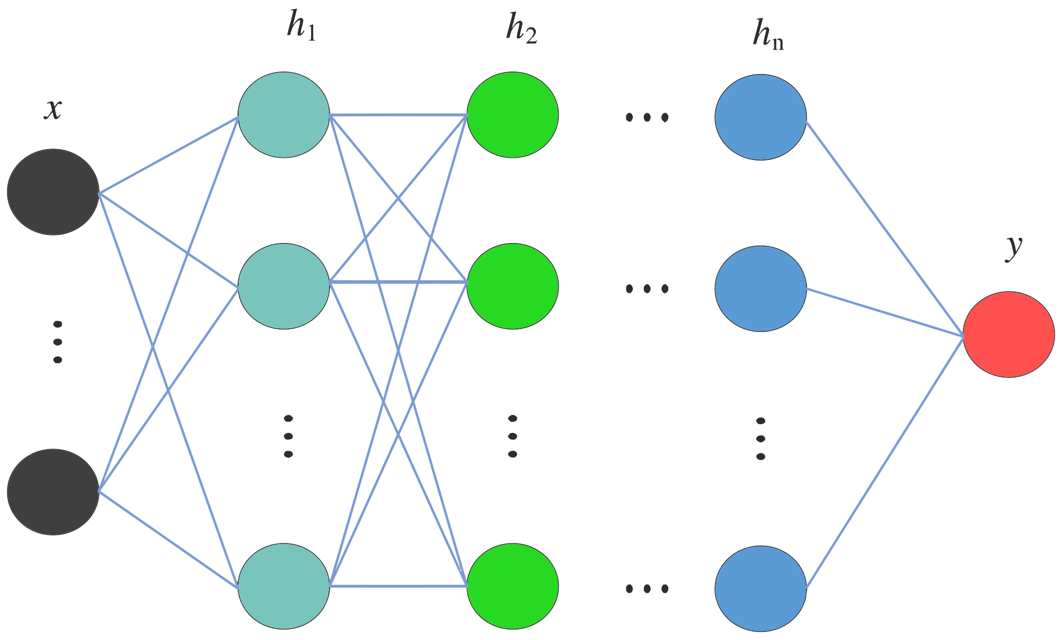

Figure 1. Structure of the MLP

model prediction result is denoted as $f(x)$, which exhibits a mean-square generalisation error:

$$E_{X,y}[(y - f(X))^2] \tag{6}$$

Suppose there are n decision trees, then the mean of the predicted values $\{f(\theta, X_n)\}$ of the n decision trees is the prediction of the random forest regression. When $n \to \infty$, then there are:

$$E_{X,y}[y - \bar{f}_n(X, \theta_n)]^2 \to E_{X,y}[y - E_\theta(X, \theta_n)]^2 \tag{7}$$

When the number of decision trees n is infinite, the average generalisation error of a single decision tree is denoted GE.

$$GE = E_\theta E_{X,y}\left[y - f(X, \theta)\right]^2 \tag{8}$$

$$E_{X,y}\left[y - E_\theta(X, \theta_n)\right]^2 \leq \overline{\rho} GE \tag{9}$$

where $\overline{\rho}$ refers to the weighted correlation coefficient between residuals.

The final regression function for RF is:

$$Y = E_\theta f(X, \theta) \tag{10}$$

For the noise reduction problem, the training dataset of RF has a high degree of randomness, which brings variability to each tree model. When the tree model itself has a strong fitting learning ability, when the training set of each tree model contains the complete distribution of the target variable. In this case, each tree model represents the predicted value of that label under the influence of noise, i.e., it can be regarded as a sampling of that true value under noise interference.

Based on the above analysis, inspired by the randomness of random forests, the strong fitting of neural networks, and the accuracy of KDEs, this work combines the three to propose an integrated neural network based on KDEs. The integrated neural network is built based on the open-source deep learning platform PyTorch, and the number of layers l, the number of hidden layer nodes m, the model training learning rate $p1$, the model training optimiser Adam, and the number of training cycles ep1 are set for a single lit-tree neural network, and a set of training data is randomly drawn from the training set to form a sub-model training set. The parameters of each tree model are initialised, so that the distribution of layer l parameters W_i is shown below:

$$W_l \sim N\left(0, \sqrt{\frac{2}{n_l}}\right) \tag{11}$$

where n is the number of neurons in the l-th layer.

With the above distribution process, the exponential effect of the signal in forward propagation and the exponential effect of the gradient in backward propagation can be avoided when the tree model is trained, i.e., the problem of vanishing gradients is avoided from some points of view. The forward propagation process is shown below:

$$z_l = W_l X_{l-1} + b_l \tag{12}$$

$$X_l = f(z_l) \tag{13}$$

where W_l is the l-th linear layer weight matrix, X_{l-1} is the output of the previous layer which is the input of this layer, Z_l is the output of this layer after linear layer and X_l is the input of the next linear layer.

Using `ReLU()` as the activation function, it can be obtained:

$$Var(z_l) = n_{l-1}E(W_l)^2 E(X_{l-1})^2 = n_{l-1}Var(W_l)E(X_{l-1})^2 \tag{14}$$

And we expect the state of each intermediate layer to remain unchanged during forward propagation.

$$\frac{1}{2}n_{l-1}\mathrm{Var}(W_l) = 1 \tag{15}$$

It can thus be seen that by initialisation, each state layer of the network will not be biased by the propagation of the distribution of each state layer, leading to the problem of vanishing gradients due to state bias.

When the tree models are all trained, the RF model $\{f_i(\theta_i), i = 0, 1, \ldots, n-1\}$ is used for label prediction. When the data is disrupted, each sub-model training set will contain complete information about the noise distribution, and due to the high fit of the neural network, it is able to learn the noise situation under a certain moment. Then, KDE fitting is performed to obtain the window h according to Equation (4) and the probability density value of the corresponding point is calculated based on Equation (3). We can obtain the probability density estimate $p(y)$ under the corresponding fitted probability density function.

$$p(y) = \frac{1}{n \cdot h}\sum_{i=1}^{m} K\left(\frac{y - x_i}{h}\right) \tag{16}$$

Finally, the pre-processed data samples are trained using an integrated KDE-based neural network to obtain throughput prediction models and label models, and the models are scored.

The closer the importance of a feature is to 0, the less impact the feature has on throughput or latency. If the importance of a feature is equal to 0, it means that the feature has no effect on throughput or latency. Feature selection is performed during the training of the model to get the throughput and latency prediction model. Taking the workload UH as an example, the importance of all features in the throughput prediction model is shown in Table 2.

Table 2. Significance of features of throughput model under workload UH.

Parameter characteristics	degree of importance
ycsb.client.threads	0.3671
hbase.regionserver.handler.count	0.0729
ycsb.operationcount	0.059
hbaseregiosnservermaxlogs	0.0341
hbase.ipc.server.callqueue.scan.ratio	0.0298
hbase.client.max.perserver.tasks	0.0213
hbase.hstore.compactionThreshold	0.0206
io.storefile.bloom.block.size	0.0204

4. Performance tuning of hybrid Fruit fly-based optimisation algorithms.

4.1. Problem definition.

Improving the performance of distributed clusters on the basis of existing resource devices has become a problem that must be solved in this project. To address this problem, this paper implements a performance tuning method, which is divided into two main steps:

The first step is to model the distributed performance using integrated neural networks. The goal is to obtain the throughput as well as the relationship between latency and features $(a_1, a_2, \ldots, a_n)$.

$$T = F(a_1, a_2, \ldots, a_n) \tag{17}$$

$$L = G(a_1, a_2, \ldots, a_n) \tag{18}$$

A system needs to consider both throughput and latency, so it needs to find a balance between throughput and latency. Define the performance of distributed storage as the weighted sum of throughput and latency, then the relationship between performance and characteristics is shown below:

$$\mathrm{pref} = w_1 \times T + w_2 \times L \quad (w_1 + w_2 = 1) \tag{19}$$

where pref denotes the performance of distributed storage, w_1 denotes the weight of throughput, and w_2 denotes the weight of latency.

The second step is to implement the performance optimisation algorithm. An optimisation algorithm is designed to obtain the optimal performance pref according to Equation (19) and the optimisation algorithm along with the corresponding parameter configuration $(a'_1, a'_2, \ldots, a'_n)$.

In this paper, the relationship between distributed storage performance and features is investigated by using integrated neural networks to predict throughput and latency. In addition, an hybrid FOA algorithm is used to optimise the performance of the distributed storage system, and the optimal solution as well as the corresponding optimal parameter configurations are obtained, which ultimately implements a performance tuning method.

The main objective of the study is to obtain the optimal solution for performance and the corresponding parameter configuration, i.e., to obtain the maximum value of performance and the corresponding parameter configuration. Using the hybrid FOA algorithm with throughput and delay prediction models, the performance is optimised to obtain the parameter configurations when the performance is optimal, as shown in Figure 2. The two performance models and the set of parameter configurations are used as inputs to automatically perform the performance optimisation and search for the optimal parameter configurations on a global scale.

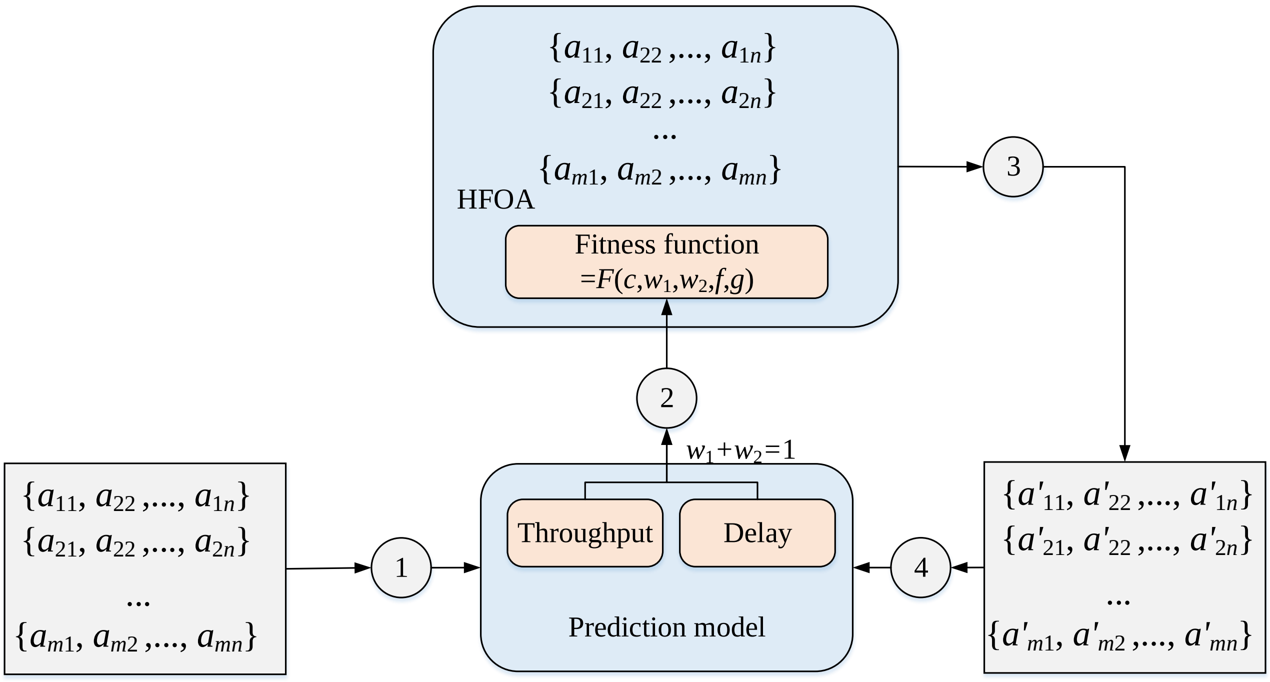

Figure 2. Performance tuning methodology

4.2. **Hybrid hybrid fruit fly optimisation algorithm.** According to the characteristics of the throughput and latency prediction models, multiple swarm co-evolutionary methods are used to improve the position update method of the traditional fruit fly optimisation algorithm. The performance of the distributed storage system is optimised using the hybrid fruit fly optimisation algorithm, and the optimal solution as well as the corresponding optimal parameter configuration are obtained.

In this paper, we use the multiple swarm co-evolutionary strategies of the Sparrow Search Algorithm (SSA) [29] to improve the Fruit Fly Optimisation Algorithm (FOA) [30], and design a Hybrid Fruit Fly Optimization Algorithm (HFOA).

(1) Population coding.

Set the position vector of fruit fly individual i to be X_i, then the component x_i^j should satisfy the initial search range of fruit fly, $1 \leq x_i^j \leq n$.

(2) Location update.

FOA has the advantages of simple and clear optimisation mechanism, easy to understand, and easy to implement in programming, etc. In order to retain its essential optimisation characteristics and to solve the problem that the algorithm tends to fall into local extremes, instead of adopting the traditional way of updating the position of the population, the traditional way of updating the position of the population is used here, and instead, it combines the multiple swarms of the co-evolutionary method in SSA.The core idea of SSA is that the sparrow population is divided into the discoverers and the joiners, and the discoverers are responsible for the population's finding food and providing foraging areas and directions for the entire sparrow population, and the joiners use the discoverers to obtain food. In addition, when the sparrow population is aware of the danger, it will make anti-predator behaviour, so as to prevent the algorithm from falling into local optimality.

Based on the core idea of the sparrow search algorithm, we set up a fruit fly group by sorting the individual fruit fly odour concentration values from small to large. The first 50% are "discoverers", the second 50% are "joiners", and "anti-predator behaviour" only occurs in the first 10% of "discoverers". Anti-predation behaviour occurs only in the first 10 per cent of the "discoverers".

Setting up fruit fly "discoverers" to conduct broad-scale searches is a diversity conservation strategy.

$$X_i^{t+1} = \begin{cases} X_i^t + \mathrm{rand}(\mathrm{FR}_1), & R_i \le \mathrm{ST}_1; \\ X_i^t + \mathrm{rand}(\mathrm{FR}_2), & \text{otherwise.} \end{cases} \tag{20}$$

where t is the number of iterations, x_i^t is the position vector of fruit fly individual i in generation t, FR1 is the fully random flight range, which is determined by the performance tuning problem, and FR2 is the semi-random flight range, which is determined by the constraints of hardware resources.

Setting up to perform a secondary search obeys a strategy to accelerate convergence.

$$X_i^{t+1} = \begin{cases} \mathrm{Xp}^{t+1} + \mathrm{rand}(\mathrm{FR}_3), \mathrm{Smell}_i^t \le \mathrm{ST}_2; \\ X_i^t + \gamma_1 \cdot \mathrm{rand}(\mathrm{FR}_2), \text{otherwise.} \end{cases} \tag{21}$$

where Smell_i is the value of odour concentration of fruit fly individual i, Xp is the current optimal position occupied by the "discoverer", and γ_1 is the perturbation factor. The individual flight range FR3 is determined by Xp.

By setting "anti-predator behaviour", the fruit fly is prevented from always staying at a certain position and from falling into a local optimum.

$$X_i^{t+1} = \begin{cases} X_i^{t+1}, \mathrm{Smell}_i^{t+1} < S_{\mathrm{best}}^t; \\ X_{\mathrm{best}}^t + \gamma_2 \cdot \mathrm{rand}(\mathrm{FR}_4), \text{otherwise.} \end{cases} \tag{22}$$

where X_{best} is the current global historical optimal position, S_{best} is the current global historical optimal odour concentration value, γ_2 is a perturbation factor, and FR4 is the individual flight range, as determined by X_{best}. Individual flies flew randomly to the vicinity of the global historical optimal position, and the flight distance was constrained by controlling the number of FR4 components that were zero.

4.3. **HFOA-based performance tuning.** To compute the fitness value of an individual using HFOA, it needs to be decoded and the decoded individual is substituted into the fitness function for computation. Therefore, the real coded value of each feature corresponding to the actual value of the feature is calculated as shown below:

$$x_i = \lfloor (x_{i_{\max}} - x_{i_{\min}}) \times gene_i + x_{i\ \min} \rfloor \tag{23}$$

where x_i denotes the actual value of the i-th feature, $x_{i_{\max}}$ denotes the maximum value allowed for the i-th feature, $x_{i_{\min}}$ denotes the minimum value allowed for the i-th feature, and $gene_i$ denotes the real number encoding of the i-th feature. For example, the first feature in an individual, hbase.client.max.peregrion.tasks, has a real number encoding of 0.13 and a value range of [1, 4], so the actual value corresponding to this configuration parameter is 1.

When solving optimisation problems using HFOA, there is a measure of the degree of individual superiority or inferiority, which is the fitness function. In real problems, the fitness function and the objective of the problem are not necessarily identical. The optimisation problem in this paper is to find the maximum value of the objective function pref. What needs to be optimised in this paper is the weighted sum of throughput and delay, which belongs to the single-objective, multi-dimensional optimisation problem, so the fitness function used is shown below:

$$fit(f(x)) = f(x) \tag{24}$$

where $f(x)$ denotes the objective function.

5. **Experimental results and analyses.**

5.1. **Experimental environment.** Four machine learning algorithms are used to construct models respectively, and the prediction results of multiple models are compared and validated and analysed to verify the accuracy of the prediction models in this paper. The optimisation results of the HFOA algorithm and three optimisation algorithms are compared and analysed to verify the efficiency of the HFOA algorithm.

Since the required distributed storage system for legal electronic data contains one master node and eight slave nodes, it is necessary to create nine virtual machines on a well-performing server and install CentOS system to build a distributed cluster. Secondly, this paper uses YCSB tool to test the performance, so it is also necessary to create a virtual machine on the server, install CentOS system and install YCSB tool to simulate the client. The specification parameters and software versions of the server and virtual machine in the experiment are shown in Table 3, Table 4, and Table 5.

5.2. **Predictive model validation.** According to Algorithm 1, 3200 sets of training sample data are obtained and the sample data are preprocessed. Then, the data are trained by four algorithms to generate the corresponding prediction models. four different workloads are shown in Table 6.

Table 3. Server Parameter Specifications.

Parameters	**Numerical value**
Operating system	Centos 7.5
Processor model	Intel Xeon CPU E5-2698 v3 @2.30GHz
Number of cores	16
Random access memory (RAM)	128 G
Hard drive	10 TB

Table 4. Parametric specifications of the laboratory computer.

Parameters	**Numerical value**
Number of virtual machines	9
Operating system	Centos 7.5
Processor model	Intel i5-13600kf
Number of cores	14
Random access memory (ram)	8 g
Hard drive	80 gb
Bandwidths	1.0 gbps

Table 5. Experimental environment software version.

Software name	**Hadoop**	**HBase**	**YCSB**	**JDK**
Version number	2.6.0	1.0.3	0.13.0	1.8.0

Table 6. 4 different workloads.

Filename	**Paradigm**
Workloada	update-heavy (UH)
Workloadb	read-heavy (RH)
Workloadc	read-only (RO)
Workloadd	read-latest (RL)

For different workloads, integrated neural network, RF, MLP and Support Vector Machine (SVM) [31] are used to train the sample data and construct the prediction models to obtain four different throughput prediction models and four different delay prediction models, respectively. Then 300 groups of test samples were randomly selected and substituted into the four throughput and delay prediction models to compare the obtained predicted values with the actual measured values. These prediction models are compared and analysed to verify the accuracy of the prediction models of the KDE-based integrated neural network. Figure 3 illustrates the error rates of the throughput models for four different workloads. Figure 4 shows the error rates of the delay models for these four

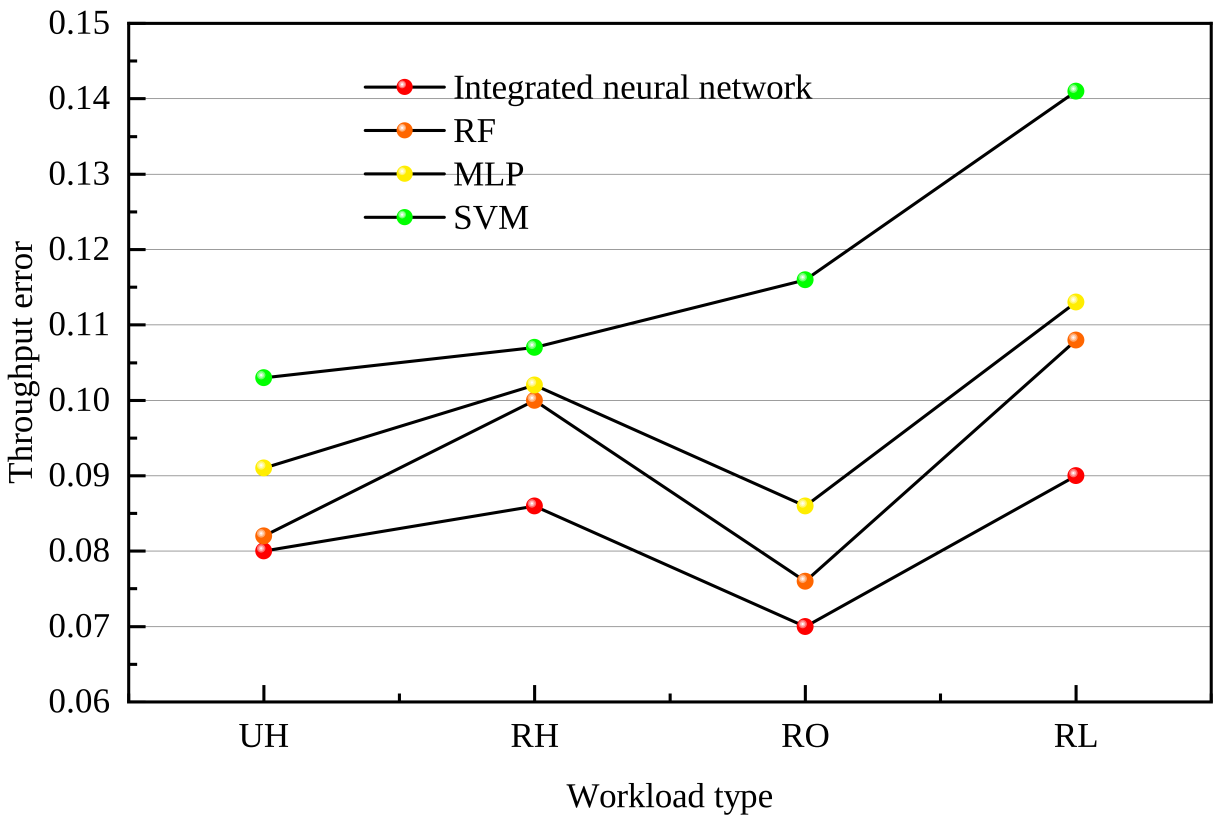

Figure 3. Error rates of throughput models for four different workloads

workloads, and similarly, the integrated neural network is analysed in comparison with different delay prediction models constructed by RF, MLP and SVM.

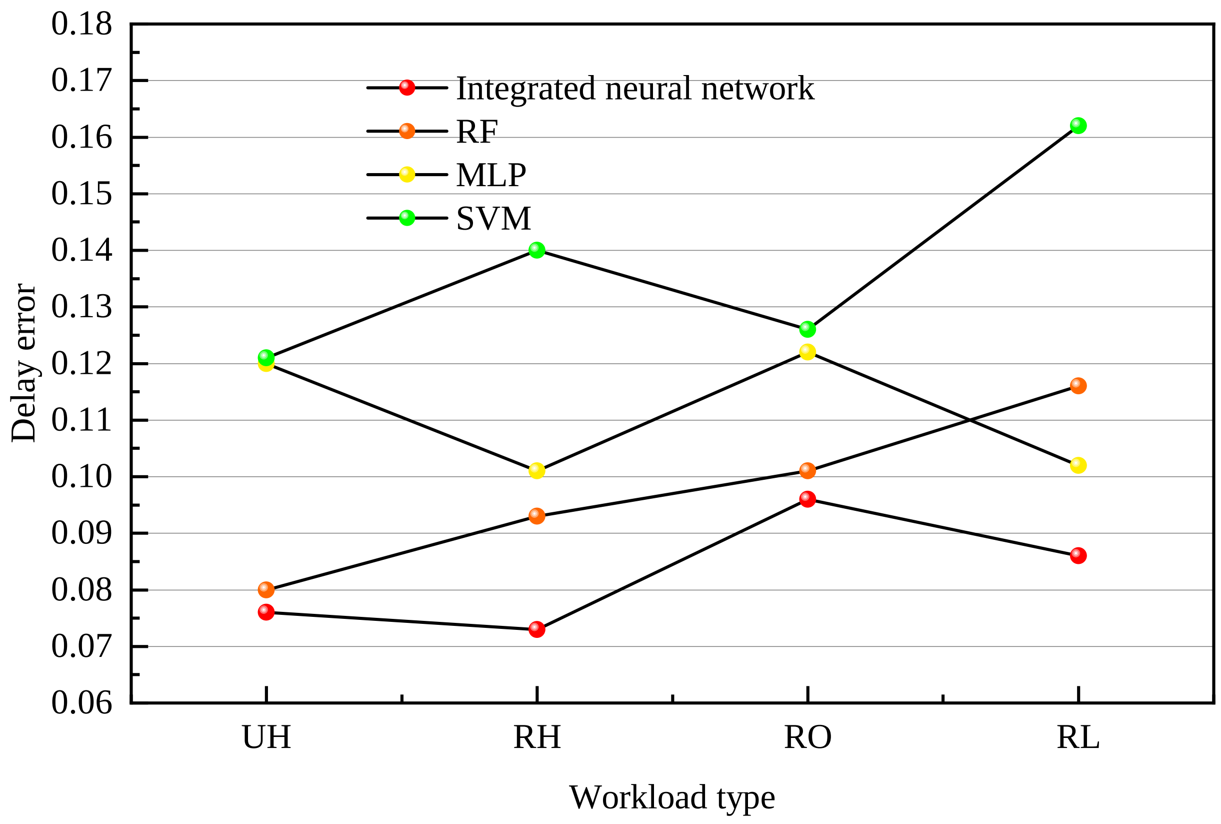

Figure 4. Error rates of delay models under four different workloads

It can be seen that for different workload conditions, the error rate of the delay prediction model constructed by the integrated neural network, the error rate of the model under the workload RO is 9.6 %, which is the largest among the four workloads.The average error rate of the model among the four workloads is 8.2 %. While the average error rate of the delay prediction models constructed by RF, MLP and SVM are 9.5 %, 10.7 % and 12.6 % respectively. Thus it can be seen that the prediction models constructed by the integrated neural network are more accurate as compared to the models constructed by the traditional machine learning methods.

5.3. **Analysis of tuning performance.** After obtaining the throughput and latency prediction models, HFOA was used to optimise the distributed storage performance and obtain the optimal solution. In order to verify the efficiency of HFOA, HFOA is analysed against FOA, PSO and GA.

This experiment takes the workload RO as an example and sets the weights of throughput and latency as $w_1 = 0$ and $w_2 = 1$, indicating that the weight of throughput in distributed performance is 0 and the weight of latency is 1. Therefore, the latency prediction model is used as the objective function of the optimisation algorithm. Four different optimisation algorithms are used to optimise the objective function, the initialised population size is uniformly set to 200, the number of iterations of the four algorithms is set to 500, and then the experiments are executed for 20 times to take the average, focusing on comparing the size and stability of the optimal solution obtained by the four optimisation algorithms. Figure 5 shows the convergence of the delayed optimal solutions obtained by the four optimisation algorithms.

HFOA has converged to the optimal value when the number of iterations reaches 150, whereas FOA and PSO converge to the optimal value at 200 iterations and GA converges

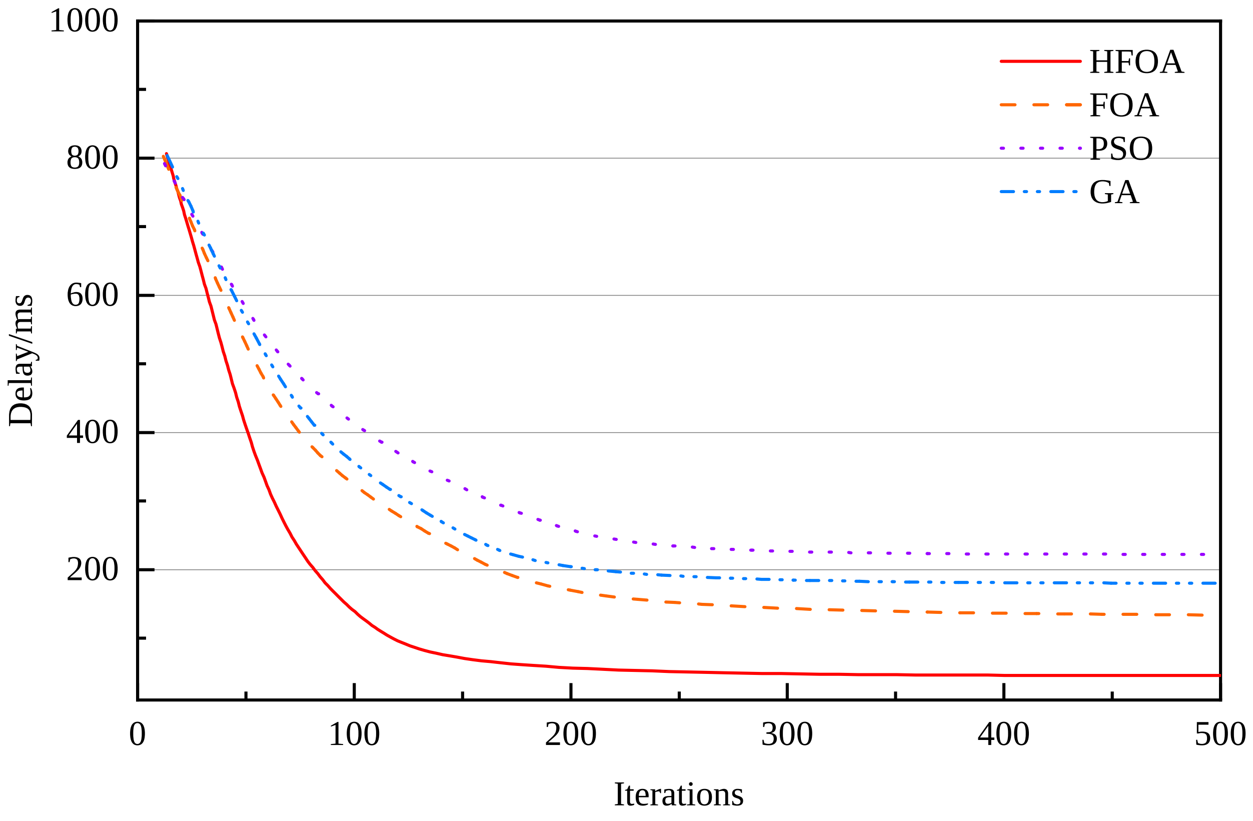

Figure 5. Convergence comparison of 4 optimisation algorithms

to the optimal solution at 250 iterations. Compared with other optimisation algorithms, HFOA converges to the smallest optimal solution, followed by FOA. It shows that the HFOA proposed in this paper not only improves in convergence speed, but also works better than other optimisation algorithms in convergence value.

6. **Conclusions.** This work proposes a performance tuning methodology applicable to distributed storage systems for legal electronic data in order to obtain the optimal parameter configuration of distributed clusters for a given device resource. A KDE-based integrated neural network model is proposed for the sample data contains unbiased noise problem. The integrated neural network model is used to train and learn from the sample data, and further feature selection is performed to obtain throughput and delay prediction models. According to the characteristics of the throughput and delay prediction models, multiple swarm co-evolution methods are used to improve the position updating method of the traditional FOA. The HFOA algorithm is used to optimise the performance of the distributed storage system, and the optimal solution and the corresponding optimal parameter configuration are obtained. The experimental results show and compare with multiple machine learning algorithms that the prediction model based on integrated neural networks has the highest accuracy. Comparative analysis with multiple optimisation algorithms shows that the HFOA-based performance tuning approach has better optimal solutions and convergence speed.

REFERENCES

[1] J. R. Brown, "Electronic Brains and the Legal Mind: Computing the Data Computer's Collision with Law," *The Yale Law Journal*, vol. 71, no. 2, pp. 239-254, 1961.

[2] H. L. Roitblat, A. Kershaw, and P. Oot, "Document categorization in legal electronic discovery: computer classification vs. manual review," *Journal of the American Society for Information Science and Technology*, vol. 61, no. 1, pp. 70-80, 2010.

[3] Y.-H. Tan, and W. Thoen, "INCAS: a legal expert system for contract terms in electronic commerce," *Decision Support Systems*, vol. 29, no. 4, pp. 389-411, 2000.

[4] A. J. Meijer, "Transparent government: Parliamentary and legal accountability in an information age," *Information Polity*, vol. 8, no. 1-2, pp. 67-78, 2003.

[5] M. R. Ghazi, and D. Gangodkar, "Hadoop, MapReduce and HDFS: a developers perspective," *Procedia Computer Science*, vol. 48, pp. 45-50, 2015.

[6] Y. Luo, S. Luo, J. Guan, and S. Zhou, "A RAMCloud storage system based on HDFS: Architecture, implementation and evaluation," *Journal of Systems and Software*, vol. 86, no. 3, pp. 744-750, 2013.

[7] I. A. T. Hashem, N. B. Anuar, A. Gani, I. Yaqoob, F. Xia, and S. U. Khan, "MapReduce: Review and open challenges," *Scientometrics*, vol. 109, pp. 389-422, 2016.

[8] S. Rajendran, O. I. Khalaf, Y. Alotaibi, and S. Alghamdi, "MapReduce-based big data classification model using feature subset selection and hyperparameter tuned deep belief network," *Scientific Reports*, vol. 11, no. 1, pp. 24138, 2021.

[9] S. Shakya, "An efficient security framework for data migration in a cloud computing environment," *Journal of Artificial Intelligence*, vol. 1, no. 01, pp. 45-53, 2019.

[10] X. Yang, X. Tao, E. Dutkiewicz, X. Huang, Y. J. Guo, and Q. Cui, "Energy-efficient distributed data storage for wireless sensor networks based on compressed sensing and network coding," *IEEE Transactions on Wireless Communications*, vol. 12, no. 10, pp. 5087-5099, 2013.

[11] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: A distributed storage system for structured data," *ACM Transactions on Computer Systems (TOCS)*, vol. 26, no. 2, pp. 1-26, 2008.

[12] A. Zarras, "Applying model-driven architecture to achieve distribution transparencies," *Information and Software Technology*, vol. 48, no. 7, pp. 498-516, 2006.

[13] Y. Zhao, Y. Cao, Y. Chen, M. Zhang, and A. Goyal, "Rake: Semantics assisted network-based tracing framework," *IEEE Transactions on Network and Service Management*, vol. 10, no. 1, pp. 3-14, 2012.

[14] C. Pham, L. Wang, B. C. Tak, S. Baset, C. Tang, Z. Kalbarczyk, and R. K. Iyer, "Failure diagnosis for distributed systems using targeted fault injection," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 2, pp. 503-516, 2016.

[15] Y. Yu, X. Li, X. Leng, L. Song, K. Bu, Y. Chen, J. Yang, L. Zhang, K. Cheng, and X. Xiao, "Fault management in software-defined networking: A survey," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 1, pp. 349-392, 2018.

[16] G. Betta, C. Liguori, and A. Pietrosanto, "Propagation of uncertainty in a discrete Fourier transform algorithm," *Measurement*, vol. 27, no. 4, pp. 231-239, 2000.

[17] T. Vieira, and J. Eisner, "Learning to prune: Exploring the frontier of fast and accurate parsing," *Transactions of the Association for Computational Linguistics*, vol. 5, pp. 263-278, 2017.

[18] Z. M. Jiang, and A. E. Hassan, "A survey on load testing of large-scale software systems," *IEEE Transactions on Software Engineering*, vol. 41, no. 11, pp. 1091-1118, 2015.

[19] Y. Gao, M. Kim, C. Thapa, A. Abuadbba, Z. Zhang, S. Camtepe, H. Kim, and S. Nepal, "Evaluation and optimization of distributed machine learning techniques for internet of things," *IEEE Transactions on Computers*, vol. 71, no. 10, pp. 2538-2552, 2021.

[20] A. Krishnan, L. J. Williams, A. R. McIntosh, and H. Abdi, "Partial Least Squares (PLS) methods for neuroimaging: a tutorial and review," *Neuroimage*, vol. 56, no. 2, pp. 455-475, 2011.

[21] R. T. Lottering, M. Govender, K. Peerbhay, and S. Lottering, "Comparing partial least squares (PLS) discriminant analysis and sparse PLS discriminant analysis in detecting and mapping Solanum mauritianum in commercial forest plantations using image texture," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 159, pp. 271-280, 2020.

[22] H. S. Dhiman, D. Deb, and J. M. Guerrero, "Hybrid machine intelligent SVR variants for wind forecasting and ramp events," *Renewable and Sustainable Energy Reviews*, vol. 108, pp. 369-379, 2019.

[23] F. Kamalov, "Kernel density estimation based sampling for imbalanced class distribution," *Information Sciences*, vol. 512, pp. 1192-1201, 2020.

[24] Y. Ma, Y. Peng, and T.-Y. Wu, "Transfer learning model for false positive reduction in lymph node detection via sparse coding and deep learning," *Journal of Intelligent & Fuzzy Systems*, vol. 43, no. 2, pp. 2121-2133, 2022.

[25] T.-Y. Wu, A. Shao, and J.-S. Pan, ”CTOA: Toward a Chaotic-Based Tumbleweed Optimization Algorithm,” *Mathematics*, vol. 11, no. 10, 2339, 2023.

[26] T.-Y. Wu, H. Li, and S.-C. Chu, ”CPPE: An Improved Phasmatodea Population Evolution Algorithm with Chaotic Maps,” *Mathematics*, vol. 11, no. 9, 1977, 2023.

[27] C.-M. Chen, Y. Gong, and J. M.-T. Wu, ”Impact of Technical Indicators and Leading Indicators on Stock Trends on the Internet of Things,” *Wireless Communications and Mobile Computing*, vol. 2022, pp. 1-15, 2022.

[28] J. L. Speiser, M. E. Miller, J. Tooze, and E. Ip, ”A comparison of random forest variable selection methods for classification prediction modeling,” *Expert Systems with Applications*, vol. 134, pp. 93-101, 2019.

[29] F. S. Gharehchopogh, M. Namazi, L. Ebrahimi, and B. Abdollahzadeh, ”Advances in sparrow search algorithm: a comprehensive survey,” *Archives of Computational Methods in Engineering*, vol. 30, no. 1, pp. 427-455, 2023.

[30] X. Yang, W. Li, L. Su, Y. Wang, and A. Yang, ”An improved evolution fruit fly optimization algorithm and its application,” *Neural Computing and Applications*, vol. 32, pp. 9897-9914, 2020.

[31] M. Shao, X. Wang, Z. Bu, X. Chen, and Y. Wang, ”Prediction of energy consumption in hotel buildings via support vector machines,” *Sustainable Cities and Society*, vol. 57, 102128, 2020.