SECURITY AND COMMUNICATION NETWORKS
Security Comm. Networks 2016; **9**:4002–4012
Published online 14 August 2016 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/sec.1582

RESEARCH ARTICLE

A speculative approach to spatial-temporal efficiency with multi-objective optimization in a heterogeneous cloud environment

Qi Liu[1], Weidong Cai[1], Jian Shen[2], Zhangjie Fu[3]*, Xiaodong Liu[4] and Nigel Linge[5]

[1] School of Computer and Software, Nanjing University of Information Science and Technology, Nanjing, China
[2] Jiangsu Engineering Centre for Network Monitoring, Nanjing University of Information Science and Technology, Nanjing, China
[3] Jiangsu Collaborative Innovation Center of Atmospheric Environment and Equipment Technology (CICAEET), Nanjing University of Information Science & Technology, Nanjing, China
[4] School of Computing, Edinburgh Napier University, 10 Colinton Road, Edinburgh EH10 5DT, U.K.
[5] School of Computing Science and Engineering, University of Salford, Salford, U.K.

ABSTRACT

A heterogeneous cloud system, for example, a Hadoop 2.6.0 platform, provides distributed but cohesive services with rich features on large-scale management, reliability, and error tolerance. As big data processing is concerned, newly built cloud clusters meet the challenges of performance optimization focusing on faster task execution and more efficient usage of computing resources. Presently proposed approaches concentrate on temporal improvement, that is, shortening MapReduce time, but seldom focus on storage occupation; however, unbalanced cloud storage strategies could exhaust those nodes with heavy MapReduce cycles and further challenge the security and stability of the entire cluster. In this paper, an adaptive method is presented aiming at spatial–temporal efficiency in a heterogeneous cloud environment. A prediction model based on an optimized Kernel-based Extreme Learning Machine algorithm is proposed for faster forecast of job execution duration and space occupation, which consequently facilitates the process of task scheduling through a multi-objective algorithm called time and space optimized NSGA-II (TS-NSGA-II). Experiment results have shown that compared with the original load-balancing scheme, our approach can save approximate 47–55 s averagely on each task execution. Simultaneously, 1.254 ‰ of differences on hard disk occupation were made among all scheduled reducers, which achieves 26.6% improvement over the original scheme. Copyright © 2016 John Wiley & Sons, Ltd.

KEYWORDS
MapReduce; cloud storage; load balancing; multi-objective optimization; prediction model

***Correspondence**
Zhangjie Fu, Jiangsu Collaborative Innovation Center of Atmospheric Environment and Equipment Technology (CICAEET), Nanjing University of Information Science & Technology, Nanjing, China.
E-mail: wwwfzj@126.com

1. INTRODUCTION

In recent years, distributed computing has been widely investigated and deployed in both academic and industrial fields because of its features of large-scale virtualization, failure control among connected components, and asynchronized communication. Cloud computing as one of the successful commercial distributed systems provides users with on-demand services by allocating rational computing and storage resources transparently [1,2].

MapReduce paradigm proposed by Google is being exploited by a fast growing number of companies and research institutes [3]. Hadoop, as a type of open-source implementation provided by Apache, gives them a good chance to conduct efficient big data processing and discover potential and valuable information in a non-traditional way. Enterprises and companies therefore benefit from analyzing and dealing with real-time data. At the moment, data analysis applications in a cloud have shown different complexity, resource requirements, and data delivery deadlines; such diversity has created new requirements of job scheduling, workload management, and program design in a cloud. Several projects have been launched to reduce challenges on writing complex programs for data analysis and/or data mining, for example, Pig [4] built upon the MapReduce engine in the Hadoop environment. In addition, HBase [5] and Hive [6], implemented by Apache, are widely used in a cloud environment to achieve better performance. In these applications, however, low-level improvement based on MapReduce is still

required because of its direct interaction with Hadoop Distributed File System (HDFS) [7]. An outstanding strategy that improves the security and stability of a cloud system is necessary.

While the optimization of job scheduling in MapReduce has been widely conducted in recent activities [8–20], current Hadoop systems still suffer from poor load-scheduling strategies because of their lack of consideration on the usage of cloud storage, which would bring heavy loads on certain data nodes and therefore cause a long delay on total execution. Although theoretically infinite computing resources can be provided in a cloud system, unreasonable increment of mappers/reducers cannot achieve processing efficiency and even waste more storage to complete.

A scheme is therefore presented in this paper to achieve process efficiency and load balance in a cloud system both spatially and temporally. Our contributions are three folds as follows:

(1) A prediction model called PMK-ELM is firstly built providing prediction on the number of reducers needed for newly coming tasks, as well as possible execution duration and storage size they may take.
(2) An optimized algorithm based on NSGA-II [21] called TS-NSGA-II is then designed to maintain such an equalized status that the total time completing the job distributed in each reducer is almost the same while keeping the ratio of hard disk space similar.
(3) A practical Hadoop environment is constructed to verify the feasibility and performance of the scheme.

The remainder of this paper is organized into five sections. Related work on load balancing is reviewed in Section 2. In Section 3, preliminaries of core algorithms manipulated in our approach are introduced. Section 4 explains the adaptive method to achieve fair loads during map and reduce processes. Results are presented and evaluated in Section 5 with comparison of corresponding algorithms. Finally, Section 6 concludes the paper and identifies potential future work.

2. RELATED WORK

Because of the load deflection of reduce jobs, it balanced load is hard to be achieved in datanodes of a MapReduce system of a MapReduce system [8]. An optimized method was presented in [9] to repartition map and reduce tasks in a low-specification data node to a faster one by monitoring running map and reduce tasks to ensure that all available datanodes can complete their missions at the same time. This method can handle all kinds of load deflection, but Hadoop has been greatly changed with complicated implementation. In addition, additional redistribution of tasks has brought extra network costs. There were other research efforts conducted on optimizing partition of reducers, for example, based on historical data [10] and sampling results [11] to facilitate reduce partition processes. These methods can achieve load balancing dynamically, but neither of them was verified in a real Hadoop system.

Offline/online profiling was proposed to predict application resource requirements using a benchmark or real application workloads. A prediction model based on Support Vector Machine (SVM), for instance, was presented in a heterogeneous environment [12]. An adaptive algorithm called heterogeneity-aware partitioning (HAP) was designed for managing distribution of tasks based on estimated work thresholds. However, input data of a reduce task need to be divided into some splits in the HAP. The progress causes extra time when merging the splits and occupies more storage space in data nodes as they are stored as a block which is a storage unit of HDFS. Additionally, the training stage of the HAP takes a large amount of time.

Deployment efficiency on virtualization has also been investigated. A general approach was introduced in [13] to estimate the resource requirements of applications running in a virtualized environment. Different types of virtualization overheads were profiled, so a regression-based model was built to map native system profiles into a virtualized system. In [14], dynamic demands of resources when starting a new Virtual Machine (VM) instance were studied so that a prediction model was presented for adaptive resource provision in a cloud. General solutions for performance evaluation and load efficiency in a cloud system have been examined and presented. PQR2, an approach to accurate performance evaluation of distributed application in a cloud, was presented [15]. A model for predicting resource consumption of MapReduce processes was set up based on a classification and regression tree [16].

Besides the previous methods, some researchers are studying optimizing the speculative execution strategy in MapReduce. Zaharia *et al.* proposed a modified version of speculative execution called Longest Approximate Time to End (LATE) algorithm that uses a different metric to schedule tasks for speculative execution [17]. Instead of considering the progress made by a task so far, it computes the estimated time directly. At the same time, LATE gives a more clear assessment of struggling tasks' impacts on the overall job running time. But the time that every stage occupies is not stable, and the standard error used in LATE cannot represent all cases. To solve the disadvantages in LATE, MCP [18] was proposed. MCP uses average progress rate to identify slow tasks; while in reality, the progress rate can be unstable and misleading. Struggles can be appropriately judged in homogeneous environments. There are still a lot of disadvantages in MCP, including average progress rate and its mediocre performance in heterogeneous environments.

Data placement schemes have also been researched. To address this problem, a new Data-gRouping-AWare data placement scheme was proposed in [19]. It extracts optimal data groupings and re-organizes data layouts to achieve load balancing in per group. CoHadoop was proposed in [20]; it permits applications to decide where data should be stored. However, these schemes are aimed at the data

placement when storing the data and not fit for MapReduce. Furthermore, they cannot be applied when data have been stored.

Comprehensive load and usage efficiency have achieved large improvements in a distributed environment. However, it is still challenging to achieve spatial–temporal efficiency in a cloud system, especially in a heterogeneous one.

3. PRELIMINARIES

A detailed introduction about some advanced techniques used in this paper is given in this section.

3.1. MapReduce

In MapReduce, computation works are implemented through map tasks and reduce tasks. Map tasks put different pairs of data into multiple lists grouped by different keys. So data having same key are distributed into a same list. Then, results generated by map tasks, as intermediate data, are pulled by reduce tasks to process further and obtain the final result [22].

MapReduce jobs are divided into multiple tasks; then, these generated tasks are distributed to nodes and executed in the cluster. The map stage is partitioned into map tasks according to a logical split of input data that generally resides on HDFS [23]. Reduce tasks are produced according to an equation in reduce stage. The map task reads the data from HDFS as input data; map functions designed by user are then applied and put the results into buffers. These data are written to the memory of the node executing the map task when it is less than the threshold user set. Otherwise, this data will be spilled into hard disk of the nodes. There are three phases in reduce tasks, called shuffle (copy), sort (merge), and reduce. In the shuffle phase, the reduce tasks pull the intermediate data files from the already completed map tasks. In the sort phase, the intermediate files from all the map tasks are sorted. After all the intermediate data are shuffled, a final pass is made to merge all these sorted files. Thus, the shuffle and sort phases are interleaved, and in this paper, we combine these activities under the shuffle phase.

Finally, in the reduce phase, the sorted intermediate data are passed to the user-defined reduce function. The output from the reduce function is generally written back to HDFS. Job scheduling in Hadoop is performed by name node, which manages a number of datanodes in the cluster. In MapRedeuce 2.x, each datanode will prepare containers for map tasks and reduce tasks, which can be seen as an abstraction of resource and used to execute task. The number of map and reduce container is calculated at the configuration file. Application Master periodically checks the heartbeats coming from datanodes and calculates the reported state of free resources and current progress of tasks that they are currently executed.

3.2. Basic ELM

Recently, artificial neural networks have been widely applied in applications involving classification or function approximation [24]. However, they also suffer from low learning speed, which has become the main bottleneck when applying an artificial neural network algorithm to practical applications. In order to overcome this drawback, many researchers explore the approximation capability of feedforward neural networks, especially in a limited training set, from the point of view of mathematics. A novel machine learning algorithm called extreme learning machine (ELM) [25,26] was therefore designed based on single-hidden layer feedforward neural networks (SLFNs) [27].

Let $X = \{x_1, x_2, \ldots, x_N | x_i \in R^D, i = 1, 2, \ldots, N\}$ denote the training set with N samples and D represent dimension. Let $Y = \{y_1, y_2, \ldots, y_N | y_i \in R\}$ denote the vectorised label where column $j(\{j = 1, 2, \ldots, P\})$ set by 1 for class j while other columns set by 0, and P is the number of classes. Then, the model of a single-hidden layer neural network having L hidden neurons and an activation function $g(x)$ can be expressed as

$$\sum_{j=1}^{L} \beta_j \cdot g\left(< w_j, x_i > + b_j\right) = y_i \tag{1}$$

where $i = 1, 2, \ldots, N$, w_j, and β_j represent the weight vectors from inputs to hidden layer and from hidden layer to output layer, respectively, b_j is the bias of jth hidden neuron, and $g\left(< w_j, x_i > + b_j\right)$ is the output of the jth hidden neuron with respect to the input sample x_i . Note that (1) can be rewritten in a compact form as

$$H \cdot \beta = Y' \tag{2}$$

where H is the hidden layer output matrix of SLFNs, β is the output weight matrix, and Y' is the transpose of Y. Optimal weights and bias of SLFNs can be found by using back propagation learning algorithms, which requires users to specify learning rates and momentum. However, there is no guarantee that the global minimum error rate can be found. Thus, the learning algorithm suffers from local minima and over-training. In exploration of the approximation capability of feedforward neural networks in a finite training set, it is found that SLFNs can reach the approximate capacity at a specified error. $\varepsilon(\varepsilon > 0)$ level with the hidden layer neurons is much less than the number of training samples. And based on the minimum norm least-squares function, the weight matrix β in (2) can be solved by

$$\beta = H^+ \cdot Y \tag{3}$$

where H^+ is a MooreCPenrose matrix generalized inverse of matrix H.

3.3. K-ELM

Kernel-based Extreme Learning Machine (K-ELM) has simplified the complexity of the ELM algorithm, with improvement of the operation speed. Meanwhile, it improves the simulation precision of the algorithm and the

DOI: 10.1002/sec

fitting ability based on the original ELM algorithm. In K-ELM, a positive number is added to the diagonal of H^TH or HH^T, which makes the ELM algorithm more stable and presents a better generalization performance [28,29]. The prediction model established based on the training set can be described as follows:

Minimum value:

$$L_{P_{ELM}} = \frac{1}{2} \parallel \beta \parallel^2 + \frac{1}{2} C \sum_{i=1}^{N} \parallel \xi_i \parallel^2 \tag{4}$$

Constraint:

$$h(x_i)\beta = y_i^T - \xi_i^T, i = 1, 2, \ldots, N \tag{5}$$

where $\beta = [\beta_1, \beta_2, \ldots, \beta_L]$ is the weight of the hidden layer outputs. C is the ridge regression parameter. ζ_i is the error vector between expected outputs and training outputs. $h(x_i)$ is output vector of hidden neurons corresponding to the training sample x_i. Finally, the output function of ELM regression can be expressed as

$$\begin{aligned} f(x) &= h(x)H^T \left(\frac{I}{C} + HH^T \right)^{-1} T \\ &= \begin{pmatrix} K(x, x_1) \\ \vdots \\ K(x, x_N) \end{pmatrix} \left(\frac{I}{C} + \Omega_{ELM} \right)^{-1} T \end{aligned} . \tag{6}$$

Similar to SVM, nuclear ELM (or kernel-based ELM, K-ELM) is not required to set the number of neurons in the hidden layer and the activation function types. Common kernel functions are shown as follows.

Linear: $K(x_i, x_j) = x_i \cdot x_j$
Polynomial: $K(x_i, x_j) = (x_i \cdot x_j + b)^d, b \geq 0$
RBF: $K(x_i, x_j) = exp\left(-\sigma \| x_i - x_j \|^2\right), \sigma > 0$
Sigmoid: $K(x_i, x_j) = \tan(ax_i \cdot x_j + b), a > 0, b < 0$

3.4. NSGA-II

NSGA-II as one of the multi-objects optimization algorithms has lots of operations that are the same as those in Genetic Algorithm (GA). For example, in NSGA-II, the population undergoes initialization, crossover, and mutation as usual. However, there are three main differences:

(1) each chromosome is sorted based on non-domination sorting into a front to obtain a fitness value;
(2) crowding distance used to measure the diversity of the population is employed to decide the distance between individuals; and
(3) the population with the current population and current offspring (obtained by crossover and mutation) is sorted again based on the rank and the crowding distance.

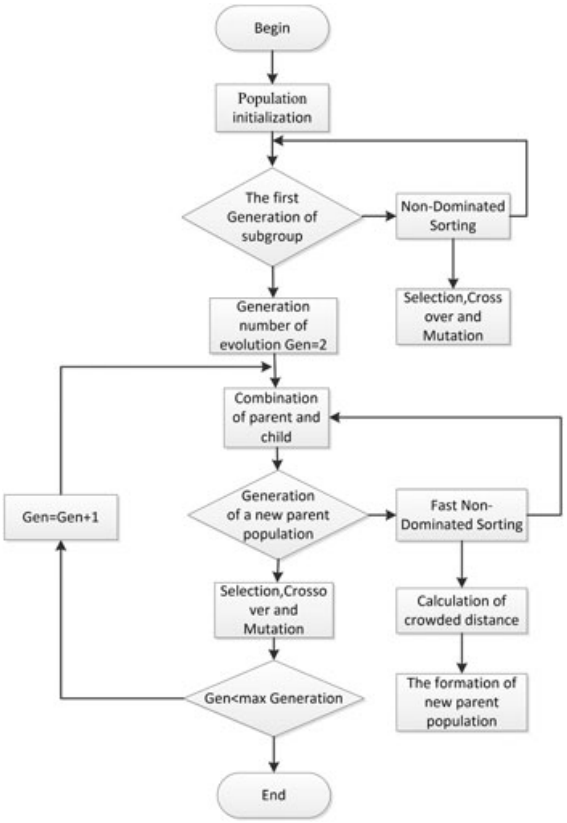

Figure 1. Flow chart of NSGA-II.

After that, the best N (population size) individuals are selected to be the next generation. The main consideration in the design of the NSGA-II algorithm consists of six aspects, involving code generation, determination of the initial population, fitness evaluation, selection, crossover, and mutation. Detailed procedure is shown in Figure 1.

4. APPROACH TO LOAD BALANCING

4.1. A method for partition reconstruction

MapReduce uses a hash function as the original partition function, where data chunks are generated and distributed to different reducers. The original hash function may lead to sever load skew, especially in a heterogeneous environment, which will decrease the speed of some node. However, the overall job finishing time is decided by the node that finishes the task at last according to the wooden barrel effect. Algorithm1 depicts the way that fairly equal size of data chunks is ensured for distribution, which helps the system dispense different volume of data to different node having different computing capacity. Before starting the work, we run the WordCount application on each node separately to obtain the approximate capacity of each node. Then, volume of data is given away according to different

Algorithm 1 Partition reconstruction

Input:
The input size of reduce stage, *size*;
The number of data chunks , *number*;
Output: *partion_list*
Get the list of capacity of each server *Lc*
Set *iterator* = 1
for *iterator* < *number* ,*iterator* + + **do**
Get the *ratio_list* according to *ratio_list* = *capacity*/*avg_capacity*
Maxr = *Max*(*ratio_list*)
Minr = *Min*(*ratio_list*)
if *Maxr*/*Minr* > 1.5 **then**
Maxr = *minr* = (*Maxr* + *Minr*)/2
Add *Minr* and *Maxr* to *ratio_list*
else
Break
end if
end for
partion_list = *size***ratio_list*
return *partion_list*

capacity. According to algorithm 1, the list of the partition has a relatively balanced data amount according to different capacities.

4.2. A prediction model for load balancing based on K-ELM

In this section, the training set is set as: $TS = \{time, reducer\_no, datanode\_no, input\_size, shuffle\_size\}$, where *reducer_no* represents the reducer number; actually, it also indicates the sequence when reducers run. *datanode_no* represents the number of a datanode. Generally, a datanode can be mapped to several reducers. Here, *input_size* does not represent the input size of the whole task but the input size of reducers at the reduce stage. *shuffle_size* denotes the data size of a reducer that needs to shuffle when map processes have finished.

In details, the building progress of prediction model for execution time based on K-ELM (PMK-ELM) is as follows:

Step 1: Data pre-processing. First, samples that contain great network congestion are removed. Then the trimmed datasets are divided into training samples and test samples. The training samples are used for training the prediction model, whereas the test ones are for checking if the prediction model has been well trained.

Step 2: Model training. To build the PMK-ELM, training parameters of the model are obtained by using the training set sample generated by Step 1. The specific processes are as follows:

(1) No need to set number of hidden layer neurons, just randomly generated weights between the input layer and the hidden layer, and between the hidden layer neurons w and the threshold value b;
(2) Use the hidden layer neuron activation function to calculate the hidden layer output matrix H; and
(3) Work out output layer weights.

Step 3: Data validation. Datasets generated by Step 1 are used to validate the PMK-ELM algorithm. According to the parameters trained in Step 2, the predictive values of test sets can be retrieved, which are then compared with the actual values to verify the prediction performance of the model.

4.3. TS-NSGA-II

4.3.1. Mathematical model.

When a map task is completed, the data will be shuffled and merged and then assigned to different reducers; however, the amount of data assigned to each reducer is not equal, which consequently causes uneven allocation of reducers to datanodes. In order to make reduce tasks consume less time and hard disk space occupation, the following conditions should be satisfied:

(1) The data amount handled by a reducer assigned to a datanode cannot be more than disk usage of the datanode; and
(2) A reducer can only be assigned to a datanode, but a datanode can handle multiple reducers, as in Figure 2.

Although an actual reduce process is parallel, it is assumed in a virtual serialization line. A datanode called F is further abstracted so that when the procedure arrives at F, the reduce task is completed, as shown in Figure 3.

Assuming that the output of map tasks can be randomly divided into m data chunks, and there are n datanodes in the clusters. If t_{mn} represents the execution time that each reducer needs, then the execution time of each split can be noted as a matrix M_t, as shown:

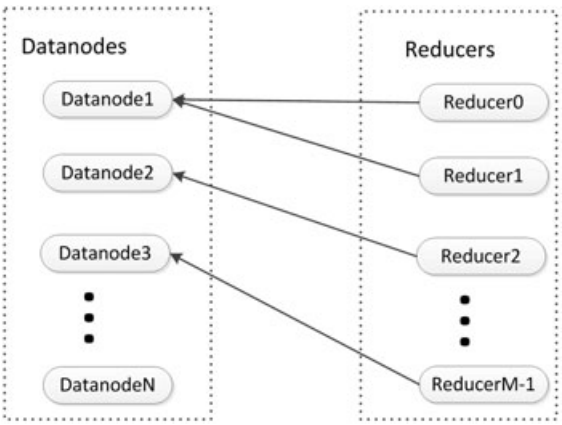

Figure 2. Relationship between datanodes and reducers.

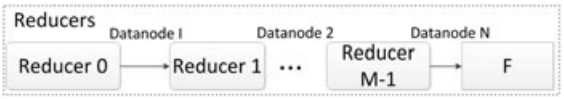

Figure 3. Virtual serialization.

$$Mt = \begin{pmatrix} t_{11} & \cdots & t_{1n} \\ \vdots & \ddots & \vdots \\ t_{m1} & \cdots & t_{mn} \end{pmatrix}$$

In order to evaluate the usage of storage space, the percentage of input size S_{mn} from total unused size sl_{mn} is calculated and noted as ps_{mn}.

$$ps_{mn} = s_{mn}/sl_{mn} \tag{7}$$

Then the hard disk space ratio of each split can be described as M_s:

$$Ms = \begin{pmatrix} ps_{11} & \cdots & ps_{1n} \\ \vdots & \ddots & \vdots \\ ps_{m1} & \cdots & ps_{mn} \end{pmatrix}$$

Finally, the elements of M_t and M_s are combined to format a new matrix M with new elements expressed as $(t, ps)_{mn}$, as shown:

$$M = \begin{pmatrix} (t,ps)_{11} & \cdots & (t,ps)_{1n} \\ \vdots & \ddots & \vdots \\ (t,ps)_{m1} & \cdots & (t,ps)_{mn} \end{pmatrix}$$

The real execution time of datanode i can be described as t_i, whereas the split size can be represented as S_i. Accordingly, the real processing results list L can be calculated as follows:

$$L = \{(t,ps)_1, (t,ps)_2, \ldots, (t,ps)_n\}$$

Here, two objective functions can be formatted as shown in (8) and (9), whereas the constraints are shown as (10) and (11); where in (10), *InSum* represents the total sum of reduce input size.

$$\min T = \sum_{i=1}^{n} \left| \frac{\bar{t} - t_i}{\bar{t}} \right| \tag{8}$$

$$\min S = \sum_{i=1}^{n} \left| ps_i - \overline{ps} \right| \tag{9}$$

$$InSum = \sum_{i=1}^{n} s_i \tag{10}$$

$$t_i > 0, ps_i > 0. \tag{11}$$

4.3.2. Design of TS-NSGA-II.

The design of algorithm consists of six aspects, including determination of the initial population, fitness evaluation, selection, mutation, code generation, and crossover. Major changes have been made on the latter two.

(1) Code generation

Non-negative integers are used as the index of reducers, that is, $0, 1, 2, \ldots, M-1$ for M reducers, however. On the other hand, N datanodes are indexed using positive integers, that is, $1, 2, \ldots, N$. In this case, distribution of M reducers to N datanodes may generate N^M possible combinations.

(2) Crossover

The original NSGA-II algorithm uses simulated binary crossover [19] in this stage; however, in our scheme, crossover probability called pc is used for better grouping after being selected. The crossover stage in this scheme consists of two steps:

(1) Randomly match a group of chromosomes;
(2) During matching chromosomes, randomly set intersections to make matched individual chromosomes exchange their information.

Chromosome should always be kept permutations, so the procedure of crossover is as follows: after randomly selecting paired chromosomes, two crossover positions are randomly generated; the cross section of elements on the other side of the parent is also removed. Then, the new cross section is added to the sequence of the parent that has cut out some of the elements, taking two pairs of chromosomes as an example, where chromosome A= 2313|1122|32 and chromosome B = 3123|2213|12. The cross section is divided by a vertical bar. First, the element corresponding to |1122| of A is removed from B, so B' = 312312; then a gene fragment of A is added to B, so the offspring B" is 3123|1211|22. Similarly, the offspring A" is 2313|3222|13. For new produce offsprings A" and B", it needs to be decided whether the total data size is bigger than the storage quota. If not, they are regarded as effective; otherwise, iteration will be operated. The complete procedure of the algorithm is shown in Algorithm 2 as follow:

5. EXPERIMENT AND ANALYSIS

In order to test the performance and benefits of the load balancing scheme, a practical heterologous cloud testing environment was implemented, which consists of a desktop computer and a server. The server has 288 GB of memory and 10 TB of SATA hard disks. The desktop contains 12 GB of memory, a single 500 GB disk, and a Core 2 Quad processor. Eight virtual machines were created in the server with different amounts of memory and number of shared processors. The detailed information is shown in Table I.

Algorithm 2 Crossover

Input:
The list of chromosomes, L_i;
Crossover probability, p_c;
The hard disk space ratio of each split, PS_{mn};
Output: New list of chromosomes, *NewLi*
Randomly match a group of individual in *Li* according to *pc* noted as *A* and *B*
while true **do**
Randomly generate two number not larger than the length of A, described as m, $n(m <= n)$
Divide *A* into 3 parts:$SeqAm, SeqAc, SeqAn$
Do the same operator to *B*
Get $SeqBm, SeqBc, SeqBn$
$A' = SeqAm \cup SeqAn, B' = SeqAm \cup SeqAn$
$A'' = A' \cup SeqBc, B'' = B' \cup SeqAc$
Get the *ps* according to ps_{mn}
if *ps* is smaller than 1 **then**
Break
else
Continue
end if
end while
Replace *A* with A'' and *B* with B'' in *NewLi*
return *NewLi*

Table I. The detailed information of each virtual machine.

NodeId	Memory (GB)	Core processors
Node1	10	8
Node2	8	4
Node3	8	1
Node4	8	8
Node5	4	8
Node6	4	4
Node7	18	4
Node8	12	8

K-means (KM) and WordCount algorithms were manipulated to evaluate the performance of load scheme. The Purdue MapReduce Benchmarks Suite provides us with the KM clustering workload, where 26 GB of free datasets and a free datasets of 50 GB in WordCount clustering workload [30] were selected as the inputs. All our test applications were built based on Hadoop 2.6.0. According to the Apache Hadoop documents, mapreduce.tasktracker.reduce.tasks.maximum has been set as 1.

Overall testing processes were conducted in three stages.

(1) Dataset collection. A Hadoop analysis tool was implemented to obtain historical data.
(2) Execution time prediction. The PMK-ELM was enabled to predict the execution time of next reduce tasks.
(3) Load balancing. The core shuffle and MRContainerAlllocator class were modified in the Hadoop system to enable TS-NSGA-II.

Table II. Experiment parameters.

	Dataset size (pieces)	Training dataset size (pieces)	Testing dataset size (pieces)
K-means	910	800	110
WordCount	800	700	100

Table III. The best parameters generated by GA.

	K-means		WordCount	
	PMK-ELM	PM-SVM	PMK-ELM	PM-SVM
C	15.838	–	20.521	–
σ	0.069	–	0.867	–
b	–	2.285	–	6.961
p	–	41.967	–	16.583
MAPE	10.05%	10.60%	12.64%	13.42%

Table IV. The performance comparison between PMK-ELM and PM-SVM.

		Training time (s)	Testing time (s)
K-means	PMK-ELM	0.055	0.004
	PM-SVM	4.462	0.250
WordCount	PMK-ELM	0.043	0.03
	PM-SVM	3.324	0.307

5.1. Evaluation of PMK-ELM

To evaluate the performance of PMK-ELM, different input sizes and different numbers of reducers were tested during experiments, as depicted in Table II. SVM (PM-SVM) proposed in [12] was also replicated in the testing environment for comparison purposes. A log analysis tool was developed to collect training and test sets.

A GA was employed to generate the parameters that PM-SVM and PMK-ELM need. In the experiments, *max_gen* was set as 200, and the range of C and b was from 0 to 1000. σ and p were set between 0 and 100. The size of population was set as 50. The results generated by GA are shown in Table III. *MAPE* was used to evaluate the results, same as the method mentioned in [12].

In the Table IV, the results are the average value after having run for 50 times. The training time of PMK-ELM is almost 80 times shorter than PM-SVM. Moreover, for both group, the test time of PMK-ELM is about 80 times shorter than PM-SVM. Besides, the accuracy of PMK-ELM is higher than PM-SVM, too.

In Figures 4, 5, 6, and 7, the detailed results of PMK-ELM and PM-SVM are depicted. In Figures 4 and 5, the line of PMK-ELM lays more closely to the real value than that of PM-SVM in two groups. On the peaks, this phenomenon is more apparent in both pictures. Although values predicted by PMK-ELM are not very accurate under some circumstance, accuracy of PMK-ELM is relatively higher compared with PM-SVM. In Figures 6 and 7, the

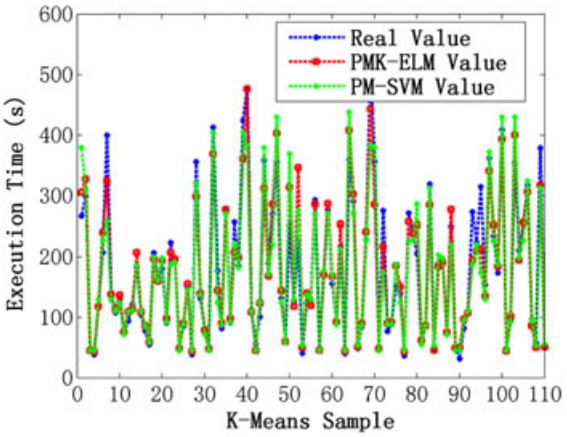

Figure 4. Comparison between PMK-ELM and PM-SVM in execution time of K-Means.

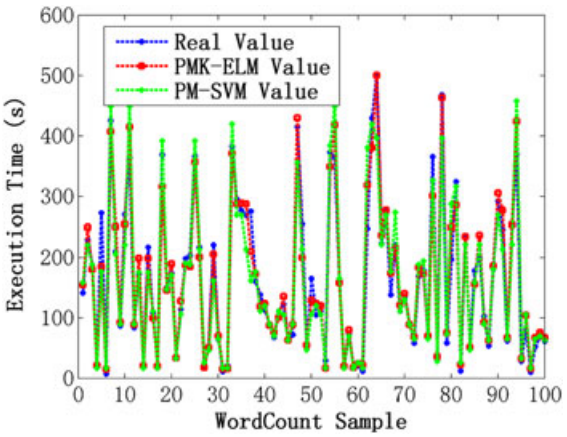

Figure 5. Comparison between PMK-ELM and PM-SVM in execution time of WordCount.

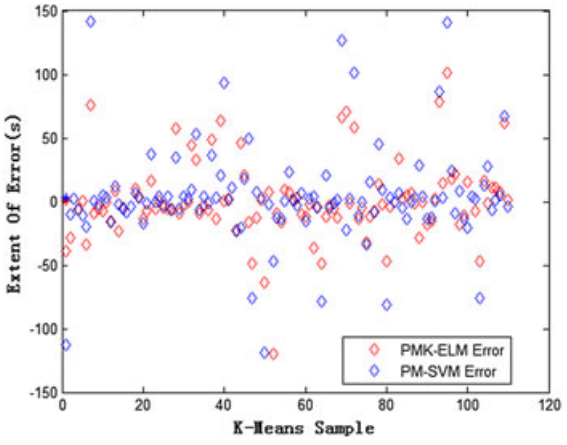

Figure 6. Distribution of error of K-Means.

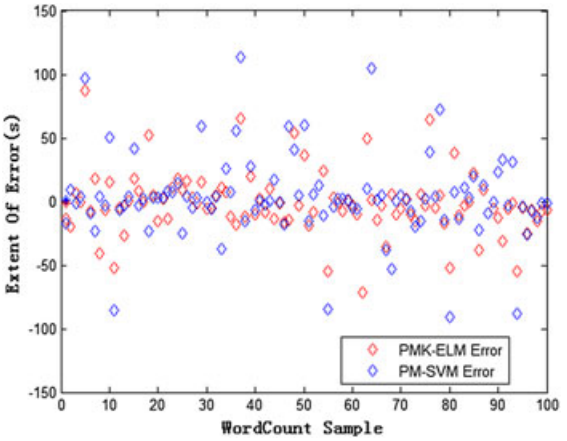

Figure 7. Distribution of error of WordCount.

errors of PMK-ELM are distributed near zero intensively, while PM-SVM shows separate distribution. Trend shown in these pictures in consistent with that shown in Figures 4 and 5 shows the performance of the PMK-ELM is better than PM-SVM. Furthermore, when the training time and test time are taken into consideration, PMK-ELM is obviously a better choice.

5.2. The performance of proposed load balancing scheme

In this section, the KM experiment is firstly run once with its execution time and hard disk space recorded. Corresponding results are shown in Tables V and VI.

Table V. Hard disk space change with original Hadoop settings.

NodeId	Before Execution(GB)	After Execution(GB)
Node1	405.16	403.08
Node2	406.79	404.69
Node3	404.82	402.75
Node4	412.36	410.23
Node5	405.09	402.83
Node6	413.44	411.32
Node7	404.71	404.71
Node8	404.51	402.11

Table VI. Execution time of different reducers.

NodeId	Reducer group	Reducer execution time (s)
Node1	Reducer0	196
Node2	Reducer5	199
Node3	Reducer1	227
Node4	Reducer4	226
Node5	Reducer3	240
Node6	Reducer6	269
Node7	–	–
Node8	Reducer2	181

From Tables V and VI, we can see that Reducer3 and Reducer6 was consumed when executing the task, so the overall execution time is decided by the longest time. In Table VI, Node1 did not take part in the task, which has a better performance and may help the overall task finish earlier.

Then, we deleted the results generated by the application and applied PMK-ELM and TS-NSGA-II to this application, and we obtained a better performance. The points shown in Figures 8 and 9 are all the feasible solutions created by our scheme in two groups of experiments. Our scheme randomly chooses a group of solutions form each group, one is group A= $\{1, 4, 6, 2, 8, 5, 3\}$, which represents assigning reducer0 to datanode1, reducer1 to datanode4, and so on, the other is group B= $\{1, 5, 6, 4, 7, 8, 3\}$. The benefits we obtained are shown in Figures 8 and 9 and Tables VII, VIII, and IX.

As shown in Figure 10, the maxim reducer execution time of groups A and B is shorter than the original group, which determines the groups A and B finish the reduce stage faster than the original. The results shown in Table VI also prove it. Not only does our load balancing scheme make the application run faster but also helps the hard disk occupation more reasonable. Table VIII shows the hard disk occupation when PMK-ELM and TS-NSGA-II are applied. S in Table IX is an evaluation parameter that has been described in Section 4, which also shows our scheme has a better performance in job execution time.

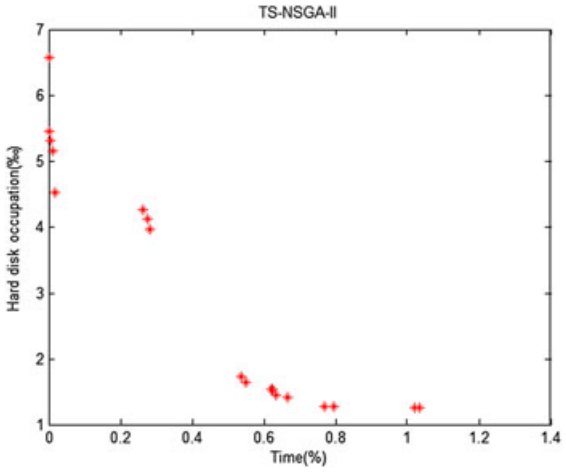

Figure 8. Results of Group A.

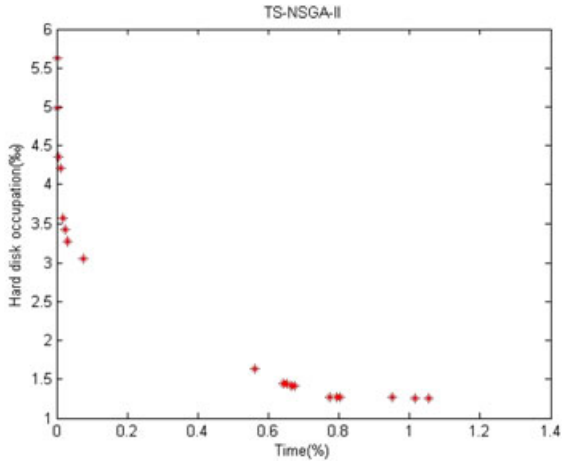

Figure 9. Results of Group B.

Table VII. Hard disk space change with original Hadoop settings.

NodeId	Before execution (GB)	After execution (GB) A	B
Node1	405.16	403.08	403.08
Node2	406.79	404.53	406.79
Node3	404.82	402.70	402.70
Node4	412.36	410.29	408.03
Node5	405.09	402.99	403.03
Node6	413.44	411.05	411.05
Node7	404.71	404.71	402.58
Node8	404.51	402.38	402.41

Table VIII. Comparison between original and optimize schemes in disk balancing (S).

	Original	A	B
S(‰)	1.709	1.415	1.125

Table IX. The overall execution time change with PMK-ELM and TS-NSGA-II.

	Original (s)	A (s)	B (s)
Overall job Execution time	615	560	568

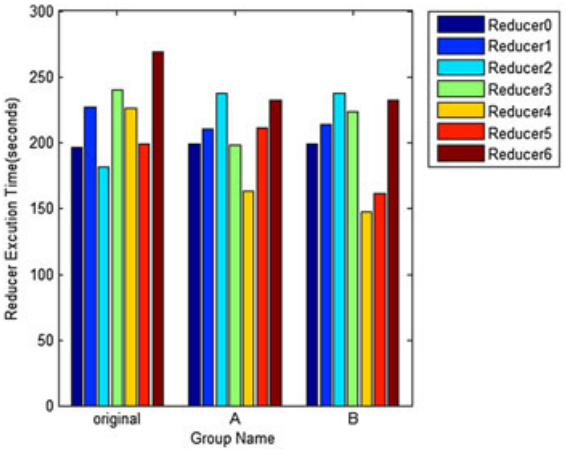

Figure 10. Comparison between original and optimized schemes in reducer execution time.

DOI: 10.1002/sec

6. CONCLUSIONS

In this paper, an adaptive approach is proposed combined with a prediction model, PMK-ELM, and a multi-object selective algorithm, TS-NSGA-II. The PMK-ELM can help facilitate the prediction of execution time of tasks, whereas the TS-NSGA-II is designed to facilitate the selection of suitable number of reducers. The experiment results have shown that both models achieve a good performance. About 47–55 s have been saved during experiments. In terms of storage efficiency, only 1.254‰of differences on hard disk occupation were made among all scheduled reducers, which achieves 26.6% improvement than the original scheme. In the future, we would like to optimize the speculative strategy in MapReduce and try to improve the performance of the strategy.

ACKNOWLEDGEMENTS

This work is supported by the NSFC (61300238, 61300237, 61232016, 1405254, 61373133), Marie Curie Fellowship (701697-CAR-MSCA-IFEF-ST), the 2014 Project of six personnel in Jiangsu Province under Grant No. 2014-WLW-013, the 2015 Project of six personnel in Jiangsu Province under grant no. R2015L06, Basic Research Programs (Natural Science Foundation) of Jiangsu Province (BK20131004) and the PAPD fund.

REFERENCES

1. Armbrust M, Fox A, Griffith R, Joseph A, Katz R, Konwinski A, *et al.* A view of cloud computing. *Communications of the ACM* 2010; **53**(4): 50–58.
2. Fu Z, Sun X, Liu Q, Zhou L, Shu J. Achieving efficient cloud search services: multi-keyword ranked search over encrypted cloud data supporting parallel computing. *IEICE Transactions on Communications* 2015; **E98B**(4): 190–200.
3. Sandholm T, Lai K. MapReduce optimization using regulated dynamic prioritization. *ACM Sigmetrics Performance Evaluation Review* 2015; **37**(1): 299–310.
4. Anyanwu K, Kim HS, Ravindra P. Algebraic optimization for processing graph pattern queries in the cloud. *IEEE Internet Computing* 2013; **17**(2): 52–61.
5. Shamsi J, Khojaye MA, Qasmi MA. Data-intensive cloud computing: requirements, expectations, challenges, and solutions. *Journal of Grid Computing* 2013; **11**(2): 281–310.
6. Lee KH, Lee YJ, Choi H, Chung YD, Moon B. Parallel data processing with mapreduce: a survey. *ACM Sigmod Record* 2012; **40**(4): 11–20.
7. Wu Y, Ye F, Chen K, Zheng W. Modeling of distributed file systems for practical performance analysis. *IEEE Transactions on Parallel and Distributed Systems* 2014; **25**(1): 156–166.
8. Palanisamy B, Singh A, Liu L. Cost-effective resource provisioning for MapReduce in a cloud. *IEEE Transactions on Parallel and Distributed Systems* 2015; **26** (5): 1265–1279.
9. Kwon Y, Balazinska M, Howe B, Rolia J. Skewtune: mitigating skew in mapreduce applications. *ACM SIGMOD International Conference on Management of Data*, Scottsdale, Arizona, USA, 2012; 25–36.
10. Fu J, Du Z. Load balancing strategy on periodical mapreduce job. *Computer Science* 2013; **40** (30): 38–40, (In Chinese).
11. Gufler B, Augsten N, Reiser A, Kemper A. Handling data skew in mapreduce. *International Conference on Cloud Computing and Services Science*, Noordwijkerhout, The Netherlands, 2011; 574–583.
12. Yuanquan F, Weiguo W, Yunlong X, Heng C. Improving mapreduce performance by balancing skewed loads. *China Communications* 2014; **11**(8): 85–108.
13. Mei Y, Liu L, Pu X, Sivathanu S, Dong X. Performance analysis of network I/O workloads in virtualized data centers. *IEEE Transactions on Services Computing* 2013; **6**(1): 48–63.
14. Islam S, Keung J, Lee K, Liu A. IEmpirical prediction models for adaptive resource provisioning in the cloud. *Future Generation Computer Systems* 2012; **28**(1): 155–162.
15. Breb S, Beier F, Rauhe H, Sattler KU, Schallehn E, Saake G. Efficient co-processor utilization in database query processing. *Information Systems* 2013; **38**(8): 1084–1096.
16. Piao JT, Yan J. Computing resource prediction for mapreduce applications using decision tree. *In Web Technologies and Applications*, Kunming, China, 2012; 570–577.
17. Xiao Z, Song W, Chen Q. Dynamic resource allocation using virtual machines for cloud computing environment. *IEEE Transactions on Parallel and Distributed Systems* 2013; **24**(6): 1107–1117.
18. Chen Q, Liu C, Xiao Z. Improving MapReduce performance using smart speculative execution strategy. *IEEE Transactions on Computers* 2014; **63** (4): 954–967.
19. Bui LT, Abbass H, Barlow M, Bender A. Robustness against the decision-maker's attitude to risk in problems with conflicting objectives. *IEEE Transactions on Evolutionary Computation* 2012; **16**(1): 1–19.
20. Wang J, Shang P, Yin J. DRAW: a new data-grouping-aware data placement scheme for data intensive applications with interest locality. *IEEE Transactions on Magnetics* 2013; **49**(6): 2514–2520.
21. Eltabakh MY, Tian Y, Özcan F, Gemulla R, Krettek A, Mcpherson J. CoHadoop: flexible data placement and its exploitation in Hadoop. *Proceedings of the Vldb Endowment* 2011; **4**(9): 575–585.

22. Aji A, Wang F, Vo H, Lee R, Liu Q, Zhang X, Saltz J. Hadoop GIS: a high-performance spatial data warehousing system over mapreduce. *Proceedings of the VLDB Endowment* 2013; **6**(11): 1009–1020.
23. Lu X, Islam S, Wasi-ur-Rahman M, Jose J, Subramoni H, Wang H, Panda d. High-Performance Design of Hadoop RPC with RDMA over InfiniBand. *42nd International Conference on Parallel Processing (ICPP)*, Lyon, France, 2013; 641–650.
24. Huang G, Huang GB, Son S, You K. Trends in extreme learning machines: a review. *Neural Networks* 2015; **61**: 32–48.
25. Samat A, Du P, Liu S, Li J, Cheng L. Ensemble extreme learning machines for hyperspectral image classification. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* 2014; **77**(4): 1060–1069.
26. Huang G, Zhou H, Ding X, Zhang R. Extreme learning machine for regression and multiclass classification. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics* 2012; **42**(2): 513–529.
27. Savitha R, Suresh S, Kim HJ. A meta-cognitive learning algorithm for an extreme learning machine classifier. *Cognitive Computation* 2014; **6**(2): 253–263.
28. Jemai J, Manel Z, Khaled M. An NSGA-II algorithm for the green vehicle routing problem. *Evolutionary Computation in Combinatorial Optimization*, Málaga, Spain, 2012; 37–48.
29. Rui ME, Rui P, Rong C. K-means clustering in the cloud C: a mahout test. *IEEE Workshops of International Conference on Advanced Information Networking and Applications (WAINA)*, Biopolis, Singapore, 2011; 514–519.
30. Esteves R, Pais R, Rong C. Tarazu: optimizing MapReduce on heterogeneous clusters. *ACM SIGARCH Computer Architecture News* 2012: 61–74.