A Parallel Random Forest Algorithm for Big Data in a Spark Cloud Computing Environment

Jianguo Chen, Kenli Li, *Senior Member, IEEE*, Zhuo Tang, *Member, IEEE*, Kashif Bilal, Shui Yu, *Member, IEEE*, Chuliang Weng, *Member, IEEE*, and Keqin Li, *Fellow, IEEE*

Abstract— With the emergence of the big data age, the issue of how to obtain valuable knowledge from a dataset efficiently and accurately has attracted increasingly attention from both academia and industry. This paper presents a Parallel Random Forest (PRF) algorithm for big data on the Apache Spark platform. The PRF algorithm is optimized based on a hybrid approach combining data-parallel and task-parallel optimization. From the perspective of data-parallel optimization, a vertical data-partitioning method is performed to reduce the data communication cost effectively, and a data-multiplexing method is performed is performed to allow the training dataset to be reused and diminish the volume of data. From the perspective of task-parallel optimization, a dual parallel approach is carried out in the training process of RF, and a task Directed Acyclic Graph (DAG) is created according to the parallel training process of PRF and the dependence of the Resilient Distributed Datasets (RDD) objects. Then, different task schedulers are invoked for the tasks in the DAG. Moreover, to improve the algorithm's accuracy for large, high-dimensional, and noisy data, we perform a dimension-reduction approach in the training process and a weighted voting approach in the prediction process prior to parallelization. Extensive experimental results indicate the superiority and notable advantages of the PRF algorithm over the relevant algorithms implemented by Spark MLlib and other studies in terms of the classification accuracy, performance, and scalability.

Index Terms—Apache Spark, Big Data, Cloud Computing, Data Parallel, Random Forest, Task Parallel.

1 Introduction

1.1 Motivation

With the continuous emergence of a variety of new information dissemination methods, and the rise of cloud computing and Internet of Things (IoT) technologies, data increase constantly with a high speed. The scale of global data continuously increases at a rate of 2 times every two years [1]. The application value of data in every field is becoming more important than ever. There exists a large amount of worthwhile information in available data.

The emergence of the big data age also poses serious problems and challenges besides the obvious benefits. Because of business demands and competitive pressure, almost every business has a high demand for data processing in real-time and validity [2]. As a result, the first problem is how to mine valuable information from massive data efficiently and accurately. At the same time, big data hold characteristics such as high dimensionality, complexity, and noise. Enormous data often hold properties found in various input variables in hundreds or thousands of levels, while each one of them may contain a little information. The second problem is to choose appropriate techniques that may lead to good classification performance for a high-dimensional dataset. Considering the aforementioned facts, data mining and analysis for large-scale data have become a hot topic in academia and industrial research.

The speed of data mining and analysis for large-scale data has also attracted much attention from both academia and industry. Studies on distributed and parallel data mining based on cloud computing platforms have achieved abundant favorable achievements [3, 4]. Hadoop [5] is a famous cloud platform widely used in data mining. In [6, 7], some machine learning algorithms were proposed based on the MapReduce model. However, when these algorithms are implemented based on MapReduce, the intermediate results gained in each iteration are written to the Hadoop Distributed File System (HDFS) and loaded from it. This costs much time for disk I/O operations and also massive resources for communication and storage. Apache Spark [8] is another good cloud platform that is suitable for data mining. In comparison with Hadoop, a Resilient Distributed Datasets (RDD) model and a Directed Acyclic Graph (DAG) model built on a memory computing framework is supported for Spark. It allows us to store a data cache in memory and to perform computation and iteration for the same data directly from memory. The Spark platform saves huge amounts of disk I/O operation time. Therefore, it is more suitable for data mining with iterative computation.

The Random Forest (RF) algorithm [9] is a suitable data mining algorithm for big data. It is an ensemble learning algorithm using feature sub-space to construct the model. Moreover, all decision trees can be trained concurrently, hence it is also suitable for parallelization.

- *Jianguo Chen, Kenli Li, Zhuo Tang, and Keqin Li are with the College of Computer Science and Electronic Engineering, Hunan University, and the National Supercomputing Center in Changsha, Hunan, Changsha 410082, China.*
  Corresponding author: Kenli Li, Email: lkl@hnu.edu.cn.
- *Kashif Bilal is with the Qatar University,Doha 2713, Qatar, and the Comsats Institute of Information Technology, Islamabad 45550, Pakistan.*
- *Shui Yu is with the School of Information Technology, Deakin University, Melbourne, Vic.3216, Australia.*
- *Chuliang Weng is with the School of Computer Science and Software Engineering, Institute for Data Science and Engineering, East China Normal University, Shanghai 200241, China.*
- *Keqin Li is also with the Department of Computer Science, State University of New York, New Paltz, NY 12561, USA.*

1.2 Our Contributions

In this paper, we propose a Parallel Random Forest (PRF) algorithm for big data that is implemented on the Apache Spark platform. The PRF algorithm is optimized based on a hybrid approach combining data-parallel and task-parallel optimization. To improve the classification accuracy of PRF, an optimization is proposed prior to the parallel process. Extensive experiment results indicate the superiority of PRF and depict its significant advantages over other algorithms in terms of the classification accuracy and performance. Our contributions in this paper are summarized as follows.

- An optimization approach is proposed to improve the accuracy of PRF, which includes a dimension-reduction approach in the training process and a weighted voting method in the prediction process.
- A hybrid parallel approach of PRF is utilized to improve the performance of the algorithm, combining data-parallel and task-parallel optimization. In the data-parallel optimization, a vertical data-partitioning method and a data-multiplexing method are performed.
- Based on the data-parallel optimization, a task-parallel optimization is proposed and implemented on Spark. A training task DAG of PRF is constructed based on the RDD model, and different task schedulers are invoked to perform the tasks in the DAG. The performance of PRF is improved noticeably.

The rest of the paper is organized as follows. Section 2 reviews the related work. Section 3 gives the RF algorithm optimization from two aspects. The parallel implementation of the RF algorithm on Spark is developed in Section 4. Experimental results and evaluations are shown in Section 5 with respect to the classification accuracy and performance. Finally, Section 6 presents a conclusion and future work.

2 Related Work

Although traditional data processing techniques have achieved good performance for small-scale and low-dimensional datasets, they are difficult to be applied to large-scale data efficiently [10–12]. When a dataset becomes more complex with characteristics of a complex structure, high dimensionality, and a large size, the accuracy and performance of traditional data mining algorithms are significantly declined [13].

Due to the need to address the high-dimensional and noisy data, various improvement methods have been introduced by researchers. Xu [14] proposed a dimension-reduction method for the registration of high-dimensional data. The method combines datasets to obtain an image pair with a detailed texture and results in improved image registration. Tao et al. [15] and Lin et al. [16] introduced some classification algorithms for high-dimensional data to address the issue of dimension-reduction. These algorithms use multiple kernel learning framework and multilevel maximum margin features and achieve efficient dimensionality reduction in binary classification problems. Strobl [17] and Bernard [18] studied the variable importance measures of RF and proposed some improved models for it. Taghi et al. [19] compared the boosting and bagging techniques and proposed an algorithm for noisy and imbalanced data. Yu et al. [20] and Biau [21] focused on RF for high-dimensional and noisy data and applied RF in many applications such as multi-class action detection and facial feature detection, and achieved a good effort. Based on the existing research results, we propose a new optimization approach in this paper to address the problem of high-dimensional and noisy data, which reduces the dimensionality of the data according to the structure of the RF and improves the algorithm's accuracy with a low computational cost.

Focusing on the performance of classification algorithms for large-scale data, numerous studies on the intersection of parallel/distributed computing and the learning of tree models were proposed. Basilico et al. [22] proposed a COMET algorithm based on MapReduce, in which multiple RF ensembles are built on distributed blocks of data. Svore et al. [23] proposed a boosted decision tree ranking algorithm, which addresses the speed and memory constraints by distributed computing. Panda et al. [24] introduced a scalable distributed framework based on MapReduce for the parallel learning of tree models over large datasets. A parallel boosted regression tree algorithm was proposed in [25] for web search ranking, in which a novel method for parallelizing the training of GBRT was performed based on data partitioning and distributed computing.

Focusing on resource allocation and task-parallel execution in a parallel and distributed environment, Warneke et al. [26] implemented a dynamic resource allocation for efficient parallel data processing in a cloud environment. Lena et al. [27] carried out an energy-aware scheduling of MapReduce jobs for big data applications. Luis et al. [28] proposed a robust resource allocation of data processing on a heterogeneous parallel system, in which the arrival time of datasets are uncertainty. Zhang et al. [29] proposed an evolutionary scheduling of dynamic multitasking workloads for big data analysis in an elastic cloud. Meanwhile, our team also focused on parallel tasks scheduling on heterogeneous cluster and distributed systems and achieved positive results[30, 31].

Apache Spark Mllib [32] parallelized the RF algorithm (referred to Spark-MLRF in this paper) based on a data-parallel optimization to improve the performance of the algorithm. However, there exist many drawbacks in the Spark-MLRF. First, in the stage of determining the best split segment for continuous features, a method of sampling for each partition of the dataset is used to reduce the data transmission operations. However, the cost of this method is its reduced accuracy. In addition, because the data-partitioning method in Spark-MLRF is a horizontal partition, the data communication of the feature variable gain ratio computing is a global communication.

To improve the performance of the RF algorithm and mitigate the data communication cost and workload imbalance problems of large-scale data in parallel and distributed environments, we propose a hybrid parallel approach for RF combining data-parallel and task-parallel optimization based on the Spark RDD and DAG models. In comparison with the existing study results, our method reduces the volume of the training dataset without decreasing the algorithm's accuracy. Moreover, our method mitigates the data communication and workload imbalance problems of

large-scale data in parallel and distributed environments.

3 Random Forest Algorithm Optimization

Owing to the improvement of the classification accuracy for high-dimensional and large-scale data, we propose an optimization approach for the RF algorithm. First, a dimension-reduction method is performed in the training process. Second, a weighted voting method is constructed in the prediction process. After these optimizations, the classification accuracy of the algorithm is evidently improved.

3.1 Random Forest Algorithm

The random forest algorithm is an ensemble classifier algorithm based on the decision tree model. It generates k different training data subsets from an original dataset using a bootstrap sampling approach, and then, k decision trees are built by training these subsets. A random forest is finally constructed from these decision trees. Each sample of the testing dataset is predicted by all decision trees, and the final classification result is returned depending on the votes of these trees.

The original training dataset is formalized as $S = \{(x_i, y_j), i = 1, 2, ..., N; j = 1, 2, ..., M\}$, where x is a sample and y is a feature variable of S. Namely, the original training dataset contains N samples, and there are M feature variables in each sample. The main process of the construction of the RF algorithm is presented in Fig. 1.

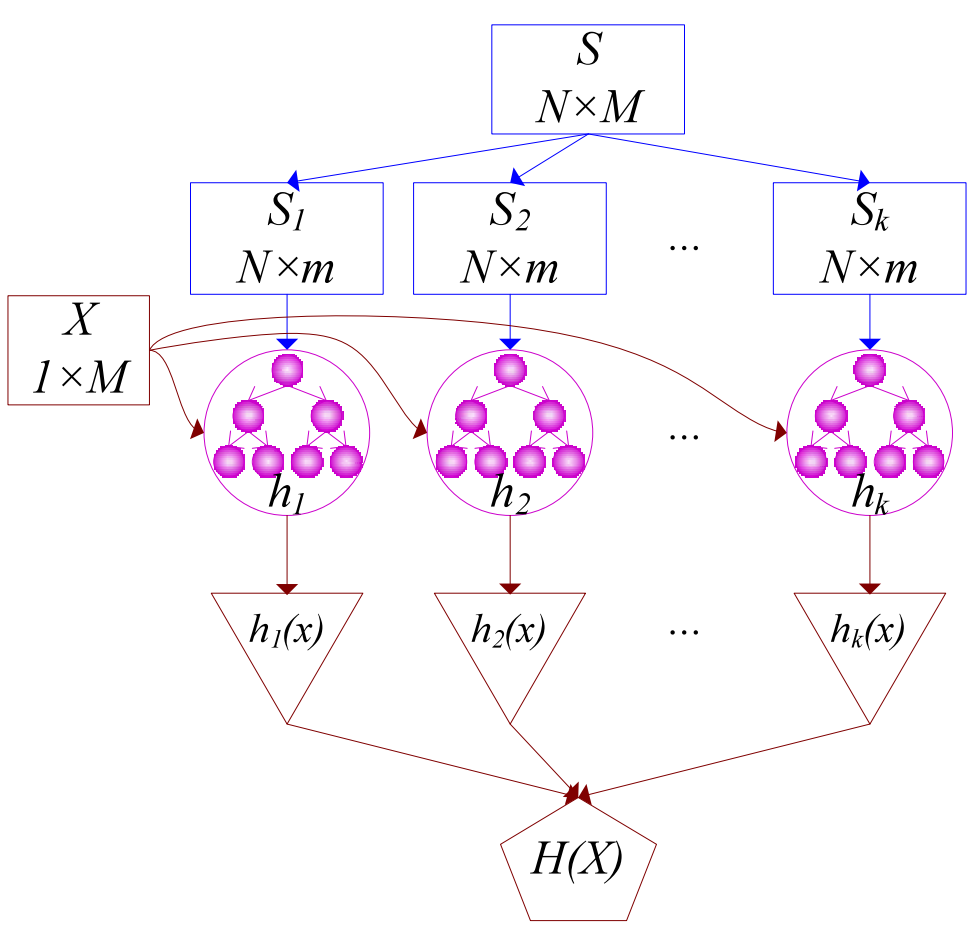

Fig. 1. Process of the construction of the RF algorithm

The steps of the construction of the random forest algorithm are as follows.

Step 1. Sampling k training subsets.

In this step, k training subsets are sampled from the original training dataset S in a bootstrap sampling manner. Namely, N records are selected from S by a random sampling and replacement method in each sampling time. After the current step, k training subsets are constructed as a collection of training subsets S_{Train}:

$$S_{Train} = \{S_1, S_2, ..., S_k\}.$$

At the same time, the records that are not to be selected in each sampling period are composed as an Out-Of-Bag (OOB) dataset. In this way, k OOB sets are constructed as a collection of S_{OOB}:

$$S_{OOB} = \{OOB_1, OOB_2, ..., OOB_k\},$$

where $k \ll N$, $S_i \bigcap OOB_i = \phi$ and $S_i \bigcup OOB_i = S$. To obtain the classification accuracy of each tree model, these OOB sets are used as testing sets after the training process.

Step 2. Constructing each decision tree model.

In an RF model, each meta decision tree is created by a C4.5 or CART algorithm from each training subset S_i. In the growth process of each tree, m feature variables of dataset S_i are randomly selected from M variables. In each tree node's splitting process, the gain ratio of each feature variable is calculated, and the best one is chosen as the splitting node. This splitting process is repeated until a leaf node is generated. Finally, k decision trees are trained from k training subsets in the same way.

Step 3. Collecting k trees into an RF model.

The k trained trees are collected into an RF model, which is defined in Eq. (1):

$$H(X, \Theta j) = \sum_{i=1}^{k} h_i(x, \Theta j), (j = 1, 2, ..., m), \quad (1)$$

where $h_i(x, \Theta j)$ is a meta decision tree classifier, X are the input feature vectors of the training dataset, and Θj is an independent and identically distributed random vector that determines the growth process of the tree.

3.2 Dimension Reduction for High-Dimensional Data

To improve the accuracy of the RF algorithm for the high-dimensional data, we present a new dimension-reduction method to reduce the number of dimensions according to the importance of the feature variables. In the training process of each decision tree, the Gain Ratio (GR) of each feature variable of the training subset is calculated and sorted in descending order. The top k variables ($k \ll M$) in the ordered list are selected as the principal variables, and then, we randomly select $(m - k)$ further variables from the remaining $(M - k)$ ones. Therefore, the number of dimensions of the dataset is reduced from M to m. The process of dimension-reduction is presented in Fig. 2.

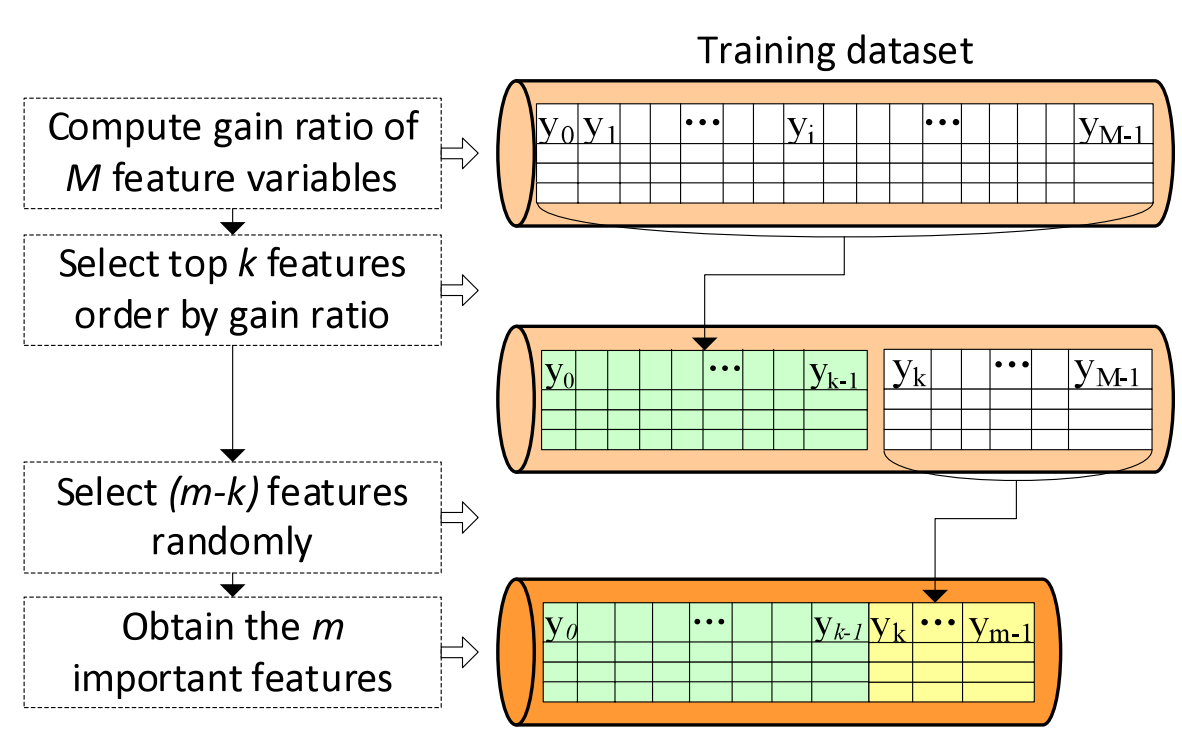

Fig. 2. Dimension-reduction in the training process

First, in the training process of each decision tree, the entropy of each feature variable is calculated prior to the node-splitting process. The entropy of the target variable in the training subset S_i $(i = 1, 2, ..., k)$ is defined in Eq. (2):

$$Entropy(S_i) = \sum_{a=1}^{c1} -p_a \log p_a, \quad (2)$$

where $c1$ is the number of different values of the target variable in S_i, and p_a is the probability of the type of value a within all types in the target variable subset.

Second, the entropy for each input variable of S_i, except for the target variable, is calculated. The entropy of each input variable y_{ij} is defined in Eq. (3):

$$Entropy(y_{ij}) = \sum_{v \in V(y_{ij})} \frac{|S_{(v,i)}|}{|S_i|} Entropy(v(y_{ij})), \quad (3)$$

where the elements of Eq. (3) are described in Table 1.

TABLE 1
Explanation of the elements of Eq. (3)

Element	Description
y_{ij}	the j-th feature variable of S_i, $j = 1, 2, ..., M$.
$V(y_{ij})$	the set of all possible values of y_{ij}.
$\|S_i\|$	the number of samples in S_i.
$S_{(v,i)}$	a sample subset in S_i, where the value of y_j is v.
$\|S_{(v,i)}\|$	the number of the sample subset $S_{(v,i)}$.

Third, the self-split information $I(y_{ij})$ of each input variable is calculated, as defined in Eq. (4):

$$I(y_{ij}) = \sum_{a=1}^{c2} -p_{(a,j)} \log_2(p_{(a,j)}), \quad (4)$$

where $c2$ is the number of different values of y_{ij}, and $p_{(a,j)}$ is the probability of the type of value a within all types in variable y_j. Then, the information gain of each feature variable is calculated, as defined in Eq. (5):

$$\begin{aligned} G(y_{ij}) =& Entropy(S_i) - Entropy(y_{ij}) \\ =& Entropy(S_i) \\ &- \sum_{v \in V(y_{ij})} \frac{|S_{(v,i)}|}{|S_i|} Entropy(v(y_{ij})), \end{aligned} \quad (5)$$

where $v(y_j) \in V(y_j)$.

By using the information gain to measure the feature variables, the largest value is selected easily, but it will lead to an over fitting problem. To overcome this problem, a gain ratio value is taken to measure the feature variables, and the features with the maximum value are selected. The gain ratio of the feature variable y_{ij} is defined in Eq. (6):

$$GR(y_{ij}) = \frac{G(y_{ij})}{I(y_{ij})}. \quad (6)$$

To reduce the dimensions of the training dataset, we calculate the importance of each feature variable according to the gain ratio of the variable. Then, we select the most important features and delete the ones with less importance. The importance of each feature variable is defined as follows.

Definition 1. The importance of each feature variable in a training subset refers to the portion of the gain ratio of the feature variable compared with the total feature variables. The importance of feature variable y_{ij} is defined as $VI(y_{ij})$ in Eq. (7):

$$VI(y_{ij}) = \frac{GR(y_{ij})}{\sum_{(a=1)}^{M} GR(y_{(i,a)})}. \quad (7)$$

The importance values of all feature variables are sorted in descending order, and the top k ($k \ll M, k < m$) values are selected as the most important. We then randomly select $(m - k)$ further feature variables from the remaining $(M - k)$ ones. Thus, the number of dimensions of the dataset is reduced from M to m. Taking the training subset S_i as an example, the detailed steps of the dimension-reduction in the training process are presented in Algorithm 3.1.

Algorithm 3.1 Dimension-reduction in the training process

Input:
  S_i: the ith training subset;
  k: the number of important variables selected by VI;
  m: the number of the selected feature variables.
Output:
  F_i: a set of m important feature variables of S_i.

```
create an empty string array F_i;
calculate Entropy(S_i) for the target feature variable;
for each feature variable y_ij in S_i do
    calculate Entropy(y_ij) for each input feature variable;
    calculate gain G(y_ij) ← Entropy(S_i) − Entropy(y_ij);
    calculate split information I(y_ij) ← Σ_{a=1}^{c2} −p_(a,j) log_2(p_(a,j));
    calculate gain ratio GR(y_ij) ← G(y_ij) / I(y_ij);
end for
calculate variable importance VI(y_ij) ← GR(y_ij) / Σ_{(a=1)}^{M} GR(y_(i,a))
    for feature variable y_ij;
sort M feature variables in descending order by VI(y_ij);
put top k feature variables to F_i[0, ..., k − 1];
set c ← 0;
for j = k to M − 1 do
    while c < (m − k) do
        select y_ij from (M − k) randomly;
        put y_ij to F_i[k + c];
        c ← c + 1;
    end while
end for
return F_i.
```

In comparison with the original RF algorithm, our dimension-reduction method ensures that the m selected feature variables are optimal while maintaining the same computational complexity as the original algorithm. This balances the accuracy and diversity of the feature selection ensemble of the RF algorithm and prevents the problem of classification over fitting.

3.3 Weighted Voting Method

In the prediction and voting process, the original RF algorithm uses a traditional direct voting method. In this case, if the RF model contains noisy decision trees, it likely leads to a classification or regression error for the testing dataset. Consequently, its accuracy is decreased. To address this problem, a weighted voting approach is proposed in this section to improve the classification accuracy for the testing dataset. The accuracy of the classification or regression of each tree is regarded as the voting weight of the tree.

After the training process, each OOB set OOB_i is tested by its corresponding trained tree h_i. Then, the classification accuracy CA_i of each decision tree h_i is computed.

Definition 2. The classification accuracy of a decision tree is defined as the ratio of the average number of votes in the correct classes to that in all classes, including error classes, as classified by the trained decision tree. The classification accuracy is defined in Eq. (8):

$$CA_i = \frac{I(h_i(x) = y)}{I(h_i(x) = y) + \sum I(h_i(x) = z)}, \tag{8}$$

where $I(\cdot)$ is an indicator function, y is a value in the correct class, and z is a value in the error class ($z \neq y$).

In the prediction process, each record of the testing dataset X is predicted by all decision trees in the RF model, and then, a final vote result is obtained for the testing record. When the target variable of X is quantitative data, the RF is trained as a regression model. The result of the prediction is the average value of k trees. The weighted regression result $H_r(X)$ of X is defined in Eq. (9):

$$\begin{aligned} H_r(X) &= \frac{1}{k}\sum_{i=1}^{k}[w_i \times h_i(x)] \\ &= \frac{1}{k}\sum_{i=1}^{k}[CA_i \times h_i(x)], \end{aligned} \tag{9}$$

where w_i is the voting weight of the decision tree h_i.

Similarly, when the target feature of X is qualitative data, the RF is trained as a classification model. The result of the prediction is the majority vote of the classification results of k trees. The weighted classification result $H_c(X)$ of X is defined in Eq. (10):

$$\begin{aligned} H_c(X) &= Majority_{i=1}^{k}[w_i \times h_i(x)] \\ &= Majority_{i=1}^{k}[CA_i \times h_i(x)]. \end{aligned} \tag{10}$$

The steps of the weighted voting method in the prediction process are described in Algorithm 3.2.

Algorithm 3.2 Weighted voting in the prediction process

Input:
 X: a testing dataset;
 $PRF_{trained}$: the trained PRF model.
Output:
 $H(X)$: the final prediction result for X.
for each testing data x in X **do**
  for each decision tree T_i in $PRF_{trained}$ **do**
   predict the classification or regression result $h_i(x)$ by T_i;
  end for
end for
set classification accuracy CA_i as the weight w_i of T_i;
for each testing data x in X **do**
  if (operation type == classification) **then**
   vote the final result $H_c(x) \leftarrow Majority_{i=1}^{k}[w_i \times h_i(x)]$;
   $H(X) \leftarrow H_c(x)$;
  else if (operation type == regression) **then**
   vote the final result $H_r(x) \leftarrow \frac{1}{k}\sum_{i=1}^{k}[w_i \times h_i(x)]$;
   $H(X) \leftarrow H_r(x)$;
  end if
end for
return $H(X)$.

In the weighted voting method of RF, each tree classifier corresponds to a specified reasonable weight for voting on the testing data. Hence, this improves the overall classification accuracy of RF and reduces the generalization error.

3.4 Computational Complexity

The computational complexity of the original RF algorithm is $O(kMN \log N)$, where k is the number of decision trees in RF, M is the number of features, N is the number of samples, and $\log N$ is the average depth of all tree models. In our improved PRF algorithm with dimension-reduction (PRF-DR) described in Section 3, the time complexity of the dimension reduction is $O(MN)$. The computational complexity of the splitting process for each tree node is set as one unit (1), which contains functions such as $entropy()$, $gain()$, and $gainratio()$ for each feature subspace. After the dimension reduction, the number of features is reduced from M to m ($m \ll M$). Therefore, the computational complexity of training a meta tree classifier is $O(MN + mN \log N)$, and the total computational complexity of the PRF-DR algorithm is $O(k(MN + mN \log N))$.

4 Parallelization of the Random Forest Algorithm on Spark

To improve the performance of the RF algorithm and mitigate the data communication cost and workload imbalance problems of large-scale data in a parallel and distributed environment, we propose a Parallel Random Forest (PRF) algorithm on Spark. The PRF algorithm is optimized based on a hybrid parallel approach combining data-parallel and task-parallel optimization. From the perspective of data-parallel optimization, a vertical data-partitioning method and a data-multiplexing method are performed. These methods reduce the volume of data and the number of data transmission operations in the distributed environment without reducing the accuracy of the algorithm. From the perspective of task-parallel optimization, a dual-parallel approach is carried out in the training process of the PRF algorithm, and a task DAG is created according to the dependence of the RDD objects. Then, different task schedulers are invoked to perform the tasks in the DAG. The dual-parallel training approach maximizes the parallelization of PRF and improves the performance of PRF. Then task schedulers further minimize the data communication cost among the Spark cluster and achieve a better workload balance.

4.1 Data-Parallel Optimization

We introduce the data-parallel optimization of the PRF algorithm, which includes a vertical data-partitioning and a data-multiplexing approach. First, taking advantage of the RF algorithm's natural independence of feature variables and the resource requirements of computing tasks, a vertical data-partitioning method is proposed for the training dataset. The training dataset is split into several feature subsets, and each feature subset is allocated to the Spark cluster in a static data allocation way. Second, to address the problem that the data volume increases linearly with the increase in the scale of RF, we present a data-multiplexing method for PRF by modifying the traditional sampling method. Notably, our data-parallel optimization method reduces the volume of data and the number of data

transmission operations without reducing the accuracy of the algorithm. The increase in the scale of the PRF does not lead to a change in the data size and storage location.

4.1.1 Vertical Data Partitioning

In the training process of PRF, the gain-ratio computing tasks of each feature variable take up most of the training time. However, these tasks only require the data of the current feature variable and the target feature variable. Therefore, to reduce the volume of data and the data communication cost in a distributed environment, we propose a vertical data-partitioning approach for PRF according to the independence of feature variables and the resource requirements of computing tasks. The training dataset is divided into several feature subsets.

Assume that the size of training dataset S is N and there are M feature variables in each record. $y_0 \sim y_{M-2}$ are the input feature variables, and y_{M-1} is the target feature variable. Each input feature variable y_j and the variable y_{M-1} of all records are selected and generated to a feature subset FS_j, which is represented as:

$$FS_j = \begin{bmatrix} < 0, y_{0j}, y_{0(M-1)} >, \\ < 1, y_{1j}, y_{1(M-1)} >, \\ ..., \\ < i, y_{ij}, y_{i(M-1)} >, \\ ..., \\ < (N-1), y_{(N-1)j}, y_{(N-1)(M-1)} > \end{bmatrix},$$

where i is the index of each record of the training dataset S, and j is the index of the current feature variable. In such a way, S is split into $(M-1)$ feature subsets before dimension-reduction. Each subset is loaded as an RDD object and is independent of the other subsets. The process of the vertical data-partitioning is presented in Fig. 3.

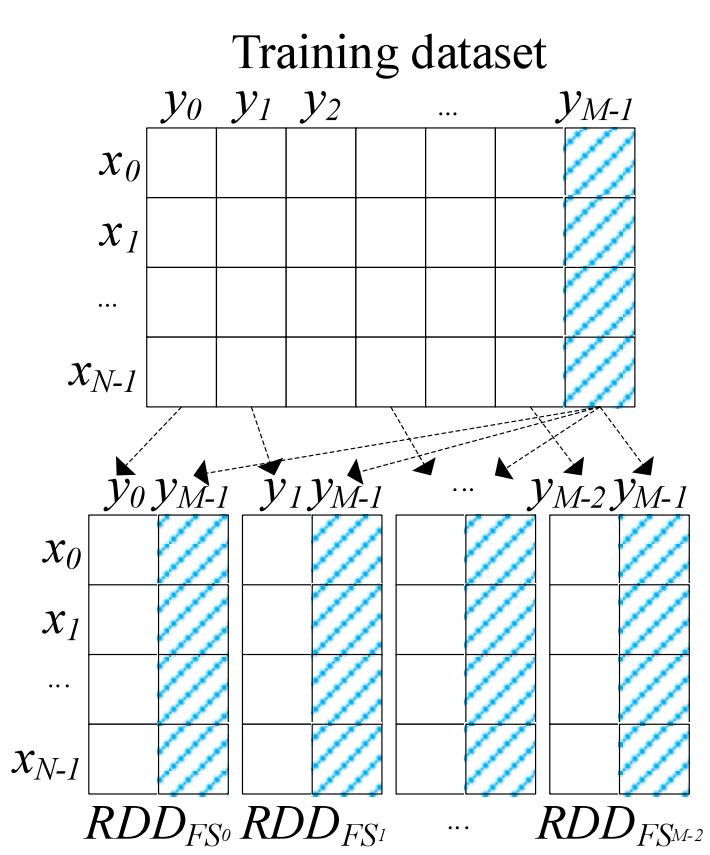

Fig. 3. Process of the vertical data-partitioning method

4.1.2 Data-Multiplexing Method

To address the problem that the volume of the sampled training dataset increases linearly with the increase in the RF scale, we present a data-multiplexing method for PRF by modifying the traditional sampling method. In each sampling period, we do not copy the sampled data but just note down their indexes into a Data-Sampling-Index (DSI) table. Then, the DSI table is allocated to all slave nodes together with the feature subsets. The computing tasks in the training process of each decision tree load the corresponding data from the same feature subset via the DSI table. Thus, each feature subset is reused effectively, and the volume of the training dataset will not increase any more despite the expansion of the RF scale.

First, we create a DSI table to save the data indexes generated in all sampling times. As mentioned in Section 3.1, the scale of a RF model is k. Namely, there are k sampling times for the training dataset, and N data indexes are noted down in each sampling time. An example of the DSI table of PRF is presented in Table 2.

TABLE 2
Example of the DSI table of PRF

		Data indexes of training dataset						
Sampling times	$Sample_0$	1	3	8	5	10	0	...
	$Sample_1$	2	4	1	9	7	8	...
	$Sample_2$	9	1	12	92	2	5	...
	$Sample_3$	3	8	87	62	0	2	...
	...	...	...	...	...	...	...	...
	$Sample_{k-1}$	9	1	4	43	3	5	...

Second, the DSI table is allocated to all slave nodes of the Spark cluster together with all feature subsets. In the subsequent training process, the gain-ratio computing tasks of different trees for the same feature variable are dispatched to the slaves where the required subset is located.

Third, each gain-ratio computing task accesses the relevant data indexes from the DSI table, and obtains the feature records from the same feature subset according to these indexes. The process of the data-multiplexing method of PRF is presented in Fig. 4.

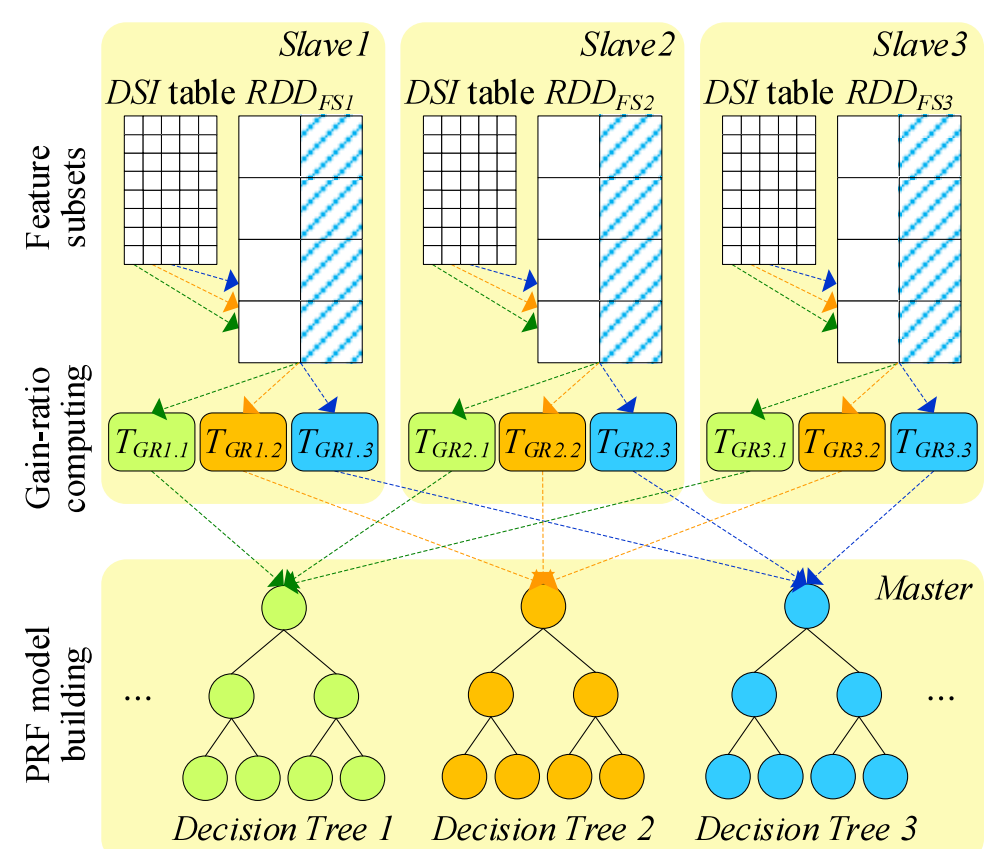

Fig. 4. Process of the data-multiplexing method of PRF

In Fig. 4, each RDD_{FS} refers to an RDD object of a feature subset, and each T_{GR} refers to a gain-ratio computing task. For example, we allocate tasks $\{T_{GR1.1}, T_{GR1.2}, T_{GR1.3}\}$ to $Slave1$ for the feature subset RDD_{FS1}, allocate tasks $\{T_{GR2.1}, T_{GR2.2}, T_{GR2.3}\}$ to $Slave2$ for RDD_{FS2}, and allocate tasks $\{T_{GR3.1}, T_{GR3.2}, T_{GR3.3}\}$ to $Slave3$ for RDD_{FS3}. From the perspective of the decision trees, tasks in the same slave node belong to different trees. For example, tasks $T_{GR1.1}$, $T_{GR1.2}$, and $T_{GR1.3}$ in the $Slave1$ belong to "$DecisionTree1$", "$DecisionTree2$", and "$DecisionTree3$", respectively. These tasks obtain records from the same feature subset according to the corresponding

indexes in DSI, and compute the gain ratio of the feature variable for different decision trees. After that, the intermediate results of these tasks are submitted to the corresponding subsequent tasks to build meta decision trees. Results of the tasks $\{T_{GR1.1}, T_{GR2.1}, T_{GR3.1}\}$ are combined for the tree node splitting process of "$DecisionTree1$", and results of the tasks $\{T_{GR1.2}, T_{GR2.2}, T_{GR3.2}\}$ are combined for that of "$DecisionTree2$".

4.1.3 Static Data Allocation

To achieve a better balance of data storage and workload, after the vertical data-partitioning, a static data allocation method is applied for the feature subsets. Namely, these subsets are allocated to a distributed Spark cluster before the training process of PRF. Moreover, because of the difference of the data type and volume of each feature subset, the workloads of the relevant subsequent computing tasks will be different as well. As we know, a Spark cluster is constructed by a master node and several slave nodes. We define our allocation function to determine each feature subset be allocated to which nodes, and allocate each feature subset according to its volume. There are 3 scenarios in the data allocation scheme. Examples of the 3 scenarios of the data allocation method are shown in Fig. 5.

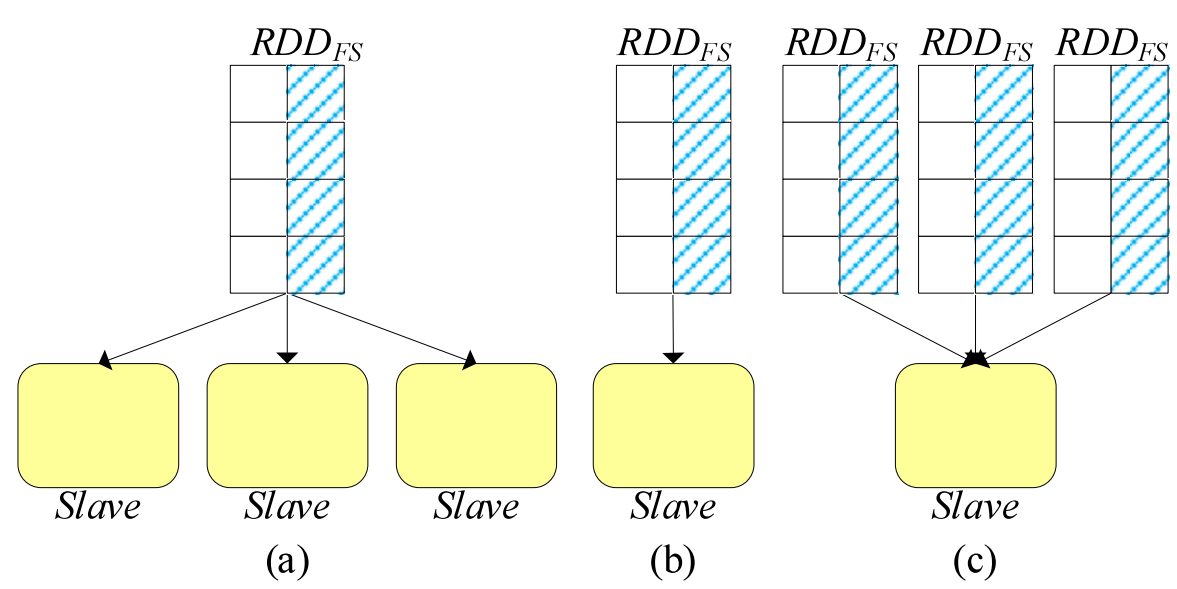

Fig. 5. Example of 3 scenarios of the data allocation

In Fig. 5, (a) when the size of a feature subset is greater than the available storage capacity of a slave node, this subset is allocated to limited multiple slaves that have similar physical locations. (b) When the size of a feature subset is equal to the available storage capacity of a slave node, the subset is allocated to the node. (c) When the size of a feature subset is smaller than the available storage capacity of a slave node, this node will accommodate multiple feature subsets. In case (a), the data communication operations of the subsequent gain-ratio computing tasks occur among the slave nodes where the current feature subset is located. These data operations are local communications but not global communications. In cases (b) and (c), no data communication operations occur among different slave nodes in the subsequent gain-ratio computation process. The steps of the vertical data-partitioning and static data allocation of PRF are presented in Algorithm 4.1.

In Algorithm 4.1, RDD_o is split into $(M-1)$ RDD_{FS} objects via the vertical data-partitioning function firstly. Then, each RDD_{FS} is allocated to slave nodes according to its volume and the available storage capacity of the slave nodes. To reuse the training dataset, each RDD object of the feature subset is allocated and persisted to Spark cluster via a $dataAllocation()$ function and a $persist()$ function.

Algorithm 4.1 Vertical data-partitioning and static data allocation of PRF

Input:
RDD_o: an RDD object of the original training dataset S.
Output:
L_{FS}: a list of the indexes of each feature subset's RDD object and the allocated slave nodes.

for $j = 0$ to $(M-2)$ **do**
  $RDD_{FSj} \leftarrow RDD_o$.**map**
    $< i, y_{ij}, y_{i(M-1)} > \leftarrow RDD_o$.verticalPartition(j);
  end map.collect()
  $slaves \leftarrow$ findAvailableSlaves().sortbyIP();
  if RDD_{FSj}.size $< slaves[0]$.availablesize **then**
    dataAllocation(RDD_{FSj}, $slaves[0]$);
    $slaves[0]$.availablesize $\leftarrow$ $slaves[0]$.availablesize - RDD_{FSj}.size;
    $L_{FS} \leftarrow < RDD_{FSj}.id, slaves[0].nodeid >$;
  else
    while $RDD_{FSj} \neq null$ **do**
      $(RDD'_{FSj},\ RDD''_{FSj}) \leftarrow$ dataPartition(RDD_{FSj}, $slaves[i]$.availablesize));
      dataAllocation(RDD'_{FSj}, $slaves[i]$);
      RDD'_{FSj}.persist();
      $slaves[i]$.availablesize $\leftarrow$ $slaves[i]$.availablesize - RDD'_{FSj}.size;
      $slavesids \leftarrow slaves[i].nodeid;$
      $RDD_{FSj} \leftarrow RDD''_{FSj};$
      $i \leftarrow i + 1;$
    end while
    $L_{FS} \leftarrow < RDD_{FSj}.id, slavesids >;$
  end if
end for
return L_{FS}.

4.2 Task-Parallel Optimization

Each decision tree of PRF is built independent of each other, and each sub-node of a decision tree is also split independently. The structures of the PRF model and decision tree model make the computing tasks have natural parallelism. Based on the results of the data-parallel optimization, we propose a task-parallel optimization for PRF and implement it on Spark. A dual-parallel approach is carried out in the training process of PRF, and a task DAG is created according to the dual-parallel training process and the dependence of the RDD objects. Then, different task schedulers are invoked to perform the tasks in the DAG.

4.2.1 Parallel Training Process of PRF

In our task-parallel optimization approach, a dual-parallel training approach is proposed in the training process of PRF on Spark. k decision trees of the PRF model are built in parallel at the first level of the training process. And $(M-1)$ feature variables in each decision tree are calculated concurrently for tree node splitting at the second level of the training process.

There are several computing tasks in the training process of each decision tree of PRF. According to the required data resources and the data communication cost, the computing tasks are divided into two types, gain-ratio computing tasks and node-splitting tasks, which are defined as follows.

Definition 3. Gain-ratio-computing task (T_{GR}) is a task set that is employed to compute the gain ratio of a feature

variable from the corresponding feature subset, which includes a series of calculations for each feature variable, i.e., the entropy, the self-split information, the information gain, and the gain ratio. The results of T_{GR} tasks are submitted to the corresponding subsequent node-splitting tasks.

Definition 4. Node-splitting task (T_{NS}) is a task set that is employed to collect the results of the relevant T_{GR} tasks and split the decision tree nodes, which includes a series of calculations for each tree node, such as determining the best splitting variable holds the highest gain ratio value and splitting the tree node by the variable. After the tree node splitting, the results of T_{NS} tasks are distributed to each slave to begin the next stage of the PRF's training process.

The steps of the parallel training process of the PRF model are presented in Algorithm 4.2.

Algorithm 4.2 Parallel training process of the PRF model

Input:
k: the number of decision trees of the PRF model;
T_{DSI}: the DSI table of PRF;
L_{FS}: a list of the indexes of each feature subset's RDD object and the allocated slave nodes.
Output:
$PRF_{trained}$: the trained PRF model.

for i = 0 to $(k-1)$ **do**
  for j = 0 to $(M-2)$ **do**
    load feature subset $RDD_{FSj} \leftarrow$ loadData($L_{FS}[i]$);
    $//T_{GR}$:
    $RDD_{(GR,best)} \leftarrow$ sc.parallelize(RDD_{FSj}).**map**
      load sampled data $RDD_{(i,j)} \leftarrow (T_{DSI}[i], RDD_{FSj})$;
      calculate the gain ratio $GR_{(i,j)} \leftarrow GR(RDD_{(i,j)})$;
    end map
    $//T_{NS}$:
    $RDD_{(GR,best)}$.collect().sorByKey(GR).top(1);
    for each value $y_{(j,v)}$ in $RDD_{(GR,best)}$ **do**
      split tree node $Node_j \leftarrow < y_{(j,v)}, Value >$;
      append $Node_j$ to T_i;
    end for
  end for
  $PRF_{trained} \leftarrow T_i$;
end for
return $PRF_{trained}$.

According to the parallel training process of PRF and the dependence of each RDD object, each job of the program of PRF's training process is split into different stages, and a task DAG is constructed with the dependence of these job stages. Taking a decision tree model of PRF as an example, a task DAG of the training process is presented in Fig. 6.

There are several stages in the task DAG, which correspond to the levels of the decision tree model. In stage 1, after the dimension-reduction, $(m-1)$ T_{GR} tasks ($T_{GR1.0} \sim T_{GR1.(m-2)}$) are generated for the $(m-1)$ input feature variables. These T_{GR}s compute the gain ratio the corresponding feature variable, and submit their results to T_{NS1}. T_{NS1} finds the best splitting variable and splits the first tree node for the current decision tree model. Assuming that y_0 is the best splitting variable at the current stage, and the value of y_0 is in the range of $\{v_{01}, v_{02}, v_{03}\}$. Hence, the first tree node is constructed by y_0, and 3 sub-nodes are split from the node, as shown in Fig. 6(b). After tree node splitting, the intermediate result of T_{NS1} are distributed to all slave nodes. The result includes information of the

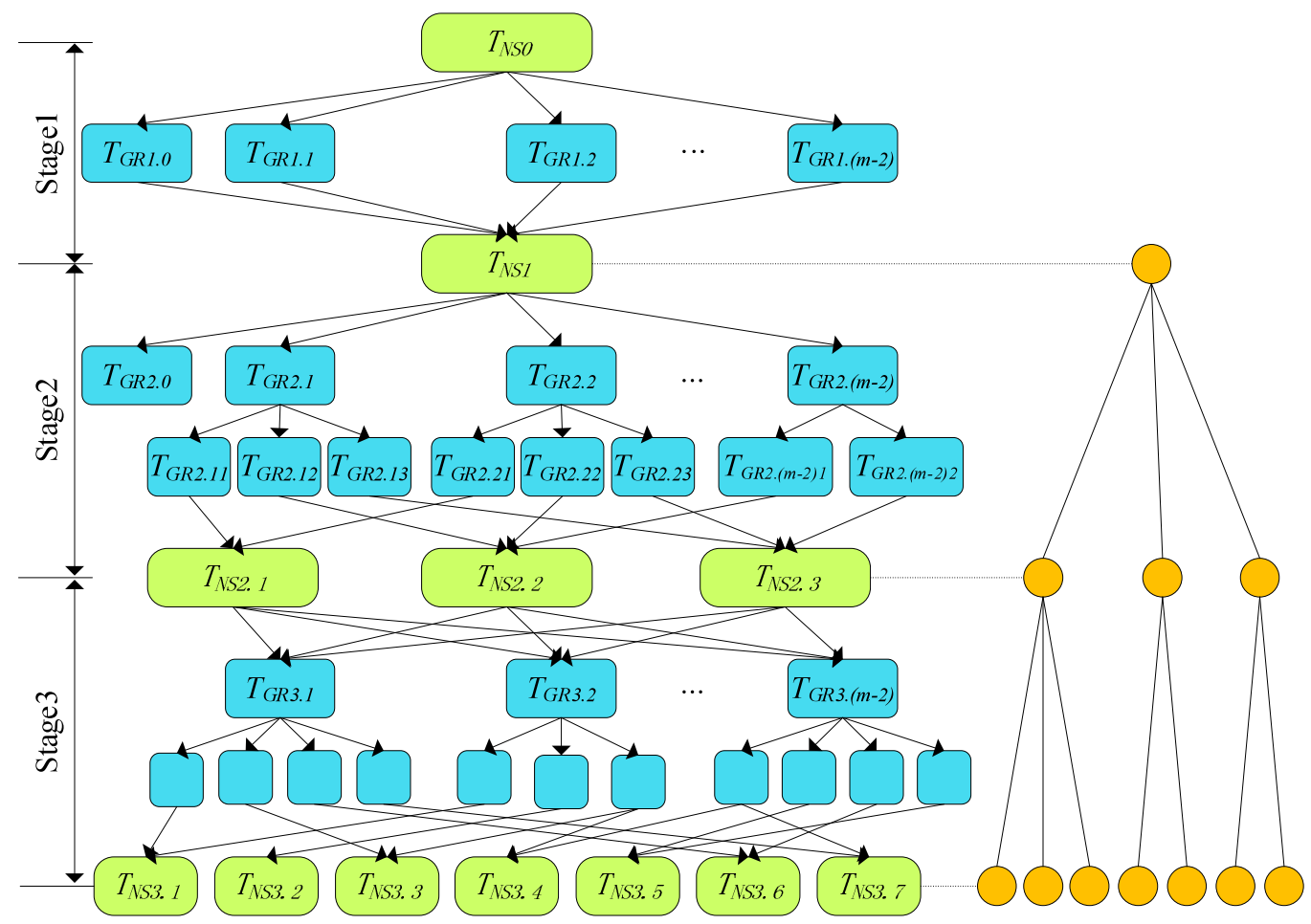

Fig. 6. Example of the task DAG for a decision tree of PRF

splitting variable and the data index list of $\{v_{01}, v_{02}, v_{03}\}$.

In stage 2, because y_0 is the splitting feature, there is no T_{GR} task for FS_0. The potential workload balance problem of this issue will be discussed in Section 4.3.4. New T_{GR} tasks are generated for all other feature subsets according to the result of T_{NS1}. Due to the data index list of $\{v_{01}, v_{02}, v_{03}\}$, there are no more than 3 T_{GR} tasks for each feature subset. For example, tasks $T_{GR2.11}$, $T_{GR2.12}$, and $T_{GR2.13}$ calculate the data of FS_1 with the indexes corresponding to v_{01}, v_{02}, and v_03, respectively. And the condition is similar in tasks for $FS_2 \sim FS_{(m-2)}$. Then, the results of tasks $\{T_{GR2.11}, T_{GR2.21}, T_{GR2.(m-2)1}\}$ are submitted to task $T_{NS2.1}$ for the same sub-tree-node splitting. Tasks of other tree nodes and other stages are performed similarly. In such a way, a task DAG of the training process of each decision tree model is built. In addition, k DAGs are built respectively for the k decision trees of the PRF model.

4.2.2 Task-Parallel Scheduling

After the construction of the task DAGs of all the decision trees, the tasks in these DAGs are submitted to the Spark task scheduler. There exist two types of computing tasks in the DAG, which have different resource requirements and parallelizables. To improve the performance of PRF efficiently and further minimize the data communication cost of tasks in the distributed environment, we invoke two different task-parallel schedulers to perform these tasks.

In Spark, the $TaskSchedulerListener$ module monitors the submitted jobs, splits the job into different stages and tasks, and submits these tasks to the $TaskScheduler$ module. The $TaskScheduler$ module receives the tasks and allocates and executes them using the appropriate executors. According to the different allocations, the $TaskScheduler$ module includes 3 sub-modules, such as $LocalScheduler$, $ClusterScheduler$, and $MessosScheduler$. Meanwhile, each task holds 5 types of locality property value: $PROCESS_LOCAL$, $NODE_LOCAL$, NO_PREF, $PACK_LOCAL$, and ANY. We set the value of the locality properties of these two types of tasks and submit them into different task schedulers. We invoke $LocalScheduler$ for T_{GR} tasks and $ClusterScheduler$ to perform T_{NS} tasks.

(1) $LocalScheduler$ for T_{GR} tasks.

The $LocalScheduler$ module is a thread pool of the local computer, all tasks submitted by $DAGScheduler$ is executed in the thread pool, and the results will then be returned to $DAGScheduler$. We set the locality property value of each T_{GR} as $NODE_LOCAL$ and submit it to a $LocalScheduler$ module. In $LocalScheduler$, all T_{GR} tasks of PRF are allocated to the slave nodes where the corresponding feature subsets are located. These tasks are independent of each other, and there is no synchronization restraint among them. If a feature subset is allocated to multiple slave nodes, the corresponding T_{GR} tasks of each decision tree are allocated to these nodes. And there exist local data communication operations of the tasks among these nodes. If one or more feature subsets are allocated to one slave node, the corresponding T_{GR} tasks are posted to the current node. And there is no data communication operation between the current node and the others in the subsequent computation process.

(2) $ClusterScheduler$ for T_{NS} tasks.

The $ClusterScheduler$ module monitors the execution situation of the computing resources and tasks in the whole Spark cluster and allocates tasks to suitable workers. As mentioned above, T_{NS} tasks are used to collect the results of the corresponding T_{GR} tasks and split the decision tree nodes. T_{NS} tasks are independent of all feature subsets and can be scheduled and allocated in the whole Spark cluster. In addition, these T_{NS} tasks rely on the results of the corresponding T_{GR} tasks, therefore, there is a wait and synchronization restraint for these tasks. Therefore, we invoke the $ClusterScheduler$ to perform T_{NS} tasks. We set the locality property value of each T_{NS} as ANY and submit to a $ClusterScheduler$ module. The task-parallel scheduling scheme for T_{NS} tasks is described in Algorithm 4.3. A diagram of task-parallel scheduling for the tasks in the above DAG is shown in Fig. 7.

Algorithm 4.3 Task-parallel scheduling for T_{NS} tasks

Input:
TS_{NS}: a task set of all T_{NS} submitted by $DAGScheduler$.
Output:
ER_{TS}: the execution results of TS_{NS}.
create $manager \leftarrow$ new TaskSetManager(TS_{NS});
append to taskset manager $activeTSQueue \leftarrow manager$;
if hasReceivedTask == false **then**
  create $starvationTimer \leftarrow$ scheduleAtFixedRate(new TimerTask);
  rank the priority of $TS2 \leftarrow activeTSQueue$.FIFO();
  for each task T_i in $TS2$ **do**
    get available worker $executor_a$ from $workers$;
    $ER_{TS} \leftarrow executor_a$.launchTask($T_i$.taskid);
  end for
end if
return ER_{TS}.

4.3 Parallel Optimization Method Analysis

We discuss our hybrid parallel optimization method from 5 aspects as follows. In comparison with Spark-MLRF and other parallel methods of RF, our hybrid parallel optimization approach of PRF achieves advantages in terms of performance, workload balance, and scalability.

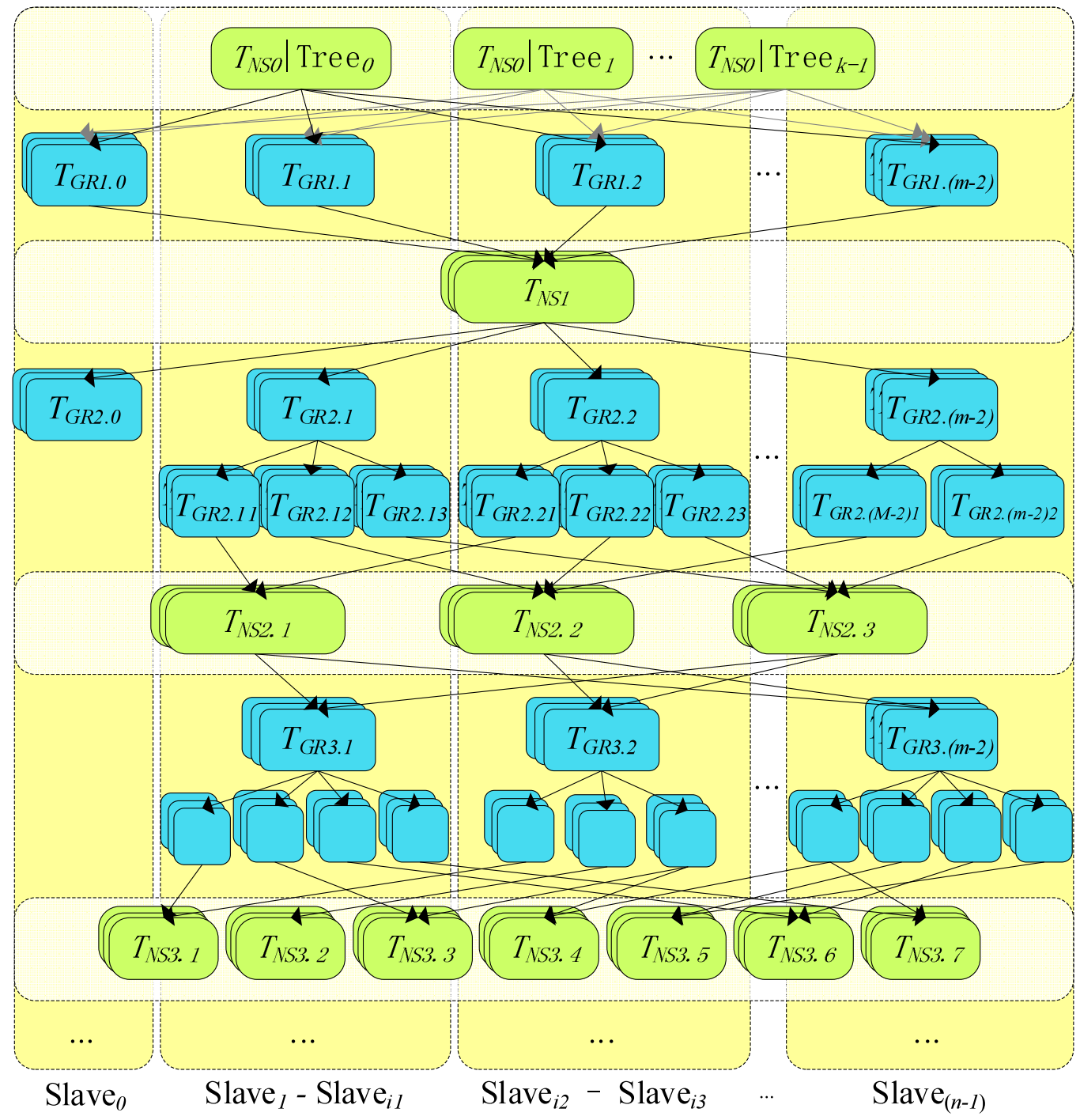

Fig. 7. Task-parallel scheduling based on the DAG in Fig. 6

4.3.1 Computational Complexity Analysis

As discussed in Section 3.4, the total computational complexity of the improved PRF algorithm with dimension-reduction is $O(k(MN + mN \log N))$. After the parallelization of the PRF algorithm on Spark, M features of training dataset are calculated in parallel in the process of dimension-reduction, and k trees are trained concurrently. Therefore, the theoretical computational complexity of the PRF algorithm is $O(\frac{k(MN+mN \log N)}{k \times M}) \approx O(N(\log N + 1))$.

4.3.2 Data Volume Analysis

Taking advantage of the data-multiplexing method, the training dataset is reused effectively. Assume that the volume of the original dataset is $(N \times M)$ and the RF model's scale is k, the volumes of the sampled training dataset in the original RF and Spark-MLRF are both $(N \times M \times k)$. In our PRF, the volume of the sampled training dataset is $(N \times 2 \times (M - 1)) \approx (2NM)$. Moreover, the increase of the scale of PRF does not lead to changes in the data size and storage location. Therefore, compared with the sampling method of the original RF and Spark-MLRF, the data-parallel method of our PRF decreases the total volume of the training dataset for PRF.

4.3.3 Data Communication Analysis

In PRF, there exist data communication operations in the process of data allocation and the training process. Assume that there are n slaves in a Spark cluster, and the data volume of the sampled training dataset is $(2NM)$. In the process of data allocation, the average data communication cost is $(\frac{2MN}{n})$. In the process of the PRF model training, if a feature subset is allocated to several computer nodes, local data communication operations of the subsequent computing tasks occur among these nodes. If one or more feature subsets are allocated to one computer node, there is no

data communication operation among different nodes in the subsequent computation process. Generally, there is a small amount of data communication cost for the intermediate results in each stage of the decision tree's training process. The vertical data-partitioning and static data allocation method mitigates the amount of data communication in the distributed environment and overcomes the performance bottleneck of the traditional parallel method.

4.3.4 Resource and Workload Balance Analysis

From the view point of the entire training process of PRF in the whole Spark cluster, our hybrid parallel optimization approach achieves a better storage and workload balance than other algorithms. One reason is that because the different volumes of feature subsets might lead to different workloads of the T_{GR} tasks for each feature variable, we allocate the feature subsets to the Spark cluster according to its volume. A feature subset with a large volume is allocated to multiple slave nodes. And the corresponding T_{GR} tasks are scheduled among these nodes in parallel. A feature subsets with a small volume are allocated to one slave node. And the corresponding T_{GR} tasks are scheduled on the current node.

A second reason is that with the tree nodes' splitting, the slave nodes where the split variables' feature subsets are located will revert to an idle status. From the view point of the entire training process of PRF, profit from the data-multiplexing method of PRF, each feature subset is shared and reused by all decision trees, and it might be split for different tree nodes in different trees. That is, although a feature subset is split and useless to a decision tree, it is still useful to other trees. Therefore, our PRF not only does not lead to the problem of waste of resources and workload imbalance, but also makes full use of the data resources and achieves an overall workload balance.

4.3.5 Algorithm Scalability Analysis

We discuss the stability and scalability of our PRF algorithm from 3 perspectives. (1) The data-multiplexing method of PRF makes the training dataset be reused effectively. When the scale of PRF expands, namely, the number of decision trees increases, the data size and the storage location of the feature subsets need not change. It only results in an increase in computing tasks for new decision trees and a small amount of data communication cost for the intermediate results of these tasks. (2) When the Spark cluster's scale expands, only a few feature subsets with a high storage load are migrated to the new computer nodes to achieve storage load and workload balance. (3) When the scale of the training dataset increases, it is only necessary to split feature subsets from the new data in the same vertical data-partitioning way, and append each new subset to the corresponding original one. Therefore, we can draw the conclusion that our PRF algorithm with the hybrid parallel optimization method achieves good stability and scalability.

5 EXPERIMENTS

5.1 Experiment Setup

All the experiments are performed on a Spark cloud platform, which is built of one master node and 100 slave nodes. Each node executes in Ubuntu 12.04.4 and has one Pentium (R) Dual-Core 3.20GHz CPU and 8GB memory. All nodes are connected by a high-speed Gigabit network and are configured with Hadoop 2.5.0 and Spark 1.1.0. The algorithm is implemented in Scala 2.10.4. Two groups of datasets with large scale and high dimensionality are used in the experiments. One is from the UCI machine learning repository [33], as shown in Table 3. Another is from a actual medical project, as shown in Table 4.

TABLE 3
Datasets from the UCI machine learning repository

Datasets	Instances	Features	Classes	Data Size (Original)	Data Size (Maximum)
URL Reputation (URL)	2396130	3231961	5	2.1GB	1.0TB
You Tube Video Games (Games)	120000	1000000	14	25.1GB	2.0TB
Bag of Words (Words)	8000000	100000	24	15.8GB	1.3TB
Gas sensor arrays (Gas)	1800000	1950000	15	50.2GB	2.0TB

TABLE 4
Datasets from a medical project

Datasets	Instances	Features	Classes	Data size (Original)	Data size (Maximum)
Patient	279877	25652	18	3.8GB	1.5TB
Outpatient	3657789	47562	9	10.6GB	1.0TB
Medicine	7502058	52460	12	20.4GB	2.0TB
Cancer	3568000	46532	21	5.8GB	2.0TB

In Table 3 and Table 4, $Datasize(Original)$ refers to the original size of the data from the UCI and the project, and $Datasize(Maximum)$ refers to the peak size of data sampled by all of the comparison algorithms.

In the Spark platform, the training data not be loaded into the memory as a whole. Spark can be used to process datasets that are greater than the total cluster memory capacity. RDD objects in a single executor process are accessed by an iteration, and the data are buffered or thrown away after the processing. The cost of memory is very small when there is no requirement of caching the results of the RDD objects. In this case, the results of the iterations are retained in a memory pool by the cache manager. When the data in the memory are not applicable, they will be saved on disk. In this case, part of the data can be kept in the memory and the rest is stored in the disk. Therefore, the training data with the peak size of 2.0TB can be executed on Spark.

5.2 Classification Accuracy

We evaluate the classification accuracy of PRF by comparison with RF, DRF, and Spark-MLRF.

5.2.1 Classification Accuracy for Different Tree Scales

To illustrate the classification accuracy of PRF, experiments are performed for the RF, DRF [18], Spark-MLRF, and PRF algorithms. The datasets are outlined in Table 3 and Table 4. Each case involves different scales of the decision tree. The experimental results are presented in Fig. 8.

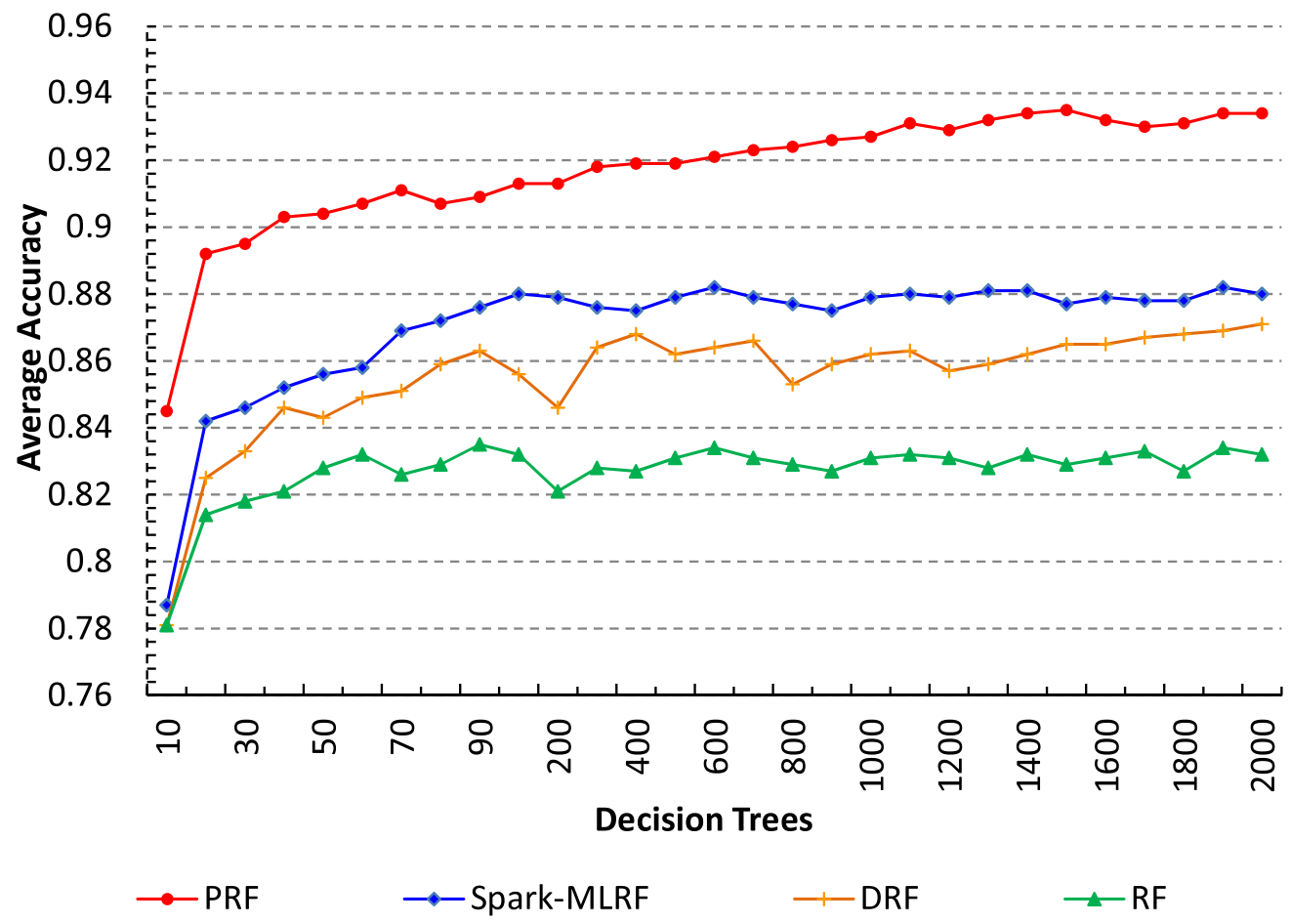

Fig. 8. Average classification accuracy for different tree scales

Fig. 8 shows that the average classification accuracies of all of the comparative algorithms are not high when the number of decision trees is equal to 10. As the number of decision trees increases, the average classification accuracies of these algorithms increase gradually and have a tendency toward a convergence. The classification accuracy of PRF is higher than that of RF by 8.9%, on average, and 10.6% higher in the best case when the number of decision trees is equal to 1500. It is higher than that of DRF by 6.1%, on average, and 7.3% higher in the best case when the number of decision trees is equal to 1300. The classification accuracy of PRF is higher than that of Spark-MLRF by 4.6% on average, and 5.8% in the best case when the number of decision trees is equal to 1500. Therefore, compared with RF, DRF, and Spark-MLRF, PRF improves the classification accuracy significantly.

5.2.2 Classification Accuracy for Different Data Sizes

Experiments are performed to compare the classification accuracy of PRF with the RF, DRF, and Spark-MLRF algorithms. Datasets from the project described in Table 4 are used in the experiments. The experimental results are presented in Fig. 9.

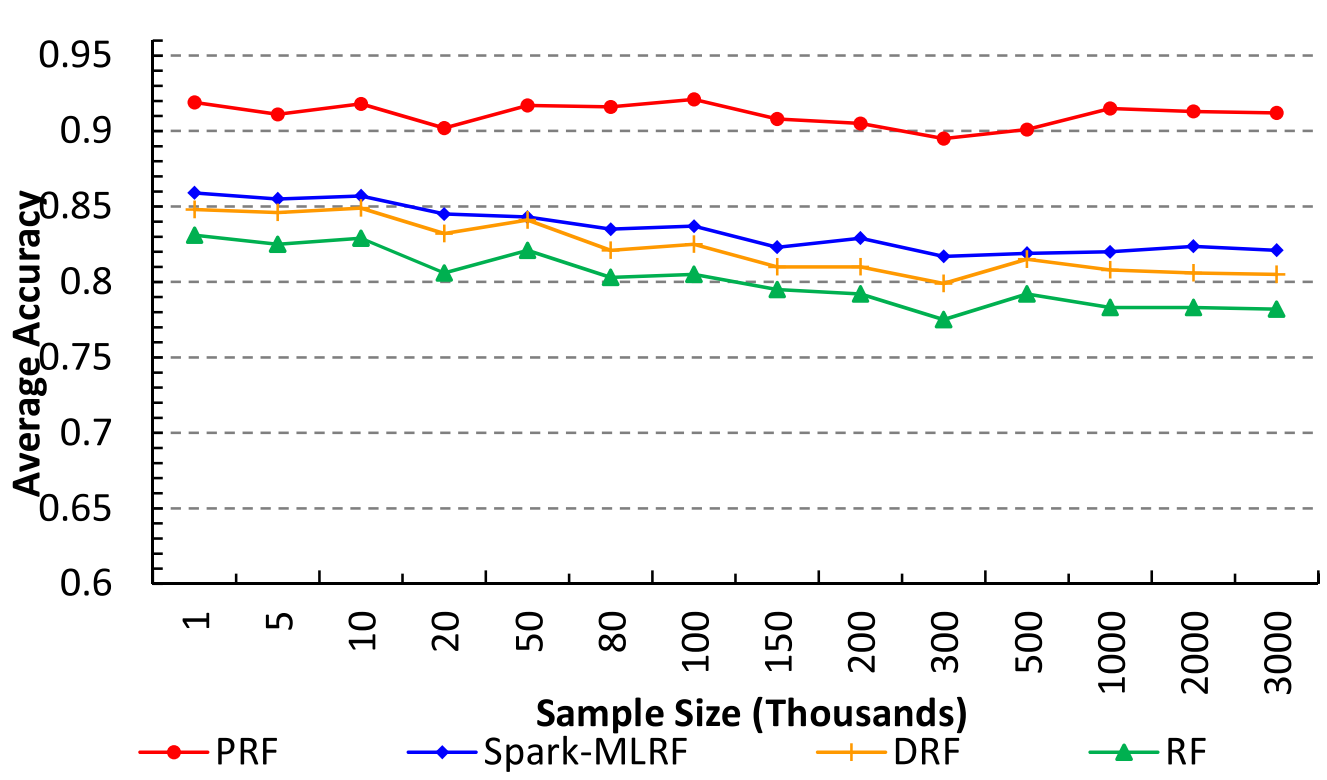

Fig. 9. Average classification accuracy for different data sizes

The classification accuracies of PRF in all of the cases are greater than that of RF, DRF, and Spark-MLRF obviously for each scale of data. The classification accuracy of PRF is greater than that of DRF by 8.6%, on average, and 10.7% higher in the best case when the number of samples is equal to 3,000,000. The classification accuracy of PRF is greater than that of Spark-MLRF by 8.1%, on average, and 11.3% higher in the best case when the number of samples is equal to 3,000,000. For Spark-MLRF, because of the method of sampling for each partition of the dataset, as the size of the dataset increases, the ratio of the random selection of the dataset increases, and the accuracy of Spark-MLRF decreases inevitably. Therefore, compared with RF, DRF, and Spark-MLRF, PRF improves the classification accuracy significantly for different scales of datasets.

5.2.3 OOB Error Rate for Different Tree Scales

We observe the classification error rate of PRF under different conditions. In each condition, the $Patient$ dataset is chosen, and two scales (500 and 1000) of decision trees are constructed. The experimental results are presented in Fig. 10 and Table 5.

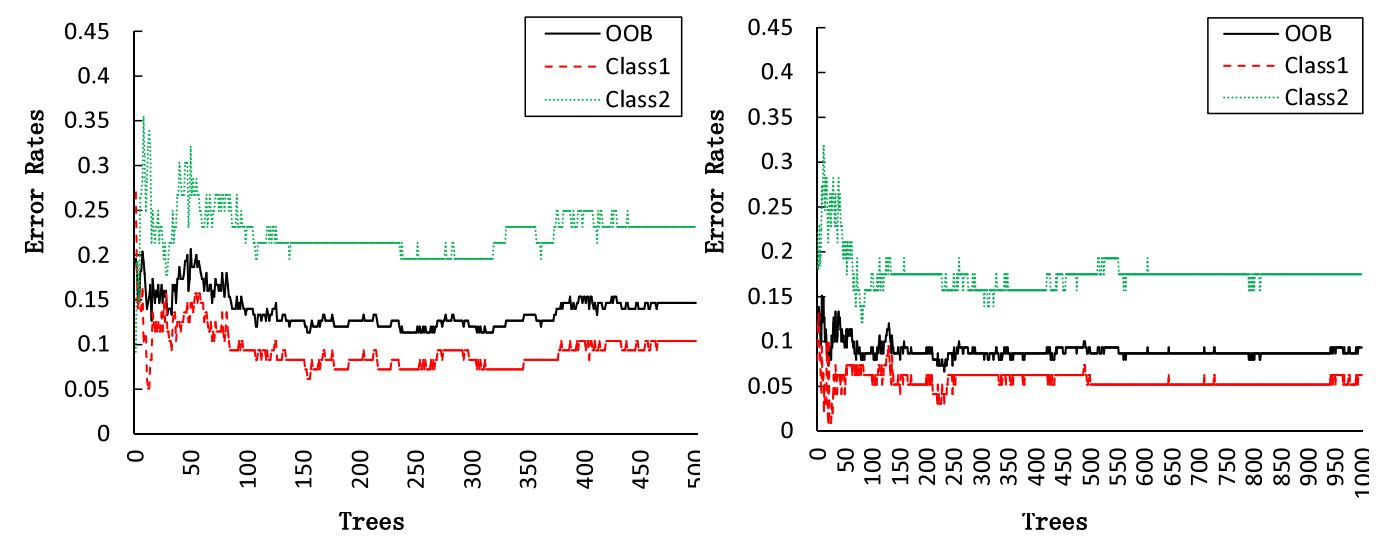

Fig. 10. OOB error rates of PRF for different tree scales

When the number of decision trees of PRF increases, the OOB error rate in each case declines gradually and tends to a convergence condition. The average OOB error rate of PRF is 0.138 when the number of decision trees is equal to 500, and it is 0.089 when the number of decision trees is equal to 1000.

TABLE 5
OOB error rates of PRF for different tree scales

	Tree=500			Tree=1000		
Rate	OOB	Class1	Class2	OOB	Class1	Class2
max	0.207	0.270	0.354	0.151	0.132	0.318
min	0.113	0.051	0.092	0.067	0.010	0.121
mean	0.138	0.094	0.225	0.089	0.056	0.175

5.3 Performance Evaluation

Various experiments are constructed to evaluate the performance of PRF by comparison with the RF and Spark-MLRF algorithms in terms of the execution time, speedup, data volume, and data communication cost.

5.3.1 Average Execution Time for Different Datasets

Experiments are performed to compare the performance of PRF with that of RF and Spark-MLRF. Four groups of training datasets are used in the experiments, such as URL, $Games$, $Outpatient$, and $Patient$. In these experiments, the number of decision trees in each algorithm is both 500, and the number of Spark slaves is 10. The experimental results are presented in Fig. 11.

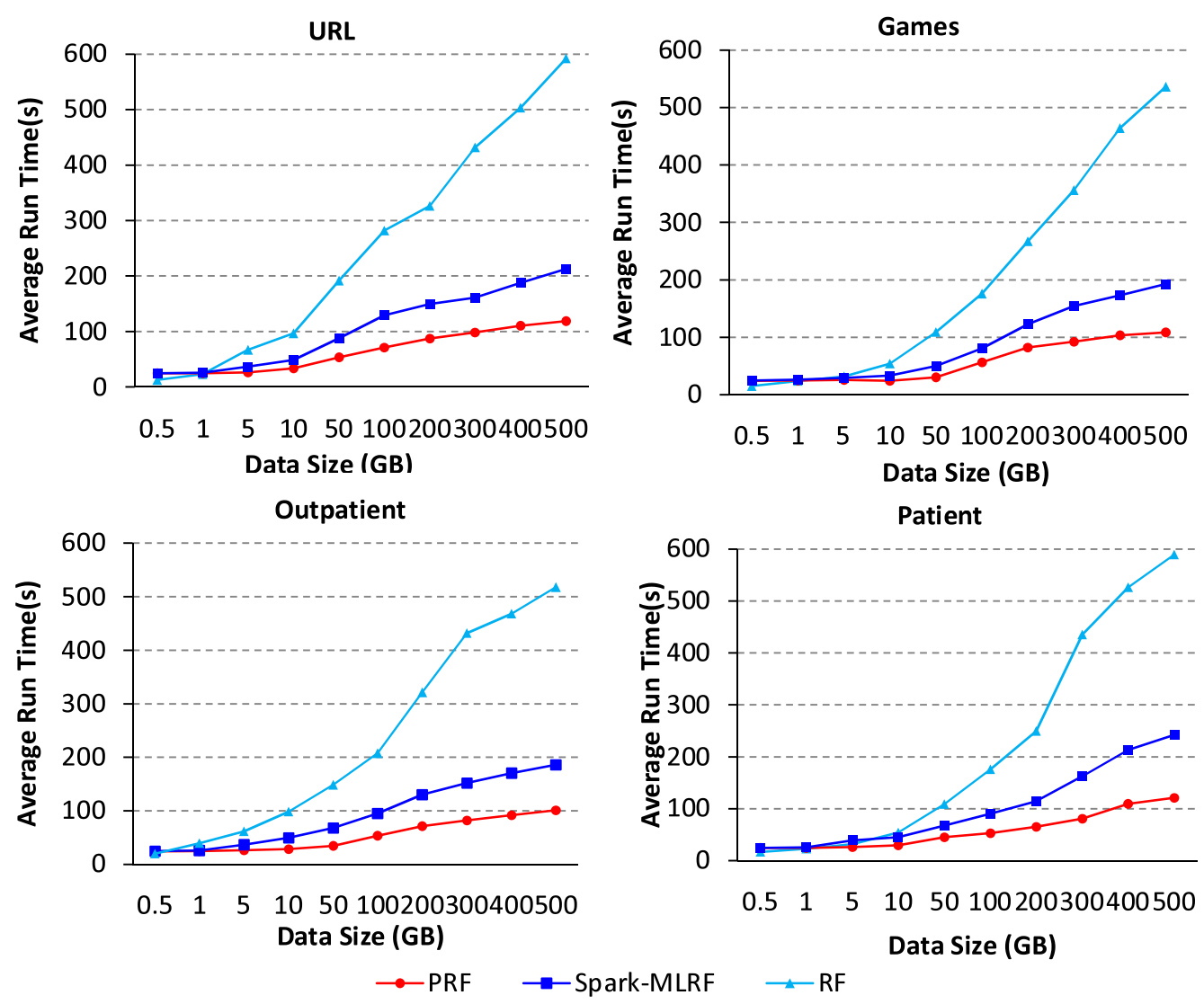

Fig. 11. Average execution time of the algorithms for different datasets

When the data size is small (e.g., less than 1.0GB), the execution times of PRF and Spark-MLRF are higher than that of RF. The reason is that there is a fixed time required to submit the algorithms to the Spark cluster and configure the programs. When the data size is greater than 1.0GB, the average execution times of PRF and Spark-MLRF are less than that of RF in the four cases. For example, in the $Outpatient$ case, when the data size grows from 1.0 to 500.0GB, the average execution time of RF increases from 19.9 to 517.8 seconds, while that of Spark-MLRF increases from 24.8 to 186.2 seconds, and that of PRF increases from 23.5 to 101.3 seconds. Hence, our PRF algorithm achieves a faster processing speed than RF and Spark-MLRF. When the data size increases, the benefit is more noticeable. Taking advantage of the hybrid parallel optimization, PRF achieves significant strengths over Spark-MLRF and RF in terms of performance.

5.3.2 Average Execution Time for Different Cluster Scales

In this section, the performance of PRF on the Spark platform for different scales of slave nodes is considered. The number of slave nodes is gradually increased from 10 to 100, and the experiment results are presented in Fig. 12.

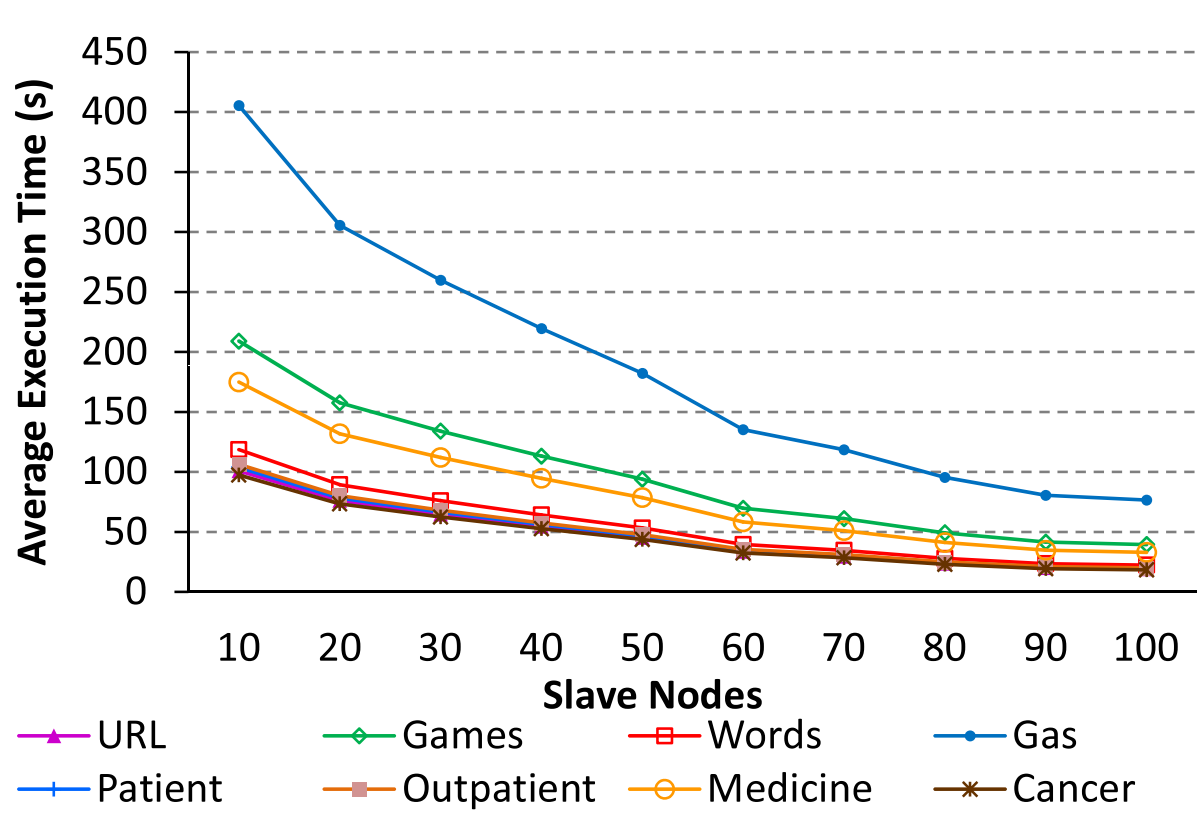

Fig. 12. Average execution time of PRF for different cluster scales

In Fig. 12, because of the different data sizes and contents of the training data, the execution times of PRF in each case are different. When the number of slave nodes increases from 10 to 50, the average execution times of PRF in all cases obviously decrease. For example, the average execution time of PRF decreases from 405.4 to 182.6 seconds in the Gas case and from 174.8 to 78.3 seconds in the $Medicine$ case. By comparison, the average execution times of PRF in the other cases decrease less obviously when the number of slave nodes increases from 50 to 100. For example, the average execution time of PRF decreases from 182.4 to 76.0 seconds in the Gas case and from 78.3 to 33.0 seconds in the $Medicine$ case. This is because when the number of the Spark slaves greater than that of training dataset's feature variables, each feature subset might be allocated to multiple slaves. In such a case, there are more data communication operations among these slaves than before, which leads to more execution time of PRF.

5.3.3 Speedup of PRF in Different Environments

Experiments in a stand-alone environment and a Spark cloud platform are performed to evaluate the speedup of PRF. Because of the different volume of training datasets, the execution times of PRF are not in the same range in different cases. To observe the comparison of the execution time intuitively, a normalization of execution time is taken. Let $T_{(i,sa)}$ be the execution time of PRF for dataset S_i in the stand-alone environment, and first normalized to 1. The execution time of PRF on Spark is normalized as described in Eq. (11):

$$T_i^{'} = \begin{cases} \frac{T_{(i,sa)}}{T_{(i,sa)}} = 1 & Stand-alone, \\ \frac{T_{(i,Spark)}}{T_{(i,sa)}} & Spark. \end{cases} \quad (11)$$

The speedup of PRF on Spark for S_i is defined in Eq. (12):

$$Speedup_{(i,Spark)} = \frac{T^{'}_{(i,Spark)}}{T^{'}_{(i,sa)}}. \quad (12)$$

The results of the comparative analyses are presented in Fig. 13. Taking benefits of the parallel algorithm and cloud environment, the speedup of PRF on Spark tends to increase in each experiment with the number of slave nodes. When the number of slave nodes is equal to 100, the speedup factor of PRF in all cases is in the range of 60.0 - 87.3, which is less than the theoretical value (100). Because there exists data communication time in a distributed environment and a fixed time for the application submission and configuration, it is understandable that the whole speedup of PRF is less than the theoretical value. Due to the different data volumes and contents, the speedup of PRF in each case is different.

When the number of slave nodes is less than 50, the speedup in each case shows a rapid growth trend. For instance, compared with the stand-alone environment, the speedup factor of Gas is 65.5 when the number of slave nodes is equal to 50, and the speedup factor of $Patient$ is 61.5. However, the speedup in each case shows a slow growth trend when the number of slave nodes is greater than 50. This is because there are more data allocation, task

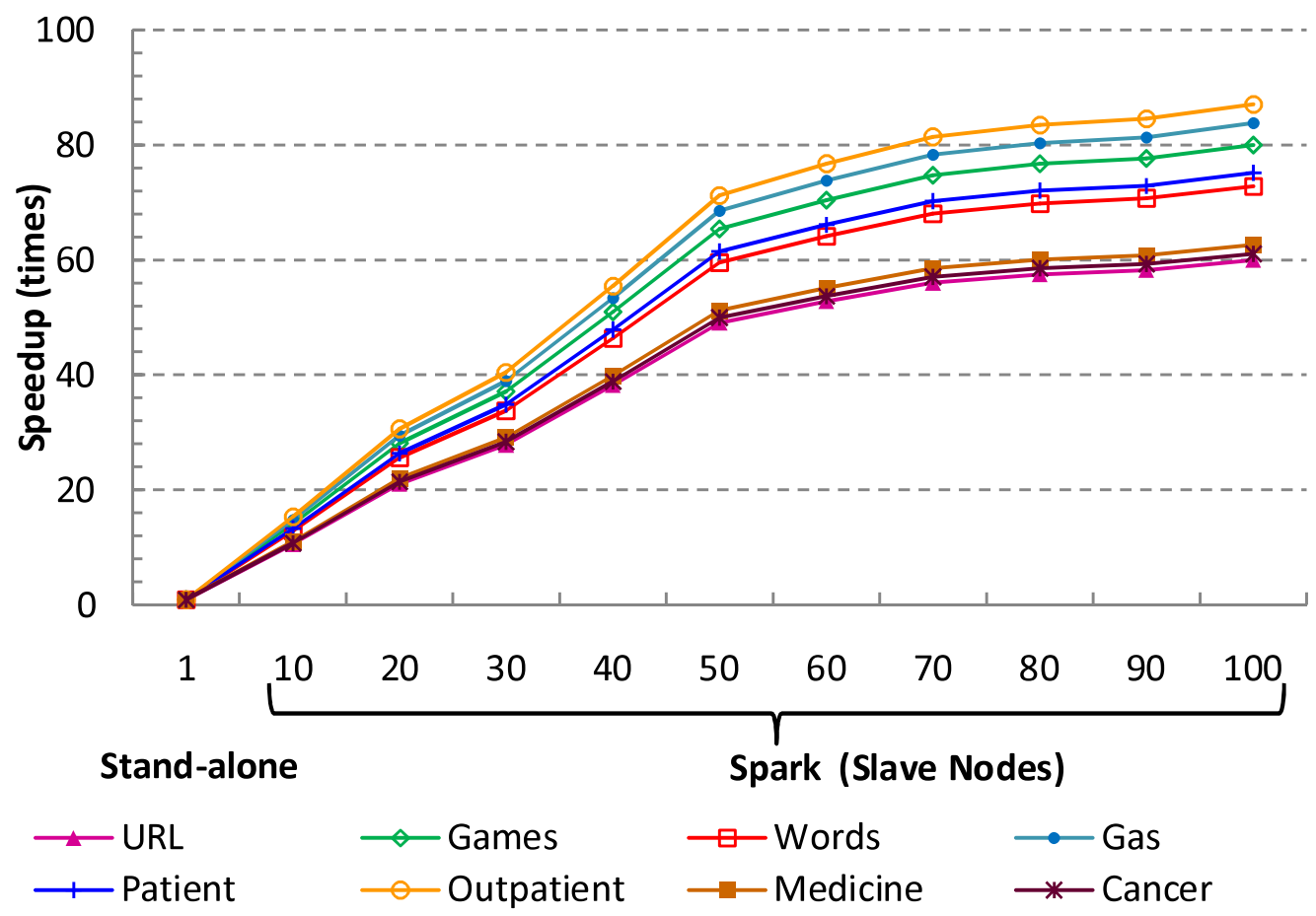

Fig. 13. Speedup of PRF in different environments

scheduling, and data communication operations required for PRF.

5.3.4 Data Volume Analysis for Different RF Scales

We analyze the volume of the training data in PRF against RF and Spark-MLRF. Taking the *Games* case as an example, the volumes of the training data in the different RF scales are shown in Fig. 14.

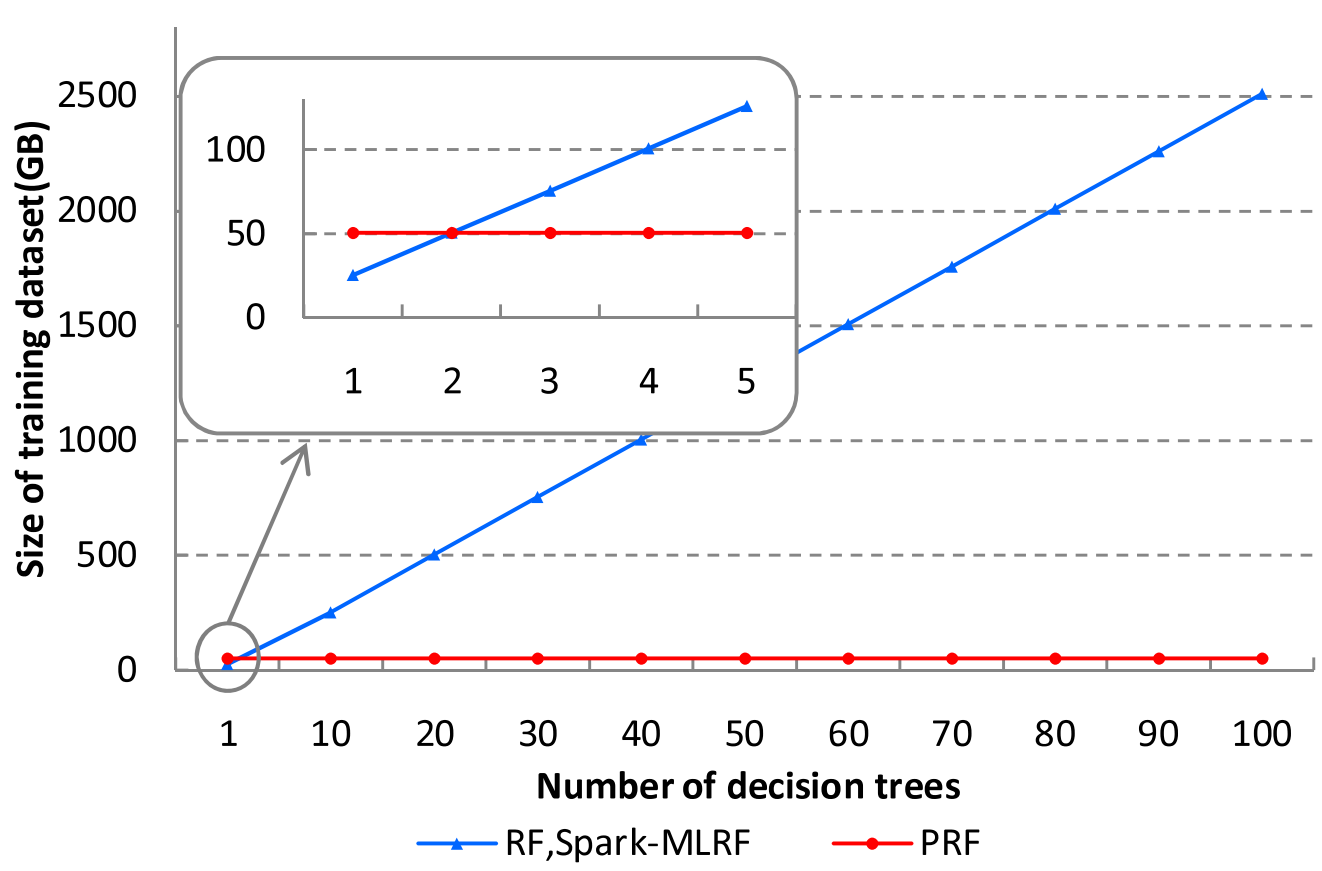

Fig. 14. Size of training dataset for different RF scales

In Fig. 14, due to the use of the same horizontal sampling method, the training data volumes of RF and Spark-MLRF both show a linear increasing trend with the increasing of the RF model scale. Contrary, in PRF, the total volume of all training feature subsets is 2 times the size of the original training dataset. Making use of the data-multiplexing approach of PRF, the training dataset is effectively reused. When the number of decision trees is larger than 2, despite the expansion of RF scale, the volume of the training data will not increases any further.

5.3.5 Data Communication Cost Analysis

Experiments are performed for different scales of the Spark cluster to compare the Data Communication Cost (C_{DC}) of PRF with that of Spark-MLRF. The suffer-write size of slave nodes in the Spark cluster is monitored as the C_{DC} of

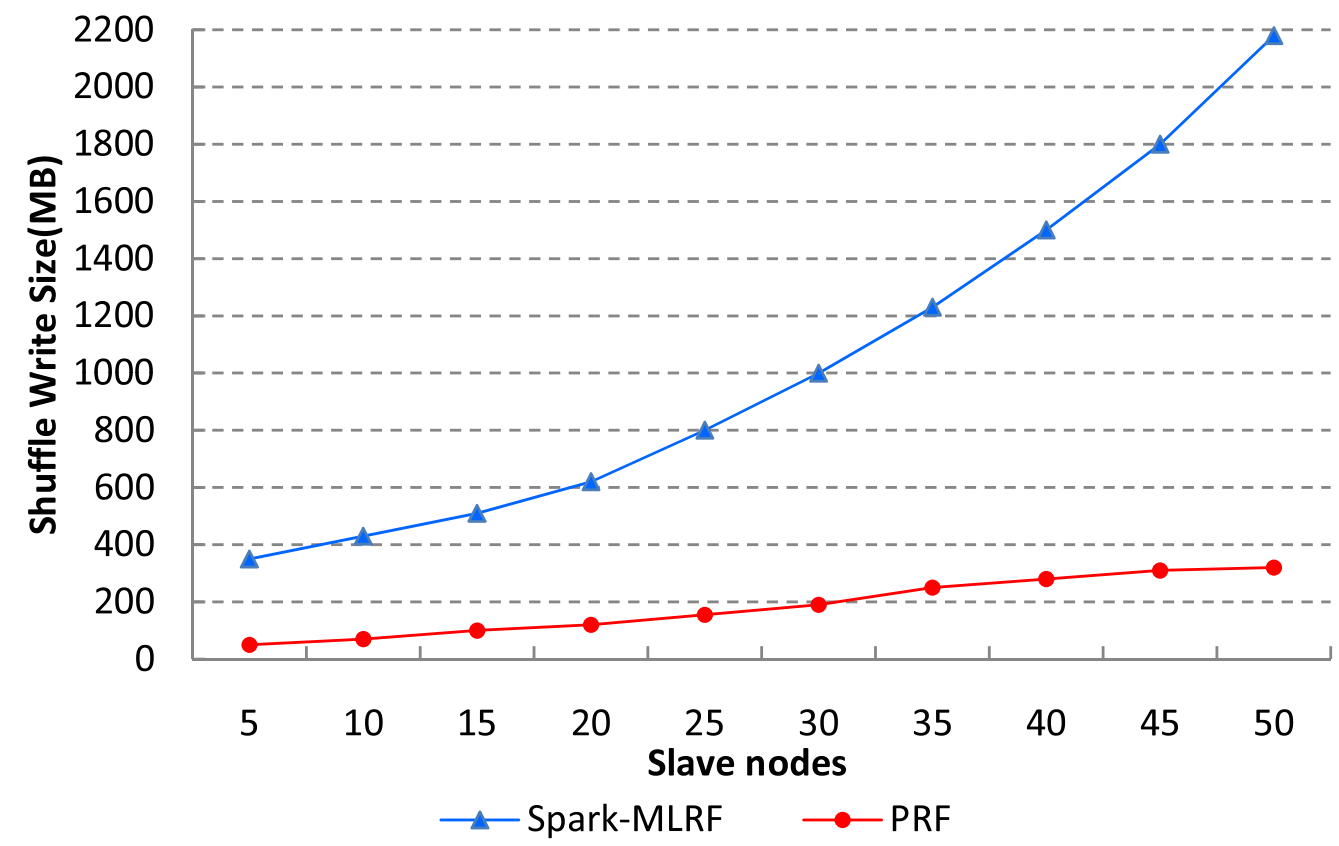

Fig. 15. Data communication costs of PRF and Spark-MLRF

the algorithms. Taking the $Patient$ case as an example, the results of the comparison of C_{DC} are presented in Fig. 15.

From Fig. 15, it is clear that the C_{DC} of PRF are less than that of Spark-MLRF in all cases, and the distinction is larger with increasing number of slave nodes. Although Spark-MLRF also uses the data-parallel method, the horizontal partitioning method for training data makes the computing tasks have to frequent access data across different slaves. As the number of slaves increases from 5 to 50, the C_{DC} of Spark-MLRF increases from 350.0MB to 2180.0MB. Different from Spark-MLRF, in PRF, the vertical data-partitioning and allocation method and the task scheduling method make the most of the computing tasks (T_{GR}) access data from the local slave, reducing the amount of data transmission in the distributed environment. As the number of slaves increases from 5 to 50, the C_{DC} of PRF increases from 50.0MB to 320.0MB, which is much lower than that of Spark-MLRF. Therefore, PRF minimizes the C_{DC} of RF in a distributed environment. The expansion of the cluster's scale does not lead to an obviously increase in C_{DC}. In conclusion, our PRF achieves a superiority and notable advantages over Spark-MLRF in terms of stability and scalability.

6 Conclusions

In this paper, a parallel random forest algorithm has been proposed for big data. The accuracy of the PRF algorithm is optimized through dimension-reduction and the weighted vote approach. Then, a hybrid parallel approach of PRF combining data-parallel and task-parallel optimization is performed and implemented on Apache Spark. Taking advantage of the data-parallel optimization, the training dataset is reused and the volume of data is reduced significantly. Benefitting from the task-parallel optimization, the data transmission cost is effectively reduced and the performance of the algorithm is obviously improved. Experimental results indicate the superiority and notable strengths of PRF over the other algorithms in terms of classification accuracy, performance, and scalability. For future work, we will focus on the incremental parallel random forest algorithm for data streams in cloud environment, and improve the data allocation and task scheduling mechanism for the algorithm on a distributed and parallel environment.

Acknowledgment

The research was partially funded by the Key Program of National Natural Science Foundation of China (Grant Nos. 61133005, 61432005), the National Natural Science Foundation of China (Grant Nos. 61370095, 61472124, 61202109, 61472126,61672221), the National Research Foundation of Qatar (NPRP, Grant Nos. 8-519-1-108), and the Natural Science Foundation of Hunan Province of China (Grant Nos. 2015JJ4100, 2016JJ4002).

References

[1] X. Wu, X. Zhu, and G.-Q. Wu, "Data mining with big data," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 26, no. 1, pp. 97–107, January 2014.

[2] L. Kuang, F. Hao, and Y. L.T., "A tensor-based approach for big data representation and dimensionality reduction," *Emerging Topics in Computing, IEEE Transactions on*, vol. 2, no. 3, pp. 280–291, April 2014.

[3] A. Andrzejak, F. Langner, and S. Zabala, "Interpretable models from distributed data via merging of decision trees," in *Computational Intelligence and Data Mining (CIDM), 2013 IEEE Symposium on*. IEEE, 2013, pp. 1–9.

[4] P. K. Ray, S. R. Mohanty, N. Kishor, and J. P. S. Catalao, "Optimal feature and decision tree-based classification of power quality disturbances in distributed generation systems," *Sustainable Energy, IEEE Transactions on*, vol. 5, no. 1, pp. 200–208, January 2014.

[5] Apache, "Hadoop," Website, June 2016, http://hadoop.apache.org.

[6] S. del Rio, V. Lopez, J. M. Benitez, and F. Herrera, "On the use of mapreduce for imbalanced big data using random forest," *Information Sciences*, vol. 285, pp. 112–137, November 2014.

[7] K. Singh, S. C. Guntuku, A. Thakur, and C. Hota, "Big data analytics framework for peer-to-peer botnet detection using random forests," *Information Sciences*, vol. 278, pp. 488–497, September 2014.

[8] Apache, "Spark," Website, June 2016, http://spark-project.org.

[9] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, October 2001.

[10] G. Wu and P. H. Huang, "A vectorization-optimization-method-based type-2 fuzzy neural network for noisy data classification," *Fuzzy Systems, IEEE Transactions on*, vol. 21, no. 1, pp. 1–15, February 2013.

[11] H. Abdulsalam, D. B. Skillicorn, and P. Martin, "Classification using streaming random forests," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 23, no. 1, pp. 22–36, January 2011.

[12] C. Lindner, P. A. Bromiley, M. C. Ionita, and T. F. Cootes, "Robust and accurate shape model matching using random forest regression-voting," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 25, no. 3, pp. 1–14, December 2014.

[13] X. Yun, G. Wu, G. Zhang, K. Li, , and S. Wang, "Fastraq: A fast approach to range-aggregate queries in big data environments," *Cloud Computing, IEEE Transactions on*, vol. 3, no. 2, pp. 206–218, April 2015.

[14] M. Xu, H. Chen, and P. K. Varshney, "Dimensionality reduction for registration of high-dimensional data sets," *Image Processing, IEEE Transactions on*, vol. 22, no. 8, pp. 3041–3049, August 2013.

[15] Q. Tao, D. Chu, and J. Wang, "Recursive support vector machines for dimensionality reduction," *Neural Networks, IEEE Transactions on*, vol. 19, no. 1, pp. 189–193, January 2008.

[16] Y. Lin, T. Liu, and C. Fuh, "Multiple kernel learning for dimensionality reduction," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 33, no. 6, pp. 1147–1160, June 2011.

[17] C. Strobl, A. Boulesteix, T. Kneib, and T. Augustin, "Conditional variable importance for random forests," *BMC Bioinformatics*, vol. 9, no. 14, pp. 1–11, 2007.

[18] S. Bernard, S. Adam, and L. Heutte, "Dynamic random forests," *Pattern Recognition Letters*, vol. 33, no. 12, pp. 1580–1586, September 2012.

[19] T. M. Khoshgoftaar, J. V. Hulse, and A. Napolitano, "Comparing boosting and bagging techniques with noisy and imbalanced data," *Systems, Man and Cybernetics, IEEE Transactions on*, vol. 41, no. 3, pp. 552–568, May 2011.

[20] G. Yu, N. A. Goussies, J. Yuan, and Z. Liu, "Fast action detection via discriminative random forest voting and top-k subvolume search," *Multimedia, IEEE Transactions on*, vol. 13, no. 3, pp. 507–517, June 2011.

[21] G. Biau, "Analysis of a random forests model," *Journal of Machine Learning Research*, vol. 13, no. 1, pp. 1063–1095, January 2012.

[22] J. D. Basilico, M. A. Munson, T. G. Kolda, K. R. Dixon, and W. P. Kegelmeyer, "Comet: A recipe for learning and using large ensembles on massive data," in *IEEE International Conference on Data Mining*, October 2011, pp. 41–50.

[23] K. M. Svore and C. J. Burges, "Distributed stochastic aware random forests efficient data mining for big data," in *Big Data (BigData Congress), 2013 IEEE International Congress on*. Cambridge University Press, 2013, pp. 425–426.

[24] B. Panda, J. S. Herbach, S. Basu, and R. J. Bayardo, "Planet: Massively parallel learning of tree ensembles with mapreduce," *Proceedings of the Vldb Endowment*, vol. 2, no. 2, pp. 1426–1437, August 2009.

[25] S. Tyree, K. Q. Weinberger, and K. Agrawal, "Parallel boosted regression trees for web search ranking," in *International Conference on World Wide Web*, March 2011, pp. 387–396.

[26] D. Warneke and O. Kao, "Exploiting dynamic resource allocation for efficient parallel data processing in the cloud," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 22, no. 6, pp. 985–997, June 2011.

[27] L. Mashayekhy, M. M. Nejad, D. Grosu, Q. Zhang, and W. Shi, "Energy-aware scheduling of mapreduce jobs for big data applications," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 26, no. 3, pp. 1–10, March 2015.

[28] L. D. Briceno, H. J. Siegel, A. A. Maciejewski, M. Oltikar, and J. Brateman, "Heuristics for robust resource allocation of satellite weather data processing on a heterogeneous parallel system," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 22, no. 11, pp. 1780–1787, February 2011.

[29] F. Zhang, J. Cao, W. Tan, S. Khan, K. Li, and A. Zomaya, "Evolutionary scheduling of dynamic multitasking workloads for big-data analytics in elastic cloud," *Emerging Topics in Computing, IEEE Transactions on*, vol. 2, no. 3, pp. 338–351, August 2014.

[30] K. Li, X. Tang, B. Veeravalli, and K. Li, "Scheduling precedence constrained stochastic tasks on heterogeneous cluster systems," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 64, no. 1, pp. 191–204, January 2015.

[31] Y. Xu, K. Li, J. Hu, and K. Li, "A genetic algorithm for task scheduling on heterogeneous computing systems using multiple priority queues," *Information Sciences*, vol. 270, no. 6, pp. 255–287, June 2014.

[32] A. Spark, "Spark mllib - random forest," Website, June 2016, http://spark.apache.org/docs/latest/mllib-ensembles.html.

[33] U. of California, "Uci machine learning repository," Website, June 2016, http://archive.ics.uci.edu/ml/datasets.

Jianguo Chen received the Ph.D. degree in College of Computer Science and Electronic Engineering at Hunan University, China. He was a visiting Ph.D. student at the University of Illinois at Chicago from 2017 to 2018. He is currently a postdoctoral in University of Toronto and Hunan University. His major research areas include parallel computing, cloud computing, machine learning, data mining, bioinformatics and big data. He has published research articles in international conference and journals of data-mining algorithms and parallel computing, such as *IEEE TPDS*, IEEE/ACM TCBB, and *Information Sciences*.

Kenli Li received the Ph.D. degree in computer science from Huazhong University of Science and Technology, China, in 2003. He was a visiting scholar at University of Illinois at Urbana-Champaign from 2004 to 2005. He is currently a full professor of computer science and technology at Hunan University and director of National Supercomputing Center in Changsha. His major research areas include parallel computing, high-performance computing, grid and cloud computing. He has published more than 180 research papers in international conferences and journals, such as *IEEE-TC*, *IEEE-TPDS*, *IEEE-TSP*, *JPDC*, *ICPP*, *CCGrid*. He is an outstanding member of CCF. He is a senior member of the IEEE and serves on the editorial board of *IEEE Transactions on Computers*.

Zhuo Tang received the Ph.D. in computer science from Huazhong University of Science and Technology, China, in 2008. He is currently an associate professor of Computer Science and Technology at Hunan University. His research interests include security model, parallel algorithms, and resources scheduling for distributed computing systems, grid and cloud computing. He is a member of ACM and CCF.

Kashif Bilal received his PhD from North Dakota State University USA. He is currently a post-doctoral researcher at Qatar University, Qatar. His research interests include cloud computing, energy efficient high speed networks, and robustness.Kashif is awarded CoE Student Researcher of the year 2014 based on his research contributions during his doctoral studies at North Dakota State University.

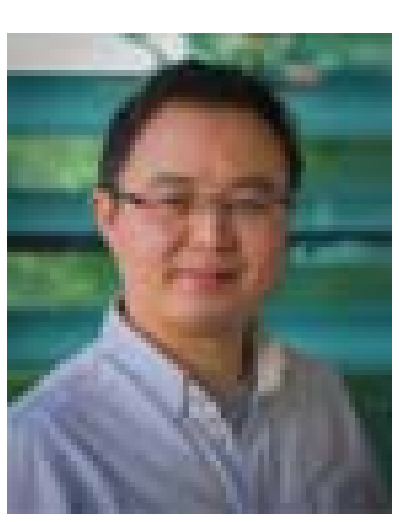

Shui Yu is currently a Senior Lecturer of School of Information Technology, Deakin University. He is a member of Deakin University Academic Board (2015-2016), a Senior Member of IEEE, and a member of AAAS, the vice chair of Technical Subcommittee on Big Data Processing, Analytics, and Networking of IEEE Communication Society. He is currently serving the editorial boards of *IEEE TPDS*, *IEEE CST*, *IEEE Access*.

Chuliang Weng is a principal researcher at Huawei Shannon Lab. He received his Ph.D. from Shanghai Jiao Tong University in 2004, and from 2011 to 2012, he was a visiting scientist with the Department of Computer Science at Columbia University in the City of New York. His research interests include parallel and distributed systems, operating systems and virtualization, and storage systems. He is a member of the IEEE, ACM and CCF.

Keqin Li is a SUNY Distinguished Professor of computer science in the State University of New York. His current research interests include parallel computing and high-performance computing, distributed computing, energy-efficient computing and communication, heterogeneous computing systems, cloud computing, big data computing, CPU-GPU hybrid and cooperative computing, multi-core computing, storage and file systems, wireless communication networks, sensor networks, peer-to-peer file sharing systems, mobile computing, service computing, Internet of things and cyber-physical systems. He has published over 590 journal articles, book chapters, and refereed conference papers, and has received several best paper awards. He is currently serving or has served on the editorial boards of IEEE Transactions on Parallel and Distributed Systems, IEEE Transactions on Computers, IEEE Transactions on Cloud Computing, IEEE Transactions on Services Computing, and IEEE Transactions on Sustainable Computing. He is an IEEE Fellow.