Multiobjective Task Scheduling for Energy-Efficient Cloud Implementation of Hyperspectral Image Classification

Jin Sun, *Member, IEEE*, Heng Li, Yi Zhang, Yang Xu, *Member, IEEE*, Yaoqin Zhu, Qitao Zang, Zebin Wu, *Senior Member, IEEE*, and Zhihui Wei

***Abstract*—Cloud computing has become a promising solution to efficient processing of remotely sensed big data, due to its high-performance and scalable computing capabilities. However, existing cloud solutions generally involve the problems of low resource utilization and high energy consumption when processing large-scale remote sensing datasets, affecting the quality-of-service of the cloud system. Aiming at hyperspectral image classification applications, this article proposes an energy-efficient cloud implementation by employing a multiobjective task scheduling algorithm. We first present a parallel computing mechanism for a fusion-based classification method based on Apache Spark. With the general classification flow represented by a workflow model, we formulate a multiobjective scheduling framework that jointly minimizes the total execution time as well as energy consumption. We further develop an effective scheduling algorithm to solve the multiobjective optimization problem and produce a set of Pareto-optimal solutions, providing the tradeoff between computational efficiency and energy efficiency. Experimental results demonstrate that the multiobjective scheduling approach proposed in this work can substantially reduce the execution time and energy consumption for performing large-scale hyperspectral image classification on Spark. In addition, our proposed algorithm can generate better tradeoff solutions to the multiobjective scheduling problem as compared to competing scheduling algorithms.**

***Index Terms*—Energy consumption, hyperspectral image classification, makespan, multiobjective optimization, task scheduling.**

I. Introduction

HYPERSPECTRAL remote sensing has been popularly used in a variety earth observation fields such as environment monitoring, object identification, and military defense [1]–[3]. Hyperspectral sensors can now simultaneously measure hundreds of contiguous spectral bands with high spectral resolution. Due to the large data volume and high computational complexity of hyperspectral image applications, the processing of hyperspectral data, which generally involves computation- and data-intensive operations, naturally becomes a big data problem [4]. Motivated by the increasing demand for hyperspectral big data analytics, there emerges many research studies oriented toward the parallel implementation of hyperspectral image applications on cloud computing architectures [5]–[7]. The fundamental idea is to use the distributed file system to cope with the storage of hyperspectral big data, and utilize the distributed computing scheme to support the intensive computation during the processing flow.

In this article, we intend to illustrate the use of cloud computing architectures to tackle the problem of large-scale hyperspectral data processing. We focus on the problem of hyperspectral image classification, which is one of the most important and representative techniques for hyperspectral image interpretation. Recently, spatial-spectral classification methods [8]–[11] have demonstrated their advantages by exploiting spatial-contextual information in combination with spectral information via a preprocessing or postprocessing procedure. For instance, Lu *et al.* [12] introduced a general fusion-based approach for hyperspectral image classification that uses different types of features to achieve highly discriminative information. Despite the promising discriminative capability, this category of classification methods in general have very high computational complexity due to their complicated processing flows, and would suffer from extremely long computation time when processing large-scale hyperspectral datasets. As hyperspectral data volume continues to increase, computational efficiency is of greater importance to processing massive hyperspectral data by using these advanced techniques. Considering the heavy computational load from hyperspectral applications as well as the scalable computing capabilities provided by cloud computing, it is necessary to develop cloud-based solutions to accelerate the processing of large-scale hyperspectral datasets.

As revealed in a recent study, task scheduling strategies are of great importance to improving the utilization rate of cloud computing resources and in turn the computational efficiency of remote sensing applications [13]. The underlying idea is to characterize the application flow as a workflow model consisting

Manuscript received June 30, 2020; revised October 10, 2020; accepted November 3, 2020. Date of publication November 10, 2020; date of current version January 6, 2021. This work was supported in part by the National Natural Science Foundation of China under Grant 61872185 and Grant 61772274, in part by the Natural Science Foundation of Jiangsu Province of China under Grant BK20180018, in part by the Fundamental Research Funds for the Central Universities under Grant 30919011402 and Grant 30920021132, and in part by the Jiangsu Planned Projects for Postdoctoral Research Funds under Grant 2019K025. *(Corresponding authors: Yi Zhang; Heng Li.)*

Jin Sun, Heng Li, Yi Zhang, Yang Xu, Yaoqin Zhu, Zebin Wu, and Zhihui Wei are with the School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing 210094, China (e-mail: sunj@njust.edu.cn; liheng@njust.edu.cn; yzhang@njust.edu.cn; xuyangth90@gmail.com; zhuyaoqin@163.com; zebin.wu@gmail.com; gswei@njust.edu.cn).

Qitao Zang is with the Hangzhou Hikvision Digital Technology Co., Ltd., Hangzhou 310051, China (e-mail: zangqitao@163.com).

Digital Object Identifier 10.1109/JSTARS.2020.3036896

of many subtasks, e.g., directed acyclic graph (DAG), and exploit the parallelisms within and among tasks through an optimized assignment of tasks onto computing resources. In this study, the attention of task scheduling is limited to minimizing the completion time of all tasks, which is formally defined as the application's *makespan* [14]. However, energy consumption is another challenging problem and becomes more and more severe in today's cloud datacenters [15]–[17]. As reported in the literature, the electricity demand by datacenters accounts for approximately one percent of all electricity consumed worldwide [18]. Therefore, it is crucial to incorporate the energy efficiency concern into task scheduling algorithms for executing the remote sensing applications on cloud computing architectures.

In this article, we propose an energy-efficient cloud implementation of hyperspectral image classification by means of a multiobjective task scheduling algorithm. We first present a cloud implementation of a fusion-based classification method based on Apache Spark. By employing the MapReduce distributed computing scheme [19], we develop the parallel computing mechanisms for the N-FINDR hyperspectral image unmixing method [20] and the support vector machine (SVM) classifier [21], which are two main operations during the classification process. With the general classification flow represented by a DAG model, we formulate a multiobjective scheduling framework that jointly minimizes the total execution time as well as energy consumption by determining the most appropriate task assignment solution. We further develop an effective scheduling algorithm based on the immune algorithm (IA) optimization framework [22], [23] to solve the multiobjective optimization problem. The proposed multiobjective IA (MOIA) scheduling strategy is capable of producing a set of tradeoff solutions, providing the tradeoff between computational efficiency and energy efficiency. Specifically, the main contributions of this article can be summarized as follows.

1) We develop a parallel implementation to accelerate the N-FINDR unmixing method and the SVM classifier in a fusion-based hyperspectral image classification application on Apache Spark.
2) We formulate a multiobjective scheduling framework that incorporates both the execution time and energy consumption for processing the classification flow as the optimization objectives.
3) We propose an effective algorithm MOIA to solve the multiobjective scheduling problem and produce Pareto-optimal solutions to achieve the joint optimization of computational efficiency and energy efficiency.

We perform extensive experiments to justify the efficacy of our proposed scheduling method. Experimental results demonstrate that MOIA can substantially reduce the execution time and energy consumption for hyperspectral image classification. In addition, MOIA can generate high-quality tradeoff solutions as compared to competing scheduling algorithms.

The reminder of this article is organized as follows. Section II discusses related work. Section III presents the cloud implementation of a fusion-based classifier for hyperspectral image classification. Sections IV and V detail the formulation of the multiobjective scheduling model and the proposed MOIA algorithm, respectively. Section VI presents performance evaluation results. Finally, Section VII concludes this article.

II. Related Work

This section provides a brief discussion about existing cloud-based solutions to facilitating the processing of remotely sensed big data on cloud computing architectures. In addition, since the cloud implementation in this work is based on a task scheduling strategy, we also briefly discuss those generally used scheduling strategies that are closely related to the proposed multiobjective scheduling algorithm.

A. Cloud-Based Remote Sensing Big Data Processing

With the development of remote sensing satellites and sensing instruments, the amount of remotely sensed data is increasing at an extremely fast velocity. The traditional single-machine platform can no longer cope with the processing of large-scale remote sensing datasets, due to the lack of storage and computing capabilities. Cloud computing has the superior advantages of distributed storage and scalable computing capabilities, and therefore, provides an ideal platform for efficient processing of remote sensing big data applications.

Quirita *et al.* [24] proposed a distributed framework for supervised classification of remote sensing big data on a cloud computing architecture. This framework consists of three abstraction layers and supports the definition and implementation of applications by researchers in different fields. By allocating storage and computing resources depending on the data volume, this framework can effectively handle large-scale remote sensing datasets. Pan and Zhang [25] developed a cloud system based on Hadoop for remote sensing image analysis. On the one hand, this system redefines the data structures that specifically fit into the features of remote sensing images. On the other hand, this systems integrates several Java programs that are implemented based on the MapReduce scheme, such that the remote sensing applications can be adapted to the cloud computing environment. Haut *et al.* [26] presented a cloud implementation of the popular K-means algorithm for hyperspectral image analysis. Taking the advantages of cloud computing architecture, this implementation can process massive hyperspectral image data sets efficiently.

The abovementioned approaches focus on solely exploiting data-level parallelism by data partitioning, and therefore, cannot tackle remote sensing applications with complicated workflow structure, such as the fusion-based hyperspectral image classifier studied in this work. Sun *et al.* [13] first proposed using scheduling strategies to exploit task-level parallelism for enhancing the computational efficiency. However, this scheduling strategy does not take into account the consideration of energy consumption. In a distinct manner, the multiobjective scheduling algorithm in this work attempts to minimize both the makespan of hyperspectral image classification and the energy consumed by the cloud computing resources simultaneously. The scheduling solution obtained by our scheduling algorithm is beneficial for achieving the overall quality-of-service of the cloud system.

B. Task Scheduling in Cloud Computing Environments

Cloud computing can effectively solve the problem of massive data processing by means of data partitioning and distributed computing. In cloud computing environments, task scheduling strategies are of vital importance to the performance of the cloud system and have been intensively studied in the literature. Task scheduling is to determine the optimal assignment of tasks onto virtual machines (VMs), which are the essential processing elements in cloud computing, with various scheduling objectives including load balancing, makespan minimization, energy efficiency, etc. Task scheduling strategies can be generally classified into two categories: batch scheduling algorithms and metaheuristic scheduling algorithms. Batch schedulers arrange all tasks in a certain order and allocate tasks in a sequential order. Typical batch scheduling algorithms include first come first service [27], short job first [28], and round robin. Although this type of scheduling algorithms are easy to implement and cost effective, they are in general incapable of remote sensing applications due to the huge data volume and complicated processing flow. Alternatively, metaheuristic algorithms are more preferred for solving task scheduling problems in this scenario.

It is worth discussing several multiobjective scheduling algorithms that rely on swarm intelligence algorithms, a category of metaheuristic algorithms, to solve scheduling problems in cloud computing environments. For instance, Gabi *et al.* [29] proposed a multiobjective task scheduling algorithm based on cat swarm optimization and simulated annealing algorithm. This approach uses execution time, execution cost, and service quality as indicators jointly to analyze the scheduling performance. Zheng *et al.* [30] presented a multiobjective scheduling method based on an improved differential evolution (DE) algorithm. By setting adaptive parameters and redefining crossover and selection operators, the improved DE algorithm can address the limitation of slow convergence rate in traditional DE algorithms. Alkayal *et al.* [31] proposed an multiobjective task scheduling approach based upon the particle swarm optimization (PSO) framework. The optimization objectives are to minimize waiting time and to maximize system throughput. In this multiobjective PSO algorithm, a new ranking strategy is incorporated to prioritize tasks when allocating tasks onto VMs.

In this article, by analyzing the workflow of the parallel implementation of a fusion-based classification application, we develop a IA-based multiobjective task scheduling algorithm to jointly optimization the application's makespan and energy consumption. IA is a metaheuristic algorithm with advantages in population diversity, robustness, and global convergence, and thereby is suitable for solving the multiobjective scheduling problem studied in this work with high-quality tradeoff solutions.

III. Cloud Implementation of a Fusion-Based Classification Method

This section studies a fusion-based hyperspectral image classification method proposed by Lu *et al.* [12] and develops a cloud implementation to support its distributed processing on Spark. This method extracts multiple features from hyperspectral image at subpixel level, pixel level, and super-pixel level, respectively. The kernels of the obtained features at three different levels are fused to form a composite kernel, which is incorporated with an SVM classifier to classify each pixel of the hyperspectral image. This fusion-based classifier takes full advantage of multiple features at different levels, leading to high classification accuracy. However, due to the large amount of hyperspectral data and massive matrix calculation, heavy computation burden would be incurred when applying this method to process large-scale datasets.

In this article, we develop the following strategies to facilitate the distributed processing of this fusion-based classifier on Spark. On the one hand, in the step of subpixel-level feature extraction, we replace the original constrained energy minimization algorithm [32] by the N-FINDR unmixing model. N-FINDR is one of the most widely used unmixing algorithms in the field of hyperspectral imagery. Its main idea is to calculate the largest simplex volume in the feature space of hyperspectral data. The endmembers on the vertices of the simplex volume can be identified as the endmember set required for hyperspectral image classification. N-FINDR is very effective in hyperspectral image unmixing especially when the number of endmembers to be extracted is determined. More importantly, the unmixing flow of N-FINDR involves many operations that can be performed in parallel and can be easily adapted for cloud implementation [33]. On the other hand, at model training phase, SVM algorithm is in requirement of grid search and cross validation to determine the optimal combination of hyperparameters. Considering that there are many repeated and independent operations in this procedure, we can implement parallel search relying on the distributed computing scheme in cloud computing. It is worth mentioning that, other unmixing methods and classifiers, such as vertex component analysis [34], also can be incorporated into this cloud implementation, as long as the computation load of these methods can be divided and distributed across multiple computing nodes to facilitate parallel processing.

A. Parallel Implementation of N-FINDR Unmixing Model

The N-FINDR unmixing model mainly consists of the following three parts:

1) perform dimensionality reduction on the original hyperspectral data;
2) perform endmember extraction on the dimensionality-reduced image data to obtain the endmember set;
3) calculate the corresponding abundance maps based on the endmember set.

Fig. 1 summarizes the parallel processing flow of N-FINDR unmixing on Spark. In what follows, we describe the implementation details for these three main procedures separately.

We use the principle component analysis (PCA) algorithm to reduce the dimensionality of hyperspectral data. First, we use Spark's resilient distributed datasets (RDDs) [35] to generate the RDD data `DataRDD` based on the original image matrix `Data`. As illustrated in Fig. 1, each data partition is denoted by `DataX` where $X = 1, 2, \ldots, K$ and K is the number of partitions. On a Spark cluster consisting of a driver node and a set of executor nodes, we perform a "map" operation on all RDD partitions in `DataRDD` to obtain `CovRDD` at each executor

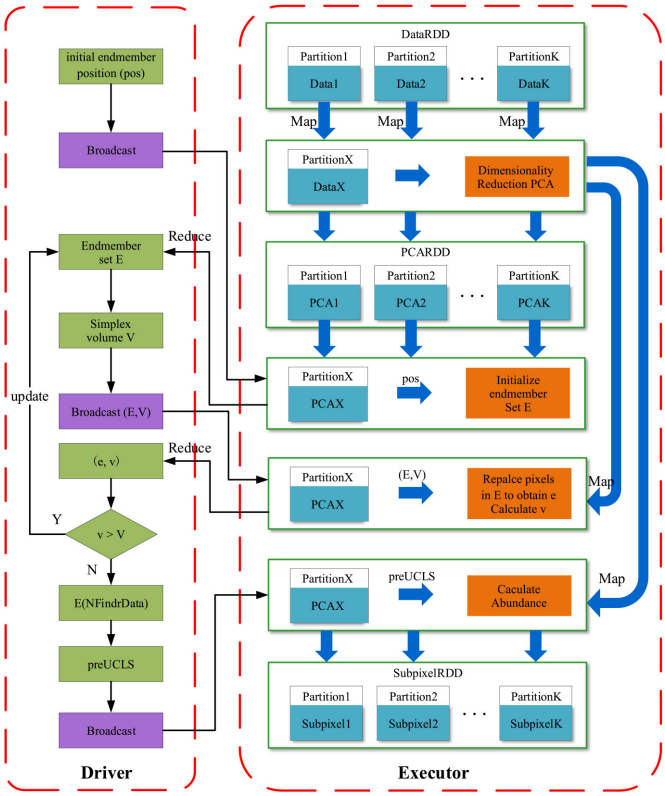


Fig. 1. Parallel implementation of N-FINDR unmixing model on Spark.

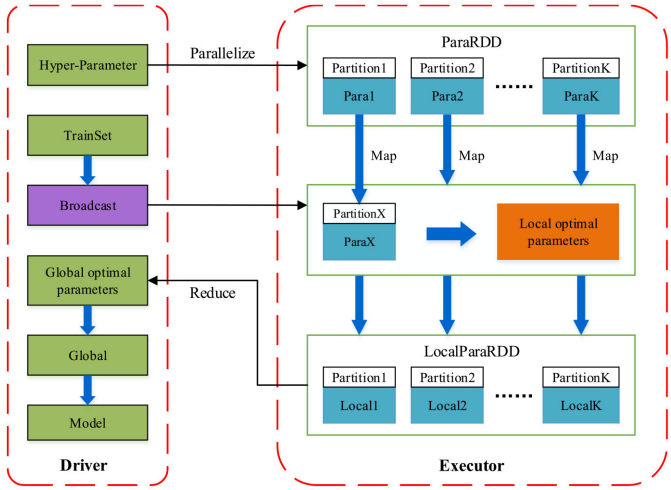


Fig. 2. Parallel implementation of SVM parameter tuning on Spark.

node, and perform a "reduce" operation on the results submitted by executor nodes to obtain the corresponding covariance matrix Cov. We then calculate the feature vector $V_{m\times L}$, in which m and L stand for the number of dimensions and the number of bands, respectively, on the driver node and broadcast it to all executors. Finally, we perform a "map" operation on `DataRDD` again to obtain `PCARDD`, i.e., the hyperspectral data after dimensionality reduction.

We randomly generate an array pos of size M to represent the initial endmember position, where M is the number of endmembers in the hyperspectral image. We then invoke the `Broadcast()` function to broadcast the endmember position pos. Once all executor nodes receive the information of pos, MapReduce operations are performed on `PCARDD`, in order to extract the pixel information stored in pos from the dimensionality-reduced data. The initial endmember set E and the corresponding simplex volume V are broadcast after they have been determined. We use `PCAX` $(X = 1, 2, \ldots, K)$ to denote each partition of `PCARDD`, as shown in Fig. 1. The next step is to iteratively perform an updating operation, during which the pixel information in `PCAX` will be included in the endmember set E to obtain an updated set e and the corresponding largest simplex volume v in set e. If $v > V$, the simplex volume V and the endmember set E should be updated by an "collect" operation. During each updating operation, the up-to-date simplex volume V and endmember set E need to be submitted to the driver node and will be broadcast again. When the iteration terminates, we can obtain the final endmember set after N-FINDR unmixing on the driver node, which is denoted by `NFindrData`.

To estimate the abundance maps, we follow the original N-FINDR model [36] and apply unconstrained least squares (UCLS) to unmix the pixels in the endmember set. To be specific, we determine endmember abundance fractions by calculating $(E^T E)^{-1} E^T \times$ `PCARDD` where E is the endmember set (i.e., `NFindrData`). To avoid redundant computations on executor nodes, we first calculate $(EE^T)^{-1}E$ on the driver node and broadcast this preprocessing result, denoted by `preUCLS`, to all executor nodes. We then perform a "map" operation on `PCARDD` such that each executor node can execute a matrix multiplication operation on the partitioned data `PCAX`. The final result `preUCLS` $\times$ `PCARDD`^{T} is the corresponding abundance maps for the hyperspectral data, i.e., the subpixel-level features `SubpixelFeature`.

B. Parallel Implementation of SVM Classifier

In the fusion-based classification method, SVM classifier is used to predict the label of each hyperspectral image pixel into a set of predefined classes. The most important step in SVM is to determine the optimal model parameters at the training stage, which have direct impact on the classification accuracy at the prediction stage. Due to the high computational complexity of parameter tuning, this step is very time-consuming for large-scale hyperspectral datasets. Specifically, in this work, we use the grid search method to investigate each combination of the regularization coefficient C and the kernel parameter γ, evaluate the classification accuracy by means of cross validation, and determine the optimal parameter combination leading to the highest accuracy.

We rely on the distributed computing scheme in cloud computing to parallelize the parameter tuning procedure. Fig. 2 presents the parallel processing flow of SVM parameter tuning on Spark. Considering that the two parameters C and γ can be tuned independently, we partition the parameter matrix and distribute the partitioned submatrices over all computing nodes. Each executor node performs a grid search on the assigned submatrix to determine the locally optimal parameter combination

on this node. After all executor nodes finish the grid search procedure, the driver node determines the globally optimal parameter combination among all locally optimal results submitted by executor nodes. let a and b denote the numbers of candidate values for parameters C and γ, respectively. The main steps of the distributed tuning procedure are summarized as follows.

1) Broadcast the training set `TrainSet` to all k executor nodes.
2) Partition the parameter matrix `Hyper-Parameter` into k submatrices. Each executor node converts the submatrix into a RDD dataset that is denoted by `ParaX` ($X = 1, 2, \ldots, K$). The converted RDD datasets on all executor nodes form `ParaRDD`.
3) Use grid search and cross validation on each executor node to determine the best parameter combination $(C_i,\ \Gamma_j)$ $(i < a,\ j < b)$, which is denoted by `LocalX`.
4) Perform a "collect" operation on all RDD partitions of `LocalParaRDD`, and select the best parameter combination in `LocalParaRDD` to generate the SVM model.

IV. Scheduling Model Formulation

This section formulates the multiobjective scheduling model for joint optimization of computational efficiency and energy efficiency of hyperspectral image classification on Spark. We first present the DAG model for characterizing the workflow of the parallel method implemented in Section III, which is the input to the scheduling framework. We, then, present the models for the application's makespan, i.e., the completion time of all tasks, and energy consumption, and finally presents the formulation of the multiobjective scheduling model.

A. Workflow Model

Referring to the parallel implementation of hyperspectral image classification, like other general-purpose applications, the distributed processing flow can be regarded as a workflow application that can be characterized by a DAG structure. Formally, a DAG can be represented by $G = (T, E, W)$, where $T = \{T_1, T_2, \ldots, T_n\}$ represents the set of n tasks belonging to a workflow application, $E = \{e_{ij} | 1 \le i \le n, 1 \le j \le n\}$ represents the set of communication edges among tasks, and $W = \{W_1, W_2, \ldots, W_n\}$ represents the amount of computational load associated with each task. Constrained by the precedence relations specified by the communication edges, each task node in the DAG structure has one or more predecessor and successor nodes. For each communication edge connecting two task nodes, the successor node cannot start its execution until the execution of all its predecessor nodes has completed.

Fig. 3 illustrates the DAG model for the distributed processing flow of the fusion-based classification algorithm. Based on this DAG structure, we can formulate the multiobjective optimization model for minimizing both task completion time and total energy consumption, and employ an efficient task scheduling algorithm to find an optimized solution of task assignment. Note that "N-FINDR" and "SVM Training" are two parallelizable tasks that can be processed in parallel on multiple computing nodes. For these two tasks, the execution times and in turn the corresponding energy consumptions are not fixed, but vary depending on the number of computing nodes assigned for processing them in parallel.

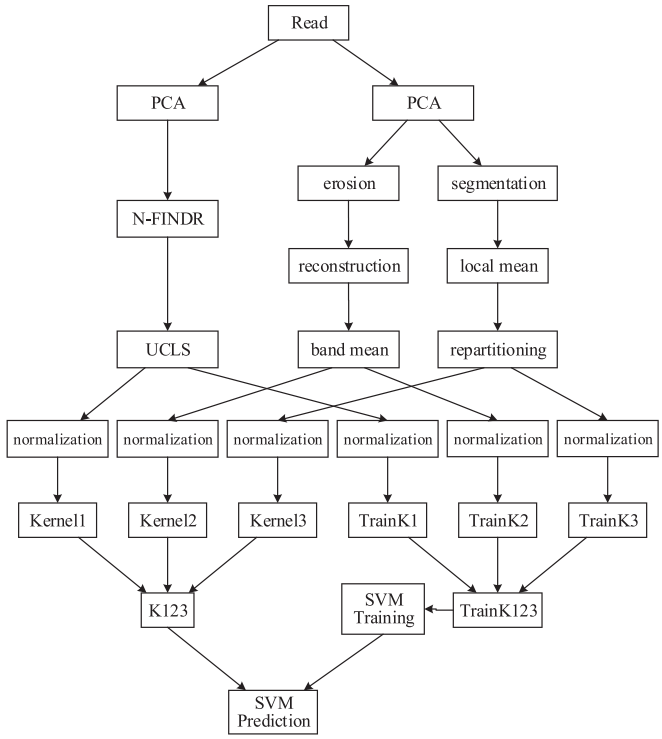


Fig. 3. DAG model for the cloud implementation of the fusion-based classification algorithm

B. Makespan Model

Assume that the tasks to be allocated onto the computing nodes are denoted by $(t_1, t_2, \ldots, t_i)$, and the corresponding execution times or durations are $(D_1, D_2, \ldots, D_i)$. Assume further that there are j available computing nodes $(v_1, v_2, \ldots, v_j)$. We use $T_{\text{start}}^{(m,n)}$ and $T_{\text{end}}^{(m,n)}$ to indicate the start time and completion time of the nth task $(1 \le n \le i)$ on the mth $(1 \le m \le j)$ computing node v_m, respectively. In addition, we use T_{exec}^{m} to indicate v_m's total operation time for processing all tasks assigned to it. Apparently, task t_n's start time $T_{\text{start}}^{(m,n)}$ is constrained by the precedence relations and can be calculated by

$$T_{\text{start}}^{(m,n)} = \begin{cases} 0, \text{ if } n = 1 \\ \max\left\{\max\left\{\text{Pred}(t_n)\right\}, T_{\text{end}}^{(m,p)}\right\}, \text{ otherwise} \end{cases} \tag{1}$$

where $T_{\text{end}}^{(m,p)}$ $(1 \le p \le n \le i)$ denotes the completion time of a task that is also assigned to v_m and is scheduled for execution ahead of t_n. $\text{Pred}(t_n)$ records the completion times of all t_n's predecessor tasks. Obviously, the start time of the first task $T_{\text{start}}^{(m,1)}$ is 0, since it has no predecessor nodes. As indicated by (1), the start time of a specific task is affected by its predecessor tasks, as it has to wait until all its predecessor tasks have completed execution before its execution can start. To conclude, the actual task start time is dependent on not only the completion

time of its prior task in the task queue, but also on the latest completion time of its predecessor tasks. Once task start time can be decided, task completion time $T_{\text{end}}^{(m,n)}$ can be accordingly determined as

$$T_{\text{end}}^{(m,n)} = T_{\text{start}}^{(m,n)} + D_n. \tag{2}$$

The completion time of all tasks, i.e., the makespan of the whole application flow, is exactly the maximum value among all task completion times, i.e.,

$$\mathbb{C} = \max_{n=1,2,..,i} \left\{ T_{\text{end}}^{(m,n)} \right\}. \tag{3}$$

Since task execution time D_n in (3) is a varying value depending on the number of partitioned tasks it is divided into, different task scheduling solutions would lead to various makespan values.

In addition, the total operation time of each computing node T_m is the sum of task execution times over all tasks assigned to this node, which is given by

$$T_{\text{exec}}^{m} = \sum_{t_k \text{ is assigned to } v_m} D_k. \tag{4}$$

C. Energy Model

The energy consumption of the computing nodes on Spark includes the static portion and the dynamic portion. The static energy denotes the energy consumption when the computing node is idle and not executing any computation tasks. In contrast, the dynamic energy denotes the energy consumed for executing tasks assigned to the computing node. Following the energy model in [37] and [38], there is generally a linear correlation between power consumption and CPU utilization. Accordingly, the total energy consumption in the scheduling model can be defined as

$$E_{\text{total}} = Ps \times \sum_{m=1}^{j} ts_m + \sum_{m=1}^{j} T_{\text{exec}}^{m} \times Pd \times \tau_m \tag{5}$$

where Ps represents the static power per unit time (watts), Pd represents the dynamic power per unit time (watts), E_{total} is the total energy consumption generated by the computing nodes when all tasks have completed execution (joules), τ_m is the CPU utilization of computing node, and ts_m denotes the idle time of the kth computing nodes during task execution, which can be calculated as

$$ts_m = \mathbb{C} - T_{\text{exec}}^{m}. \tag{6}$$

D. Scheduling Model

Before presenting the formulation of the scheduling model, we provide the fundamental concepts of multiobjective optimization. In a multiobjective optimization problem, there are multiple optimization objectives that cannot both reach their optimum values by an individual solution. Pareto-optimality is introduced to ensure the best overall performance [39], [40]. A solution is called a Pareto-optimal solution (or nondominated solution) if there exists no other solutions such that at least one optimization objective has a better value while the remaining objective values are the same or better. In the space of optimization objectives, the surface consisting of all the points generated by all Pareto-optimal solutions is called the Pareto frontier.

The multiobjective scheduling problem studied in this work is to determine a set of Pareto-optimal scheduling solutions such that the makespan of the hyperspectral image classification on Spark and the total energy consumed by all computing nodes. When allocating the tasks of the classification flow to the computing nodes, we need to take into account the number of available computing nodes and the precedence constraints upon tasks. The optimization model for the multiobjective scheduling problem is formulated in

$$\min \max_{n=1,2,..,i} \left\{ T_{\text{start}}^{(m,n)} + D_n \right\} \tag{7}$$

$$\min Ps \times \sum_{m=1}^{j} (\mathbb{C} - T_{\text{exec}}^{m}) + \sum_{m=1}^{j} T_{\text{exec}}^{m} \times Pd \times \tau_m \tag{8}$$

$$\text{s.t.} \arg\min\{n|i-n\}, 1 \le n \le j. \tag{9}$$

V. Multiobjective Immune Algorithm

This section presents our proposed MOIA algorithm that employs the IA optimization framework to solving the multiobjective scheduling problem. IA is an intelligent optimization algorithm, in which the optimization objective corresponds to the antigen in the immune response, the feasible solution corresponds to the antibody, and the quality of feasible solution corresponds to the affinity between antibody and antigen. Specifically, IA first generates an initial antibody population, and then updates the antibody population in an iterative manner according to the affinity value of each antibody. The update operations (immune operations) include antibody cloning, antibody mutation, and immune inhibiting. When the termination condition is satisfied, the optimal solution to the optimization problem will be obtained. Fig. 4 illustrates the flowchart of traditional IA algorithm. In what follows, we discuss in detail these fundamental procedures in our proposed MOIA algorithm, including the immune operators, solution encoding scheme, fitness calculation, and crowding distance calculation methods.

A. Solution Encoding

The solution encoding scheme not only impacts most evolutionary operators of IA, such as the crossover operator, mutation operator, and cloning operator. But also exerts great influence on IA's solution exploration capability. In this work, we use a binary string composed of binary symbols 0 and 1 to represent the allocation of a specific task. The kernel issue in binary encoding is how to represent the number of computing nodes assigned for task execution as well as the states of the computing nodes by the binary variables in the binary string. Due to the limited computing resources in the cloud platform, there is a maximum number of computing nodes that can be designated for any task. We use a task sequence $(t_1, t_2, \ldots, t_i)$ to denote the binary strings for all tasks, and $(v_1, v_2, \ldots, v_j)$ to denote the set of computing nodes, in which j stands for the total number of computing nodes available for task execution. The state of each computing node is either 1 and 0, indicating whether it is in operation or idle.

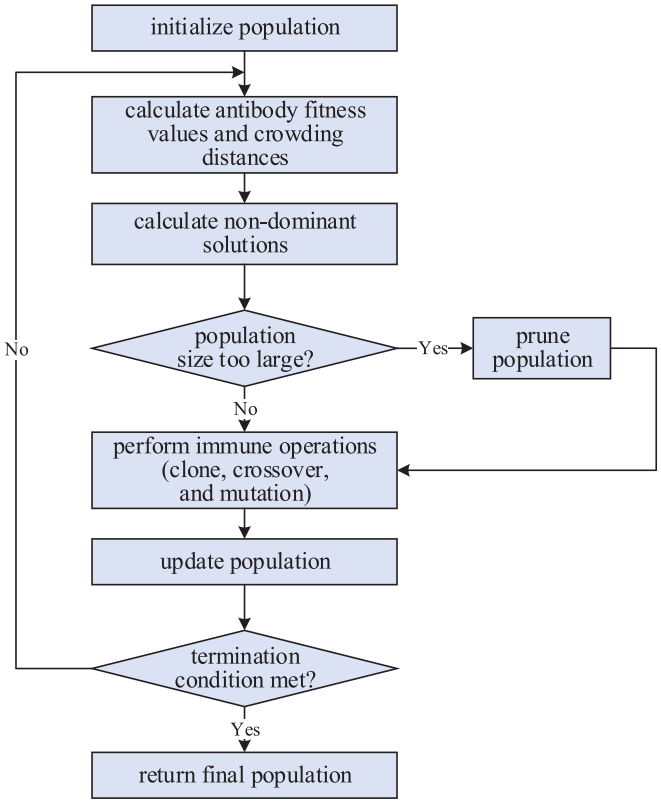


Fig. 4. Flowchart of traditional IA algorithm

Since each task can execute on one or more computing nodes, there could be $2^j - 1$ valid binary strings for a task that can be executed in parallel, and j valid binary strings for a nonparallelizable task. For example, with 8 computing nodes deployed, $t_k = 11101011$ $(1 \leq k \leq i)$ is a valid string to represent the allocation of a parallelizable task onto computing nodes, indicating that computing nodes v_1, v_2, v_3, v_5, v_7, and v_8 are designated for executing this task. We define the number of designated computing nodes as the parallelism of a specific task (i.e., the number of "1"s in the binary string). We can, then, determine task execution time according to its parallelism. For a task that cannot be executed in parallel, since it can only execute on a single computing node, there would be one and only one "1" in the corresponding binary string, and its parallelism is simply one. The binary strings for all parallelizable and nonparallelizable tasks form a task sequence $(t_1, t_2, \ldots, t_i)$, which is considered as a scheduling solution in this work. When the number of computing nodes varies, we can adjust the length of the binary string to comply with the new solution representation.

B. Fitness Calculation

In our MOIA framework, each task sequence consisting of the binary strings for all tasks $X = (t_1, t_2, \ldots, t_i)$ is regarded as an immune antibody in the population. We use Algorithm 1 to calculate the fitness value of an immune antibody. By decoding each binary string in the given task sequence, we can determine the parallelism and the execution time of the corresponding task, and in turn can determine the total operation time and complete time of each computing node. Finally, we can calculate the makespan and total energy consumption by (3) and (5), respectively, to evaluate the fitness of this particular scheduling solution.

Algorithm 1: Fitness Calculation.

Input: an immune antibody $X = (t_1, t_2, \cdots, t_i)$;
Output: Total energy consumption and makespan;
initialize the task set that has been performed $preTask \leftarrow \Phi$, computing node's operation time $T_{\text{exec}} \leftarrow [0, 0, \cdots, 0]$, and task completion time $T_{\text{end}} \leftarrow [0, 0, \cdots, 0]$;
for $k = 1$ *to* i **do**
  get the binary string corresponding to t_k;
  if *(the current task is the predecessor of a task in the* $preTask$*)* **then**
    determine task execution time T_{Dura} according to the number of computing nodes assigned for t_k;
    update computing node's operation time $T_{\text{exec}}[i] = T_{\text{exec}}[i] + T_{\text{Dura}}$;
    update task complete time $T_{\text{end}}[i] = T_{\text{end}}[i] + T_{\text{Dura}}$;
    add task t_k into $preTask$;
  else
    identify all t_k's predecessor tasks in $preTask$;
    calculate the maximum completion time of all these predecessor tasks, and denote it by $T_{\text{end}}^{\max}$;
    determine task execution time T_{Dura} according to the number of computing nodes assigned for t_k;
    update computing node's operation time $T_{\text{exec}}[i] = T_{\text{end}}^{\max} + T_{\text{Dura}}$;
    update task complete time $T_{\text{end}}[i] = T_{\text{end}}^{\max} + T_{\text{Dura}}$;
    add task t_k into $preTask$;
calculate the makespan and total energy consumption by Eqs. (3) and (5);
return makespan and energy consumption by Eqs. (3) and (5);

C. Crowding Distance Calculation

Crowding distance [41] is used to evaluate the individual density around an individual antibody i in population D, and can be defined as the mean distance between this antibody individual and other individuals on each objective function. Fig. 5 provides an illustrative example of crowding distance for the biobjective scenario. To be precise, the calculation the crowding distance [41] is given by

$$I(d, D) = \sum_{i=1}^{k} \frac{I_i(d, D)}{f_i^{\max} - f_i^{\min}} \tag{10}$$

$$I_i(d, D) = \begin{cases} \infty, \text{ if } f_i(d) = \min\limits_{d' \in D} \{f_i(d')\} \text{ or } \max\limits_{d' \in D} \{f_i(d')\} \\ \min\limits_{d', d'' \in D: f_i(d'') < f_i(d) < f_i(d')} \{f_i(d') - f_i(d'')\}, \text{ otherwise} \end{cases} \tag{11}$$

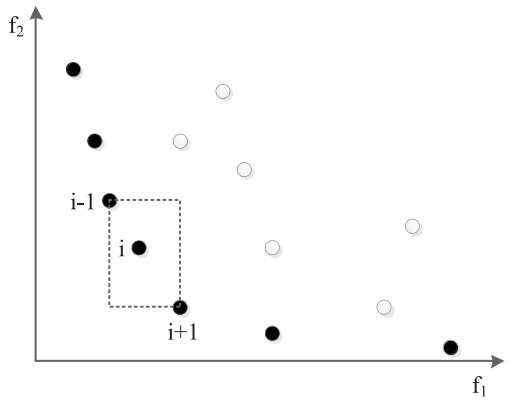


Fig. 5. Diagram of crowding distance.

where the nondominated antibody d belongs to population D, i.e., $d \in D$, and $f_i^{\max}$ and $f_i^{\min}$ represent the maximum and minimum value of the ith objective function, respectively. According to the calculation method defined in (10) and (11), the crowding distance of each nondominated individual d in population D can be obtained. If $I(d, D) > I(d', D), d, d' \in D$, the crowding distance of the individual d' is lower than that of d.

D. Proposed MOIA

Based upon the aforementioned operators, we present the details about the proposed MOIA algorithm. Let $G_{\max}$, n_t, n_d, and n_c denote the maximum number of iterations, the total number of tasks, the population size and the size of cloned population, respectively. The overall procedure of MOIA is described in Algorithm 2. To ensure that all predecessor tasks can be completed ahead of successor tasks, we sort all tasks according to the ascending order of their earliest start times (ESTs) and generate the initial task sequence (line 1). In our scheduling problem, some tasks that can be executed in parallel would have varying task durations. Accordingly, we estimate each task' EST according to its duration in the serial method. The "while" loop is the main part of the proposed algorithm, whereas the "for" loop indicates the evolutionary procedure of the immune algorithm. Line 5 randomly generates a population, and line 6 evaluates the initial population. Lines 10–13 construct the nondominated set D, and lines 16 and 17 generate the cloned population C. Lines 18 and 19 perform the crossover and mutation operators on each individual in the cloned population C with prespecified probability values, respectively, and line 20 updates the antibody population. Lines 23–28 update the result set Res, which is a set of Pareto-optimal or nondominated solutions, and line 29 uses a swapping operation to generate a new task sequence that will be used in the next iteration. Line 26 returns the final results when the "while" loop terminates.

Algorithm 2: Proposed MOIA Algorithm.

Input: $G_{\max}$, n_t, n_d, n_c;
Output: Pareto optimal solution set Res;

```
1  generate an initial task sequence by the ESTF rule;
2  set Res ← Φ and k ← 0;
3  while k < n_t do
4      set D ← Φ, C ← Φ, and t ← 0;
5      generate an antibody population B of size n_d at
       random;
6      calculate the fitness of each antibody in B;
7      for (t = 0 to G_max) do
8          set DT ← Φ;
9          select all non-dominated antibodies in B
           according to their fitness values and add them
           into DT;
10         if (|B| > n_d) then
11             calculate the crowding distances of all
               antibodies in DT;
12             arrange the individuals in the population in a
               descending order of crowding distance;
13             take the first n_d individuals to form a set D;
14         else
15             set D ← DT;
16         arrange the antibody individuals in D in a
           descending order of crowding distance;
17         take the first n_c individuals to construct a set C;
18         perform the single crossover operator on each
           individual in C with the crossover probability;
19         perform the binary mutation operator on each
           individual in C with the mutation probability;
20         update the population by B = C ∪ D;
21         if (Res is empty) then
22             set Res ← D ∪ Res;
23         else
24             add antibodies that are not dominated by Res
               into D;
25         swap the positions of the k-th and the (k + 1)-th
           tasks in the task sequence;
26 return Res;
```

VI. Performance Evaluation

This section introduces the experimental environment for performance evaluation, the information about the hyperspectral datasets, and parameter settings for the scheduling algorithms to be evaluated, followed by thorough evaluation results to justify our proposed approach in terms of computational efficiency and energy efficiency.

A. Experimental Setup

The proposed multiobjective scheduling algorithm was implemented and evaluated on a cloud system built on a Spark cluster consisting of one driver node and up to 32 executor nodes. The driver node is built on a server with an eight-core CPU and 12 GB memory, and each executor node is equipped with two cores operating at 1.87 GHz and 15 GB memory. All computing nodes have installed Ubuntu 16.04 as the operating system. For comparison purposes, we also implement the genetic algorithm

TABLE I
CLASSIFICATION ACCURACIES BY USING THE SERIAL METHOD AND CLOUD IMPLEMENTATION

	Serial Version	Cloud Version			
		4 nodes	8 nodes	16 nodes	32 nodes
Accuracy	95.56%	95.16%	95.24%	95.03%	95.08%

and PSO to solve the multiobjective scheduling problem, and name these two algorithms as MOGA and MOPSO. In addition, for energy estimation, the static power and dynamic power of each computing node is set as 100 and 1000 W, respectively.

We use the publicly available "University of Pavia" dataset to evaluate the total execution time and energy consumption by using the fusion-based classification method. The hyperspectral image in this dataset contains 103 spectral bands, and each band consists of 610 × 340 pixels. The spatial resolution is 1.3 m, and the spectral coverage ranges from 0.43 to 0.86 μm. The data size of the original hyperspectral image is 226 MB. There are totally nine different land covers contained in this image scene. We randomly select 60 samples from each of the nine prelabeled categories to generate the training set for SVM classification. In addition, to examine the scheduling performance for large-scale hyperspectral data, we generate two additional hyperspectral datasets of larger sizes by mosaicking the original "University of Pavia" image. The data sizes of the moisacked images in the two newly generated datasets are 452 MB (610 ×680× 103) and 904 MB (610 ×1360× 103), respectively.

In order to verify the effectiveness of scheduling algorithms, some parameters are set identically, including the number of computing nodes, the population size, and the maximum number of iterations. The specific parameter settings are as follows: the number of virtual machines is 8, population size is 200, and the maximum number of iterations is 200. In addition, the parameters relevant to our MOIA algorithm are set as follows: crossover probability is 0.95, mutation probability is 0.2. For MOGA algorithm, the parameter settings of crossover probability and mutation probability are the same as those of MOIA. For MOPSO algorithm, the individual cognition factor and social communication factor are set as 1.5 and 1.5, respectively. The constriction factor and inertia weight are 2 and 1, respectively. It is worth emphasizing that, the algorithms used for evaluation and comparison could be sensitive to their inherent parameters, in terms of convergence rate and solution exploration capability. In experiments, we start with parameter values that are suggested in the literature, and fine-tune the parameters to their most appropriate values that lead to best scheduling performance. Moreover, since these algorithms are metaheuristic algorithms that involve randomness, we perform each algorithm five times independently and use the average result for performance evaluation.

B. Evaluation Results

The first step is to examine the classification performance of our cloud implementation of the fusion-based method. Table I presents the classification accuracies obtained by using the serial version and the cloud version with different numbers of computing nodes. The comparison results indicate that the difference in classification accuracy between the serial version and cloud version is negligible. In addition, the classification accuracy is stable as the number of computing nodes changes. Compared with the serial version, the accuracy loss by executing the fusion-based classification flow in parallel on Spark is less than 0.5%.

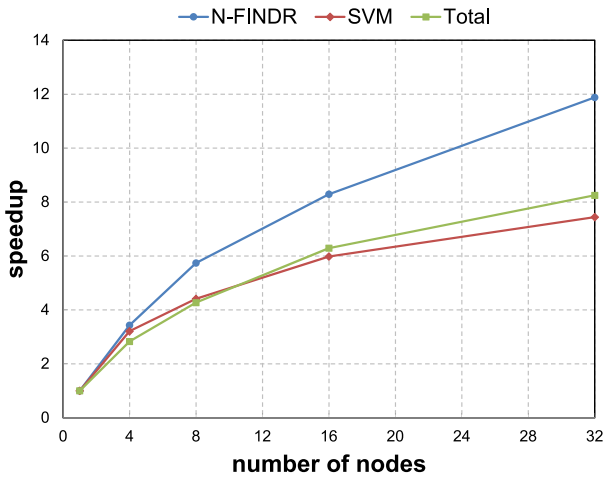


Fig. 6. Parallel processing flow of N-FINDR unmixing model on Spark.

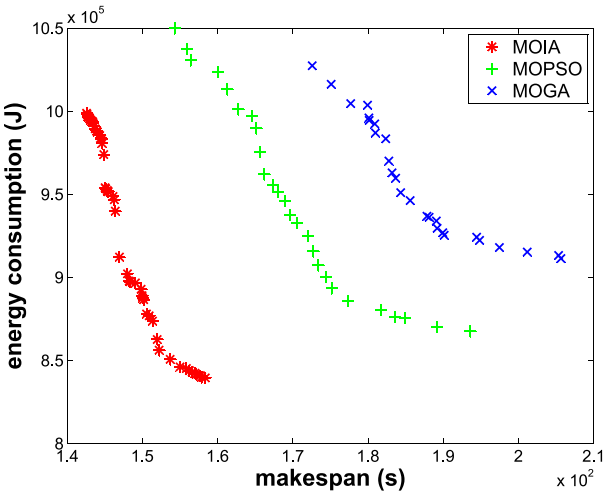


Fig. 7. Comparison of energy-makespan tradeoff solutions obtained by different algorithms with 8 computing nodes.

However, the cloud implementation can significantly improve the computational efficiency of image classification by means of distributed computing. Fig. 6 presents the speedups of N-FINDR unmixing, SVM prediction, and the total classification flow, respectively, over the serial version by deploying different numbers of computing nodes. We can observe that as the number of computing nodes grows, the speedup over the serial version increases but becomes less significant, since more computing nodes would lead to the quickly increasing communication overhead. The speedup of the overall flow reaches 8.25× with 32 computing nodes deployed on the Spark cluster.

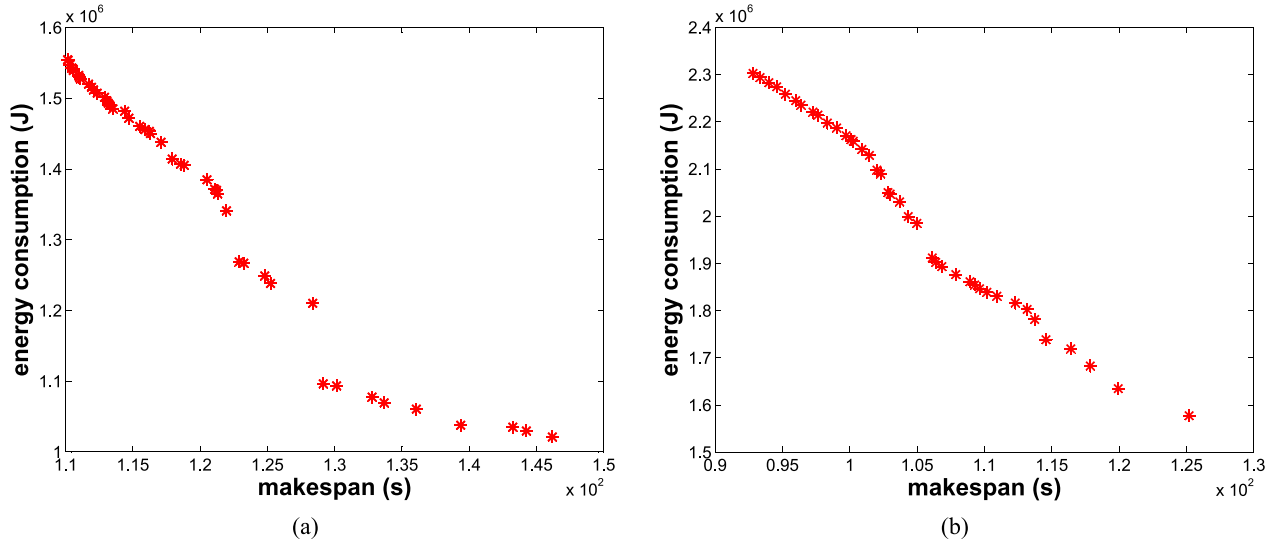


Fig. 8. Pareto frontiers with different numbers of computing nodes. (a) 16 computing nodes (b) 32 computing nodes.

Having verified the classification accuracy and computational efficiency of the cloud implementation, we further evaluate the performance of our multiobjective task scheduling strategy. Fig. 7 shows the Pareto frontiers obtained by the three algorithms when the number of computing nodes is 8. The red points represent the Pareto-optimal solution obtained by MOIA, and the green points and blue points represent the Pareto-optimal solutions obtained by MOPSO and MOGA, respectively. The x-axis and y-axis represent the total execution time of hyperspectral image classification (in seconds) and the energy consumption by the computing nodes (in joules), respectively. We can observe that, compared with the two competing algorithms, our proposed MOIA can produce a better Pareto frontier in terms of computational efficiency and energy efficiency. In other words, MOIA obtains a set of energy-makespan tradeoff solutions that lead to shorter makespans or lower energy consumptions.

In order to further verify the performance of MOIA with different numbers of computing nodes, we perform additional experiments and present in Fig. 8 the Pareto frontiers with 16 and 32 computing nodes, respectively. The results justify the high quality of the Pareto-optimal solutions (i.e., the energy-makespan tradeoff solutions) in MOIA-generated Pareto frontiers. By investigating the frontiers in Figs. 7 and 8, we observe that there exist certain disconnected regions on the frontiers. This observation is because that task reallocations defined in the antibody individuals may possibly cause unevenly distributed solutions in the energy-makespan space. In addition, as the number of computing nodes increases, since the size of solution space grows exponentially, the effect of task reallocation upon the distribution of energy-makespan tradeoff solutions becomes less obvious. One of our future work directions is to obtain well-distributed Pareto frontiers by 1) increasing the antibody population size to enhance MOIA's solution exploration capability, and 2) incorporating advanced refinement strategies [42], [43] to eliminate the disconnected regions on the obtained Pareto frontier.

We also compare the evaluation results by using the proposed MOIA with those by using the serial and parallel versions of the classification method, with different numbers of computing nodes deployed on Spark. For this reason, we calculate the average values of makespan and energy consumption for all Pareto-optimal solutions, considering that the result obtained by the multiobjective scheduling algorithm is a set of Pareto-optimal solutions. The comparison results with 8, 16, and 32 computing nodes are listed in Tables II– IV, respectively. Table II show that, with 8 computing nodes, our proposed MOIA uses 148.27 s for processing the fusion-based classification method and consumes 9.25×10^5J energy. In contrast, the serial algorithm consumes 658.91s and 6.59×10^5J, whereas the parallel implementation on Spark consumes 175.35 s and 14.5×10^5J. In other words, the parallel implementation without scheduling and the propose MOIA outperform the serial algorithm by 73.39% and 77.50% in terms of the computational efficiency, respectively. On the side of energy consumption, compared with the serial algorithm, both the parallel implementation and MOIA consume more energy due to the distributed computing on multiple nodes. Specially, the proposed IA is less energy-consuming than the Spark parallel algorithm by 36.21%. Similar observations can be obtained from Tables III and IV. We can conclude that compared with the serial algorithm, the makespans by both the parallel method and MOIA are significantly reduced, and MOIA outperforms the parallel method in terms of computational efficiency. The reason is that the parallel methods supports the parallel computing and the computational load are distributed over multiple computing nodes. Moreover, the MOIA scheduling algorithm determines the Pareto-optimal scheduling solutions to further improve the computational efficiency. On the other hand, the proposed MOIA is also advantageous over the parallel implementation with lower energy consumption for processing the classification flow. In summary, our proposed MOIA outperforms the parallel method in terms of both the computational efficiency and the amount of energy consumption.

TABLE II
SCHEDULING RESULTS BY DIFFERENT METHODS WITH 8 COMPUTING NODES

	Serial		Parallel		MOIA	
Scheduling Objective	makespan (s) 658.91	energy (J) 6.59×10^5	makespan (s) 175.35	energy (J) 14.5×10^5	makespan (s) 148.27	energy (J) 9.3×10^5
v.s. 'Serial'	——	——	73.39%	-120.0%	77.50%	-40.36%
v.s. 'Parallel'	——	——	——	——	15.44%	36.21%

TABLE III
SCHEDULING RESULTS BY DIFFERENT METHODS WITH 16 COMPUTING NODES

	Serial		Parallel		MOIA	
Scheduling Objective	makespan (s) 658.91	energy (J) 6.59×10^5	makespan (s) 128.45	energy (J) 21.2×10^5	makespan (s) 118.10	energy (J) 13.7×10^5
v.s. 'Serial'	——	——	80.50%	-221.6%	82.08%	-107.8%
v.s. 'Parallel'	——	——	——	——	8.06%	35.38%

TABLE IV
SCHEDULING RESULTS BY DIFFERENT METHODS WITH 32 COMPUTING NODES

	Serial		Parallel		MOIA	
Scheduling Objective	makespan (s) 658.91	energy (J) 6.59×10^5	makespan (s) 104.31	energy (J) 34.5×10^5	makespan (s) 101.74	energy (J) 20.1×10^5
v.s. 'Serial'	——	——	84.17%	-423.5%	84.56%	-205.0%
v.s. 'Parallel'	——	——	——	——	2.46%	41.73%

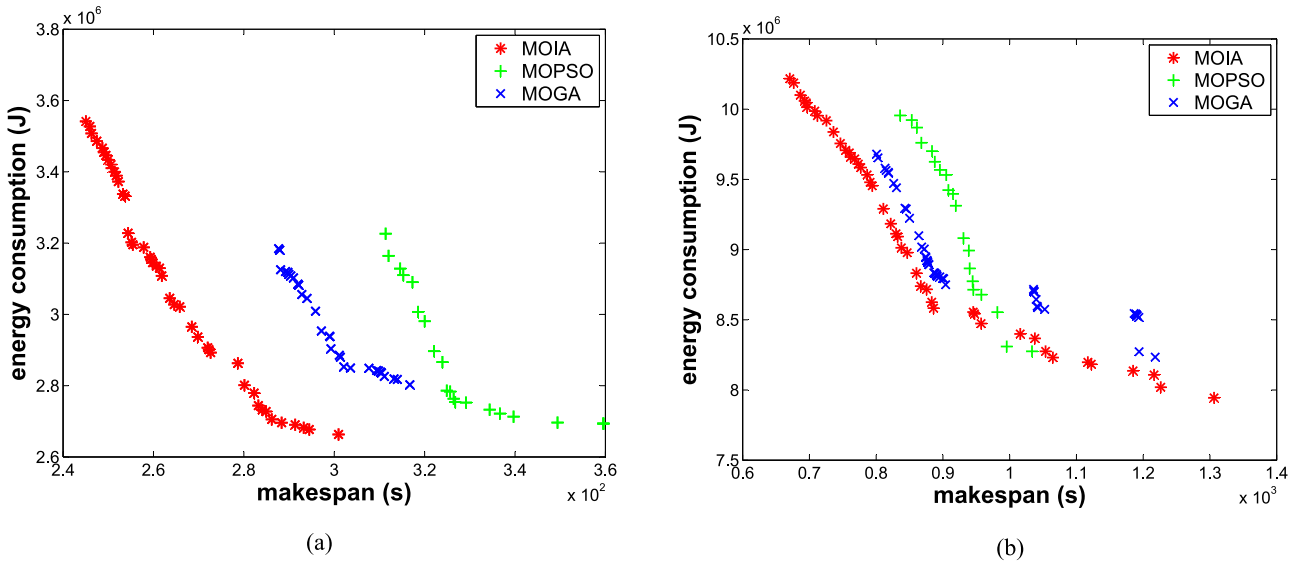


Fig. 9. Comparison of energy-makespan tradeoff solutions by different scheduling algorithms for larger-scale hyperspectral datasets.

Finally, in order to validate the scalability of our MOIA algorithm, we perform an additional set of experiments on the two datasets generated by moisacking the original hyperspectral image, which are of sizes 452 and 904 MB, respectively. Fig. 9 shows the Pareto frontiers obtained by the three scheduling algorithms used for comparison. The red, green, and blue points denote the Pareto-optimal soluitons generated by MOIA, MOPSO, and MOGA, respectively. We can observe that for larger sizes of hyperspectral data, MOIA is still capable of producing better energy-makespan tradeoff solution as compared to the competing algorithms. To confirm this observation in a quantitative manner, Table V further presents the average energy consumptions and makespans of tradeoff solutions obtained by different scheduling algorithms. The

TABLE V
ENERGY CONSUMPTIONS AND MAKESPANS BY DIFFERENT SCHEDULING ALGORITHMS FOR LARGER-SCALE HYPERSPECTRAL DATASETS

Data Size	Energy Consumption (J)			Makespan (s)		
	MOIA	MOPSO	MOGA	MOIA	MOPSO	MOGA
452 MB	2.82×10^6	2.86×10^6	2.96×10^6	279.86	328.26	299.80
904 MB	8.95×10^6	9.20×10^6	8.98×10^6	848.30	925.11	957.42

energy consumptions by using MOIA are slightly lower than those by using MOPSO and MOGA, but MOIA improves the makespan of the classification application considerably. The makespan reduction is up to 14.74% for the 452 MB dataset, and is up to 11.40% for the 904 MB dataset. The comparison results indicate that the proposed approach is effective as the hyperspectral data volume grows.

VII. CONCLUSION

This article proposed an energy-efficient cloud implementation of hyperspectral image classification by developing a multiobjective task scheduling algorithm. We first presented a cloud implementation of a fusion-based classification method based on Apache Spark. With the general classification flow represented by a DAG model, we formulated a multiobjective scheduling framework that jointly minimizes the total execution time as well as energy consumption. We further developed an effective MOIA scheduling algorithm to solve the multiobjective scheduling problem. Experimental results demonstrated that MOIA can generate high-quality tradeoff solutions that reduce the execution time and energy consumption for hyperspectral image classification.

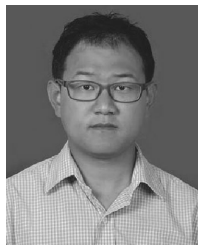
To extend the applicability of the multiobjective scheduling approach to other applications and to cloud environments, our future work can be directed toward the following aspects. On the one hand, our scheduling strategy can be easily adapted for other representative hyperspectral image applications. As long as we can characterize the application workflow by a DAG structure and identify those tasks that can be decomposed and processed in parallel, we can use the proposed multiobjective task algorithm to co-optimize the execution time and energy consumption on a cloud system. Specifically, if the following two conditions are available: 1) the application flow can be decomposed into a set of substasks with dependencies among them, and 2) the computation load of certain tasks can be partitioned and allocated onto multiple computing nodes, our approach can be applied to achieve the co-optimization of computational efficiency and energy efficiency. In addition, since the scheduling approach is built upon a workflow model that characterizes the dependencies and parallelism among application subtasks, we plan to study the automatic profiling of application workflows to further enhance the generality of the proposed scheduling approach.

On the other hand, in this article, we sought the optimal scheduling solution for efficient processing of large-scale hyperspectral data in a private cloud environment. Therefore, the scheduling framework takes into account only execution time and energy consumption, which are the two main concerns of private cloud providers. However, if the cloud users migrate to public clouds or hybrid clouds for executing hyperspectral applications, cloud usage cost would become the primary concern. To cope with various cloud environments, the scheduling framework has to be adapted by incorporating more optimization concerns, e.g., cost minimization, load balancing, and resource utilization. As a result, several operators in the multiobjective scheduling algorithm, including fitness calculation and crowding distance calculation, also need to be redesigned to comply with the new optimization model.

REFERENCES

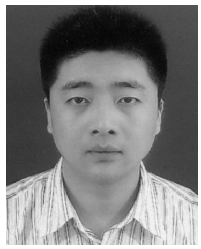
[1] C. Yu *et al.*, "Hyperspectral image classification method based on CNN architecture embedding with hashing semantic feature," *IEEE J. Select. Topics Appl. Earth Observ. Remote Sens.*, vol. 12, no. 6, pp. 1866–1881, Jun. 2019.
[2] N. He *et al.*, "Feature extraction with multiscale covariance maps for hyperspectral image classification," *IEEE Trans. Geosci. Remote Sens.*, vol. 57, no. 2, pp. 755–769, Feb. 2019.
[3] L. Wang, T. Zhang, Y. Fu, and H. Huang, "HyperReconNet: Joint coded aperture optimization and image reconstruction for compressive hyperspectral imaging," *IEEE Trans. Image Process.*, vol. 28, no. 5, pp. 2257–2270, May 2019.
[4] M. Yan *et al.*"Remote sensing big data computing: Challenges and opportunities," *Future Gen. Comput. Syst.*, vol. 51, pp. 47–60, 2015.
[5] D. Lunga, J. Gerrand, L. Yang, C. Layton, and R. Stewart, "Apache Spark accelerated deep learning inference for large scale satellite image analytics," *IEEE J. Select. Topics Appl. Earth Observ. Remote Sens.*, vol. 13, pp. 271–283, 2020.
[6] Z. Wu *et al.*, "Scheduling-guided automatic processing of massive hyperspectral image classification on cloud computing architectures," *IEEE Trans. Cybern.*, to be published, doi: 10.1109/TCYB.2020.3026673.
[7] H. Yu, L. Gao, W. Liao, B. Zhang, L. Zhuang, M. Song, and J. Chanussot, "Global spatial and local spectral similarity-based manifold learning group sparse representation for hyperspectral imagery classification," *IEEE Trans. Geosci. Remote Sens.*, vol. 58, no. 5, pp. 3043–3056, May 2020.
[8] F. Luo, D. Bo, L. Zhang, L. Zhang, and D. Tao, "Feature learning using spatial-spectral hypergraph discriminant analysis for hyperspectral image," *IEEE Trans. Cybern.*, vol. 49, no. 7, pp. 2406–2419, Jul. 2019.
[9] J. Feng *et al.*, "CNN-based multilayer spatial–spectral feature fusion and sample augmentation with local and nonlocal constraints for hyperspectral image classification," *IEEE J. Select. Topics Appl. Earth Observ. Remote Sens.*, vol. 12, no. 4, pp. 1299–1313, Apr. 2019.
[10] H. Huang, Y. Duan, H. He, and G. Shi, "Local linear spatial–spectral probabilistic distribution for hyperspectral image classification," *IEEE Trans. Geosci. Remote Sens.*, vol. 58, no. 2, pp. 1259–1272, Feb. 2020.
[11] H. Tang, Y. Li, X. Han, Q. Huang, and W. Xie, "A spatial–spectral prototypical network for hyperspectral remote sensing image," *IEEE Geosci. Remote Sens. Lett.*, vol. 17, no. 1, pp. 167–171, Jan. 2020.
[12] T. Lu, S. Li, L. Fang, X. Jia, and J. A. Benediktsson, "From subpixel to superpixel: A novel fusion framework for hyperspectral image classification," *IEEE Trans. Geosci. Remote Sens.*, vol. 55, no. 8, pp. 4398–4411, Aug. 2017.
[13] J. Sun *et al.*, "An efficient and scalable framework for processing remotely sensed big data in cloud computing environments," *IEEE Trans. Geosci. Remote Sens.*, vol. 47, no. 7, pp. 4294–4308, Jul. 2019.
[14] H. El-Rewini, T. G. Lewis, and H. H. Ali, *Task Scheduling in Parallel and Distributed Systems*. Englewood Cliffs, NJ, USA: Prentice-Hall, 2007.

[15] Q. Wu, F. Ishikawa, Q. Zhu, and Y. Xia, "Energy and migration cost-aware dynamic virtual machine consolidation in heterogeneous cloud datacenters," *IEEE Trans. Services Comput.*, vol. 12, no. 4, pp. 550–563, Jul./Aug. 2019.
[16] Y. Lee and A. Zomaya, "Energy efficient utilization of resources in cloud computing systems," *J. Supercomput.*, vol. 60, no. 2, pp. 268–280, 2012.
[17] S. Prathiba and S. Sankar, "Architecture to minimize energy consumption in cloud datacenter," in *Proc. Int. Conf. Intell. Comput. Control Syst.*, 2019, pp. 1044–1048.
[18] X. Li, P. Garraghan, X. Jiang, Z. Wu, and J. Xu, "Holistic virtual machine scheduling in cloud datacenters towards minimizing total energy," *IEEE Trans. Parallel Distribut. Syst.*, vol. 29, no. 6, pp. 1317–1331, Jun. 2018.
[19] D. Miner and A. Shook, *MapReduce Design Patterns*. Sebastopol, CA, USA: O'Reilly Media, 2012.
[20] S. W. Dowler, R. Takashima, and M. Andrews, "Reducing the complexity of the N-FINDR algorithm for hyperspectral image analysis," *IEEE Trans. Image Process.*, vol. 22, no. 7, pp. 2835–2848, Jul. 2013.
[21] F. Melgani and L. Bruzzone, "Classification of hyperspectral remote sensing images with support vector machines," *IEEE Trans. Geosci. Remote Sens.*, vol. 42, no. 8, pp. 1778–1790, Aug. 2004.
[22] Q. Lin *et al.*"A hybrid evolutionary immune algorithm for multiobjective optimization problems," *IEEE Trans. Evol. Comput.*, vol. 20, no. 5, pp. 711–729, Oct. 2016.
[23] M. Gong, L. Jiao, H. Du, and L. Bo, "Multiobjective immune algorithm with nondominated neighbor-based selection," *Evol. Comput.*, vol. 16, no. 2, pp. 225–255, 2008.
[24] V. A. A. Quirita *et al.*"A new cloud computing architecture for the classification of remote sensing data," *IEEE J. Select. Topics Appl. Earth Observ. Remote Sens.*, vol. 10, no. 2, pp. 409–416, Feb. 2017.
[25] X. Pan and S. Zhang, "A remote sensing image cloud processing system based on Hadoop," in *Proc. Int. Conf. Cloud Comput. Intell. Syst.*, 2012, pp. 492–494.
[26] J. M. Haut, M. Paoletti, J. Plaza, and A. Plaza, "Cloud implementation of the K-means algorithm for hyperspectral image analysis," *J. Supercomput.*, vol. 73, pp. 514–529, 2017.
[27] H. Leontyev and J. H. Anderson, "Tardiness bounds for FIFO scheduling on multiprocessors," in *Proc. Euromicro Conf. Real-Time Syst.*, 2007, pp. 71:1–71:10.
[28] M. A. Alworafi, A. Dhari, A. A. Al-Hashmi, A. B. Darem, and Suresha,"An improved SJF scheduling algorithm in cloud computing environment," in *Proc. Int. Conf. Elect., Electron., Commun., Comput. Optim. Techn.*, 2016, pp. 208–212.
[29] D. Gabi, A. S. Ismail, A. Zainal, Z. Zakaria, and A. Al-Khasawneh, "Cloud scalable multi-objective task scheduling algorithm for cloud computing using cat swarm optimization and simulated annealing," in *Proc. Int. Conf. Inf. Technol.*, 2017, pp. 1007–1012.
[30] Z. Zheng, K. Xie, S. He, and J. Deng, "A multi-objective optimization scheduling method based on the improved differential evolution algorithm in cloud computing," in *Proc. Int. Conf. Cloud Comput. Sec.*, 2017, pp. 226–238.
[31] E. S. Alkayal, N. R. Jennings, and M. F. Abulkhair, "Efficient task scheduling multi-objective particle swarm optimization in cloud computing," in *Proc. Conf. Local Comput. Netw. Workshops*, 2016, pp. 17–24.
[32] D. Qian, H. Ren, and C. I. Chang, "A comparative study for orthogonal subspace projection and constrained energy minimization," *IEEE Trans. Geosci. Remote Sens.*, vol. 41, no. 6, pp. 1525–1529, Jun. 2003.
[33] Y. Chen, Z. Wu, Z. Wei, and Y. Li, "PN-FINDR: A parallelized N-FINDR algorithm with spark," in *Proc. Int. Conf. Advan. Cloud Big Data*, 2016, pp. 127–132.
[34] J. M. P. Nascimento and J. M. B. Dias, "Vertex component analysis: A fast algorithm to unmix hyperspectral data," *IEEE Trans. Geosci. Remote Sens.*, vol. 43, no. 4, pp. 898–910, Apr. 2005.
[35] S. Ryza, U. Laserson, S. Owen, and J. Wills, *Advanced Analytics With Spark: Patterns for Learning from Data at Scale*. Sebastopol, CA, USA: O'Reilly Media, 2015.
[36] M. E. Winter, "N-FINDR: An algorithm for fast autonomous spectral end-member determination in hyperspectral data," in *Proc. Int. Soc. Opt. Photon.*, 1999, pp. 266–275.
[37] Y. Gao, H. Guan, Z. Qi, H. Yang, and L. Liang, "A multi-objective ant colony system algorithm for virtual machine placement in cloud computing," *J. Comput. Syst. Sci.*, vol. 79, no. 8, pp. 1230–1242, 2013.
[38] X. Fan, W.-D. Weber, and L. A. Barroso, "Power provisioning for a warehouse-sized computer," in *Proc. Int. Symp. Comput. Architecture*, 2007, pp. 13–23.
[39] I. Paprocka and B. Skołud, "A hybrid multi-objective immune algorithm for predictive and reactive scheduling," *J. Scheduling*, vol. 20, no. 2, pp. 165–182, 2017.
[40] H. Mokhtari and A. Hasani, "An energy-efficient multi-objective optimization for flexible job-shop scheduling problem," *Comput. Chem. Eng.*, vol. 104, pp. 339–352, 2017.
[41] N. Srinivas and K. Deb, "Muiltiobjective optimization using nondominated sorting in genetic algorithms," *Evol. Computat.*, vol. 2, no. 3, pp. 221–248, 1994.
[42] S. Jiang and S. Yang, "An improved multiobjective optimization evolutionary algorithm based on decomposition for complex Pareto fronts," *IEEE Trans. Cybern.*, vol. 46, no. 2, pp. 421–437, Feb. 2016.
[43] Q. Zhang, W. Zhu, B. Liao, X. Chen, and L. Cai, "A modified PBI approach for multi-objective optimization with complex Pareto fronts," *Swarm Evol. Comput.*, vol. 40, pp. 216–237, 2018.



Jin Sun (Member, IEEE) received the B.S. degree in computer science and technology from the Nanjing University of Science and Technology, Nanjing, China, in 2004, and the Ph.D. degree in electrical and computer engineering from the University of Arizona, Tucson, AZ, USA, in 2011.

From January 2012 to September 2014, he was with Orora Design Technologies, Inc. as a member of Technical Staff. He is currently an Associate Professor with the School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing, China. His research interests include high-performance computing and electronic design automation.



Heng Li received the B.S. degree in information management and information system and the M.S. degree in information science from the Nanjing University of Science and Technology Nanjing, China, in 2014 and 2016, respectively. He is currently working toward the Ph.D. degree in computer application technology with the School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing, China.

His research interests include high-performance computing and image processing.

Yi Zhang received the B.S. degree in computer science and technology and Ph.D. degree in computer application technology from the School of Computer Science and Engineering, Southeast University, Nanjing, China, in 2005 and 2011, respectively.

From July 2009 to December 2009, he was an intern with IBM China Research Laboratory, Beijing, China. In 2011, he joined Huawei Tech. Co., Nanjing, as a Member of Technical Research Staff. He is currently an Associate Professor with the School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing, China. His research interests include project scheduling, workflow optimization, resource management, and allocation in cloud computing and mobile computing.

Dr. Zhang was a recipient of the IBM Ph.D. Fellowship.

Yang Xu (Member, IEEE) received the B.S. degree in applied mathematics and the Ph.D. degree in pattern recognition and intelligence systems from the Nanjing University of Science and Technology Nanjing, China, in 2011 and 2016, respectively.

He is currently an Associate Professor with the School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing, China. His research interests include hyperspectral image classification, hyperspectral detection, image processing, and machine learning.

Yaoqin Zhu received the B.S. degree in computer science and technology and the Ph.D. degree in computer applied technology from the Nanjing University of Science and Technology, Nanjing, China, in 2000 and 2005, respectively.

She is currently an Assistant Professor with the School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing, China. Her research interests include multimedia processing, virtual reality, and computer simulation.

Qitao Zang received the B.S. degree in computer science and technology and the M.S. degree in computer technology from the Nanjing University of Science and Technology, Nanjing, China, in 2016 and 2019, respectively.

He is currently with Hangzhou Hikvision Digital Technology Company, Ltd., Hangzhou, China as a Software Development Engineer. His research interests include hyperspectral image processing, cloud computing, and task scheduling.

Zebin Wu (Senior Member, IEEE) received the B.Sc. and Ph.D. degrees in computer science and technology from the Nanjing University of Science and Technology, Nanjing, China, in 2003 and 2007, respectively.

From 2014 to 2015, he was a Visiting Scholar with the Hyperspectral Computing Laboratory, Department of Technology of Computers and Communications, Escuela Politécnica, University of Extremadura, Cáceres, Spain. He was a Visiting Scholar with the Department of Mathematics, University of California, Los Angeles, CA, USA, from 2016 to 2017. He was also a Visiting Scholar with the GIPSA-lab, Grenoble INP, the Université Grenoble Alpes, Grenoble, France, in 2018. He is currently a Professor with the School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing, China. His research interests include hyperspectral image processing, high-performance computing, and computer simulation.

Zhihui Wei received the B.Sc. and M.Sc. degrees in applied mathematics, and the Ph.D. degree in communication and information system from Southeast University, Nanjing, China, in 1983, 1986, and 2003, respectively.

He is currently a Professor and Doctoral Supervisor with the Nanjing University of Science and Technology, Nanjing, China. His research interests include partial differential equations, image processing, multiscale analysis, sparse representation, and compressed sensing.