Scheduling-Guided Automatic Processing of Massive Hyperspectral Image Classification on Cloud Computing Architectures

Zebin Wu, *Senior Member, IEEE*, Jin Sun, *Member, IEEE*, Yi Zhang, Yaoqin Zhu, Jun Li, *Senior Member, IEEE*, Antonio Plaza, *Fellow, IEEE*, Jón Atli Benediktsson, *Fellow, IEEE*, and Zhihui Wei, *Member, IEEE*

***Abstract*—The large data volume and high algorithm complexity of hyperspectral image (HSI) problems have posed big challenges for efficient classification of massive HSI data repositories. Recently, cloud computing architectures have become more relevant to address the big computational challenges introduced in the HSI field. This article proposes an acceleration method for HSI classification that relies on scheduling metaheuristics to automatically and optimally distribute the workload of HSI applications across multiple computing resources on a cloud platform. By analyzing the procedure of a representative classification method, we first develop its distributed and parallel implementation based on the MapReduce mechanism on Apache Spark. The subtasks of the processing flow that can be processed in a distributed way are identified as divisible tasks. The optimal execution of this application on Spark is further formulated as a divisible scheduling framework that takes into account both task execution precedences and task divisibility when allocating the divisible and indivisible subtasks onto computing nodes. The formulated scheduling framework is an optimization procedure that searches for optimized task assignments and partition counts for divisible tasks. Two metaheuristic algorithms are developed to solve this divisible scheduling problem. The scheduling results provide an optimized solution to the automatic processing of HSI big data on clouds, improving the computational efficiency of HSI classification by exploring the parallelism during the parallel processing flow. Experimental results demonstrate that our scheduling-guided approach achieves remarkable speedups by facilitating the automatic processing of HSI classification on Spark, and is scalable to the increasing HSI data volume.**

***Index Terms*—Cloud computing, distributed and parallel processing, divisible task scheduling, hyperspectral image (HSI) classification, partitioning factor.**

Manuscript received April 2, 2020; revised July 21, 2020; accepted September 18, 2020. This work was supported in part by the National Natural Science Foundation of China under Grant 61772274 and Grant 61872185; 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 30917015104, Grant 30919011103, Grant 30919011402, and Grant 30920021132; in part by the Junta de Extremadura (Decreto 14/2018, de 6 de febrero, por el que se establecen las bases reguladoras de las ayudas para la realizacion de actividades de investigacion y desarrollo tecnologico, de divulgacion y de transferencia de conocimiento por los Grupos de Investigacion de Extremadura) under Grant GR18060; and in part by the European Union's Horizon 2020 Research and Innovation Programme (EOXPOSURE) under Grant 734541. This article was recommended by Associate Editor P. P. Angelov. *(Corresponding author: Jin Sun.)*

Zebin Wu, Jin Sun, Yi Zhang, Yaoqin Zhu, and Zhihui Wei are with the School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing 210094, China (e-mail: wuzb@njust.edu.cn; sunj@njust.edu.cn; yzhang@njust.edu.cn; zhuyaoqin@njust.edu.cn; gswei@njust.edu.cn).

Jun Li is with the Guangdong Provincial Key Laboratory of Urbanization and Geo-Simulation, Center of Integrated Geographic Information Analysis, School of Geography and Planning, Sun Yat-sen University, Guangzhou 510275, China (e-mail: lijun48@mail.sysu.edu.cn).

Antonio Plaza is with the Hyperspectral Computing Laboratory, Department of Technology of Computers and Communications, University of Extremadura, E-10071 Cáceres, Spain (e-mail: aplaza@unex.es).

Jón Atli Benediktsson is with the Faculty of Electrical and Computer Engineering, University of Iceland, 101 Reykjavik, Iceland (e-mail: benedikt@hi.is).

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCYB.2020.3026673

I. Introduction

DURING recent years, hyperspectral remote sensing has been widely applied in various fields of earth observation and space exploration [1]–[3]. Hyperspectral sensors are now able to simultaneously measure hundreds of contiguous spectral bands with high spectral resolution. In hyperspectral images (HSIs), each pixel of the collected data cube can be represented by a vector of which the entries correspond to the spectral bands, providing detailed spectral information of the underlying materials within the pixel [4]. Due to the substantial amount of spectral and spatial information offered by HSI data, HSI has been popularly used in a variety of remote sensing applications [5], [6].

Due to the large number of bands contained in HSIs, the processing of high-dimensional HSI data generally requires huge storage space and heavy computational load, and therefore falls within the category of big data problems [7]. Recently, cloud computing has become a promising solution to the efficient processing of HSI big data due to its superior capabilities in high-performance computing. As the demand for HSI big data processing continues to increase, there have been more research studies seeking distributed processing of large-scale HSI datasets on cloud computing architectures [8]–[10]. The fundamental idea of these cloud-based approaches is to use the distributed file system to cope with the storage and management of HSI big data, and utilize MapReduce [11], a distributed computing model, to support

the parallel processing of data- and/or computation-intensive tasks in HSI applications.

As one of the most popular and important techniques for HSI interpretation, HSI classification aims at labeling HSI pixels into a set of predefined classes, normally using some previously available information about the classes [12], [13]. Supervised and semisupervised classifications with kernel methods, for example, support vector machines (SVMs) [14], [15], graph-based approaches [16], sparse representation-based methods [17], [18], probabilistic methods [19], and fusion-based methods [20], are all important trends and have been extensively used for HSI classification. Recently, spatial-spectral classification approaches [21]–[23] have also shown noticeable success by exploiting spatial-contextual information. Lu *et al.* [20] introduced a general feature fusion-based strategy for HSI classification that integrates several different strategies for pixel interpretation, achieving high classification accuracies. This method, called subpixel-, pixel-, and superpixel-level feature classification (SPS-FC), uses different kinds of features to achieve highly discriminative information for classification purposes, adopting a general fusion framework. By fusing multiple feature-induced kernels to form one composite kernel, this method can effectively improve the discrimination capability of the classifier. However, due to its complicated classification flow, SPS-FC becomes much less efficient when the HSI data size increases. Also, SPS-FC requires manual intervention during the classification procedure. Therefore, it is necessary to develop an efficient cloud-based solution to implement the automatic processing and acceleration of the SPS-FC method for large-scale HSI datasets.

Similar to other techniques for HSI classification, the SPS-FC method consists of a set of subtasks, some of which can be processed in parallel for distributing the computation load over a group of processing elements (PEs). In addition, there is a considerable number of dependencies among the subtasks of the entire classification flow that impose an order of precedence on their execution. An immediate conclusion is that when performing HSI classification on clouds, a crucial step is the assignment of these subtasks onto PEs and the order of task execution. This step, which is referred to as *scheduling* [24], [25], determines the parallelism of HSI classification flow on clouds. The scheduling problem is generally NP-complete in most cases [26]. To solve this category of problems, metaheuristic algorithms are extensively studied to search for near-optimal solutions in solution space, for example, simulated annealing [27], particle swarm optimization [28], and the quantum-inspired evolutionary algorithm (QEA) [29].

In addition to the intertask parallelism that depends on the assignment of tasks onto cloud computing resources, there exists another kind of parallelism that can be exploited in the parallel implementation of HSI classification, that is, the intratask parallelism originating from the MapReduce-based partitioning mechanism. The computation load of certain tasks can be partitioned and assigned to multiple PEs. Tasks of this kind are accordingly referred to as *divisible tasks*. In contrast, indivisible tasks that cannot be partitioned have to be processed in their entirety on a single PE. The manner in which task partitioning can be done depends on the task's divisibility property, that is, the property that determines whether the computation load can be decomposed into any number of load partitions, as long as it does not exceed the resource limit [30].

Motivated by the above-mentioned intratask parallelism, we define *partitioning factor* for each divisible task, indicating the number of partitions that its computation load is decomposed into. Moreover, it should be noted that the distributed computing mechanism would induce additional communication overhead. Due to the overhead of data communication among computing nodes, the speedup achieved by employing the parallel computing mechanism does not increase linearly when the partitioning factor increases. For this reason, when designing a scheduling strategy, it is of great importance to take into account the partitioning factors for all divisible tasks. An appropriate decision regarding the partitioning factors can be greatly beneficial for reducing the communication cost and improving the overall efficiency of distributed processing. The determination of an optimal mapping of indivisible and divisible tasks falls into the category of *divisible scheduling* problems [30], [31], and requires efficient scheduling algorithms to achieve an optimal solution to parallel implementation of HSI classification.

With the aforementioned ideas in mind, we propose a new scheduling-guided method that supports the automatic processing of complicated HSI classifications on Apache Spark. We first develop the distributed and parallel implementation of the SPS-FC method based on the MapReduce mechanism. The tasks, whose computation load can be partitioned to facilitate parallel processing, are identified as divisible tasks. All the subtasks (divisible and indivisible) of the SPS-FC flow and the precedence relations among them are further represented by a directed acyclic graph (DAG). The optimal execution of SPS-FC flow on Spark is further formulated as a divisible scheduling framework that explores both intertask parallelism and intratask parallelism. The formulated scheduling framework is fundamentally an optimization procedure that searches for the best partitioning factors for divisible tasks, as well as the mapping of all tasks onto PEs. Different from existing approaches that focus on exploiting data-level parallelism by data partitioning [32], [33], our divisible scheduling framework aims at the automatic and efficient processing of complicated HSI applications on clouds by determining the optimal intertask and intratask parallelisms. The optimized solution of partitioning factors and task assignments obtained by our scheduling-guided approach is beneficial for achieving high utilization of PEs and reduced execution time when processing this parallel HSI classification flow on Spark. Specifically, this work makes the following innovative contributions.

1) We develop a new parallel implementation to accelerate the processing of a general and highly illustrative HSI classification method on cloud computing platforms.
2) We propose employing scheduling strategies to facilitate the automatic processing of HSI classifications with complicated flows by exploiting the intertask

and intratask parallelisms involved in the Spark implementation.

3) We formulate the resulting divisible scheduling problem as an optimization framework that incorporates task assignments and partitioning factors as decision variables.
4) We develop effective metaheuristic algorithms to solve the formulated divisible scheduling problem, searching for the most appropriate solution to the automatic processing of HSI classification on clouds.

The effectiveness of the scheduling-guided approach is verified on two HSI datasets by using two representative classification applications. Our experimental results demonstrate that guided by the divisible scheduling solutions, our approach leads to a significant improvement in computational efficiency with different numbers of Spark nodes and for different HSI data sizes. In addition, the proposed approach exhibits linear scalability with regard to the increasing volume of the HSI dataset.

The remainder of this article is organized as follows. Section II describes the parallel implementation of the SPS-FC classification method on Spark. Section III discusses the significance of exploiting intertask and intratask parallelisms in HSI classification, and further formulates the divisible scheduling model. Section IV details the proposed scheduling algorithms for solving the formulated model. Section V presents the experimental results on public HSI datasets. Finally, concluding remarks and discussions are provided in Section VI.

II. Distributed and Parallel Implementation of SPS-FC Classification Method

The SPS-FC method is a sophisticated and effective feature fusion approach for HSI classification. The processing flow of this method consists of many subtasks with strong dependencies among them. In this section, we provide a cloud implementation of this method that partitions the computation- and/or data-intensive tasks into multiple small tasks relying on the MapReduce mechanism. With these divisible tasks, which can be distributed and processed on multiple PEs, finding an optimal allocation of all tasks on limited cloud computing resources naturally becomes a divisible scheduling problem.

A. Serial Processing Flow

The SPS-FC starts with the extraction of subpixel-level, pixel-level, and superpixel-level features from the HSI, respectively. Then, multiple feature-induced kernels are fused to form one composite kernel. The fused composite kernel is further incorporated with an SVM classifier to determine the labels of pixels. The serial processing flow of SPS-FC classification is illustrated in Fig. 1, and mainly consists of 17 subtasks. The detailed procedures are summarized in Algorithm 1, including ten essential steps.

As shown by Algorithm 1, the SPS-FC method involves feature extraction at three different levels, as well as pixel-based classification based on the fused features. The techniques included in the SPS-FC are representative of several HSI data processing operations, including dimensionality reduction [principal component analysis (PCA) and minimum noise fraction (MNF)], clustering (k-means), spectral unmixing constrained energy minimization (CEM), feature extraction (morphology), classification (SVM), and superpixel-based segmentation entropy rate superpixel (ERS). As a result, it represents a general processing chain and a highly representative framework for HSI data interpretation. Several of these procedures require massive data processing or large-scale matrix computations. Such a complicated algorithmic workflow would inevitably limit its computational efficiency on a single machine. By taking advantage of the MapReduce scheme on Spark, the computation load of those computation- and data-intensive procedures can be divided and distributed across multiple PEs to accelerate their execution.

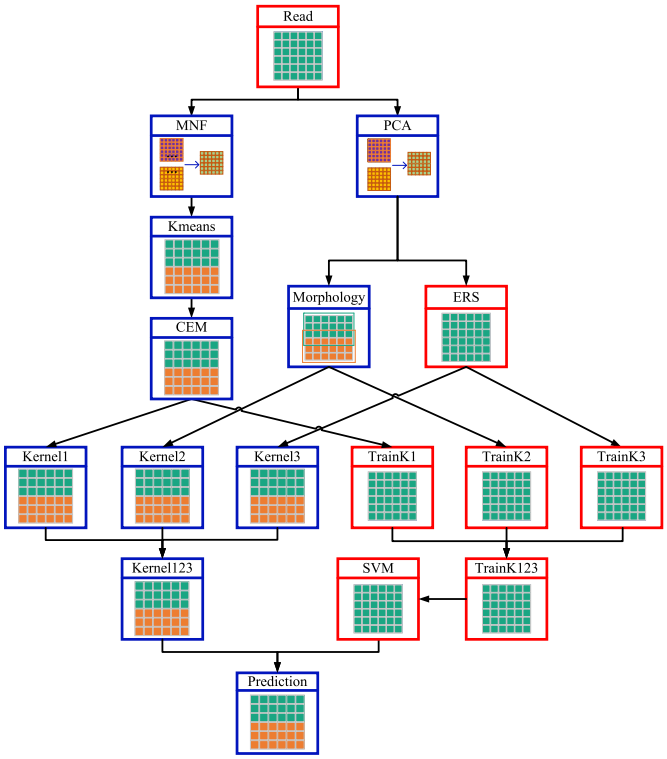


Fig. 1. Serial processing flow of the SPS-FC method.

B. Distributed and Parallel Implementation on Spark

We use Apache Spark, an advanced cloud computing architecture, as the platform for parallel processing of HSI big data. Spark uses resilient distributed datasets (RDDs) as the fundamental data structure to support fault-tolerant, in-memory cluster computation [37]. In addition, Spark relies on a driver program to manage a cluster of worker nodes to support distributed computing. The workers can store RDD partitions in random access memory (RAM) across operations. The driver is responsible for defining, invoking, and tracking these RDD partitions [38]. During runtime, the driver program launches multiple workers to load RDD data from a distributed file system and holds the loaded data partitions in memory.

Considering that the extraction of superpixel-level features in the SPS-FC method is largely dependent on the spatial

Algorithm 1: Serial SPS-FC

1 Apply MNF transformation [34] to the original HSI for the purpose of HSI denoising.
2 Perform unsupervised clustering using the k-means algorithm and use the cluster centers as endmembers.
3 Use CEM [35] algorithm to calculate the abundance fractions, that is, the subpixel-level features (denoted by `Feature1`).
4 Perform PCA on the original HSI for dimensionality reduction.
5 Perform morphological operations, e.g., erosion and reconstruction, on each dimension and combine the results on all dimensions to obtain the pixel-level features (denoted by `Feature2`).
6 Employ ERS algorithm [36] to perform superpixel segmentation, in order to generate the label matrix.
7 Rely on the label matrix to compute the mean values of spectral vectors, that is, the superpixel-level features (denoted by `Feature3`).
8 Apply radial basis functions to feature1, feature2, and feature3 to obtain the corresponding subpixel-, pixel- and superpixel-level kernels (denoted by `Kernel1`, `Kernel2`, and `Kernel3`, respectively). Fuse the three kernels form one composite kernel `Kernel123`.
9 Generate the training set of fused features `TrainK123` through random sampling on `feature1`, `feature2`, and `feature3`. The final training model is obtained by tuning SVM model parameters.
10 Determine the final labels of pixels according to the composite kernel and training model.

Algorithm 2: Parallel SPS-FC

1 Load and convert the original HSI data from HDFS. The converted data in RDD format are denoted by `DataRDD`.
2 Perform a 'map' operation on `DataRDD` for the parallel processing of MNF transformation on all workers and obtain `mnfRDD`.
3 Perform a 'map' operation on `mnfRDD` to implement the parallel k-means clustering. Broadcast the clustering results `KMeansResult` to all workers.
4 Perform another 'map' operation on `mnfRDD` to implement the parallel CEM algorithm. Due to the spatial correlations among pixels, it is infeasible to directly partition the pixel matrix to perform matrix multiplication $\texttt{mnfRDD}\times\texttt{mnfRDD}^T$. Thus, we compute $\texttt{mnfRDD}\times\texttt{mnfRDD}$ on driver end and broadcast the results `Rinv`. All workers compute in parallel $\texttt{KMeansResult}\times\texttt{Rinv}\times\texttt{mnfRDD}$ to obtain the subpixel-level features `Feature1RDD`.
5 Perform another 'map' operation on `DataRDD` for the parallel processing of PCA method and obtain `pcaRDD`.
6 Perform a 'reduce' operation on all `pcaRDD` partitions to merge the results. The merged data are stored on the driver end and denoted by `pcaData`.
7 Perform a 'map' operation on `DataRDD` for the parallel processing of erosion and reconstruction on the partitioned data. The results are merged to obtain pixel-level features `Feature2RDD`.
8 On the driver end, generate the label matrix by using `pcaData` as the input to ERS algorithm. Compute the superpixel-level features `Feature3RDD` based on the label matrix. Broadcast `Feature3RDD` to all workers.
9 Perform 'map' operations on `Feature1RDD`, `Feature2RDD`, and `Feature3RDD` to generate kernel matrices `Kernel1RDD`, `Kernel2RDD`, and `Kernel3RDD`, respectively. Use the generated RDDs to form a new kernel matrix `K123RDD`.
10 Sample a set of data from `K123RDD` and store the training samples on the driver end. Train the SVM model and broadcast it to all workers.
11 Perform a 'map' operation on `K123RDD` for the parallel processing of pixel classification and obtain `FinalRDD`.
12 Perform a 'collect' operation on the driver end to store `FinalRDD` into HDFS.

correlation information, simply partitioning the original HSI data for distributed processing (without preserving the correlation information) may have a significant impact on the final classification results. However, for certain tasks, for example, MNF transformation and k-means clustering, the original dataset can be partitioned to facilitate distributed processing. In general, a massive HSI data processing procedure can be implemented in a distributed fashion by the following steps.

1) First, we decompose the original HSI dataset into many spatial-domain partitions and store the data partitions on Hadoop's distributed file system (HDFS), which the filesystem adopted by Spark. Each data partition stored in HDFS is assigned an offset value for fast lookup. In this manner, the Apache driver program is able to read the data partitions of the original HSI dataset.
2) When data partitions have been loaded into memory, we perform a "map" operation to convert them into RDD format. The converted RDD instances can be then processed by the workers simultaneously. Managed by the driver program on the master node, all workers work simultaneously to process the partitioned data on individual RDD instances. The workers submit the results to the master after finalization.
3) The driver program performs the "reduce" operation to merge the results produced by all worker nodes. The "reduce" operation analyzes all key-value pairs and combines the values that shared the same key following the driver program. The final result will be broadcasted to all worker nodes and will be stored into HDFS.

To be specific, the detailed procedures of the distributed and parallel SPS-FC implementation are described in Algorithm 2. As can be observed in most steps, the key idea is to identify those subtasks whose computation load can be decomposed into multiple load partitions and use the distributed computing scheme to accelerate their execution. Note that for certain tasks, we need to perform necessary transformations to the original data to ensure their task divisibility. Fig. 2 summarizes the overall workflow of the parallel version of the SPS-FC method. Compared with the serial workflow in Fig. 1, a number of "map" and "reduce" operations as well as RDD partitions are introduced to enable the distributed and parallel implementation of the algorithm.

Despite the efficiency of parallel processing on Spark, intertask and intratask parallelisms enable further improvements in terms of distributed processing efficiency by means of scheduling strategies. On the one hand, there are many dependencies among the subtasks of the processing workflow that impose an order of precedence on their execution. On the other hand, certain divisible tasks can be executed in parallel, and are, therefore, in demand of an appropriate allocation of computing resources. For example, as it can be observed from Fig. 2, the three feature extraction tasks (`Feature1RDD`,

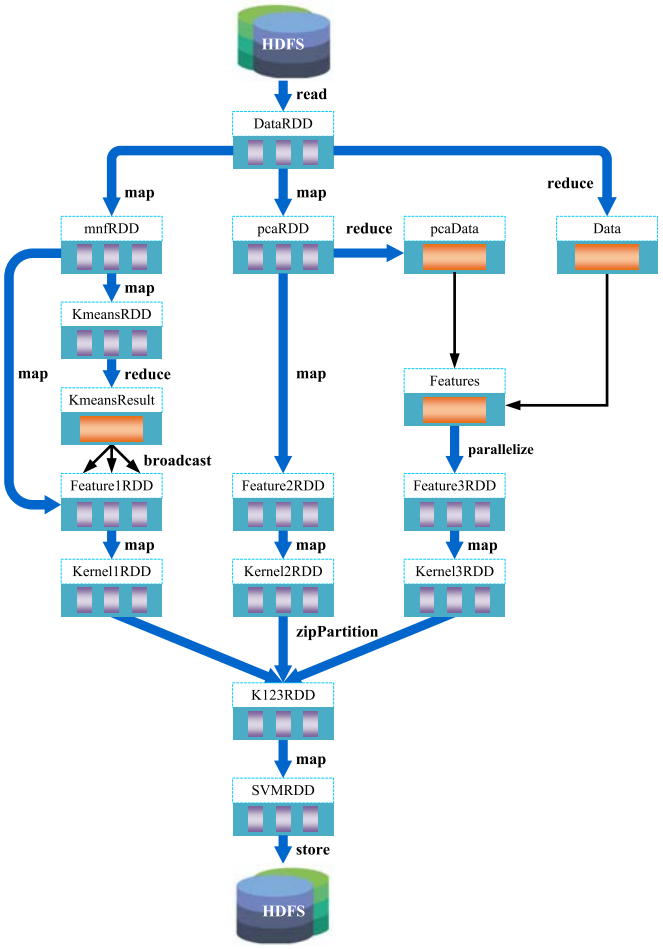


Fig. 2. Parallel processing workflow of SPS-FC method on Spark.

`Feature2RDD`, and `Feature3RDD`) can be executed in parallel, but they all take precedence over `K123RDD` during task execution. In subsequent sections, we propose our own scheduling model and effective scheduling algorithms to address the aforementioned issues.

III. Divisible Scheduling Model for Optimal Distributed Processing

According to the preceding analysis, it is crucial to employ scheduling techniques to further improve the computational efficiency of the HSI classification method by exploiting the intertask and intratask parallelisms during its distributed processing. In this section, we formulate a divisible scheduling model that seeks for the optimal scheduling solution of task assignments and partitioning factors for achieving the best acceleration rate on clouds.

We first discuss the parallelism among tasks. The processing flow of an HSI application, for example, the SPS-FC classification flow illustrated in Fig. 1, is typically composed of a set of subtasks. Also, there may exist dependencies among the tasks belonging to a specific application workflow. DAG is a commonly used representation to characterize the application workflow [39]. Specifically, a DAG can be denoted by $G = (V, E)$, where $V = \{v_1, v_2, \ldots, v_n\}$ denotes a set of tasks to be allocated onto PEs, and $E = \{(i, j)\}$ imposes a set of precedence constraints upon any two communicating tasks. In addition, the duration time of each task v_i is denoted by w_i. Referring to Fig. 1, the classification process of the SPS-FC method is apparently a typical workflow that complies with the definition of DAG.

As mentioned previously, the scheduling procedure is to determine the optimal mapping relationship between tasks and PEs. In this work, we propose employing scheduling strategies to minimize the total execution time of the SPS-FC classification process. This performance measure is known as the schedule length or the makespan of task execution. When determining the mapping from tasks to PEs, it is required to satisfy all precedence constraints. For any edge $\{(i, j)\}$ belonging to the DAG, the precedence relation between task v_i and task v_j implies the following constraint:

$$s_i + w_i \leq s_j \quad \forall (i, j) \in E \tag{1}$$

where s_i and s_j stand for the start times and end times for v_i and v_j, respectively, and w_i denotes v_i's actual execution time on the PE it is mapped onto. Note that for a divisible task, its actual execution time is not a constant value, but varying value depending on its partitioning factor.

We then discuss the parallelism within a divisible task. For a divisible task, due to the overhead of data communications among PEs, the speedup achieved by the MapReduce mechanism does not increase linearly with more data partitions. In other words, the ratio of speedup to partition count would decrease due to the increasing communication overhead. As will be demonstrated in our experimental results, there is a critical number of data partitions that corresponds to the best tradeoff between communication overhead and computational efficiency. With a limited number of PEs, it is necessary to determine the number of partitions for distributing the computation load of each divisible task or, equivalently, the number of PEs that should be assigned for executing each divisible task. In this work, we define this set of decision variables as *partitioning factors*. If several divisible tasks can be executed in parallel, then the scheduling model has to assign appropriate partitioning factors for divisible tasks, according to their computation workloads. Otherwise, the computational efficiency of distributed processing would be compromised due to the inefficient utilization of cloud computing resources. Note that the partitioning factors have a direct impact upon not only the number of required PEs but also the varying execution time of partitioned tasks. For instance, suppose that there are two divisible tasks in parallel, and their critical values of partitioning factors are both below the number of available PEs. In this scenario, simply assigning all PEs to execute a single divisible task may introduce a considerable communication overhead. It would be more judicious to let the partitioning factors be close to their critical values to achieve higher efficiency of parallel computing.

Taking into account the above-mentioned two concerns, Fig. 3 illustrates the overall framework of the proposed divisible scheduling problem, in which task assignments and partitioning factors are both incorporated as decision variables for minimizing the total execution time of an HSI application.

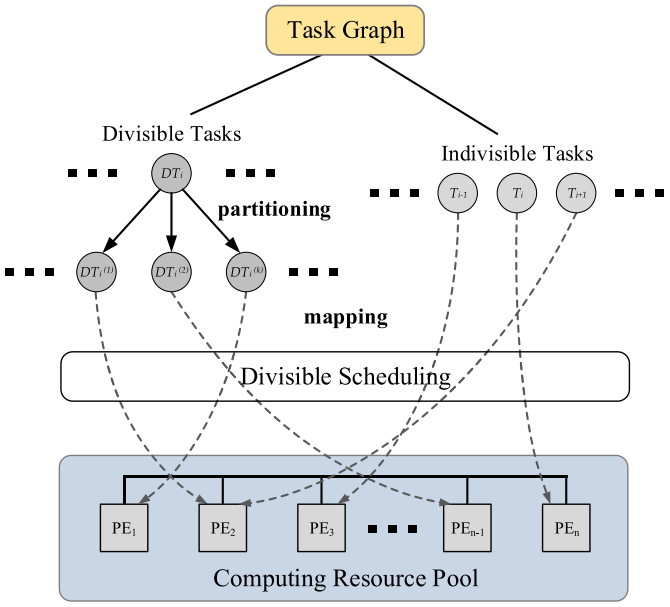


Fig. 3. Framework of divisible scheduling.

In order to represent the SPS-FC processing flow using a DAG, we first identify the divisible tasks that can be partitioned into multiple small tasks. The partitioning factors of divisible tasks are a set of decision variables to be determined by the scheduling model. All partitioned tasks, together with the indivisible tasks, will be allocated to a limited number of PEs according to a mapping function, which represents another set of decision variables. The divisible scheduling problem in this work is to determine the optimal values of these decision variables under the constraint of limited computing resources.

In consistence with the scheduling framework in Fig. 3, we now present a rigorous mathematical model that formulates the proposed divisible scheduling model as an integer program (IP). The capacity of the resource pool is R. Any resource k ($k = 1, 2, \ldots, R$) is identical to each other. Given a workflow ω with n tasks, $e_{pi} = 1$ denotes that task v_p is a predecessor of task v_i, and $e_{pi} = 0$ otherwise. We define $P_i = \{v_p | e_{pi} = 1\}$ to denote the predecessor set of task v_i. We define two binary variables x_{it} and y_{ik} as the decision variables. $x_{it} = 1$ denotes that task v_i is in execution on time slot t, and $x_{it} = 0$ otherwise. $y_{ik} = 1$ indicates that the kth PE is allocated to execute task i, and $y_{ik} = 0$ otherwise. Accordingly, the total number of PEs for processing task v_i is given by

$$r_i = \sum_{k=1}^{R} y_{ik}. \tag{2}$$

Then, the execution duration when r_i resources are allocated to the task is denoted by d_{r_i}. Note that not all tasks can be executed in parallel. As mentioned in Section II-B, those tasks that can be processed in parallel are identified as divisible tasks, and the other tasks are indivisible ones. Obviously, the start time is given by $s_i = \arg\min\{t | x_{it} = 1\}$. In order to comply with the precedence relationships between task v_i and its predecessors, for task i's any predecessor task $\forall v_p \in P_i$, we have the following constraint:

$$\arg\min_t \{x_{it} = 1\} \geq \max\left\{\arg\min_t \{x_{pt} = 1\} + d_{r_i}\right\}. \tag{3}$$

The minimal value of s_i is in fact task v_i's earliest start time. We use EST_i to denote it and determine it by

$$EST_i = \max\left\{s_p + d_i^{r_i}\right\} \quad \forall v_p \in P_i. \tag{4}$$

With the symbols and concepts defined above, we finally present the following optimization model for the proposed divisible scheduling model:

$$\begin{aligned}
\text{min.} \quad & c_\omega = \max\{S_i + d_i\} \\
& = \max\left\{\arg\min_t \{x_{it} = 1\} + D_i\right\} && (5) \\
\text{s. t.} \quad & \arg\min_t \{x_{it} = 1\} \\
& \geq \max\left\{\arg\min_t \{x_{pt} = 1\} + d_{pj}\right\} \quad \forall v_p \in P_i && (6) \\
& x_{it} \in \{0, 1\}, \ r_i \in \{1, 2, \ldots, R\} && (7) \\
& \sum_{i=1}^{n} x_{it} r_i \leq R && (8)
\end{aligned}$$

where the optimization objective is to minimize the makespan c_ω of an application workflow. Integer variables x_{it} and r_i ($i = 1, 2, \ldots, n$), which are the two sets of decision variables in the optimization model, stand for the assignment of tasks onto PEs and partitioning factors for divisible tasks, respectively. In addition, task precedence constraints and computing resource constraint are imposed.

IV. Scheduling Algorithm

This section provides the details about the metaheuristic algorithms, which are built upon the QEA framework, specifically for solving the formulated divisible scheduling problem. QEA is advantageous to many existing metaheuristics since the Q-bit representation in QEA can enhance population diversity and solution exploration capability. In addition to the standard QEA (SQEA) scheduling algorithm, we further design a hybrid QEA (HQEA) scheduling metaheuristic by incorporating a solution initialization scheme and a solution exploration method into SQEA. Before presenting the details of the algorithm, we first provide a brief introduction to quantum computing, which is the foundation of our proposed QEA-based algorithms.

A. Preliminaries of Quantum Computing

Q-bit is the fundamental concept in the QEA framework. Q-bit is the smallest information unit stored in a two-state quantum computer. The main characteristic of Q-bit is that it could be in the "1" state or in the "0" state, or in the linear superposition of the two states. The probabilities of a Q-bit being "0" state and state "1" can be represented by $|\alpha|^2$ and $|\beta|^2$, respectively, where α and β are two complex numbers satisfying $|\alpha|^2 + |\beta|^2 = 1$ [40]. A Q-bit individual as a string of n Q-bits can be formulated as $\begin{pmatrix} \alpha_1 & \alpha_2 & \ldots & \alpha_n \\ \beta_1 & \beta_2 & \ldots & \beta_n \end{pmatrix}$, where $|\alpha|^2 +$

$|\beta|^2 = 1$ holds for $i = 1, 2, \ldots, n$. We can convert a Q-bit individual into a binary string by collapsing each state into either "0" state or "1" state. The exact state of each Q-bit depends on the values of $|\alpha|^2$ and $|\beta|^2$ [41].

The state of a Q-bit can be changed by a quantum gate [40]. A quantum gate is a reversible gate and can be denoted as a unitary operator U acting on the Q-bit states, satisfying $U^+U = UU^+ = I$, in which U^+ is the Hermitian adjoint of U. One commonly used quantum gate is rotation gate, which has the following format:

$$U(\Delta\theta) = \begin{bmatrix} \cos(\Delta\theta) & -\sin(\Delta\theta) \\ \sin(\Delta\theta) & \cos(\Delta\theta) \end{bmatrix}$$

where $\Delta\theta$ is the rotation angle. Accordingly, the updating rule for the Q-bit coefficients can be expressed as

$$\begin{bmatrix} \alpha' \\ \beta' \end{bmatrix} = U(\Delta\theta)\begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} \cos(\Delta\theta) & -\sin(\Delta\theta) \\ \sin(\Delta\theta) & \cos(\Delta\theta) \end{bmatrix}\begin{bmatrix} \alpha \\ \beta \end{bmatrix}. \quad (9)$$

The value of $\Delta\theta$ can be determined in several ways, for example, lookup table [40], or the distance between the objectives of the current best Q-bit individual and its offsprings [42].

B. Solution Representation

In this work, we use a binary string converted from a Q-bit individual to represent a scheduling solution. The converted binary string would be of size $n \times m$, where n is the number of tasks belonging to a given DAG, R is the number of PEs for executing the tasks, and m is an integer satisfying $2^{(m-1)} < R \leq 2^m$. We can further convert each binary string into a decimal string of length n by decoding each m consecutive bits in the binary string to a decimal number. The obtain decimal string can represent a valid scheduling solution as it specifies a unique mapping from tasks to available computing resources. In case that the decimal value converted from the binary string may be greater than R, we use a modulo operation to ensure a valid candidate solution.

C. Solution Initialization

We employ the earliest start time first (ESTF) rule to generate an initial solution, for the purpose of providing the HQEA metaheuristic with a good start. To this end, we calculate the ESTs of all tasks to be scheduled according to (4) and sort all tasks by their EST values in a nondescending order. For indivisible tasks whose execution durations are prespecified, their ESTs can be uniquely determined. On the other hand, the execution duration of divisible tasks varies depending on the number of allocated PEs, a divisible task's EST is also a varying value. In this work, we use task durations in the serial processing flow (i.e., the execution times on a single PE) to calculate ESTs and sort the tasks.

D. Solution Evaluation

Given a candidate solution in the format of a binary string converted from a Q-bit individual, we can obtain the corresponding solution to the divisible scheduling problem and further evaluate the quality of this solution, that is, the makespan of HSI classification that the scheduling solution leads to. Solution evaluation depends on a task sequence *ts* and processes tasks in *ts* sequentially. This procedure selects all the tasks whose predecessor tasks have been completed and adds these tasks into a set Φ. If a task v_i in Φ is an indivisible task, the q_ith PE, which is selected by mapping the ith m consecutive bits to a decimal number, would be designated for processing v_i. For those divisible tasks in Φ, we allocate the other $R - l$ PEs (l is the number of PEs that have been assigned for indivisible tasks) to them according to their serial execution durations, with the guarantee that each task is assigned to exactly one PE. For example, suppose there are three divisible tasks whose serial durations are a, b, and c, respectively. The numbers of PEs assigned for executing them (i.e., partitioning factors) are $a \times (R-l)/(a+b+c)$, $b \times (R-l)/(a+b+c)$, and $c \times (R-l)/(a+b+c)$, respectively. In case that $a \times (R-l)/(a+b+c)$, $b \times (R-l)/(a+b+c)$, and $c \times (R-l)/(a+b+c)$ are not integers, we apply floor operations $\lfloor a \times (R-l)/(a+b+c) \rfloor$, $\lfloor b \times (R-l)/(a+b+c) \rfloor$, and $R - \lfloor a \times (R-l)/(a+b+c) \rfloor - \lfloor b \times (R-l)/(a+b+c) \rfloor$. In this manner, we can obtain the duration times of divisible tasks (i.e., $d_i^{r_i}$) according to their partitioning factors and calculate their end times by $e_i = EST_i + d_i^{r_i}$. Once all tasks in *ts* have been scheduled, the makespan of application workflow can be determined by $\max\{e_i | i = 1, 2, \ldots, n\}$. Algorithm 3 summarizes the algorithmic details about solution evaluation with a low complexity of $\mathcal{O}(n)$.

Algorithm 3: Solution Evaluation

Input: A candidate solution ζ, and a prespecified task sequence *ts*
Output: The corresponding makespan $c_\omega(\zeta)$

```
Convert ζ into a decimal string ζ_d;
Set the i-th decimal number in ζ_d as q_i;
Initialize an empty task set Φ;
while (ts is not empty) do
    Select all the tasks in ts of which the predecessor tasks
    have been completed and add them into Φ;
    Allocate the q_i-th PE to each indivisible task v_i in Φ;
    Allocate the remaining PEs to divisible tasks according to
    their serial execution durations;
    Determine the execution duration d_i^{r_i} of each task in Φ
    according to the number of allocated PEs r_i;
    Calculate the end time of each task in Φ by
    e_i = EST_i + d_i^{r_i};
    Update c_ω(ζ) = max{c_ω(ζ), max{e_i | i ∈ Φ}};
    Remove all tasks in Φ from ts;
return c_ω(ζ);
```

E. Solution Exploration

We use an insertion-based exploration method (IEM) to generate new task sequences based on a given one. For a task sequence *temp* consisting of n tasks, IEM removes the uth task ($u = 1, 2, \ldots, T$) from *temp* and later reinserts this task into *temp* at every position other than its original position. In this manner, IEM is capable of generating $n - 1$ different task sequences, which will be used by the SQEA algorithm to explore more high-quality solutions. If a certainly generated task sequence *gts* finds a scheduling solution better than the given task sequence *temp*, SQEA would replace *temp* by *gts*.

Algorithm 4: SQEA Scheduling

```
   Input: an application workflow G, and ts
   Output: The minimal makespan c_ω(best)
1  Initialize all the individuals in Ψ;
2  for each φ ∈ Ψ do
3      Convert ζ into the corresponding solution ζ_φ;
4      Perform Algorithm 3 to evaluate ζ_φ;
5  Record the best solution in Ψ as ζ_best;
6  Set ζ_best's corresponding makespan as c_ω(best);
7  while (termination criterion is not met) do
8      for each φ ∈ Ψ do
9          Use the rotation gate to update φ;
10         Convert the updated φ to obtain its corresponding
           solution ζ_φ;
11         Perform Algorithm 3 to evaluate ζ_φ;
12         if (φ is better than ζ_best) then
13             Set ζ_best ← φ and c_ω(best) ← c_ω(φ);
14     return c_ω(best);
```

Algorithm 5: HQEA Scheduling

```
   Input: an application workflow G
   Output: The shortest makespan c_ω(best)
1  Apply the ESTF rule to generate a task sequence ts;
2  Set c_ω(best) ← SQEA(ts);
3  Set temp ← ts;
4  Initialize an empty set of task sequences Ψ;
5  for (u ← 1 to n) do
6      Remove the uth task in task sequence temp;
7      Reinsert the removed task into temp at every position other
       than the original one;
8      Add all n − 1 generated task sequences into Ψ;
9      for (each gts ∈ Ψ) do
10         Set c_ω(gts) ← SQEA(gts);
11         if (c_ω(gts) < c_ω(best)) then
12             Set temp ← gts;
13             Set c_ω(temp) ← c_ω(best) ← c_ω(gts);
14 return c_ω(best);
```

IEM repeats the same procedure for all other tasks in *temp* to generate new task sequences until the Tth task in *temp* has been used for solution exploration.

F. Algorithm Description

We now present the proposed SQEA algorithm and its extended version HQEA. Algorithms 4 and 5 describes the details about SQEA and HQEA, respectively. Referring to Algorithm 4, after initializing all Q-bit individuals (line 1), SQEA converts each individual into the corresponding solution (line 3) and uses the solution valuation method in Algorithm 3 to calculate the quality of each converted solution (line 4). Then, as long as the termination criterion is not met (line 8), SQEA iteratively applies the rotation gate to update each individual, converts the updated individual into a scheduling solution, and evaluates solution quality (lines 10–12). The best solution found during the iterative procedure is identified as the final scheduling solution and the makespan corresponding to the final solution is returned by SQEA. It is worth emphasizing that during the solution evaluation procedure, SQEA uses a randomly generated task sequence *ts* to assign tasks. In other words, SQEA depends on *ts*. Thus, the effectiveness of SQEA depends on the quality of task sequence *ts*.

To address the above-mentioned limitation of SQEA, we construct HQEA by incorporating the solution initialization and IEM strategies. As described in Algorithm 5, starting with a task sequence *ts* generated by the ESTF rule (line 1), HQEA employs SQEA to produce a high-quality scheduling solution (line 2). Based on this solution, SQEA uses IEM to generate a set of new task sequences and inputs each generated sequence to SQEA to explore better solutions (lines 5–16). The final result of HQEA is the makespan corresponding to the best-explored solution (line 17).

V. Experimental Results

The performance of the scheduling-guided method was evaluated on a cloud system implemented on a Spark cluster consisting of one master node and six slave nodes, which are all created by virtualization tools. The master node is built on a host equipped with a 24-core Intel Xeon E5-2680 v3 CPU operating at 2.5 GHz and 242-GB memory. We deploy six slave nodes on three IBM blade servers. Each of the two slave nodes on a blade server is assigned with an Intel Xeon E5-2680 v3 CPU and 240-GB memory. All Spark nodes have installed Ubuntu 16.04 as the operating system. In Spark configuration, we launch 20 workers on each slave node and each worker is assigned with a single core.

We use the widely used and publicly available "University of Pavia" HSI dataset to evaluate the performance of the proposed scheduling-guided approach for accelerating the SPS-FC classification flow in terms of both classification accuracy and computational efficiency. The "University of Pavia" image contains 103 spectral bands after discarding the most seriously noisy bands. Each band is of size 610×340, and the data size of the original image is 226 MB. The spatial resolution of this image is 1.3 m, and the spectral coverage ranges from 0.43 to 0.86 μm. We randomly select 60 samples from these nine prelabeled categories to generate a training set with 540 samples for evaluation purposes.

To fully justify the effectiveness of the proposed approach, we perform additional experiments on the "Indian Pines" HSI dataset by using the spatial correlation regularized sparse representation classification (SCSRC) method [43]. SCSRC is a typical HSI classification method that relies on the alternating direction method of multipliers (ADMMs) [44] to solve the sparse representation regularization problem. We select 200 spectral bands out of the 220 bands contained in the "Indian Pines" image for extracting discriminative information. Each band consists of 145×145 pixels.

A. Evaluation of Parallel Implementation

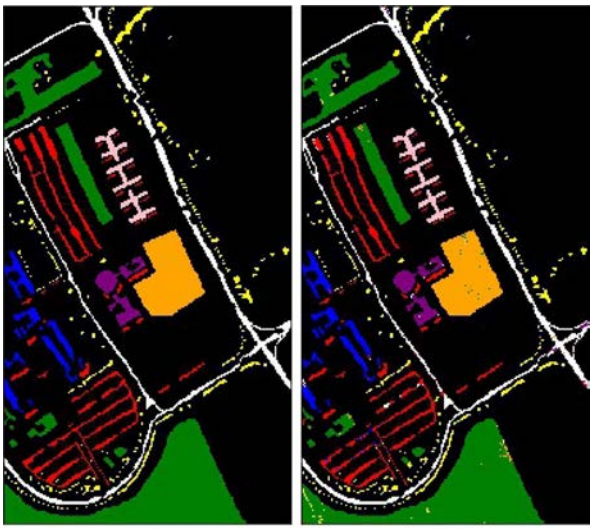
As the very first step, we first present the classification results by performing the SPS-FC method in a distributed way on Spark. Following the parallel processing flow in Section II, the divisible tasks of the SPS-FC method are partitioned and processed on multiple workers relying on Spark's MapReduce mechanism. For the sake of comparison, we also implemented

TABLE I
CLASSIFICATION ACCURACIES BY USING THE SERIAL AND PARALLEL SPS-FC METHODS

	Serial SPS-FC	Parallel SPS-FC				
		4 workers	8 workers	16 workers	32 workers	64 workers
Accuracy	95.59%	95.57%	95.58%	95.56%	95.55%	95.57%

TABLE II
RUNTIME STATISTICS FOR ALL DIVISIBLE TASKS (SECONDS)

Divisible Task Name	Serial SPS-FC	Parallel SPS-FC				
		4 workers	8 workers	16 workers	32 workers	64 workers
MNF	10.41	4.88	3.70	3.32	3.18	3.14
k-means	16.66	7.84	5.74	3.65	3.54	2.70
CEM	3.07	0.48	0.39	0.29	0.25	0.33
PCA	1.88	1.15	0.83	0.75	0.76	0.78
Erosion	16.63	6.81	4.42	3.60	3.26	3.04
Reconstruction	8.16	3.41	2.21	1.80	1.76	1.71
Spectral Mean	0.18	0.19	0.18	0.18	0.23	0.22
Kernel1	10.87	3.28	1.42	0.72	0.55	0.42
Kernel2	26.57	6.25	2.50	1.11	0.66	0.41
Kernel3	45.26	10.32	4.33	2.13	1.41	1.02
K123	0.81	0.27	0.11	0.05	0.03	0.02
Prediction	31.31	6.52	5.82	2.60	1.17	0.87
Total	171.81	51.40	31.65	20.20	16.80	14.66
Speedup	——	3.34×	5.42×	8.51×	10.22×	11.72×



(a) (b)

Fig. 4. Classification results by using the (a) serial and (b) parallel SPS-FC methods.

the serial version of the SPS-FC method by executing all the subtasks on a single computing node. Fig. 4(a) and (b) shows the classification results obtained by the serial SPS-FC and parallel SPS-FC, respectively. We observe that the result by parallel SPS-FC matches well with the result in the serial version. In the parallel implementation, all 64 workers are employed to produce the classification result in Fig. 4(b).

We further investigate the classification accuracies by varying the number of workers, as listed in Table I. Considering that the random sampling operation in the classification flow may lead to a slight difference in classification result even with the same experimental setup and Spark configuration, we repeat each set of experiments five times and calculating the average accuracy. When executing the parallel SPS-FC on Spark, we notice that the classification accuracy is almost unchanged with different worker counts. More importantly, the difference in classification accuracy between the parallel SPS-FC and serial SPS-FC is negligible, demonstrating the effectiveness of processing HSI data in parallel on Spark.

We then justify the advantage of the parallel SPS-FC method over the serial flow in total execution time. For all divisible tasks in SPS-FC processing flow, Table II provides the execution times by employing different numbers of workers on Spark. The original HSI data are split into RDD partitions according to the number of available workers. In other words, without the consideration of scheduling strategies yet, the partitioning factor is equal to the worker count. The execution time measured for the serial SPS-FC is also provided for comparison. The runtime statistics in columns three to seven of Table II show that the total execution time decreases as the number of workers grows. The last two rows record the total execution time of all divisible tasks, as well as the speedups over the serial SPS-FC. When the worker count is relatively small (below 32), an obvious increase in speedup can be observed. However, if the worker count exceeds a certain value, the achieved speedup becomes less significant. For instance, employing 32 workers only achieves a 10.22× speedup, and the speedup obtained by deploying 64 workers is merely 11.72×. This observation is due to the fact that more worker nodes would lead to the increasing time overhead induced by frequent communications among Spark nodes. The

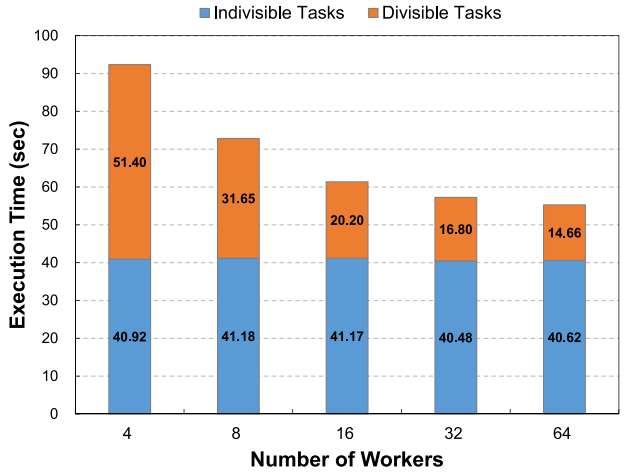


Fig. 5. Execution times with different numbers of workers.

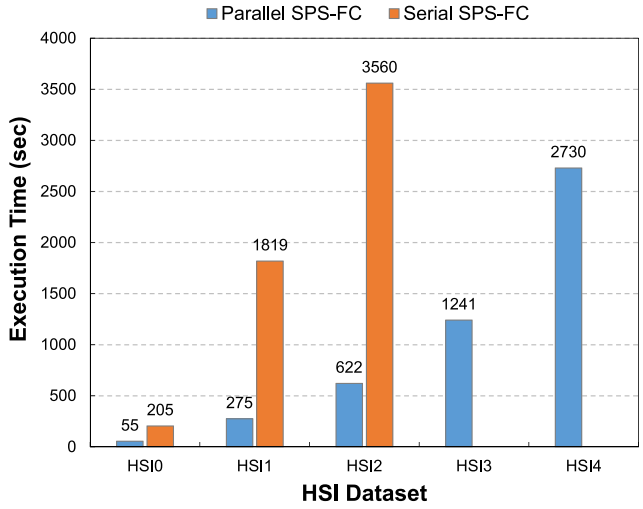


Fig. 6. Execution times for different HSI datasets. The results for HSI3 and HSI4 in serial SPS-FC are unavailable due to the large data volume.

amount of time spent in data transmissions accounts for a substantial proportion of the total execution time. In addition, certain divisible tasks involve frequent data exchanges among Spark nodes, for example, MNF, *k*-means, and morphological operations, and therefore result in limited improvements of computational efficiency.

Fig. 5 further presents the total execution time for all divisible and indivisible tasks. As shown in the figure, the runtime for indivisible tasks is independent of the worker count, as they have to be executed serially on a single PE. Due to these indivisible tasks that cannot be accelerated, the total execution time becomes stable when the worker count exceeds 16. It is, therefore, reasonable to predict that no further significant reduction in execution time would be achieved by continuing to increase the worker count. This important observation indicates that the overall improvement in computational efficiency in accelerating an HSI application is greatly dependent on the application's processing workflow, more specifically, on the relationship between the runtime for indivisible tasks and the runtime for all tasks.

We further examine the scalability of the parallel SPS-FC method when the HSI dataset is of large volume. For this purpose, we mosaick the original "University of Pavia" dataset to generate large-scale HSI datasets of different sizes. The generated large-scale datasets are denoted by HSI1 ($610 \times 3400 \times 103$, 2.21 GB), HSI2 ($610 \times 6800 \times 103$, 4.42 GB), HSI3 ($610 \times 13600 \times 103$, 8.82 GB), and HSI4 ($610 \times 27200 \times 103$, 17.66 GB), respectively. We increase the data size to observe the computational efficiency when performing SPS-FC classification. In this set of experiments, we fix the worker count at 64. Fig. 6 presents the runtime statistics for executing the serial and parallel SPS-FC methods on different HSI datasets. For HSI1 and HSI2, the parallel SPS-FC achieves speedups of 5.72× and 6.61×, respectively. For HSI3 and HSI4, since the serial SPS-FC is incapable of handling HSI data of this high volume, the runtime results are not presented. The results in Fig. 6 also show that the parallel SPS-FC is scalable when processing large-scale HSI datasets.

B. Evaluation of Scheduling-Guided Acceleration

Despite the promising improvement in computational efficiency achieved by the parallel implementation on Spark, it is worth emphasizing several issues that may limit the full usage of the entire cloud system. On the one hand, the precedence constraints among tasks may prevent the parallel processing of multiple tasks, leading to low utilization of the cloud system. For instance, if an indivisible task takes precedence over a number of succeeding tasks on the DAG, all its successors have to wait until the indivisible task has completed execution on a single PE and then start execution. Such dependency on indivisible tasks may cause a long period of system idle time and slow down task execution speed. On the other hand, the execution speed of a divisible task is closely dependent on its partitioning factor. As the partitioning factor increases, a significant amount of communication overhead, including MapReduce job initialization, synchronization, data transmission, etc., would also be introduced.

We show that the total execution time of the parallel SPS-FC flow can be further reduced by employing the proposed scheduling strategies. In consistency with the scheduling model in Section III, we use the concept of "makespan" to denote the total time for executing the complete SPS-FC flow. Specifically, we use the two scheduling algorithms, SQEA and HQEA described in Section IV, to solve the divisible scheduling problem formulated in (5)–(8), and evaluate the task execution times. Fig. 7 presents the makespans obtained by SQEA and HQEA scheduling algorithms for different numbers of workers. The makespan of the parallel implementation without using scheduling strategies is also provided for comparison. We can observe that the considered scheduling strategies effectively reduce the makespan of task execution by assigning appropriate partitioning factors and mapping relations. Besides, the reduction in makespan increases as the number of workers grows. When the worker count reaches 64, the SQEA algorithm achieves an improvement of 16.28% compared with the "no scheduling" case. Moreover, by introducing the insertion-based search heuristic,

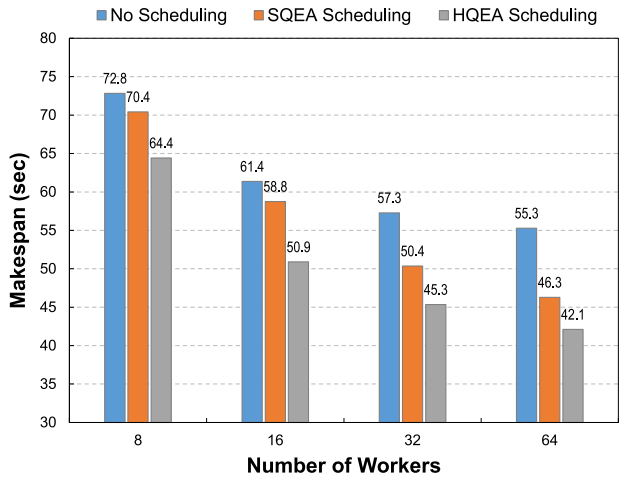


Fig. 7. Makespans with and without using scheduling strategies with different numbers of workers.

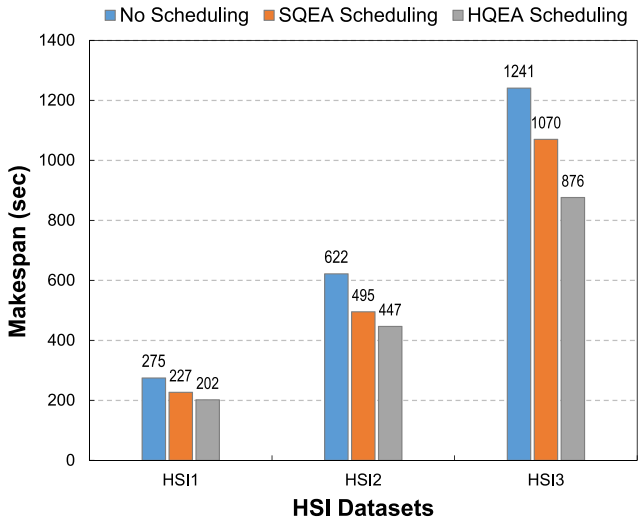


Fig. 8. Makespans with and without using scheduling strategies for different HSI datasets.

the HQEA algorithm further reduces the makespan by 9.01%, resulting in a total makespan reduction of 23.82%.

We perform another set of experiments to verify the scheduling results on HSI datasets of different sizes. Fig. 8 provides the task execution makespans by using no scheduling, SQEA scheduling, and HQEA scheduling strategies, respectively. All makespan results are obtained by deploying 64 workers. The results show that HQEA and SQEA algorithms outperform the "no scheduling" case in terms of reduced task execution makespans. For the large-scale datasets HSI1, HSI2, and HSI3, makespan reductions ranging from 13.74% to 20.34% are achieved by using the SQEA strategy, whereas the makespan reductions achieved by the HQEA strategy range from 26.50% to 29.39%. More importantly, the improvement in makespan by the HQEA strategy is more stable than that by the SQEA strategy as the HSI data volume varies.

Table III finally presents the classification accuracies by using our cloud implementations under the guide of SQEA and HQEA scheduling solutions for various HSI data sizes and worker counts. The results show that with different numbers of workers, the classification accuracies achieved by the scheduling-guided algorithms are close to that achieved by the serial method. The insignificant fluctuation in classification accuracy with regard to worker count is introduced by the random sampling procedure in SPS-FC flow. In addition, since the large-scale datasets are generated by mosaicking the original "University of Pavia" dataset, we observe no much difference among the classification accuracies for HSI1, HSI2, and HSI3.

C. Evaluation of Extended Applicability

The divisible scheduling framework can be applied to other HSI applications as long as we can represent the application flow by a DAG and identify the divisible tasks. We perform additional experiments by using a different HSI classification application and a different HSI dataset to fully justify the applicability of the proposed scheduling-guided approach. The classification application used for evaluation is the SCSRC method that involves memory-consuming matrix computations. The HSI dataset used in this set of experiments is the well-known "Indian Pines" image, which is of size $145 \times 145 \times 200$. The configuration for the Spark cluster is as follows. We deploy a master node on a host machine equipped with a four-core Intel Xeon E5630 CPU and 16-GB memory, and eight slave nodes on four IBM BladeCenter HX5 blade servers. Each slave node is assigned with a six-core Intel Xeon E7-4807 CPU and 15-GB memory. We deploy multiple workers on each slave node. The operating system and software environment for all nodes are the same as in previous experiments.

By analyzing the processing flow of the SCSRC method, we notice that several matrix addition and multiplication operations involved in the classification procedure can be identified as divisible tasks and can be performed in parallel on Spark. We employ the proposed HQEA metaheuristic algorithm to determine the optimal intertask and intratask parallelisms by solving the divisible scheduling problem. We evaluate the performance of the scheduling-guided approach, in terms of classification accuracy, computational efficiency, and scalability, as compared to the serial SCSRC method. By deploying 4, 8, 16, and 32 worker nodes, the classification accuracies obtained by using the HQEA algorithm achieve 98.26%, 98.32%, 98.23%, and 97.68%, respectively. Compared with the serial method that leads to a classification accuracy of 98.28%, the accuracy loss by using the scheduling-guided cloud implementation is less than 0.6%. The comparison results justify that the parallel processing of divisible SCSRC tasks would not affect the accuracy of HSI classification.

Similarly, as in previous experiments, we examine the computational efficiency of the scheduling-guided approach by varying the number of workers. As presented in Fig. 9, the speedup over the serial SCSRC increases with more worker nodes deployed on Spark and becomes less significant as the worker count continues to grow. For instance, with four worker nodes, the total execution time of the SCSRC method by using the HQEA scheduling strategy is 907.04 s, indicating a $4.05\times$ speedup that is close to the number of workers. However, by deploying eight worker nodes, the scheduling-guided approach

TABLE III
CLASSIFICATION ACCURACIES BY USING SQEA AND HQEA ALGORITHMS

Datasets	Serial	8 workers		16 workers		32 workers		64 workers	
		HQEA	SQEA	HQEA	SQEA	HQEA	SQEA	HQEA	SQEA
HSI1	95.59%	95.58%	95.59%	95.58%	95.55%	95.60%	95.58%	95.57%	95.60%
HSI2	95.58%	95.56%	95.55%	95.57%	95.56%	95.58%	95.57%	95.59%	95.55%
HSI3	95.58%	95.59%	95.58%	95.60%	95.54%	95.57%	95.58%	95.57%	95.56%

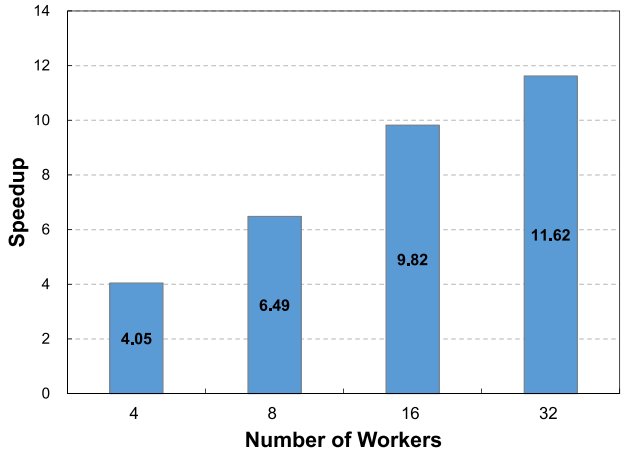


Fig. 9. Speedups over the serial SCSRC with different numbers of workers.

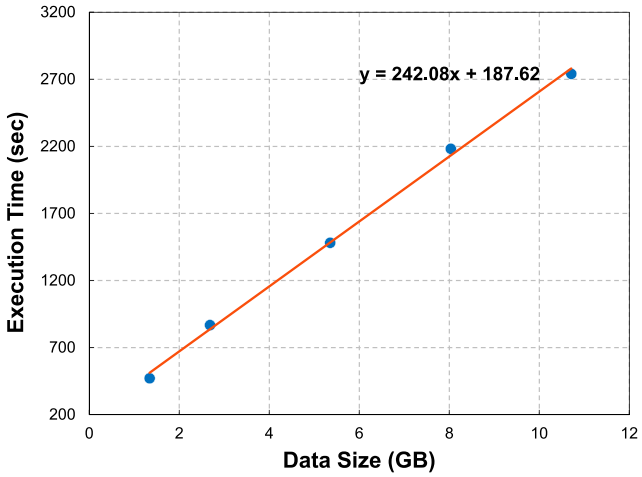


Fig. 10. Execution times of the SCSRC method for different HSI datasets.

achieves a total execution time of 566.25 s and a speedup of 6.49×. When the number of workers increases to 32, the execution time has been reduced to 315.78 s and the corresponding speedup is 11.62×. As analyzed previously, more worker nodes would lead to a considerable amount of communication overhead and in turn a nonlinear improvement in computational efficiency.

In order to further validate the scalability of the scheduling-guided algorithm, we conduct additional experiments on HSI datasets of large sizes. We mosaick the original "Indian Pines" image to generate large-scale HSI datasets, in which the volume of the intermediate data during the SCSRC classification procedure ranges from 1.34 to 10.71 GB. We fix the number of workers as 16 and implement the parallel processing of the SCSRC on Spark guided by the HQEA scheduling results. Fig. 10 provides the total execution times for different data sizes. A linear regression analysis indicates that the increase in execution time is proportional to the increase in dataset size. Such a linear dependency also can be observed by fixing the number of workers at other values. Therefore, we draw the conclusion that the proposed multiobjective scheduling algorithm is scalable to the increasing volume of the HSI dataset.

VI. CONCLUSION

This article proposes a new scheduling-guided parallel processing method for accelerating HSI classification techniques on cloud computing architectures. Specifically, a new divisible scheduling model for fully exploiting the intertask and intratask parallelisms during the distributed processing of a representative HSI classification method is developed. With a limited number of computing resources, the proposed divisible scheduling model takes into account both task assignments and task partitioning factors as decision variables. We formulate the divisible scheduling problem as an optimization framework and further develop effective metaheuristics to solve it. We use two HSI datasets and two representative classification applications to justify the performance of the proposed scheduling-guided approach, in items of classification accuracy, computational efficiency, and scalability to large-scale HSI data.

The divisible scheduling model and algorithms in this work can be easily adapted for other widely used HSI applications, for example, pan sharpening, spectral unmixing, and target detection. As long as we can characterize the processing flow of an HSI application by a DAG and identify the divisible tasks on the DAG, the divisible scheduling model can be formulated to minimize the execution time under the resource constraint. By solving the formulated scheduling problem using the proposed metaheuristic algorithms, an optimized solution of task assignments and partitioning factors can be obtained to accelerate the automatic processing of HSI applications on clouds. As long as the application flow can be divided into a set of subtasks with complicated precedence relations and the computation load of certain tasks can be partitioned and distributed across multiple PEs, our approach would be effective in reducing the execution time on Spark by determining the optimized intertask and intratask parallelisms.

REFERENCES

[1] C. Yu *et al.*, "Hyperspectral image classification method based on CNN architecture embedding with hashing semantic feature," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 12, no. 6, pp. 1866–1881, Jun. 2019.

[2] L. Zhang, Q. Zhang, B. Du, X. Huang, Y. Y. Tang, and D. Tao, "Simultaneous spectral–spatial feature selection and extraction for hyperspectral images," *IEEE Trans. Cybern.*, vol. 48, no. 1, pp. 16–28, Jan. 2018.

[3] L. Zhang, L. Zhang, D. Tao, X. Huang, and B. Du, "Hyperspectral remote sensing image subpixel target detection based on supervised metric learning," *IEEE Trans. Geosci. Remote Sens.*, vol. 52, no. 8, pp. 4955–4965, Aug. 2014.

[4] A. Zare, J. Bolton, J. Chanussot, and P. Gader, "Foreword to the special issue on hyperspectral image and signal processing," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 7, no. 6, pp. 1841–1843, Jun. 2014.

[5] C. Ye *et al.*, "Landslide detection of hyperspectral remote sensing data based on deep learning with constrains," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 12, no. 12, pp. 5047–5060, Dec. 2019.

[6] T. Xie, S. Li, L. Fang, and L. Liu, "Tensor completion via nonlocal low-rank regularization," *IEEE Trans. Cybern.*, vol. 49, no. 6, pp. 2344–2354, Jun. 2019.

[7] M. Chi, A. Plaza, J. A. Benediktsson, Z. Sun, J. Shen, and Y. Zhu, "Big data for remote sensing: Challenges and opportunities," *Proc. IEEE*, vol. 104, no. 11, pp. 2207–2219, Nov. 2016.

[8] Z. Wu, Y. Li, A. Plaza, J. Li, F. Xiao, and Z. Wei, "Parallel and distributed dimensionality reduction of hyperspectral data on cloud computing architectures," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 9, no. 6, pp. 2270–2278, Jun. 2016.

[9] V. A. A. Quirita *et al.*, "A new cloud computing architecture for the classification of remote sensing data," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 10, no. 2, pp. 409–416, Feb. 2017.

[10] 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.

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

[12] M. Zhang, W. Li, Q. Du, L. Gao, and B. Zhang, "Feature extraction for classification of hyperspectral and LiDAR data using patch-to-patch CNN," *IEEE Trans. Cybern.*, vol. 50, no. 1, pp. 100–111, Jan. 2020.

[13] A. Plaza *et al.*, "Recent advances in techniques for hyperspectral image processing," *Remote Sens. Environ.*, vol. 113, no. 1, pp. S110–S122, 2009.

[14] Z. Wu, J. Liu, A. Plaza, J. Li, and Z. Wei, "GPU implementation of composite kernels for hyperspectral image classification," *IEEE Geosci. Remote Sens. Lett.*, vol. 12, no. 9, pp. 1973–1977, Sep. 2015.

[15] 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.

[16] G. Camps-Valls, N. Shervashidze, and K. M. Borgwardt, "Spatio-spectral remote sensing image classification with graph kernels," *IEEE Geosci. Remote Sens. Lett.*, vol. 7, no. 4, pp. 741–745, Oct. 2010.

[17] Y. Gu, C. Wang, D. You, Y. Zhang, S. Wang, and Y. Zhang, "Representative multiple kernel learning for classification in hyperspectral imagery," *IEEE Trans. Geosci. Remote Sens.*, vol. 50, no. 7, pp. 2852–2865, Jul. 2012.

[18] Y. Yuan, J. Lin, and Q. Wang, "Hyperspectral image classification via multitask joint sparse representation and stepwise MRF optimization," *IEEE Trans. Cybern.*, vol. 46, no. 12, pp. 2966–2977, Dec. 2016.

[19] J. Li, J. M. Bioucas-Dias, and A. Plaza, "Spectral–spatial hyperspectral image segmentation using subspace multinomial logistic regression and Markov random fields," *IEEE Trans. Geosci. Remote Sens.*, vol. 50, no. 3, pp. 809–823, Mar. 2012.

[20] 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.

[21] 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.

[22] Y. Zhou and Y. Wei, "Learning hierarchical spectral–spatial features for hyperspectral image classification," *IEEE Trans. Cybern.*, vol. 46, no. 7, pp. 1667–1678, Jul. 2016.

[23] Z. Wu, Q. Wang, A. Plaza, J. Li, L. Sun, and Z. Wei, "Parallel spatial–spectral hyperspectral image classification with sparse representation and Markov random fields on GPUs," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 8, no. 6, pp. 2926–2938, Jun. 2015.

[24] F. Magoules, J. Pan, and F. Teng, *Cloud Computing: Data-Intensive Computing and Scheduling*. London, U.K.: Chapman Hall, 2012.

[25] H. El-Rewini, T. G. Lewis, and H. H. Ali, *Task Scheduling in Parallel and Distributed Systems*. New York, NY, USA: Prentice-Hall, 2007.

[26] W. Liu, Z. Gu, J. Xu, X. Wu, and Y. Ye, "Satisfiability modulo graph theory for task mapping and scheduling on multiprocessor systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 8, pp. 1382–1389, Aug. 2011.

[27] S. Kirkpatrick, "Optimization by simulated annealing: Quantitative studies," *J. Stat. Phys.*, vol. 34, nos. 5–6, pp. 975–986, 1984.

[28] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proc. Int. Conf. Neural Netw.*, 2002, pp. 1942–1948.

[29] G. Zhang, "Quantum-inspired evolutionary algorithms: A survey and empirical study," *J. Heurist.*, vol. 17, no. 3, pp. 303–351, 2011.

[30] V. Bharadwaj, D. Ghose, and T. G. Robertazzi, "Divisible load theory: A new paradigm for load scheduling in distributed systems," *Clust. Comput.*, vol. 6, no. 1, pp. 7–17, 2003.

[31] J. Jia, B. Veeravalli, and J. Weissman, "Scheduling multisource divisible loads on arbitrary networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 21, no. 4, pp. 520–531, Apr. 2010.

[32] J. Haut, M. Paoletti, A. Paz-Gallardo, J. Plaza, A. Plaza, and J. Vigo-Aguiar, "Cloud implementation of logistic regression for hyperspectral image classification," in *Proc. Int. Conf. Comput. Math. Methods Sci. Eng.*, 2017, pp. 1063–2321.

[33] R. Zaatour, S. Bouzidi, and E. Zagrouba, "Parallel and distributed local fisher discriminant analysis to reduce hyperspectral images on cloud computing architectures," in *Proc. Int. Conf. Adv. Concepts Intell. Vis. Syst.*, 2018, pp. 245–257.

[34] A. A. Green, M. Berman, P. Switzer, and M. D. Craig, "A transformation for ordering multispectral data in terms of image quality with implications for noise removal," *IEEE Trans. Geosci. Remote Sens.*, vol. 26, no. 1, pp. 65–74, Jan. 1988.

[35] L. N. Xun, Y. H. Fang, and L. I. Xin, "Target detection algorithm in hyperspectral image based on CEM," *Opto-Electron. Eng.*, vol. 34, no. 7, pp. 18–21, 2007.

[36] M.-Y. Liu, O. Tuzel, S. Ramalingam, and R. Chellappa, "Entropy rate superpixel segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Providence, RI, USA, 2011, pp. 2097–2104.

[37] 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.

[38] M. Zaharia *et al.*, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *Proc. USENIX Conf. Netw. Syst. Design Implement.*, 2012, pp. 1–14.

[39] J. Sun, L. Yin, M. Zou, Y. Zhang, T. Zhang, and J. Zhou, "Makespan-minimization workflow scheduling for complex networks with social groups in edge computing," *J. Syst. Archit.*, vol. 108, Sep. 2020, Art. no. 101799.

[40] K.-H. Han and J.-H. Kim, "Quantum-inspired evolutionary algorithm for a class of combinatorial optimization," *IEEE Trans. Evol. Comput.*, vol. 6, no. 6, pp. 580–593, Dec. 2002.

[41] C. Y. Chung, H. Yu, and K. P. Wong, "An advanced quantum-inspired evolutionary algorithm for unit commitment," *IEEE Trans. Power Syst.*, vol. 26, no. 2, pp. 847–854, May 2011.

[42] J. G. Vlachogiannis and K. Y. Lee, "Quantum-inspired evolutionary algorithm for real and reactive power dispatch," *IEEE Trans. Power Syst.*, vol. 23, no. 4, pp. 1627–1636, Nov. 2008.

[43] Z. Wu, Q. Wang, A. Plaza, J. Li, J. Liu, and Z. Wei, "Parallel implementation of sparse representation classifiers for hyperspectral imagery on GPUs," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 8, no. 6, pp. 2912–2925, Jun. 2015.

[44] J. M. Bioucas-Dias and M. A. T. Figueiredo, "Alternating direction algorithms for constrained sparse regression: Application to hyperspectral unmixing," in *Proc. 2nd Workshop Hyperspectral Image Signal Process. Evol. Remote Sens.*, Reykjavik, Iceland, 2010, pp. 1–4.

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 August 2016 to September 2016. He was also a visiting scholar with the GIPSA-Lab, Grenoble INP, 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. His research interests include hyperspectral image processing, parallel computing, and remotely sensed big data processing.

Jin Sun (Member, IEEE) received the B.S. and M.S. degrees in computer science from the Nanjing University of Science and Technology, Nanjing, China, in 2004 and 2006, respectively, and the Ph.D. degree in electrical and computer engineering from the University of Arizona, Tucson, AZ, USA, in 2011.

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

Yi Zhang received the B.S. and Ph.D. degrees in computer science and engineering from Southeast University, Nanjing, China, in 2005 and 2011, respectively.

In 2009, he was an intern with the IBM China Research Laboratory, Beijing, China, after he was awarded the IBM Ph.D. Fellowship. In 2011, he joined the Huawei Tech. Company, Shenzhen, China, as a Member of the Technical Research Staff. He is currently an Associate Professor with the School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing. His research interests include task scheduling and resource management in cloud computing and mobile computing.

Yaoqin Zhu received the B.S. and Ph.D. degrees in computer science and 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. Her research interests include multimedia processing, virtual reality, and computer simulation.

Jun Li (Senior Member, IEEE) received the B.S. degree in geographic information systems from Hunan Normal University, Changsha, China, in 2004, the M.E. degree in remote sensing from Peking University, Beijing, China, in 2007, and the Ph.D. degree in electrical engineering from the Instituto de Telecomunicações, Instituto Superior Técnico, Universidade Técnica de Lisboa, Lisbon, Portugal, in 2011.

From 2011 to 2012, she was a Postdoctoral Researcher with the Hyperspectral Computing Laboratory, Department of Technology of Computers and Communications, Escuela Politécnica, University of Extremadura, Cáceres, Spain. She is currently a Professor with the School of Geography and Planning, Sun Yat-sen University, Guangzhou, China. She has authored or coauthored a total of 69 journal citation report articles, 48 conference international conference articles, and one book chapter. Her main research interests include remotely sensed hyperspectral image analysis, signal processing, supervised/semisupervised learning, and active learning.

Prof. Li has received a significant number of citations to her published works, with several articles distinguished as Highly Cited articles in Thomson Reuters' Web of Science-Essential Science Indicators.

Antonio Plaza (Fellow, IEEE) received the M.Sc. and Ph.D. degrees in computer engineering from the Department of Technology of Computers and Communications, University of Extremadura, Badajoz, Spain, in 1999 and 2002, respectively.

Hi is currently a Full Professor and the Head of the Hyperspectral Computing Laboratory, Department of Technology of Computers and Communications, University of Extremadura. He has authored more than 600 publications and guest edited 10 journal special issues. He has reviewed more than 500 manuscripts for over 50 different journals.

Prof. Plaza served as the Editor-in-Chief for the IEEE TRANSACTIONS ON GEOSCIENCE AND REMOTE SENSING from 2013 to 2017. He is included the Highly Cited Researchers List (Clarivate Analytics) from 2018 to 2019.

Jón Atli Benediktsson (Fellow, IEEE) received the Cand.Sci. degree in electrical engineering from the University of Iceland, Reykjavik, Iceland, in 1984, and the M.S.E.E. and Ph.D. degrees in electrical engineering from Purdue University, West Lafayette, IN, USA, in 1987 and 1990, respectively.

In 2015, he became the President and Rector of the University of Iceland. From 2009 to 2015, he was the Pro Rector of Science and Academic Affairs and a Professor of Electrical and Computer Engineering with the University of Iceland, where he had been a Faculty Member since 1991. His research interests are in remote sensing, biomedical analysis of signals, pattern recognition, image processing, and signal processing, and he has been published extensively in those fields.

Dr. Benediktsson has received many recognitions for his research. He has been a Web of Science Group Highly Cited Researcher since 2018. He was the President of the IEEE Geoscience and Remote Sensing Society (GRSS) from 2011 to 2012, and GRSS AdCom from 2000 to 2014. He was an Editor-in-Chief of the IEEE TRANSACTIONS ON GEOSCIENCE AND REMOTE SENSING from 2003 to 2008, and has served as an Associate Editor for IEEE TRANSACTIONS ON GEOSCIENCE AND REMOTE SENSING since 1999. He is a Fellow of SPIE.

Zhihui Wei (Member, IEEE) received the B.Sc. degree in applied mathematics, the M.Sc. degree in applied mathematics, and the Ph.D. degree in communication and information systems 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. His research interests include partial differential equations, image processing, multiscale analysis, sparse representation, and compressed sensing.