An Efficient and Scalable Framework for Processing Remotely Sensed Big Data in Cloud Computing Environments

Jin Sun, *Member, IEEE*, Yi Zhang, Zebin Wu, *Senior Member, IEEE*, Yaoqin Zhu, Xianliang Yin, Zhongzheng Ding, Zhihui Wei, Javier Plaza, *Senior Member, IEEE*, and Antonio Plaza, *Fellow, IEEE*

***Abstract*— The large amount of data produced by satellites and airborne remote sensing instruments has posed important challenges to efficient and scalable processing of remotely sensed data in the context of various applications. In this paper, we propose a new big data framework for processing massive amounts of remote sensing images on cloud computing platforms. In addition to taking advantage of the parallel processing abilities of cloud computing to cope with large-scale remote sensing data, this framework incorporates task scheduling strategy to further exploit the parallelism during the distributed processing stage. Using a computation- and data-intensive pan-sharpening method as a study case, the proposed approach starts by profiling a remote sensing application and characterizing it into a directed acyclic graph (DAG). With the obtained DAG representing the application, we further develop an optimization framework that incorporates the distributed computing mechanism and task scheduling strategy to minimize the total execution time. By determining an optimized solution of task partitioning and task assignments, high utilization of cloud computing resources and accordingly a significant speedup can be achieved for remote sensing data processing. Experimental results demonstrate that the proposed framework achieves promising results in terms of execution time as compared with the traditional (serial) processing approach. Our results also show that the proposed approach is scalable with regard to the increasing scale of remote sensing data.**

Manuscript received April 8, 2018; revised September 11, 2018 and December 3, 2018; accepted December 27, 2018. Date of publication January 25, 2019; date of current version June 24, 2019. This work was supported in part by the National Natural Science Foundation of China under Grant 61772274, Grant 61872185, Grant 61502234, Grant 71501096, Grant 61471199, Grant 61701238, Grant 11431015, and Grant 91538108, 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 30916011325 and Grant 30917015104, and in part by the China Post-Doctoral Science Foundation under Grant 2015M581801. *(Corresponding author: Zebin Wu.)*

J. Sun, Y. Zhang, Y. Zhu, X. Yin, Z. Ding, and Z. 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; yzhang@njust.edu.cn; zhuyaoqin@163.com; 115106000730@njust.edu.cn; 515106001780@njust.edu.cn; gswei@njust.edu.cn).

Z. Wu is with the School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing 210094, China, and also with Nanjing Robot Research Institute Co. Ltd., Nanjing 211135, China (e-mail: zebin.wu@gmail.com).

J. Plaza and A. Plaza are with the Hyperspectral Computing Laboratory, Department of Technology of Computers and Communications, University of Extremadura, 10071 Cáceres, Spain (e-mail: jplaza@unex.es; aplaza@unex.es).

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

Digital Object Identifier 10.1109/TGRS.2018.2890513

***Index Terms*— Big data processing, cloud computing, remote sensing, task scheduling.**

I. Introduction

THE amount of data available from remote sensing systems is increasing at an extremely fast pace due to recent advances in modern earth observation technologies [1]–[3]. The tremendous amount of remotely sensed data has posed serious challenges for efficient and scalable processing of remotely sensed data in the context of various applications. Considering the high volume and fast generation velocity of remotely sensed data, the processing of such data can be naturally regarded as a big data problem [4], [5].

Cloud computing offers the capability to tackle big data processing by means of its distributed computing mechanism. The use of cloud computing for the analysis of large-scale remote sensing data repositories represents a natural solution, as well as an evolution of previously developed techniques. With the continuously increasing demand for massive data processing in remote sensing applications, there have been several efforts in the literature oriented toward exploiting cloud computing infrastructure for processing remote sensing big data [6]–[10]. For example, Wu *et al.* [8] proposed a parallel and distributed implementation of the principal component analysis algorithm for hyperspectral dimensionality reduction. Based on the MapReduce parallel model [11], this implementation utilizes Hadoop's distributed file system (HDFS) to realize distributed storage and uses Apache Spark to perform parallel computing of hyperspectral data. Quirita *et al.* [9] proposed a new distributed architecture for supervised classification of large volumes of earth observation data. This architecture supports distributed execution, network communication, and fault tolerance to the user. However, the existing cloud computing solutions are mainly developed toward solving big data problems in a specific category of remote sensing applications.

In addition to the parallel and distributed computing mechanism, a majority of existing cloud computing platforms rely on task scheduling strategies to further exploit the parallelism in workflow processing and to promote the utilization of cloud computing resources [12]–[14]. Similar to other general-purpose applications, a remote sensing application

can be divided into a set of subtasks that allow for the distribution of the application's data processing workloads across multiple computing resources [15], [16]. In general, there are certain dependencies among the tasks belonging to a specific application, which impose an order of precedence on their execution [16]–[18]. For instance, in remote sensing image classification applications, the training of the classification model is required to be completed prior to the classification of a testing data set [9], [19]. Similarly, in many remote sensing image superresolution and pan-sharpening applications, the procedure of image reconstruction cannot start execution until the necessary preprocessing procedure has been finished [20], [21]. When processing a remote sensing application in cloud computing environments, a crucial step referred to as *scheduling* [22], [23] is to determine an appropriate assignment of tasks onto computing resources and the order of their execution. The quality of scheduling solution fundamentally determines the application's parallelism and eventually the efficiency of big data processing on clouds.

In this paper, we propose a novel big data processing framework for massive remote sensing image processing that makes use of cloud computing technologies. This framework not only takes advantage of the parallel processing abilities of cloud computing to cope with large-scale remote sensing data but also incorporates the task scheduling strategy to further exploit the parallelism during the distributed computing stage. The proposed framework starts by profiling the remote sensing data processing application and characterizing the application as a directed acyclic graph (DAG). With the obtained DAG representing the application, we develop an optimization framework that incorporates the distributed computing mechanism and task scheduling strategy to minimize the total execution time. A distinguishing property of the proposed framework is that we take into account task divisibility during the scheduling procedure. The decision variables of the optimization problem include not only task assignments but also the number of data partitions for each task that can be partitioned and processed in parallel on Spark (i.e., the required number of computing resources assigned for executing it). The formulated scheduling model is, therefore, complicated and in the requirement of an efficient scheduling algorithm. For this reason, we propose a metaheuristic scheduling strategy based on a quantum-inspired evolutionary algorithm (QEA) to solve the scheduling problem. By solving this optimization problem, high utilization of cloud computing resources and, therefore, a significant speedup can be achieved for processing of remote sensing big data. It is worth mentioning several existing distributed processing approaches [24], [25] that also make use of the parallelism information provided by DAGs. These approaches use an effective strategy to exploit task and data parallelism, in which a task can start executing once its predecessor tasks are completed. Built upon a more complicated optimization model, our framework can be expected to explore a more efficient solution of distributed processing. We use a pan-sharpening method based on deep neural networks (DNNs) as a study case to demonstrate the efficiency, scalability, as well as the superiority of our proposed big data processing framework.

Specifically, the technical contributions of this paper can be summarized as follows.

1) This paper provides a scheduling-enabled cloud computing solution to the efficient processing of remote sensing applications in which the workflows can be well represented by DAGs.
2) It develops an optimization framework that searches for an optimal distributed processing solution, including the numbers of partitions for partitionable tasks and the assignments of workers executing all tasks, under the constraint of limited computing resources.
3) It further develops a cost-effective, QEA-based metaheuristic scheduling algorithm to solve the above-mentioned optimization problem and obtain the optimized distributed processing solution.
4) Experimental results demonstrate that the proposed framework achieves promising results in terms of execution time and speedups as compared with the serial processing approach and is scalable to the increasing scale of remotely sensed data sets.

The remainder of this paper is organized as follows. Section II discusses the processing flow of a typical pan-sharpening method, which is used as the testcase in this paper. Section III introduces the distributed mechanism and task scheduling strategy in cloud computing and then details the proposed optimization framework. Experimental results on a private cloud environment are presented in Section IV. Section V discusses potential strategies to extend the proposed framework to cope with public and/or commercial clouds. Finally, Section VI concludes this paper with some remarks.

II. Serial Flow of a Pan-Sharpening Method Using Deep Neural Networks

Multispectral pansharpening is a postprocessing method that is widely used in remote sensing data processing to produce a high spatial resolution image by fusing the information of a set of panchromatic images [26]. This section briefly describes the fundamental process of a pan-sharpening method using DNNs [20]. We show that the processing flow of this method involves not only expensive computational cost but also strong dependencies among the subtasks.

DNN has a significant representational power for complex image structures and has demonstrated superior performance in the field of image processing [27]–[29]. The DNN-based method in [20] aims to reconstruct high-resolution (HR) multispectral images with high spatial resolution and small spectral distortions. It mainly consists of three operations: patch extraction for generating a training set, DNN training, and image reconstruction. Fig. 1 presents the complete flowchart of the fusion process based on DNN training.

The inputs include an HR panchromatic image and a low-resolution (LR) multispectral image. We performed interpolation amplification on the LR multispectral image and further compute the weighted average of all bands of the amplified LR multispectral image to generate an LR panchromatic image. At the training stage, we use a small convolution matrix to randomly sample a sufficiently large number of HR image

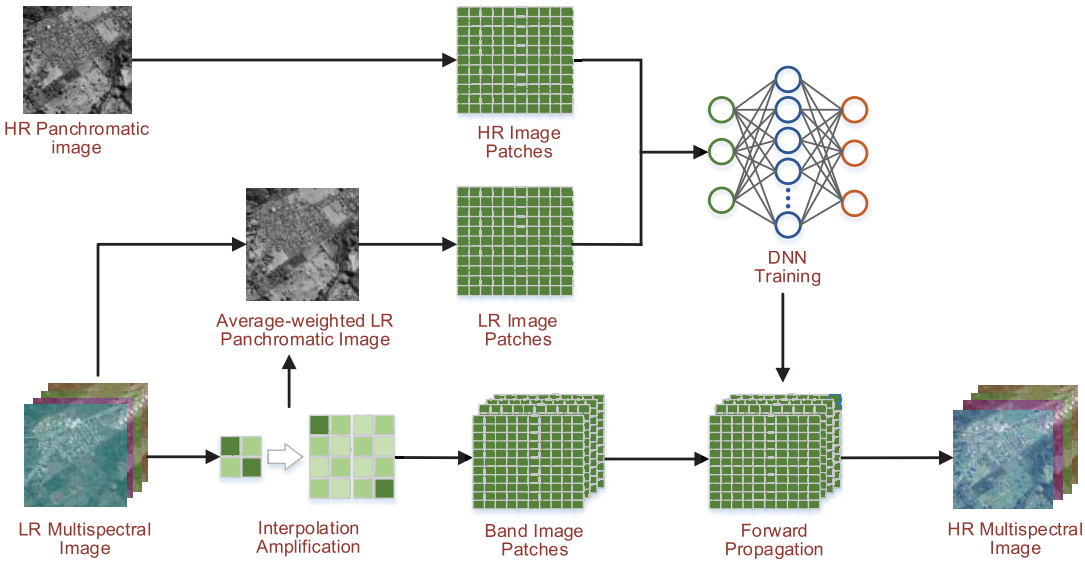


Fig. 1. Flowchart of the DNN-based pan-sharpening method in [20].

patches and LR image patches from the HR panchromatic image and the LR panchromatic image generated from the LR multispectral image, respectively. We select a subset of the randomly generated image patches for DNN training. The selected HR image patches and LR image patches together form the training set, based upon which the DNN model can be trained.

At the reconstruction stage, each band of the LR multispectral image is divided into LR multispectral image patches $\{y_k^j\}$ with overlapping, where k represents the kth band, and j stands for the jth image patch. The HR multispectral image patches $\{x_k^j\}$ are obtained by feedforwarding the test LR multispectral image patch y_k^j using the trained DNN. Finally, the sharpened multispectral image is reconstructed by the overlapping image patches in all individual bands.

The feedforward procedure for obtaining the HR multispectral image is as follows. Let $\{x_p^i\}_{i=1}^N$ and $\{y_p^i\}_{i=1}^N$ denote the HR and LR image patches in the training set, where N denotes the number of image patches used for training. The feedforward functions consist of an encoder and a decoder

$$h\left(y_p^i\right) = s\left(Wy_p^i + b\right) \tag{1}$$

$$x\left(y_p^i\right) = s\left(W'h\left(y_p^i\right) + b'\right) \tag{2}$$

where $s(x) = (1 + \exp(-x))^{-1}$ is the sigmoid activation function, W and b are the encoding weights and biases, W' and b' are the decoding weights and biases, $h(y_p^i)$ is the hidden layer's activation, and $x(y_p^i)$ is the reconstruction of the input. Once the DNN has been trained, for each LR multispectral image patch y_k^j, the trained DNN can reconstruct the corresponding HR multispectral image patch x_k^j. Specifically, if we use y_k^j as the input to the trained DNN, the sharpened HR multispectral image patch x_k^j can be obtained according to the feedforward functions in (1) and (2).

By observing the fusion process in Fig. 1, we note that the training and tuning of DNN would inevitably lead to a high computational complexity [30], [31]. The serial processing of large-scale remote sensing data would be computationally intractable on a single machine, due to insufficient memory. In fact, when processing a 24.3-MB HR panchromatic image and a 6.1-MB LR multispectral image by using the DNN-based pan-sharpening method, the fusion process would generate about 1.8 GB of training data and up to 5.7 GB of image patches in the reconstruction stage. Therefore, it is highly required to process such remote sensing big data through parallel and distributed computing. More importantly, the subtasks of the fusion process exhibit strong dependencies among each other. These dependencies would exert constraints on the execution order of the tasks when assigning the tasks onto computing resources. Therefore, it is also important to determine an optimal assignment of these subtasks on cloud computing resources to achieve better efficiency. These two observations reveal the importance of our proposed framework that integrates distributed computing and task scheduling.

III. Proposed Optimization Framework

Motivated by the conclusions drawn in Section II, this section introduces a new distributed mechanism and task scheduling concept for cloud computing architectures and further details our proposed optimization framework for accelerating remote sensing data processing. The main strategies used in the optimization framework are described as follows.

1) By employing the MapReduce mechanism, when executing certain subtasks of a specific application, the original data set can be divided into a set of partitions to facilitate the parallel and distributed processing. Under the constraint of a limited number of computing resources, an appropriate MapReduce solution would be beneficial for reducing the duration time of computation-intensive operations and therefore the total execution time.
2) As discussed previously, there are, in general, precedence relationships among operations that restrict the

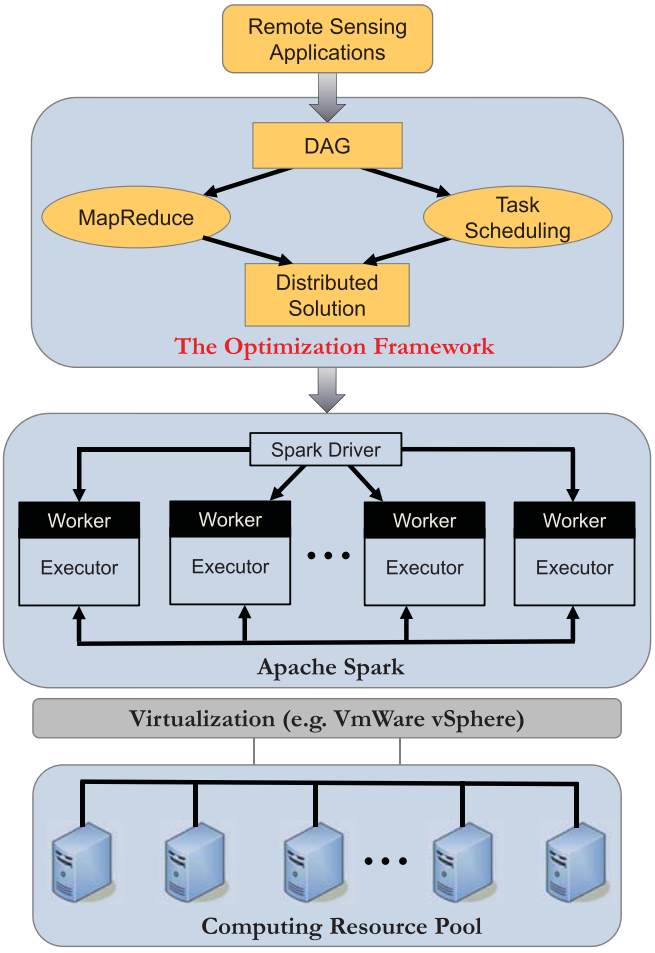


Fig. 2. Overview of the proposed optimization framework.

order of their execution. Task scheduling aims at determining the optimal assignment of application tasks on computing resources. A high-quality scheduling solution is beneficial for maximizing the utilization of computing resources and minimizing the application's makespan.

With the above-mentioned optimization strategies in mind, Fig. 2 illustrates the overview of the proposed optimization framework built upon Apache Spark [32]. For a specific remote sensing application that can be profiled as a DAG, this framework provides an optimized solution for processing remote sensing big data in cloud computing environments. With the profiled DAG and a limited number of computing resources, this framework is able to obtain the optimal distributed processing solution considering both MapReduce and task scheduling strategies, by solving a formulated optimization problem. Guided by the obtained solution, the Spark platform assigns the tasks to cloud computing resources for distributed processing in an optimal manner.

In this paper, we use Apache Spark instead of Hadoop to implement distributed computing, in order to take full advantage of the high-performance in-memory computing engine provided by Apache Spark. A variety of remote sensing algorithms involve data-intensive and/or computation-intensive operations, which are in need of efficient in-memory computations. Different from Apache Hadoop, which only supports simple one-pass computations (e.g., aggregation or database queries), Apache Spark is generally more appropriate for multipass algorithms. By implementing a fault-tolerant abstraction for in-memory cluster computing, Spark provides fast and general data processing on large clusters. It not only supports simple one-pass computations but also can be extended to the case of multipass algorithms that are commonly required for more complex data analytics. Spark extends the MapReduce model to include primitives for data sharing, named resilient distributed data sets (RDDs) and offers an application programming interface based on coarse-grained transformations that allow them to recover data efficiently using lineage. In addition, Apache Spark has an advanced execution engine that supports cyclic data flow and in-memory computing and, therefore, can run programs up to 100× faster than Hadoop MapReduce in memory or 10× faster on disk.

A. Accelerating Data Processing by Distributed Computing

Apache Spark is an advanced computing platform for large-scale data processing, which implements a fault-tolerant abstraction for in-memory cluster computing, and provides fast and general data processing on large clusters. Spark extends the MapReduce model to include primitives for data sharing, named RDDs. To implement distributed processing on Spark, the user needs to write a driver program that defines one or more RDDs, invokes actions on them, and tracks the lineage of RDDs. The driver program is usually connected to a cluster of workers, which are long-lived processes that can store RDD partitions in random access memory across operations [33].

Referring to the fusion processing of the DNN-based pan-sharpening method in Fig. 1, for certain operations, e.g., DNN training and DNN parameter tuning, the original data set can be partitioned to facilitate distributed processing. On Spark platform, we store the original remote sensing data sets on HDFS in a distributed and flexible way. Since the original data sets are divided into many spatial-domain partitions, we read every partition of the data on HDFS as a key-value pair in which the key is the offset of this partition in the original data set, and the value is the corresponding data partition. When performing the pan-sharpening procedure, we use TensorFlowOnSpark [34], a Spark implementation of an open-source software library for machine learning across a range of tasks, to train the DNN. The TensorFlowOnSpark implementation classifies the computation jobs into a parameter server (PS) job and a number of worker jobs. The PS job is responsible for storing and updating the model's parameters. The worker jobs are responsible for optimizing the parameters and updating the parameters to the PS job.

Fig. 3 provides an illustrative explanation of how data partitions are mapped onto designated computing resources, as well as how the distributed computing outcomes are reduced to produce the required results. When performing the training procedure, we first load the data partitions of HR and LR panchromatic images from HDFS. A "map" operation is required to convert the loaded data partitions into RDD format.

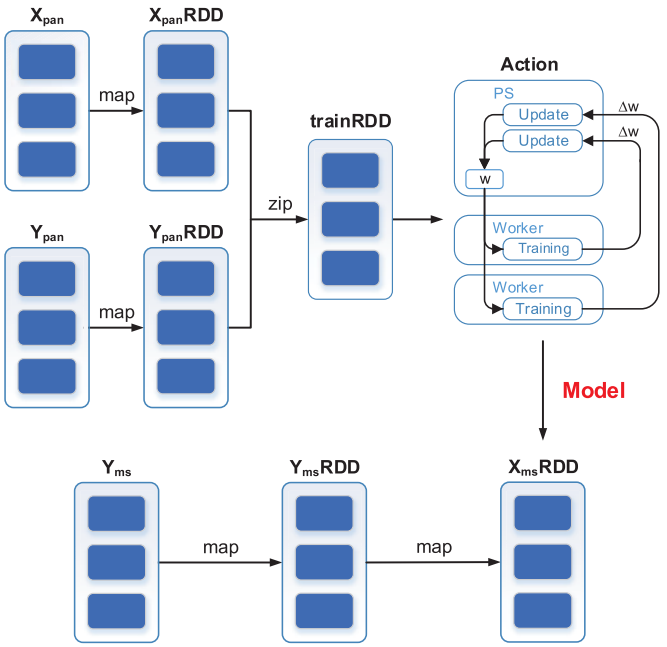


Fig. 3. Partitioning of remote sensing image for distributed processing of the DNN-based fusion method.

The data partitions after conversion, denoted by X_{pan}RDD and Y_{pan}RDD, can be then processed in parallel on Spark. We further use a "zip" operation to merge the obtained RDDs, in order to generate the training data set trainRDD. Guided by the tuned training parameters, all worker nodes take actions to establish the training model simultaneously in a parallel way. At the reconstruction stage, the RDD partitions of LR multispectral image Y_{ms}RDD are generated in the same manner by performing the "map" operation. Then, relying on the trained DNN, the HR multispectral image can be reconstructed by fusing the data sets stored in Y_{ms}RDD partitions. Each worker node is responsible for sending the updated weights to the PS node after each iteration of computation. The PS node is in charge of averaging the weights received from all worker nodes and broadcasting the averaged weight to all workers for the next iteration of weight calculation.

The distributed processing of data partitions has the capability to accelerate the processing of large-scale remote sensing data. However, it is worth emphasizing the tradeoff between computational efficiency and the time overhead introduced by distributed computing. The increasing number of workers may induce a substantial amount of time overhead, including the following: 1) the time cost spent in creating and initializing MapReduce jobs; 2) the time elapsed during the job synchronization process; and 3) the time cost for data transmission and information exchange (e.g., the weight parameters of the DNN model). Taking into account the above-mentioned time overhead introduced by the MapReduce mechanism inherently, the speedup achieved by distributed computing does not grow linearly with the increasing number of distributed computing resources. Fig. 4 shows the speedup of execution time achieved by performing the DNN training process on TensorFlowOnSpark, with regard to the number of workers for different sizes of training data. We assume a single PS node and observe

Fig. 4. Speedup achieved by distributed processing of DNN training process on Spark.

the speedup achieved with different numbers of worker nodes. For different sizes of training data, the speedup exhibits approximately a linear increase if the number of workers is below eight. However, if we keep on increasing the number of workers, the communication overhead becomes comparable to the execution time, leading to a less significant improvement of speedup. For example, the speedup is compromised to 9.39× for a data set of size 156 MB. As a result, in the proposed optimization framework, it would be of great importance to determine an optimal number of partitions that a particular task can be divided into, and accordingly the number of workers required for executing this task.

B. Exploiting Parallelism by Task Scheduling

This section introduces the concept of task scheduling in distributed processing of a remote sensing application. A scheduling model consists of an application, a target computing environment, and evaluation criteria for scheduling. In general, an application is fundamentally composed of sets of tasks, e.g., the fusion process of the DNN-based pansharpening method in Fig. 1. Tasks within a particular flow may have dependencies among them, while tasks from different flows tend to have very few or no dependencies among them. In this manner, an application is usually represented by a DAG, denoted by $G = (V, E)$, where $V = \{v_1, v_2, \ldots, v_n\}$ is a set of tasks to be executed, and $E = \{(i, j)\}$ is a set of edges between the tasks. Each edge $(i, j) \in E$ specifies a precedence constraint on two connected tasks: task v_i must be completed before v_j can start. In addition, each task is associated with a weight value D_i representing the duration time of task v_i. Fig. 5 provides a simple example of a task graph.

Let S_i and S_j denote the starting times for v_i and v_j, respectively. The precedence relationship exerts the following set of constraints:

$$S_i + D_i \leq S_j, \quad \forall (i, j) \in E \tag{3}$$

where D_i stands for the actual execution time of task v_i on a computing resource, in terms of the number of time slots. It is

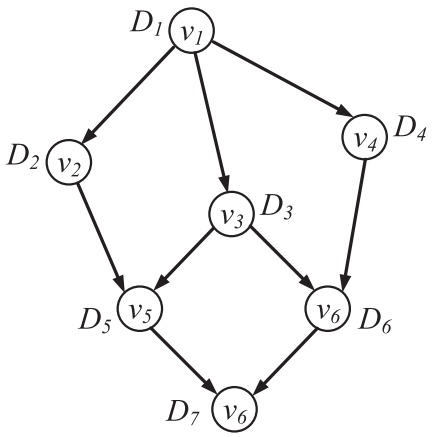


Fig. 5. Task graph representing a task flow.

worthwhile to mention that depending on the partitioning of original remote sensing data set, D_i is a varying value rather than a fixed value in the scheduling model.

The scheduling problem is fundamental to determine the mapping function that maps each task to a computing resource for minimizing the scheduling goal. A schedule of a DAG on available computing resources can be illustrated as a Gantt chart that consists of a list of all computing resources, and for each computing resource, a list of assigned tasks of which the execution order is determined by task starting and ending times [35]. In this paper, the scheduling goal is to minimize the total completion time of a remote sensing application. This performance measure is known as the makespan of application, which is determined by the maximum ending time of any task.

We now discuss how to profile the processing flow of a remote sensing application into a DAG representation. In a DAG, a task without any parent is called an entry task and a task without any child is called an exit task. In certain cases, the scheduling problem requires single-entry and single-exit DAGs. However, there might be more than one entry/exit tasks on the DAG. For instance, the processing flow in Fig. 1 has two entry tasks. To deal with this situation, the multiple entry tasks are connected to a zero-weight pseudo entry/exit task, which has no effect to the schedule [36]. In addition, when profiling a remote sensing application, if there exist loops within the processing flow, the entire loop should be considered as a single task on the DAG. Moreover, the task weights associated with the DAG have to be normalized.

Performance and efficiency are two characteristics used to evaluate a scheduling strategy. We should evaluate a scheduling system based on the quality of the produced task assignment and the efficiency of the scheduling algorithm. The produced schedule is judged based on the performance metric to be optimized, while the scheduling algorithm is evaluated based on its time complexity. As pointed out in the literature, the scheduling problem is known to be computationally intractable in many cases [37].

C. Problem Formulation

Based upon the optimization strategies for improving the efficiency of distributed computing, this section formulates the considered problem as an optimization problem and provides the formal optimization model. Assume that the capacity of the resource pool is R, i.e., the number of available computing resources. Given a remote sensing application represented by a DAG, we define a binary variable e_{pi} to indicate a specific precedence constraint. $e_{pi} = 1$ denotes that task v_p is a predecessor of task v_i, and $e_{pi} = 0$ otherwise. We accordingly define $P_i = \{p | e_{pi} = 1\}$ as the predecessor set of task v_i.

We first discuss the decision variables and constraints of the optimization model. Let x_{it} be an binary variable. $x_{it} = 1$ denotes that task v_i is in execution on time slot t, and $x_{it} = 0$ otherwise. Accordingly, the starting time of the ith task S_i is given by

$$\arg\min \{t | x_{it} = 1\}. \tag{4}$$

Obviously, in order to satisfy the precedence relationships between task v_i and its predecessors, we have the following set of constraints:

$$S_i \geq \max \{S_p + D_p\} \quad \forall p \in P_i \tag{5}$$

where S_i and S_p stand for the starting time of task i and starting time of task v_p, respectively, and D_p denotes the duration time of task v_p whose value is dependent on the number of computing resources assigned for executing it. Note that in the proposed scheduling model, for a task that can be partitioned and processed in parallel by the MapReduce mechanism, its duration time is a varying metric depending on the number of partitions the original task is divided into, or equivalently the required number of computing resources assigned for executing this task. Therefore, not only the task assignments but also the numbers of partitions for partitionable tasks would have a direct impact on the total execution time of a given application.

Combining (4) and (5) together, the precedence constraints can be expressed as follows:

$$\arg\min \{t | x_{it} = 1\} \geq \max \{\arg\min \{t | x_{pt} = 1\} + D_p\}. \tag{6}$$

Let c_ω be the makespan of the specified application, the considered problem can be formulated as the following integer program (IP) (7)–(11):

$$\begin{aligned} \min\ & c_\omega = \max\{S_i + D_i\} \\ & \quad = \max\{\arg\min \{t | x_{it} = 1\} + D_i\} && (7) \\ \text{s.t. } & \arg\min \{t | x_{it} = 1\} \\ & \geq \max \{\arg\min \{t | x_{pt} = 1\} + D_p\}, \quad \forall p \in P_i && (8) \\ & x_{it} \in \{0, 1\}, \quad r_i \in \{1, 2, \ldots, m_i\} && (9) \\ & \sum_{i=1}^{n} x_{it} r_i \leq R && (10) \\ \text{variables } & x_{it}, r_i, \quad i = 1, 2, \ldots, n. && (11) \end{aligned}$$

In this model, (7) describes the optimization goal, which is intended to minimize the total execution time of remote sensing data processing. Equation (11) lists the decision variables in the optimization model, including a set of variables x_{it} indicating the mapping relationships of tasks onto computing resources, as well as a set of variables r_i indicating the number of data partitions for a specified task. The optimization model also involves several constraints to formulate the

scheduling framework. Equation (8) implies the execution order of dependent tasks for satisfying precedence relationships. Equation (9) forces that the mapping variables x_{it} must be binary, and r_i must be integer variables. Equation (10) indicates that the total number of computing resources occupied for processing the application cannot exceed the capacity of the resource pool.

We now revisit the dependence of task duration upon the number of partitions for a specific task that can be partitioned and processed in parallel on Spark. On the one hand, as indicated in Fig. 4, the speedup in task execution time achieved by distributed computing would be compromised as the number of partitions increases for a data set of a particular size. On the other hand, we observe from the experimental results that with a fixed number of workers, task execution time is directly proportional to the size of data set. For this reason, we use the following fitting method to model task duration as a function of data size as well as the number of workers:

$$D_i = D(S^{\#}, r_i) \quad \forall v_i \in V \text{ and } r_i \leq m_i \tag{12}$$

where $S^{\#}$ denotes the size of data set, r_i denotes the number of data partitions for task v_i (which is exactly the number of workers that it requires), and m_i denotes the upper bound limit of r_i, i.e., the maximum number of partitions that task v_i can be divided into. Obviously, $m_i = 1$ implies that task v_i cannot be processed in parallel. Based upon this fitting model, task duration time for a particular data set can be easily obtained according to the number of assigned workers. In this manner, we only need to measure task durations for a few number of configurations and data sets. At the scheduling stage, we rely on the fitting model to predict the durations of partitionable tasks and to further evaluate the total execution time of the target application.

For the pan-sharpening application studied in this paper, task duration is modeled to have a linear dependence upon data size (as will be verified in experimental results). Also, according to the analysis in Section III-A, task duration is inversely proportional to the number of workers. For applications in which the tasks are not well-behaved in terms of execution times, a more complicated fitting function is required to accurately model the dependence of task duration upon data size and worker configuration. For tasks that have certain stochastic components depending on the distribution of input data, we need to either develop a stochastic model to capture the variations of task processing times or employ an uncertainty set to reduce the resultant stochastic optimization problem to a deterministic one. However, these potential solutions are in requirement of advanced uncertainty analysis techniques and, therefore, are beyond the scope of this paper. Moreover, the current framework can be easily adapted to cope with applications in which the processing times can be influenced by certain parameters, by rebuilding the fitting model based on premeasured task durations.

We further discuss the practicability of this task duration fitting model. On the one hand, for a particular application, building a fitting model upon a small set of premeasured duration data is a one-time operation. For different data set sizes and worker configurations, we can use the established fitting model to predict task durations and eventually determine the optimal scheduling solution. On the other hand, for different applications, it may be necessary to rebuild the fitting model based on measured task durations. Once the fitting model is established, the duration of a partitionable task of any data size with any number of workers can be predicted, and the optimal solution that minimizes the total execution time can be determined by the scheduling procedure.

The optimization problem formulated in (7)–(11) is fundamentally an IP, which is known to be NP-hard [38]. The varying values of task durations in this distributed processing framework, which are closely associated with the decision variables r_is, differentiate the scheduling strategy from traditional schedulers. Assigning appropriate values of r_is according to the computation loads of partitionable tasks is beneficial for improving the efficiency of parallel processing. Scheduling strategies that are popularly used in existing cloud computing platforms [23], [39], e.g., round-robin and first come first served, cannot solve this scheduling problem properly. Therefore, this paper proposes using metaheuristic algorithms to solve the optimization problem in (7)–(11). Metaheuristics do not guarantee optimal solutions to the problem but attempt to find near-optimal solutions alternatively. In Section III-D, we develop a scheduling algorithm based on QEA to solve the formulated IP.

D. QEA-Based Scheduling Algorithm

This section details an efficient QEA-based algorithm for solving the optimization problem described in (7)–(11). Similar to other existing evolutionary algorithms, e.g., genetic algorithms [40], [41] and evolutionary programming [42], [43], QEA is also characterized by the representation of individual (i.e., a solution candidate in search space), the evaluation function, and the population dynamics. The main advantage of QEA is that the Q-bit representation in QEA has a better characteristic of population diversity than the representations in other evolutionary algorithms, eventually resulting in a high-quality optimization solution.

1) Preliminaries of Quantum Computing: Before describing the proposed QEA-based algorithm, we briefly introduced the preliminaries of quantum computing. The smallest information unit used in QEA is called a Q-bit, which may be in the "1" state or in the "0" state, or in the linear superposition of the two states. The state of a Q-bit can be represented as

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \tag{13}$$

where α and β are two complex numbers. $|\alpha|^2$ and $|\beta|^2$ denote the probabilities of a Q-bit being the state of "0" and that of "1," and they satisfy $|\alpha|^2 + |\beta|^2 = 1$. More explanations are provided in [44].

A Q-bit individual consisting of m Q-bits can be expressed as follows:

$$\begin{pmatrix} \alpha_1 & \alpha_2 & \cdots & \alpha_m \\ \beta_1 & \beta_2 & & \beta_m \end{pmatrix} \tag{14}$$

where $|\alpha|^2 + |\beta|^2 = 1$ holds for $(i = 1, 2, \ldots, m)$. A Q-bit individual can be converted to a binary string by collapsing

each state into either the "0" state or the "1" with the probabilities of $|\alpha|^2$ and $|\beta|^2$, respectively. This collapsing procedure is, in fact, used to convert a quantum state into a single deterministic state [45], [46].

The state of a Q-bit can be changed by a quantum gate [44]. A quantum gate is a reversible gate and can be denoted as a unitary operator U acting on the Q-bit states. The operator U satisfies $U^+U = UU^+ = I$, where U^+ is the Hermitian adjoint of U. There are several kinds of quantum gates, among which rotation gate is the one most commonly adopted in QEAs, which can be represented as

$$U(\Delta\theta) = \begin{bmatrix} \cos(\Delta\theta) & -\sin(\Delta\theta) \\ \sin(\Delta\theta) & \cos(\Delta\theta) \end{bmatrix} \tag{15}$$

where $\Delta\theta$ is the rotation angle. In terms of the rotation gate, the parameters of a Q-bit can be updated by the following rule:

$$\begin{bmatrix} \alpha' \\ \beta' \end{bmatrix} = U(\Delta\theta) \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} \cos(\Delta\theta) & -\sin(\Delta\theta) \\ \sin(\Delta\theta) & \cos(\Delta\theta) \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix}. \tag{16}$$

The value of $\Delta\theta$ can be determined by a predetermined lookup table [44], [47] or can be calculated in terms of the normalized difference between the objective values of Q-bit individuals [48].

2) Solution Representation, Initialization, and Rotation Gate: In this paper, the scheduling solution can be represented by a binary string of length $m \times k$, where k is an integer with the restriction $2^{(k-1)} \leq R \leq 2^k$ (R is the total number of available computing resources). Note that by decoding each k consecutive bits to a decimal number, this binary string can be mapped to a decimal string of length m, representing the mapping relationships between tasks and computing resources. Let the ith k consecutive bits be mapped to a decimal number q_i. This means that the q_ith resource is allocated to the task i. It is worth emphasizing that q_i may be greater than R, in this scenario, the following mod operation is required:

$$q_i \leftarrow (q_i \mod R). \tag{17}$$

In this QEA-based algorithm, we use the initialization method and the rotation gate in [44]. All Q-bits in a Q-bit individual are initialized as $(1/\sqrt{2},\ 1/\sqrt{2})$ under the assumption that all Q-bits have identical probabilities to collapse to "0" or "1." In this paper, we use a predetermined lookup table introduced in [44] to decide the value of rotation angle $\Delta\theta$, based on which a new Q-bit individual can be generated.

3) Solution Evaluation: Given a solution (i.e., a binary string), after obtaining its corresponding decimal string, we need to calculate its objective to evaluate the solution's quality. In this paper, we propose an evaluation method to achieve this target. The evaluation starts with a task sequence (ts) which is generated by arranging all tasks in a nondescending order by their earliest starting times (EST). In our considered problem, a task may have multiple durations corresponding to different the number of its allocated resources. To solve this problem, we consider the shortest one among each task's all durations as its duration when we calculate tasks' ESTs. Obviously, all predecessors of a task should be positioned in front of the task in "ts." The evaluation schedules each task i in ts one by one to determine their starting times S_i, ending times et_i and the number of resources allocated to them (r_i). To be specific, S_i should not be earlier than the maximal ending time of all its predecessors, i.e., to satisfy precedence constraint (5). In order to minimize the makespan, task durations for all possible number of workers should be checked and the one with the minimal ending time will be accepted. The procedure of solution evaluation is described in Algorithm 1.

Algorithm 1 Solution Evaluation

Input: A solution ζ, and a task sequence ts
Output: The objective of the input solution $c_\omega(\zeta)$

1: Decode the input solution ζ to generate its corresponding decimal string ζ_d;
2: **for** $l = 1$ to n **do**
3: Get the l-th task from ts and get this obtained task's index i;
4: Get the i-th decimal number in ζ_d and set it as q_i;
5: Calculate the maximal ending time of all predecessors of the task i and set it as S_i;
6: **if** $(m_i = 1)$ **then** ▷ for tasks that cannot be partitioned
7: From the time slot s_i, find the first available time of the q_i-th resource (denoted as t) while the q_i-th resource is available between the time slots from t to $t + D_i - 1$;
8: Set $et_i \leftarrow t + D_i - 1$;
9: Set the q_i-th resource to be unavailable at time slots from t to $t + D_i - 1$;
10: **else** ▷ for partitionable tasks
11: Set $et_i \leftarrow MAX$; ▷ a sufficiently large number
12: **for** $j = 1$ to m_i **do**
13: Set $r_i \leftarrow j$;
14: Use the task duration fitting model to predict the duration time D_i;
15: From the time slot s_i, find the first available time of r_i resources (denoted as t_j) while these r_i resources are all available between the time slots from t_j to $t_j + D_i - 1$;
16: **if** $(t_j + D_i - 1 < et_i)$ **then**
17: Set $et_i \leftarrow t_j + D_i - 1$;
18: Add the indices of these allocated resource to a set Ψ;
19: **end if**
20: **end for**
21: Set each resource with the index in Ψ to be unavailable at time slots from t_j to $t_j + D_i - 1$;
22: **end if**
23: **end for**
24: Calculate $c_\omega(\zeta)$ according to (7);
25: Return $c_\omega(\zeta)$;

In what follows, we briefly discuss the computational complexity of Algorithm 1. The complexity of the evaluation procedure cannot be determined explicitly, because it is difficult to decide the complexities of Lines 7 and 14. Instead, we evaluate the lower bound complexity. The lower bound complexities of Lines 7 and 14 are $\mathcal{O}(D_i)$ and $\mathcal{O}(m_i \times D_i)$, respectively. When the termination condition in Line 6 is met, both the lower

Algorithm 2 QEA-Based Scheduling Algorithm

1: Use EST to generate a task sequence ts;
2: Initialize all Q-bit individuals in Φ;
3: **for** each $\phi \in \Phi$ **do**
4: Convert ϕ to its corresponding solution ζ_ϕ;
5: Evaluate ζ_ϕ by the evaluation procedure shown in Algorithm 1;
6: **end for**
7: Set the best obtained solution as ζ_{best};
8: **while** (termination criterion is not met) **do**
9: **for** each $\phi \in \Phi$ **do**
10: Update ϕ by means of the rotation gate;
11: Convert ϕ to its corresponding solution ζ_ϕ;
12: Evaluate ζ_ϕ through the evaluation procedure given in Algorithm 1;
13: **if** (ζ_ϕ is better than ζ_{best}) **then**
14: $\zeta_{best} \leftarrow \zeta_\phi$;
15: **end if**
16: **end for**
17: **return** ζ_{best};
18: **end while**

bound complexities of Lines 7 and 14 are $\mathcal{O}(r_i \times D_i)$. As a result, the lower bound complexity of the evaluation procedure is determined as $\mathcal{O}(n \times m \times r \times d)$, in which $m = \min\{m_i\}$, $r = \min\{r_i\}$ and $d = \min\{D_i\}$, respectively.

4) Optimization Algorithm Description: In order to solve the optimization problem in (7)–(11), we propose a new QEA-based algorithm based on the classical QEA in [44]. This new QEA-based algorithm consists of a basic QEA structure, the solution representation in Section III-D2 as well as solution evaluation method in Algorithm 1 to evaluate all the individuals. The description of the QEA-based scheduling procedure is presented in Algorithm 2. To start with, we first generate a ts according to the ESTs of all tasks, which will be later used in solution evaluation procedures. We then use the initialization method in [44] to initialize all the Q-bit individuals in Φ. For each Q-bit individual ϕ, we convert it to its corresponding solution and perform the evaluation procedure in Algorithm 1 to evaluate the solution's quality. The solution with the highest quality is identified as ζ_{best}. Next, as long as the termination criterion in Line 6 is not met, we apply the updating rule in (16) to update ϕ and further obtain its corresponding solution ζ_ϕ. We use Algorithm 1 to evaluate the quality of the obtained solution ζ_ϕ. If ζ_ϕ outperforms ζ_{best} with higher quality, we use ζ_ϕ to replace ζ_{best}, i.e., the best solution identified in the previous iteration. When this iterative procedure terminates, ζ_{best} is returned as the final solution to the scheduling problem.

The computational efficiency of the scheduling procedure is dependent on the complexity of the solution evaluation procedure described in Algorithm 1. It is worth mentioning that the proposed scheduling algorithm is performed in a static way (i.e., before the application's tasks have been submitted to cloud for execution). The time spent in determining an optimized scheduling solution is, in fact, negligible compared with the execution time for the complete pan-sharpening flow. As will be demonstrated in experiments, for different numbers of workers and data sizes, solving the formulated scheduling problem by the QEA-based algorithm can be done in seconds. In contrast, by employing all available workers, the execution time for the pan-sharpening flow can range from several hundred seconds to over one thousand seconds. Also, the computation cost of the scheduling procedure is not associated with the input data size, as long as the task duration fitting model in (12) has been established relying on a set of premeasure duration data. Therefore, the overhead of the QEA-based algorithm has very little impact on the overall computational efficiency of the proposed approach.

TABLE I
SOFTWARE DESCRIPTION OF EXPERIMENTAL ENVIRONMENT

Software	Version Number
Spark	2.1.1
Hadoop	2.7.3
Ubuntu	16.04.2 LTS
VMware vSphere	5.1.0
Java SE	1.8.0
Scala	2.12.1
Python	2.7
TensorFlow	0.12.1

IV. EXPERIMENTAL RESULTS

A. Experimental Setup

In this paper, the cloud computing platform used for experimental evaluation is built on a Spark cluster consisting of one master node and six slave nodes. The master node is a virtual machine built on a host equipped with eight-core Intel Xeon E500 CPUs with 242-GB memory operating at 2.5 GHz. The six slave nodes are virtual machines built on three blade machines, each of which is equipped with eight-core Intel Xeon E5-2680 CPUs with 242-GB memory operating at 2.5 GHz. After virtualization using VMware vSphere, the master node is allocated with 24 cores (logic processors) and 242-GB memory, and each slave node is allocated with 24 cores and 240-GB memory. During Apache Spark execution, every slave node launches 20 TensorFlowOnSpark worker instances, and each worker is allocated with one core and 12-GB memory. All nodes have installed Apache Spark, Hadoop, and Ubuntu as the operating system. The parallel version of the DNN-based pan-sharpening method was implemented by Java and Scala hybrid programming. Table I lists all software and their version numbers used in experiments.

We use QuickBird satellite data to verify the accuracy and computational efficiency by using the proposed optimization framework to execute the DNN-based fusion process. The QuickBird satellite provides a panchromatic image with 0.7-m resolution and a multispectral image with 2.8-m resolution and four bands. The remote sensing data set used in

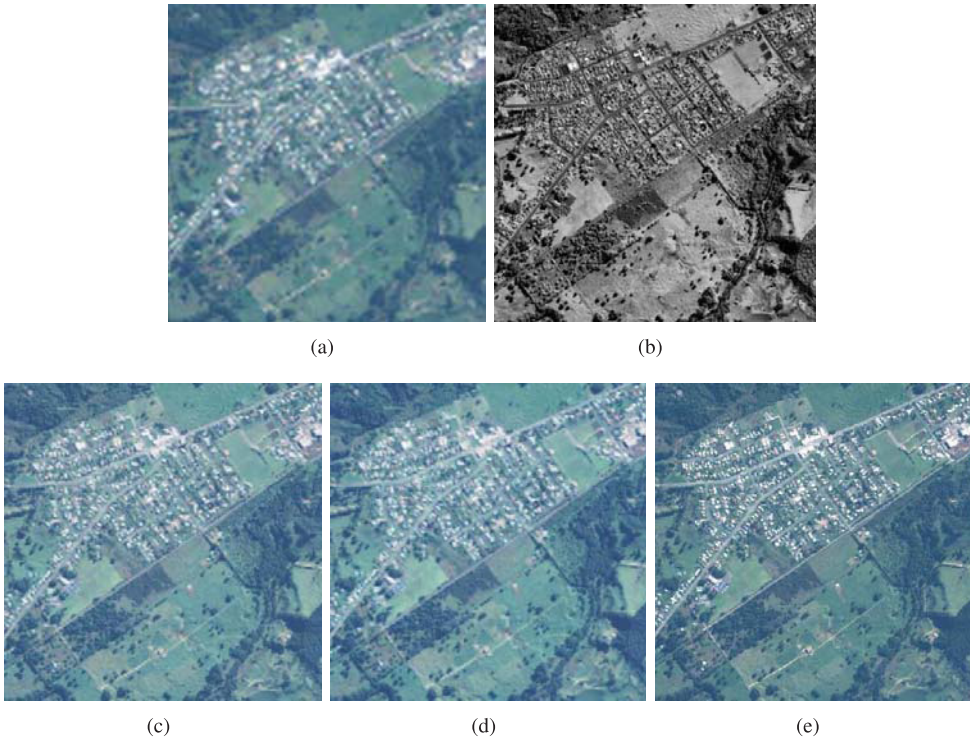


Fig. 6. QuickBird images and experimental results by the DNN-based pan-sharpening method. (a) LR multispectral image. (b) HR panchromatic image. (c) Fused image by the serial implementation of the DNN-based method. (d) Fused image by the proposed cloud computing framework. (e) Pseudocolor image of the reference multispectral image.

experiments was captured over North Island, New Zealand, in August 2012. The size of the HR panchromatic image is 600×600 pixels, while the size of the LR multispectral image is $150 \times 150 \times 4$ pixels. We first perform interpolation amplification on the LR multispectral image to obtain an LR multispectral image with $600 \times 600 \times 4$ pixels and further obtain the average-weighted LR panchromatic image with 600×600 pixels. At the training stage, we used a 7×7 convolution matrix to extract 352 836 pairs from the HR and LR panchromatic images and randomly selected 200 000 pairs among them as the training data for the DNN model. The size of the training data is accordingly 156 MB, including HR panchromatic image patches of size 78.8 MB ($200\,000 \times 49$ pixels) and LR panchromatic image patches of size 78.8 MB ($200\,000 \times 49$ pixels), respectively. The selected pairs of image patches are used as the input data to train the DNN. As claimed previously, when performing the pan-sharpening flow, the original data set is split into multiple partitions to facilitate distributed processing.

B. Accuracy Evaluation

We first show visually the pan-sharpening results by using the proposed distributed parallel implementation of the DNN-based method. We perform the DNN-based pan-sharpening process by using the proposed framework that incorporates both MapReduce mechanism and scheduling strategy to explore the optimal cloud computing solution. We also implement the serial version of the DNN-based method, in which all subtasks of the pan-sharpening process are executed in serial on a single baseline machine. Fig. 6(a) and (b) shows the HR multispectral image and LR panchromatic image, respectively. Fig. 6(c) and (d) presents the fused image by the serial version DNN-based method and the proposed method, respectively. For comparison purpose, the pseudocolor image of the reference multispectral image is also provided in Fig. 6(e). We can observe that the DNN-based method performs well in remote sensing image fusion, both in the serial version and the distributed version.

To quantitatively evaluate the accuracy of image pan-sharpening, we use root-mean-square error (RMSE) to measure the spectral similarity between the fused image and the original image. A small RMSE value indicates that the pixel values of the fused image are close to the original image, demonstrating high fusion accuracy. The RMSE value is calculated as follows:

$$\mathrm{RMSE}_k = \sqrt{\frac{1}{\mathrm{MN}} \sum_{i=1}^{M} \sum_{j=1}^{N} (X_k(i,j) - Y_k(i,j))^2} \qquad (18)$$

where X_k and Y_k stand for the kth band of reference image X and kth band of fused image, respectively, and the image of each band is of size $M \times N$. We average the RMSE values of all bands, $\mathrm{RMSE}_{\mathrm{AVG}}$, to evaluate the image fusion quality. For comparison, we perform the same experiments by using two traditional pan-sharpening methods, á trous wavelet transform (ATWT) [49] and SparseFI [50], to justify the accuracy of the proposed method. Table II lists the $\mathrm{RMSE}_{\mathrm{AVG}}$ values achieved by different pan-sharpening methods.

TABLE II
COMPARISON OF PAN-SHARPENING ACCURACY AMONG DIFFERENT METHODS

	ATWT	SparseFI	PS_DNN_S	PS_DNN_P					
				2 workers	4 workers	8 workers	16 workers	32 workers	64 workers
$RMSE_{AVG}$	0.0070	0.0064	0.0053	0.0054	0.0056	0.0055	0.0056	0.0057	0.0056

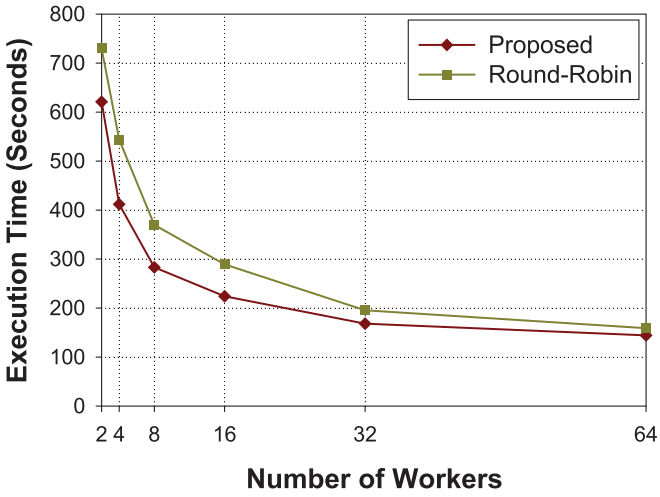


Fig. 7. Execution time comparison between the proposed scheduling algorithm and round-robin algorithm with different numbers of workers.

"PS_DNN_S" and "PS_DNN_P" denote the serial version and the parallel version of the DNN-based fusion method, respectively. When executing "PS_DNN_P" by using the proposed big data processing framework, we gradually increase the number of workers and observe that the $RMSE_{AVG}$ value keeps stable as the number of workers grows. More importantly, compared with ATWT and SparseFI methods, the proposed method leads to smaller $RMSE_{AVG}$ values, indicating a higher image fusion quality.

C. Computational Efficiency Evaluation

Having verified the image fusion accuracy by distributed processing of the pan-sharpening method, we further evaluate the computational efficiency of the proposed optimization framework. One key contribution of this paper is to improve computational efficiency by making decisions on both task partitioning and task assignments. We first verify that the proposed scheduling approach outperforms traditional schedulers by reducing the total execution time. Based upon the same optimization framework, we evaluated the execution time of the DNN-based pan-sharpening flow by the proposed QEA-based scheduling algorithm and compare it with the result by round-robin algorithm, which is a widely used scheduling strategy in existing cloud computing architectures. The comparison results in Fig. 7 demonstrate that for various numbers of workers as well as data sizes, the execution time for the whole pan-sharpening flow on the original QuickBird data set has been reduced to different extents. Note that the number of workers on the horizontal axis denotes the maximum number of all available workers deployed on TensorFlowOnSpark. As reported in Fig. 7, the reduction in execution time ranges from 9.26% to 24.22%. The reason is that in the round-robin strategy, tasks are randomly allocated onto workers, whereas the QEA-based scheduling algorithm is capable of assigning an appropriate number of workers for partitionable tasks according to their computation loads. Note that for each configuration of worker nodes, we repeat the round-robin scheduling procedure several times and present the average value of execution time.

We then evaluate the performance of the proposed optimization framework for processing remote sensing data of different sizes. For this purpose, we mosaicked the original QuickBird data set (an HR panchromatic image and an LR multispectral image) to produce large-scale images. It is worthwhile to point out that due to the high computational complexity of DNN training, even a small-scale data set would generate a large amount of training data. To be specific, for the original QuickBird data set, the amount of training data may reach 158 MB during the training stage and up to 556 MB during the reconstruction stage. Following the same fusion process, the amount of training data also grows proportionally to the sizes of mosaicked panchromatic and multispectral images, resulting in large-scale training data of sizes 884 MB, 1.8, 3.6, 7.2, and 14.4 GB, respectively. When performing the DNN-based pan-sharpening flow, it would be a particularly challenging task to deal with remotely sensed big data on a single machine.

Table III presents the execution time statistics by this parallel processing framework for various data sizes with different number of workers. The first column provides the amounts of training data generated used for DNN training. Training sets of sizes 884 MB, 1.8, 3.6, 7.2, and 14.4 GB are taken into consideration in experiments. As the number of workers increases, the execution time of the whole processing flow for each data set is reported. Also, we use the scenario of a single worker as the baseline scenario (i.e., to perform the pan-sharpening flow in a serial manner) and evaluate the computational efficiency in other scenarios with multiple workers over the baseline scenario. The speedup achieved by increasing the number of workers, which is calculated as the ratio of execution time, is provided in Fig. 8. Each curve illustrates the speedup trend with regard to worker count for a data set of a particular size.

Referring to the execution time statistics reported in Table III and Fig. 8, we observe that when the number of workers is below a certain value, the speedup shows roughly a linear increase with regard to worker count. This observation

TABLE III
EXECUTION TIME STATISTICS (SECONDS) FOR VARIOUS DATA SIZES WITH DIFFERENT NUMBERS OF WORKERS

Training Data Size	PS_DNN_S	PS_DNN_P					
		2 workers	4 workers	8 workers	16 workers	32 workers	64 workers
158 MB	1195.03	620.74	413.02	284.05	223.88	169.10	145.10
884 MB	3620.11	1906.10	1035.47	597.55	403.27	307.47	252.47
1.8 GB	7208.11	3682.95	1934.70	1045.27	644.75	448.25	386.70
3.6 GB	13087.47	6584.09	3503.17	1889.95	1129.98	743.49	590.65
7.2 GB	23626.10	11807.92	6040.56	3190.22	1914.75	1240.07	953.18
14.4 GB	43858.63	22050.93	11068.53	5887.67	3401.99	2097.48	1663.93

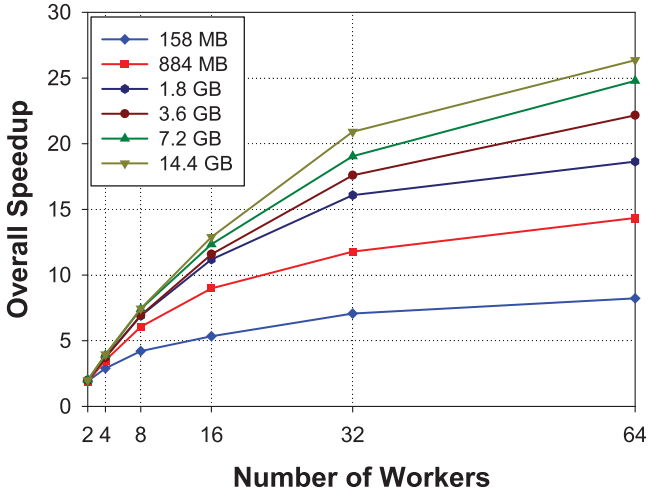


Fig. 8. Speedup achieved by the proposed method for various data sizes with different numbers of workers.

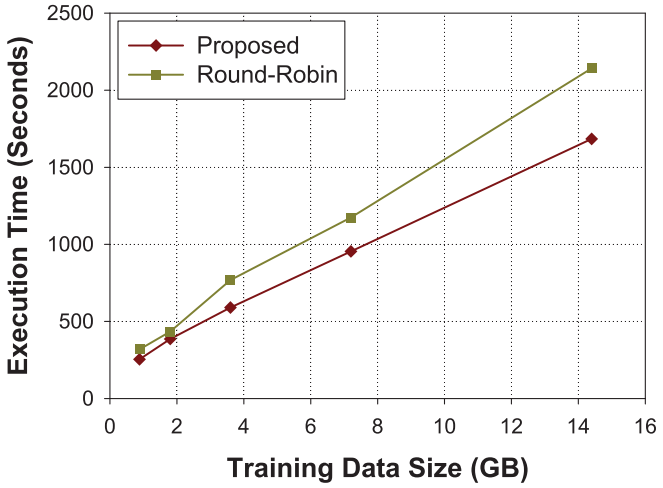


Fig. 9. Execution time comparison between the proposed scheduling algorithm and round-robin algorithm for different data sizes with 64 workers.

is due to the fact that with few worker nodes, the actual task computation time dominates the communication time among nodes. However, if we keep on adding more workers, the speedup would be compromised. For example, when the size of training data is 1.8 GB, employing 32 workers leads to a 16.08× speedup, and the speedup has been improved to 18.64% by doubling the number of workers. Similar speedup trends with regard to worker count can be observed for all data sets. The reason is that with an increasing number of workers, the communication overhead between the PS node and worker nodes also increases considerably. The amount of time spent in updating node parameters and synchronizing node jobs is no longer negligible compared to the amount of actual computation time. This observation also validates the importance of the optimization strategy in the proposed framework to fully utilize the computing resources. Having studied the dependence of speedup upon the number of workers, we further investigate the relationship between computational efficiency and data size. The last column in Table III lists the speedups achieved for different data sizes with worker count fixed at 64. The results demonstrate that the computational efficiency has been continuously improved as data size scales up. For example, the speedup is merely 8.23× for the original data set of size 158 MB but reaches 26.36× for the largest data set of size 14.4 GB. This improvement is because the system and communication overhead is more closely dependent on worker count rather than data size. With the increasing data size, the actual computation time starts to account for a majority of the overall execution time, as its ratio to the induced time overhead goes up. It is also worth mentioning that the scheduling strategy also contributes to reducing the total execution time and improving the computational efficiency for large-scale data sets.

Finally, we verify the applicability and scalability of the proposed optimization framework when processing large-scale remote sensing data. With the number of workers fixed at 64, we further observe the change in execution time for performing the whole pan-sharpening flow as the training data size increases. Fig. 9 presents the execution time for processing 884-MB, 1.8-, 3.6-, 7.2-, and 14.4-GB training data, respectively. For comparison purpose, the execution time statistics for these data sets by round-robin strategy are also provided. By using QEA-based scheduling strategy, the execution time for processing 884-MB data is 145.10 s. When the data size increases by four times (3.6 GB), the execution time is 590.65 s, and the execution time for

14.4 GB data is 1663.93 s. The results verify the scalability of the proposed method when processing large-scale remote sensing data. More importantly, compared with round-robin scheduling strategy, the QEA-based scheduling algorithm can produce high-quality scheduling results, leading to up to 23.17% reductions in execution time.

V. Extension to Public/Commercial Clouds

In this paper, we assume that the distributed processing of remote sensing big data was done in a private cloud environment. For this reason, only resource constraints are considered in the optimization procedure. In this section, we discuss strategies to extend the current framework to be applicable for public clouds and hybrid clouds.

Specifically, we consider the following two different scenarios in public clouds.

1) One the one hand, cloud users may rent a fixed number of computing resources. In this scenario, there exists an upper bound limit on available computing resources, which is similar to our experimental environment, and our proposed method can accordingly fit into a public cloud environment.
2) On the other hand, if cloud users can rent cloud resources in an on-demand manner and pay per use, cost would become the primary concern, which is distinct from our problem concerning computing resource capacity.

To cope with various cloud environments, the proposed optimization framework has to be adapted by incorporating a set of additional optimization objectives and constraints for different purposes, e.g., energy efficiency [51], [52], quality-of-service improvement, and cost minimization.

Another important concern in public cloud environments is that task durations may vary slightly due to glitches in the virtualization mechanisms or unpredictable latency issues. To cope with such variations, task durations should be modeled as random variables rather than deterministic values. Consequently, the scheduling problem becomes a stochastic optimization problem, in which task starting times and ending times would have uncertainties induced by task duration variations. Under this circumstance, a straightforward method is to replace the nominal task duration by its interval information (i.e., its upper bound value and lower bound value), based upon which we can predict the range of task completion time and satisfy the precedence constraints. We may also employ advanced uncertainty analysis techniques, e.g., expansion methods and stochastic response surface methods, to handle the variations of task durations and the resultant uncertainty in the scheduling framework. Nevertheless, by using either interval analysis or uncertainty analysis methods, it is necessary to know in advance the stochastic characteristics of task durations in public/commercial clouds.

VI. Conclusion

This paper presents a new cloud computing framework that integrates a distributed processing mechanism and a task scheduling strategy into an optimization procedure to enable efficient and scalable processing of large-scale remotely sensed data. As a case study, we optimize a DNN-based fusion method on Spark to verify the efficiency of the proposed framework. Experimental results demonstrate that the proposed framework achieves promising speedups as compared with the serial processing approach. More importantly, this framework is also scalable with regard to the increasing scale and dimensionality of remote sensing data.

The proposed framework for parallel processing of remote sensing applications involves both theoretical optimization and practical implementation. On the one hand, we used a theoretical analysis to construct a rigorous optimization model for the concerned scheduling problem. The decision variables of the optimization problem include task assignments as well as the number of data partitions for each partitionable task. We further develop a QEA-based metaheuristic algorithm to solve this complicated optimization problem. During the problem-solving procedure, certain practical information is required to determine the task durations for specified configurations. On the other hand, we can apply the results obtained by theoretical optimization to the practical implementation of the parallel pan-sharpening flow on Spark. The optimization results include the numbers of partitions for partitionable tasks and the assignments of workers executing all tasks. By incorporating these results into Spark and replacing the built-in scheduler, the practical execution time for the pan-sharpening flow can be expected to reduce significantly.

Acknowledgment

The authors would like to thank the Editors and the Anonymous Reviewers for their outstanding comments and suggestions, which greatly helped them to improve the technical quality and presentation of this paper.

References

[1] M. M. U. Rathore, A. Paul, A. Ahmad, B. W. Chen, B. Huang, and W. Ji, "Real-time big data analytical architecture for remote sensing application," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 8, no. 10, pp. 4610–4621, Oct. 2015.

[2] 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.

[3] L. Wang, H. Zhong, R. Ranjan, A. Zomaya, and P. Liu, "Estimating the statistical characteristics of remote sensing big data in the wavelet transform domain," *IEEE Trans. Emerg. Topics Comput.*, vol. 2, no. 3, pp. 324–337, Sep. 2014.

[4] J.-G. Lee and M. Kang, "Geospatial big data: Challenges and opportunities," *Big Data Res.*, vol. 2, no. 2, pp. 74–81, 2015.

[5] X. Wu, X. Zhu, G.-Q. Wu, and W. Ding, "Data mining with big data," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 1, pp. 97–107, Jan. 2014.

[6] F. Lin, L. Chung, C. Wang, W. Ku, and T. Chou, "Storage and processing of massive remote sensing images using a novel cloud computing platform," *Gisci. Remote Sens.*, vol. 50, no. 3, pp. 322–336, 2013.

[7] P. Cappelaere, S. Sánchez, S. Bernabé, A. Scuri, D. Mandl, and A. Plaza, "Cloud implementation of a full hyperspectral unmixing chain within the NASA Web coverage processing service for EO-1," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 6, no. 2, pp. 408–418, Apr. 2013.

[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] P. Wang, J. Wang, Y. Chen, and G. Ni, "Rapid processing of remote sensing images based on cloud computing," *Future Gener. Comput. Syst.*, vol. 29, no. 8, pp. 1963–1968, 2013.
[11] D. Miner and A. Shook, *MapReduce Design Patterns*. Sebastopol, CA, USA: O'Reilly Media, 2012.
[12] M. N. Kavyasri and B. Ramesh, "Comparative study of scheduling algorithms to enhance the performance of virtual machines in cloud computing," in *Proc. Int. Conf. Emerg. Trends Eng., Technol. Sci.*, 2016, pp. 1–5.
[13] N. Patil and D. Aeloor, "A review—Different scheduling algorithms in cloud computing environment," in *Proc. Int. Conf. Intell. Syst. Control*, 2017, pp. 182–185.
[14] C.-W. Tsai and J. J. P. C. Rodrigues, "Metaheuristic scheduling for cloud: A survey," *IEEE Syst. J.*, vol. 8, no. 1, pp. 279–291, Mar. 2014.
[15] X. Lu, B. Wang, X. Zheng, and X. Li, "Exploring models and data for remote sensing image caption generation," *IEEE Trans. Geosci. Remote Sens.*, vol. 56, no. 4, pp. 2183–2195, Apr. 2018.
[16] G. Cheng, P. Zhou, and J. Han, "Learning rotation-invariant convolutional neural networks for object detection in VHR optical remote sensing images," *IEEE Trans. Geosci. Remote Sens.*, vol. 54, no. 12, pp. 7405–7415, Dec. 2016.
[17] J. Han, R. Quan, D. Zhang, and F. Nie, "Robust object co-segmentation using background prior," *IEEE Trans. Image Process.*, vol. 27, no. 4, pp. 1639–1651, Apr. 2018.
[18] G. Cheng, J. Han, and X. Lu, "Remote sensing image scene classification: Benchmark and state of the art," *Proc. IEEE*, vol. 105, no. 10, pp. 1865–1883, Oct. 2017.
[19] X. Lu, X. Li, and L. Mou, "Semi-supervised multitask learning for scene recognition," *IEEE Trans. Cybern.*, vol. 45, no. 9, pp. 1967–1976, Sep. 2015.
[20] W. Huang, L. Xiao, Z. Wei, H. Liu, and S. Tang, "A new pan-sharpening method with deep neural networks," *IEEE Geosci. Remote Sens. Lett.*, vol. 12, no. 5, pp. 1037–1041, May 2015.
[21] W. Huang, L. Xiao, H. Liu, and Z. Wei, "Hyperspectral imagery super-resolution by compressive sensing inspired dictionary learning and spatial-spectral regularization," *Sensors*, vol. 15, no. 1, pp. 2041–2058, 2015.
[22] F. Magoulés, J. Pan, and F. Teng, *Cloud Computing: Data-Intensive Computing and Scheduling*. London, U.K.: Chapman & Hall, 2012.
[23] H. El-Rewini, T. G. Lewis, and H. H. Ali, *Task Scheduling in Parallel and Distributed Systems*. New York, NY, USA: Prentice-Hall, 2007.
[24] R. S. Ferreira *et al.*, "A set of methods to support object-based distributed analysis of large volumes of earth observation data," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 10, no. 2, pp. 681–690, Feb. 2017.
[25] G. A. O. P. Costa, C. Bentes, R. S. Ferreira, R. Q. Feitosa, and D. A. B. Oliveira, "Exploiting different types of parallelism in distributed analysis of remote sensing data," *IEEE Geosci. Remote Sens. Lett.*, vol. 14, no. 8, pp. 1298–1302, Aug. 2017.
[26] X. Lu, W. Zhang, and X. Li, "A coarse-to-fine semi-supervised change detection for multispectral images," *IEEE Trans. Geosci. Remote Sens.*, vol. 56, no. 6, pp. 3587–3599, Jun. 2018.
[27] J. Han, D. Zhang, G. Cheng, N. Liu, and D. Xu, "Advanced deep-learning techniques for salient and category-specific object detection: A survey," *IEEE Signal Process. Mag.*, vol. 35, no. 1, pp. 84–100, Jan. 2018.
[28] G. Cheng, C. Yang, X. Yao, L. Guo, and J. Han, "When deep learning meets metric learning: Remote sensing image scene classification via learning discriminative CNNs," *IEEE Trans. Geosci. Remote Sens.*, vol. 56, no. 5, pp. 2811–2821, May 2018.
[29] X. Lu, Y. Chen, and X. Li, "Hierarchical recurrent neural hashing for image retrieval with hierarchical convolutional features," *IEEE Trans. Image Process.*, vol. 27, no. 1, pp. 106–120, Jan. 2018.
[30] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.
[31] J. Xie, L. Xu, and E. Chen, "Image denoising and inpainting with deep neural networks," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2012, pp. 341–349.
[32] 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.
[33] 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.
[34] M. Abadi *et al.*, "TensorFlow: A system for large-scale machine learning," in *Proc. USENIX Symp. Oper. Syst. Design Implement.*, 2016, pp. 265–283.
[35] H. El-Rewini and M. Abd-El-Barr, *Advanced Computer Architecture and Parallel Processing*. Hoboken, NJ, USA: Wiley, 2005.
[36] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, no. 3, pp. 260–274, Mar. 2002.
[37] J. D. Ullman, "NP-complete scheduling problems," *J. Comput. Syst. Sci.*, vol. 10, no. 3, pp. 384–393, 1975.
[38] L. A. Wolsey and G. L. Nemhauser, *Integer and Combinatorial Optimization*. Hoboken, NJ, USA: Wiley, 1999.
[39] O. Sinnen, *Task Scheduling for Parallel Systems*. Hoboken, NJ, USA: Wiley, 2007.
[40] J. S. F. Barker, "Simulation of genetic systems by automatic digital computers," *Austral. J. Biol. Sci.*, vol. 11, no. 4, pp. 603–612, 1958.
[41] J. H. Holland, *Adaptation in Natural and Artificial Systems*. Cambridge, MA, USA: MIT Press, 1992.
[42] L. J. Fogel, A. J. Owens, and M. J. Walsh, *Artificial Intelligence Through Simulated Evolution*. New York, NY, USA: Wiley, 1966.
[43] H. P. Schwefel, *Evolution and Optimum Seeking*. New York, NY, USA: Wiley, 1995.
[44] 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.
[45] G. Zhang, "Quantum-inspired evolutionary algorithms: A survey and empirical study," *J. Heuristics*, vol. 17, no. 3, pp. 303–351, Jun. 2011.
[46] 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.
[47] Q. Niu, T. Zhou, and S. Ma, "A quantum-inspired immune algorithm for hybrid flow shop with makespan criterion," *J. Universal Comput. Sci.*, vol. 15, no. 4, pp. 765–785, 2009.
[48] 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.
[49] J. Nunez, X. Otazu, O. Fors, A. Prades, V. Palà, and R. Arbiol, "Multiresolution-based image fusion with additive wavelet decomposition," *IEEE Trans. Geosci. Remote Sens.*, vol. 37, no. 3, pp. 1204–1211, May 1999.
[50] X. X. Zhu and R. Bamler, "A sparse image fusion algorithm with application to pan-sharpening," *IEEE Trans. Geosci. Remote Sens.*, vol. 51, no. 5, pp. 2827–2836, May 2013.
[51] F. Xiao, Z. Jiang, X. Xie, L. Sun, and R. Wang, "An energy-efficient data transmission protocol for mobile crowd sensing," *Peer-Peer Netw. Appl.*, vol. 10, no. 3, pp. 510–518, 2017.
[52] F. Xiao, G. Ge, L. Sun, and R. Wang, "An energy-efficient data gathering method based on compressive sensing for pervasive sensor networks," *Pervasive Mobile Comput.*, vol. 41, pp. 343–353, Oct. 2017.

Jin Sun (M'17) 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 a Technical Staff Member with Orora Design Technologies, Inc., Redmond, WA, USA. 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, integrated circuit modeling and analysis, and computer-aided design.

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

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

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

Zebin Wu (M'13–SM'18) was born in Hangzhou, Zhejiang, China, in 1981. He 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 Univ. 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, high-performance computing, and computer simulation.

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.

Xianliang Yin received the B.S. degree in computer technology from Nanchang University, Nanchang, China, in 2015, and the M.S. degree in computer technology from the Nanjing University of Science and Technology, Nanjing, China, in 2018.

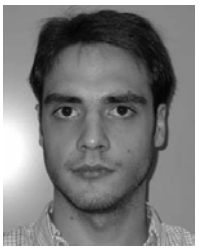
His research interests include hyperspectral image classification, cloud computing, and task scheduling.

Zhongzheng Ding received the B.S. degree in computer science and technology from the Anhui University of Technology, Ma'anshan, China, in 2015, and the M.S. degree in computer technology from the Nanjing University of Science and Technology, Nanjing, China, in 2018.

His research interests include multispectral image fusion and task scheduling in cloud computing.

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 a 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.



Javier Plaza (M'09–SM'15) received the M.Sc. and Ph.D. degrees in computer engineering from the University of Extremadura, Cáceres, Spain, in 2004 and 2008, respectively.

He is currently a member of the Hyperspectral Computing Laboratory, Department of Technology of Computers and Communications, University of Extremadura, Badajoz, Spain. He has authored more than 120 publications, including 40 JCR journal papers, 10 book chapters, and 80 peer-reviewed conference proceeding papers. He has guest edited two special issues on hyperspectral remote sensing for different journals. His research interests include hyperspectral data processing and parallel computing of remote sensing data.

Dr. Plaza was a recipient of the Outstanding Ph.D. Dissertation Award from the University of Extremadura in 2008, the Best Column Award of *IEEE Signal Processing Magazine* in 2015, the most highly cited paper in the *Journal of Parallel and Distributed Computing* from 2005 to 2010, and best paper awards at the IEEE International Conference on Space Technology and IEEE Symposium on Signal Processing and Information Technology. He is an Associate Editor for IEEE GEOSCIENCE AND REMOTE SENSING LETTERS and the IEEE Remote Sensing Code Library.

Antonio Plaza (M'05–SM'07–F'15) 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.

He is currently the Head of the Hyperspectral Computing Laboratory, Department of Technology of Computers and Communications, University of Extremadura. He has authored more than 600 publications, including 200 JCR journal papers, 145 IEEE journals, 23 book chapters, and 285 peer-reviewed conference proceeding papers. He has guest edited ten special issues on hyperspectral remote sensing for different journals. He has reviewed more than 500 manuscripts for more than 50 different journals. His research interests include hyperspectral data processing and parallel computing of remote sensing data.

Dr. Plaza was a member of the Editorial Board of *IEEE Geoscience and Remote Sensing Newsletter* from 2011 to 2012 and *IEEE Geoscience and Remote Sensing Magazine* in 2013. He is a Fellow of IEEE for contributions to hyperspectral data processing and parallel computing of Earth observation data, and also a member of the Steering Committee of the IEEE JOURNAL OF SELECTED TOPICS IN APPLIED EARTH OBSERVATIONS AND REMOTE SENSING (JSTARS). He was a recipient of the recognition of Best Reviewers of IEEE GEOSCIENCE AND REMOTE SENSING LETTERS in 2009, the recognition of Best Reviewers of the IEEE TRANSACTIONS ON GEOSCIENCE AND REMOTE SENSING in 2010, the Best Column Award of *IEEE Signal Processing Magazine* in 2015, the 2013 Best Paper Award of the JSTARS, the most highly cited paper in the *Journal of Parallel and Distributed Computing* from 2005 to 2010, and the best paper awards at the IEEE International Conference on Space Technology and the IEEE Symposium on Signal Processing and Information Technology. He served as an Associate Editor for the IEEE TRANSACTIONS ON GEOSCIENCE AND REMOTE SENSING. He is an Associate Editor of the IEEE ACCESS. He served as the Director of Education Activities for the IEEE Geoscience and Remote Sensing Society (GRSS) from 2011 to 2012, and as a President of the Spanish Chapter of IEEE GRSS from 2012 to 2016.