Parallel and Distributed Dimensionality Reduction of Hyperspectral Data on Cloud Computing Architectures

Zebin Wu, *Member, IEEE*, Yonglong Li, Antonio Plaza, *Fellow, IEEE*, Jun Li, *Member, IEEE*, Fu Xiao, *Member, IEEE*, and Zhihui Wei

***Abstract*—Cloud computing offers the possibility to store and process massive amounts of remotely sensed hyperspectral data in a distributed way. Dimensionality reduction is an important task in hyperspectral imaging, as hyperspectral data often contains redundancy that can be removed prior to analysis of the data in repositories. In this regard, the development of dimensionality reduction techniques in cloud computing environments can provide both efficient storage and preprocessing of the data. In this paper, we develop a parallel and distributed implementation of a widely used technique for hyperspectral dimensionality reduction: principal component analysis (PCA), based on cloud computing architectures. Our implementation utilizes Hadoop's distributed file system (HDFS) to realize distributed storage, uses Apache Spark as the computing engine, and is developed based on the map-reduce parallel model, taking full advantage of the high throughput access and high performance distributed computing capabilities of cloud computing environments. We first optimized the traditional PCA algorithm to be well suited for parallel and distributed computing, and then we implemented it on a real cloud computing architecture. Our experimental results, conducted using several hyperspectral datasets, reveal very high performance for the proposed distributed parallel method.**

Manuscript received September 29, 2015; revised March 08, 2016; accepted March 11, 2016. This work was supported in part by the National Natural Science Foundation of China under Grant 61471199, Grant 91538108, and Grant 11431015, in part by the Research Fund of Jiangsu High Technology Research Key Laboratory for Wireless Sensor Networks under Grant WSNLBKF201507, and in part by the Jiangsu Province Six Top Talents project of China under Grant WLW-011.

Z. Wu is with the School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing 210094, China, also with the Jiangsu High Technology Research Key Laboratory for Wireless Sensor Networks, Nanjing 210003, China, and also with the Hyperspectral Computing Laboratory, Escuela Politécnica, University of Extremadura, E-10003 Cáceres, Spain (e-mail: zebin.wu@gmail.com).

Y. Li and Z. Wei are with the School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing 210094, China (e-mail: sky_lyl@sina.cn; gswei@njust.edu.cn).

A. Plaza is with the Hyperspectral Computing Laboratory, Department of Technology of Computers and Communications, Escuela Politécnica, University of Extremadura, E-10003 Cáceres, Spain (e-mail: aplaza@unex.es).

J. Li is with the Guangdong Provincial Key Laboratory of Urbanization and Geo-Simulation, and 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).

F. Xiao is with the School of Computer, Nanjing University of Posts and Telecommunications, Nanjing 210003, China, and also with the Jiangsu High Technology Research Key Laboratory for Wireless Sensor Networks, Nanjing 210003, China (e-mail: xiaof@njupt.edu.cn).

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/JSTARS.2016.2542193

***Index Terms*—Cloud computing, dimensionality reduction, Hadoop, hyperspectral imaging, principal component analysis (PCA), spark.**

I. Introduction

HYPERSPECTRAL images comprise hundreds of contiguous spectral bands, thus imposing significant requirements in terms of storage and data processing. It is important to efficiently reduce the dimensionality of hyperspectral images and extract the primary features from hundreds of highly correlated spectral bands [1]. Since high-dimensional data spaces are mostly empty and hyperspectral data concentrate primarily in a subspace [2], it can be often reduced to a subspace without affecting the data quality.

In recent decades, many techniques have been exploited for bringing hyperspectral data to a lower dimensional space [3]. Principal component analysis (PCA) is a statistical procedure that uses orthogonal transformations to convert a set of observations of (possibly correlated) variables into a set of values from linearly uncorrelated variables called principal components [4]. Since the neighboring bands in a hyperspectral image are often correlated, PCA can effectively transform the original data and remove such correlation among the bands [5]. However, the PCA algorithm is computationally intensive. To overcome this problem, some researchers have resorted to multicore central processing units (CPUs) and graphics processing units (GPUs) [6]–[8] to accelerate the PCA transform for hyperspectral data reduction.

Nevertheless, recent technological advances in satellite and airborne remote sensing have increased exponentially the volume of remote sensing image data that has been already collected and stored in hyperspectral data repositories. The availability of new hyperspectral missions producing large amounts of data on a daily basis has posed important challenges for scalable and efficient processing of hyperspectral data in the context of different applications such as those requiring dimensionality reduction for data interpretation.

For example, the data collection rate of the NASA Jet Propulsion Laboratory's Airborne Visible/Infrared Imaging Spectrometer (AVIRIS) [9] is 2.55 MB/s, which means that it can obtain nearly 9 GB of data in 1 h. The Chinese Pushbroom Hyperspectral Imager (PHI) has a data collection rate of 7.2 MB/s, and it can collect more than 25 GB data in 1 h. The space-borne Hyperion instrument collects a hyperspectral data

cube with 256×6925 pixels, 242 spectral bands and radiometric resolution of 12 bits in 30 s, collecting almost 71.9 GB data in 1 h (over 1.6 TB daily). Other satellite missions that will be soon in operation such as the environmental analysis and mapping program (EnMAP) present similar data collection ratios. Since hyperspectral data repositories are becoming increasingly massive and often distributed among several geographic locations due to their volume, it is hard to meet the storage and computational requirements of large-scale hyperspectral data processing applications without resorting to distributed computing facilities.

Distributed computing technologies are highly demanded for this kind of dynamical, on-demand processing of large hyperspectral datasets. In the past, commodity clusters [10], grid computing [11], [12], and cloud computing [13] platforms have been explored for remotely sensed data processing. Recently, cloud computing has become a standard for distributed computing due to its advanced capabilities for internet-scale computing, service-oriented computing, and high-performance computing [14]. It offers the potential to tackle massive data processing workloads by means of its distributed parallel architecture. Cloud computing can be seen as an evolution of grid computing that relies on grid computing principles as its backbone and infrastructure, so as to keep high performance distributed computing capabilities [13]. As a result, the use of cloud computing for the analysis of large hyperspectral data repositories represents a natural solution as well as an evolution of previously developed techniques for other kinds of computing platforms. Nevertheless, to the best of our knowledge (and despite the increasing demands of massive data processing in the hyperspectral imaging field), there are very few efforts in the literature oriented toward exploiting cloud computing infrastructure for hyperspectral imaging techniques in general and for dimensionality reduction algorithms in particular.

In this paper, we introduce a new parallel and distributed framework for massive hyperspectral image processing based on cloud computing. Specifically, we use dimensionality reduction as a case study to demonstrate the applicability and efficiency of utilizing cloud computing technologies to efficiently perform distributed parallel processing of hyperspectral data and accelerate hyperspectral data computations. To that end, we develop a parallel and distributed implementation of the PCA algorithm on a cloud environment, using advanced technologies such as Hadoop's distributed file system (HDFS) and Apache Spark as well as a map-reduce methodology. The efficiency of our implementation is evaluated in terms of accuracy and parallel execution performance, as compared with a serial PCA implementation on a single CPU and a distributed parallel version of PCA using Hadoop.

The remainder of this paper is organized as follows. Section II describes the proposed parallel and distributed framework. Section III presents the PCA algorithm and its distributed parallel implementation for cloud computing architectures. Section IV experimentally assesses the proposed method in terms of both accuracy and computational performance. Finally, Section V concludes the paper with some remarks and hints at plausible future research lines.

II. Parallel and Distributed Framework Design

In order to develop a parallel and distributed framework for PCA on cloud computing architectures, three main issues need to be addressed: 1) the distributed programming model; 2) the computing engine; and 3) how to obtain dynamical storage.

For distributed programming, we resort to the map-reduce model for parallel processing across clusters of computers [15], taking full advantage of the high-performance capabilities provided by the cloud computing architecture. In this model, a task is processed by two distributed operations: map and reduce [13], [16]. The datasets are organized as key/value pairs, and the map function processes a key/value pair to generate a set of intermediate pairs, dividing a task into several independent subtasks to be run in parallel. The reduce function is in charge of processing all intermediate values associated with the same intermediate key, then collecting all the subtask results to gather the result for the whole task.

Regarding the distributed computing engine, a possible solution is Apache Hadoop[1] due to its reliability and scalability as well as its completely open source nature [15]. It has been successfully applied by IBM, Yahoo, and Facebook [13]. However, Apache Hadoop only supports simple one-pass computations (e.g., aggregation or database SQL queries) and is generally not appropriate for multipass algorithms. Apache Spark[2] is a newly developed computing engine for large-scale data processing on cloud computing architectures, which implements a fault-tolerant abstraction for in-memory cluster computing, and provides fast and general data processing on large clusters. It not only supports simple one-pass computations but can also be extended to the case of multipass algorithms, required for more complex data analytics [17]. It extends the MapReduce model to include primitives for data sharing, named resilient distributed datasets (RDDs), and offers an API based on coarse-grained transformations that allows them to recover data efficiently using lineage. Aparche Spark has an advanced directed acyclic graph (DAG) execution engine that supports cyclic data flow and in-memory computing, and can run programs up to $100\times$ faster than Hadoop MapReduce in memory, or $10\times$ faster on disk. To use Apache Spark, developers should write a driver program that defines one or more RDDs, invokes actions on them, and tracks the lineage of RDDs. The driver program usually connects to a cluster of workers, which are long-lived processes that can store RDD partitions in random access memory (RAM) across operations [18]. At the Apache Spark's runtime, the user's driver program launches multiple workers, which read data blocks from a distributed file system and can persist computed RDD partitions in memory.

Our application demands dynamical storage able to distribute the data among several geographic locations. Fortunately, the HDFS[3] has been designed to be deployed on low-cost hardware, as well as to provide high throughput access to application data, especially for large datasets. We use the files in HDFS as our input RDDs on Apache Spark for which we can use common interfaces [18]. For example, the function `partitions ()`

[1][Online]. Available: https://hadoop.apache.org

[2][Online]. Available: https://spark.apache.org/

[3][Online]. Available: http://hadoop.apache.org/docs/r1.2.1/hdfs_design.html

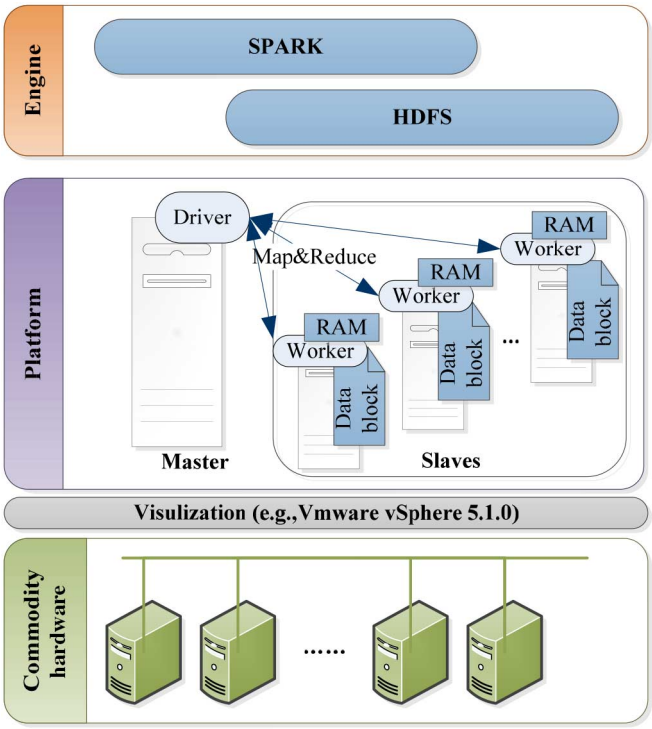


Fig. 1. Overview of our distributed parallel framework.

returns one partition for each block of the file (with the block's offset stored in each partition object). Similarly, the function `preferredLocations ()` gives a list of the nodes that contain the block, and function `iterator ()` can be used to read the block. With the aforementioned issues in mind, the design of our distributed parallel framework for hyperspectral data processing using Apache Spark and HDFS is graphically outlined in Fig. 1.

III. Distributed Parallel Implementation of PCA

A. PCA Algorithm

Let us assume that $\boldsymbol{X} = [\boldsymbol{X}_{ij}]_{N\times L}$ is a hyperspectral image with N pixel vectors and L spectral bands, where $\boldsymbol{X}_k = (\boldsymbol{X}_{k1}, \boldsymbol{X}_{k2}, \ldots, \boldsymbol{X}_{kL})^T$ is an L-dimensional vector of the kth hyperspectral pixel observation. The traditional PCA algorithm [5], [19] can be summarized by three steps:

Step 1) Compute the covariance matrix $\boldsymbol{\Sigma} = [\boldsymbol{\Sigma}_{ij}]_{L\times L}$ as

$$\boldsymbol{\Sigma}_{ij} = \frac{1}{N-1}\sum_{t=1}^{N}(\boldsymbol{X}_{ti} - \bar{\boldsymbol{X}}_i)(\boldsymbol{X}_{tj} - \bar{\boldsymbol{X}}_j) \quad (1)$$

where $\bar{\boldsymbol{X}}_j = \frac{1}{N}\sum_{t=1}^{N} X_{tj}$ denotes the mean value of the jth band.

Step 2) Perform eigendecomposition on the covariance matrix. The eigenvalues $\boldsymbol{\lambda} = [\lambda_1, \lambda_2, \ldots \lambda_m]$, arranged in descending order (so that $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_m$), and the corresponding eigenvectors $\boldsymbol{V} = [\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_m]$ can be obtained by the eigendecomposition of the covariance matrix $\boldsymbol{\Sigma}$.

Step 3) Projection transformation. The original data $\boldsymbol{X}$ can be transformed by multiplying it by $\boldsymbol{V}$ as $\boldsymbol{Y} = \boldsymbol{X}\boldsymbol{V}$. In practice, we select $k(k < m)$ eigenvalues and its eigenvectors $\boldsymbol{V}'$ (instead of $\boldsymbol{V}$) to transform $\boldsymbol{X}$, which reduces data redundancy.

B. Algorithm Optimization for Distributed Parallel Implementation

The traditional PCA algorithm is not suitable for parallel and distributed optimization due to the strong existing correlation among the pixel vectors when calculating the covariance matrix. It is quite expensive to compute the covariance matrix, which has to do the decentration for every pixel vector, and needs $2 \times N \times L \times L$ subtractions. Therefore, we optimize (1) as follows:

$$\begin{aligned}\boldsymbol{\Sigma}_{ij} &= \frac{1}{N-1}\sum_{k=1}^{N}(\boldsymbol{X}_{ki} - \bar{\boldsymbol{X}}_i)(\boldsymbol{X}_{kj} - \bar{\boldsymbol{X}}_j) \\ &= \frac{1}{N-1}\left(\sum_{k=1}^{N}\boldsymbol{X}_{ki} * \boldsymbol{X}_{kj} - N * \bar{\boldsymbol{X}}_i * \bar{\boldsymbol{X}}_j\right). \quad (2)\end{aligned}$$

By introducing $2 \times L \times L$ multiplications, we only need to perform $L \times L$ reductions. As in the case of hyperspectral data $N \gg L$, this optimization greatly accelerates the computation.

Then we have

$$\boldsymbol{\Sigma} = \frac{1}{N-1}(\boldsymbol{X}_{N\times L}^T * \boldsymbol{X}_{N\times L} - N * \bar{\boldsymbol{X}}^T * \bar{\boldsymbol{X}}) \quad (3)$$

where $\bar{\boldsymbol{X}} = (\bar{\boldsymbol{X}}_1, \bar{\boldsymbol{X}}_2, \ldots, \bar{\boldsymbol{X}}_L)$.

On the other hand, we have

$$\boldsymbol{M}_{L\times L} = [\boldsymbol{M}_{ij}]_{L\times L} = \boldsymbol{X}_{N\times L}^T * \boldsymbol{X}_{N\times L}. \quad (4)$$

Normally, the calculation of $\boldsymbol{M}_{ij}$ is based on multiplying rows by columns of the hyperspectral data matrix (as (5) and Fig. 2 show) and needs to traverse every pixel in a certain band, thus having a significant requirement in terms of RAM resources. Since N is generally large, it is difficult to meet the RAM requirements. Moreover, the multiplication of rows by columns is unsuitable to be implemented on distributed environments

$$\boldsymbol{M}_{ij} = \sum_{t=1}^{N}\boldsymbol{X}_{ti} * \boldsymbol{X}_{tj}. \quad (5)$$

Therefore, we optimize (5) by multiplying rows by rows, as described in (6) and graphically illustrated in Fig. 3

$$\boldsymbol{M}_{L\times L} = \sum_{i=1}^{N}(\boldsymbol{X}_{i*}^T * \boldsymbol{X}_{i*}) \quad (6)$$

where $\boldsymbol{X}_{i*} = (X_{i1}, X_{i2}, \ldots, X_{iL})$.

Now each pixel vector just needs to be multiplied by itself, and there is no correlation among the pixel vectors when

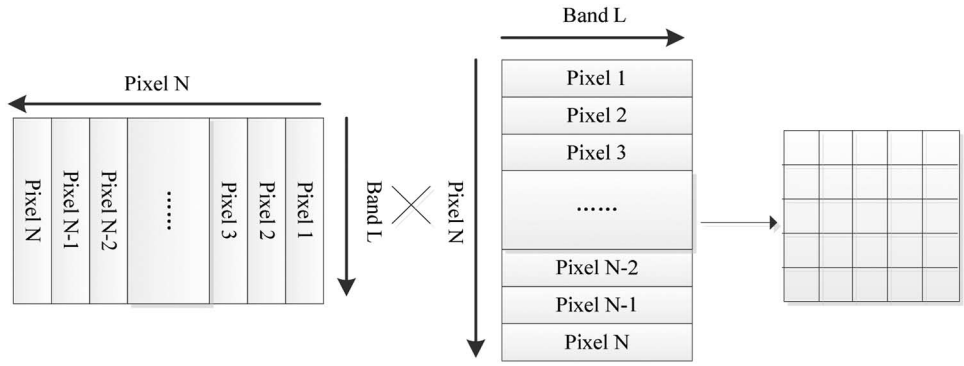


Fig. 2. Calculation of $\boldsymbol{M}_{ij}$.

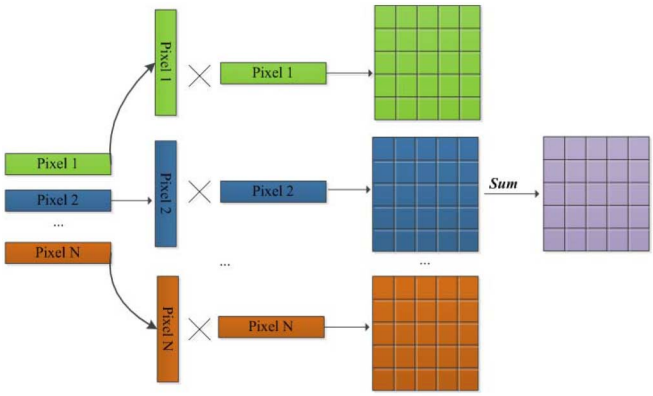


Fig. 3. Optimized calculation of $\boldsymbol{M}_{ij}$.

calculating $\boldsymbol{\Sigma}$, which is well suited for parallel and distributed computing. Moreover, every pixel vector can be sequentially read from the data by rows, leading to a good data locality of the program, and makes the cache memory more efficiently exploited.

C. Parallel and Distributed Implementation on Spark

In this section, we describe the parallel and distributed implementation of the different phases of PCA algorithm and further describe the architecture-related optimizations carried out in the development of the distributed parallel implementation.

As described above, the eigendecomposition of the covariance matrix is the most crucial and time-consuming procedure of the optimized PCA algorithm. To take full advantage of the high-performance capabilities provided by the cloud computing architecture, we first utilize the map-reduce model to optimize this procedure on Spark, as summarized in Algorithm 1.

Bearing in mind the distributed parallel framework described in Section II and the map-reduce model, the PCA algorithm can be implemented in parallel and distributed form by the following steps, which are graphically summarized in Fig. 4.

1) First, we store the original hyperspectral datasets on HDFS in a distributed and flexible way. Since the original datasets 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 (named Offset) is the offset of this partition in the original dataset, and the value (named Pixels) is the hyperspectral data partition (of byte type). Thus, the hyperspectral dataset is read from HDFS to NewHadoopRDD instances, which are then mapped into MappedRDD. After that, the value pixels are transformed into $X'_{p\times L}$, where p denotes the number of pixels in the partition. The MappedRDD is cached in the RAM for fast access.
2) Perform the map operation on DataRDD to get PCARDD. The map operation is in charge of calculating the summation $\boldsymbol{Sum}_{1\times L}$ of $X'_{p\times L}$ (the partition corresponding to the DataRDD) by bands, and the cumulative sum matrix $\boldsymbol{M'}_{L\times L} = \sum_{i=1}^{p} (\boldsymbol{X'}^{T}_{i*} * \boldsymbol{X'}_{i*})$. Since $\boldsymbol{M'}$ is a typical symmetric matrix, we just need to store the upper triangular matrix of $\boldsymbol{M'}$ as a vector r to reduce the transmission overhead associated to the intermediate data. At this moment, PCARDD will be a pair $(\boldsymbol{Sum}_{1\times L}, \boldsymbol{r})$, obtained as the output of the map operation. Then, the reduce operation is conducted on PCARDD to aggregate

Algorithm 1. The eigendecomposition of covariance matrix on Spark

Map Stage

$\boldsymbol{Input}$: $(\boldsymbol{X'}_{p\times L})$
$\boldsymbol{Output}$: $(\boldsymbol{\Lambda}^z, \boldsymbol{S}^z)$
1. $for(i = 1; i <= p; i++)\{$
2. $for(j = 1; j <= L; j++)\{$
3. $\boldsymbol{S}_j += \boldsymbol{X'}_{ij}$
4. $for(k = j; k <= L; k++)\{$
5. $\boldsymbol{\Lambda}^z_{jk} += \boldsymbol{X'}_{ij} * \boldsymbol{X'}_{ik}\}\}\}$
6. $\boldsymbol{output}(\boldsymbol{\Lambda}^t, \boldsymbol{S}^t)$

Reduce Stage

$\boldsymbol{Input}$: $((\boldsymbol{\Lambda}^1, \boldsymbol{\Lambda}^2, \ldots, \boldsymbol{\Lambda}^r), (\boldsymbol{S}^1, \boldsymbol{S}^2, \ldots, \boldsymbol{S}^r))$,
$\boldsymbol{Output}$: $(\boldsymbol{V}, \boldsymbol{\lambda})$
1. $\boldsymbol{\Lambda} = sum(\boldsymbol{\Lambda}^1, \boldsymbol{\Lambda}^2, \ldots, \boldsymbol{\Lambda}^r)$
2. $\boldsymbol{S} = sum(\boldsymbol{S}^1, \boldsymbol{S}^2, \ldots, \boldsymbol{S}^r)$
3. $\bar{\boldsymbol{X}} = \frac{1}{n}\boldsymbol{S}$
4. $for(j = 1; j <= L; j++)\{$
5. $for(k = j; k <= L; k++)\{\boldsymbol{\Lambda}_{kj} = \boldsymbol{\Lambda}_{jk}\}\}$
6. $\boldsymbol{\Sigma} = \frac{1}{N-1}(\boldsymbol{\Lambda} - N * \bar{\boldsymbol{X}}^T * \bar{\boldsymbol{X}})$
7. $[\boldsymbol{V}, \boldsymbol{\lambda}] = Eigendecomposition(\boldsymbol{\Sigma})$
8. $\boldsymbol{output}$:$(\boldsymbol{V}, \boldsymbol{\lambda})$

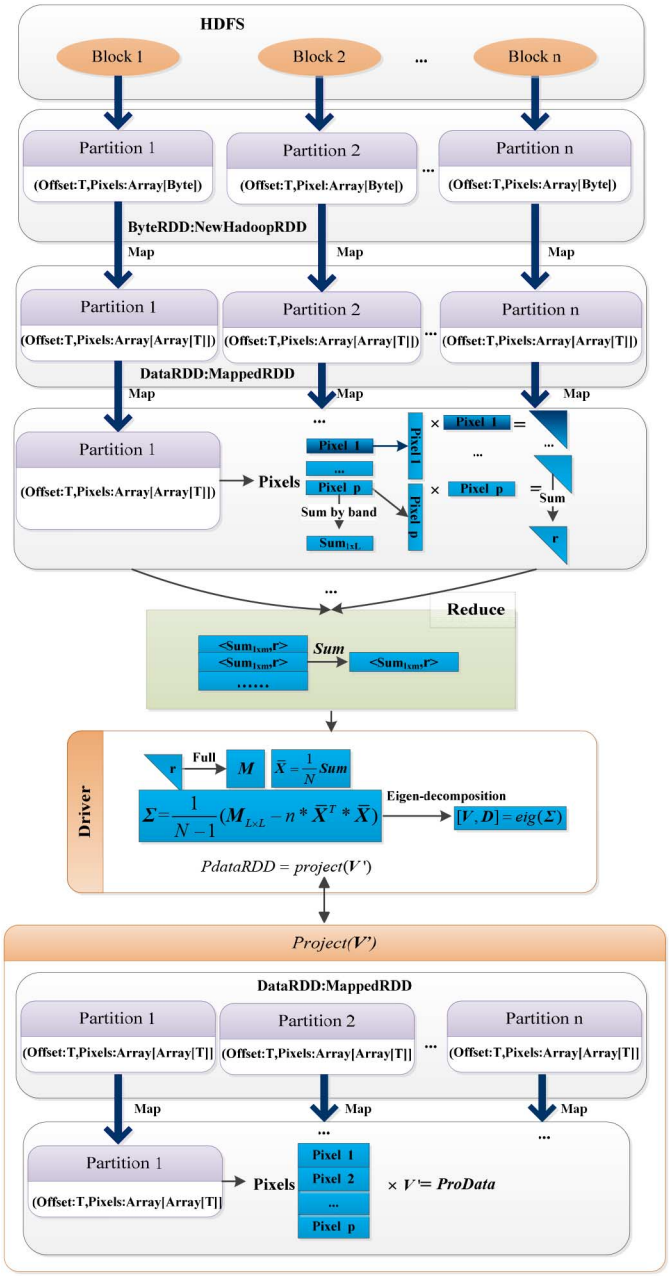


Fig. 4. Parallel and distributed implementation of PCA.

all the pairs $(\boldsymbol{Sum}_{1\times L}, \boldsymbol{r})$ by summation, and the final result is submitted to the driver node.

3) Now the mean vector $\bar{\boldsymbol{X}}$ is first computed on the driver node, and the input vector r is extended to a symmetric matrix $\boldsymbol{M}_{L\times L}$. After that, the covariance matrix $\boldsymbol{\Sigma}$ is calculated and its eigendecomposition is efficiently realized by invoking the `EigenvalueDecomposition` function included in the JAMA (Java Matrix Package)[4] library.
4) Now, we can obtain the first k eigenvalues (in descending order) and their corresponding eigenvectors to make up $\boldsymbol{V'}$ and then broadcast them to worker nodes. At this moment, `PdataRDD` is generated by the map operation, which calculates the projection of the partition data $Pixel_{p\times L}$ on the eigenvectors matrix $\boldsymbol{V'}$ named $\boldsymbol{Prodata}$. The results are paired $(Offset, \boldsymbol{Prodata})$ as the output of the map operation.

IV. Experiments

A. Experimental Configuration

The cloud computing platform used for experimental evaluation in this work is built on a cluster that comprises nine nodes. The master node is the virtual machine, built on a host (IBM X3650M3) equipped with two Intel Xeon E5630 CPUs (eight cores in total) at 2.53 GHz, with 5 GB RAM and 292 GB SAS hard disk. The eight slave nodes are virtual machines built on 4 IBM BladeCenter HX5 blade computers. Each blade computer is equipped with two Intel Xeon E7-4807 CPUs (12 cores in total) at 1.86 GHz, and is connected to a 12 TB disk array by SAS bus. After virtualization (using Vmware[5] vSphere 5.1.0), every virtual machine (slave node) is allocated with six cores (logic processors). All nodes have Java 1.6.45, Apache Spark 1.4.1, Hadoop-1.2.1, and Ubuntu 12.04 as operating system installed. During the Apache Spark execution, every slave node launches two worker instances. The distributed parallel version on Hadoop is implemented by Java programming, and the version on Spark is realized by Java and Scala hybrid programming in which Java is used for the computation part coding (the same as the Hadoop version) and Scala is used for the logic control part coding. Besides, the serial version is implemented according to the algorithm described in Section III-A and executed on one slave node.

The experiments were conducted using the well-known AVIRIS Cuprite[6] image with 224 spectral bands between 0.4 and 2.5 μm. Water absorption and low signal-to-noise ratio (SNR) bands were removed prior to the analysis, including bands $1-3$, $105-115$, $150-170$, and $223-224$. The final dataset consists of 614×512 pixels, 187 bands and a total size of about 112 MB. In order to test the performance on big datasets, five larger datasets have been artificially generated by simply mosaicking the original 112 MB dataset (denoted as Dataset1) as follows: Dataset2, with 3070×512 pixels and size of 560 MB; Dataset3 with 6140×512 pixels and size of 1.09 GB; Dataset4, with 12280×512 pixels and size of 2.18 GB; Dataset5, of 18420×512 pixels and size of 3.28 GB; Dataset6, with 24560×512 pixels and size of 4.37 GB; Dataset7, with 30700×512 pixels and size of 5.47 GB; Dataset8, with 61400×512 pixels and size of 10.95 GB; Dataset9, with 92100×512 pixels and size of 16.42 GB; and Dataset10, with 122800×512 pixels and size of 21.89 GB. The computational performance and accuracy have been assessed in our experiments in which k has been set to 100. All of the measurements reported in the following experiments are achieved after 10 Monte Carlo runs.

B. Accuracy Evaluation

First of all, we evaluate the accuracy of our parallel and distributed implementation of PCA on the original Dataset1, as compared to the serial version and a MATLAB implementation

[4][Online]. Available: http://math.nist.gov/javanumerics/jama/

[5][Online]. Available: http://www.vmware.com/

[6][Online]. Available: http://aviris.jpl.nasa.gov/data/free_data.html

TABLE I
FIRST FOUR EIGENVALUES OBTAINED BY THE THREE PCA VERSIONS

	MATLAB version	Serial version	Parallel version
Eigenvalue 1	47 572 539.585767	47 572 539.585767	47 572 539.585767
Eigenvalue 2	2 894 606.4528845	2 894 606.4528845	2 894 606.4528845
Eigenvalue 3	1 457 124.4918719	1 457 124.4918719	1 457 124.4918719
Eigenvalue 4	383 493.31328995	383 493.31328995	383 493.31328995

TABLE II
LAST FOUR EIGENVALUES OBTAINED BY THE THREE PCA VERSIONS

	MATLAB version	Serial version	Parallel version
Eigenvalue 97	24.019379371	24.019379371	24.019379371
Eigenvalue 98	23.501908381	23.501908381	23.501908381
Eigenvalue 99	23.250682031	23.250682030	23.250682030
Eigenvalue 100	22.351175438	22.351175439	22.351175439

TABLE III
EXECUTION TIME AND SPEEDUP OF THE SERIAL AND DISTRIBUTED PARALLEL VERSIONS OF PCA WITH DATASET1

	Partition size (MB)	Initialization		Total (s)	Speedup (x)
		Time (s)	Percentage (%)		
Serial	–	1.02	0.37	275.49	–
Parallel(1)	22.46	4.00	44.44	9.00	30.61
Parallel(2)	11.32	4.33	59.09	7.33	37.57
Parallel(4)	5.66	4.67	73.68	6.33	43.50
Parallel(8)	2.80	5.00	80.65	6.20	44.43

of PCA. Tables I and II report a quantitative comparison based on the first four largest eigenvalues and the last four smallest eigenvalues obtained by the three PCA versions.

From Tables I and II, we can conclude that the proposed serial and distributed parallel versions of PCA produce exactly the same eigenvalues, with very slight differences with regards to the MATLAB version.

C. Computational Performance Evaluation

In this section, we assess the computational performance of the distributed parallel implementation on the 10 datasets described in Section IV-A.

First, the experiments were performed on Dataset1 and Dataset3 to test the acceleration effect of the distributed parallel implementation. Speedup comparisons between the serial and parallel versions are given in Tables III and IV, where Parallel(x) denotes the parallel version executed on a distributed platform consisting of one master node and x slaves, with $x = 1, 2, 4, 8$. It is worth mentioning that we performed spatial-domain partitioning, thus every pixel vector (i.e., spectral signature) is stored in the same node. The partition sizes (the number of pixels in each partition) of every execution are listed in Tables III and IV, but the size of the last partition might be smaller.

TABLE IV
EXECUTION TIME AND SPEEDUP OF THE SERIAL AND DISTRIBUTED PARALLEL VERSIONS OF PCA WITH DATASET3

	Partition size (MB)	Initialization		Total (s)	Speedup (x)
		Time (s)	Percentage (%)		
Serial	–	9.44	0.35	2728.08	–
Parallel(1)	224.25	11.00	17.52	63.33	43.08
Parallel(2)	112.13	9.67	20.01	37.00	73.73
Parallel(4)	56.06	9.00	40.01	22.33	122.15
Parallel(8)	28.03	6.00	46.44	13.00	209.85

TABLE V
EXECUTION TIME OF THE DISTRIBUTED VERSION WITH DIFFERENT NUMBERS OF PARTITIONS ON DATASET3

Partition size (MB)	Number of partitions	Initialization (s)	Eigen-decomposition (s)	Projection (s)	Total (s)
140.16	8	12.33	18.00	15.00	45.33
70.0	16	8.00	9.00	7.67	24.67
46.72	24	7.00	6.33	5.00	18.33
35.04	32	7.00	5.00	4.00	16.00
28.03	**40**	**6.00**	**4.00**	**3.00**	**13.00**
23.36	48	6.33	7.33	5.00	18.67
20.02	56	7.00	4.00	7.33	18.33

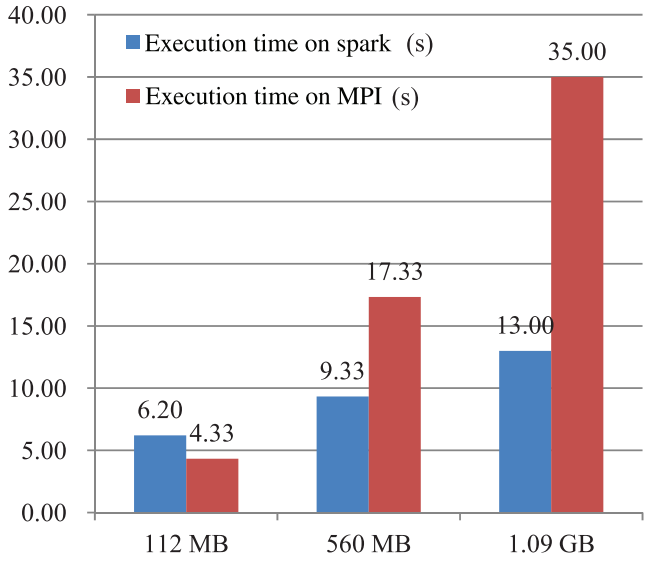
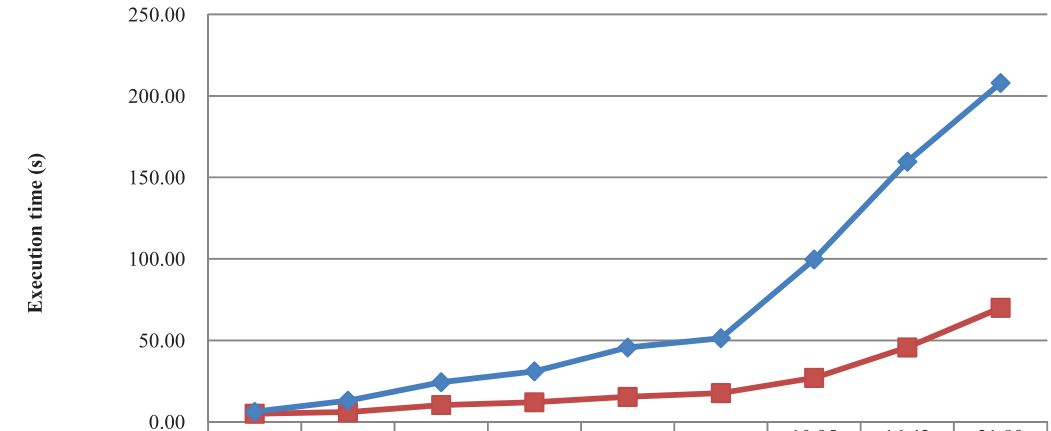


Fig. 5. Execution time comparison between the Spark version and the MPI version.

Since the size of the partitions has a relevant impact on the execution of the distributed parallel implementation, we empirically set it to reach more than 80% CPU utilization of every slave node during execution. To further demonstrate the impact of partition size on the parallel implementation, let us take the execution of Parallel(8) on Dataset3 as an example. Different



	112 MB	1.09 GB	2.19 GB	3.28 GB	4.38 GB	5.47 GB	10.95 GB	16.42 GB	21.89 GB
Initialization (s)	5.00	6.00	10.33	12.00	15.33	17.67	27.00	45.67	70.00
Total (s)	6.20	13.00	24.33	31.00	45.66	51.33	99.66	159.67	208.00

Fig. 6. Execution time of the distributed version with different datasets.

numbers of partitions (8, 16, 24, 32, 40, 48, and 56) are chosen for tests. Table V shows that 40 partitions represent the best choice on the current platform consisting of one master node and eight slaves.

It should also be noted that the initialization time comprises the reading of the data, launching Apache Spark, and registering the corresponding nodes. From Tables III and IV, we can conclude that the proposed parallel and distributed implementation significantly accelerates the PCA computation on both datasets. However, for the small Dataset1, using more nodes cannot lead to higher speedup. This is expected, as in this case we need more time to initialize, which represents a significant portion of the total execution in the case of the small dataset. Fortunately, since the initialization only takes a small percentage of the total execution time in the case of Dataset3, the speedup increases with the number of nodes as shown on Table IV. The distributed version achieved a significant speedup of nearly 210× on the considered platform (with one master and eight slaves).

In order to illustrate the superiority of our proposed framework based on Spark, a comparison with the common cluster programming approach message passing interface (MPI)[7] is performed on Dataset1, Dataset2, and Dataset3. The MPI-like Java library MPJ Express,[8] which is an open source Java message passing library that allows application developers to write and execute parallel applications for compute clusters, is installed on the previous platform and is used to implement the MPI version of PCA according to [20] based on master–slave mode. Fig. 5 shows that the parallel and distributed version based on Spark is more efficient than the MPI version when processing the larger datasets. Further analysis reveals that the bottleneck of the MPI version is data transfer. In our MPI version of PCA, the original hyperspectral dataset is stored on the master node. During the execution, the master needs to partition the dataset into several blocks, distribute them to the slave nodes, and collect the results from these slave nodes after the local computations have been completed. If we take the MPI execution for Dataset2 and Dataset3 as an example, we can observe that the data distribution and results collection steps take more than 15 s for Dataset2 and 25 s for Dataset3. This means that our MPI version of PCA is not compute-bound when dealing with large hyperspectral datasets. On the other side, our Spark implementation greatly benefits from the dynamical storage of HDFS, which is specifically designed to optimize the distributed processing and management of very large files. As a result, the Spark version is compute-bound, even for large hyperspectral datasets. Although we are aware that our MPI version could be further optimized, we have found that the use of HDFS greatly simplifies the implementation leading to very good performance in our context.

To evaluate the efficiency of our proposed distributed parallel implementation of PCA on larger datasets, we also conducted experiments with Dataset4, Dataset5, Dataset6, Dataset7, Dataset8, Dataset9, and Dataset10. In these cases, the serial version cannot run on one node due to the limitation of its memory resources. The results obtained using one master node and eight slaves are given in Fig. 6. From this plot, we can conclude that our proposed distributed parallel implementation scales almost linearly with the size of the dataset. Moreover, the parallel implementation is compute-bound for large datasets (e.g., only 33.65% of the time is consumed by initialization for the execution on Dataset10). This is an important consideration, as it indicates that the proposed implementation scales better as the data size becomes larger. For a hyperspectral dataset of size larger than 21 GB, our approach needs less than 210 s to reduce its dimensionality and store efficiently the data in the system. This opens new avenues to exploit cloud computing technology in hyperspectral imaging problems.

Furthermore, an experimental comparison was performed on a Hadoop equipped platform, which has one NameNode and eight DataNodes. The NameNode is a virtual machine created on the host with an Intel Xeon E5630 CPU at 2.53 GHz with eight cores by the Vmware vSphere. The DataNodes are implemented by eight virtual machines created based on the virtualization of a four-blade IBM BladeCenter HX5 with 48 cores by Vmware vSphere. Both NameNode and DataNodes

[7][Online]. Available: http://www.mcs.anl.gov/research/projects/mpi/
[8][Online]. Available: http://mpj-express.org/

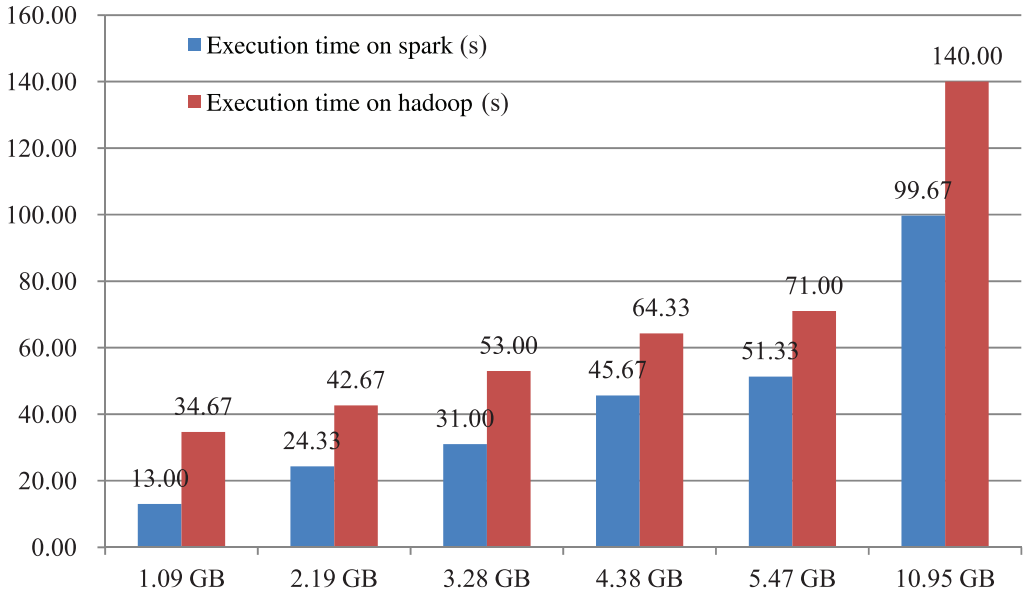


Fig. 7. Execution time comparison between the Spark platform and the Hadoop platform.

are installed with Ubuntu 12.04, Hadoop 1.2.1, and Java 1.6.45. Experimental results demonstrate that the two distributed parallel versions of PCA algorithm produce exactly the same results, and the only difference between them is the computing performance, as shown graphically in Fig. 7. It proves that, benefit from in-memory cluster computing, the Spark platform provides faster data processing than the Hadoop platform.

V. Conclusion and Future Research Lines

In this paper, we have discussed the possibility of exploiting cloud computing architectures for parallel and distributed dimensionality reduction and storing of remotely sensed hyperspectral datasets in large data repositories. As a case study, we have presented a cloud computing implementation of the PCA algorithm on Spark platform. Our experimental results show the effectiveness of the proposed parallel and distributed implementation, not only in terms of accuracy but also in terms of computational performance. The proposed implementation achieved significant speedups when compared to the serial version, which is compelling not only for parallel hyperspectral computing but also for distributed processing of the data, which has not been as widely exploited in previous works. As future work, we will implement other dimensionality reduction and classification/unmixing algorithms using cloud computing infrastructures.

References

[1] A. Plaza, J. M. Bioucas-Dias, A. Simic, and W. J. Blackwell, "Foreword to the special issue on hyperspectral image and signal processing," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 5, no. 2, pp. 347–353, Apr. 2012.

[2] A. Plaza, P. Martinez, J. Plaza, and R. Perez, "Dimensionality reduction and classification of hyperspectral image data using sequences of extended morphological transformations," *IEEE Trans. Geosci. Remote Sens.*, vol. 43, no. 3, pp. 466–479, Mar. 2005.

[3] C. I. Chang, *Hyperspectral Data Processing: Algorithm Design and Analysis*, 1st ed. Hoboken, NJ, USA: Wiley, Mar. 2013.

[4] I. T. Jolliffe, *Principal Component Analysis*, 2nd ed. New York, NY, USA: Springer, 2002.

[5] C. Rodarmel and J. Shan, "Principal component analysis for hyperspectral image classification," *Surv. Land Inf. Sci.*, vol. 62, no. 2, pp. 115–122, 2002.

[6] S. Sanchez, R. Ramalho, L. Sousa, and A. Plaza, "Real-time implementation of remotely sensed hyperspectral image unmixing on GPUs," *J. Real-Time Image Proc.*, vol. 10, no. 3, pp. 469–483, Sep. 2015.

[7] J. Zhang and K. H. Lim, "Implementation of a covariance-based principal component analysis algorithm for hyperspectral imaging applications with multi-threading in both CPU and GPU," in *Proc. IEEE Int. Geosci. Remote Sens. Symp. (IGARSS)*, Jul. 2012, pp. 4264–4266.

[8] M. Andrecut, "Parallel GPU implementation of iterative PCA algorithms," Arxiv preprint arXiv:0811.1081, Nov. 2008.

[9] R. Green *et al.*, "Imaging spectroscopy and the airborne visible/infrared imaging spectrometer (AVIRIS)," *Remote Sens. Environ.*, vol. 65, no. 3, pp. 227–248, Sep. 1998.

[10] A. Plaza, D. Valencia, J. Plaza, and P. Martinez, "Commodity cluster-based parallel processing of hyperspectral imagery," *J. Parallel Distrib. Comput.*, vol. 66, no. 3, pp. 345–358, 2006.

[11] G. Aloisio and M. Cafaro, "A dynamic earth observation system," *Parallel Comput.*, vol. 29, no. 10, pp. 1357–1362, Oct. 2003.

[12] D. Gorgan, V. Bacu, T. Stefanut, D. Rodila, and D. Mihon, "Grid based satellite image processing platform for earth observation application development," in *Proc. IEEE Int. Workshop Intell. Data Acquisition Adv. Comput. Syst.: Technol. Appl. (IDAACS'09)*, Piscataway, NJ, USA, Sep. 2009, pp. 21–23.

[13] Z. Chen, N. Chen, C. Yang, and L. Di, "Cloud computing enabled web processing service for earth observation data processing," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 5, no. 6, pp. 1637–1649, Dec. 2012.

[14] K. Stanoevska-Slabeva, T. Wozniak, and S. Ristol, *Grid and Cloud Computing: A Business Prespective on Technology and Applications*. New York, NY, USA: Springer, 2010.

[15] J. Venner, *Pro Hadoop*. Berlin, Germany: Springer-Verlag, 2009.

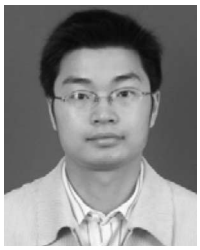
[16] The Apache Software Foundation. (2008). *MapReduce Tutorial* [Online]. Available: https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.pdf

[17] M. Zaharia, "An architecture for fast and general data processing on large clusters," Elect. Eng. Comput. Sci., Univ. California at Berkeley, Berkeley, CA, USA, Tech. Rep. UCB/EECS-2014-12, Feb. 2014.

[18] M. Zaharia *et al.*, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *Proc. 9th USENIX Symp. Netw. Syst. Des. Implement.*, San Jose, CA, USA, Apr. 2012, pp. 1–14.

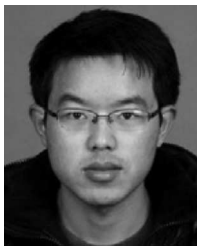
[19] J. A. Richards and X. Jia, *Remote Sensing Digital Image Analysis*, 4th ed. Berlin, Germany: Springer-Verlag, 2006.

[20] A. Shafi, B. Carpenter, and M. Baker, "Nested parallelism for multi-core HPC systems using Java," *J. Parallel Distrib. Comput.*, vol. 69, no. 6, pp. 532–545, Jun. 2009.



Zebin Wu (M'13) was born in Zhejiang, China, in 1981. He received the B.Sc. and Ph.D. degrees in computer science and technology from Nanjing University of Science and Technology, Nanjing, China, in 2003 and 2007, respectively.

He is currently an Associate Professor and Doctoral Supervisor with the School of Computer Science and Engineering, Nanjing University of Science and Technology, and also a Visiting Scholar at the Hyperspectral Computing Laboratory, Department of Technology of Computers and Communications, Escuela Politécnica, University of Extremadura, Badajoz, Spain. His research interests include hyperspectral image processing, high-performance computing, and computer simulation.



Yonglong Li was born in Jiangsu, China, in 1990. He received the B.Sc. degree in electronics engineering and computer science from the School of Network Engineering, Yancheng Teachers University, Yancheng, China, in 2013. He is currently pursuing the M.S. degree in computer science and engineering at Nanjing University of Science and Technology, Nanjing, China.

His research interests include hyperspectral image unmixing and distributed parallel computing.

Antonio Plaza (M'05–SM'07–F'15) was born in Caceres, Spain, in 1975. He received the computer engineer degree in 1997, the M.Sc. degree in 1999, and the Ph.D. degree in 2002, all in computer engineering.

He is an Associate Professor (with accreditation for Full Professor) with the Department of Technology of Computers and Communications, University of Extremadura, Badajoz, Spain, where he is the Head of the Hyperspectral Computing Laboratory (HyperComp). He has been the Advisor of 12 Ph.D. dissertations and more than 30 M.Sc. dissertations. He was the Coordinator of the Hyperspectral Imaging Network, a European project with total funding of €2.8 million. He has authored more than 400 publications, including 130 JCR journal papers (82 in IEEE journals), 20 book chapters, and more than 240 peer-reviewed conference proceeding papers (94 in IEEE conferences). He has edited a book on *High-Performance Computing in Remote Sensing* (CRC press/Taylor & Francis) (the first book on this topic in the published literature) and guest edited eight special issues on hyperspectral remote sensing for different journals. His research interests include remotely sensed hyperspectral image analysis and efficient implementations of large-scale scientific problems on high-performance computing architectures.

Dr. Plaza served as an Associate Editor for the IEEE TRANSACTIONS ON GEOSCIENCE AND REMOTE SENSING from 2007 to 2012. He is also an Associate Editor for IEEE ACCESS and a member of the Editorial Board of the IEEE GEOSCIENCE AND REMOTE SENSING NEWSLETTER (2011–2012) and the *IEEE Geoscience and Remote Sensing Magazine* (2013). He was also a member of the Steering Committee of the IEEE JOURNAL OF SELECTED TOPICS IN APPLIED EARTH OBSERVATIONS AND REMOTE SENSING (JSTARS). He is currently an Associate Editor for the *Journal of Real-Time Image Processing*. He is a coauthor of the 2011 Best Student Paper at the IEEE International Conference on Space Technology. He served as the Director of Education Activities for the IEEE Geoscience and Remote Sensing Society (GRSS), in 2011–2012, and has been serving as President of the Spanish Chapter of IEEE GRSS since 2012. He has served as a Proposal Evaluator for the European Commission (Marie Curie Actions, Engineering Panel), the European Space Agency, the Belgium Science Policy, the Israel Science Foundation, and the Spanish Ministry of Science and Innovation. He has participated in the Tenure Track Selection Committee of different Universities in Italy, Spain, and Australia. He has reviewed more than 500 articles for over 50 different journals. He is also currently serving as the Editor-in-Chief of the IEEE TRANSACTIONS ON GEOSCIENCE AND REMOTE SENSING JOURNAL. He was the recipient of the recognition of Best Reviewers of the 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 2013 Best Paper Award of the JSTARS journal, the most highly cited paper (2005–2010) in the *Journal of Parallel and Distributed Computing*, the 2008 Best Paper award at the IEEE Symposium on Signal Processing and Information Technology, and the Best Ph.D. Dissertation Award at University of Extremadura, in 2002. Four of his students have received the Best Ph.D. Dissertation Award at the University of Extremadura, and one of his students has received the Best Ph.D. Dissertation Award at Complutense University of Madrid, Madrid, Spain.

Jun Li (M'13) received the B.S. degree in geographic information systems from Hunan Normal University, Changsha, China, the M.E. degree in remote sensing from Peking University, Beijing, China,, and the Ph.D. degree in electrical engineering from the Instituto de Telecomunicações, Instituto Superior Técnico (IST), Universidade Técnica de Lisboa, Lisbon, Portugal, in 2004, 2007, and 2011, respectively.

From 2007 to 2011, she was a Marie Curie Research Fellow with the Departamento de Engenharia Electrotécnica e de Computadores and the Instituto de Telecomunicações, IST, Universidade Técnica de Lisboa, in the framework of the European Doctorate for Signal Processing (SIGNAL). She has also been actively involved in the Hyperspectral Imaging Network, a Marie Curie Research Training Network involving 15 partners in 12 countries and intended to foster research, training, and cooperation on hyperspectral imaging at the European level. Since 2011, she has been a Postdoctoral Researcher with the Hyperspectral Computing Laboratory, Department of Technology of Computers and Communications, Escuela Politécnica, University of Extremadura, Caceres, Spain. Her research interests include hyperspectral image classification and segmentation, spectral unmixing, signal processing, and remote sensing.

Dr. Li has been a Reviewer of several journals, including the IEEE TRANSACTIONS ON GEOSCIENCE AND REMOTE SENSING, IEEE GEOSCIENCE AND REMOTE SENSING LETTERS, *Pattern Recognition*, *Optical Engineering*, *Journal of Applied Remote Sensing*, and *Inverse Problems and Imaging*. She received the 2012 Best Reviewer Award of the IEEE JOURNAL OF SELECTED TOPICS IN APPLIED EARTH OBSERVATIONS AND REMOTE SENSING.

Fu Xiao (M'12) received the Ph.D. Degree in computer science and technology from Nanjing University of Science and Technology, Nanjing, China, in 2007.

He is currently a Professor and Doctoral Supervisor with the School of Computer, Nanjing University of Posts and Telecommunications, Nanjing, China. His research interest includes wireless sensor networks.

Zhihui Wei was born in Jiangsu, China, in 1963. He received the B.Sc. degree in applied mathematics, the M.Sc. degree in applied mathematics, and the Ph.D. degree in communication and information system from South East University, Nanjing, China, in 1983, 1986, and 2003, respectively.

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