sensors

MDPI

Article

A Distributed Parallel Algorithm Based on Low-Rank and Sparse Representation for Anomaly Detection in Hyperspectral Images

Yi Zhang [1], Zebin Wu [1,2,*], Jin Sun [1], Yan Zhang [2], Yaoqin Zhu [1], Jun Liu [1] and Qitao Zang [1] and Antonio Plaza [3]

1 School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing 210094, China; yzhang@njust.edu.cn (Y.Z.); sunj@njust.edu.cn (J.S.); zhuyaoqin@163.com (Y.Z.); liuj302@126.com (J.L.); zangqitao@163.com (Q.Z.)
2 Lianyungang E-Port Information Development Co. Ltd., Lianyungang 222042, China; yzhang@lygeport.gov.cn
3 Hyperspectral Computing Laboratory, Department of Technology of Computers and Communications, University of Extremadura, 10003 Caceres, Spain; aplaza@unex.es
* Correspondence: wuzb@njust.edu.cn; Tel.: +86-25-8431-8108

Received: 4 September 2018; Accepted: 23 October 2018; Published: 25 October 2018

Abstract: Anomaly detection aims to separate anomalous pixels from the background, and has become an important application of remotely sensed hyperspectral image processing. Anomaly detection methods based on low-rank and sparse representation (LRASR) can accurately detect anomalous pixels. However, with the significant volume increase of hyperspectral image repositories, such techniques consume a significant amount of time (mainly due to the massive amount of matrix computations involved). In this paper, we propose a novel distributed parallel algorithm (DPA) by redesigning key operators of LRASR in terms of MapReduce model to accelerate LRASR on cloud computing architectures. Independent computation operators are explored and executed in parallel on Spark. Specifically, we reconstitute the hyperspectral images in an appropriate format for efficient DPA processing, design the optimized storage strategy, and develop a pre-merge mechanism to reduce data transmission. Besides, a repartitioning policy is also proposed to improve DPA's efficiency. Our experimental results demonstrate that the newly developed DPA achieves very high speedups when accelerating LRASR, in addition to maintaining similar accuracies. Moreover, our proposed DPA is shown to be scalable with the number of computing nodes and capable of processing big hyperspectral images involving massive amounts of data.

Keywords: hyperspectral images; anomaly detection; distributed and parallel computing; apache spark; clouds

1. Introduction

During recent years, hyperspectral remote sensing has been widely used in various fields of Earth observation and space exploration [1–6]. Hyperspectral sensors are able to simultaneously measure hundreds of contiguous spectral bands with fine spectral resolution [7]. In hyperspectral image (HSI) cubes, each pixel can be represented by a vector whose entries correspond to the spectral bands, providing a representative spectral signature of the underlying materials within the pixel [8–13]. This type of imagery offers detailed ground information in both the spectral and spatial domains. Therefore, HSIs have been popular in remote sensing applications such as urban mapping, environmental monitoring, object identification, and military defence [14–18].

Anomaly detection is one of the most popular and important techniques for distinguishing subtle differences among ground objects by exploiting the hundreds of narrow and nearly continuous bands provided by HSIs [19]. In fact, anomalous targets can be detected in HSIs without prior knowledge, i.e., the background refers to non-target pixels that are predominant in an image compared with the target pixels [20]. In the literature, many methods were proposed to improve the efficiency and effectiveness of hyperspectral anomaly detection [20–29]. An anomaly detection method based on low-rank and sparse representation (LRASR) was proposed in [30], which achieves robust anomaly detection with high accuracy, by taking full advantages of the correlations of all the pixels in HSIs and global information. However, LRASR involves a massive amount of matrix computations that are both computation-intensive and data-intensive, and becomes inapplicable to process big HSIs, since a single machine, even with powerful GPUs (note that GPUs have already been used to accelerate some serial algorithms [31–33]), can hardly provide sufficient computational resources. For instance, LRASR consumes 1.1 GB of memory when processing a small HSI with the size of 8000×162. However, it could not process a big HSI with the size of $800{,}000 \times 162$ on a single machine with 12 GB of memory, since all the memory is exhausted and the execution of LRASR is terminated with an "out of memory" exception, accordingly.

Distributed computing technologies are highly demanded for efficient processing of big HSIs. Cloud computing offers the potential to tackle massive data processing workloads by means of its distributed and parallel architecture. With the continuously increasing demand for massive data processing in HSI applications, there have been several efforts in the literature oriented towards exploiting cloud computing infrastructure for processing big HSIs [34–40]. Several parallel anomaly detection algorithms were proposed based on a master and slave model in [38–40] and executed on commodity clusters (or called public clouds). Based on the MapReduce parallel model [41–45], the approaches [34–37] use the Hadoop Distributed File System (HDFS) to implement distributed storage, and use Apache Spark [46] to execute HSI applications in parallel. Actually, Spark is an effective and state-of-the-art platform for parallel computation designed to efficiently deal with iterated computational procedures in terms of the MapReduce parallel model [47]. Spark employs HDFS to store the processed data in terms of blocks. The main abstraction of Spark is given by Resilient Distributed Datasets (RDDs), which represent a distributed, immutable, and fault-tolerant collection of objects [48]. When a task is executed on a computation node (called node for simplicity in this paper), all the data sets required by this need to be loaded into an RDD to construct a partition. In other words, a task corresponds to a partition. Meanwhile, users can also explicitly cache an RDD in memory across machines, and reuse it in multiple MapReduce parallel operations.

In this paper, we propose a novel distributed parallel algorithm (DPA) based on low-rank and sparse representations for hyperspectral anomaly detection. The proposed algorithm intends to accelerate LRASR on clouds. Independent computation operators in LRASR are first explored and executed in parallel on Spark. The hyperspectral data are reconstituted in an appropriate format for efficient DPA processing. The optimized storage is designed and a new pre-merge mechanism is developed for the purpose of reducing data transmission. Mean while, a new repartitioning policy is designed to improve DPA's efficiency. Our experimental results show that the proposed DPA is able to achieve not only similarly good accuracies as LRASR but also very high speedups. Furthermore, DPA is verified to be scalable with the number of nodes and capable of processing big HSIs involving a massive amount of data.

The reminder of this paper is organized as follows. Section 2 gives a brief introduction to LRASR. Section 3 describes the newly proposed DPA, followed by the a presentation of experimental results in Section 4. Section 5 concludes with some remarks and hints at plausible future research lines.

2. Anomaly Detection Using Low-Rank and Sparse Representation (LRASR)

In a remotely sensed hyperspectral image, Anomalous pixels should be different from background pixels, while there usually exist strong correlation among background pixels. On this basis, our

assumption is that background pixels can be linearly represented by some other background pixels, meaning that a HSI X can be decomposed into an anomalous part E and a background part $D \times S$, where D is the background dictionary constructed from m background pixels, and S denotes the representation coefficients. In other words, we have $X = DS + E$. The data model of LRASR is shown in Figure 1 [30].

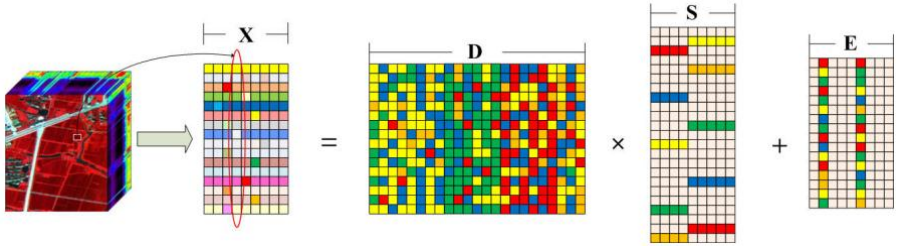

Figure 1. Data model of LRASR.

The math model of LRASR can be described using Equations (1) and (2), where $\|\cdot\|$ denotes the matrix nuclear norm (sum of singular values of a matrix), and $\|\cdot\|_1$ is the l_1 norm of a matrix, i.e., the sum of the absolute value of all entries in the matrix. $\|\cdot\|_{2,1}$ is the $l_{2,1}$ norm defined as the sum of l_2 norm of the column of a matrix. Besides, $\beta > 0$ is a parameter to trade off low rankness and sparsity. $\lambda > 0$ is used to balance the effects of the two parts.

$$min_{S,E}\{\|S\|_* + \beta\|S\|_1 + \lambda\|E\|_{2,1}\} \tag{1}$$

s.t.

$$X = DS + E \tag{2}$$

The algorithm LRASR was developed to solve the aforementioned problem, and its flowchart is given in Figure 2. From this figure, it can be seen that, after reading data from disks to obtain a HSI X, LRASR first employs the K-means algorithm to cluster all pixels, and then uses a dictionary construction method to obtain the background dictionary matrix D based on the obtained K clusters. Afterwards, an alternating direction multiplier method (ADMM) is used to generate the anomaly matrix E. Finally, the obtained anomaly matrix E is written to disks. Details of LRASR are given in Algorithm 1, in which Steps 1–16 denote the K-means algorithm, whereas Steps 17–24 represent the dictionary construction method. Step 25 is the ADMM, which is a complex algorithm and can be seen in [30].

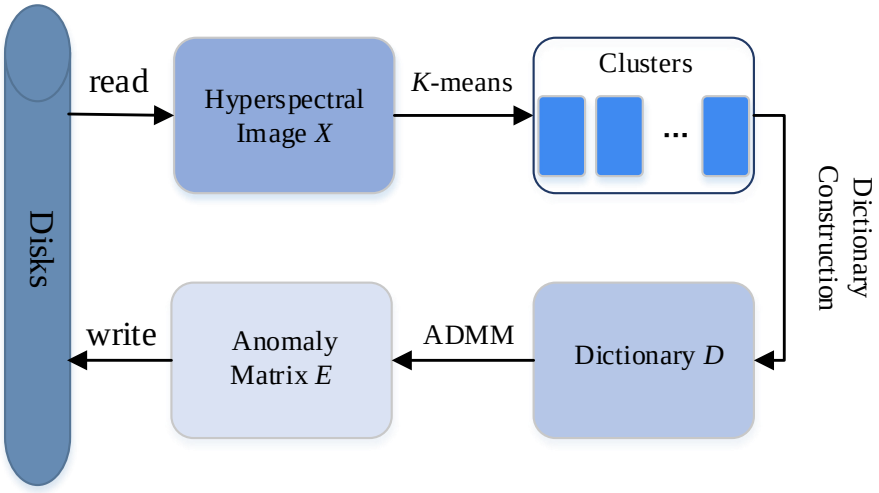

Figure 2. Flowchart of LRASR.

Algorithm 1: LRASR

Input: A HSI X
Output: An anomaly matrix E
1 Select K pixels from X as centers randomly;
2 **while** *(TRUE)* **do**
3 Record the K centers as old centers;
4 **for** *(each pixel x in X)* **do**
5 Calculate the Euclidean distances between x and all the K centers;
6 Assign the pixel x to the cluster having the center that has the shortest distance between it and x;
7 **end**
8 **for** *(each cluster $X^i(i = 1, 2, \cdots, K)$)* **do**
9 Calculate the mean of all the pixels in X^i;
10 Set the center of X^i to be the obtained mean;
11 **end**
12 Calculate the Euclidean distances between the obtained new centers to those old centers respectively;
13 **if** *(the maximum among the obtained K distances is lower than a given threshold)* **then**
14 break;
15 **end**
16 **end**
17 Set the dictionary matrix $D = \emptyset$;
18 **for** *(each cluster $X^i(i = 1, 2, \cdots, K)$)* **do**
19 Calculate its mean value μ and covariance matrix C;
20 Calculate the RX detector of each pixel x in X^i by $RX(x) = (x - \mu)^T C^{-1}(x - \mu)$;
21 Select P pixels $x_1, x_2, \cdots, x_P$ whose RX detectors are lower than those of the other pixels in X^i;
22 Construct $D^i = [x_1, x_2, \cdots, x_P]$;
23 Set $D = D \bigcup D^i$;
24 **end**
25 Use ADMM to generate the anomaly matrix E;
26 **return** E;

3. Distributed Parallel Algorithm (DPA)

Our newly developed DPA first explores the independent computation operators in LRASR and processes them in parallel on multiple nodes provided by Spark. In Algorithm 1, we can see that all the pixels are clustered by the K-means algorithm (Lines 1–16). In other words, the processing operator on each pixel, i.e., the calculation of Euclidean distance between the pixel and all K centers (Line 5) and the assignment of the pixel to a cluster whose center has the shortest distance to the pixel (Line 6), is not dependent on other pixels. Accordingly, the processing operators can be executed in parallel, and we propose a distributed parallel K-means algorithm to achieve this goal. Similarly, in the dictionary construction method (Lines 17–24), the processing operator on each cluster, including the calculation of the cluster's mean value and covariance matrix (Line 19), the calculation of each pixel's RX detector in the cluster (Line 20), the selection of P pixels from the cluster (Line 21) and the construction of D^i (Line 22), is independent from other clusters. Accordingly, we establish a distributed parallel dictionary construction method, in which a new repartition policy is developed to use K nodes for the purpose of executing the processing operators of clusters in parallel. However, there is no independent computation operator in ADMM (details can be seen in [30]), since ADMM includes a loop in which each iteration depends on the previous one. Nevertheless, we construct a distributed parallel ADMM

in which ADMM is conducted on each partition to generate a partial anomaly matrix, and a complete E can be obtained by composing all the partial matrices. The flowchart of our DPA is illustrated in Figure 3. Before detailing the three distributed parallel methods, we need to first describe data organization and storage optimization methods, which help the proposed DPA to reduce the data transmission and improve the efficiency.

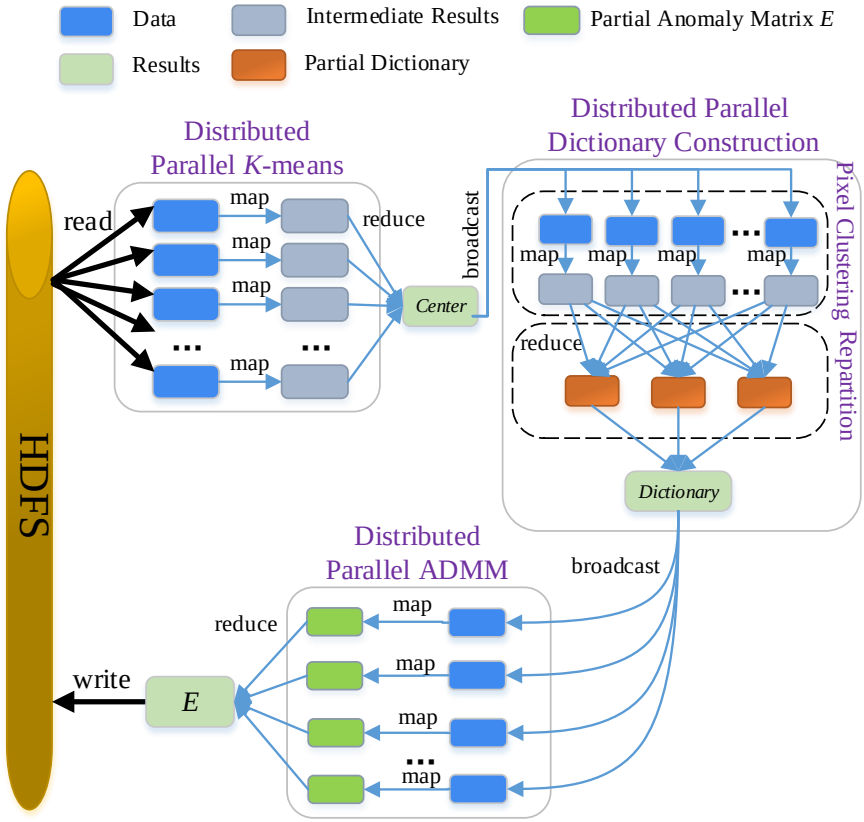

Figure 3. Flowchart of DPA.

3.1. Data Organization and Storage Optimization Methods

Band interleaved by line (BIL), band interleaved by pixel (BIP) and band sequential (BSQ) are three common formats for arranging HSIs' data. BSQ stores bands sequentially, whereas BIL and BIP store bands in terms of lines and pixels, respectively. Being similar as LRASR, the proposed DPA processes pixels one by one. Accordingly, BIP is selected to organize HSIs' data.

As the proposed DPA is executed in parallel on Spark, we need to upload HSIs' data to HDFS storing data in blocks, whose size can be set according to different conditions and its default value is 64 MB. Given a program that can be divided into m computing tasks executed on m nodes in parallel (i.e., the level of parallelism is m), if the data required by one task are stored in multiple blocks on multiple nodes, all these blocks should be transferred to the node where this task is processed. Obviously, if a task's required data are included in one block and the task is scheduled to the node storing the block, no data transmission is needed. Fortunately, Spark's scheduler always targets an "optimized" assignment, i.e., assigning a task to a node where the task's required data reside. Thus, a key aspect becomes how to make a task's required data included in one block. This problem can be easily solved by setting the size of blocks to be d/m, where d is the total size of HSIs' data. Note that for some special situations (e.g., the target node is extremely busy), Spark's scheduler may fail to achieve the "optimized" assignment. When this failure occurs, data transmission is still needed. However, since the failure occurrence is in a relatively low probability, our proposed optimization method by determining the block size can still reduce data transmission.

From Figure 3, we can see that the HSIs' data read from HDFS are used by all the three distributed parallel methods. For the purpose of fast data loading, these HSIs' data are always maintained in memory after they are first read from HDFS.

3.2. Distributed Parallel K-Means Algorithm

The *K*-means algorithm shown by Lines 1–16 in Algorithm 1 contains three steps, which are described as follows: (1) calculate the Euclidean distances between each pixel and all the *K* centers (Line 5), and assign each pixel to a cluster with the center having the shortest distance to the pixel (Line 6); (2) for each cluster, obtain the mean of all the pixels in this cluster and set the cluster's center to be the yielded mean (Lines 8–11); (3) check whether the termination criterion of the *K*-means algorithms is met (Lines 12–15). As mentioned before, the first step can be executed in parallel and is accordingly implemented as the Map method. Accordingly, as the second step needs all the pixels in each cluster to update each cluster's center, and the third step needs all clusters' new centers, both the two steps are implemented as the Reduce method. The algorithm outputs a center that is a set containing all obtained clusters' centers. However, as shown in Figure 4, pixels belonging to the same cluster may appear in different nodes and need to be transferred to the Driver (i.e., a node where the Reduce method is executed). Alternatively, there would be much data transmission.

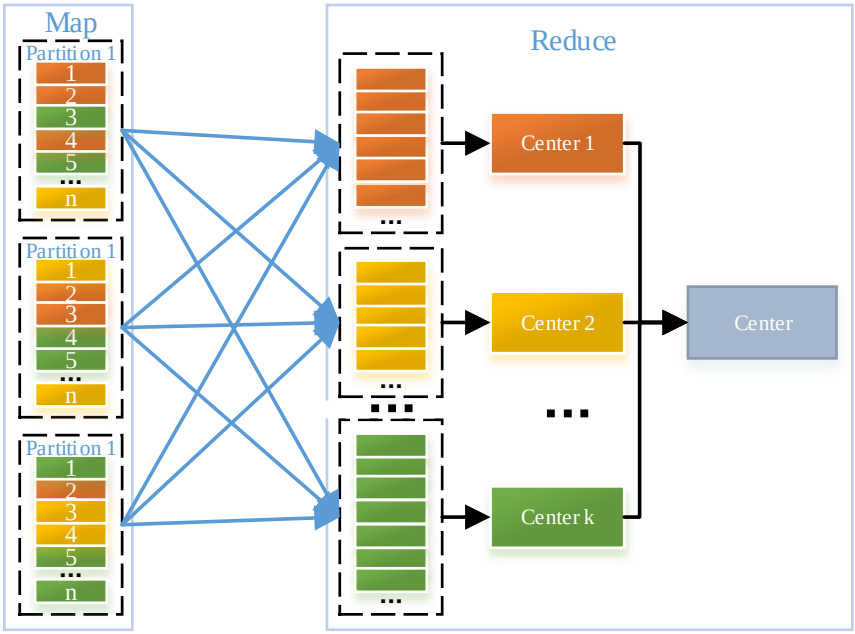

Figure 4. Data transmission before using the pre-merge mechanism.

A pre-merge mechanism is proposed to reduce the data transmission in our context. Before the data transmission, local pixels on each node are clustered using current clusters' centers and then the sums of the pixels in each cluster are calculated. Rather than pixels, the obtained sums, the number of pixels involved to generate these sums and the corresponding clusters' IDs that are necessarily required by the Reduce method are transferred from nodes where map methods are executed to the Driver. After all the map methods on multiple nodes are finished, the Reduce method on the Driver is able to calculate the mean of all the pixels in each cluster with these received data and update each cluster's center. Afterwards, the Reduce method can check whether the termination criterion is met. Obviously, as the amount of transferred data (including the obtained sums, the number of pixels involved in generating these sums and the corresponding clusters' IDs) is much lower than that of all the pixels, the pre-merge mechanism is able to reduce the data transmission. To clarify the pre-merge mechanism, we illustrate the data transmission after using the pre-merge mechanism in

Figure 5. By comparing Figures 4 and 5, we can observe that the pre-merge mechanism significantly contributes to the reduction of data transmission.

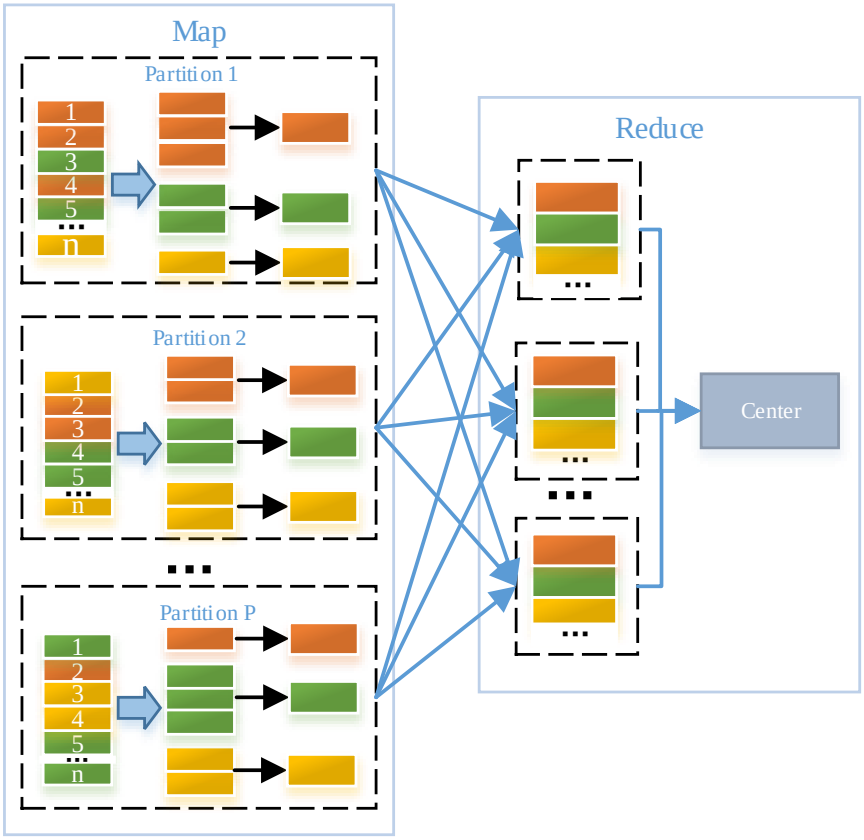

Figure 5. Data transmission after using the pre-merge mechanism.

3.3. Distributed Parallel Dictionary Construction

According to LRASR, the background dictionary is constructed by composing the partial dictionary of each cluster obtained by the *K*-means algorithm. For each cluster, *P* pixels (whose *RX* scores are lower than those of the other pixels in the same cluster) are selected to generate the partial dictionary. In other words, the dictionary construction requires all the pixels of each cluster. However, the proposed distributed parallel *K*-means algorithm generates the *K* clusters without recording those pixels of each cluster. The reason is that, if we want to record pixels of each cluster, we need to write these pixels and their corresponding clusters' IDs to RDD in every iteration. This procedure tries to write data into memory first, and then to disks (if there is not sufficient memory). As a result, this writing procedure consumes much memory and is time-consuming if insufficient memory exists, since writing data to disks is inefficient. To address this issue, we use a parallel pixel clustering operator implemented as the Map method (i.e., the pixel clustering phase in distributed parallel dictionary construction shown in Figure 3) which attributes all pixels to the *K* clusters. Afterwards, the partial dictionary construction is implemented as the Reduce method. As aforementioned, since the processing operator on each cluster (including the calculation of the cluster's mean value and covariance matrix, the calculation of each pixel's *RX* detector in the cluster, the selection of *P* pixels from the cluster and the construction of D^i) is independent from other clusters, we propose a repartition policy to employ *K* nodes to execute the processing operators of all the clusters in parallel. This procedure is quite different from the general one using the Driver only. Obviously, this repartition policy is able to improve DPA's efficiency.

3.4. Distributed Parallel ADMM

As mentioned before, there are no independent computation operators in ADMM. Accordingly, we construct a distributed parallel ADMM, in which ADMM is conducted on each partition to generate a partial anomaly matrix, and a complete E can be obtained by composing all those partial matrices. As shown in Figure 3, these two procedures are implemented as the Map and the Reduce methods, respectively.

3.5. Comparison and Analysis

Compared with LRASR using three serial methods (i.e., the K-means algorithm, the dictionary construction method and ADMM), our proposed DPA employs three distributed parallel ones including the distributed parallel K-means algorithm, the distributed parallel dictionary construction method and the distributed parallel ADMM. Obviously, the advantage of our proposed DPA is that independent computation operators can be executed in parallel and HSIs' data can also be processed in parallel.

We now analyze both the algorithms' complexities. As our proposed DPA is derived from LRASR, we first calculate the complexity of LRASR and that of DPA can be obtained accordingly. Obviously, the complexity of the K-means algorithm is $O(WKT_1)$, in which W is the total number of pixels, K denotes the number of centers and T_1 represents the total number of loops. The complexity of the dictionary construction method is $O(WP^2)$ where P is the number of pixels selected from each cluster, whereas that of the ADMM is $O(PKT_2W^2)$ where T_2 is the total number of loops. Let m be the number of nodes/data partitions. Accordingly, each node processes $\frac{W}{m}$ pixels. We can thus obtain the complexities of the distributed parallel K-means algorithm, the distributed parallel dictionary construction method and the distributed parallel ADMM are $O(\frac{WKT_1}{m})$, $O(\frac{WP^2}{m})$ and $O(\frac{PKT_2W^2}{m^2})$, respectively. In other words, the complexities of our proposed distributed parallel methods are $\frac{1}{m}$, $\frac{1}{m}$ and $\frac{1}{m^2}$ of those of the serial ones used by LRASR, respectively. Consequently, our proposed DPA is able to accelerate LRASR, remarkably.

4. Experimental Results

To verify the proposed DPA's accuracy and efficiency, we perform four experiments on three Spark clusters (denoted as Spark1, Spark2 and Spark3) and six HSIs (called HSI1-HSI6). In the first two experiments (denoted as Experiment 1 and Experiment 2), HSI1 and HSI2 are processed by LRASR and our proposed DPA on Spark1, respectively. In the third experiment (regarded as Experiment 3), HSI1 is processed by LRASR and DPA on Spark2, separately. We design the experiments in this way to show DPA's accuracy and efficiency when it is employed to process different HSIs on different platforms. Furthermore, in order to verify the DPA's capability for processing big HSIs, we conduct the fourth experiment (denoted as Experiment 4), in which four HSIs (called HSI3-HSI6) with data sizes of 1 GB, 2 GB, 4 GB and 8 GB are generated by jointing HSI1 and used, respectively. HSI3-HSI6 are processed by DPA on Spark3, respectively. Details about the four experiments are summarized in Table 1.

Table 1. Details about the four experiments.

Experiments	HSIs	Platforms
Experiment 1	HSI1	Spark1
Experiment 2	HSI2	Spark1
Experiment 3	HSI1	Spark2
Experiment 4	HSI3-HSI6	Spark3

Spark1 is a computing cluster which consists of 1 Master node and 8 Slave nodes. The Master is a virtual machine equipped with 4 cores (2G HZ) and 15 GB RAM. The Slaves are deployed on 4-blade IBM Blade Center HX5. Every Slave is configured with 6 cores (1.67G HZ) and 15 GB RAM. The Master

and all the Slaves are installed with Ubuntu 16.04, Hadoop-2.7.3, Spark-2.1.1, Java 1.8 and Scala 2.11.6. As Spark considers each core in Slaves as a node, the total number of nodes is 48. In other words, the maximal level of parallelism of Spark1 is 48. Spark 2 includes 1 Master and 3 Slaves with identical configuration: 24 cores (2.3G HZ) and 64G of RAM. The Master and all the Slaves are installed with CentOS 6.6, Hadoop-2.7.3, Spark-2.1.1, Java 1.8 and Scala 2.11.7. The total number of nodes in Spark2 is 72, i.e., the maximal level of parallelism of Spark2 is 72. Spark3 is a very powerful cluster which consists of 1 Master and 6 Slaves. The Master and all the Slaves are virtual machines, each of which is equipped with 24 cores (2.5G HZ) and 242G RAM. The Master and all the Slaves are installed with Ubuntu 16.04, Hadoop-2.7.3, Spark-2.1.1, Java 1.8 and Scala 2.11.6. Accordingly, the total number of nodes in Spark3 is 144, i.e., the maximal level of parallelism of Spark3 is 144.

Both HSI1 and HSI2 were used in [30]. HSI1 was collected by the hyperspectral data collection experiment (HYDICE) obtained from an aircraft platform. HSI1 covers an urban area (i.e., Fort Hood, TX, USA), comprising a vegetation area, a construction area, and several roads including some vehicles. A spectral resolution of 10 nm and a spatial resolution of 1 m are contained in this image. Multiple bands including the low-SNR and water vapor absorption bands (i.e., 1–4, 76, 87, 101–111, 136–153, and 198–210) are removed and 162 bands remain as a result. The entire image illustrated in Figure 6a has a size of 307 × 307 pixels, from which an area in the upper rightmost containing 80 × 100 pixels is selected for our experiments, since the ground truth shows that there are several anomalous targets (i.e., cars and roofs embedded in the different backgrounds) in this chosen area. A false color representation and the ground-truth map are given in Figure 6b,c, respectively.

(**a**) (**b**) (**c**)

Figure 6. The first Hyperspectral Image (HSI1). (**a**) The false color image of the entire image; (**b**) The false color image of the chosen area for detection; (**c**) The ground-truth map of the chosen area [30].

HSI2 was collected by the Airborne Visible/Infrared Imaging Spectrometer (AVIRIS) over San Diego, CA, USA, comprising 224 bands in wavelengths ranging from 370 to 2510 nm. Some bands including low-SNR, water absorption and bad bands (i.e., 1–6, 33–35, 94–97, 107–113, 153–166, and 221–224) are removed. Consequently, 186 bands are left and used in our experiments. The entire image shown in Figure 7a has a size of 400 × 400, from which the up-left area including 100 × 100 pixels is selected to perform experiments. In this chosen area, there are buildings with different roofs, parking aprons with different materials, an airport runway, and a small quantity of vegetation. The airplanes

are the anomalies to be detected. The false color image and the ground-truth map are illustrated in Figure 7b,c, respectively.

(**a**) (**b**) (**c**)

Figure 7. The second Hyperspectral Image (HSI2). (**a**) The false color image of the entire image; (**b**) The false color image of the chosen area for detection; (**c**) The ground-truth map of the chosen area [30].

For the evaluation of accuracies and efficiencies of compared algorithms, AUCs (Area Under Curve) and consumed times of involved algorithms are recorded [49,50]. Larger AUCs indicate higher accuracies. In Experiments 1–3, LRASR is run on a single machine (i.e., the Driver node of each experiment corresponding Spark platform) since it is a serial algorithm, whereas DPA is run under different levels of parallelism evaluated by different numbers of nodes. The consumed times and AUCs of all the three experiments are recorded and shown in Table 2, in which consumed times are evaluated in seconds. On one hand, from Table 2, we can see that in Experiment 1, AUC of LRASR is 0.9181, whereas those of DPA with the numbers of nodes 2, 4, 8, 16 and 32 are 0.9202, 0.9184, 0.9131, 0.9195 and 0.9203, respectively. Meanwhile, we find that in Experiment 2, LRASR yields an AUC with the value of 0.9595, whereas DPA generates AUCs with the values of 0.9596, 0.9601, 0.9609, 0.9616 and 0.9607 when the numbers of nodes are 2, 4, 8, 16 and 32, respectively. Finally, in Experiment 3, LRASR obtains an AUC 0.9184, while DPA achieves AUCs 0.9218, 0.9141, 0.9188, 0.9116 and 0.9184 when the numbers of nodes are 2, 4, 8, 16 and 32, respectively. Please note that these AUCs in Experiment 3 are slightly different from those in Experiment 1. The reason is that the initial K centers are randomly selected from pixels in the K-means algorithm (see Line 1 in Algorithm 1). These results indicate that DPA achieves similarly good AUCs as LRASR, regardless of the level of parallelism.

On the other hand, in Experiment 1, we can see that LRASR consumes 3987 s, while DPA with the numbers of nodes 2, 4, 8, 16 and 32 consumes 1788 s, 955 s, 451 s, 233 s and 116 s. In Experiment 2, LRASR consumes 4957 s, whereas the consumed times of DPA are 2613 s, 1223 s, 586 s, 290 s and 149 s when the numbers of nodes are 2, 4, 8, 16 and 32, respectively. In Experiment 3, the times consumed by LRASR is 3657, while those consumed by DPA are 1699, 842, 418, 232 and 111, corresponding to the numbers of nodes 2, 4, 8, 16 and 32, respectively. These results indicate that our proposed DPA accelerates LRASR considerably in the premise of achieving similarly high accuracies. The reason is that, in our proposed DPA, independent computation operators are executed in parallel and all those data are also be processed in parallel.

Table 2. AUCs and Consumed Times(s) obtained by LRASR and DPA with different numbers of nodes in Experiments 1–3.

Number of Nodes	Times (EX1)	AUC (EX1)	Times (EX2)	AUC (EX2)	Times (EX3)	AUC (EX3)
LRASR	3987	0.9181	4957	0.9595	3657	0.9184
DPA with 2 Nodes	1788	0.9202	2613	0.9596	1699	0.9218
DPA with 4 Nodes	955	0.9184	1223	0.9601	842	0.9141
DPA with 8 Nodes	451	0.9193	586	0.9609	418	0.9188
DPA with 16 Nodes	233	0.9195	290	0.9616	232	0.9116
DPA with 32 Nodes	116	0.9203	149	0.9607	111	0.9184

Based on those aforementioned consumed times, we can further calculate the speedups. The consumed times of LRASR is selected as the baseline. The result is illustrated in Figure 8. We can see that the speedups of DPA in Experiment 1 are 2.23, 4.17, 8.84, 17.11 and 34.37 when the numbers of nodes are 2, 4, 8, 16 and 32, respectively. This result indicates that the speedup of DPA increases linearly with the number of nodes. This conclusion illustrates the good scalability of the proposed DPA with the number of nodes. Meanwhile, the speedups of DPA are 2.23, 4.17, 8.84, 17.11 and 34.37 when the numbers of nodes are 2, 4, 8, 16 and 32, respectively. This result denotes that the speedups are larger than the numbers of nodes used for parallel computing, respectively. The reason is that, according to Section 3.5, the complexity of the serial ADMM is $O(PKT_2W^2)$, whereas that of the distributed parallel ADMM is $O(\frac{PKT_2W^2}{m^2})$. In other words, the times consumed by the distributed parallel ADMM are $\frac{1}{m^2}$ of those consumed by the serial ADMM. Conclusively, the distributed parallel ADMM is able to achieve a remarkably high speedup.

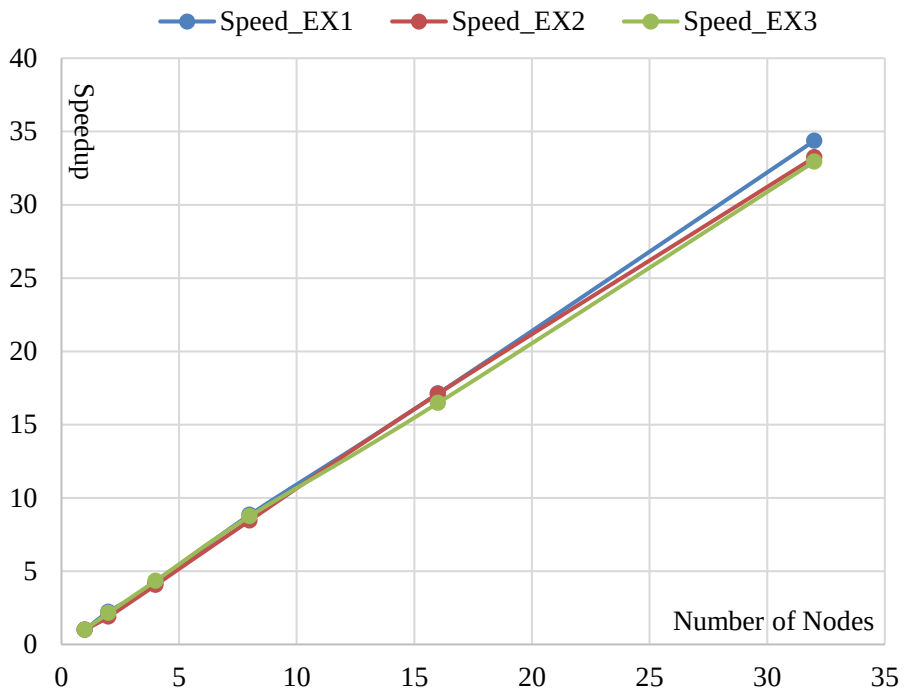

Figure 8. Speedups of DPA with different numbers of nodes in Experiments 1–3.

For Experiment 2, it can be seen that the speedups of DPA are 1.9, 4.05, 8.46, 17.1 and 33.23 with the numbers of nodes 2, 4, 8, 16 and 32, respectively. Except the first speedup 1.9 is a little lower than its corresponding number of nodes 2, the others are larger than their corresponding numbers of nodes. The reason is the same as in Experiment 1. Meanwhile, we can get similar observations, i.e., the speedup of DPA increases linearly with the number of nodes and the distributed parallel ADMM obtains a good speedup. Furthermore, we can conclude that the proposed DPA achieves both

good accuracies and speedups when processing different HSIs. For Experiment 3, we can see that the speedups of DPA are 2.15, 4.34, 8.75, 16.47 and 32.94 with the numbers of nodes 2, 4, 8, 16 and 32, respectively. By comparing this result with that of Experiment 1, we can obtain similar conclusions. In other words, both high accuracies and speedups can be achieved by the proposed DPA running on different platforms.

To show the advantage of our proposed DPA on memory consumption, the memory consumption of LRASR and DPA with different number of nodes in Experiment 1 is illustrated in Figure 9, in which the mean amounts of memory consumed by the three methods (i.e., the *K*-means algorithm, the dictionary construction method and the ADMM) of LRASR and DPA on all involved nodes are recorded. Please note that the three methods of LRASR are serial ones, whereas those of DPA are distributed parallel ones. Figure 9 indicates that the mean amounts of consumed memory of DPA's methods are lower than those of LRASR's methods, without respect of the DPA's parallelism, respectively. Meanwhile, we can also see that the mean amounts of consumed memory of DPA's methods are reduced while the number of nodes increases. These conclusions imply that our proposed DPA could process big HSIs, especially when DPA is executed in high parallelism.

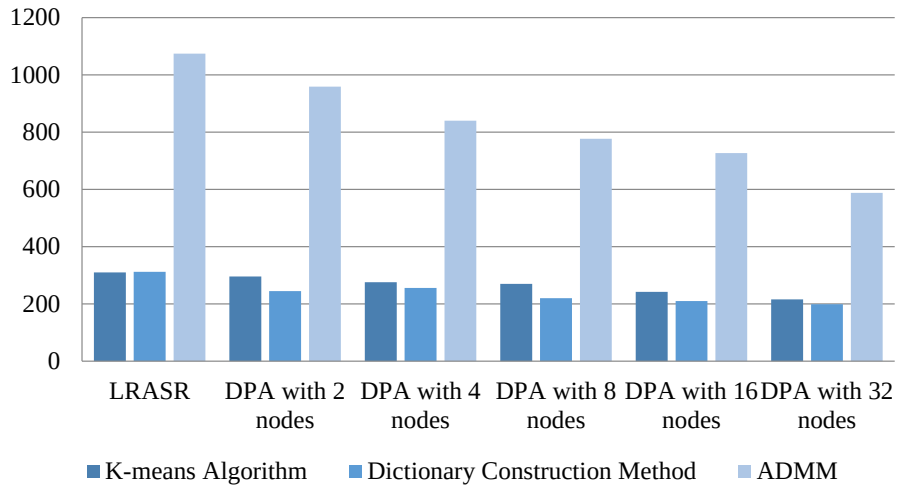

Figure 9. Memory consumption (MB) of LRASR and DPA with different number of nodes in Experiment 1.

To verify the capability of the proposed DPA to process big HSIs with massive amounts of data, we perform Experiment 4, in which the level of parallelism is fixed to a large value 120. The results are shown in Figure 10, which shows that the consumed times are 3603, 8284, 17,367 and 44,421 s for HSI3–HSI6, respectively. The experimental results indicate that, by means of distributed parallel computing, the proposed DPA is able to efficiently process the big HSIs, which cannot be processed by LRASR running on a single computing machine with limited resources.

To show more insights, we compare the results of LRASR and our proposed DPA running on one node. In other words, DPA is running serially and denoted as SDPA, accordingly. Both of the two compared algorithms are executed using HSI1 on the Driver node of Spark1. The experimental results show that SDPA achieves an AUC 0.9197 and consumes 4320 s. On one hand, compared with the LRASR's AUC 0.9181, SDPA's AUC is similarly high. On the other hand, by comparison with the LRASR's consumed times 3987 s, it can be seen that SDPA requires more computation times. The reason is that SDPA is actually a parallel-designed algorithm in terms of the MapReduce model. Accordingly, although it is serially executed on a single node, there will also be much intermediate data generated and written to the shared memory by map methods, and afterwards, these intermediate data will be loaded by the reduce method from the shared memory. These overheads, including data generation and data written/loaded to/from shared memory, reduce the efficiency. However, DPA achieves good speedups when it is executed in parallel on multiple nodes. Consequently, we can conclude that Spark is appropriate to process parallel-designed algorithm in parallel rather than in serial.

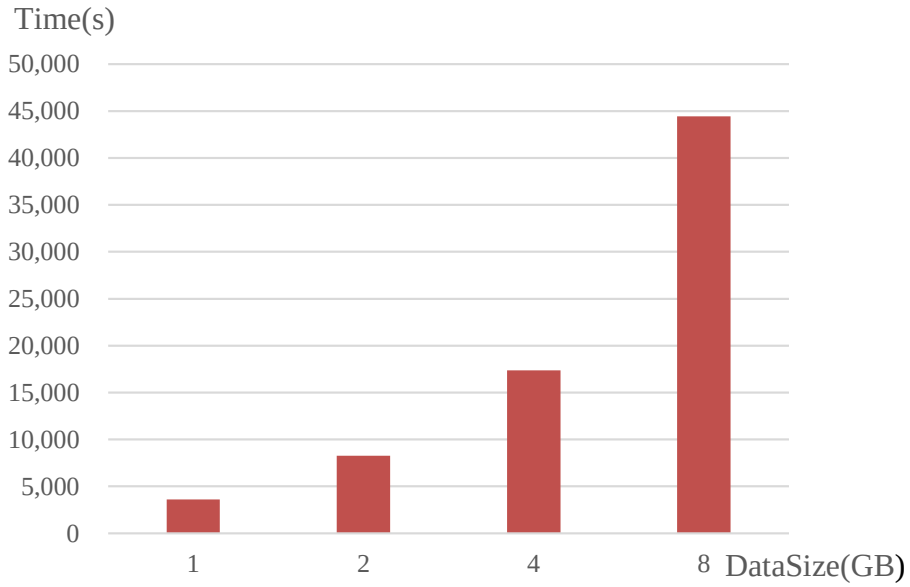

Figure 10. Speedups of DPA processing big HSIs with different data sizes.

Furthermore, we re-implement DPA and execute the obtained algorithm (denoted as HDPA) on Hadoop that is similar as Spark, i.e., both of them are able to execute MapReduce-based applications. The main difference between them is that Spark supports in-memory computation while Hadoop does not. In this experiment, HSI1 is used and Hadoop is deployed on the same cluster as Spark1. The HDPA's results (including AUCs and consumed times) are given in Table 3. By comparing these results with those of DPA in Experiment 1 (seen in Table 2), we can observe that both HDPA and DPA achieve similarly good AUCs, whereas DPA outperforms HDPA in terms of consumed times for all the cases with different number of nodes. This conclusion indicates that Spark is more effective than Hadoop due to its in-memory computation capability.

Table 3. AUCs and Consumed Times(s) obtained by HDPA with different numbers of nodes.

Metrics	2 Nodes	4 Nodes	8 Nodes	16 Nodes	32 Nodes
AUC	0.9216	0.9166	0.9187	0.9141	0.9155
Times	4320	2524	2272	1706	1706

5. Conclusions and Future Work

With the ever growing size and dimensionality of HSIs, it is a challenge to perform anomaly detection in big hyperspectral images on a single computing machine. This paper introduces a novel DPA algorithm to accelerate an LRASR method for hyperspectral anomaly detection on cloud computing architectures. The newly developed algorithm presents several important contributions. First, independent computation operators are explored and executed in parallel on Apache Spark, which is a promising distributed computing platform to process massive data in parallel. In addition, the hyperspectral data are reconstituted in an appropriate format for efficient DPA processing. Meanwhile, a new pre-merge mechanism has been developed for further reducing data transmission, and a new repartitioning policy was used to improve DPA's efficiency. We evaluated the accuracy and efficiency of our newly developed DPA by means of multiple experiments. The experimental results demonstrated that DPA not only obtained similarly high accuracies as LRASR, but also achieved considerably higher speedups. Moreover, DPA was also verified to be scalable with the number of nodes and capable of processing big HSIs with massive amounts of data. These conclusions indicate that the Spark is very appropriate for processing HSIs' applications that are both computation-intensive and data-intensive (e.g., LRASR). As massive

amounts of intermediate data would be generated and transferred among nodes, we need to design some optimized strategies (e.g., our proposed pre-merge mechanism in the distributed parallel *K*-means algorithm) to reduce data transmission. In future work, we will develop cloud implementation of other HSI processing algorithms, including techniques for classification and unmixing.

Author Contributions: Y.Z. (Yi Zhang), Y.Z. (Yan Zhang) and Z.W. conceived and designed the experiments; J.L. and Q.Z. performed the experiments; J.S. and Y.Z. (Yaoqin Zhu) analyzed the data; Y.Z. (Yi Zhang) and Z.W. and A.P. wrote the paper.

Funding: This research was funded by the National Natural Science Foundation of China (NSFC) grants numbers 71501096, 61502234, 61502250, 61471199, 61772274, 61872185 and 61802185, the Jiangsu Provincial Natural Science Foundation of China grants numbers BK20150785, BK20180470 and BK20180018, the Fundamental Research Funds for the Central Universities grants numbers 30916011325 and 30917015104, the Project funded by China Postdoctoral Science Foundation under Grant No. 2015M581801, and MINECO Project TIN2015-63646-C5-5-R.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Liao, W.; Mura, M.D.; Chanussot, J.; Piźurica, A. Fusion of spectral and spatial information for classification of hyperspectral remote-sensed imagery by local graph. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **2016**, *9*, 583–594. [CrossRef]
2. Fauvel, M.; Chanussot, J.; Benediktsson, J.A. A spatial-spectral kernel-based approach for the classification of remote-sensing images. *Pattern Recognit.* **2012**, *45*, 381–392. [CrossRef]
3. Zhang, L.; Zhang, L.; Tao, D.; Huang, X.; Du, B. Hyperspectral remote sensing image subpixel target detection based on supervised metric learning. *IEEE Trans. Geosci. Remote Sens.* **2014**, *52*, 4955–4965. [CrossRef]
4. Xia, J.; Chanussot, J.; Du, P.; He, X. (Semi-) supervised probabilistic principal component analysis for hyperspectral remote sensing image classification. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **2014**, *7*, 2224–2236. [CrossRef]
5. Gu, Y.; Wang, C.; You, D.; Zhang, Y.; Wang, S.; Zhang, Y. Representative Multiple Kernel Learning for Classification in Hyperspectral Imagery. *IEEE Trans. Geosci. Remote Sens.* **2012**, *50*, 2852–2865. [CrossRef]
6. Cheng, G.; Yang, C.; Yao, X.; Guo, L.; Han, J. When Deep Learning Meets Metric Learning: Remote Sensing Image Scene Classification via Learning Discriminative CNNs. *IEEE Trans. Geosci. Remote Sens.* **2018**, *56*, 2811–2821. [CrossRef]
7. Xiao, F.; Chen, L.; Sha, C.; Sun, L.; Wang, R.; Liu, A.X.; Ahmed, F. Noise Tolerant Localization for Sensor Networks. *IEEE/ACM Trans. Netw.* **2018**, *26*, 1701–1714. [CrossRef]
8. Zare, A.; Bolton, J.; Chanussot, J.; Gader, P. Foreword to the Special Issue on Hyperspectral Image and Signal Processing. *IEEE Trans. Geosci. Remote Sens.* **2010**, *7*, 1841–1843. [CrossRef]
9. Sun, X.; Qu, Q.; Nasrabadi, N.M.; Tran, T.D. Structured Priors for Sparse-Representation-Based Hyperspectral Image Classification. *IEEE Geosci. Remote Sens. Lett.* **2014**, *11*, 1235–1239.
10. Wu, Z.; Wang, Q.; Plaza, A.; Li, J.; Liu, J.; Wei, Z. Parallel Implementation of Sparse Representation Classifiers for Hyperspectral Imagery on GPUs. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **2015**, *8*, 2912–2925. [CrossRef]
11. Wu, Z.; Shi, L.; Li, J.; Wang, Q.; Sun, L.; Wei, Z.; Plaza, J.; Plaza, A. GPU Parallel Implementation of Spatially Adaptive Hyperspectral Image Classification. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **2017**, *11*, 1131–1143. [CrossRef]
12. Wu, Z.; Wang, Q.; Plaza, A.; Li, J.; Sun, L.; Wei, Z. Parallel Spatial-Spectral Hyperspectral Image Classification With Sparse Representation and Markov Random Fields on GPUs. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **2015**, *8*, 2926–2938. [CrossRef]
13. Liu, C.; He, L.; Li, Z.; Li, J. Feature-Driven Active Learning for Hyperspectral Image Classification. *IEEE Trans. Geosci. Remote Sens.* **2017**, *56*, 341–354. [CrossRef]
14. Chang, C.I. *Hyperspectral Data Exploitation: Theory and Applications*; Wiley: New York, NY, USA, 2007.
15. Pontius, J.; Martin, M.; Plourde, L.; Hallett, R. Ash decline assessment in emerald ash borer-infested regions: A test of tree-level, hyperspectral technologies. *Remote Sens. Environ.* **2008**, *112*, 2665–2676. [CrossRef]

16. Fauvel, M.; Tarabalka, Y.; Benediktsson, J.A.; Chanussot, J.; Tilton, J.C. Advances in Spectral-Spatial Classification of Hyperspectral Images. *Proc. IEEE* **2013**, *101*, 652–675. [CrossRef]
17. Feng, L.; Zhu, S.; Lin, F.; Su, Z.; Yuan, K.; Zhao, Y.; He, Y.; Zhang, C. Detection of Oil Chestnuts Infected by Blue Mold Using Near-Infrared Hyperspectral Imaging Combined with Artificial Neural Networks. *Sensors* **2018**, *18*, 1944. [CrossRef] [PubMed]
18. Chu, B.; Yu, K.; Zhao, Y.; He, Y. Development of Noninvasive Classification Methods for Different Roasting Degrees of Coffee Beans Using Hyperspectral Imaging. *Sensors* **2018**, *18*, 1259. [CrossRef] [PubMed]
19. Zhang, Y.; Du, B.; Zhang, L.; Wang, S. A Low-Rank and Sparse Matrix Decomposition-Based Mahalanobis Distance Method for Hyperspectral Anomaly Detection. *IEEE Trans. Geosci. Remote Sens.* **2016**, *54*, 1376–1389. [CrossRef]
20. Du, B.; Zhang, L. Random-Selection-Based Anomaly Detector for Hyperspectral Imagery. *IEEE Trans. Geosci. Remote Sens.* **2011**, *49*, 1578–1589. [CrossRef]
21. Reed, I.S.; Yu, X. Adaptive multiple-band CFAR detection of an optical pattern with unknown spectral distribution. *IEEE Trans. Acoust. Speech Signal Process.* **1990**, *38*, 1760–1770. [CrossRef]
22. Nasrabadi, N.M. Regularization for spectral matched filter and RX anomaly detector. *Proc. Spie Int. Soc. Opt. Eng.* **2008**, *6966*, 696604.
23. Matteoli, S.; Diani, M. Improved estimation of local background covariance matrix for anomaly detection in hyperspectral images. *Opt. Eng.* **2010**, *49*, 258. [CrossRef]
24. Guo, Q.; Zhang, B.; Ran, Q.; Gao, L.; Li, J.; Plaza, A. Weighted-RXD and Linear Filter-Based RXD: Improving Background Statistics Estimation for Anomaly Detection in Hyperspectral Imagery. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **2014**, *7*, 2351–2366. [CrossRef]
25. Sakla, W.; Chan, A.; Ji, J.; Sakla, A. An SVDD-Based Algorithm for Target Detection in Hyperspectral Imagery. *IEEE Geosci. Remote Sens. Lett.* **2011**, *8*, 384–388. [CrossRef]
26. Du, B.; Zhang, L. A Discriminative Metric Learning Based Anomaly Detection Method. *IEEE Trans. Geosci. Remote Sens.* **2014**, *52*, 6844–6857.
27. Chen, Y.; Nasrabadi, N.M.; Tran, T.D. Simultaneous Joint Sparsity Model for Target Detection in Hyperspectral Imagery. *IEEE Geosci. Remote Sens. Lett.* **2011**, *8*, 676–680. [CrossRef]
28. Chen, Y.; Nasrabadi, N.M.; Tran, T.D. Hyperspectral Image Classification via Kernel Sparse Representation. *IEEE Trans. Geosci. Remote Sens.* **2012**, *51*, 217–231. [CrossRef]
29. Li, W.; Du, Q. Collaborative Representation for Hyperspectral Anomaly Detection. *IEEE Trans. Geosci. Remote Sens.* **2015**, *53*, 1463–1474. [CrossRef]
30. Xu, Y.; Wu, Z.; Li, J.; Plaza, A.; Wei, Z. Anomaly Detection in Hyperspectral Images Based on Low-Rank and Sparse Representation. *IEEE Trans. Geosci. Remote Sens.* **2016**, *54*, 1990–2000. [CrossRef]
31. Blanchard, J.D.; Tanner, J. GPU accelerated greedy algorithms for compressed sensing. *Math. Program. Comput.* **2013**, *5*, 267–304. [CrossRef]
32. Fiandrotti, A.; Fosson, S.M.; Ravazzi, C.; Magli, E. GPU-accelerated algorithms for compressed signals recovery with application to astronomical imagery deblurring. *Int. J. Remote Sens.* **2017**, *39*, 2043–2065. [CrossRef]
33. Paz, A.; Plaza, A. GPU implementation of target and anomaly detection algorithms for remotely sensed hyperspectral image analysis. *Proc. SPIE* **2010**, *7810*, 78100R.
34. Ferreira, R.S.; Bentes, C.; Costa, G.A.O.P.; Oliveira, D.A.B.; Happ, P.N.; Feitosa, R.Q.; Gamba, P. A set of methods to support object-based distributed analysis of large volumes of earth observation data. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **2017**, *10*, 681–690. [CrossRef]
35. Wu, Z.; Li, Y.; Plaza, A.; Li, J.; Xiao, F.; Wei, Z. Parallel and dstributed dimensionality reduction of hyperspectral data on cloud computing architectures. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **2016**, *9*, 2270–2278. [CrossRef]
36. Quirita, V.A.A.; da Costa, G.A.O.P.; Happ, P.N.; Feitosa, R.Q.; Ferreira, R.D.S.; Oliveira, D.A.B.; Plaza, A. A new cloud computing architecture for the classification of remote sensing data. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **2017**, *10*, 409–416. [CrossRef]
37. Costa, G.A.O.P.; Bentes, C.; Ferreira, R.S.; Feitosa, R.Q.; Oliveira, D.A.B. Exploiting different types of parallelism in distributed analysis of remote sensing data. *IEEE Geosci. Remote Sens. Lett.* **2017**, *PP*, 1–5. [CrossRef]

38. Paz, A.; Plaza, A.; Plaza, J. Comparative analysis of different implementations of a parallel algorithm for automatic target detection and classification of hyperspectral images. *Proc. SPIE* **2009**, *7455*, 74550X.
39. Paz, A.; Molero, J.M.; Garzon, E.M.; Martinez, J.A.; Plaza, A. A New Parallel Implementation of the RX Algorithm for Anomaly Detection in Hyperspectral Images. In Proceedings of the International Conference on Computational and Mathematical Methods in Science and Engineering, Almeria, Spain, 27–30 June 2010.
40. Paz, A.; Plaza, A. Clusters versus GPUs for parallel target and anomaly detection in hyperspectral images. *EURASIP J. Adv. Signal Process.* **2010**, *2010*, 915639. [CrossRef]
41. Dean, J.; Ghemawat, S. MapReduce: Simplified data processing on large clusters. *Commun. ACM* **2008**, *51*, 107–113. [CrossRef]
42. Yang, M.; Song, W.; Mei, H. Efficient Retrieval of Massive Ocean Remote Sensing Images via a Cloud-Based Mean-Shift Algorithm. *Sensors* **2017**, *17*, 1693. [CrossRef] [PubMed]
43. Li, Z.; Su, D.; Zhu, H.; Li, W.; Zhang, F.; Li, R. A Fast Synthetic Aperture Radar Raw Data Simulation Using Cloud Computing. *Sensors* **2017**, *17*, 113. [CrossRef] [PubMed]
44. Liu, Q.; Cai, W.; Jin, D.; Shen, J.; Fu, Z.; Liu, X.; Linge, N. Estimation Accuracy on Execution Time of Run-Time Tasks in a Heterogeneous Distributed Environment. *Sensors* **2016**, *16*, 1386. [CrossRef] [PubMed]
45. Hussain, S.; Bang, J.H.; Han, M.; Ahmed, M.I.; Amin, M.B.; Lee, S.; Nugent, C.; Mcclean, S.; Scotney, B.; Parr, G. Behavior life style analysis for mobile sensory data in cloud computing through MapReduce. *Sensors* **2014**, *14*, 22001. [CrossRef] [PubMed]
46. Ryza, S.; Laserson, U.; Owen, S.; Wills, J. *Advanced Analytics with Spark: Patterns for Learning from Data at Scale*; O'Reilly Media: Sebastopol, CA, USA, 2015.
47. Reyes-Ortiz, J.L.; Oneto, L.; Anguita, D. Big Data Analytics in the Cloud: Spark on Hadoop vs MPI/OpenMP on Beowulf. *Procedia Comput. Sci.* **2015**, *53*, 121–130. [CrossRef]
48. Zaharia, M.; Chowdhury, M.; Franklin, M.J.; Shenker, S.; Stoica, I. Spark: Cluster computing with working sets. In Proceedings of the Usenix Conference on Hot Topics in Cloud Computing, Boston, MA, USA, 22–25 June 2010; p. 10
49. Moreno-Torres, J.G.; Saez, J.A.; Herrera, F. Study on the impact of partition-induced dataset shift on k-fold cross-validation. *IEEE Trans. Neural Netw. Learn. Syst.* **2012**, *23*, 1304–1312. [CrossRef] [PubMed]
50. Senthilnath, J.; Sindhu, S.; Omkar, S.N. GPU-based normalized cuts for road extraction using satellite imagery. *J. Earth Syst. Sci.* **2014**, *123*, 1759–1769. [CrossRef]