Hindawi
Journal of Sensors
Volume 2018, Article ID 2075057, 12 pages
https://doi.org/10.1155/2018/2075057

Research Article

Spark Sensing: A Cloud Computing Framework to Unfold Processing Efficiencies for Large and Multiscale Remotely Sensed Data, with Examples on Landsat 8 and MODIS Data

Hai Lan,[1] **Xinshi Zheng**,[2] **and Paul M. Torrens**[1,2]

[1]*Department of Computer Science and Engineering, Tandon School of Engineering, New York University, Brooklyn, NY 11201, USA*
[2]*Center for Urban Science + Progress, New York University, Brooklyn, NY 11201, USA*

Correspondence should be addressed to Hai Lan; hai.lan@nyu.edu

Received 26 April 2018; Accepted 8 July 2018; Published 23 August 2018

Academic Editor: Victor Mesev

Inquiry using data from remote Earth-observing platforms often confronts a straightforward but particularly thorny problem: huge amounts of data, in ever-replenishing supplies, are available to support inquiry, but scientists' agility in converting data into actionable information often struggles to keep pace with rapidly incoming streams of data that amass in expanding archival silos. Abstraction of those data is a convenient response, and many studies informed purely by remotely sensed data are by necessity limited to a small study area with a relatively few scenes of imagery, or they rely on larger mosaics of images at low resolution. As a result, it is often challenging to thread explanations across scales from the local to the global, even though doing so is often critical to the science under pursuit. Here, a solution is proposed, by exploiting Apache Spark, to implement parallel, in-memory image processing with ability to rapidly classify large volumes of multiscale remotely sensed images and to perform necessary analysis to detect changes on the time series. It shows that processing on three different scales of Landsat 8 data (up to ~107.4 GB, five-scene, time series image sets) can be accomplished in 1018 seconds on local cloud environment. Applying the same framework with slight parameter adjustments, it processed same coverage MODIS data in 54 seconds on commercial cloud platform. Theoretically, the proposed scheme can handle all forms of remote sensing imagery commonly used in the Earth and environmental sciences, requiring only minor adjustments in parameterization of the computing jobs to adjust to the data. The authors suggest that the "Spark sensing" approach could provide the flexibility, extensibility, and accessibility necessary to keep inquiry in the Earth and environmental sciences at pace with developments in data provision.

1. Introduction

Data provided by remote sensing have long presented as a critical resource in monitoring, measuring, and explaining natural and physical phenomena. Indeed, remote sensing might justly be characterized as one of the first "big data" sciences [1]. Steadfastly, for the advances in the sensing capabilities of remote, Earth-observing platforms have continued to produce more and more data, with increasing observational breadth and finesse of detail. These developments carry a dual benefit and problem: analysis and inquiry in the environmental and Earth sciences not only are routinely awash with data but also often struggle to match pace in building empirical knowledge from those data because the data are incoming with such haste and heft. Strategies to manage big remotely sensed data are required to fully exploit the benefits those data hold for applied scientific inquiry, and the topic of how computing might be leveraged to ease pathways between science and sensing holds significant currency across many fields, with particularly rapid adoption of high-performance computing [2] and cloud computing in the geographical sciences [3].

Remote sensing imagery is a commonly used source to support those studies of sustainable ecosystems, such as ecosystem dynamics, grassland degradation, and urban ecosystem restoration, especially in large areas [4]. Traditionally, studies with pure remotely sensed data involved only a few scenes of data in a limited study area, or they rely on

low-resolution remotely sensed images in large-area experiments [5]. Those traditions are changing as new data have dramatically altered the underlying substrate for analysis. For example, in the past few decades, the space-borne and air-borne Earth observation sensors are continually providing large-volume datasets. For example, Landsat 8, the latest Landsat mission launched in 2013, can collect more than 700 images per day, corresponding to approximately 86 terabytes of data per year [6], which is 14 times as much as that in the 1980s [4]. Processing the massive volume of remotely sensed data is now not the only problem: the intrinsic complexity of those data is also an important issue that must be considered.

The sensors that are actuated in remote sensing are usually designed to serve specific requirements of analysis for different fields of study. To fulfill those different needs, sensors are usually tasked to capture images at different resolutions. For example, high-resolution satellite sensors such as *WorldView-4* can produce imagery with a spatial resolution of 0.31 m in panchromatic vistas and 1.24 m spatial resolution in multispectral vistas [7], while the *QuickBird* platform can image the Earth with 0.61 m spatial resolution in panchromatic form and 2.44 m spatial resolution in multispectral form [8]. These resolutions, on the order of fractions of a meter to a few meters in spatial resolution, presented significant opportunities to monitor the Earth and represent the state of the art in Earth-observing imaging detail.

Concurrently, other remote sensing platforms are tasked with refreshing observations of the whole Earth's surface, aiming for coverage of large areas with temporal consistency, rather than small-area detail. For example, relatively medium-resolution sensors, such as *Landsat*, and relatively low-resolution sensors, such as *MODIS*, are deployed as long-term Earth observatories. Landsat provides 15 m panchromatic and 30 m multispectral imagery, which is very widely used in studies of large-area grassland degradation and urban land cover dynamics [9]. MODIS offers 250 m multispectral imagery and can build a mosaic view of the entire Earth once every few days. MODIS data has been widely adopted in global-scale research studies, particularly those trained on studying vegetation canopies for investigation of worldwide forest cover dynamics [10].

Many sensors support multispectral imaging. For example, WorldView-4 data includes four spectral bands, and Landsat 8 OLI/TIRS provides 11 spectral bands. For some spectrally sensitive studies, higher spectral resolution imagery is required. In those studies, hyperspectral sensors (such as *Hyperion*, which generates 220 bands between 0.4–2.5 μm [11]) can produce detailed spectral data over a very small wavelength range. Furthermore, different sensors offer different temporal resolutions in their rate of imaging as well as the timing of their coverage of subjects under their purview. For example, the WorldView-4 satellite is capable of revisiting views every 4.5 days (sometimes sooner), while Landsat can deliver repeat views every 16 days [12]. Higher temporal resolution in return views facilitates the study of dynamics on the Earth surface, so that the time series and the time interval between visits become significant attributes of the observation, alongside the spatial resolution and spectral range. Therefore, methods to streamline a feasible, effective, and efficient approach to processing archived and continually incoming multispatial, multispectral, and multitemporal remote sensing data are an ongoing requirement across many potential applications of remote sensing to applied scientific inquiry.

In this paper, possible scalable solutions are introduced to address issues of processing multiscale large-volume remote sensing datasets in multispatial, multispectral, and multitemporal cases. The aim is to implement a tool that can process different proposed remote-sensed tasks with only minor adjustments rather than fully rebuild new toolkits. Furthermore, this solution should be fully capable of exploiting benefits from cutting-edge cloud computing technologies, resources, and platforms to help researchers process and analyze large remotely sensed datasets that are difficult to process on local machines in an effective and efficient manner.

1.1. Cloud Computing as a Resource for Big Data Processing. Many researchers have made significant progress in advancing feasible, effective, and efficient processing for multiprong attributes of remotely sensed data, using developments in computer engineering. In particular, research into how graphical processing units (GPUs) and cluster-based high-performance computing (HPC) might be leveraged to advance image processing for remote sensing has been particularly fruitful [13]. More recently, cloud computing is increasingly being considered as a resource in processing remotely sensed imagery, largely because of cloud computing's native abilities to scale computing in kind as the data being processed also the scale. Furthermore, significant cloud computing resources are now available commercially, on a "pay as you go" model, from providers such as *Amazon Web Services* (AWS) [14], *Microsoft Azure* [15], and *Google*'s *Compute Engine* [16]. These resources can be brought to bear on image processing tasks as IaaS (infrastructure as a service), PaaS (platform as a service), or SaaS (software as a service).

Cloud computing is useful in providing some of the flexibility required to match pace between incoming data, large existing data silos, and evolving analytical needs in image processing that authors alluded to in the introduction. Cloud computing affords this flexibility by allowing users to allocate and share software and hardware resources on the Internet in a distributed fashion, by splitting large computational tasks into many small parallel computing tasks, then assigning them to as many computing instances as are required to achieve computing goals based on data size, data fusion, resource use, or computing time. After all the distributed nodes of the cloud service have completed their assigned tasks, the results are bundled and returned to the users' local database. In this way, virtual instances, applications, and software are provided on an as-requested basis, and users may pay for those services as demanded. This affords a user access to a theoretically limitless size computing capacity (although very strong limits of available financial budgets to pay for the services quickly dock theoretical capacities to tangible practical realities in many real instances).

A promising community of computing frameworks has codeveloped alongside cloud computing hardware, and

several of these frameworks hold significant promise for processing remotely sensed imagery of the Earth's surface. For example, MapReduce was introduced by Dean and Ghemawat [17], ten years ago. In the decade since, a number of open-source implementations of the MapReduce model have emerged as promising frameworks for mediating the computing between image processing for remotely sensed data and cloud resources that are available to distribute and/or accelerate that computing on commercial (or user-run) clouds. Chief among these open-source implementations of MapReduce is *Apache Hadoop*. While Hadoop MapReduce relies on reading and writing data to a disk, another variant, Apache Spark [18] maintains data partitions in memory (a so-called in-memory computing framework). Spark also provides a network buffer for each reducing task, rather than merging outputs into a single partition, with the result that Spark can be one hundred times faster than Hadoop MapReduce on some big data tasks [19]. Nevertheless, one advantage that Hadoop might hold over Spark is that Hadoop allows parallel processing of large amounts of data that are bigger in physical storage size than the available memory. In fact, many remote sensing datasets are of a size that is so massive that they exceed the memory available in local machines or small clusters. Furthermore, physical disk resources are usually much less expensive in financial cost (of owning or accessing) than memory resources are. So, in cases for which limited memory may become a constraining factor, Hadoop presents as a better option in some cases for processing large amounts of remote sensing data.

However, cloud computing frameworks are agile relative to resource constraints. And that novel advances in cloud computing technologies and cloud platforms allow Spark to leverage resources from and across different cloud computing platforms, with the possibility that memory limitations may no longer loom as large a constraint for big remote sensing data processing scenarios. Consider memory as a resource that can be drawn upon on an as-needed basis, researchers can access theoretically unlimited memory on the cloud. Furthermore, Spark can run on a single workstation, as well as local computing clusters and cloud platforms. And Spark could access diverse data sources, such as *Amazon S3*, *Hadoop Distributed File System* (HDFS), *Cassandra*, and *HBase*. In other words, Spark not only can access local private data warehouses but also can reach cloud-stored big remote sensing datasets and do so via the cloud platform directly, with the ability to process those data on the cloud and then stream back the required results.

1.2. Cloud Computing for Processing Remotely Sensed Images. One of the common computing solutions to the burden of processing, analyzing, and managing large-scale remote sensing data is to parallelize the remote sensing processing tasks: to spread the burden over multiple computing units to reduce the overall processing time [6]. For example, Huang and her colleagues used the message passing interface (MPI) as a computing framework for their work on dust storm simulation and forecasting on the *Amazon EC2* commercial cloud service [13]. They deployed MPI on the Amazon cloud and applied loosely coupled nested models to process a high-resolution dust storm dataset. Their performance tests showed efficient and economical results. Cavallaro et al. [20] used GPUs to implement a support vector machine (SVM) classifier with MPI and *openMPI* frameworks. As an alternative to using HPC computing frameworks, other researchers have developed their own bespoke parallel large-scale remote sensing data processing platforms. For example, Wang et al. [21] developed *pipsCloud*, which is a cloud-based HPC approach to process remote sensing on-demand and in real time. To further enhance performance, Wang et al. [21] used Hilbert R^+-tree indexing.

The turn toward development of tools by remote sensing scientists, for remote sensing scientists, leveraging computing techniques but departing in ways that are special to remote sensing applications, is a wonderful development for remote sensing science. Nevertheless, bespoke solutions (particularly in academic settings) cannot feasibly contribute to the massive levels of computing available now commercially, with the result that absolute performance will always lag behind that which might otherwise be available on the marketplace. Also, applying MPI or self-developed systems often requires significant research and development effort into programming, debugging, and tuning the computing system and environment, and one might perhaps make an argument that the time devoted to these tasks could be used on the applied science instead. In some cases, building these systems on a bespoke basis is very challenging. For example, consider MPI, programming tasks designed for serial computing and converting them to parallel form can be significantly burdensome and particularly so for some complex image processing algorithms. Moreover, the networking security, near ubiquitous availability, and fast-moving hardware compatibility (e.g., in shared memory clusters) of commercial platforms offer significant practical advantages. Some existing work points to the potential advantages that are obtainable in cloud processing of big geospatial data. For example, Chen and Zhou [22] demonstrated that Apache Hadoop can be leveraged for partitioning using a mean shift algorithm. With a local mode test, they successfully increased the processing speed by ~2 times [22]. Also, Giachetta [23] introduced a Hadoop-based geospatial data management and processing toolkit, *AEGIS*, which he compared against many existing MapReduce-based frameworks, such as *SpatialHadoop*, *Hadoop-GIS*, *HIPI*, and *MrGeo* with spatial join, query, and aggregation operations [23].

Compared to MapReduce-based approaches, solutions based on Apache Spark can usually generate results at higher efficiency, as mentioned in "Introduction." For example, Sun et al. [24] used *MLlib* in Spark to test the multi-iteration singular value decomposition (SVD) algorithm on high-resolution hyperspectral remote sensing images. Compared with the *Apache Mahout* (MapReduce) approach, they found that the Spark approach can essentially trounce MapReduce in their tests, once Spark is able to access enough hardware resources. Another study using Spark to process massive remote sensing data, by Huang and his colleagues [25], demonstrated a series of comprehensive performance tests, using Spark to implement different types of algorithms

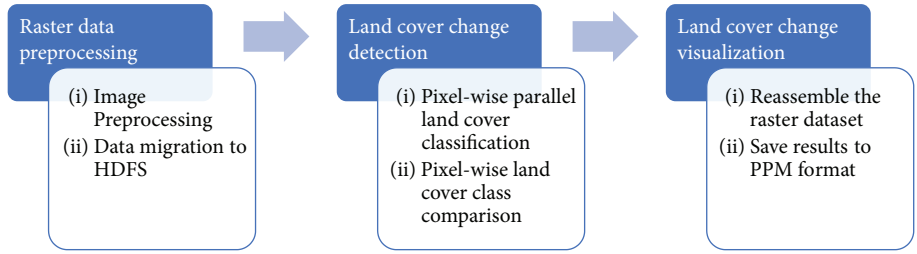

Figure 1: Spark-based remote sensing image processing workflow.

in remote sensing. Huang et al. [25] discussed the performance of each of the tests on different running environments, including local, standalone, and *yet another resource negotiator* (YARN). They also proposed a self-defined, strip-based partitioning approach to replace the default hash partitioning method [25].

2. Methodology and Data Sources

In this paper, authors propose to extend upon recent developments in cloud computing as a resource for processing remotely sensed imagery. Specifically, a Spark-based large-scale remote sensing image processing tool that can be deployed on cloud platforms will be introduced. In the following sections, authors will discuss how to design experiments by applying two commonly used classifiers—normalized difference vegetation index (NDVI) and normalized difference water index (NDWI)—as a testbed to assess the feasibility and performance of our tool. A scheme for designing a change detection scheme based on classification results generated by the proposed tool on a time series will be also introduced. One of the key advantages of this approach is in its ability to easily and straightforwardly support users' access to processed land cover changes within a large set of spatial-temporal images. Moreover, the paper will demonstrate a visualization approach to save text-based output from the analysis in common picture format, allowing users to examine results easily and quickly.

2.1. Methodology. The approach presented in this paper is based on the YARN cloud environment. The goal, in using YARN, is to facilitate the availability of the processing environment in ways that are widely applicable to real-world scenarios. YARN has been widely used in many current cloud environments [26]. Unlike traditional versions of Hadoop MapReduce, YARN allows for allocation of system resources as containers to the various applications. In other words, different computing frameworks can be deployed on a single physical cluster in a noninterfering manner. In YARN mode, a Spark framework is composed of a master instance and many workers. The master instance is responsible for negotiating with the YARN *Resource Manager* to request enough computing resources, as needed, by analyzing the Spark applications submitted by clients. Once the master instance is loaded, it will schedule the tasks to executors with allocated containers. Until all tasks have been finished, Resource Manager will revoke all allocated resources for further possible tasks.

Resilient distributed datasets (RDD) form the basic abstraction of a dataset in Spark. RDDs can be created from an external dataset, or from existing RDDs. For cases in this research, RDDs will be created from the original remote sensing images stored in HDFS. For each image, three RDDs will be created for green, red, and near-infrared bands for NDVI and NDWI land cover classification at the initial stage. RDDs contain the data partitions and the metadata records. According to the Spark mechanism, RDDs will go through a series of transformations and action operations to process the data partitions in a distributive manner. The transformation operations of RDDs will be executed on individual partitions of an RDD and those operations will return a new RDD. Action operations will summarize information from an RDD by user-defined functions and return a result. For example, the joining operation, as a commonly used transformation operation in Spark, involves joining partitions belonging to two RDDs and creating a new one. In this case, the data blocks will not be moved at the current stage because of the lazy mechanism. However, action operations, such as count, will process the data partitions in RDD and perform real computing.

Based on the features of Spark, a workflow has been designed as illustrated in Figure 1. The first step in that workflow is to extract each band of a raster dataset by using the *Geospatial Data Abstraction Library* (GDAL), which is an open-source translator library for raster and vector geospatial data formats [27]. Each line of the extracted data contains the geographical coordinates and the digital number (DN) value of the raster cell in the original raster dataset. These files are then put into the HDFS as data input sources for distributive processing.

The second step is to perform parallel processing of image classification and land cover change detection. As introduced before, for each image that must be processed by this tool, three RDDs will be created for its green, red, and near-infrared bands. The basic units of the RDD are key-value pairs. In this case, the geographical coordinates are set as the key, and the DN value of a raster cell will be the value. In this manner, parallel land cover classifications can be conducted across many computer units in a cluster. The extent of parallel processing depends on the Spark application configuration and the cluster hardware specifications. Because the basic parallel processing unit is pixel-wise, this method can be highly flexible and scalable, allowing raster datasets to be partitioned in any manner, regardless of the spatial structures of the raster. However, partitioning

```
1: Input= TIF_1, TIF_2, … , TIF_k; Dimension = X*Y
2: For k = 1 to K-1 do:
3:     InputTIF_1 = TIF_k
4:     InputTIF_2 = TIF_k+1
5:     For m = 1 to 2 do:
6:         For x in 1 to X
7:             For y in 1 to Y:
8:                 ndvi_xy = (InputTIF_xy.NIR − InputTIF_xy.R)/(InputTIF_xy.NIR + InputTIF_xy.R)
9:                 ndwi_xy = (InputTIF_xy.G − InputTIF_xy.NIR)/(InputTIF_xy.G + InputTIF_xy.NIR)
10:                landcover_xy = CLASSIFIED(ndvi_xy, ndwi_xy)
11:                landcover_m[x, y] = landcover_xy
12:            endFor
13:        endFor
14:    endFor
15:    landcover_change_k = CHANGE(landcover_1, landcover_2)
16:    Output = landcover_cover_k
17: endFor
```

Algorithm 1: NDVI/NDWI classification and changing detection.

strategies still must be seriously considered, because an appropriate partitioning strategy will help to optimize the performance with better usage of the computing resources of the cluster. Spark supports hash partitioning and range partitioning by default; these applications are appropriate for many cases in the real world. However, according to the results of the study of Huang et al. [25], the partition scale cannot be too small or too large. A very small partition scale will result in low-performance computing and even increase the fault recovery cost. A very large partition scale may lead to the out-of-memory error. Inspired by their strip-based partitioning method, splitting data into chunks with a configured HDFS block size is needed in following experiments.

In Algorithm 1, the detailed algorithm of using NDVI and NDWI as classifiers on remote sensing images has been illustrated. Several transformation operations are performed on each partition in RDDs. For k input *GeoTiFF* remote sensing images, the algorithm will first parse the dimension of them. Moreover, those created RDDs will be marked by a time sequence in the time series. For each data chunk, in each pair of image RDDs (e.g., a pair of images in 2013 and images in 2014), the NDVI and NDWI classifier will be applied to each pixel to calculate the indicated values. The NDVI [28] can be calculated according to its definition as

$$\mathrm{NDVI} = \frac{\mathrm{NIR} - \mathrm{red}}{\mathrm{NIR} + \mathrm{red}}. \quad (1)$$

The range of NDVI is from negative one to one, which can use different ranges to denote sparse vegetation, dense vegetation, barren rock and sand, and water. However, NDVI may not always correctly distinguish the water body, especially when there is a noisy signal in the water area (e.g., mud). To further improve the accuracy of classification, NDWI is applied according to McFeeters [29] as follows:

$$\mathrm{NDWI} = \frac{\mathrm{green} - \mathrm{NIR}}{\mathrm{green} + \mathrm{NIR}}. \quad (2)$$

NDWI can distinguish the water feature with the positive value indicator. From the perspective of algorithm implementation, RDDs of three bands will be joined as a single RDD for NDVI and NDWI calculation of each raster cell with mapping operations. Land cover classification can, therefore, be conducted in parallel by using calculated NDVI and NDWI results. Once the land cover features are classified by NDVI and NDWI indicators, each feature can be labelled with a class ID and pass those IDs when changing the detection function. The changing detection function will compare the feature IDs from each image pair and summarize how those land features change from the former year to the current year.

The final step is to visualize land cover change, as illustrated in pseudo code in Algorithm 2. With all k numbers of land cover change results generated by Algorithm 1 for each image pair, visualization images are created in PPM format from the obtained output RDD, which is a lowest common denominator color image file format [30]. Although redundant, PPM is an easy format to write and manage text-based outputs into human-readable figures. The two RDDs of land cover classifications for two images from Algorithm 1 will be joined as a single RDD, and the values of key-value pairs will be converted to colors according to a user-defined RGB color scheme. This joined RDD will then be sorted to the original order and reduced in a manner such that each key-value pair represents a row of the original raster image. Finally, to produce a PPM-format image, this RDD is appended to the PPM file header to create a complete PPM image.

1: Input = landcover_change$_1$, landcover_change$_2$, … , landcover_change$_K$; Dimension = X*Y
2: **For** k = 1 to K **do:**
3: InputKV$_k$ = landcover_change$_k$
4: rgbKV$_k$ = RGB(inputKV$_k$)
5: **Sort** rgbKV$_k$ by two dimensions
6: rgbV$_k$ = values of rgbKV$_k$
7: Reshape rgbV$_k$ to dimensions X*Y
8: Output = rgbV$_k$
9: **Save** rgbV$_k$ to files
10: **endFor**

Algorithm 2: Visualization.

An overview of how the whole process works is illustrated in Figure 2. Four steps are applied in this design: (1) read and parse each pair of input images and create RDDs for green, red, and near-infrared bands; (2) split data into chunks and process them in parallel; (3) gather results and assign labels to each pixel; and (4) reconstruct whole images by sorting output RDDs and visualize them in PPM format.

2.2. Data Sources. To prove the applied utility of the proposed scheme, three experiments are performed, and two different remote-sensed imagery datasets are involved. The first experiment is using the Landsat 8 operational land imager (OLI) dataset in three different scales to test the workflow of classification and change detection and visualization algorithm with Spark on local cloud environment. To further study the tuning performance of this scheme, the second experiment is designed to run this tool under different execution configurations with the same Landsat 8 dataset. The last experiment is set to prove this tool can consume different source remote-sensed imagery datasets with only minor parameter adjustments. By using the same algorithm presented above, a MODIS dataset on Amazon EC2 which is a real commercial cloud platform is processed.

Landsat 8 scans the entire planet surface in a 16-day period [31]. Equipped with the latest OLI sensor, Landsat 8 provides unprecedented spectral information with two additional spectral bands in the whole Landsat instrument family. In addition to offering a 15 m panchromatic band and a 30 m multispectral band, as in many previous products, Landsat 8 also includes a quality assessment (QA) band to support pixel-based cloud, shadow, and terrain occlusion filtering. A relatively short revisit period, medium spatial resolution, and seven spectral bands make Landsat 8 OLI products a commonly used freely accessible remote sensing imagery datasets for science that relies upon spatial, spectral, and temporal Earth attributes for environmental research. Furthermore, the Landsat 8 dataset is currently open to the public on cloud platforms, such as Amazon S3 [32] and *Google Earth Engine* (GEE), making it a great data source to serve cloud-based large-scale remote sensing processing in the cloud.

Like Landsat 8 OLI, MODIS satellite datasets are landed on as Amazon S3 [33] and GEE since 2017. MODIS provides a variety of planet observation products with daily temporal resolution. In this paper, MODIS/Terra Surface Reflectance Daily L2G Global 1 km and 500 m SIN Grid V006 (MOD09GA) will be used as second data input, which can provide 7 band surface spectral reflectance with 500-meter spatial resolution. Besides, a bunch of 1-kilometer resolution observation and geolocation statistic bands are offered in this product.

The coverage of each of the datasets that are used is illustrated in Figure 3. Three scales of Landsat 8 imagery datasets of small, medium, and large scale were used in experiments, as the input data source. This multiscale approach allowed us to assess whether our tool can perform multiscale remote sensing image processing with only minor parameter adjustments, to assess whether a wide array of remote sensing imagery datasets can be processed by our proposed framework in real scenarios. Because of remote sensing images, which are collected by different sensors at different spatial resolutions, with various bands of spectral information, and at diverse temporal scales are essentially formed by pixels, pixel-based algorithms implemented via Spark in the cloud should be capable of performing a very wide array of pixel-based processing and analysis. However, to further prove the presented tool can handle multisource remote sensing images in real cases, a MODIS dataset that fits for the exact same coverage of large-size Landsat 8 dataset is imported.

To generate a time series for change detection, one image per year has been chosen from a dataset spanning 2013 to 2017. The small-size images and medium-size images are approximately in one single scene of a Landsat image. Hence, those images that were acquired on a similar date for each year are preferably selected to reduce the land cover changes caused by seasonal variation factors. However, a large dataset is formed by over 15 scenes of original images. In this case, it cannot guarantee that each tile of this dataset can be filled by same-date images for each year. Hence, by broadening the filter criteria to the month level, obtaining enough tiles can be ensured to form the whole area. Another important factor is that the chosen images are preferably high quality to reduce the cloud and haze problems. For MODIS dataset, with relatively coarse spatial resolution and large coverage per scene, it takes about half of single scene to fit the large-size Landsat dataset coverage. Also, the acquisition time of it follows the date of the small- and medium-size

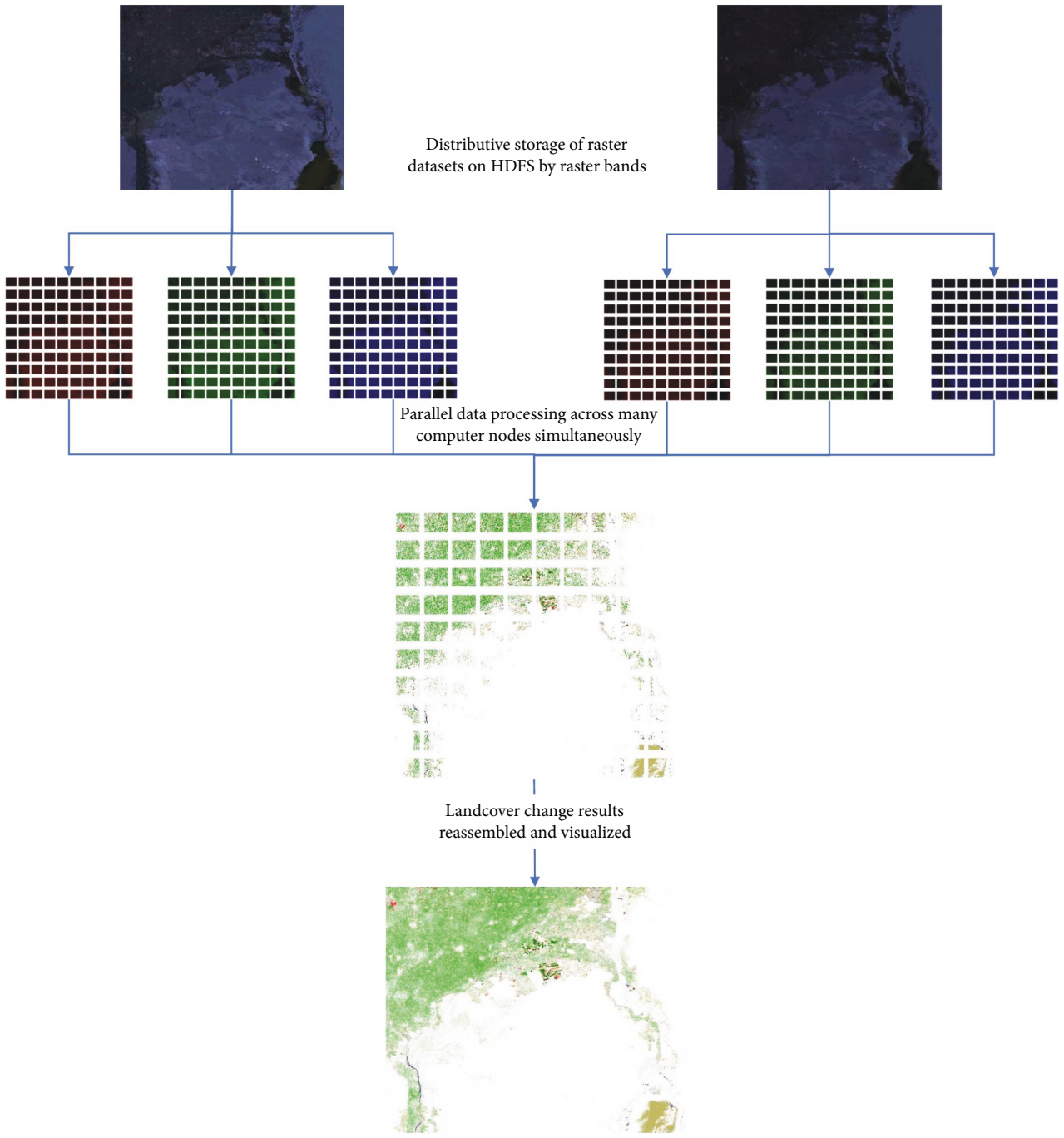

Figure 2: Detailed workflow for image classification and changing detection.

Landsat dataset because MODIS can provide daily products. The detailed acquisition time of each dataset is listed in Table 1.

The reason for the choice of the study area shown in Figure 3 is that, in the Suez Canal area, the land cover mainly includes sparse and dense vegetation, a natural water body, the Suez Canal (with water), and bare sands and rocks. Thus, the study area is a great test area for our NDVI/NDWI classifier to detect detailed land cover and generate relatively accurate change detection results. Another reason for the choice of the study area is that Suez Canal was expanded since 2013 to build another branch [34]. This project can be clearly monitored by the presented change detection process.

In those experiments, all remote sensing image datasets were exported from GEE, which is a cloud-based platform that can serve remote sensing data source with customized criteria [35], for example, setting (1) the region boundaries to acquire data belonging to our study area and (2) the cloud mask to filter the cloud pixels on images before exporting the data to our cloud drive. Landsat 8 and MODIS (MOD09GA) both provide surface reflectance produced on GEE. If other datasets are used, a strictly image-based atmospheric

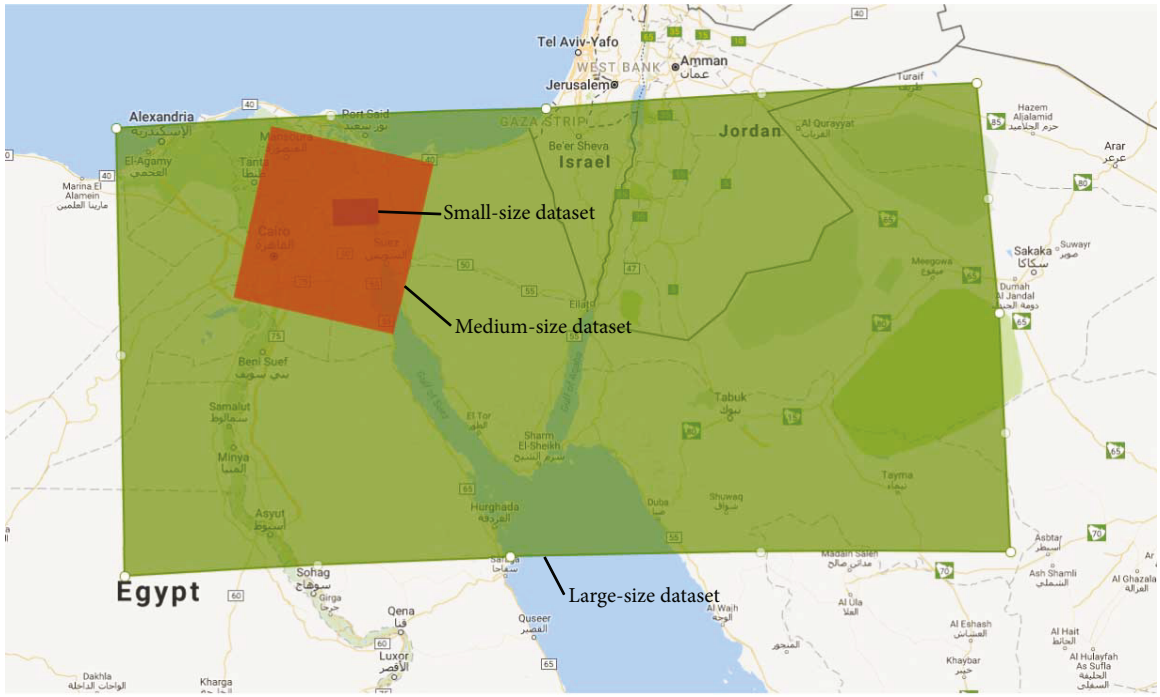

Figure 3: Landsat 8 image dataset in three different scales.

Table 1: Detailed acquisition time of each dataset.

	2013	2014	2015	2016	2017
Landsat_small	4/29	3/31	4/19	3/4	4/5
Landsat_medium	4/29	3/31	4/19	3/4	4/5
Landsat_large	Apr	Mar	Apr	Apr	Apr
MODIS	4/29	3/31	4/19	3/4	4/5

Table 2: Landsat data size and processing time.

	Scenes for each year	Size	Processing time
Small	~0.15	~1.2 G	67 s
Medium	~1	~10 G	276 s
Large	~15	~107.4 G	1018 s

correction should be followed to remove the haze impact on those images [36].

2.3. Experiment Environment. Two experiment environments are involved in this research for local cloud environment test and real commercial cloud platform test, respectively. The local computing cluster contains a total of 20 nodes. Two of them are master nodes, which equipped with 2 Intel Xeon E5-2680v4 2.4 GHz CPUs and 256 G memory each. 18 computing nodes are equipped with 2 Intel Xeon E5-2690v4 2.6 GHz CPUs (28 cores) and 256 G memory each. 10 Gbit network is assigned. In total, there are 1008 computing vcores and 5.12 Tb memory available. 2 Pb HDFS is configured and the block size is 128 M as default. Linux (Centos 6.9) is running on this cluster. Java 1.8.0_152 64-Bit Server VM is installed. Spark version is 2.2.0 with Cloudera release 1 and CDH 5.12.0.

The Amazon EC2-based computing cluster is another computing environment. Based on the computing needs of the third experiment in this paper, a 3-node cluster is built with 1 t2.xlarge instance with 4 vCPU Intel Broadwell E5-2686v4 2.3 GHz as master nodes and 2 t2.large instances with 2 vCPU Intel Broadwell E5-2686v4 2.3 GHz as slaves. All nodes are involved in computing. The total number of vcores is 8 and the overall memory is 32 G. 20 G storage per node is attached with default 128 M block size. The default HDFS replica is set as 3. Ubuntu Server 14.04 LTS (HVM) is the operation system. Similar to local computing cluster, Java 1.8.0_152 64-Bit Server VM, Spark 2.2.0 with Cloudera release 1, and CDH 5.12.0 are configured.

In the following parts of this paper, how authors applied the aforementioned workflow on real multiscale and different source remote sensing datasets to test the performance and feasibility of the Spark-sensing scheme will be discussed. The performance of the tool and the visualization of the experimental results will also be shown. Moreover, the flexibility, extensibility, and accessibility gleaned by using a Spark-based solution in remote sensing image dataset processing will be further discussed.

3. Results and Discussion

The first experiment is performed on a 20-node YARN computing cluster. In this experiment, a multiscale remote sensing image dataset with the Spark-based classification and change detection algorithm has been successfully processed.

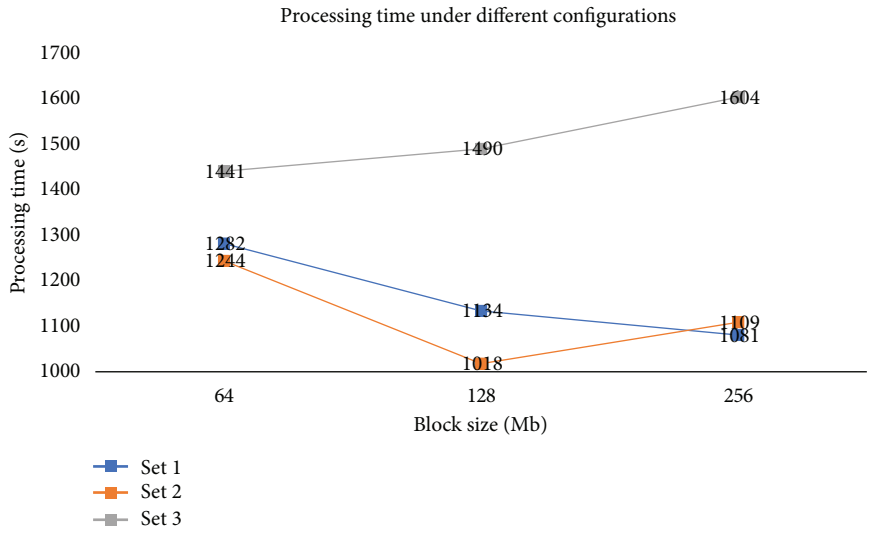

Figure 4: Processing time of large-size Landsat dataset under different configurations.

The detailed data size and processing time are shown in Table 2. The processing time is recorded under configurations with default block size, 50 executors with 20 cores each, and 20 G executor memory. Note that Spark can offer highly efficient processing for remote sensing images. Especially for a relatively large cluster, a sufficient computing resource can support the entire distributed computing process, ensuring that the processing time does not significantly increase as the data size increases. From the perspective of algorithm design, by anatomizing the overall processing time in each operation time-consuming segment, the joining operations are found to be very time-consuming, especially for a dataset with large pixels. In contrast, mapping operations for classification and change detection are much faster in comparison to joining operations. Hence, designing an image processing algorithm with fewer join operations and appropriate partitioning to reduce the data block moving may enhance the performance. Besides, repartitioning operations should also be avoided because it will result in a very time-consuming shuffle process.

Except enhancing the overall performance from algorithm design, the second experiment is developed to further study if the execution configurations may effect on the processing performance. Only the large-size Landsat 8 dataset is applied in this experiment to assess the processing time with different configurations. According to Spark performance tuning official documents [37], the executor numbers, executor, cores and executor memory are three main factors that may effect on performance of Spark-based applications. Budgeting available computing resources in advance is usually needed for users to gain satisfying processing performance. Here, three different configuration sets are assigned as follows: (1) 10 executors with 100 G memory and 100 cores each as set 1, (2) 50 executors with 20 G memory and 20 cores each as set 2, and (3) 500 executors with 2 G memory and 2 cores as set 3. Those three configuration sets are designed with the same total computing vcores and memory. It is also worth to point out that block size may sometimes effect on performance as well. As discussed above, repartition operations (especially for increase partitions) usually should be avoided to eliminate unnecessary shuffle process. Under this circumstance, partitions will be mainly decided by block size during I/O process when RDDs are created. If the block size were too small, massive number of partitions will be created especially with very large input dataset, which will lead to increasing the overhead of task management, though coalesce operation may be applied to decrease the partitions sometimes without shuffling in some cases. If the block size were too large, only a few partitions will be created so that not all the cores in the available computing resource can be sufficiently utilized. In other words, the feature of parallelism is not fully exploited to enhance the performance. Here, three different block sizes are set during the experiments. In Figure 4, the performance of processing the dataset under different execution configurations and different block sizes is represented. The shortest run is offered by 50 executors with 20 G memory and 20 cores each under default block size. With the same executor configuration set, the performance shows a bit lower with 64 M block size, which may result from the total task number which is increased with a smaller block size. However, the execution time for each task is not significantly reduced with corresponding settings. The performance of the same configuration set with 256 M block size is also beaten by it with a default block size. The reason is the partitions generated under this block size are too few so that parts of the executors are not active during processing. This problem shows more obviously when applying configuration set 3, because the number of executors is far more than the tasks under this case and too many computing resources are in idle, which leads to the lower performance. The execution configuration set 1 shows very similar performance with set 2 under each block size setting. However, by monitoring the core utilization of set 1, it is lower than it is with set 2. This may present that the increasing number of executors and decreasing the cores per executor may lift the utilization of cores and may

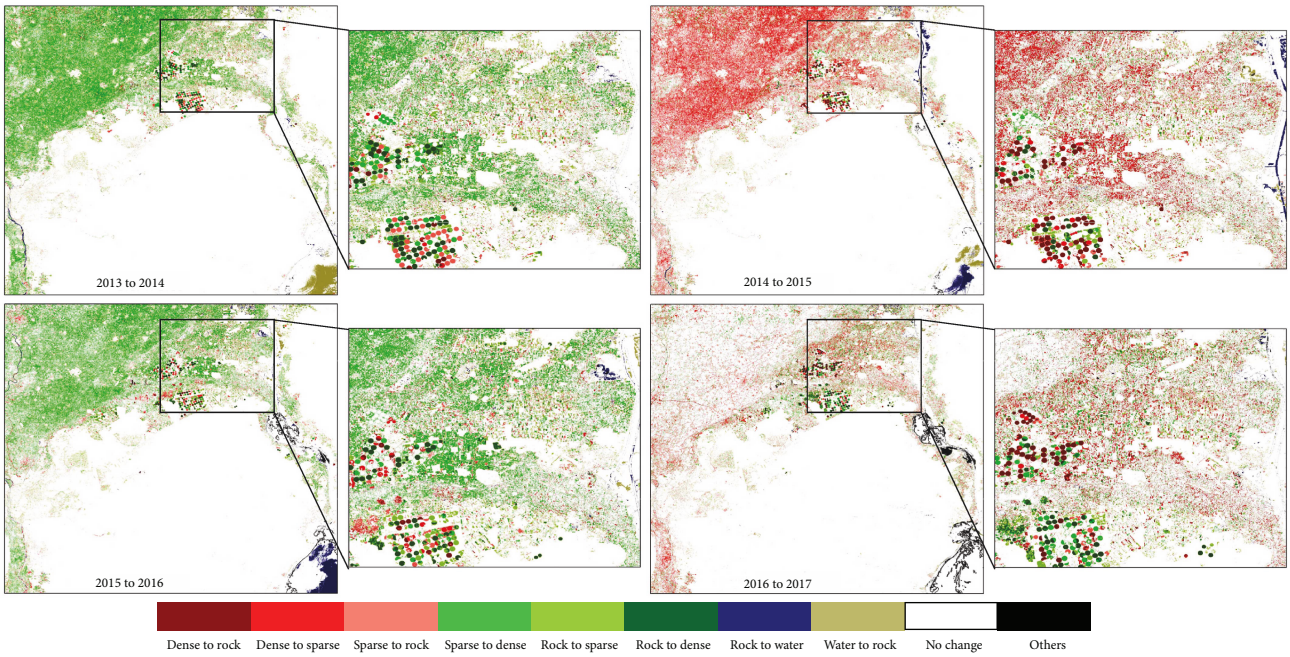

Figure 5: Changing detection visualization results of 2013 to 2014, 2014 to 2015, 2015 to 2016, and 2016 to 2017.

also help enhance the overall performance in some cases especially for very large dataset with relatively limited computing resources.

Here, the land cover feature is defined in four classes: dense vegetation, sparse vegetation, rock/sand, and water. In the legend of Figure 5, colors are set for the changes of each pixel in the study area from one feature to another. For example, "rock to water" indicates that the land cover feature was observed to have changed from sand or rock to water body in the past year. Conditions of no change indicate there are no detected changes in the past year. Others indicate those parts in the images with no data or with erroneous data. In Figure 5, the changes for each pair of images can be clearly seen. Moreover, users can generate a "final" result directly from 2013 to 2017 with a single parameter change. Taking the changing map of 2014 to 2015 in Figure 5 as an example, users can monitor the new branch of the Suez Canal being built and filled with water. It also shows that some new vegetation is growing along the Canal; such vegetation may be a new farmland because most of the vegetation regions are in artificial-like shapes.

As discussed before, the presented tool can be deployed on commercial cloud platform with no change (if the hardware configuration is highly different, strategies of balancing workload may need to be reconsidered). The third experiment is designed to use the proposed framework on Amazon EC2 to process MODIS dataset. Different from Landsat 8 products with band 4 as red, band 5 as NIR and band 3 as green, MODIS (MOD09GA) product sets red band as band 1, NIR as band 2, and green as band 4. Except changing the band index for input, the processing tool is ready to launch with no additional adjustment needed in coding. This experiment run on the 3-node Amazon EC2 cloud computing cluster. By applying 2 executors with 2 cores and 4 G memory each, the experiment is successfully completed in 54 seconds. It is worth to point out that 54-second processing time is not fast with such a small input dataset and that ~0.5 scene MODIS image is only about 50 M. Because in this case, lots of time are occupied by job submission, task management, resource allocation, and so on, the performance of Spark applications can only show significantly with relatively large dataset. However, this experiment still proves the idea that the proposed framework can be deployed on real commercial platforms to process multisource remote sensing images with only minor parameter adjustments.

Those experiments represent a robust solution for constructing remote sensing image processing tools for multiple purposes that are flexible (the ability of the tool to fit multisource datasets in different scales), extensible (the ability of the tool to grow in size and accommodate the volume of computing to resolve), and accessible (the ability of the tool to access data from multiple storage platforms and locations on local resources, data centers, and the cloud). All tested image datasets in those experiments are Landsat 8 with 30-meter spatial resolution and MODIS with 500-meter spatial resolution. However, the reader should note that the input dataset can be any raster-based images in different spatial, temporal, and spectral resolutions, because the algorithm implemented in our experiment is a pixel-based processing algorithm. Following a similar mechanism, all pixel-based algorithms can be implemented by processing the DN values of each pixel to generate required results under this framework. Hence, regardless of applying it to an existing large amount of data or filling it with a next-horizon dataset that will be collected in the future with current sensors or new sensors, this solution is capable of handling the tasks with only minor change in coding, thereby saving the high cost usually invoked in reprogramming different tools for different research goals.

With the Spark-based implementation, as demonstrated in the experiment, the main structures of this tool do not require modification as the volume of dataset changes. For example, it is becoming possible that, even if a Landsat 8 dataset covering a whole year (47.33 Tb) [6] is involved in processing at the same time, by the support of cloud platforms, users can always gain sufficient computing resources (memory especially, for Spark) in theory. Consider that Amazon EC2 now provides the "x1e.32xlarge" instance, which contains 128 virtual CPUs, ~4 Tb memory, and ~4 Tb storage. Users can apply fewer than 20 instances to implement in-memory computing with Spark-based approaches for this dataset in many different processing purposes. Nevertheless, to maintain the high performance of the tool, the partitioning strategies should be tailored based on real execution environments, especially for clusters with unevenly distributed computing resources and networking performance.

As discussed in the introduction, Spark-based approaches can easily access HDFS, Amazon S3, Cassandra, and HBase. Benefitting from cloud storage and management development, an increasing amount of data has been stored in the cloud and is open to the public. The approach as demonstrated provides cloud-based data resources to support users in performing a "pure" cloud analysis and in creating new products from it, without transferring unnecessary original and intermediate data to local storage before the final results are generated. This solution can also support data from multiple sources at the same time. For example, users can access cloud-stored public data as part of their data source and can also access and load their private data stored on local HDFS in a single Spark application.

4. Conclusions

In this paper, authors argue that the current state of the art for big data remote sensing, involving massive datasets being generated from existing and next-generation satellites and observation platforms, is, in many cases, proceeding at paces that outstrip our analytical capabilities to keep up with information products atop those data. While data and analysis are out of alignment, researchers perhaps miss opportunities to build the necessary science that might otherwise be attainable if data and analysis could be better connected. In this paper, authors discussed the current widely used approaches that have been developed by existing studies as a means to cater to the call of the community for the development of an effective and efficient large-scale processing framework to process large-volume remote sensing datasets. Based on the reviews of a comparison with other possible approaches, using Spark to build a robust scalable tool on a cloud environment is possibly an important and practical option to match data with analysis at pace. To this end, a Spark-based multiscale large remote sensing classification and change detection tool has been introduced, and its successful deployment and experimental testing in a cloud environment have been shown.

The approach in this paper suggests several promising advantages. First, the scheme for Spark-sensing offers considerable flexibility for processing big remote sensing datasets in multispatial, multispectral, and multitemporal cases. Indeed, shifting between resolutions and spectrums is possible with slight adjustment, thereby significantly saving the time cost of reprogramming brand new toolkits for different purposes. Second, this scheme makes it possible to exploit the benefits of cloud platforms to gain (theoretically) unlimited computing resources, with highly efficient performance. Third, the presented approach is natively highly accessible to multisource data storage, even in the cloud, which is useful in reducing data transformation costs.

The tool discussed in this paper is obviously just a prototypical framework. Thus, significant improvements could be made. The work here serves to prove the general principle and mechanisms necessary to get going with experiments in this area, and hopefully it can encourage others in the community to build on this foundation. An obvious extension of our approach could include the implementation of more complex remote sensing image processing algorithms, especially in classification, to better match the real cases in different research areas. Another improvement could be explored in designing better partitioning strategies to further enhance the computing performance. Moreover, using a dataset from the cloud directly may reduce the unnecessary data transferring from the data source to the cloud environment, which may better fit the usage habits in real-world problem solving.

Data Availability

The Landsat 8 surface reflectance data used to support the findings of this study have been deposited in the Google Earth Engine repository (https://explorer.earthengine.google.com/#detail/LANDSAT%2FLC08%2FC01%2FT1_SR). The MODIS/Terra Surface Reflectance data used to support the findings of this study have been deposited in the Google Earth Engine repository (https://explorer.earthengine.google.com/#detail/MODIS%2F006%2FMOD09GA).

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

The authors are grateful to Professor Huy T. Vo in the Department of Computer Science and Engineering at New York University for his advice in building the partitioning components of the work.

References

[1] M. F. Goodchild, H. Guo, A. Annoni et al., "Next-generation digital earth," *Proceedings of the National Academy of Sciences*, vol. 109, no. 28, pp. 11088–11094, 2012.

[2] C. Yang, M. Yu, F. Hu, Y. Jiang, and Y. Li, "Utilizing cloud computing to address big geospatial data challenges," *Computers, Environment and Urban Systems*, vol. 61, pp. 120–128, 2017.

[3] C. A. Lee, S. D. Gasster, A. Plaza, C. I. Chang, and B. Huang, "Recent developments in high performance computing for remote sensing: a review," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 4, no. 3, pp. 508–527, 2011.

[4] M. A. Wulder and N. C. Coops, "Make earth observations open access: freely available satellite imagery will improve science and environmental-monitoring products," *Nature*, vol. 513, no. 7516, p. 30, 2014.

[5] Z. Wang, J. Zhong, H. Lan, Z. Wang, and Z. Sha, "Association analysis between spatiotemporal variation of net primary productivity and its driving factors in Inner Mongolia, China during 1994–2013," *Ecological Indicators*, 2017.

[6] Y. Ma, H. Wu, L. Wang et al., "Remote sensing big data computing: challenges and opportunities," *Future Generation Computer Systems*, vol. 51, pp. 47–60, 2015.

[7] Satellite Imaging Corporation, "WorldView-4 satellite image gallery," March 2018, https://www.satimagingcorp.com/gallery/worldview-4/.

[8] Satellite Imaging Corporation, "Quick Bird satellite sensor," March 2017, https://www.satimagingcorp.com/satellite-sensors/quickbird/.

[9] H. Lan and Y. Xie, "A semi-ellipsoid-model based fuzzy classifier to map grassland in Inner Mongolia, China," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 85, pp. 21–31, 2013.

[10] M. A. Friedl, D. K. McIver, J. C. F. Hodges et al., "Global land cover mapping from MODIS: algorithms and early results," *Remote Sensing of Environment*, vol. 83, no. 1-2, pp. 287–302, 2002.

[11] US Geological Survey, "Sensors-Hyperion," March 2018, https://eo1.usgs.gov/sensors/hyperion.

[12] W. Turner, S. Spector, N. Gardiner, M. Fladeland, E. Sterling, and M. Steininger, "Remote sensing for biodiversity science and conservation," *Trends in Ecology & Evolution*, vol. 18, no. 6, pp. 306–314, 2003.

[13] Q. Huang, C. Yang, K. Benedict, S. Chen, A. Rezgui, and J. Xie, "Utilize cloud computing to support dust storm forecasting," *International Journal of Digital Earth*, vol. 6, no. 4, pp. 338–355, 2013.

[14] E. Amazon, "Amazon web services," 2015, November 2012, http://aws.amazon.com/es/ec2/.

[15] B. Wilder, *Cloud Architecture Patterns: Using Microsoft Azure*, O'Reilly Media, Inc., 2012.

[16] D. Sanderson, *Programming Google App Engine: Build and Run Scalable Web Apps on Google's Infrastructure*, O'Reilly Media, Inc., 2009.

[17] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[18] Apache Software Foundation, "Apache Spark," April 2018, https://spark.apache.org/.

[19] M. Guller, *Big Data Analytics with Spark: a practitioner's Guide to Using Spark for Large Scale Data Analysis*, Springer, 2015.

[20] G. Cavallaro, M. Riedel, C. Bodenstein et al., "Scalable developments for big data analytics in remote sensing," in *2015 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, pp. 1366–1369, Milan, Italy, July 2015.

[21] L. Wang, Y. Ma, J. Yan, V. Chang, and A. Y. Zomaya, "pipsCloud: high performance cloud computing for remote sensing big data management and processing," *Future Generation Computer Systems*, vol. 78, pp. 353–368, 2018.

[22] X. Chen and L. Zhou, "The remote sensing image segmentation mean shift algorithm parallel processing based on MapReduce," in *International Conference on Intelligent Earth Observing and Applications 2015, 98083T*, Guilin, China, December 2015.

[23] R. Giachetta, "A framework for processing large scale geospatial and remote sensing data in MapReduce environment," *Computers & Graphics*, vol. 49, pp. 37–46, 2015.

[24] Z. Sun, F. Chen, M. Chi, and Y. Zhu, "A spark-based big data platform for massive remote sensing data processing," in *ICDS 2015 Proceedings of the Second International Conference on Data Science*, pp. 120–126, Sydney, Australia, August 2015.

[25] W. Huang, L. Meng, D. Zhang, and W. Zhang, "In-memory parallel processing of massive remotely sensed data using an apache spark on Hadoop YARN model," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 10, no. 1, pp. 3–19, 2017.

[26] X. Fan, B. Lang, Y. Zhou, and T. Zang, "Adding network bandwidth resource management to Hadoop YARN," in *2017 Seventh International Conference on Information Science and Technology (ICIST)*, pp. 444–449, Da Nang, Vietnam, April 2017.

[27] C.-Z. Qin, L.-J. Zhan, and A.-X. Zhu, "How to apply the geospatial data abstraction library (GDAL) properly to parallel geospatial raster I/O?," *Transactions in GIS*, vol. 18, no. 6, pp. 950–957, 2014.

[28] J. W. Rouse Jr., R. H. Haas, J. A. Schell, and D. W. Deering, "Monitoring vegetation systems in the Great Plains with ERTS," in *NASA. Goddard Space Flight Center 3d ERTS-1 Symp*, College Station, TX, USA, January 1974.

[29] S. K. McFeeters, "The use of the normalized difference water index (NDWI) in the delineation of open water features," *International Journal of Remote Sensing*, vol. 17, no. 7, pp. 1425–1432, 1996.

[30] A. C. Frery and T. Perciano, "Image data formats and color representation," in *Introduction to Image Processing Using R. SpringerBriefs in Computer Science*, pp. 21–29, Springer, London, 2013.

[31] US Department of the Interior, and US Geological Survey, "Landsat 8," April 2018, https://landsat.usgs.gov/landsat-8.

[32] Amazon Web Services Inc, "Landsat on AWS," March 2018, https://aws.amazon.com/public-datasets/landsat/.

[33] Amazon Web Services Inc, "MODIS on AWS," June 2018, https://docs.opendata.aws/modis-pds/readme.html.

[34] Suez Canal Authority, "New Suez Canal," June 2017; https://www.suezcanal.gov.eg/English/About/SuezCanal/Pages/NewSuezCanal.aspx.

[35] N. Gorelick, M. Hancher, M. Dixon, S. Ilyushchenko, D. Thau, and R. Moore, "Google Earth Engine: planetary-scale geospatial analysis for everyone," *Remote Sensing of Environment*, vol. 202, pp. 18–27, 2017.

[36] P. S. Chavez, "Image-based atmospheric corrections-revisited and improved," *Photogrammetric Engineering and Remote Sensing*, vol. 62, no. 9, pp. 1025–1035, 1996.

[37] Apache Software Foundation, "Tuning Spark," June 2018, http://spark.apache.org/docs/latest/tuning.html#tuning-spark.