International Journal of Innovative
Computing, Information and Control
Volume **13**, Number **6**, December **2017**

ICIC International ©2017 ISSN 1349-4198
pp. 1915–1928

EFFICIENT STORAGE METHOD FOR MASSIVE REMOTE SENSING IMAGE VIA SPARK-BASED PYRAMID MODEL

Mengzhao Yang[1], Haibin Mei[1], Yuhao Yang[1] and Dongmei Huang[1,2,*]

[1]College of Information
[2]Digital Ocean Research Institute
Shanghai Ocean University
No. 999, Huchenghuan Rd., Nanhui New City, Shanghai 201306, P. R. China
mzyang@shou.edu.cn; *Corresponding author: dmhuang@shou.edu.cn

Received May 2017; revised September 2017

Abstract. *The amount of remote sensing (RS) image has grown considerably and the Big Data era of images has truly emerged. Traditional storage and processing methods cannot meet the requirements of the growing massive image data. So how to provide effective storage and high-performance computation service has brought a great challenge to the research community. In this paper, we present the efficient storage of massive remote sensing image via Spark-based Pyramid model. Based on selection of rational number of image layer via the maximum division layer algorithm, we design the Spark-oriented distributed construction method. We use RDDs (Resilient Distributed Datasets) on Spark system to represent data structures for metadata and ranks, and there is no need to materialize these data structures across multiple iterations. Using our construction method and RDDs caching technology, we design the efficient storage structure of RS image data, which can achieve high-performance processing of massive RS image on Spark system. Experimental results show that proper values can be obtained via optimization parameters of image tile. Also throughput rate and construction performance are much higher than MapReduce and Hadoop.*

Keywords: Remote sensing image, Efficient storage, Pyramid model, Spark system

1. **Introduction.** Nowadays, the rapid development of remote sensing technology has proliferated high-quality images that occupy larger and larger storage spaces. The remote sensing community has recognized the challenge in processing large and complex satellite datasets to derive customized products. Traditional image processing applications are inadequate, so challenges including efficient storage and high-performance computation service are produced and have to be solved urgently. Recently, cloud computing technologies [1, 2] such as MapReduce programming and Hadoop system are the open-source software framework for efficient storage and distributed processing of massive image data, which can resolve these problems and have become a research hotspot.

Several efforts have been made in the past few years towards efficient storage and processing via high-performance cloud computing technologies. MapReduce is a great parallel programming framework, and is suitable for handling Big Data such as massive RS images. Firstly, for high-performance computing services, MapReduce can split the large image datasets for distributed processing, which is amenable to a broad variety of real-world tasks [3, 4]. Then based on MapReduce programming, parallel processing of remote sensing images is presented and used to solve some problems within the field of remote sensing application such as detection, classification and retrieval [5, 6, 7, 8]. Although MapReduce is an efficient solution to big data problem, there are a lot of limitations. MapReduce is only a computing architecture, and has no sharing and storing

files system. Then, Hadoop system is presented based on the MapReduce programming model, which can provide processing, storage, and analysis of large amounts of distributed and unstructured data [9]. Sharing, storing, and retrieving large files on a Hadoop cluster can be undertaken by its distributed file system called HDFS (Hadoop Distributed File System) [10]. Over recent years, Hadoop system has become a highly popular solution to store and process a large amount of image data for analysis purpose. Cloud storage from HDFS [11, 12] can increase the storage utilization ratio and improve the storage performance by being aware of the quality of service of the DataNodes. Rajak et al. [13] present the MapReduce framework for performing remote sensing data and storing the output in HBase. The speedup and performance of their tests are shown by uilizing Hadoop. Also extending MapReduce programming model [14], better performance tests for processing large archives of Landsat images are performed with the Hadoop framework. Later, based on HDFS storage method, classification and detection of large-scale RS image [15, 16, 17] are implemented and speedup using these methods is improved greatly. Recently, Park et al. [18] explore in-storage computing challenges and opportunities for the Hadoop, and integrate a MapReduce system with devices of solid state drives (SSD) to achieve a remarkable performance gain. On the other hand, massive video data can also be stored efficiently by making use of HDFS [19], and the proposed approach can detect moving objects and provide the accuracy coordinates via MapReduce for data intensive computing. However, MapReduce and Hadoop system are suitable only for batch processing jobs, and they do not do well for graph, iterative, incremental and many other kinds. They read and write from disk and that slows down the processing speed. Thus implementing interactive jobs and models in some special applications becomes impossible due to the huge space and I/O consumption by frequent accessing jobs. Verma and Patel [20] have discussed the working model and the programming frameworks via experimental analysis, and found that Spark is many times faster than Hadoop HDFS on single node implementation.

Spark runs on top of existing HDFS infrastructure to provide enhanced and additional functionality. It takes MapReduce to the next level with less expensive shuffles in the data processing. With capabilities like in-memory data storage and near real-time processing, the performance can be several times faster than other Big Data technologies. Traditionally, it runs applications in Hadoop clusters up to 100x faster in memory and 10x faster on disk [21]. By reducing number of read/write cycle to disk and storing intermediate data in-memory, Spark makes it possible for iterative machine learning algorithms and incremental applications. Multiple K-means algorithms [22, 23] have been designed on Spark platform to prove that they are popular for fast processing particularly where iterations are involved. By incorporating strips of RS data with RDDs of Spark, Huang et al. [24] simultaneously introduce in-memory parallel processing of massive RS data and take a multi-tasking algorithm to indicate its great efficiency. More recently, in the efficient storage field on Spark, an application-attuned dynamic data storage system is presented [25], which aims to offer an affordable and efficient storage tier management. Based on RDD and in-memory clustering computing model, the storage mechanisms of memory-based distributed RDD [26, 27] are proposed, and dynamic space allocation strategy is designed. They balance the asymmetric memory requirement among execution containers, and maximize the probability of the in-memory caching of RDD data so as to improve the performance. Especially, in the Geo-spatial information science, Yu et al. [28] demonstrate GeoSpark as an efficient cluster computing framework for storing and processing large-scale spatial data at interactive performance. However, the Spark default serialization strategy has low utilization of cache which has greatly influenced the efficiency of Spark task execution. For solving this problem of low computational efficiency caused by

insufficient memory, Xia and Yang [29] propose an optimized serialized storage strategy, which combine with the running cost of RDD and count of Action. Nowdays, the size of one scene of satellite image data is even a few GB, and for example, mosaic remote sensing images may be more than 10GB or 50GB [10]. If we divide the large size RS image into too many small files, HDFS has to be modified to provide better storage performance. How to achieve efficient storage of massive metadata to fit Spark system is still a great challenge. Image Pyramid is a hierarchy model which is simple but effective to represent the large-scale satellite image at different level of details (LOD). In this paper, we focus on the construction of Spark-based Pyramid model to realize efficient storage of RS image. Using the maximum division algorithm, we evaluate the rational number of layer for image Pyramid construction. Based on RDD caching technique, we design the Spark-oriented image Pyramid model, which can meet the requirements of the efficient construction. Using our construction method, we realize encoding of image tiles and efficient storage of RS image data. The results show that throughput rate and construction performance via our method are much higher than traditional cloud computing system such as MapReduce and Hadoop platform.

The rest of this paper is organized as follows. The Pyramid model to represent remote sensing image is introduced in Section 2. Resilient distributed datasets caching technology is described in Section 3. Spark-oriented image Pyramid model is designed in Section 4. Results are compared in Section 5 and conclusions are drawn in Section 6.

2. **The Pyramid Model to Represent Remote Sensing Image.** In general, the size of one satellite image data is three or four hundred MB, and even a few GB (such as True Marble image data, each file is 1.5GB). In order to process the huge amount of data quickly such as image view, image enhancement and image editing, the general hardware configuration cannot meet the requirements. Image Pyramid model is an effective way to solve this problem, and its core idea is to block and stratify a large-scale RS image.

Image Pyramid model is a hierarchy model to represent large-scale satellite image at different LOD. The following is an example of a four-level Pyramid as shown in Figure 1. The Pyramid level refers to the number of reduced resolution datasets created when the Pyramid is built. The base of the Pyramid is the original RS image whose resolution is 1024×1024. Up the pyramid, the resolutions become smaller which are 512×512, 256×256, and 128×128 at the top of the pyramid. It can be concluded that the larger the resolution of one image is, the more levels the pyramid that is built for the image has.

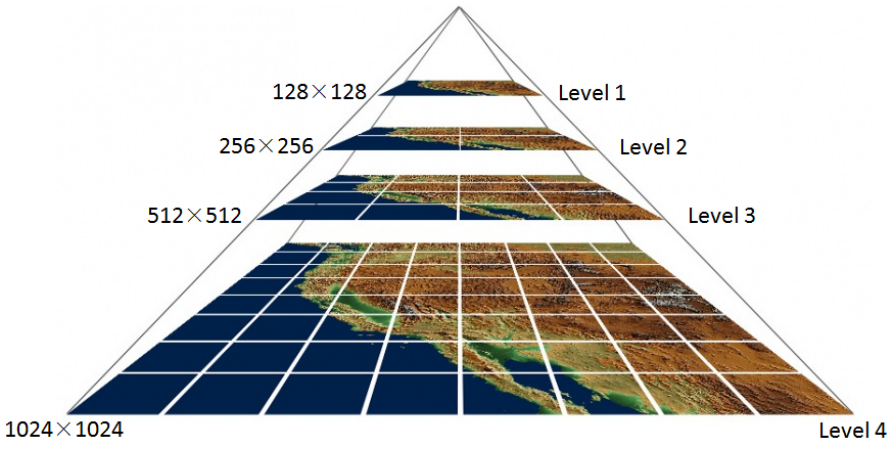

FIGURE 1. Pyramid model of an image

From Figure 1, the image of the Pyramid model is segmented into rectangular tiles, where all tiles have the same pixel dimensions, e.g., 128×128 pixels. A tile at one level of the Pyramid will therefore map onto many tiles on the immediately higher resolution level. That is, the tiles at the higher resolution level cover part of the geographical area of the former. Using this representation, it is possible to recursively resolve certain regions of a dataset in more detail than other regions.

3. **Resilient Distributed Datasets Caching Technology.** Spark uses RDD which implements in-memory data structures used to cache intermediate data across a set of nodes. Since RDD can be kept in memory, algorithms can iterate over RDD data many times without accessing from disk. There are two ways to create RDD: parallelizing an existing collection in your driver program, or referencing a dataset in an external storage system, such as a shared filesystem, HDFS, HBase, or any data source offering a Hadoop inputting format.

RDDs support two types of operations: *transformations*, which create a new dataset from an existing one, and *actions*, which return a value to the driver program after running a computation on the dataset. For example, *Map* is a *transformation* that passes each dataset element through a function and returns a new RDD representing the results. On the other hand, *Reduce* is an *action* that aggregates all the elements of the RDD using some function and returns the final result to the driver program (although there is also a parallel *reduceByKey* that returns a distributed dataset).

When you persist an RDD, each node stores any partitions of it that it computes in memory and reuses them in other *actions* on that dataset (or datasets derived from it). This allows future *actions* to be much faster (often by more than 10x). Caching is a key tool for iterative algorithms and fast interactive use. RDD caching process is shown in Figure 2.

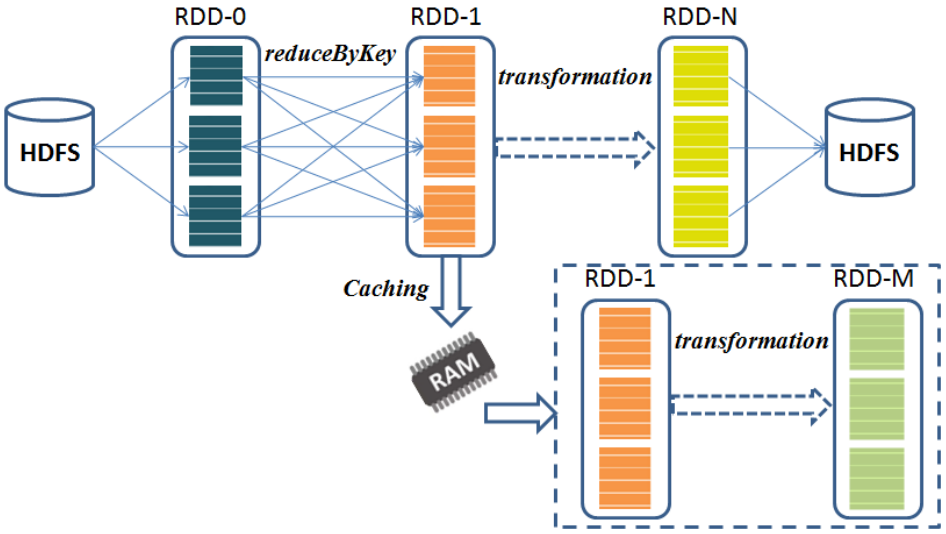

FIGURE 2. RDD caching process

By default, each transformed RDD may be recomputed each time you run an action on it. However, in this paper we persist an RDD in memory using the caching method, in which case Spark will keep the elements around on the cluster for much faster access the next time you query it. As shown in Figure 2, after a series of RDD-1 transformation, RDD-N is got finally and stored into HDFS. In this transformation process, RDD-1 will have an intermediate result. If the result is cached to memory, it will not calculate its

previous RDD-0 when implementing transformation from RDD-1 to RDD-m, which leads to substantial speedups on future reuse and computation.

4. Spark-Oriented Image Pyramid Model.

4.1. Construction flow of Spark-oriented image Pyramid.

Traditional image Pyramid construction is based on sampling operators to re-sample the images, which brings a large amount of calculation and cannot control the image layering flexibly. However, Spark-oriented image Pyramid is a more efficient model that utilizes the Pyramid organization, elastic datasets RDD and caching technology to deal with massive RS data. We consider both layering and blocking for the image in the construction of image Pyramid. According to the resolution of original RS image, the maximum number of layers is obtained by the maximum division layer algorithm (MDLA). Meanwhile, original RS image is divided into equal size tiles, and blank spaces of them are filled with null values. Based on the maximum number of layers and tile metadata structures of each layer, tile caching mechanism of multi-resolution hierarchical image is built finally. Figure 3 shows the construction flow of Spark-oriented image Pyramid.

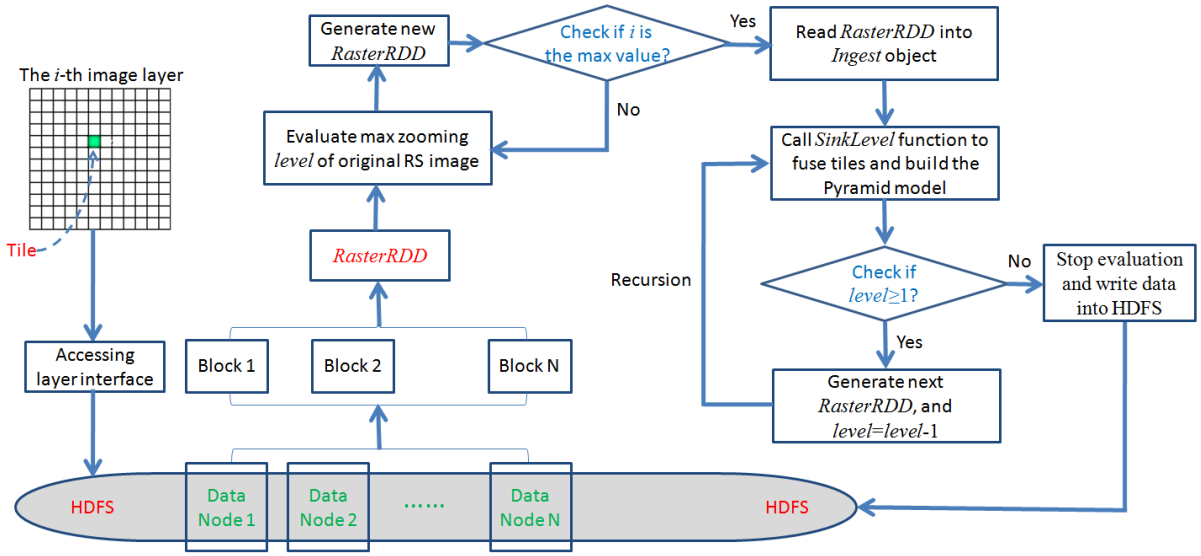

FIGURE 3. Construction flow of Spark-oriented image Pyramid

From Figure 3, after the original image data is read into library, it will be blocked and stored in different storage DataNodes of HDFS. The different block data can be processed as different RDDs, and further transformed to *RasterRDD* which can be directly used by Spark. The new generated *RasterRDD* is the image data of a new layer and also is the input data in the next layer. In this process, image data of each layer is blocked according to *Quadtree* structure. After the maximum number of layers *level* is acquired, we pass two parameters *level* and *RasterRDD* into the *Sinklevel* function of Ingest object to build the Pyramid model. From the minimum resolution at the top of the image, image tiles will be reconstructed and fused with the decrease of level value. When $level \leq 1$, the generated attribute data *AttributeData* and net data *NetData* in the building Pyramid process are finally written into the HDFS.

4.2. Spark-oriented distributed construction method.

Before using remote sensing image data, it must be processed by each image for its large amount of data. After processing, the size of the original image is not too large and the number of level is generally not more than 20 layers when constructing the image Pyramid model. In this

paper, the zoom maximum level of the remote sensing image is influenced by the resolution of the original image. Meanwhile, considering different width and height of image sizes, the maximum number of layers needs to be calculated by two aspects: width and height. The final number of layers is selected as the maximum value from two aspects. The maximum division layer algorithm is described in Algorithm 1.

Algorithm 1 Maximum division layer algorithm

Require: coordinate range of original remote sensing image $Extent$; pixel value of remote sensing image $cellSize$; image tile slice size $tileSize$.

Ensure: final number of division layers $level$.

1: Evaluate pixel size of remote sensing image: $W_0 \leftarrow Extent.width$, $H_0 \leftarrow Extent.height$;
2: Evaluate resolution of the ith layer ($1 \leq i \leq 20$): $Res_w = W_i/(2^i \times titleSize)$;
3: Stop evaluation and achieve the value i as maximum value of division layers, if $cellSize.width + k \geq Res_w$ (k is a correction constant). Or repeat Step 2, if $cellSize.width + k < Res_w$;
4: Reassign $W_i \leftarrow H_i$, and evaluate Res_h according to Step 2 and Step 3;
5: Evaluate the maximum value $level \leftarrow \max(Res_w, Res_h)$;
6: **Return** final number of division layers $level$.

From Algorithm 1, we firstly evaluate the width and height of image according to coordinate range of original RS image. Then the resolutions Res_w and Res_h in width and height direction are acquired respectively while the terminate condition is checked by comparing $cellSize + k$ and Res. Finally, final number of division layers $level$ is obtained by evaluating $level \leftarrow \max(Res_w, Res_h)$.

Based on the maximum division layer algorithm above, we design the Spark-oriented distributed construction method, which depends on the resolution of the original remote sensing image. The original image is divided by slicing and converted to the data structure $rasterRDD$, which can be directly processed by Spark. Spark-oriented distributed construction method is described in Algorithm 2.

From Algorithm 2, we firstly acquire the range and cell value of inputting RS image. Then final number of division layers $level$ is acquired by using the maximum division layer algorithm $MDLA$. Later $rasterData$ is produced via checking the terminate condition and distributed Pyramid model is established by calling $SinkLevel$ function recursively. Finally, attribute data $attributeData$ and net data $netData$ are produced and written into the $HDFS$.

The data sets in the calculation process can be stored in memory by RDD caching technique, which can reduce the number of accessing data, shorten the image computation time, and improve the construction speed of Pyramid image model. Therefore, it can meet the requirements of the efficient construction of image Pyramid model and fast processing.

4.3. **Efficient storage strategy of massive image data.** Traditional database storage platforms such as ArcSDE or Oracle are quite complicated for massive RS image. We use HDFS to store and manage image tiles, and do not need to build additional indexes information, which can greatly improve accessing efficiency of massive images. Here we mainly study the efficient storage way that is based on the distributed file system in Spark. After splitting and blocking the original image, the image layer of Pyramid model is generated by stitching it, and the pixels in each image layer are mapped to the original image according to the mapping algorithm. In this paper, the tile data of each partition image is not stored separately when processing image partition, but it is implemented

Algorithm 2 Spark-oriented distributed construction method

Require: data source of remote sensing image *source*; image tile slice size *tileSize*.
Ensure: final number of hierarchical layers *level*; attribute data *attributeData*; meta data *metaData*;

1: Read the partitioned image data into Spark and acquire range of remote sensing image $Extent$;
2: Evaluate cell value of remote sensing image $cellSize = Extent.width/cols$;
3: Divide layers using the maximum division layer algorithm $MDLA\ (tileSize, cellSize, Extent.width)$;
4: Acquire image resolution Res at each layer and the number of division layer i;
5: If the terminal condition is satisfied, return i and implement Step 6, else $i = i + 1$, and produce $rasterData$ in the next layer and return Step 4;
6: Ingest data of current layer to Spark and input two parameters: $level = i$, $rdd = rasterRDD$;
7: Call $SinkLevel$ function to establish the distributed Pyramid model and judge whether the *level* is larger than the terminate condition;
8: If *level* meets the condition, call $SinkLevel$ function to produce next $rasterRDD$ and $level = level - 1$. Then call $Sinklevel$ recursively;
9: If *level* does not meet the condition, the calculation is stopped. The produced attribute data $attributeData$ and net data $netData$ are written into the $HDFS$;

with distributed storage for each layer of image after its partition. So tile data of a few layers will be stored into one or more DataNodes on Spark system. Then the range of image tile data is determined by encoding value $SpatialKey$ and level number i. Finally, data values in the metadata tiles are obtained. Figure 4 shows the coding process of the Pyramid structure of image tiles.

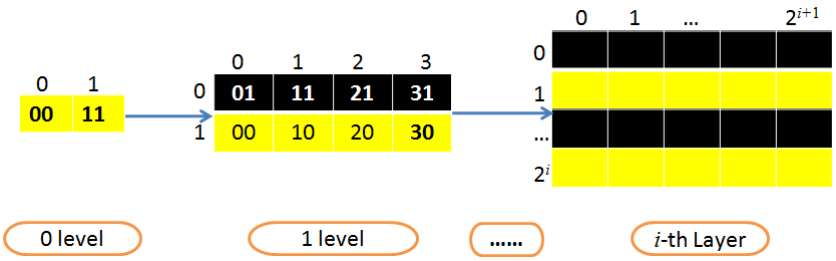

FIGURE 4. Image tiles coding process

From Figure 4, we use the TMS (Tile Map Service) [30] algorithm to code the remote sensing image tiles. TMS is the specification which is established by OSGeo (Open Source Geospatial Foundation). The basic principle is cutting the map into many tiles in advance, and storing tiles on the server according to the layer level. The hierarchical images can be obtained by exploiting Quadtree method later. In this paper, the easy URL way is used by TMS to request image tiles, and the request way with multiple parameters is not considered here. Given an $http : //URLTyphoonLayerName/z/x/y.png$, URL is the server net address, $TyphoonLayerName$ is the layer name of Typhoon, z represents the zoom level of image, and (x, y) shows the image tiles coordinates.

In the coding process, tile encoding value is regular in Cartesian coordinates system and projection algorithm is also introduced into programming. So an accurate index relationship is established between the encoding value and coordinates. Through latitude and longitude of coordinates, tile encoding value can be obtained. On the contrary,

longitude interval can be also found via encoding value. This mapping process includes following computation functions.

$$n = 2^{zoom} - 1 \tag{1}$$

where $zoom$ is the level number of current layer.

Then x and y coordinates of image tile: $xtile$ and $ytile$, can be acquired by the accurate index relationship as follows.

$$xtile = \left(\frac{lon\_deg + 180}{360}\right) \times n \tag{2}$$

$$ytile = \frac{(1 - \log(\tan(lat\_deg) + \sec(lat\_deg))/\pi)}{2} \times n \tag{3}$$

where lon_deg is the degree of longitude. lat_deg is the degree of latitude.

On the contrary, using tile coding value, values of longitude and latitude can also be got by Equation (2) and Equation (3). The computation process is as follows.

$$lon\_deg = \frac{xtile}{n} \times 360 - 180 \tag{4}$$

$$lat\_deg = \arctan\left(\sinh\left(\pi \times \left(1 - \frac{2 \times ytile}{n}\right)\right)\right) \tag{5}$$

Therefore, in the process when constructing Pyramid model on Spark system, we can determine the loaded image tiles according to the central coordinates and level number of original remote sensing image. Similarly, when clicking on base map, we can also determine the range of latitude and longitude according to the encoding value of image. Finally coordinates of one pixel can be exactly fixed on the image tile.

The Pyramid model in this paper is constructed based on Spark framework, and its purpose is to accelerate the process of image processing by Spark in-memory computing. We process the original remote sensing image as computable data organization and store it in HDFS. Two types of data structures are produced after processing. One class data is attribute data of JSON file format, and the other data is net data without head information as shown in Figure 5.

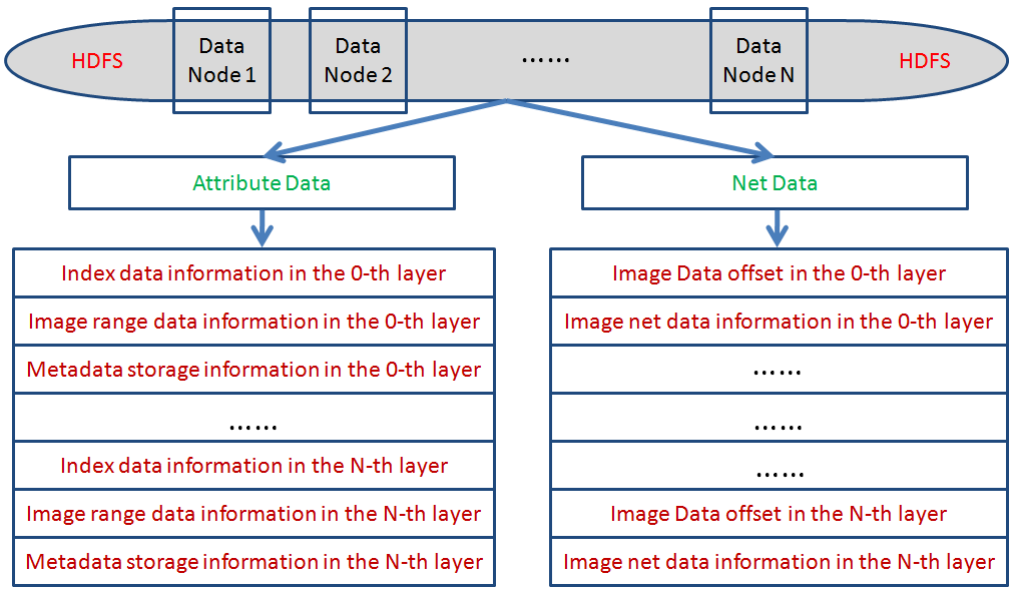

FIGURE 5. Two types of data for distributed storage

From Figure 5, the attribute data includes index data, image range data and metadata, which is respectively stored in the 0th layer to the Nth layer. Net data includes data offset and image net data information, which is respectively stored in the 0th layer to the Nth layer. When constructing the image Pyramid model, these image data will be stored in each node of HDFS.

5. Prototype System and Experiment Results.

5.1. Design of system framework. In our experiment, 8 Inspur servers are deployed in OpenStack cloud platform. Using virtualization technology of OpenStack, we set up 2 control nodes of NameNode, SecondaryNameNode and 5 Worker nodes in the Minicomputer cluster. Server nodes are connected to each other by Gigabit Ethernet card and each node contains 8 cores and 16G computing memory. Our cloud platform is equipped with Ubuntu 14.04 system. Open source software includes Hadoop 2.6.3 and Spark 1.5.2. Based on the construction algorithm of image Pyramid model, we develop interface library of algorithm, and realize the fast layering, slice blocking and parallel construction of image Pyramid model. The overall architecture of the experiment is shown in Figure 6.

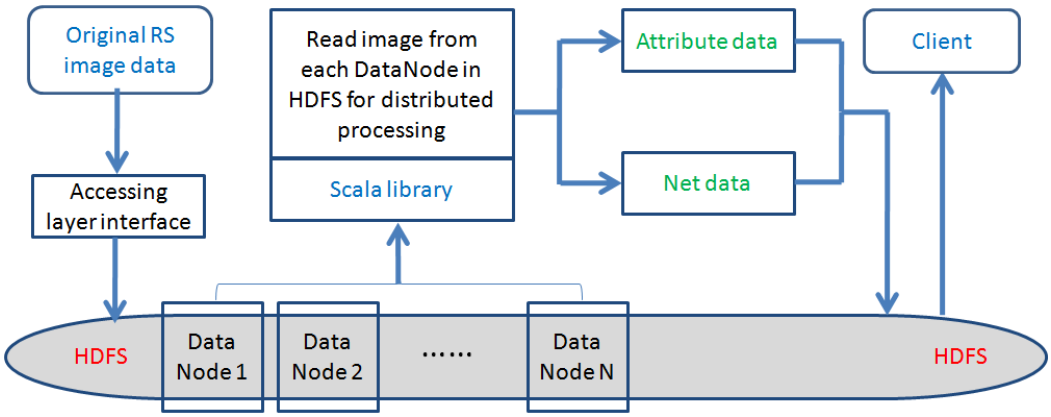

FIGURE 6. Design of our system framework

From Figure 6, design and implementation of our system is based on the distributed file storage system. In the actual operation, firstly the large amount of remote sensing data is stored in HDFS via accessing layer interface. Then, using the developed interface library, remote sensing data is read into the Spark cluster for distributed computing. Finally, the remote sensing image is processed and classified into two types of data, and restored in the distributed file system for sharing and rendering in all the clients. In the process of hierarchical and block calculation of the Spark mode, the generated intermediate image data can be cached into the memory via Spark RDD caching method. So they can be directly invoked by the client until the intermediate result is released by the memory. There is no need for multiple I/O operations, which is the greatest advantage of the Spark calculation mode.

5.2. Parameters optimization and selection of image tile. The performance results are greatly influenced by two parameters, which are image data size and slice value of image tile (tileSize). In order to verify the advantage of our method, slice value parameter of image tile should be determined firstly. Without considering the transmission efficiency of the LAN network environment, we design a Spark-oriented system of image processing according to the overall structure of the experiment. We select three groups of RS images

as test pattern. These three groups of images are separated to the small size group with images 1GB, the middle size group of images from 4GB, and the big size group of images 8GB. Comparisons of slice time and response time are given by selecting two different parameters: data size and tileSize. The results of response time and slice time (the unit of time is second) are respectively shown in Figure 7 and Figure 8.

tileSize value / Data Size	64	128	256	512	1024
1G	0.57	0.83	1.03	4.1	11.2
4G	0.56	0.85	1.07	4.3	12.1
8G	0.64	1.01	1.19	4.6	12.9

FIGURE 7. Response time when selecting different data size and tileSize value

tileSize value / Data Size	64	128	256	512	1024
1G	16.73	8.32	4.54	3.84	2.66
4G	23.12	17.23	12.61	6.95	3.16
8G	31.65	24.13	17.33	11.36	5.82

FIGURE 8. Slice time when selecting different data size and tileSize value

From Figure 7 and Figure 8, the larger the tile value is, the longer the response time is and the shorter the slice time for the image tile is. At the same time, when the data size of RS image increases, the slice time is increasing, but the response time of the client is basically unchanged in the error range. Taking account of the rapid response requirement of client, we should set value of tileSize parameter as small as possible, but too small block is not easy for the client to give a coherent zoom. In addition, when the data size of remote sensing image is large, it can improve the efficiency of image processing, but it is not easy for the client to render. So we should select equilibrium parameter value to meet both requirement of client response and image processing.

5.3. **Performance comparison of image Pyramid construction.** In order to test distributed construction efficiency of image Pyramid model, we evaluate the throughput rate when layering image and slicing tile, and compare their performance of image Pyramid model using different methods. Based on optimization parameters of image tile, we set $tileSize = 256$ and select large-scale Typhoon images of "Sepat" sent by FY-2C satellite to verify our algorithm.

Firstly, the construction algorithm is respectively applied on Hadoop stand-alone mode and Hadoop distributed mode (using MapReduce programming model). Data size in the experiment is 100MB, 1GB, 10GB, 100GB respectively and comparison results are shown in Figure 9 and Figure 10.

From Figure 9 and Figure 10, when the data size of RS image is small, the throughput rate of stand-alone mode is lower than that of the distributed mode; however, construction performance of its image Pyramid is faster than that of the distributed mode. When the data size is increasing, both the throughput rate and construction performance in distributed model of image Pyramid model are higher.

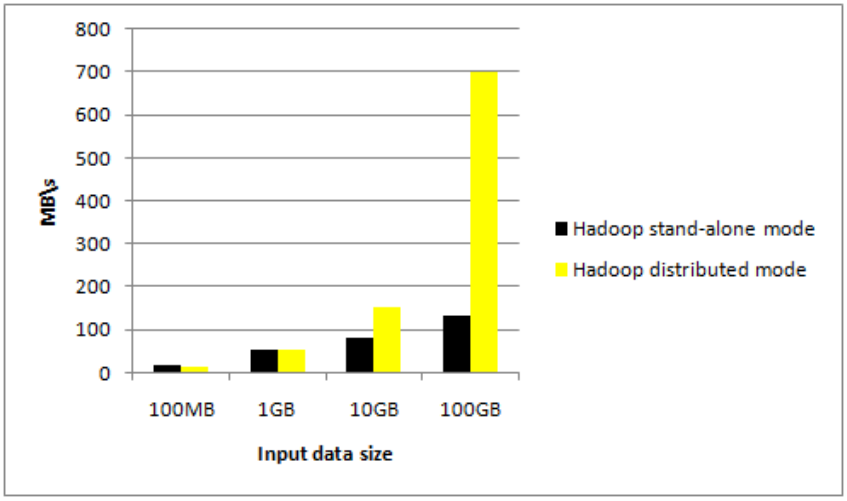

FIGURE 9. Throughput rate comparison between two Hadoop modes

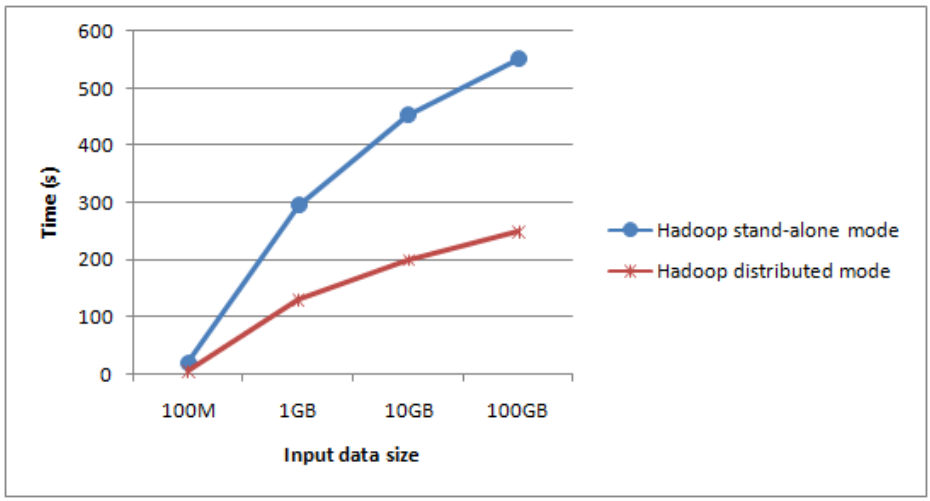

FIGURE 10. Construction performance comparison between two Hadoop modes

Secondly the algorithm is applied in Spark framework and Hadoop MapReduce programming framework respectively, and test parameters in the experiment remain unchanged. In order to show computation efficiency clearly, we increase data size up to 10GB, 100GB, 500GB and 1TB respectively. Experimental results are shown in Figure 11 and Figure 12.

From Figure 11 and Figure 12, when the image data size is small, both throughput rate and construction performance show small difference between two kinds of frameworks in the construction of RS image Pyramid model. However, when the image data size is increasing, throughput rate and construction performance based on Spark are much higher than Hadoop. Especially, when the data size reaches up to 1TB, the throughput rate and construction performance will be improved by $40\% \sim 50\%$ approximately.

Above all, in the first part test as shown in 5.2, we found that the algorithm in the distributed environment is not necessarily better over stand-alone mode, but when the image data size is larger enough, the distributed environment has a faster efficiency. In the second test as shown in 5.3, under two different distributed computing frameworks, we have compared the advantages and disadvantages of the algorithm in the throughput rate and construction performance. The experimental results show that the construction method

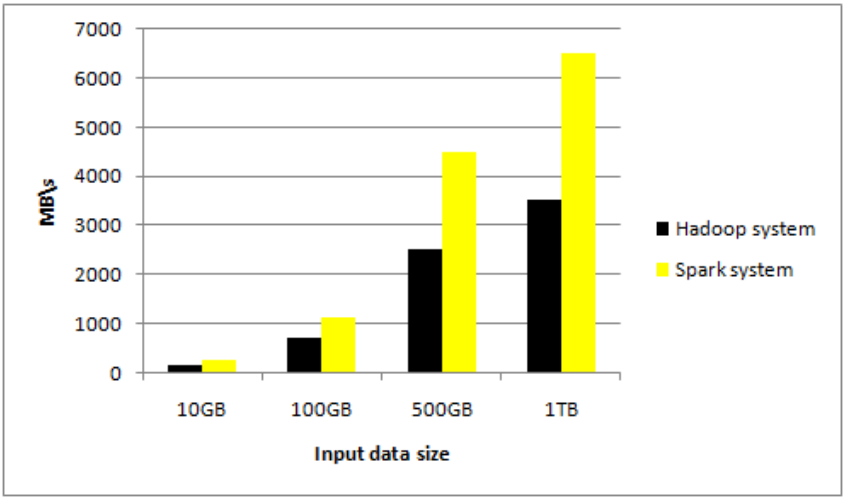

FIGURE 11. Throughput rate comparison between two modes

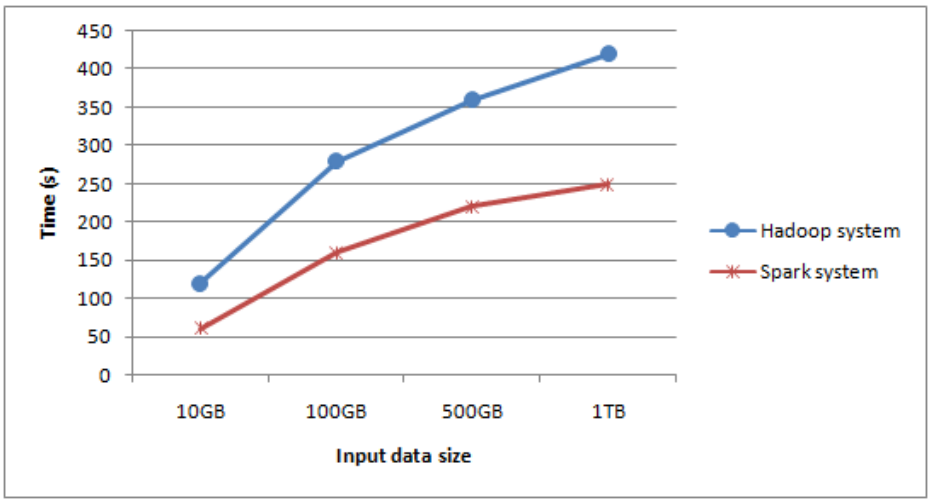

FIGURE 12. Construction performance comparison between two modes

of Spark-based Pyramid model has better efficiency and performance. This is because RDD caching on Spark is much more efficient than other low-level caching approaches such as OS buffer caches and HDFS caching, which can only reduce disk I/O. Especially in the massive remote sensing image, our method can improve efficiency greatly.

6. **Conclusion.** We have implemented an efficient storage method for massive remote sensing image via Spark-based Pyramid model. From experiment and results, we have compared slice time and response time when selecting two different parameters: data size and tileSize. Based on the comparison, we set suitable parameter of image tile and give performance comparison of image Pyramid construction. The results show that throughput rate and construction performance are much higher. Especially, when the data size of remote sensing image is massive, performance of our method is more obvious over Hadoop and MapReduce, which shows the achievement of this proposed method. The main causes of these speedups are in-memory computation via RDD caching method and efficient storage strategy of massive image data.

We have demonstrated that we have achieved some good advantages in throughput rate and construction performance, but many exciting directions remain to be explored. In

future work, we plan on implementing some practical applications such as image detection, image retrieval and image classification based on our storage method of Spark-based image Pyramid model.

Acknowledgment. This work was supported by the National Natural Science Foundation of China (No. 41671431), the Ability Construction Project in Local University of Shanghai Science and Technology Commission (No. 15590501900), the Youth Science and Technology Project of Shanghai Ocean University (No. A2-0203-00-100216), and the Doctoral Start-up Fund for Scientific Research of Shanghai Ocean University (No. A2-0203-00-100346).

REFERENCES

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

[2] D. Fustes, D. Cantorna, C. Dafonte, A. Iglesias and B. Arcay, Applications of cloud computing and GIS for ocean monitoring through semote sensing, *Smart Sensing Technology for Agriculture and Environmental Monitoring, Lecture Notes in Electrical Engineering*, vol.146, pp.303-321, 2012.

[3] J. Dean and S. Ghemawat, MapReduce: Simplified data processing on large clusters, *Conference on Symposium on Operating Systems Design & Implementation*, pp.107-113, 2004.

[4] J. Dean, Experiences with MapReduce, an abstraction for large-scale computation, *International Conference on Parallel Architecture and Compilation Techniques*, pp.1-10, 2006.

[5] E. B. Tesfamariam, *Distributed Processing of Large Remote Sensing Images Using MapReduce – A Case of Edge Detection*, LAP LAMBERT Academic Publishing, 2011.

[6] Z. Lv, Y. Hu, H. Zhong, J. Wu, B. Li and H. Zhao, Parallel K-means clustering of remote sensing images based on MapReduce, *International Conference on Web Information Systems and Mining*, pp.162-170, 2010.

[7] S. Venkatraman and S. Kulkarni, MapReduce neural network framework for efficient content based image retrieval from large datasets in the cloud, *International Conference on Hybrid Intelligent Systems*, pp.63-68, 2013.

[8] M. H. Almeer, Cloud Hadoop map reduce for remote sensing image analysis, *Journal of Emerging Trends in Computing & Information Sciences*, vol.3, no.4, pp.637-644, 2012.

[9] X. Pan and S. Zhang, A remote sensing image cloud processing system based on Hadoop, *IEEE International Conference on Cloud Computing and Intelligence Systems*, pp.492-494, 2012.

[10] F. C. Lin, L. K. Chung, C. J. Wang, W. Y. Ku and T. Y. Chou, Storage and processing of massive remote sensing images using a novel cloud computing platform, *Giscience & Remote Sensing*, vol.50, no.3, pp.322-336, 2013.

[11] G. H. Song, J. N. Chuai, B. W. Yang and Y. Zheng, QDFS: A quality-aware distributed file storage service based on HDFS, *IEEE International Conference on Computer Science and Automation Engineering*, pp.203-207, 2011.

[12] A. Patel and M. A. Mehta, A novel approach for efficient handling of small files in HDFS, *IEEE Advance Computing Conference*, pp.1258-1262, 2015.

[13] R. Rajak, D. Raveendran, M. C. Bh and S. S. Medasani, High resolution satellite image processing using Hadoop framework, *IEEE International Conference on Cloud Computing in Emerging Markets*, pp.16-21, 2015.

[14] R. Giachetta, A framework for processing large scale geospatial and remote sensing data in MapReduce environment, *Computers & Graphics*, vol.49, no.C, pp.37-46, 2015.

[15] I. Chebbi, W. Boulila and I. R. Farah, Improvement of satellite image classification: Approach based on Hadoop/MapReduce, *International Conference on Advanced Technologies for Signal and Image Processing*, pp.31-34, 2016.

[16] B. C. Sunny, R. Ramesh, A. Varghese and V. Vazhayil, Map-reduce based framework for instrument detection in large-scale surgical videos, *IEEE International Conference on Control Communication & Computing India*, pp.606-611, 2016.

[17] M. Z. Yang, H. B. Mei and D. M. Huang, An effective detection of satellite image via K-means clustering on Hadoop system, *International Journal of Innovative Computing, Information and Control*, vol.13, no.3, pp.1037-1046, 2017.

[18] D. Park, J. Wang and Y. S. Kee, In-storage computing for Hadoop MapReduce framework: Challenges and possibilities, *IEEE Trans. Computers*, vol.PP, no.99, pp.1-14, 2016.

[19] J. Parsola, D. Gangodkar and A. Mittal, Efficient storage and processing of video data for moving object detection using Hadoop/MapReduce, *International Conference on Signal, Networks, Computing, and Systems*, pp.137-147, 2017.

[20] J. P. Verma and A. Patel, Comparison of MapReduce and spark programming frameworks for big data analytics on HDFS, *International Journal of Computer Science & Communication*, vol.7, no.2, pp.80-84, 2016.

[21] J. L. Reyes-Ortiz, L. Oneto and D. Anguita, Big data analytics in the cloud: Spark on Hadoop vs MPI/OpenMP on Beowulf, *Procedia Computer Science*, vol.53, no.1, pp.121-130, 2015.

[22] T. Sharma, V. Shokeen and S. Mathur, Multiple K means++ clustering of satellite image using Hadoop MapReduce and Spark, *International Journal of Advanced Studies in Computer Science and Engineering*, vol.5, pp.23-31, 2016.

[23] I. Kusuma, M. A. Ma'Sum, N. Habibie, W. Jatmiko and H. Suhartanto, Design of intelligent K-means based on Spark for big data clustering, *IEEE International Workshop on Big Data and Information Security*, pp.89-95, 2017.

[24] 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 & Remote Sensing*, pp.1-17, 2016.

[25] K. R. Krish, B. Wadhwa, M. S. Iqbal, M. M. Rafique and A. R. Butt, On efficient hierarchical storage for big data processing, *IEEE International Symposium on Cluster, Cloud and Grid Computing*, pp.403-408, 2016.

[26] X. J. Tan, C. S. Deng and X. G. Dong, SparkDE: A paralleled version differential evolution based on RDD model in cloud computing, *Computer Science*, vol.43, no.9, pp.116-119, 2016.

[27] H. H. Wang, *Research on the In-Memory Data Management Technology on Spark Data Processing Framework*, Master Thesis, Beijing University of Technology, 2016.

[28] J. Yu, J. Wu and M. Sarwat, A demonstration of GeoSpark: A cluster computing framework for processing big spatial data, *IEEE the 32nd International Conference on Data Engineering*, pp.1410-1413, 2016.

[29] Y. Xia and F. Yang, Research on serialization storage strategy based on Spark cluster, *The 5th International Conference on Machinery, Materials and Computing Technology*, pp.454-459, 2017.

[30] D. Xu, Z. Yuan, T. Yu, D. Xie and F. Zheng, The research of remote sensing image segmentation and release which are based on Tile Map Service, *International Symposium on Geomatics for Integrated Water Resources Management*, pp.1-4, 2012.