A BIG DATA PLATFORM FOR REMOTE SENSING DATA PROCESSING

Chen Xu (1,2), Xiaoping Du (1)*

[1] Key Laboratory of Digital Earth, Institute of Remote Sensing and Digital Earth (RADI), Chinese Academy of Sciences, 100094 Beijing, China
[2] Université de Lyon, INSA Lyon, 25 Avenue Jean Capelle, 69621 Villeurbanne Cedex, France
*Email:duxp@radi.ac.cn

ABSTRACT: Remote sensing data are growing exponentially while few of which are used efficiently due to the lack of data processing capabilities. Consequently, mass remote sensing data processing is currently one of the most popular topics in the field of Geosciences. In this study, Remote Sensing Big Data Platform (RSBDP), a cluster-based data processing framework, is introduced. The aim of RSBDP is to store, manage and process large scale of remote sensing data in a cluster computing environment. The platform consists of three main parts: Hadoop-based RS file storage system, Quad-Tree and Hilbert curve based RS index system and Spark-based geospatial data processing system. Data is stored separately and redundantly in the platform while high-speed and high-concurrency data requests are supported. Structured data and indexed data are organized as key-value pair in HBase. A spatial index based on Quad-Tree and Hilbert curve is constructed for heterogeneous tiled remote sensing data which makes efficient data retrieval in HBase. We use Apache Spark as the analytics engine for big remote sensing data processing. Distributed in-memory computing architecture allows high performance remote sensing data analytics and applications. The result of tests proves RSBDP prototype can store, retrieve, and process heterogeneous remote sensing data efficiently. Meanwhile, Python code can be transplanted to RSBDP conveniently. It reveals RSBDP's potentials and capabilities of big remote sensing data processing.

KEY WORDS: big data, remote sensing data processing, distributed file system, HBase, Spark

1. INTRODUCTION

In recent years, the progress of high-resolution earth observation approaches gives birth to the proliferation of RS data. The data gathered by a satellite centre are accumulating at a rate of terabytes per day [1]. Based on large-scale remote sensing data, the demand for various RS applications need the support of enormous computational power [2]. Great efforts have been towards the availability of RS data and computation. Cluster based [3] and Cloud based HPC [4] are two dominating patterns for RS bigdata system. To achieve the high availability of RS data, parallel files systems have been widely applied, e.g. OrangeFS [5] and HDFS [6]. In regards to the programming approaches, OpenMP [7] and some other programming models involve low-level API issues. RS algorithms are difficult to implement directly through these models. MapReduce [8] is a successful and accessible model, but efficiency of MapReduce is expected to be improved.

In this paper, we introduce a new cloud-based RS big data processing prototype platform, Remote Sensing Big Data Platform (RSBDP). RSBDP relies on HDFS and HBase to store RS data. The RS images are divided into tiles and stored in HBase, combined with efficient and continuous index keys based on Quad-Tree and Hilbert curve. The computational work is

performed by Spark [9], a powerful memory-based distributed computing engine which have flexible access to RS algorithm. From the experiments, the prototype platform proves quick data query and powerful RS data computing capabilities.

The remainder of this paper is organized as follows. In Session 2, we describe the architecture of the platform ,explain the design and implementation of main components. In Session 3, we describe the experimental validation of the platform. Finally, we give a conclusion in Session 4.

2. METHODOLOGY

As the proliferation of RS data, conventional approaches are time-consuming and sometimes incapable. We propose Remote Sensing Big Data Platform (RSBDP) as a prototype platform for storage and processing large scale of RS data.

2.1 General architecture of RSBDP

The system is built upon virtual machines with OpenStack Cloud. Cloud offers flexible and guaranteed containers in support of the platform. The body of RSBDP consists of three main parts, RS file storage system, RS index system and geospatial data processing system. RS File Storage System is a storage framework to store and share RS data where Hadoop HDFS is applied as filesystem. NoSQL database named HBase is applied to store heterogeneous tiled RS data. RS Index System is a RS data index framework following the rules of Quad-Tree and Hilbert curve. The framework rules the structure of unique key for HBase and provides access for rapid and flexible query capacities. Geospatial Data Processing System is an on-demand computing framework on top of the Hadoop YARN. Spark is proposed to enable distributed in-memory computation.

2.2 RS data storage with HDFS and HBase

The availability of RS data is the essential concern in regards to intensive RS data subscription and distribution service. HDFS forms the base of the RS File Storage System to provide high-throughput access to RS data. HDFS is a distributed file system where data are replicated and stored in the distributed cluster. The RS file are replicated as 3 in storage system as default. Especially, part of hotspot data should be replicated more copies to meet up with the frequent subscription requirement. Following this manner, the data are secured with the multi-transcript stored in the different nodes. Additionally, load balance is enabled automatically to avoid network bottleneck of single node. To be specific, files stored in HDFS are always divided into blocks with certain size, the default size is 128MB. To reach the targeted RS data, a query should be performed on the related HBase table to obtain the file path redirecting to the file stored in HDFS.

However, under certain circumstances, e.g. change detection, targeted data are hidden inside several giant files. Subscription of all these files is not acceptable. In addition to the direct storage in HDFS, we propose to save the RS raster data in HBase as well. Firstly, giant RS files are divided into small tile files with the same pixel size, e.g. 256*256 pixels. Secondly, the binary tile file is converted to string with Base64 which is support by HBase. Each tile is then marked with a unique indexing row key with which we can find the tile in a large table of HBase. HBase supports the storage of large quantities of data as well as efficient query with row key. The pattern to index and query each tile will be discussed in the following part.

2.3 RS tile data index and query with Quad-Tress and Hilbert curve

The index and query of RS tile data is very important, compared with the data of the file metadata in HDFS, the tile data in HBase has greater query requirements. The efficiency of index and query directly affects the overall efficiency of RSBDP. HBase's query and indexing capabilities are not inherently powerful and only key-based queries are supported. Therefore, HBase storage indexes and queries for RS data need to be optimized. In this part, we discuss mainly about the index and query of RS tile data with Quad-Tree and Hilbert curve.

HBase supports immense unstructured data storage as well as rapid row key index. The efficiency of index relies on the design of row key. Firstly, row key should be as short as possible while the row key is uniquely pointed to each tile. For HBase, data are listed and index with the dictionary order of row key while continuous query is much efficient in regards to single query. Hence, we should guarantee the continuity of the row keys of which the data are space and time related. As illustrated in Fig. 1, an instance of a row key structure designed for tile data is composed with 4 main parts which takes about 9 chars. The first 2 chars represent the dataset the file belonging to. The band and year information take about 1 and 2 chars respectively. The last part of the row key indicates the space location of the tile in the image.

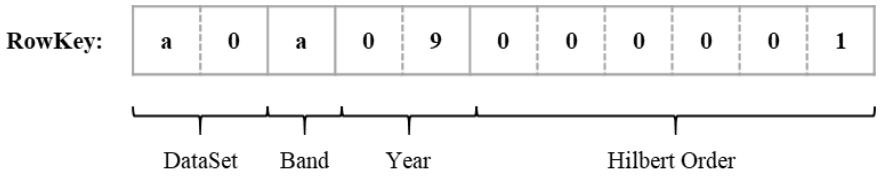

Fig. 1 Structure of row key for RS tiles

We propose to apply the Hilbert curve to fill the image while a unique order is assigned along with the curve pass by the tile. Compared with other linearization curves, e.g. Peano and row, Hilbert curve retains the spatial concentration of orders to the greatest extent. Compared with Hilbert curve, there are more mutations in Peano curve. For a range query in shape of a rectangle, Hilbert orders contained in the query range are segmented continuous.

2.4 RS data processing with Spark

In this part, we propose Spark to deal with large scale of RS data processing tasks. Spark is an in-memory parallel computation engine which is widely applied. First, as illustrated in Fig. 2, according to Spark's programming paradigm, the algorithm is implemented in some programming languages, such as Java, Python, and Scala. Each programming language has a rich library of deep learning and image processing, while Spark has a rich set of components for processing deep learning, databases, etc. We combine the extended library of programming languages with Spark's operators to solve various RS processing problems. The core concept of Spark is Resilient Distributed Datasets (RDD). For data-intensive problems such as RS processing, this method can effectively save memory resources and achieve fault tolerance. Only the action operator will be saved in memory. For the creation of general RDD, we need to import RS data. According to the method mentioned above, data can be obtained from HBase or from HDFS. Spark then analyses the logic of the code, generating execution steps for calculations and processing, which are represented in the form of a directed acyclic graph. The job is then divided into a number of tasks and assigned to the cluster for execution.

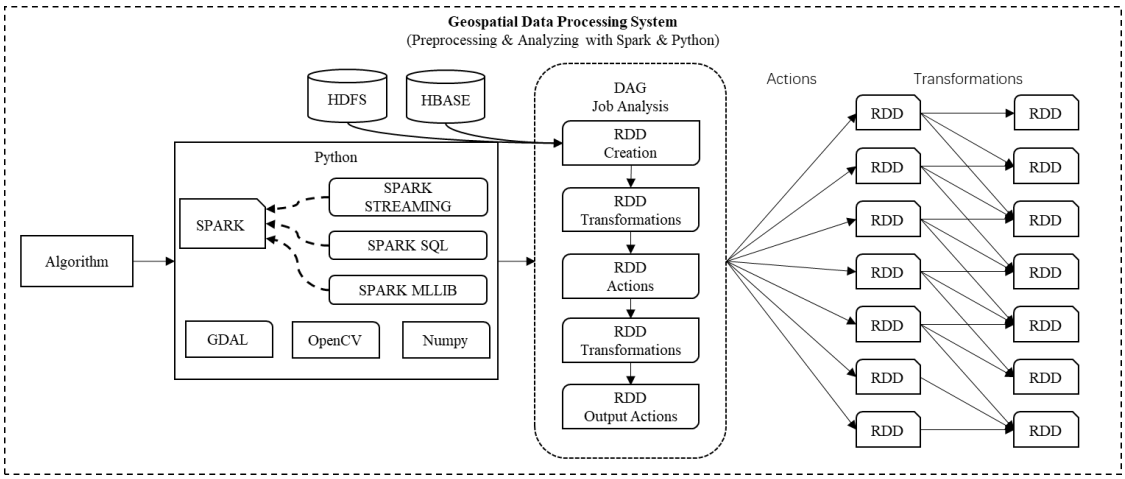

Fig. 2 programming model with Spark and Python

3. EXPERIMENTS AND DISCUSSION

In this part, we implanted an experimental prototype of RSBDP. The platform consists of 7 virtual machines with OpenStack. For purpose to verify the performance of RSBDP, some experiments are conducted. Firstly, we tested the query speed of HBase with two different patterns. Besides, the ability of spatial concentration of Hilbert curve is performed compared with row curve mentioned earlier. Secondly, we apply RSBDP to perform a parallel RS algorithm and compare the efficiency of RSBDP with a single machine.

3.1 Experiment environment

The prototype of RSBDP consists of 7 nodes while 2 of 7 are master nodes which control and guarantee the runtime of RSBDP. As illustrated in Table 1.The nodes are virtual machines supported by cloud through OpenStack. 4 nodes possess 8 cores of Intel Xeon E5-2690v4 16G of memory and 500G of storage space while the rest 3 nodes possess 4 cores of Intel Xeon E5-2690v4 8G of memory and 300G of storage space.

Table 1 Configuration and parameters of virtual machines of prototype

Node Name	CPU	Computer Configuration
hadoop-1-master	Intel Xeon E5-2690v4	8 cores 16GB 500GB
hadoop-2-master	Intel Xeon E5-2690v4	8 cores 16GB 500GB
hadoop-3-worker	Intel Xeon E5-2690v4	8 cores 16GB 500GB
hadoop-4-worker	Intel Xeon E5-2690v4	8 cores 16GB 500GB
hadoop-5-worker	Intel Xeon E5-2690v4	4 cores 8GB 300GB
hadoop-6-worker	Intel Xeon E5-2690v4	4 cores 8GB 300GB
hadoop-7-worker	Intel Xeon E5-2690v4	4 cores 8GB 300GB

3.2 Availability of RS data

For a data intensive task, availability of RS data relates closely with the efficiency of RSBDP. As I/O is handled by HDSF and HBase, there is no need to care about the optimization of data transmission. Only the performance of index and query method mentioned earlier will be tested.

For HBase, the query speed differs in regards to the approaches applied. The query speed with the RSBDP prototype by range query (SCAN) and single query (GET) is estimated. Range query means the row key of the data to be queried are continuous and the query is finished with one request while the single tile query is performed with each single row key. As illustrated in Fig. 3, with the number of data queried growing, the time consumed by single tile query is about 2.43 times of continuous range query.

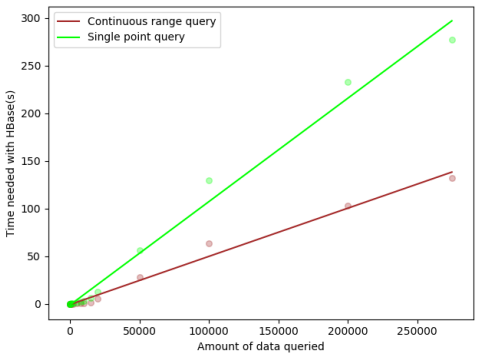

Fig. 3 Runtime required to perform the query through continuous range query and single query

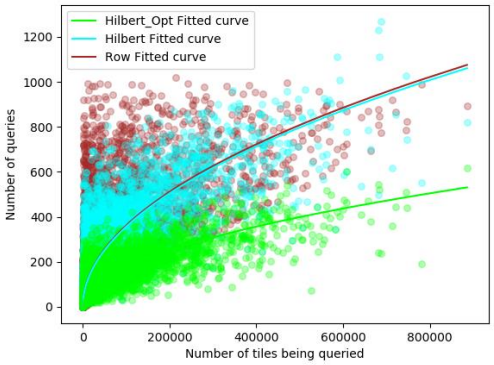

Fig. 4 Number of queried required through different linearization curves

We estimated the ability of spatial concentration of Hilbert curve which is shown in the Fig. 4. Supposing that most of the raster query are in the shape of rectangle, we created 5000 thousand of random rectangles which represent 5,000 random query regions. The tests are performed on a 9 order Hilbert curve which contains 4^9*4^9 tiles. As is shown in Fig. 4, among 5,000 trial queries, Hilbert and row required similar average number of queries. Hilbert needs 253.07 times while row requires 257.72 times. At the same time, we noticed that when the amount of query data is relatively small, Hilbert needs fewer queries than row. The results of optimized Hilbert curve are shown in green. It is very obvious that after the optimization, efficiency of Hilbert has been significantly improved. The number of queries required for the optimized Hilbert is 126.77 times on average.

3.3 Efficiency of parallel algorithms

To fully test the I/O and computational performance of the RSBDP prototype, we performed a RS parallel computation using the RSBDP prototype. Edge detection is a relatively common algorithm in the RS field. The performance test in this section is based on the Canny edge detection algorithm, which uses a 7*7 window size for edge extraction. Each tile is 256*256 pixels of RGB three channels. In order to ensure data consistency, the tests are all performed on the same tile, which is about 88Kb in size. The tile data is queried separately from HBase before each calculation. Therefore, the experiment can simultaneously test the I/O and computing performance of RSBDP.

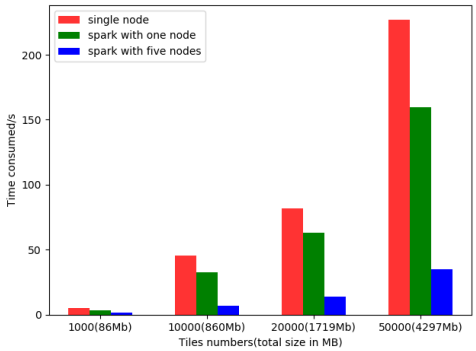

Fig. 5 Runtime required to operate Canny edge detection algorithm with 7*7 window size with different task sizes

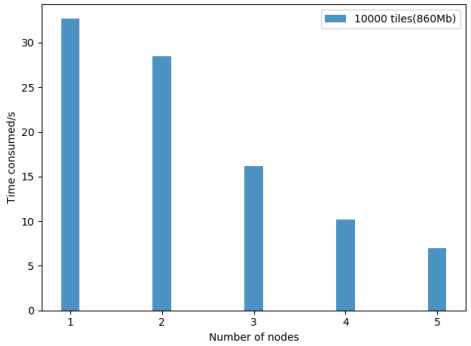

Fig. 6 Runtime required to operate Canny edge detection through Spark with different numbers of nodes

Fig. 5 shows the results in 3 cases, test on a normal machine, test on spark with one node and test on spark with 5 nodes. The single node is a machine with 8 cores and 16GB memory which is the same as the master machine of RSBDP prototype. For spark cluster, 2 cores and 2 GB memory is assigned for each spark task. It can be seen from the figure that even with a single node, Spark is more efficient than a single machine. Spark with 5 nodes requires much less time. For the test of 50,000 tiles (about 4.2G), normal machine takes about 227s while Spark with one node and Spark with 5 nodes need 160s and 35s separately. As is shown in Fig. 6, when treating 10,000 tiles (about 860M), more computing nodes need much less time.

4. CONCLUSION

The explosive growth of RS data has spawned great requirements for RS big data technology. In this study, we explain a novel cloud-based RS big data processing platform, RSBDP, providing RS big data storage, high-speed queries and powerful computing capacities. Based on distributed file system, RS data are stored in both file system and NoSQL database for rapid queries with Quad-Tree and Hilbert index. Spark is applied as programing approach which is well combined with HBase. In future study, we will refine the platform and conduct more experiments with more data to evaluate the overall performance of the platform.

5. ACKNOWLEDGMENTS

This work is supported by the Strategic Priority Research Program of Chinese Academy of Sciences, Project title: CASEarth (XDA19080103).

REFERENCES

[1] P. Gamba, P. Du, C. Juergens, and D. Maktav, “Foreword to the Special Issue on ‘Human Settlements: A Global Remote Sensing Challenge,’” IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens., vol. 4, no. 1, pp. 5–7, 2011.

[2] Y. Ma et al., “Remote sensing big data computing: Challenges and opportunities,” Futur. Gener. Comput. Syst., vol. 51, pp. 47–60, 2015.

[3] A. Plaza and C. I. Chang, High performance computing in remote sensing. 2007.

[4] M. N. O. Sadiku, S. M. Musa, and O. D. Momoh, “Cloud Computing: Opportunities and Challenges,” IEEE Potentials, vol. 33, no. 1, pp. 34–36, 2014.
[5] S. Yang, W. B. Ligon, and E. C. Quarles, “Scalable distributed directory implementation on Orange File System,” in 2011 IEEE 7th International Workshop on Storage Network Architecture and Parallel I/Os, SNAPI 2011, 2011.
[6] N. S. Islam et al., “High performance RDMA-based design of HDFS over InfiniBand,” in International Conference for High Performance Computing, Networking, Storage and Analysis, SC, 2012.
[7] E. Ayguadé et al., “The design of OpenMP tasks,” IEEE Trans. Parallel Distrib. Syst., 2009.
[8] J. Dean and S. Ghemawat, “MapReduce: Simplified data processing on large clusters,” Commun. ACM, 2008.
[9] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, “Spark : Cluster Computing with Working Sets,” HotCloud’10 Proc. 2nd USENIX Conf. Hot Top. cloud Comput., 2010.