Rasterhadoop: An Application Perspective of Raster Data Processing on Hadoop

R. Phani Bhushan, DVLN Somayajulu, S Venkatraman, RBV Subramanyam

Abstract: ***Hadoop is currently the most popular platform for parallel processing. With its two major components namely the Distributed File System (HDFS) and a parallel processing paradigm (MapReduce) in addition to its ease of installation and usage, Hadoop has become the chosen platform for efficiency whether in the commercial arena or the scientific arena such as Satellite Data Processing. The number of remote sensing satellites have also grown leaps and bounds and the data sent back by them for processing has all the three characteristics namely volume, velocity and variety that make it Big Spatial Data. In this paper, we present the extensions provided to Hadoop that enable Image Processing using legacy code and further elaborate on the various methods provided.***

Keywords: Hadoop, Raster Data, Image Processing, Big Spatial Data

I. INTRODUCTION

Hadoop consists of HDFS, which ensures data distribution on multiple nodes with a very high degree of reliability. The MapReduce paradigm utilizes this distributed platform to efficiently process large data. The physical systems' have doubled in their capacity for processing following the Moore's law, but have hit the number of transistors on a chip road block and hence distribution of processing is an inevitability which has been eased by Hadoop.

Satellites have proliferated into many domains and Geographical Information Systems that provide location based services have become very prominent ranging from the Google Maps application on Android mobiles to Bhuvan like image analysis platforms. The remotely sensed data of high-resolution satellites has a sub-metre spatial resolution and has been used to identify various ground objects. The high spatial resolution and large swath has contributed to the large volumes. The increase in the number of satellites has led to dumping of data on the ground reception stations at an unbelievable rate. The multi sensor satellites ranging from panchromatic sensors to to the variety of the data collected from these sources. Parameters such as spatial resolution, spectral resolution, temporal resolution and radiometric resolution have further increased the complexity of data processing. Added to this is the complexity of the changing programming language domain. The shift from procedural programming to Object Oriented Programming and todays scripting languages, though very English oriented, require a steep learning curve. The major problems being addressed in this paper are

A. *Usage of Commodity/Existing Hardware*

Big Data demands big resources for both storage and processing. Single monolithic storages require high-end systems for efficient retrieval, as this server becomes the gateway for data infusion and extraction. So the volume of data becomes directly proportional to the sizing of the retrieval server, which could lead to single point failures. This also calls for investing in new hardware at the expense of the existing hardware. The philosophy of Hadoop is to provide efficiency with commodity hardware, which is leveraged in this work. Additionally HDFS provides a high level of reliability as it makes replicas of the data and manages them as well.

B. *Processing satellite data as Big Data*

Usage of Hadoop provides the distributed platform for processing data in an efficient manner. MapReduce ensures that failure of processing on any node does not affect the entire processing by migrating the processes to the next available server. Secondly, large data requires large amount of memory both primary and secondary, which is optimized by the usage of data division based on [1]. Thirdly, dividing data into meaningful blocks of fixed sizes allows processing on available hardware.

C. *Providing Random access to data stored in HDFS*

Hadoop uses a single namespace for a data stored and hence every data is retrieved in its entirety. Also owing to the Hadoop's principle of write once read many, this random access to data is not envisaged. In this paper, we introduce certain functions that provide access to required blocks based on

1. Block Number
2. Corner LatLongs
3. Area names

Manuscript published on November 30, 2019.
* Correspondence Author

R. Phani Bhushan*, ADRIN, Dept. Of Space, Hyderabad, India, phani.rallapalli@adrin.res.in
DVLN Somayajulu, Dept. of CSE, IIITDM, Kurnool, India, somadvlns@gmail.com
S. Venkatraman, ADRIN, Dept. Of Space, Hyderabad, India, kalka@adrin.res.in
RBV Subramanayam, Dept. Of CSE, NIT, Warangal, India, rbvs66@gmail.

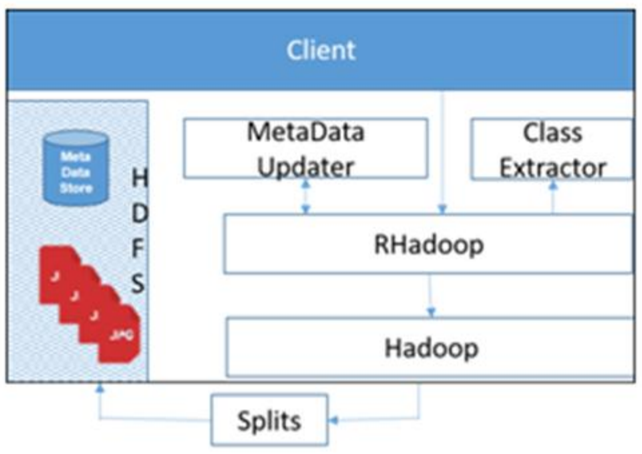


Figure 1. RHadoop System Overview

D. Usage of Legacy Code

Reflection is an important concept in OO programming languages that provides access to member functions of classes. This concept is used in order to instantiate classes and execute methods of the legacy code.

II. BACKGROUND

There have been many attempts at distributed image processing in the research community. Frameworks such as HIPI[2], HIPF[3], OpenImaj[4], MiPR[5] work on aggregating smaller images into a large image and using MapReduce based processing.

Processing of large raster data has been attempted in works such as SciDB[6], GeoTrellis[7], GeoSpark[8]. These have been works, which have built on the existing mapreduce framework and have provided extensions to data handling.

The work presented in this paper deals with organizing data in the FileSystem. The inherent feature of inter pixel relationship and its use in region growing algorithms is at the heart of this division of data across the nodes. Further, this paper discusses the methodology adopted to abstract the mapreduce paradigm thus alleviating the difficulty of application scientists to understand and code in a different way.

III. SYSTEM OVERVIEW

Figure 1 depicts the system overview. HDFS is utilized to store the imagery data, which is split by RatserHadoop (RHadoop) into manageable chunks. The distribution is based on geographical adjacency of the neighbouring pixels thus allowing for efficient performance of region growing algorithms. 1) The user interacts with RHadoop at command level or program level to provide a handle to the data that needs to be stored on HDFS. 2) RHadoop splits the data and hands it over to Hadoop for writing onto disk and extracts metadata, which is written into a NoSQL database. 3) The client interacts with RHadoop to perform an image processing function, which is defined in a class external to

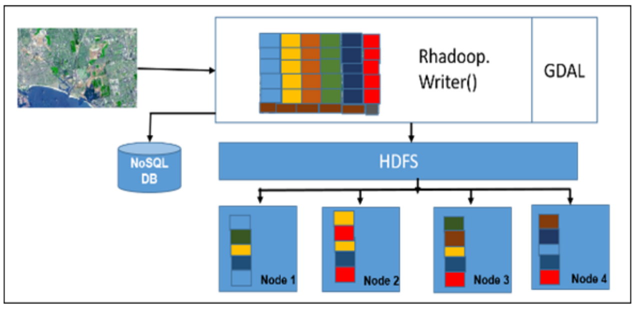


Figure 2. Data and Metadata Ingestion

The RHadoop, and this class is instantiated by the ClassExtractor component of RHadoop.

The RHadoop system deals with single large satellite images, which are the norm of high resolution imagery of the multi sensor satellites. The system provides data ingestion tools, data retrieval tools and mapreduce based data processing by embedding legacy code.

A. Data Ingestion

Large files of single images ranging from 2GB to 20 GB can be stored in HDFS. The default method of writing 128MB chunks reduces the efficiency of the processing [1]. The WriteImage() method of RHadoop provides an abstraction for writing images in dynamically sized chunks based on 1. Available nodes, 2. User Input. WriteImage() method uses Geospatial Data Abstraction Library (GDAL) to read the different formats of imagery and ingests into HDFS while it writes the corresponding metadata of the chunk into a NoSQL Database. This method of chunking the data allows efficient processing of region growing algorithms that are used in edge detection, classification and many other image-processing routines.

Figure 4 shows the size of chunks stored in HDFS while changing the chunk size from 8192*8192 to 2048*2048. This chunk size may be decided based on the number of participating nodes.

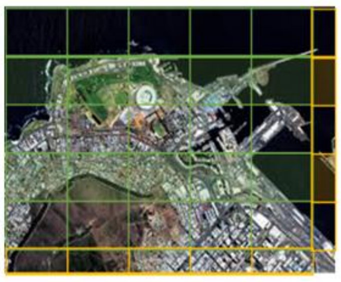
B. Data Retrieval

The data retrieval component retrieves complete or partial images based on user requirement. This method overrides the InputFormat, getSplits and RecordReader methods of MapReduce. The InputFormat method is overridden to match the logical sizes of the blocks, which vary within an image. The overridden getSplits method provides the splits with their width and height to the RecordReader method. The RecordReader method provides splits with their keys to the mapper. The key value includes the width, height and block number of the chunk.

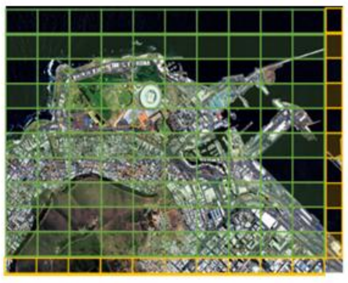
C. Mapper

The mapper creates an image from the bytes of the split and uses the reflection methods to instantiate the required class for processing the chunk of data. The individual mappers run these instantiated objects on the specific data node, thus parallelizing the activity.

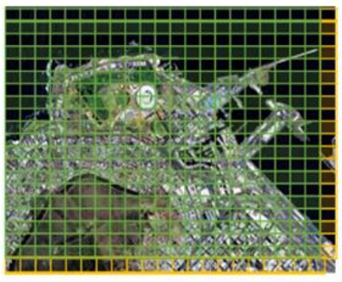
The separate chunks of data thus processed are written to a large canvas in the reducer and stored back using the data ingest engine into HDFS.



JPG image HDFS Size = 5.2GB
Scans = 43222 Pixels = 42285
Chunk Size = 8192*8192
25 Chunks of 8192*8192
5 chunks of 2262*8192
5 chunks of 8192*1325 and
1 chunk of 2262*1325



JPG image HDFS Size = 5.2GB
Scans = 43222 Pixels = 42285
Chunk Size = 4096*4096
100 Chunks of 4096*4096
10 chunks of 2262*4096
10 chunks of 4096*1325 and
1 chunk of 2262*1325



JPG image HDFS Size = 5.2GB
Scans = 43222 Pixels = 42285
Chunk Size = 2048*2048
420 Chunks of 2048*2048
21 chunks of 214*8192
20 chunks of 8192*1325 and
1 chunk of 214*1325

Figure 1. Image Is Divided Into 4 Different Sized Chunks And Rhadoop Reads These Chunks For Processing

IV. SYSTEM IMPLEMENTATION

The RHadoop system deals with reading multiple formats of large satellite data, storing the same in a manner that augurs well with random retrieval. The RHadoop createImage() module reads various formats of the satellite data using the GDAL library and based on the split factor splits the data to be written into the available nodes.

The flow in figure 2 depicts a typical flow for retrieval of data. The user queries for data fromRHadoop based on lat long or area name and RHadoop provides the block number using the getBlock()/getBlockLoc(). Then the readblock() method reads the associated block with its height and width and the mapper calls the relevant processing function and passes this data.

A. *Class Extractor*

The class extractor instantiates the relevant class which is passed as a parameter to RHadoop. The file with the classname and extension rhdp is of the form

The keywords "InImageName" and "OutImageName" are parsed to provide the relevant data to the methods of the class. The constructor of the class is either parameter-less or is hard coded with relevant parameters in the rhdp file.

Table 1 shows a list of the methods implemented by RHadoop. Figure 4 shows the content of .rhdp file for the cannyedgedetector class.

Table 1 The functions implemented by RHadoop and their description

Function	Description
RHadoop.getFilename()	Returns the filename for the block
RHadoop.readblock()	Returns the pixel data in the block
RHadoop.getBlock()	Returns the location of the block
RHadoop.getblockLoc()	Returns the spatial coordinates of the block
RHadoop.getblockWidth()	Returns the width of the block
RHadoop.getblockHeight()	Returns the height of the block
RHadoop.setblockWidth()	Set the width of the block while writing to HDFS
RHadoop.setblockHeight()	Set the height of the block while writing to HDFS
RHadoop.setblocLoc()	Set the spatial coordinates of the block
RHadoop.createImage()	Creates the Image
RHadoop.genKey()	Returns symmetric key used to encrypt the file
RHadoop.getKey()	Returns the key for given filename

```
ClassA(parameter list)

ClassA.method1(parameter list)

ClassA.method2(parameter list)

...........


Sample using Canny Edge Detector

CannyEdgeDetector detector = new CannyEdgeDetector()

Detector.setImage(<InImageName>);

<OutImageName> = Detector.process()
```

Figure 2 The Rhdp File For Class Cannyedgedetector

V. CONCLUSION

The Raster Hadoop implementation distributes data across the nodes utilizing the inherent property of neighbouring pixel relationship to enhance the efficiency of region growing algorithms.

Unlike Hadoop RHadoop provides random access to data blocks through the implemented interfaces and thus improves Region Of Interest based processing. In this paper we have presented the methods required to access the blocks of images and their metadata. We have also presented the class extractor that reads the rhdp file which instantiates the classes required for processing image data. Further, the abstractions implemented using the Class Extractor provide ease of programming to the application scientists.

REFERENCES

1. Phani Bhushan, R., Somayajulu D.V.L.N., Venkatraman, S. “A Raster Data Framework Based on Distributed Heterogeneous Cluster”, J Indian Soc Remote Sens (2018). https://doi.org/10.1007/s12524-018-0897-5
2. Sweeney, C. Liu, L., Arietta, S., Lawrence, J. ‘HIPI: A Hadoop Image Processing Interface for Image-based MapReduce Tasks.” B.S. thesis, University of Virginia, 2011.
3. Hare, Jonathon, Samangooei, Sina and Dupplaw, David (2011) OpenIMAJ and ImageTerrier: JavaLibraries and Tools for Scalable Multimedia Analysis and Indexing of Images. At ACM Multimedia 2011,Scottsdale, Arizona, USA, 28 Nov - 01 Dec 2011. ACM, 691-694
4. Sridhar Vemula, and Christopher Crick. "Hadoop image processing framework" IEEE International Congress on Big Data. 2015.
5. Andrey Sozykin, and Timofei Epanchintsev. "MIPr –a framework for distributed image processing using Hadoop" IEEE 9th International Conference on Application of Information and Communication Technologies (AICT), 2015
6. G. Planthaber, M. Stonebraker, and J. Frew. “EarthDB: Scalable Analysis of MODIS Data using SciDB”. BIGSPATIAL, 2012.
7. GeoTrellis. 2014.geotrellis | Fast Raster Processing. [Online] Azavea, 2014. [Cited: August 18, 2014.] http://geotrellis.io/.
8. J. Yu, J. Wu, and M. Sarwat, “Geospark: A cluster computing framework for pro-cessing large-scale spatial data,” inProceedings of the 23rd SIGSPATIAL Interna-tional Conference on Advances in Geographic Information Systems, p. 70, ACM,2015.

AUTHORS FROFILE

R.Phani Bhushan, is working at Dept. of Space. He is involved in developing applications on cutting edge technologies along with his research pursuits in the areas of cloud computing, Big Data Analytics, No-SQL databases, Spatial Big Data, Host and Network security solutions.

D.V.L.N. Somayajulu, is working in Department of Computer Science and Engineering at the National Institute of Technology (NIT), Warangal. Curently Director, IIITDM, Kurnool. He has over 25 years of Academic and Industry experience.. He has been awarded the Best Engineer of the Year in 2007 His area of interest is around Database Performance, Data mining and eLearning. He has carried out various Software projects sponsored by Government of India and has organized various Conferences and workshops. He has more than 100 publications to in national and international journals.

S. Venkataraman, was born on 18 June 1960. He has worked at Defence Electronics Research Laboratory (DLRL) for 5 years in the design and development of Thin Films Hybrid Microwave Integrated Circuits before joining Dept. Of Space in 1989. At DoS he has been responsible in development of Artificial Intelligence techniques in Satellite Image processing and Analysis. Currently he is steering a group of scientists in the area of cryptographic solutions for data and network security. Software and Embedded based security solutions have been developed for various projects. He has more than 45 publications.

R B V Subramanyam, is currently the Head of CSE Dept at National Institute of Technology Warangal. He has published many journal and conference papers in the areas of Data Mining. Some of his research interests include Data Mining, Distributed Data Mining, Fuzzy Data Mining, Distributed Data Mining and Big Data Analytics. He is one of the reviewers for IEEE Transactions on Fuzzy Systems and also for Journal of Information and Knowledge Management. He is member in IEEE and The Institution of Engineers (India).