Evaluating HIPI Performance on Image Segmentation Task in Different Hadoop Configurations

Mosab Shaheen[1], Madhukar B. Potdar[2], Bhadreshsinh Gohil[3]

[1] *Distributed Systems Researcher, GTU PG School, Gujarat, India*
[2] *Project Director, BISAG, Gujarat, India*
[3] *Assistant professor, GTU, Gujarat, India*

---***---

Abstract – *The available media nowadays is continuously increasing. Huge amount of images from different sources like social media, satellite images, etc. can be used for different applications. To be able to handle this vast amount of data, parallel and distributed frameworks come into the picture. Hadoop is a widely used framework for distributed processing of big data. While Hadoop showed good performance, it suffers from large number of small size files. Hadoop Image Processing Interface (HIPI) library solved this problem when working with images. In this work, we will compare HIPI with sequence files and basic Hadoop and see the improvement gained by using it, also we will use different configurations of Hadoop to see how we can get better results. We will evaluate the performance on segmentation/clustering tasks over satellite images.*

Key Words: Hadoop, HIPI, Sequence Files, Small Size Files, Image Segmentation.

1. INTRODUCTION

The increasing number of images with different image resolutions and from different sources resulted in a big amount of image data ready to be processed. Traditional platforms like sequential systems and low scale frameworks are unable to handle such vast amounts of data. Here it comes other frameworks that can work under data and computing intensive tasks.

Apache Hadoop framework allows patch processing of big data in a distributed and parallel manner and offers scalable and reliable environment that proves efficiency in many applications [2]. Hadoop has shown greater performance when dealing with large size files than large number of small size files [1] [2] [3] [4] [5] because of issues regarding storing large amount of metadata about these files and where they are located, and regarding accessing this information in Hadoop which will make an overhead on the Namenode.

For this issue, one framework called HIPI comes into the picture. HIPI is a library designated for image processing based on Hadoop framework and offers facilities like bundling images, culling/filtering, encoding/decoding, etc. [3]. HIPI has been used in many applications such as bundling video frames [6] for instrument detection.

We used satellite images as the input for the tasks. Satellite images usually contain many bands for red, green, blue, near infra-red, mid infra-red, etc. and sometimes they contain 14 bands. For this purpose, one image standard called GeoTIFF has been developed, this allows georeferencing information to be embedded within the TIFF file format. Therefore, to be able to use the satellite images in HIPI library, we added the support for this standard.

Segmentation refers to the operation that groups the pixels in an image depending on the similarity. In case of satellite images, it usually corresponds to land cover types [7]. For clustering, K-means is a popular algorithm that can both cluster the images and do the segmentation on them [7].

We will compare HIPI with Hadoop sequence files and with basic Hadoop, and will show how HIPI enhanced the performance and reduced the time for processing. In addition, we will present the important configurations in Hadoop that utilize the available resources and give better results.

The work is organized as follows: first, it presents a literature review of Hadoop and HIPI. Then, it views the problem statement and the methodology for implementing the tasks. Later it shows the important configuration parameters and the results gained from these configurations, and last is a conclusion.

2. OVERVIEW

2.1 Hadoop as Efficient Framework for Big Data Processing

Bajcsy et al. [1] gave a comparison between some parallel and distributed systems including Hadoop, Terasort, Teragen and Java Remote Method Invocation (RMI). The experiment shows that Hadoop does not give good performance for large number of small size files; on the contrary, when large size files are used, Hadoop outperforms other frameworks.

Yan et al. [2] built an engine based on Hadoop framework using OpenCV library for image processing; also they emphasized that the speed-up is greater for big size files. Moreover, Li et al. [5] showed that the performance of Hadoop on large number of small size files is less than on small number of large size files

2.2 HIPI Outperforms Other Frameworks Regarding Small Size Files Issue of Hadoop

To solve the large number of small size files issue, Sweeney et al. [3] implemented the HIPI library. HIPI creates an image bundle, which is a collection of images grouped into one file. HIPI Image Bundle (HIB) consists of two files the data file and the index file. While Hadoop Archive (HAR) files can be used as archives of files, they may give slower performance due to the technique used to deal with the files inside. Sequence files gives better performance than the standard Hadoop applications. However, sequence files must be read serially and they take considerable time to be generated. On the contrary, HIPI is not restricted to serial reading and it has similar speed to sequence files.

Sozyki et al. [4] showed another framework for image processing called MapReduce Image Processing framework (MIPr). They made a comparison between MIPr, HIPI, and OpenIMAG. It was shown that HIPI gave the best result regarding the time to perform the task.

Li et al. [5] showed that, HAR cannot be changed whenever created and the name cannot contain spaces; and sequence files are serially read. Whereas CombineFileInputFormat is an abstract class and needs implementation. The authors created HMPI library based on HIPI and compared it to HAR and standard Hadoop. The result indicated that HMPI gave the best performance.

3. Problem Statement

Apache Hadoop cannot work effectively on large number of small files; rather it works fine on large size files. For this purpose, HIPI library is created; and it shows better performance than MIPr, OpenIMAG and other Hadoop structures. However, configuring Hadoop plays the key role in performance. Weak configuration will lead to slow performance and will not make the difference in performance clear. Therefore, we will present how the configuration affects the performance and how to choose the suitable configuration.

4. Methodology

First, we have to generate the HIPI Image Bundles (HIBs) and Sequence Files for different dataset sizes. In case of HIB, we should combine all the image files into two files (data and index); and in case of sequence file, it is one file. The second thing to do is segmentation of images using the data structure (HIB, Sequence file, or Basic) and K-means algorithm. Fig -1 describes the general steps in this work.

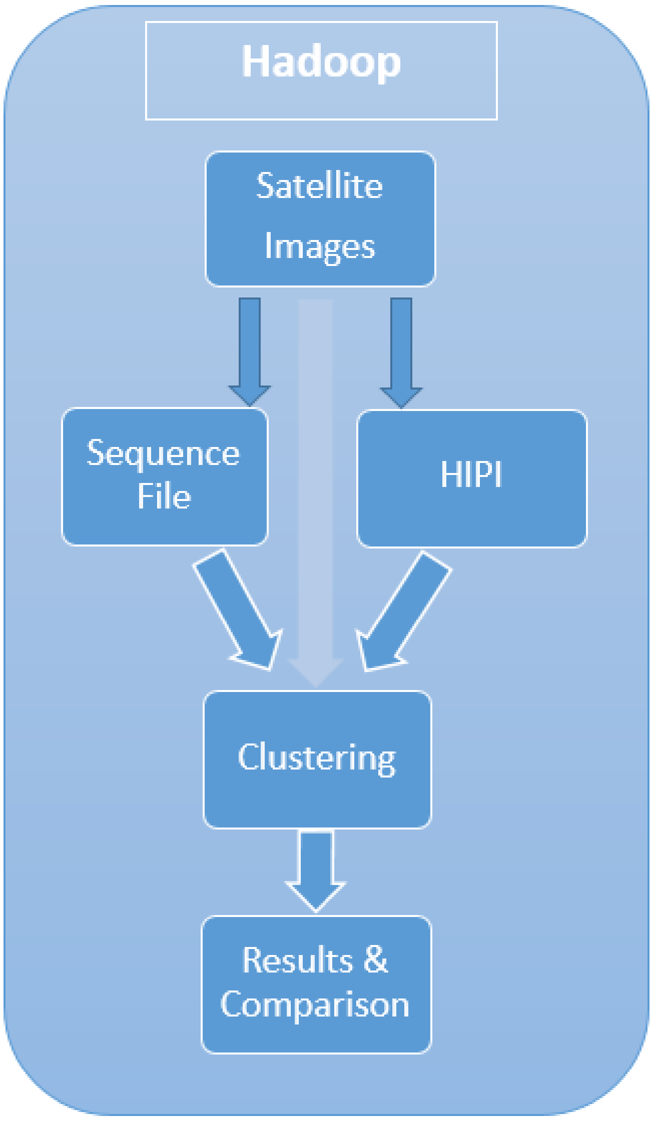

Fig -1: Block diagram of the work

To go in depth of what is happening in the second step, let's take the HIB scenario as in Fig -2.

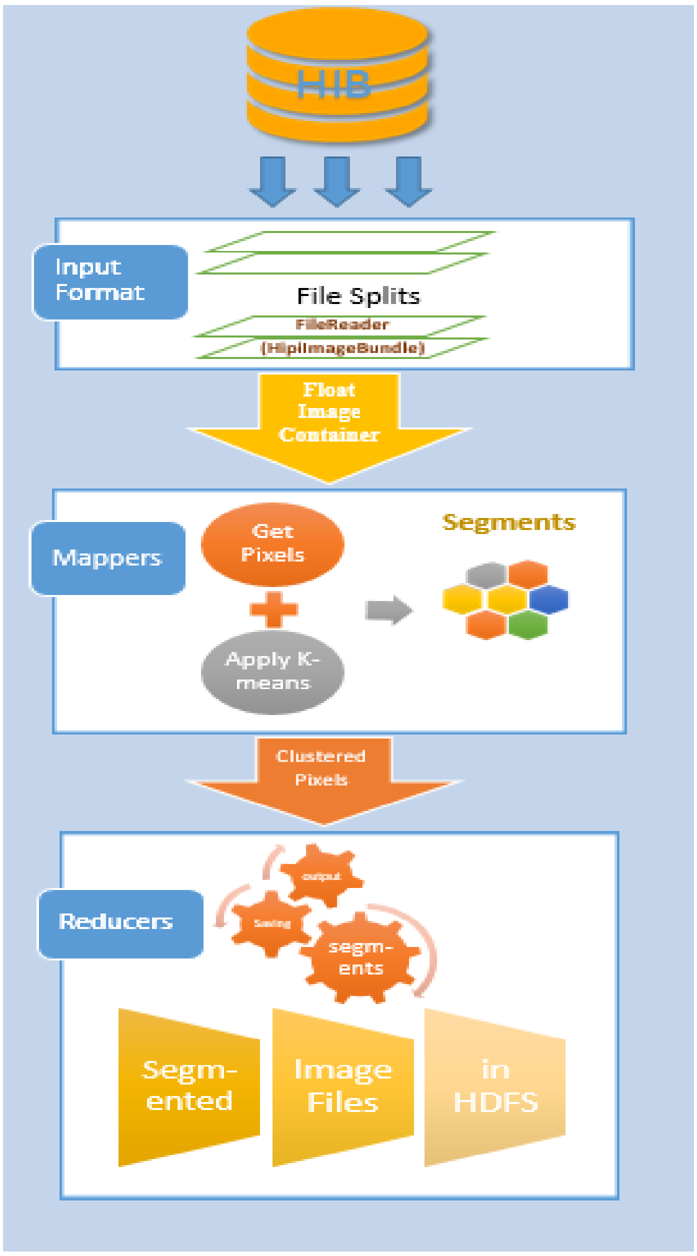

Fig -2: Clustering using HIPI and K-means

In the beginning, HIB is split into FileSplit objects. Each one holds a start and an end offsets inside HIB data file, with the help of the index file, and thus the FileSplit can span over many image files inside HIB determined by these offsets. As a result, many files now can be processed by one MapTask and we avoid the overhead of creating too many MapTasks on the system. The FileReader of HipiImageBundle is responsible for reading the FileSplit and detecting the images between the start and the end offsets, and for each image it will use the developed TIFFImageUtil class which will take the image data and decode it into the ImageHeader and TIFFFloatImage objects. The TIFFFloatImage contains the longitude and latitude information, and original information of FloatImage like: pixels, width, height, and number of bands. Each FloatImage/TIFFFloatImage will be encapsulated inside the FloatImageContainer. After that, the FloatImageContainer object will be forwarded and processed by a MapTask; which gets the pixels from the object, apply the k-means algorithm on them, and generate segments representing different covers or areas in the image. The cover types can be: water, soil, streets, forests, building, etc. The results of Mappers are <key, value> pairs containing the clustered pixels. They will be forwarded to reducers. The reducers, in turn, will save the clustered pixels i.e. segments, in RGB format inside the HDFS. These reducers are useful to minimize the time and overhead on the Mappers to do the output operations in HDFS.

5. Configuration

Hadoop configuration plays the key role in performance. Configuration determines how many Map/Reduce Tasks can run at the same time (concurrently), amount of memory for each task and container, number of processing units dedicated for each container, replication factor, and so forth. Also we should notice that Hadoop 1 configuration is different from Hadoop 2 as Yarn becomes the resource manager and task scheduler. In Yarn there is no concept of static slot allocation for Maps/Reduces that run in parallel but rather it depends on the Map/Reduce and container configuration. Every container can run only one Map/Reduce Task and multiple containers can run at the same time executing different tasks. Also if the NodeManager is not running in a node, the node will not be able to run any task but it can still function as a data node. Here is a list of the important configurations in Hadoop:

- **mapreduce.map.memory.mb**: determines the maximum physical memory that is needed to run one map task. If it is exceeded, usually you get "Container is running beyond physical memory limits". This configuration determines the number of Map Tasks that can run in parallel as Hadoop will see how much memory available and how much memory each Map Task needs. Same thing applies to **mapreduce.reduce.memory.mb** for Reduce Tasks.

- **mapreduce.map.java.opts:** determines the maximum heap memory assigned to a Map Task. If it is exceeded, usually you get "java. lang. OutOfMemoryError: Java heap space". Same thing applies to **mapreduce.reduce.java.opts** for Reduce Tasks.

- **yarn.scheduler.minimum-allocation-mb:** minimum memory for a container.

- **yarn.scheduler.maximum-allocation-mb:** maximum memory for a container.

- **yarn.nodemanager.resource.memory-mb:** maximum memory available in a node.

- **yarn.nodemanager.vmem-pmem-ratio:** the ratio between physical memory and virtual memory. We talked about the physical memory before; for a Map Task the virtual memory is calculated by multiplying mapreduce.map.memory.mb and yarn.nodemanager.vmem-pmem-ratio. Same thing applies to Reduce Tasks.

- **yarn.nodemanager.vmem-check-enabled:** sometimes the virtual memory exceeds the limit defined by the ratio mentioned above. This may kill the container and give a message like "1.1gb of 1.0gb virtual memory used. Killing container.", this can happen because of the aggressive allocation of the memory by the operation system. To stop checking, if the virtual memory exceeds the ratio, you can use this option.

- **yarn.nodemanager.resource.cpu-vcores:** specifies the number of CPU cores the node has.

We also have to keep in mind the following things:
The number of Maps equals to the number of splits. Splits are logical units that specifies parts of data. Splits can be created manually in the InputFormat class or can be automatically driven from split size = max (min split size, min (max split size, block size)), so when a file exceeds the split size it will be split. However, if the files are less than the split size (usually the block size) then each file is considered as a split. That's one reason that small files are not recommended in Hadoop. Reducers may need double amount of memory allocated to Maps depending on the application. Number of containers can be one per CPU core more than this it may cause an overhead on performance.

After, configuring, running, and completing the tasks; it is time to see the output of segmenting the satellite images using K-Means algorithm. Fig -3 shows a sample of a segmented image in RGB format with 5 clusters.

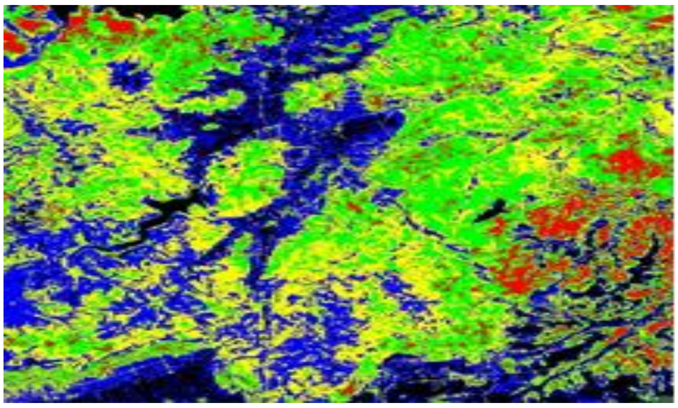
Fig -3: Sample output of a segmented image

6. Results

We applied clustering on the BISAG Dataset. The dataset is a collection of GeoTIFF images each one is around 80.9KB in size. Clustering was done using the previous methodology, which is compared to clustering with Basic Hadoop (without using Hadoop structures) and with Hadoop sequence files. The experiment is applied repeatedly, each time with different number of images to see the changes in performance when the data size grows. The work is done by creating 3 Virtual Machines (VMs) holding Ubuntu OS and Hadoop; one is Master and others are slaves. For each execution we ran 6 MapTasks and 6 ReduceTasks. We allocated 2 CPU Cores to each node, 4.7 GB RAM to each slave node and 5GB to the master node. The hosting PC is a workstation that has 8 CPU Cores and 16GB RAM. We were able to run the 6 Map/Reduce Tasks concurrently because of the fine-tuned configuration; where we used Yarn for task scheduling and resource management with 1000MB RAM to Map/Reduce Task, 800MB RAM to JVM of the task, and 500MB minimum & 3500MB maximum RAM to a container. Table -1 shows the results with 6 Map Tasks and 6 Reduce Tasks, visual representation is shown in Chart -1:

Number of Images#	Without/Basic	With Sequence File	With HIPI
6181#, 0.5 GB	14mins, 56sec	13mins, 54sec	12mins, 28sec
12962#, 1.0 GB	33mins, 17sec	32mins, 38sec	28mins, 27sec
18389#, 1.5 GB	46mins, 21sec	46mins, 24sec	44mins, 39sec
24693#, 2.0 GB	1hrs, 1mins, 14sec	1hrs, 5mins, 18sec	56mins, 41sec
30591#, 2.5 GB	1hrs, 36mins, 44sec	1hrs, 18mins, 30sec	1hrs,15mins, 41sec
37086#, 3.0 GB	1hrs, 56mins, 6sec	1hrs, 43mins, 29sec	1hrs, 5mins, 37sec

Table -1: Results of clustering, using 1 PC and 6 Map/Reduce Tasks

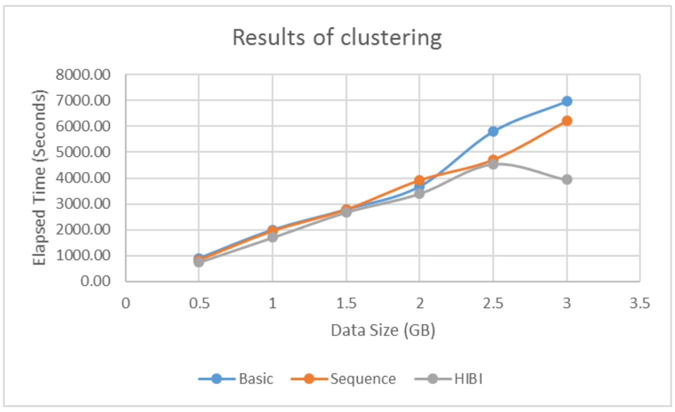

Chart -1: Visual representation of results, using 1 PC and 6 Map/Reduce Tasks

We can see from Table -1 that HIPI takes the least time to perform the task followed by Sequence File and last comes the Basic Hadoop. We can also notice that Sequence File sometimes get closer to HIPI in performance and other times get closer to Basic Hadoop, but generally it keeps in between. Although, clustering is a time consuming task and most of the time went for clustering, not for accessing the files, but still we can see clearly that working with HIPI reduced the overall time to process the small size image files.

It is worth mentioning that Hadoop is a scalable environment. As a result, when we allocate more resources, Hadoop will scale up. All what is required is few changes in the configurations. Therefore, the job can be done faster and the number of concurrent tasks can be increased easily. For example, in case we used 2PCs (every PC has 8 CPU Cores and 16GB RAM, and holds one VM with 6 CPU Cores and 12 GB RAM), then we can run 12 Map/Reduce Tasks concurrently. The results are shown in Table -2 and the visual presentation in Chart -2:

Number of Images#	Without/Basic	With Sequence File	With HIPI
6181#, 0.5 GB	6mins, 24sec	5mins, 10sec	4mins, 49sec
12962#, 1.0 GB	12mins, 49sec	10mins, 41sec	10mins, 45sec
18389#, 1.5 GB	18mins, 10sec	14mins, 20sec	13mins, 19sec
24693#, 2.0 GB	20mins, 21sec	20mins, 47sec	19mins, 4sec
30591#, 2.5 GB	24mins, 12sec	32mins, 23sec	22mins, 27sec
37086#, 3.0 GB	36mins, 6sec	32mins, 13sec	27mins, 30sec

Table -1: Results of clustering, using 2 PCs and 12 Map/Reduce Tasks

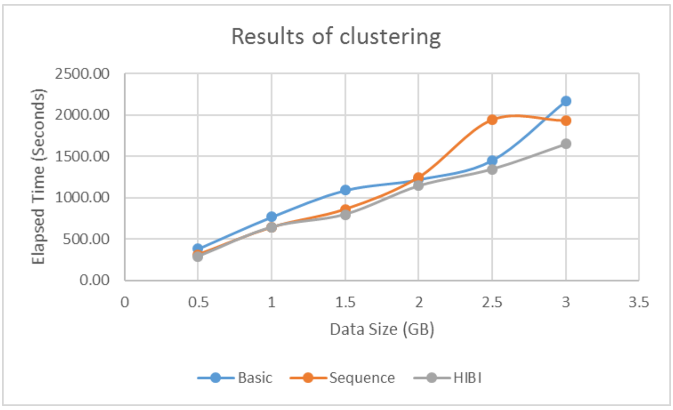

Chart -2: Visual representation of results, using 2 PCs and 12 Map/Reduce Tasks

In addition, utilizing the resources is essential key in performance. If we take the first experiment and run 3 Map/Reduce Tasks instead of 6 we will underload the CPU cores (because the default value of mapreduce.map.cpu.vcores and mapreduce.reduce.cpu.vcores is 1) and will not utilize the memory properly as the Table -3 and Chart -3 show:

Number of Images#	Without/Basic	With Sequence File	With HIPI
6181#, 0.5 GB	20mins, 18sec	19mins, 50sec	15mins, 6sec
12962#, 1.0 GB	43mins, 5sec	42mins, 46sec	42mins, 9sec
18389#, 1.5 GB	1hrs, 1mins, 16sec	1hrs, 5mins, 5sec	1hrs, 1mins, 19sec
24693#, 2.0 GB	1hrs, 29mins, 27sec	1hrs, 27mins, 47sec	1hrs, 1mins, 1sec
30591#, 2.5 GB	1hrs, 45mins, 24sec	1hrs, 53mins, 56sec	1hrs, 1mins, 15sec
37086#, 3.0 GB	2hrs, 12mins, 22sec	2hrs, 14mins, 53sec	2hrs, 2sec

Table -3: Results of clustering, using 1 PC and 3 Map/Reduce Tasks

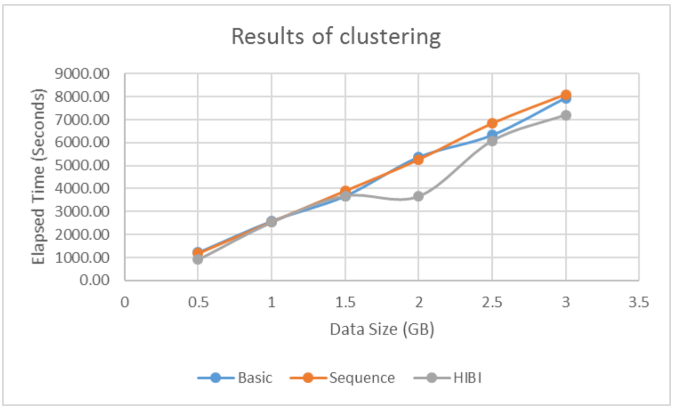

Chart -3: Visual representation of results, using 1 PC and 3 Map/Reduce Tasks

Finally, tuning the memory of Map/Reduce Tasks & containers is vital because a small memory may cause failure and killing of the Map/Reduce Task that is running. For example, if we want to run 12 Map/Reduce Tasks concurrently and allocate 500MB RAM to a Map/Reduce Task, 300MB RAM to JVM of the task, and 256MB minimum & 3500MB maximum to a container for processing the 3 GB data in our experiment (37086 images), then some containers may fail. Eventually, the job may fail completely or may take more time to be completed. This may happen due to the lack of memory required to processing the data.

7. CONCLUSIONS

Hadoop is a good framework for processing a big amount of image data. It offers handling large data sets with scalability, reliability, distribution & parallelism, and fault tolerance. Moreover, HIPI offers many facilities like bundling and supports the MapReduce model of Hadoop. The results show that HIPI gives better performance than sequence files and basic Hadoop. In addition, using HIPI library with Hadoop environment can improve the performance and make the work more efficient; especially because Hadoop cannot work with large number of small size files efficiently. Moreover, we viewed the importance of configurations in utilizing the resources and how different configurations can degrade or enhance the performance.

ACKNOWLEDGEMENT

We are thankful to Shri T. P. Singh - Director at BISAG for supporting this research and to Mr. Miren Karamta - System Manager, HPC system at BISAG for providing required infrastructure.

REFERENCES

[1] P. Bajcsy, A. Vandecreme, J. Amelot, P. Nguyen, J. Chalfoun and M. Brady, "Terabyte-sized image computations on Hadoop cluster platforms" IEEE International Conference on Big Data. 2013, Silicon Valley, CA, 2013, pp. 729-737.

[2] Yuzhong Yan, and Lei Huang, "Large-scale image processing research cloud" IARIA The Fifth International Conference on Cloud Computing, GRIDs, and Virtualization. May 25 - 29, 2014 - Venice, Italy.

[3] Chris Sweeney, Liu Liu, Sean Arietta, and Jason Lawrence. "HIPI: a Hadoop image processing interface for image-based mapreduce tasks" University of Virginia Undergraduate Thesis. 2011.

[4] 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

[5] Jia Li, Kunhui Lin, and Jingjin Wang "Design of the mass multimedia files storage architecture based on Hadoop" IEEE 8th International Conference on Computer Science & Education (ICCSE). April 26-28, 2013. Colombo, Sri Lanka.

[6] B. C. Sunny, Ramesh R, 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 (ICCC). 2015, Trivandrum, 2015, pp. 606-611.

[7] I. Chebbi, W. Boulila, and I. R. Farah, "Improvement of satellite image classification: approach based on Hadoop/Map Reduce" IEEE 2nd International Conference on Advanced Technologies for Signal and Image Processing - ATSIP. 2016 March 21-24, 2016, Monastir, Tunisia