Large-Scale Image Processing Research Cloud

Yuzhong Yan, Lei Huang
Department of Computer Science
Prairie View A&M University
Prairie View, TX
Email: yyan@student.pvamu.edu, lhuang@pvamu.edu

***Abstract*—We have entered the big data era, where massive data are generated each single day. Most of these new generated big data are images and videos. Besides the fast-increasing data size, the image and video processing algorithms become much more complex, which poses great demands to data storage and computation power. Our image processing cloud project aims to support the image processing research by leveraging the cloud computing and big data analysis technology. In this paper, we present our design for image processing cloud architecture, and big data processing engine based on Hadoop. We also report the performance scalability and analysis on the cloud using several widely used image processing algorithms.**

Keywords–Cloud Computing; Map/Reduce; Hadoop; Image Processing

I. INTRODUCTION

We have entered the so-called big data era, where massive data are generated each single day. Big data are generated by digital processing, social media, Internet, mobile devices, computer systems and a variety of sensors. Most of these new generated big data are images and videos. Big data analytics requires scalable computing power and sophisticated statistics, data mining, pattern recognition, and machine learning capabilities [1]. It is exaggerative in image processing domain since the image and video processing algorithms become more and more complicated, which demands even more power in computation. Some of these image processing requires even real-time processing capability [2]. It is time to rethink if we need to create a domain specific cloud for image processing research in order to meet these challenging requirements.

Image processing research and education are fundamental to support research in many other fields such as medical, oil & gas, and security. It has been widely used in industries. Researchers and students working on the domain are in great need of a high-level programming environment that can utilize the latest, large scale computing resources to speed up their research, since the image data have much higher resolution and the computation are much more sophisticated and intensive than before. The modern computer architectures, however, have evolved to be extraordinarily complex, and frequently becomes a challenge rather than help for general researchers and educators that use image processing technology, which is even equally true for experts in this domain. In order to utilize large scale computing resources to meet the image processing requirements, researchers will face scalability challenges and hybrid parallel programming challenges of creating code for modern computer hardware configurations with multi-level parallelism, e.g., a cluster based on multicore processor nodes. It is not only hard for researchers to implement their algorithms using existing programming environment; but, it is also challenging to them to reuse and share the existing research results since these results are largely dependent on OS, libraries, and underlying architectures.

In order to fill the gap between complicated modern architectures and emerging image processing algorithms for big data, our image processing cloud project aims to produce a high-performance and high-productivity image processing research environment integrated within a cloud computing infrastructure. The cloud will not only provide sufficient storage and computation power to image processing researchers, but also it provides a shared and open environment to share knowledge, research algorithms, and education materials. By leveraging the cloud computing and big data processing technology, our design is to hide the software and hardware complexity from researchers, so that they can focus on designing innovative image processing algorithms, instead of taking care of underlining software and hardware details.

In this paper, we discuss the related work in Section II, and then introduce our image processing cloud architectures in Section III. Further, we describe our experimental image processing applications and their performance analysis in Section IV and Section V, respectively. Last, we will discuss the future work and conclusion in Section VI.

II. RELATED WORK

There are several related work in processing images in parallel using Hadoop platform. The biggest difference between our work and others is that our solution provides a PaaS and supports the multiple languages in implementing image processing algorithms. HIPI [3] is one of them that is similar to our work. In contrast to our work, HIPI [3] creates an interface for combining multiple image files into a single large file in order to overcome the limitation of handling large number of small image files in Hadoop. The input type used in HIPI is referred to as a HipiImageBundle (HIB). A HIB is a set of images combined into one large file along with some metadata describing the layout of the images. HIB is similar with Hadoop sequence file input format, but it is more customizable and mutable [4]. However, users are required to modify the image storage using HIB, which creates additional overhead in programming. In our work, we make the image storage transparent to users, and there is no additional programming overhead for users to handle image storage.

Hadoop Mapreduce for Remote Sensing Image Analysis [5] aims to find an efficient programming method for customized processing within the Hadoop MapReduce framework. It also uses the whole image as InputFormat for Hadoop, which

 ISBN: 978-1-61208-338-4

is similar with our solution. However, the work only supports Java so that all mapper codes need to be written in Java. Compared with our solution, he performance is not as good as the ours since we use native C++ implementation for OpenCV.

Parallel Image Database Processing with MapReduce and Performance Evaluation in Pseudo Distributed Mode [6] performs parallel distributed processing of a video database by using the computational resource in a cloud environment. It uses video database to store multiple sequential video frames, and uses Ruby as programming language for Mapper, thus runs on Hadoop with streaming mode same as ours. As a result, our platform is designed to be more flexible and supports multiple languages.

Large-scale Image Processing Using MapReduce [7] try to explore the feasibility of using MapReduce model for doing large scale image processing. It packaged large number of image files into several hundreds of Key-Value collections, and split one huge image into smaller pieces. It uses Java Native Interface(JNI) in Mapper to call OpenCV C++ algorithm. Same with the above work, this work only supports a single programming language with additional overhead from JNI to Mapper.

III. PVAMU CLOUD ARCHITECTURE

The PVAMU (Prairie View A&M University) Cloud Computing infrastructure is built on top of several HPC clusters together. The cloud consists of a virtual machine farm based on Apache CloudStack [8] to provide Infrastructure as a Service (IaaS), and a Hadoop-based high-performance cluster to pvoide Platform as a Service (PaaS) to store and process big data in parallel. Although we describe the entire system in the section, the experiments conducted in the paper were on top of the Hadoop cluster. We integrated the widely-used image processing library OpenCV [9] on the Hadoop cluster to build the image processing cloud. We describe these two major components in the following sections.

Figure 1 shows the Cloud Computing infrastructure developing at PVAMU. The infrastructure consists of three major components: 1) A Cloud center with a large number of Virtual Machines (VM) farm as the cloud computing service portal to all users; 2) A bare-metal high performance cluster to support High Performance Computing (HPC) tasks and big data processing tasks; 3) a shared data storage and archive system to support data access and storage. In this system, the Cloud infrastructure functions as the service provider to meet a variety of users requirements in their research and education. For HPC, the Cloud submits these tasks to the HPC cluster to fulfill their computing power demands. For these high throughput applications, the Cloud will deliver suitable virtual machines from the VM farm to meet their requirements. The Cloud orchestrates all functionalities of the entire system; provide elastic computing capability to effectively share the resources; delivers the infrastructure/platform services to meet users research requirements; supports the big data storage and processing; and builds a bridge between end-users and the complicated modern computer architectures.

A. PVAMU Virtual Machine Farm Cloud

We create a virtual machine farm based on Apache CloudStack on top of an 56 nodes dual-core IBM cluster, and

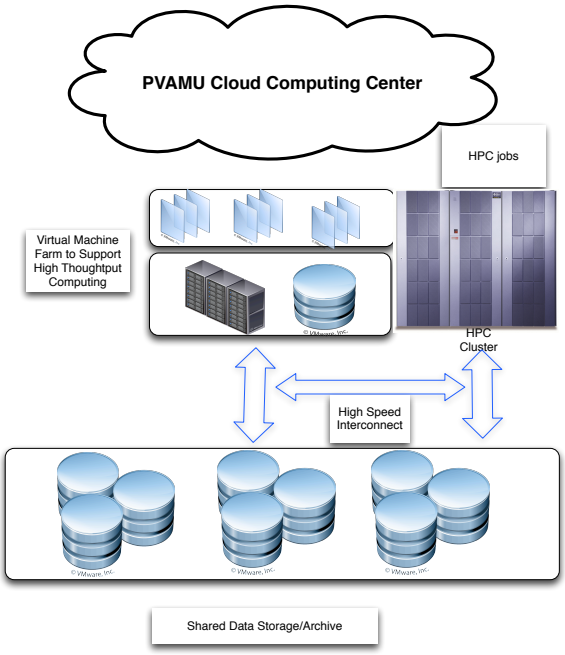

Figure 1. PVAMU Cloud and HPC Cluster for Big Data Processing

a new small Dell cluster with three 32 CPU cores servers, and one GPGPU server with 48 CPU cores and 1 NVIDIA Fermi GPGPU. Apache CloudStack is an open source software package that can deploy and manage large number of Virtual Machines to provide highly available, and highly scalable IaaS cloud computing platform. The goals of the PVAMU cloud are to provide IaaS and PaaS with customized services, to share resources, to facilitate teaching, and to allow faculty and students in different groups/institutions to share their research results and enable deeper collaborations. The CloudStack is used to manage users, to handle users requests by creating virtual machines, and allocate resources.

B. Image Processing Cloud

The image processing cloud is built by integrating the image processing library OpenCV with Hadoop platform to deliver PaaS specifically for image processing. The following describes the two major components.

1) Hadoop Cluster: We installed the Hadoop [10] big data processing framework on the bare-metal HPC cluster within PVAMU Cloud to provide PaaS. All experiments presented in the paper are conducted on the Hadoop cluster. The Hadoop cluster consists of one 8-node HP cluster with 16-core and 128GB memory each, and a 24-node IBM GPGPU cluster with 16-core and one Nvidia GPU in each node, and connected with InfiniBand interconnection. We have installed the Intel Hadoop Distribution [11] based on Apache Hadoop [10] software stack, which is a framework that is designed to store and process big data on large-scale distributed systems with simplified parallel programming models. It consists of Hadoop common utilities, Hadoop Distributed File System (HDFS) for high-throughput and fault tolerance data access, Hadoop Yarn for job scheduling and resource management, and Hadoop MapReduce [12] for parallel processing engine based on a simple parallel pattern. Besides its capabilities of storing and

 ISBN: 978-1-61208-338-4

processing big data, the built-in fault tolerance feature is also a key to complete big data analytics tasks successfully. The Hadoop cluster is used in our project to handle image and video storage, accessing and processing.

2) OpenCV Image Processing Library: We selected the widely-used OpenCV (Computer Vision) [9] library as the base image processing library integrated with our image processing cloud. OpenCV is an open source library written in C++, and it also has Java and Python interfaces supporting Windows, Linux, Mac OS, iOS and Android. It is optimized and parallelized for multicores and accelerators using OpenCL. We installed the library on the Hadoop cluster to enable image processing capability with MapReduce parallel programming model.

By combining the above two components, we are able to implement a scalable image processing cloud to deliver the capabilities as services to support researchers/faculty/students to conduct their research in image processing domain. In the next section, we present our design and implement of several image processing algorithms in the cloud, and discuss their performance.

IV. Design and Implementation Image Processing Cloud

The goal of our image processing cloud is to deliever PaaS to image processing researchers and developers. It should be able to store large amout of images and videos, as well as be able to process them and meet the performance requirements. Users should be able to work their image processing algorithms using their familiar programming langugaes with very limited knowledge in parallelism. It is a challenge to meet these requirements since image processing researchers use different programming languages in designing and implementing algorithms. The most popular-used programming models include Matlab, Python, C/C++, and Java. In order to meet the multi-language requirement, we cannot rely on native Hadoop Java programming model.

Hadoop platform provides distributed file system (HDFS) that supports large amount of data storage and access. Hadoop MapReduce programming model supports parallel processing data based on the widely-used map-and-reduce parallel execution pattern. In order to support the multiple language requirements in image processing domain, we choose Hadoop streaming programming model by revising standard input and output, and stream data to applications written with different programming languages. Moreover, the streaming model is also easy to debug in a standalone model, which is critical to test and evaluate an algorithm before going to large-scale. To achieve the best performance, we choose C++ in our underlining library implementation to keep the optimizations as much as possible.

The image processing application execution environment with MapReduce on Hadoop is shown in Figure 2. On the left side, a large number of images are stored in HDFS, which are distributed across the cluster with 128MB as one block. These images are split by Hadoop MapReduce engine with customized InputFormat, and are distributed to large number of mappers that execute image processing applications to the assigned images. The results may be merged by the reducer that exports the results to customized OutputFormat class to finally save the outputs. Since large amount raw data are transferred among split, mappers and reducers, it is very important to keep data locality to minimize network traffic. All mappers are launched on the node where the processed images are physically stored.

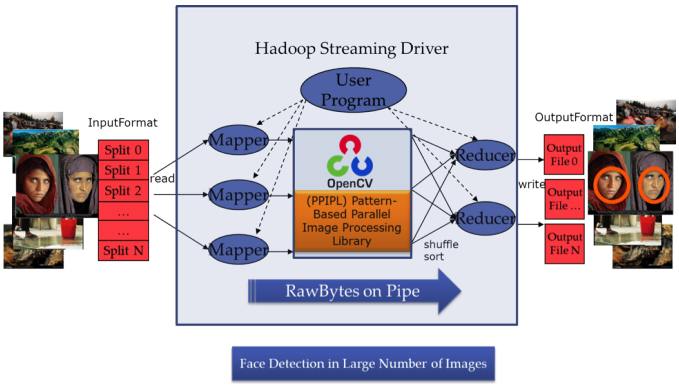

Figure 2. Image Processing Execution Environment with MapReduce

A. InputFormat

The main challenges of performing image processing on Hadoop are how to split data split and how to implement customized mappers. In Hadoop streaming mode, the input data need to be processed by InputFormat class at first, and then pass to each mapper through the standard input (Stdin). The InputFormat class in Hadoop is used to handle input data for Map/reduce job, which need to be customized for different data formats. The InputFormat class describes the input data format, and define how to split the input data into InputSplits buffer, which will be sent to each mapper. In Hadoop, another class RecordReader is called by mapper to read data from each InputSplit.

Depending on the image or video size, we implemented two different InputFormat classes to handle them. For still image processing with many individual image files, the InputFormat class is straightforward. It simply distributes these images to mappers by each image file since they are smaller than block size of Hadoop system. For the mass individual image files, ImageFileInputFormat extends FileInputFormat, which return false in isSplitable and create ImageFileRecordReader instance in getRecordReader. ImageFileRecordReader will creates Key/Value pair for mapper and read whole content of input image file actually.

For the big video file, it needs to be split and to be sent to the mapper for processing. There are different video file containers; in this project only MPEG transport stream file is considered to simplify split implementation. TSFileInputFormat is used for parsing the MPEG transport stream, and for generating split information including offset in video file and the hostname which will process the related split, and create TSFileRecordReader in the getRecordReader function. TSFileRecordReader will create Key/Value pair for mapper and read the section data from input video file, then pass it to mapper for processing.

 ISBN: 978-1-61208-338-4

B. Mapper and Reducer

Most of work for programming in Hadoop is to divide algorithms into Mapper and Reducer, and embed and implement them in them respectively. In Hadoop streaming mode, the main difference with other modes is the I/O processing in Mapper and Reducer. Both Mapper and Reducer could only get Key/Value from Stdin and output results through Stdout. A common I/O class named CommonFileIO was designed to handle different type data sources, including normal local files, Stdin/Stdout and HDFS file on Hadoop. The frequently used file system interfaces were provided, such as open, read/write and close and more. We implement our own Mapper and Reducer as independent image processing applications with input and output handled by Stdin and Stdout. By using Hadoop streaming model, we are able to launch these image processing applications as large number of Mappers or Reducers that execute in parallel.

C. OutputFormat

OutputFormat class in Hadoop describes the output-specification for a Map-Reduce job. It sets the output file name and path and creates the RecordWriter instance, which is passed to Map/Reduce framework and writes output results to file. For the image processing with small files, OutputFormat is unnecessary and the intermediate results could to be stored on HDFS directly. But for big video file, different applications will output different results. We have implemented several OutputFormat templates for reducer jobs. For example, to get the Histogram of whole file, it needs to accumulate each result of Reducer in OutputFormat; while for the template matching application, it needs to save each matched result and give a summarization in OutputFormat.

D. Results

As a result of our implementation, the image processing cloud is able to handle image processing algorithms written with multiple lanuages, including Matlab, Python, C/C++, and Java, which is the major contribution comparing with other related work. Moreover, the cloud provides scalable performance by keeping the native C++ implementation of OpenCV library internally, and takes the data locality into consideration in the task scheduling strategy. The next section discusses the performance experiments using three typical image processing algorithms.

V. EXPERIMENTS

We choose three widely-used image processing algorithms including Discrete Fourier Transform (DFT) [13], face detection, and template matching to conduct performance and programming experiments on our image processing cloud. Our images are downloaded from Internet public photos, including Google images, National geographic photo gallery, and Flickr public photos. The Fourier Transform algorithm is one of fundamental and most-widely used image processing algorithm that transforms data from spatial domain to frequency domain to facilitate more advanced image processing algorithms. The 2D DFT are frequently applied to digital image files in many image processing algorithms. Face detection on image and video is very useful in many areas, such as surveillance and entertainment equipment. The feature-based cascade classifiers provide a practical face detection algorithm; it is popular used but still need much computation for multi-images or video file. Template Matching could find the location of small template image in big image files, which is one core function in machine vision. These three algorithms are implemented on top of OpenCV, and apply them to three groups of images as test pattern. These three groups of images are separated to the small size group with images less than 1MB, the middle size group of images from 1M to 5MB, and the big size group of images from 5MB to 30MB. The experiments are conducted on our small HPC cluster with 8 HP nodes, 16 cores and 128GB memory each. In this cluster one is mater node for jobtracker and the other seven are worker nodes for computation, so we have total 112 cores. Table I shows the face detection program execution time for both sequential and Hadoop MapReduce parallel execution.

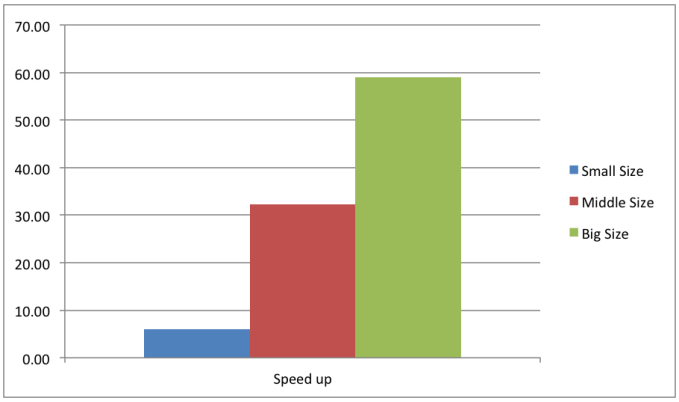

Figure 3. Face Detection Program Speedup on Hadoop

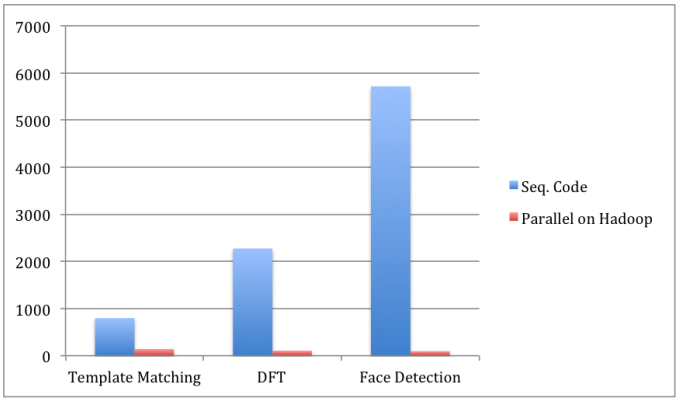

Figure 4. Execution Time of Different Algorithms Apply on Big Size Images

TABLE I. FACE DETECTION PROGRAM EXECUTION TIME FOR THREE GROUPS OF IMAGES

	Small Size Images with 5425 Files/594MB	Middle Size Images with 2539 Files/3621MB	Large Size Images with 400 Files/4436MB
Sequential codes	1386.02s	4511.35s	5716.31s
Parallel on Hadoop	228s	140s	97s

Figure 3 shows the face detection speedup of the three groups of images comparing with sequential execution. With

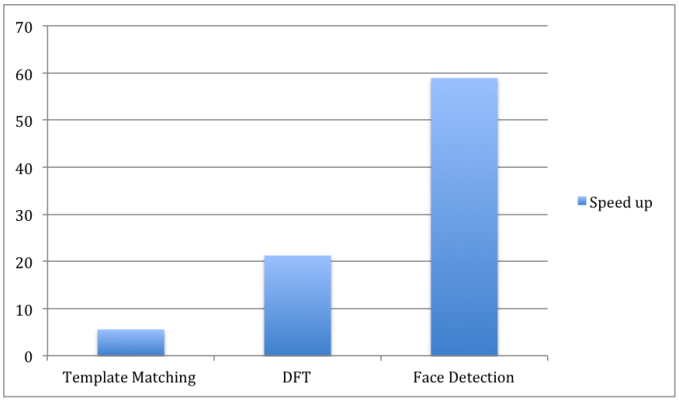

Figure 5. Speed up of Different Algorithms Apply on Big Size Images

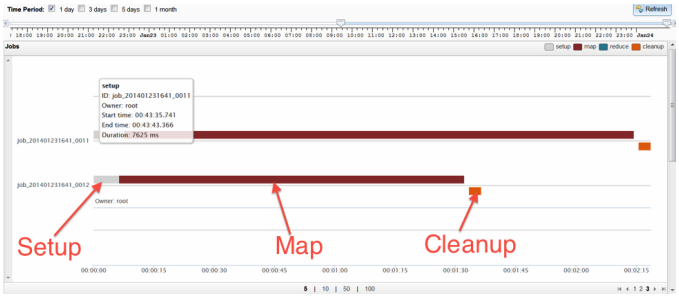

Figure 6. The job running results on Hadoop

8 nodes and 16 cores each, the experiments show a maximum 59 times speedup for big images. Apparently, the image size determines the speedup due to the I/O processing time. In Hadoop big data framework, it is designed to achieve better performance with big files and computation intensive programs. The max number of simultaneously running mappers could reach up to max CPU cores in the cluster. The small images need to be aggregated into big files to improve the performance. Figure 4 shows the execution time of different algorithms applying on big size image files. Figure 5 shows the speed up factors of different algorithms applying on big size image files. The face detection need more time to execute on single process, but could get best speed up on Hadoop platform.

The test program could be divided into three main parts: input and output, decode and encode, image processing algorithm. The total execution time of sequential codes is the sum of all images processing time. We can use the following formula to represent it.

$$Ts = (Ti + Td + Ta) \times N \quad (1)$$

Here, *Ti* is the image reading and writing time; *Td* is the image decoding and encoding time and *Ta* is the algorithm executing time.

While running on Hadoop with only mapper, the total execution time is composed of:

$$Th = Tp + (Tm + Tr) \times (N \div C) + Tc \quad (2)$$

Here, *Tp* is the job preparing/setup time for Hadoop job; *Tm* is the average mapper executing time, which is nearly equal

Figure 7. The small image processing job profiling results on Hadoop

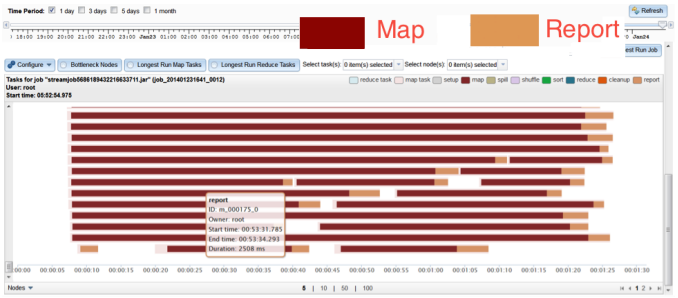

Figure 8. The big image processing job profiling results on Hadoop

to *(Ti + Td + Ta)*; *Tr* is the average mapper report time and *Tc* is the job cleanup time. In our execution environment, the *Tp* is about 8s and *Tc* is about 3s while running job as shown in Figure 6.

The profiling results of small size image pattern and big size image pattern are shown in Figure 7 and Figure 8. For the small size image pattern, assuming the average *Tr* is 2.5s, and get the average execution time of one image from sequential code execution result, which is 0.2555s, the ideal speed up factor on Hadoop system could be estimated by:

$$S = \frac{1386.02}{8 + (0.2555 + 2.5) \times \lceil \frac{5425}{112} \rceil + 3} = \frac{1386.02}{147.2} = 9.4$$

For the big size image size pattern, assuming the average *Tr* is 2.5s, and get the average execution time of one image from sequential code execution result, which is 14.3s, the ideal speed up factor could be estimated by:

$$S = \frac{5716.31}{8 + (14.3 + 2.5) \times \lceil \frac{400}{112} \rceil + 3} = \frac{5716.31}{87.65} = 65$$

Considering the overhead between mappers, the estimated results is close to our experimental results in the big size of images case, which is the ideal speed-up by considering the data movement, task startup and cleanup overheads. In order to get better performance on Hadoop, we need to reduce these overheads. One possible solution is to further improve the split function to determine a good number of mappers based on the number of available nodes, and reduce overloads of mappers startup and cleanup. The improvement will be explored in the future work.

The Hadoop system has good robustness and scalability. Comparing with the traditional MPI program, MapReduce programs are able to complete jobs even one or more computing

 ISBN: 978-1-61208-338-4

nodes in a cluster are down. New nodes could be added into the Hadoop system at runtime to meet dynamic requirements, thus get better performance in most cases and provide elastic computing as needed.

VI. Future Work and Conclusion

At the first stage of the project, our main goal is to explore the feasibility and performance of using Hadoop system to process large number of images, big size of images or videos. From our experimental results, Hadoop is able to handle these problems with scalable performance. However, there are also some issues need to be considered and addressed in future work.

The first issue is the problem of data distribution. As stated in the previous section, Hadoop is good at handling big data. The speedup is not apparent while trying to process many small images scattered across multiple nodes. Even the SequenceFile could not solve this problem efficiently. Our next plan is trying to store image files in HBase [14]. HBase could handle random, realtime reading/writing access of big data. We expect to improve performance and increase the flexibility with new solution on HBase.

The second issue is that Hadoop is not good at handle low-latency requirement. Apache Spark [15] is a fast and general-purpose cluster computing system. Because of the in-memory nature [16] of most Spark computations, Spark programs can better utilize the cluster resources such as CPU, network bandwidth, or memory. It can also handle pipeline, which is frequently used in image processing. In next step, we will try to move to Spark platform, and evaluate the performance of the experimental groups on Spark platform.

Another main goal of this project is to make it easy for users processing image using cloud computing platform. Most of users are not familiar with cloud platform, such as algorithm experts or even common users; they all have requirements of big data processing. In the next stage, a Domain Specific Language (DSL) for image processing and friendly user interface will be provided. Users could utilize the powerful platform with only limited knowledge on Cloud and use DSL to simplify their programming efforts.

Acknowledgment

This project is supported in part by National Science Foundation CRI program award #1205699. Opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

[1] J. C. Brian Dolan and J. Cohen, "MAD Skills: New Analysis Practices for Big Data," in Very Large DAta Bases(VLDB) 09. Lyon, France: ACM, Aug. 2009.

[2] C.-I. C. Hsuan Ren and S.-S. Chiang, "Real-Time Processing Algorithms for Target Detection and Classification in Hyperspectral Imagery," IEEE Transactions on Geoscience and Remote Sensing, vol. 39, no. 4, 2001, pp. 760–768.

[3] "Hadoop image processing interface," http://hipi.cs.virginia.edu/, [Retrieved: January, 2014].

[4] L. L. Chris Sweeney and J. L. Sean Arietta, "HIPI: A hadoop image processing interface for image-based map reduce tasks," pp. 2–3, 2011.

[5] M. H. Almeer, "Hadoop Mapreduce for Remote Sensing Image Analysis," International Journal of Emerging Technology and Advanced Engineering, vol. 2, 2012, pp. 443–451.

[6] K. K. Muneto Yamamoto, "Parallel Image Database Processing with MapReduce and Performance Evaluation in Pseudo Distributed Mode," International Journal of Electronic Commerce Studies, vol. 3, no. 2, 2012, pp. 211–228. [Online]. Available: http://www.academic-journals.org/ojs2/index.php/ijecs/article/viewFile/1092/124

[7] K. Potisepp, "Large-scale Image Processing Using MapReduce," Master's thesis, Tartu University, 2013.

[8] "Apache CloudStack website," http://cloudstack.apache.org, [Retrieved: January, 2014].

[9] "Open Source Computer Vision," http://www.opencv.org/, [Retrieved: January, 2014].

[10] "Hadoop Introduction," http://hadoop.apache.org/, [Retrieved: January, 2014].

[11] "Intel Distribution of Hadoop," http://hadoop.intel.com/, [Retrieved: May, 2014].

[12] J. D. S. Ghemawat, "MapReduce: simplified data processing on large clusters," in Communications of the ACM - 50th anniversary issue: 1958 - 2008, vol. 51. ACM New York, Jan. 2008, pp. 107–113.

[13] C. S. B. Thomas W. Parks, DFT/FFT and Convolution Algorithms: Theory and Implementation. John Wiley & Sons, Inc. NY, USA, 1991.

[14] "Apache Hadoop database, a distributed, scalable, big data store," http://hbase.apache.org/, [Retrieved: January, 2014].

[15] "Spark Lightning-fast cluster computing," http://spark.incubator.apache.org/, [Retrieved: January, 2014].

[16] M. Z. Mosharaf Chowdhury and T. Das, "Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing," in NSDI'12 Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation. San Jose, CA: USENIX Association Berkeley, Apr. 2012.

 ISBN: 978-1-61208-338-4