I590 BIG DATA OPEN SOURCE SOFTWARE AND PROJECTS FINAL PROJECT

FACE DETECTION WITH OPENCV

May 26, 2016

John Henderson (jphender@umail.iu.edu)
Meng Li (li545@iu.edu)
Indiana University Bloomington
School of Informatics and Computing

INTRODUCTION

In recent decades, as technologies have advanced rapidly, the price of collecting data via numerous sensing devices has decreased significantly. As a consequence, data have achieved explosive growth which brings us to the big data era. As the term suggests, big data differs from traditional data mainly in size and complexity.

Essentially, big data is characterized by a set of attributes: volume, velocity, variety, and veracity. Volume defines scale of data. As it is estimated by leading firms which are specialized in offering technical solutions to big data related business, 2.5 quintillion bytes of data are created every day and most companies in the US have at least 100 terabytes of data stored. By 2020, it is predicted that around 40 zettabytes of data will be created which is an increase of 300 times from 2005. Velocity corresponds to analysis of streaming data. It deals with the speed at which real-time data flows in from thousands of sources, such as business processes, networks and interactions between human and web applications like social media apps. It is estimated that the New York Stock Exchange captures 1 terabyte of trading data during each trading session. Normally, those streaming data records are continuously and simultaneously generated in small sizes which raises difficulties in analytics, considering storage and processing. Variety describes different forms of data. Traditional data are those structured and stored in relational databases, while big data come in both structured and unstructured forms. Example include emails, photos, video and audio data. As an estimate, more than 4 billion hours of video are watched on youtube each month. The variety of unstructured data typically creates problems for data processing. Veracity refers to uncertainty of data. It is quite common to have biases, noise and abnormality in data. According to surveys conducted by marketing research companies, poor quality data can cost US as high as trillions of US dollars per year.

Challenges thereby arise from four dimensions of big data. Variety, velocity together with volume dictate difficulties in data curation, storage, visualization and synchronization across data sources, since traditional data processing applications are inadequate. Additionally, veracity causes further problems to analytics, since data cleaning on a massive scale is required prior to analysis in order to achieve valid results. Despite challenges, there are tremendous benefits to harnessing big data. In principle, it helps organizations to build new insights in their business, sharpen their business acumen, make effective strategic decisions and create attractive business propositions. Concretely, firms are able to tailor products and services to customers' needs, discover new sources of revenue and optimize operations by leveraging big data as a valuable asset.

Motivated by the challenges and power of big data, industry has invested considerable efforts into developing both commercial and open source big data technologies. Apache Big Data Stack (ABDS) is a well established collection of open source applications designed for processing large and complex datasets. To further boost the productivity of ABDS, High Performance Computing (HPC) stack is integrated which also extends the breadth of ABDS.

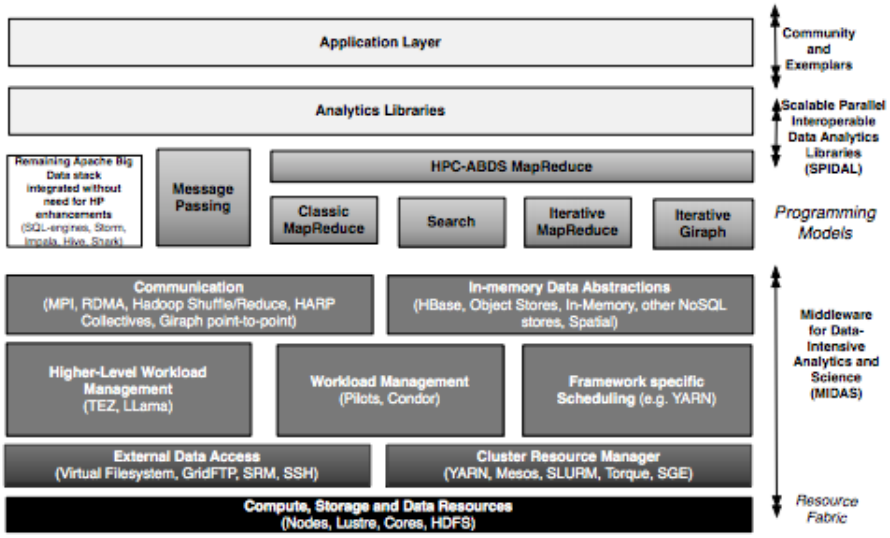

Figure 1: *Key Components of HPC-ABDS.*

The integrated platform, known as High Performance Computing Enhanced Big Data Stack (HPC-ABDS), therefore achieves a balance between performance and capabilities. The architecture and key components are shown in figure 1[2]. Two fundamental blocks are Middleware for Data-Intensive Analytics and Science (MIDAS) and the Scalable Parallel Interoperable Data Analytics Library (SPIDAL). More specifically, HPC-ABDS has 21 identified architecture layers, including IaaS Management from HPC to hypervisors, DevOps, File Systems, Cluster Resource Management, NoSQL, MapReduce, Application and Analytics, which as a whole are utilized in our project.

The problem tackled in our project is face detection with the OpenCV library. The project primarily aims to explore the HPC-ABD stack and its architecture. As an overarching big data software stack, HPC-ABDS encompasses over 350 software components, far more than one would likely find in any typical big data project. In our project, only a handful of applications are involved. Moreover, we focus on acquiring first-hand experience with those applications and gaining insights in HPC-ABDS performance and functionalities.

The project deals with detecting human faces in images, which we define as locating and segregating human face regions from arrays of images obtained from video. The project is chosen based on two major reasons. In the first place, video data are typical streaming data, on which HPC-ABDS has remarkable advantages over traditional data processing infrastructures. Secondly, face recognition, in fact, has extensive applications in areas such as content-based image retrieval, video conferencing, surveillance and security control systems, and intelligent human-computer interfaces. Since our project goal is to study big data open source software applications, rather than to study algorithms in computer vision, OpenCV is adopted for face detection. For simplicity, simulated streaming data in the form of a large

array of still images, instead of real streaming data obtained from videos, will be used.

PROBLEM STATEMENT

Research interests in facial recognition systems have been fueled for years by an ever-increasing concern for security, not only for individuals, but also for nations. Security control systems are almost everywhere, at entrances of buildings, embedded in automatic teller machines, and at the customs checkpoints of almost every country. The face recognition system is based on face detection in video images which are captured by cameras. Given features of video data, applications are required to be capable of real-time processing of high-volume data streams. However, both conventional data processing platforms and state-of-art data processing approaches are inadequate for the job, since they struggle to extract valuable information from massive volumes of data. Hence, in this project, we aim to propose an alternative solution to the face detection problem, namely, creating solutions within the HPC-ABDS framework.

Our solution leverages a fair number of the layers of HPC-ABDS. First, OpenStack is the Infrastructure-as-a-Service tool used to manage the Chameleon computer cluster and is the most popular leading virtual machine manager in the US[2]. It manages aspects of virtual machines such as virtualization and resource sharing. The DevOps layer (Ansible) is responsible for automation of deployment, installation and configuration of software applications, such as Hadoop and Java. File systems (HDFS) are employed to enable computing on distributed data. Cluster Resource Management (Yarn) is crucial for scheduling jobs submitted to computing resources and cluster management. MapReduce (Hadoop) implements programming models for parallel computing, which greatly promotes data processing scalability. Though not an ABDS technology, an open source library, OpenCV, is introduced to provide the necessary analytics for face detection. Finally, the layer of NoSQL (MongoDB) accommodates persistant storage of our results.

PURPOSE AND OBJECTIVES

As stated previously, the primary goal we hope to realize via the project is to explore HPC-ABDS. Through the construction of an automated face detection system, we would like to obtain an in-depth understanding of the performance and functionality of the big data software stack. Furthermore, we aim to acquire experience interacting with the technologies of HPC-ABDS and studying their performance and functionalities.

Concretely, given an array of images as our input data, the objective is to build executables with OpenCV and the Java programming language to output coordinates of bounding rectangles representing detected faces.

IMPLEMENTATION

Implementation of our automated face detection system consists of 3 components: deployment of Hadoop with Ansible, development of MapReduce functionality with OpenCV, and finally installation of MongoDB with Hadoop. Details of each step are presented below.

Deployment of Hadoop With Ansible

Our deployment of Hadoop with Ansible is divide into logical roles. The first role installs the Oracle 7 Java Development Kit on all nodes in typical fashion. The second role, which installs the Hadoop software on the necessary nodes, contains a number of steps. We first download the Hadoop tar file, extract it and install it. We also configure the Linux environment so that our JDK can be used with Hadoop. In the Hadoop configuration role, we configure core-site.xml, yarn-site.xml, mapred-site.xml on a master node as well as our slave nodes. Each of these roles can be applied using a single Ansible playbook.

When attempting to configure Hadoop manually, one quickly realizes that it is a task much better suited for a machine. Automating this task as quickly as possible, therefore, becomes not only a matter of efficiency, but of sanity.

Development of MapReduce Function With OpenCV

As one of the most widely adopted computer vision software libraries today, OpenCV presents itself as an intuitive choice for integrating face detection analytics into a distributed software system. Given an image as input, the OpenCV library can perform object detection with a single function call. However, deploying OpenCV with Hadoop raises two significant technical challenges. First, OpenCV is written in C++, which is not readily compatible with Hadoop. Second, OpenCV in and of itself provides no mechanism for feeding binary images into Hadoop's MapReduce implementation.

Though OpenCV is written in C++, integration with Java is possible by virtue of Java object bindings which allow Java code to call a shared object library compiled from OpenCV and installed on a Linux machine.

Applying Hadoop's MapReduce to binary image data is feasible thanks to third-party tools. In our case, the software package HIPI, Hadoop Image Processing Interface, solves the problem. The solution provided by HIPI is to convert an array of binary image files into a single data file which includes all images encoded as text. With this file as MapReduce's input, image data can be extracted in the mapper function and converted manually into an OpenCV Mat object instance. We acknowledge Dinesh Malav for sharing code to this end on his website, which has been reused for our purposes.

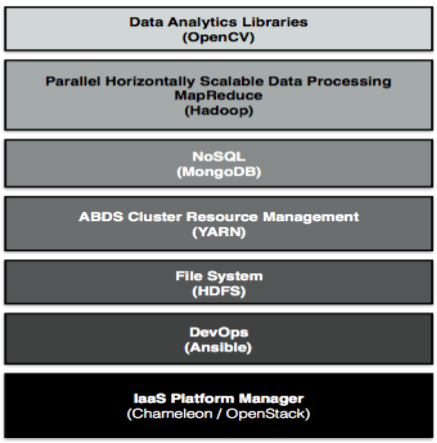

Figure 2: *Architecture of Automated Face Detection System.*

Installation of MongoDB With Hadoop

The final component of our system is the persistent storage of results with MongoDB. MongoDB is easily deployed with Ansible and provides Java device drivers, so technical overhead involved with integration is minimal. Committing results to the MongoDB server becomes simply a matter of deploying MongoDB and choosing a data structure with which to store the resulting data.

RESULTS

As it has been demonstrated in preceding sections, to build the automated face detection system, we employ OpenStack, Yarn, Ansible, MongoDB, Hadoop, HDFS and OpenCV. The architecture of the system is shown in Figure 2. Furthermore, we conduct an experiment in which we run MapReduce function on different numbers of virtual machines (Hadoop slave nodes) and compare performance. Results are summarized in Table 1.

Table 1: Summary of Run Time of MapReduce Function

Number of Hadoop Slave Nodes	Run Time (in sec)
2	284
4	208
7	220

FINDINGS

As hoped, given the open source software tools available to us, it was feasible to deploy a distributed face detection system using OpenCV and Hadoop as principal building blocks, despite a combined lack of experience in this domain.

Timing our software system on the INRIA dataset shows, as expected, that the time taken to run decreases from 284 seconds to 208 seconds as the number of slave nodes increases from 2 to 4. However, when we add another 3 slaves, the observed time taken to run face detection is surprisingly longer. The results are based on one experiment and performance of nodes in Chameleon seems to vary greatly over time. For further study, one can repeat the experiment and examine average run time with different numbers of slave nodes.

It is perhaps worthwhile to point out that the order in which we accomplished the individual tasks involved was exactly the opposite from what had originally been planned. The original intent was to implement basic face detection, then to run in parallel, and finally to automate deployment. In the end, our initial implementation of face detection turned out to be incompatible with Hadoop, and deploying Hadoop without DevOps automation proved to be intolerably inefficient and frustrating.

REFERENCES

Paper

[1] G. Fox, J. Qiu, S. Jha, S. Kamburugamuve and A. Luckow "HPC-ABDS High Performance Computing Enhanced Apache Big Data Stack", 2015, the 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, held in Shenzhen, Guangdong, China, Available at: http://grids.ucs.indiana.edu/ptliupages/publications/HPC-ABDSDescribedv2.pdf

[2] G. Fox, J. Qiu, and S. Jha, "High Performance High Functionality Big Data Software Stack", 2014. [Online]. Available at: http://www.exascale.org/bdec/sites/www.exascale.org.bdec/files/whitepapers/fox.pdf

[3] E. Hjelmas, "Face Detection: A Survey", 2001, *Computer Vision and Image Understanding,* 83, pp. 236-274.

[4] U. Park, A. K. Jain and A. Ross, "Face Recognition in Video: Adaptive Fusion of Multiple Matchers", 2007, Proc. of IEEE Computer Society Workshop on Biometrics(CVPRW).

Code

B. Abdul-Wahid, G. v. Laszewski, S. Kandagatla (2016) Big Data Stack [Source Code]. https://github.com/futuresystems/big-data-stack

D. Malav (2015) Image processing using OpenCV on Hadoop using HIPI [Source Code]. http://dinesh-malav.blogspot.com/2015/05/image-processing-using-opencv-on-hadoop.html

M. Caballer (2016) Ansible Role Hadoop [Source Code]. https://github.com/indigo-dc/ansible-role-hadoop

S. Arietta, J. Lawrence, L. Liu, C. Sweeney (2016) HIPI: Getting Started [Source Code]. http://hipi.cs.virginia.edu/gettingstarted.html

Datasets

N. Dalal (2005) INRIA Person Dataset [Dataset]. http://pascal.inrialpes.fr/data/human/