

Face Detection Using Hadoop Map-Reduce Framework

Tanmay Bagwe¹, Nishit Darji², Jayesh Gunjal³, Nisha Vanjari⁴

^{1,2,3} Student, Department Of Computer Engineering, K.J Somaiya Institute Of Engineering & IT, Maharashtra, India

⁴ Professor, Department Of Computer Engineering, K.J Somaiya Institute Of Engineering & IT, Maharashtra, India

Abstract:-As multimedia contents on web are increasing at a large scale as compared to available memory and processor's speed for processing these large amount of data, it is necessary to manage data in such a way so that it gives low performance overhead. Distributed Computing is one of the approach for efficiently handling the Big data. Hadoop is an opensource framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. In this paper face detection is taken as an case example for analyzing the performance of the Hadoop framework. We have discussed this example for the single node cluster where input image will be received from the user and Haar Feature-based Cascade Classifier for Object Detection algorithm defined in OpenCV library is used for face detection. Java Native Interface (JNI) is used to integrate OpenCV into interface.

Index Terms- HDFS, Hadoop, HDFS, MapReduce, Distributed Computing.

1. INTRODUCTION

Although the computing power of machines is increasing at a very high speed. Almost every 3 years, CPU's computing power increased twice. However size of the files keeps increasing also in an amazing rate. To store such colossal amount of data instead of using a simple Client Server architecture it'll be better to use an architecture wherein the data exhibits the property of logical independence. A system where in the data must be distributed on a large number of workstations so that it may reduce the burden of analysis on a single machine. Video processing is very well suited to distributed system implementation. Processing in the Hadoop is inherently distributed. Hadoop supports parallel running of applications on large clusters of commodity hardware.

"Hadoop Library is designed to scale up from single servers to thousands of machines, each offering local computation and storage. Rather than rely on hardware to deliver high-availability, the library itself is designed to detect and handle failures at the application layer, so delivering a highly available service on top of a cluster of computers, each of which may be prone to failures." –Apache. Our idea is very simple. Assume there is a very large video data base. Giving a set of video frames or image, we hope to find it from that database, and tell the position of that input file. The idea is simple but it is very useful in different aspects. The key point of this project is building application with high scalability. So That Even when database size is increased, the application can still handle it.

2. PROPOSED SYSTEM

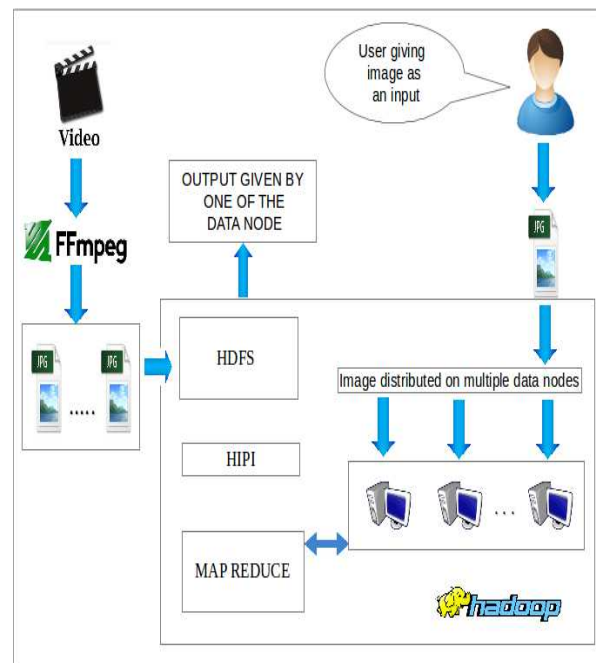


Figure 1 : System Architecture

As a multimedia content is growing at an increasing rate, it requires scalable and efficient solution to perform job scheduling and video analysis task. We present a novel implementation utilizing Apache Hadoop MapReduce framework for both analysis job scheduling and video data distribution. Our project will have face recognition as a case example for distributing the task among the multiple nodes. As shown in the system architecture FFMpeg will take the video and convert it into the images. These images are stored into the HDFS. We will be having one namenode and multiple datanode, where user enters input image onto the namenode and this namenode

further distributes this image among several datanodes to accomplish the task of face recognition.

The system architecture includes following components:

2.1. FFMpeg:

To convert video into images.

2.2. Apache Hadoop:

Opensource distributed framework.

2.2.1. HDFS (Hadoop Distributed File System) :

HDFS is designed to reliably store very large files across machines in a large cluster. It is inspired by the Google File System. HDFS is composed of NameNode and DataNode. HDFS stores each file as a sequence of blocks (currently 64 MB by default) with all blocks in a file the same size except for the last block. Blocks belonging to a file are replicated for fault tolerance. The block size and replication factor are configurable per file. Files in HDFS are write-once and can have only one writer at any given time. [5]

2.2.2. HIPI (Hadoop Image Processing Interface) :

HIPI is a library for Hadoop's MapReduce framework that provides an API for performing image processing tasks in a distributed computing environment. 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. A HIB can be created from an already existing set of images or directly through some other source (e.g. our Distributed Downloader example).

In order to improve the efficiency of some jobs, HIPI allows a user to specify a culling function that discards images that do not meet a specified set of criteria (e.g. the image must be less than 10 megapixels). The user-specified CullMapper class is then invoked on each image that passes the culling test. Images are presented to this class as a FloatImage and an associated ImageHeader. Although HIPI does not directly modify any of the default Hadoop MapReduce behavior once the Mapper's take over, a user can modify execution parameters specific to image processing tasks through the HipiJob object during setup.

2.2.3. MapReduce :

Hadoop's Programming Model.

2.3. OpenCV/JavaCV [6] :

Programming tool to perform face recognition task. It has C++, C, Python and Java interfaces and supports

Windows, Linux, Mac OS, iOS and Android. OpenCV was designed for computational efficiency and with a strong focus on real-time applications. Written in optimized C/C++, the library can take advantage of multi-core processing. Usage ranges from interactive art, to mines inspection, stitching maps on the web or through advanced robotics

3. MODULES

3.1. Video To Image Conversion Using FFMPEG :

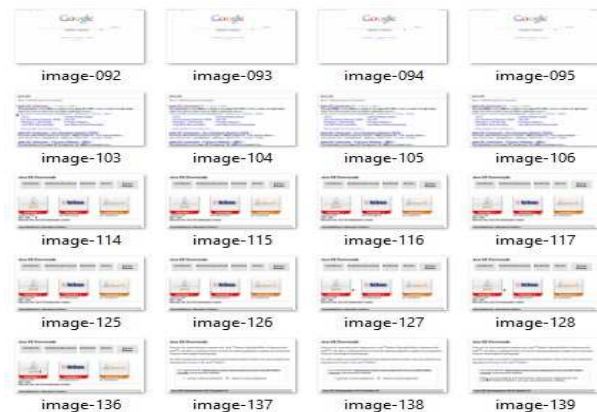


Figure 2 : Output Images

FFmpeg is a free software project that produces libraries and programs for handling multimedia data. FFmpeg includes libavcodec, an audio/video codec library used by several other projects, libavformat, an audio/video container mux and demux library, and the ffmpeg command line program for transcoding multimedia files. FFmpeg is published under the GNU Lesser General Public License 2.1+ or GNU General Public License 2+ (depending on which options are enabled)

```
ffmpeg -i inputfile.avi -r 1 -f image2 image-
%3d.jpeg
```

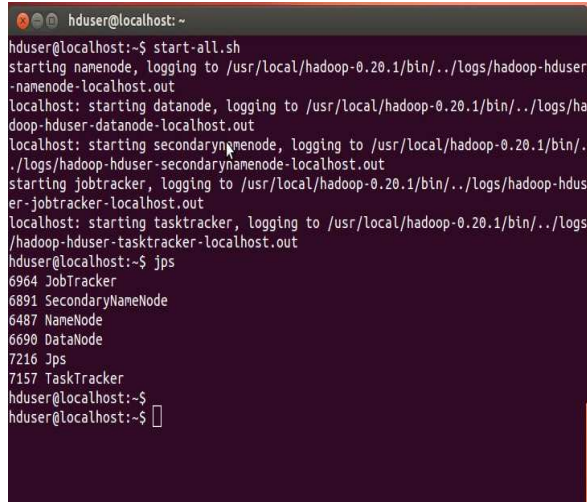
Now, let us see what all these different flags in the above command means.

- **-r** This is used to set the frame rate of video. i.e. no. of frames to be extracted into images per second. The default value is 25, using which, would have yielded a large number of images.
- **-f** This option defines the format we want to force/use, although removing this option shouldn't cause any problem.
- **image-%3d.jpeg** By %3d, we mean that we want the naming of the image files to be of the format "image-001.jpeg, image-002.jpeg.." and so on. If we had used image-%2d the names would have been image-

01.jpeg, image-02.jpeg. You can use any format as per your choice.

We can also define the image size of the extracted images using the -s flag. The default option is to use the image size same as the video resolution.

3.2. Single Node Hadoop :



```

hduser@localhost:~$ start-all.sh
starting namenode, logging to /usr/local/hadoop-0.20.1/bin/../logs/hadoop-hduser-
namenode-localhost.out
localhost: starting datanode, logging to /usr/local/hadoop-0.20.1/bin/../logs/ha
dooop-hduser-datanode-localhost.out
localhost: starting secondarynamenode, logging to /usr/local/hadoop-0.20.1/bin/.
../logs/hadoop-hduser-secondarynamenode-localhost.out
starting jobtracker, logging to /usr/local/hadoop-0.20.1/bin/../logs/hadoop-hdus
er-jobtracker-localhost.out
localhost: starting tasktracker, logging to /usr/local/hadoop-0.20.1/bin/../logs
/hadoop-hduser-tasktracker-localhost.out
hduser@localhost:~$ jps
6964 JobTracker
6891 SecondaryNameNode
6487 NameNode
6698 DataNode
7216 Jps
7157 TaskTracker
hduser@localhost:~$
hduser@localhost:~$

```

Figure 3 : Starting Hadoop

3.3. Face Detection Using MapReduce In OPENCV:

In order to apply face detection algorithm to each image, map function has to get the whole image contents as a single input record. HDFS creates splits from input files according to the configured split-size parameter. These InputSplits become the input to the MapTasks. Creating splits from files causes some files to be divided into more than one split, if their file size is larger than the split-size. Moreover, a set of files can become one InputSplit if the total size of input files is smaller than the split size. In other words, some records may not be represented as the binary content of each file. This explains why new classes for input format and record reader have to be implemented to enable MapTask to process each binary file as a whole.

In this paper, ImageFileInputFormat class is developed by deriving the FileInputFormat class of Hadoop. ImageFileInputFormat creates FileSplit from each image file. Because, each image file is not splitted, binary image content is not corrupted. In addition, ImageFileRecordReader class is developed to create image records from FileSplits for map function by deriving Hadoop's RecordReader class. In that way pixel data of images are easily fetched from Hadoop input splits into image processing tasks (map tasks). After that point, any image processing algorithm can be applied to image content. In our case, map function of the Mapper class applies the face detection algorithm to image records. Haar Feature-based Cascade Classifier for Object Detection

algorithm defined in OpenCV library is used for face detection. Java Native Interface (JNI) is used to integrate OpenCV into interface. Implementation of map function is presented below. "FaceInfoString" is the variable that contains the information about detection properties such as image name and coordinates where faces are detected.

Algorithm Of Map [2]

```

Class : Mapper
Function : Map
Map (TEXT key(filename), BytesWritable
value(imgdata), OutputCollector)
getImgBinaryData_From_Value;
convertBinaryData_To_JavaImage;
InitializeOpenCV_Via_JNI Inter face ;
runOpenCV_HaarLikeFaceDetector ;
for each (DetectedFace)
createFaceBuffer_FaceSize ;
copyFacePixels_To_Buffer ;
create_Face InfoString ;
collectOutput :
set_key_FaceInfoString ;
set_value_FaceImgBuffer ;
end_foreach

```

Hadoop generates names of output files as strings with job identification numbers (e.g.: part 0000). After face detection, our image processing interface creates output files as detected face images. In order to identify these images easily, the output file names should contain detected image name and detected coordinate information (eg: SourceImageName(100,150).jpg). ImageFileOutputFormat class is developed to store output files as images with desired naming. ReduceTask is not used for face extraction because each MapTask generates unique outputs to be stored in the HDFS. Each task processes only one image, creates output and exits. This approach degrades the system performance seriously. The overhead comes from initialization times of huge number of tasks.

In order to decrease the number of tasks, firstly, converting small-size files into single large-size file and process technique is implemented. SequenceFile is a Hadoop file type which is used for merging many small-size files [7]. SequenceFile is the most common solution for small file problem in HDFS. Many small files are packed as a single large-size file containing small-size files as indexed elements in <key, value> format. Key is file index information and value is the file data. This conversion is done by writing a conversion job that gets small-files as input and SequenceFile as output. Although general performance is increased with SequenceFile usage, input images do not preserve their image formats after merging. Preprocessing is also required for each addition of new input image set. Small files

cannot be directly accessed in SequenceFile, whole SequenceFile has to be processed to obtain an image data as one element.

Secondly, combining set of images as one InputSplit technique is implemented to optimize small-size image processing in HDFS. Hadoop CombineFileInputFormat can combine multiple files and create InputSplits from this set of files. In addition to that, CombineFileInputFormat selects files which are in the same node to be combined as InputSplit. So, amount of data to be transferred from node to node decreases and general performance increases. CombineFileInputFormat is an abstract class that does not work with image files directly. We developed CombineImageInputFormat derived from CombineFileInputFormat to create CombineFileSplit as set of image. MultiImageRecordReader class is developed to create records from CombineFileSplit. This record reader uses ImageFileRecordReader class to make each image content as single record to map function. ImageFileOutputFormat is used to create output files from detected face images and stored into HDFS.

- [2]. İ. Demir, A. Sayar, "Hadoop Optimization for Massive Image Processing: Case Study Face Detection", International Journal of Computers Communication and Control ISSN 1841-9836, 9(6):664-671, December, 2014.
- [3]. <http://hadoop.apache.org/>.
- [4]. http://hadoop.apache.org/docs/r0.18.3/mapred_tutorial.html
- [5]. http://hadoop.apache.org/docs/r1.2.1/hdfs_design.html
- [6]. <http://opencv.org>
- [7]. <http://wiki.apache.org/hadoop/SequenceFile>
- [8]. <http://hipi.cs.virginia.edu>
- [9]. <https://developer.yahoo.com/hadoop/tutorial/>

4. EXPECTED RESULT

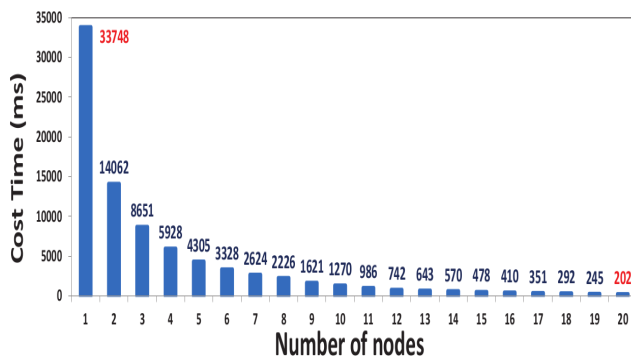


Figure 4 : Time cost of face recognition under different number of nodes in the Cloud.[1]

As shown in the figure 4 as number of nodes on the cloud increases the time required to perform face recognition task decreases.

Acknowledgments

We would like to thank Prof. Nisha Vanjari for her excellent guidance, Prof. Nilambari Joshi for encouraging us to implement such a unique project and also we are thankful to Prof. Jignasha Dalal for giving us such big opportunity to work on this project.

REFERENCES

- [1]. Xi Wang, Xi Zhao, Varun Prakash, Zhimin Gao, Tao Feng, Omprakash Gnawali, Weidong Shi. "Person-Of-Interest Detection System Using Cloud-Supported Computerized- Eyewear".