IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661,p-ISSN: 2278-8727, Volume 18, Issue 2, Ver. III (Mar-Apr. 2016), PP 82-92
www.iosrjournals.org

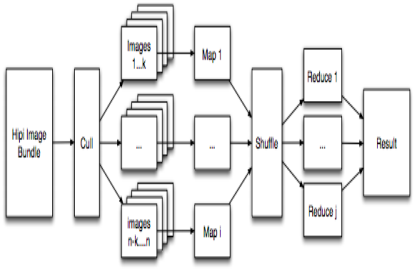
A Practical Approach Forparallel Image Processing

Neha Mangla[1], Shanthi Mahesh[2], Suhash.A.Bhyratae[3]
[1,2,3] *Associate Profesor A.I.T,Banglore*

***Abstract**: This review paper tries to solve the problem of processing big data of images on Apache Hadoop using Hadoop Image Processing Interface (HIPI) for storing and efficient distributed processing, combined with OpenCV, an open source library of rich image processing algorithms. This paper demonstrates how HIPI and OpenCV can be used together to count total number of faces in big image dataset.*
***Keywords:** Hadoop, MapReduce, HIPI, OpenCV.*

I. Background

Processing large set of images on a single machine can be very time consuming and costly. HIPI is an image processing library designed to be used with the Apache Hadoop MapReduce, a software framework for sorting and processing big data in a distributed fashion on large cluster of commodity hardware. HIPI facilitates efficient and high-throughput image processing with MapReduce style parallel programs typically executed on a cluster. It provides a solution for how to store a large collection of images on the Hadoop Distributed File System (HDFS) and make them available for efficient distributed processing. OpenCV (Open Source Computer Vision) is an open source library of rich image processing algorithms, mainly aimed at real time computer vision. Starting with OpenCV 2.4.4, OpenCV supports Java Development which can be used with Apache Hadoop.



II. Methodology

Overview of Steps

1. Download VMW are Fusion
2. Download and Setup Cloudera Quickstart VM 5.4.x
3. Getting Started with HIPI ,Setup Hadoop, Install Apache Ant, Install and build HIPI Running a sample HIPI MapReduce Program
4. Getting Started with Open CV Using Java., Download Open CV source,CMake build system,Build Open CV for Java,Running OpenCV Face Detection Program
5. Configure Hadoop for OpenCV
6. HIPI with OpenCV
7. Build FaceCount.java as facecount.jar
8. Run FaceCount MapReduce job

Big Data Set: Test images for face detection
Input: Image data containing 158 image (34MB)
Format: png image files.
Source: http://vasc.ri.cmu.edu/idb/images/face/frontal_images/
Downloaded image were in gif format, I used Mac OSX Preview program to convert these to png format.Other sources for face detection image datasets:
https://www.bioid.com/About/BioID-Face-Database
https://facedetection.com/datasets/

Technologies Used

VMWare Fusion(Software hypervisor for running Cloudera Quickstart VM),Cloudera Quickstart VM(VM for single node Hadoop cluster for testing and running map),IntelliJ IDEA 14 CE(Java IDE for editing

and compiling Java code),Hadoop Image Processing Interface (HIPI)(Image processing library designed to be used with the Apache Hadoop MapReduce parallel programming framework, for storing large collection of images on HDFS and efficient distributed processing.
),OpenCV(An image processing library aimed at real-time comput),Apache Hadoop(Distributed processing of large data sets)

Getting started with OpenCV using Java

OpenCV is an image processing library. It contains a large collection of image processing functions. Starting from version 2.4.4 OpenCV includes desktop Java bindings. We will use version 2.4.11 to build Java bindings and use it with HIPI to run image processing.The OpenCV is built with CMake, download CMake binaries from http://www.cmake.org/download/and untar the tar bundles to ~/Project/opencv directory:. We need toConfigure OpenCV for builds on Linux This will create a jar containing the Java interface (bin/opencv-2411.jar) and a native dynamic library containing Java bindings and all the OpenCV stuff (lib/libopencv_java2411.so). We'll use these files to build and run OpenCV program.

Running Open CV Face Detection Program

Following steps are run to make sure OpenCV is setup correctly and works as expected.Create a new directory sample and Create an ant build.xml file in it.

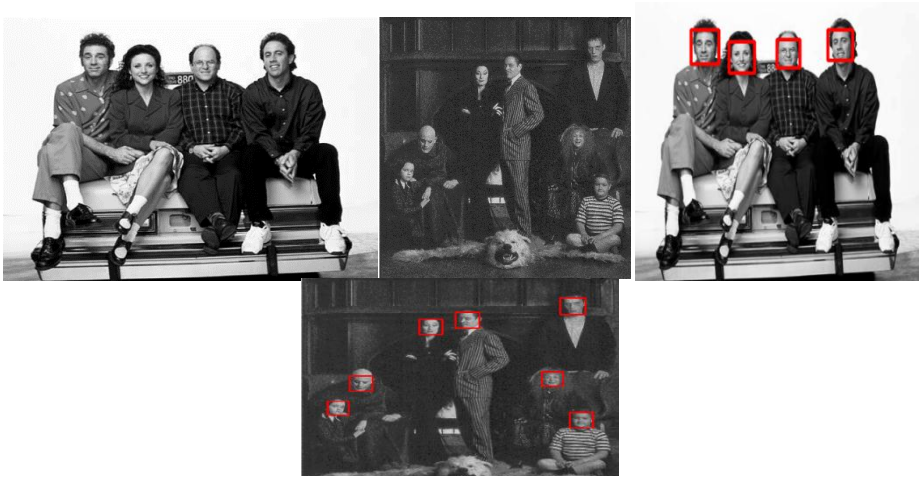
[cloudera@quickstart opencv]$ mkdir sample && cd sample
[cloudera@quickstart sample]$ vi build.xml
a program using OpenCV to detect number of faces in an image.
DetectFaces.java

```
import org.opencv.core.Core;
import org.opencv.core.Mat;
import org.opencv.core.Scalar;
import org.opencv.highgui.*;
import org.opencv.core.MatOfRect;
import org.opencv.core.Point;
import org.opencv.core.Rect;
import org.opencv.objdetect.CascadeClassifier;
import java.io.File;
public class DetectFaces {
  public void run(String imageFile) {
     System.out.println("\nRunning DetectFaceDemo");
     // Create a face detector from the cascade file in the resources
     // directory.
     String xmlPath = "/home/cloudera/project/opencv-examples/lbpcascade_frontalface.xml";
     System.out.println(xmlPath);
     CascadeClassifier faceDetector = new CascadeClassifier(xmlPath);
     Mat image = Highgui.imread(imageFile);
     // Detect faces in the image.
     // MatOfRect is a special container class for Rect.
     MatOfRect faceDetections = new MatOfRect();
     faceDetector.detectMultiScale(image, faceDetections);
     System.out.println(String.format("Detected %s faces", faceDetections.toArray().length));
     // Draw a bounding box around each face.
     for (Rect rect : faceDetections.toArray()) {
        Core.rectangle(image, new Point(rect.x, rect.y), new Point(rect.x + rect.width, rect.y + rect.height), new
Scalar(0, 255, 0));
     }
     File f = new File(imageFile);
     System.out.println(f.getName());
     // Save the visualized detection.
     String filename = f.getName();
     System.out.println(String.format("Writing %s", filename));
     Highgui.imwrite(filename, image);
  }}
```

Main.java

```
importorg.opencv.core.Core;
```

```
import java.io.File;
public class Main {
  public static void main(String... args) {
     System.loadLibrary(Core.NATIVE_LIBRARY_NAME);
if (args.length == 0) {
  System.err.println("Usage Main /path/to/images");
  System.exit(1);
}
     File[] files = new File(args[0]).listFiles();
     showFiles(files);
  }
  public static void showFiles(File[] files) {
     DetectFaces faces = new DetectFaces();
     for (File file : files) {
```



```
  if (file.isDirectory()) {
            System.out.println("Directory: " + file.getName());
            showFiles(file.listFiles()); // Calls same method again.
         } else {
            System.out.println("File: " + file.getAbsolutePath());
            faces.run(file.getAbsolutePath());
         }}}}
```

Build Face Detection Java Program

[cloudera@quickstart sample]$ant -DocvJarDir=/home/cloudera/Project/opencv/opencv-2.4.11/bin -DocvLibDir=/home/cloudera/Project/opencv/opencv-2.4.11/lib jar

```
Buildfile: /home/cloudera/Project/opencv/sample/build.xml
compile:
   [mkdir] Created dir: /home/cloudera/Project/opencv/sample/build/classes
   [javac] Compiling 2 source files to /home/cloudera/Project/opencv/sample/build/classes
jar:
   [mkdir] Created dir: /home/cloudera/Project/opencv/sample/build/jar
     [jar] Building jar: /home/cloudera/Project/opencv/sample/build/jar/Main.jar
```

Build Successful

Total time: 3 seconds

This build creates a build/jar/Main.jar file which can be used to detect faces from images stored in a directory:

[cloudera@quickstart sample]$ java -cp ../opencv-2.4.11/bin/opencv-2411.jar:build/jar/Main.jar -Djava.library.path=../opencv-2.4.11/lib Main /mnt/hgfs/CSCEI63/project/images2

```
File: /mnt/hgfs/CSCEI63/project/images2/addams-family.png
Running DetectFaceDemo
/home/cloudera/Project/opencv/sample/lbpcascade_frontalface.xml
Detected 7 faces
addams-family.png
```

Writing addams-family.png

Open CV detected faces

OpenCV does fairly good job detecting front facing faces when lbpcascade_frontalface.xml classifier is used. There are other classifier provided by OpenCV which can detect rotate faces and other face orientations.

HIPI with OpenCV

This step details Java code for combining HIPI with OpenCV.HIPI uses HipiImageBundle class to represent collection of images on HDFS, and FloatImage for representing the image in memory. This FloatImage must be converted to OpenCV Mat format for image processing, counting face in this case.

Following method is used to convert FloatImage to Mat:

```
    // Convert HIPI FloatImage to OpenCV Mat
    public Mat convertFloatImageToOpenCVMat(FloatImage floatImage) {
       // Get dimensions of image
       int w = floatImage.getWidth();
       int h = floatImage.getHeight();
       // Get pointer to image data
       float[] valData = floatImage.getData();
       // Initialize 3 element array to hold RGB pixel average
       double[] rgb = {0.0,0.0,0.0};
       Mat mat = new Mat(h, w, CvType.CV_8UC3);
       // Traverse image pixel data in raster-scan order and update running average
       for (int j = 0; j < h; j++) {
          for (int i = 0; i < w; i++) {
             rgb[0] = (double) valData[(j*w+i)*3+0] * 255.0; // R
             rgb[1] = (double) valData[(j*w+i)*3+1] * 255.0; // G
             rgb[2] = (double) valData[(j*w+i)*3+2] * 255.0; // B
             mat.put(j, i, rgb);
          }}
              return mat;
    }
```

To count the number of faces from an image we need to create a CascadeClassifier which uses a classifier file. This file must be present on HDFS, this can be accomplished by using Job.addCacheFile method and later retrieve it in Mapper class.

```
   public int run(String[] args) throws Exception {
    ....
     // Initialize and configure MapReduce job
     Job job = Job.getInstance();
          ....
          // add cascade file
     job.addCacheFile(new
URI("/user/cloudera/lbpcascade_frontalface.xml#lbpcascade_frontalface.xml"));
     // Execute the MapReduce job and block until it complets
     boolean success = job.waitForCompletion(true);
     // Return success or failure
     return success ? 0 : 1;
  }
```

Override Mapper::setup method to load OpenCV native library and create CascadeClassifier for face detections:

```
          public static class FaceCountMapper extends Mapper<ImageHeader, FloatImage, IntWritable,
IntWritable> {
     // Create a face detector from the cascade file in the resources
     // directory.
     private CascadeClassifier faceDetector;
        public void setup(Context context)
           throws IOException, InterruptedException {
        // Load OpenCV native library
        try {
           System.loadLibrary(Core.NATIVE_LIBRARY_NAME);
        } catch (UnsatisfiedLinkError e) {
```

```
            System.err.println("Native code library failed to load.\n" + e +
Core.NATIVE_LIBRARY_NAME);
            System.exit(1);
        }
        // Load cached cascade file for front face detection and create CascadeClassifier
        if (context.getCacheFiles() != null && context.getCacheFiles().length >0) {
            URI mappingFileUri = context.getCacheFiles()[0];
            if (mappingFileUri != null) {
                faceDetector = new CascadeClassifier("./lbpcascade_frontalface.xml");
            } else {
                System.out.println(">>>>>> NO MAPPING FILE");
            }
        } else {
            System.out.println(">>>>>> NO CACHE FILES AT ALL");
        }
        super.setup(context);
    } // setup()
    ....
}
```

Full listing of FaceCount.java.

Mapper:

1. Load OpenCV native library
2. Create CascadeClassifier
3. Convert HIPI FloatImage to OpenCV Mat
4. Detect and count faces in the image
5. Write number of faces detected to context

Reducer:

1. Count number of files processed
2. Count number of faces detected
3. Output number of files and faces detected

FaceCount.java

```
import hipi.image.FloatImage;
import hipi.image.ImageHeader;
import hipi.imagebundle.mapreduce.ImageBundleInputFormat;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.opencv.core.*;
import org.opencv.objdetect.CascadeClassifier;
import java.io.IOException;
import java.net.URI;
public class FaceCount extends Configured implements Tool {
  public static class FaceCountMapper extends Mapper<ImageHeader, FloatImage, IntWritable, IntWritable> {
     // Create a face detector from the cascade file in the resources
     // directory.
    private CascadeClassifier faceDetector;
     // Convert HIPI FloatImage to OpenCV Mat
     public Mat convertFloatImageToOpenCVMat(FloatImage floatImage) {
        // Get dimensions of image
```

```
        int w = floatImage.getWidth();
        int h = floatImage.getHeight();
        // Get pointer to image data
        float[] valData = floatImage.getData();
        // Initialize 3 element array to hold RGB pixel average
        double[] rgb = {0.0,0.0,0.0};
        Mat mat = new Mat(h, w, CvType.CV_8UC3);
        // Traverse image pixel data in raster-scan order and update running average
        for (int j = 0; j < h; j++) {
           for (int i = 0; i < w; i++) {
              rgb[0] = (double) valData[(j*w+i)*3+0] * 255.0; // R
              rgb[1] = (double) valData[(j*w+i)*3+1] * 255.0; // G
              rgb[2] = (double) valData[(j*w+i)*3+2] * 255.0; // B
              mat.put(j, i, rgb);
           }}
        return mat;
     }
     // Count faces in image
     public int countFaces(Mat image) {
        // Detect faces in the image.
        // MatOfRect is a special container class for Rect.
        MatOfRect faceDetections = new MatOfRect();
        faceDetector.detectMultiScale(image, faceDetections);

        return faceDetections.toArray().length;
     }
     public void setup(Context context)
           throws IOException, InterruptedException {

        // Load OpenCV native library
        try {
           System.loadLibrary(Core.NATIVE_LIBRARY_NAME);
        } catch (UnsatisfiedLinkError e) {
           System.err.println("Native code library failed to load.\n" + e +
Core.NATIVE_LIBRARY_NAME);
           System.exit(1);
        }
        // Load cached cascade file for front face detection and create CascadeClassifier
        if (context.getCacheFiles() != null && context.getCacheFiles().length >0) {
           URI mappingFileUri = context.getCacheFiles()[0];
           if (mappingFileUri != null) {
              faceDetector = new CascadeClassifier("./lbpcascade_frontalface.xml");
           } else {
              System.out.println(">>>>>> NO MAPPING FILE");
           }
        } else {
           System.out.println(">>>>>> NO CACHE FILES AT ALL");
        }
        super.setup(context);
     } // setup()
     public void map(ImageHeader key, FloatImage value, Context context)
           throws IOException, InterruptedException {
        // Verify that image was properly decoded, is of sufficient size, and has three color channels (RGB)
        if (value != null && value.getWidth() >1 && value.getHeight() >1 && value.getBands() == 3) {
           Mat cvImage = this.convertFloatImageToOpenCVMat(value);
           int faces = this.countFaces(cvImage);
           System.out.println(">>>>>> Detected Faces: " + Integer.toString(faces));
           // Emit record to reducer
           context.write(new IntWritable(1), new IntWritable(faces));
        } // If (value != null...
```

```
    } // map()
  }
  public static class FaceCountReducer extends Reducer<IntWritable, IntWritable, IntWritable, Text> {
    public void reduce(IntWritable key, Iterable<IntWritable> values, Context context)
        throws IOException, InterruptedException {
      // Initialize a counter and iterate over IntWritable/FloatImage records from mapper
      int total = 0;
      int images = 0;
      for (IntWritable val : values) {
        total += val.get();
        images++;
      }
      String result = String.format("Total face detected: %d", total);
      // Emit output of job which will be written to HDFS
      context.write(new IntWritable(images), new Text(result));
    } // reduce()
  }
  public int run(String[] args) throws Exception {
    // Check input arguments
    if (args.length != 2) {
      System.out.println("Usage: firstprog <input HIB><output directory>");
      System.exit(0);
    }
    // Initialize and configure MapReduce job
    Job job = Job.getInstance();
    // Set input format class which parses the input HIB and spawns map tasks
    job.setInputFormatClass(ImageBundleInputFormat.class);
    // Set the driver, mapper, and reducer classes which express the computation
    job.setJarByClass(FaceCount.class);
    job.setMapperClass(FaceCountMapper.class);
    job.setReducerClass(FaceCountReducer.class);
    // Set the types for the key/value pairs passed to/from map and reduce layers
    job.setMapOutputKeyClass(IntWritable.class);
    job.setMapOutputValueClass(IntWritable.class);
    job.setOutputKeyClass(IntWritable.class);
    job.setOutputValueClass(Text.class);
    // Set the input and output paths on the HDFS
    FileInputFormat.setInputPaths(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    // add cascade file
    job.addCacheFile(new
URI("/user/cloudera/lbpcascade_frontalface.xml#lbpcascade_frontalface.xml"));

    // Execute the MapReduce job and block until it complets
    boolean success = job.waitForCompletion(true);
    // Return success or failure
    return success ? 0 : 1;
  }
  public static void main(String[] args) throws Exception {
    ToolRunner.run(new FaceCount(), args);
    System.exit(0);
  }}
```

Build FaceCount.java as facecount.jar

Create new facecount directory in hipi folder (where HIPI was built) and copy FaceCount.java from previous step.

```
[cloudera@quickstart hipi]$ pwd
/home/cloudera/Project/hipi
[cloudera@quickstart hipi]$ mkdir facecount
```

[cloudera@quickstart hipi]$ cp /mnt/hgfs/CSCEI63/project/hipi/src/FaceCount.java facecount/
[cloudera@quickstart hipi]$ ls
3rdparty data **facecount** libsrc README.md sample
bin doc hipiwrapper license.txt release tool
build.xml examples lib my.diff run.sh util
[cloudera@quickstart hipi]$ ls facecount/
FaceCount.java
Make changes to HIPI build.xml ant script to build link to OpenCV jar file and add new build target facecount.
build.xml

```
<project basedir="." default="all">
<target name="setup">
....
  <!-- opencv dependencies -->
  <property name="opencv.jar" value="../opencv/opencv-2.4.11/bin/opencv-2411.jar" />
  <echo message="Properties set."/>
</target>
<target name="compile" depends="setup,test_settings,hipi">
  <mkdir dir="bin" />
   <!-- Compile -->
  <javac debug="yes" nowarn="on" includeantruntime="no" srcdir="${srcdir}" destdir="./bin"
classpath="${hadoop.classpath}:./lib/hipi-${hipi.version}.jar:${opencv.jar}">
    <compilerarg value="-Xlint:deprecation" />
  </javac>
   <!-- Create the jar -->
  <jar destfile="${jardir}/${jarfilename}" basedir="./bin">
    <zipfileset src="./lib/hipi-${hipi.version}.jar" />
    <zipfileset src="${opencv.jar}" />
    <manifest>
       <attribute name="Main-Class" value="${mainclass}" />
    </manifest>
  </jar>
 </target>
....
 <target name="facecount">
    <antcall target="compile">
    <param name="srcdir" value="facecount" />
    <param name="jarfilename" value="facecount.jar" />
    <param name="jardir" value="facecount" />
    <param name="mainclass" value="FaceCount" />
    </antcall>
 </target>
 <target name="all" depends="hipi,hibimport,downloader,dumphib,jpegfromhib,createsequencefile,covariance"
/>
 <!-- Clean -->
 <target name="clean">
   <delete dir="lib" />
   <delete dir="bin" />
   <delete>
     <fileset dir="." includes="examples/*.jar,experiments/*.jar" />
   </delete>
 </target>
</project>
```

Build FaceCount.java

```
[cloudera@quickstart hipi]$ ant facecount
Buildfile: /home/cloudera/Project/hipi/build.xml
facecount:
setup:
    [echo] Setting properties for build task...
    [echo] Properties set.
```

```
test_settings:
   [echo] Confirming that hadoop settings are set...
   [echo] Properties are specified properly.
hipi:
   [echo] Building the hipi library...
hipi:
  [javac] Compiling 30 source files to /home/cloudera/Project/hipi/lib
    [jar] Building jar: /home/cloudera/Project/hipi/lib/hipi-2.0.jar
   [echo] Hipi library built.
compile:
    [jar] Building jar: /home/cloudera/Project/hipi/facecount/facecount.jar

Build Successful
Total time: 12 seconds
```

Check facecount.jar is built under facecount directory

[cloudera@quickstart hipi]$ ls facecount/

facecount.jar FaceCount.java

Run FaceCount MapReduce job

Setup input images

[cloudera@quickstart hipi]$ ls /mnt/hgfs/CSCEI63/project/images-png/

```
217.png            eugene.png              patio.png
221.png            ew-courtney-david.png   people.png
3.png              ew-friends.png          pict_28.png
….
```

Create HIB

The primary input type to a HIPI program is a HipiImageBundle (HIB), which stores a collection of images on the Hadoop Distributed File System (HDFS). Use the hibimport tool to create a HIB (project/input.hib) from a collection of images on your local file system located in the directory /mnt/hgfs/CSCEI63/project/images-png/ by executing the following command from the HIPI root directory

[cloudera@quickstart hipi]$ hadoop fs -mkdir project

[cloudera@quickstart hipi]$ hadoop jar tool/hibimport.jar /mnt/hgfs/CSCEI63/project/images-png/ project/input.hib

```
** added: 217.png
** added: 221.png
** added: 3.png
** added: addams-family.png
** added: aeon1a.png
** added: aerosmith-double.png
.
.
.
** added: werbg04.png
** added: window.png
** added: wxm.png
** added: yellow-pages.png
** added: ysato.png
Created: project/input.hib and project/input.hib.dat
```

Run Map Reduce

Create a run-facecount.sh script to clean previous output file and execute mapreduce job:

run-facecount.sh

```
#!/bin/bash
hadoop fs -rm -R project/output
hadoop jar facecount/facecount.jar project/input.hib project/output
```

[cloudera@quickstart hipi]$ bash run-facecount.sh

15/05/12 16:48:06 INFO fs.TrashPolicyDefault: Namenode trash configuration: Deletion interval = 0 minutes, Emptier interval = 0 minutes.

```
Deleted project/output
15/05/12 16:48:17 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
15/05/12 16:48:20 WARN mapreduce.JobSubmitter: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your application with ToolRunner to remedy this.
15/05/12 16:48:21 INFO input.FileInputFormat: Total input paths to process : 1
Spawned 1map tasks
15/05/12 16:48:22 INFO mapreduce.JobSubmitter: number of splits:1
15/05/12 16:48:22 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1431127776378_0049
15/05/12 16:48:25 INFO impl.YarnClientImpl: Submitted application application_1431127776378_0049
15/05/12 16:48:25 INFO mapreduce.Job: The url to track the job: http://quickstart.cloudera:8088/proxy/application_1431127776378_0049/
15/05/12 16:48:25 INFO mapreduce.Job: Running job: job_1431127776378_0049
15/05/12 16:48:58 INFO mapreduce.Job: Job job_1431127776378_0049 running in uber mode : false
15/05/12 16:48:58 INFO mapreduce.Job:  map 0% reduce 0%
15/05/12 16:49:45 INFO mapreduce.Job:  map 3% reduce 0%
15/05/12 16:50:53 INFO mapreduce.Job:  map 5% reduce 0%
15/05/12 16:50:57 INFO mapreduce.Job:  map 8% reduce 0%
15/05/12 16:51:01 INFO mapreduce.Job:  map 11% reduce 0%
15/05/12 16:51:09 INFO mapreduce.Job:  map 15% reduce 0%
15/05/12 16:51:13 INFO mapreduce.Job:  map 22% reduce 0%
15/05/12 16:51:18 INFO mapreduce.Job:  map 25% reduce 0%
15/05/12 16:51:21 INFO mapreduce.Job:  map 28% reduce 0%
15/05/12 16:51:29 INFO mapreduce.Job:  map 31% reduce 0%
15/05/12 16:51:32 INFO mapreduce.Job:  map 33% reduce 0%
15/05/12 16:51:45 INFO mapreduce.Job:  map 38% reduce 0%
15/05/12 16:51:57 INFO mapreduce.Job:  map 51% reduce 0%
15/05/12 16:52:07 INFO mapreduce.Job:  map 55% reduce 0%
15/05/12 16:52:10 INFO mapreduce.Job:  map 58% reduce 0%
15/05/12 16:52:14 INFO mapreduce.Job:  map 60% reduce 0%
15/05/12 16:52:18 INFO mapreduce.Job:  map 63% reduce 0%
15/05/12 16:52:26 INFO mapreduce.Job:  map 100% reduce 0%
15/05/12 16:52:59 INFO mapreduce.Job:  map 100% reduce 100%
15/05/12 16:53:01 INFO mapreduce.Job: Job job_1431127776378_0049 completed successfully
15/05/12 16:53:02 INFO mapreduce.Job: Counters: 49
 File System Counters
  FILE: Number of bytes read=1576
  FILE: Number of bytes written=226139
  FILE: Number of read operations=0
  FILE: Number of large read operations=0
  FILE: Number of write operations=0
  HDFS: Number of bytes read=35726474
  HDFS: Number of bytes written=27
  HDFS: Number of read operations=6
  HDFS: Number of large read operations=0
  HDFS: Number of write operations=2
 Job Counters
  Launched map tasks=1
  Launched reduce tasks=1
  Data-local map tasks=1
  Total time spent by all maps in occupied slots (ms)=205324
  Total time spent by all reduces in occupied slots (ms)=29585
  Total time spent by all map tasks (ms)=205324
  Total time spent by all reduce tasks (ms)=29585
  Total vcore-seconds taken by all map tasks=205324
  Total vcore-seconds taken by all reduce tasks=29585
  Total megabyte-seconds taken by all map tasks=210251776
  Total megabyte-seconds taken by all reduce tasks=30295040
 Map-Reduce Framework
  Map input records=157
  Map output records=157
```

```
Map output bytes=1256
Map output materialized bytes=1576
Input split bytes=132
Combine input records=0
Combine output records=0
Reduce input groups=1
Reduce shuffle bytes=1576
Reduce input records=157
Reduce output records=1
Spilled Records=314
Shuffled Maps =1
Failed Shuffles=0
Merged Map outputs=1
GC time elapsed (ms)=11772
CPU time spent (ms)=45440
Physical memory (bytes) snapshot=564613120
Virtual memory (bytes) snapshot=3050717184
Total committed heap usage (bytes)=506802176
Shuffle Errors
BAD_ID=0
CONNECTION=0
IO_ERROR=0
WRONG_LENGTH=0
WRONG_MAP=0
WRONG_REDUCE=0
File Input Format Counters
Bytes Read=35726342
File Output Format Counters
Bytes Written=27
```

Check results:

```
[cloudera@quickstart hipi]$ hadoop fs -ls project/output
Found 2 items
-rw-r--r--   1 cloudera cloudera          0 2015-05-12 16:52 project/output/_SUCCESS
-rw-r--r--   1 cloudera cloudera         27 2015-05-12 16:52 project/output/part-r-00000
[cloudera@quickstart hipi]$ hadoop fs -cat project/output/part-r-00000
157 Total face detected: 0
```

III. Conclusion

OpenCV provides very rich set of tools for image processing, when combined with HIPI's efficient and high-throughput parallel image processing power can be a great solution for processing very large image dataset very fast. These tools can help researcher and engineers alike to achieve high performance image processing.Pros: HIPI is a great tool for processing very large volume of images in hadoop cluster, when combined with OpenCV it can be very powerful.

References

[1]. https://www.google.co.in/?gfe_rd=cr&ei=ruemVpvHEJLI8Aedk5HwDg#q=image+%2Bmapreduce+ieee+paper
[2]. http://www.cs.princeton.edu/courses/archive/spr11/cos448/web/docs/week10_reading2.pdf
[3]. http://dinesh-malav.blogspot.in/2015/05/image-processing-using-opencv-on-hadoop.html
[4]. http://www.iosrjen.org/Papers/vol2_issue8%20(part-1)/K0287882.pdf
[5]. http://ce.sysu.edu.cn/hope/UploadFiles/Education/2011/10/201110221516245419.pdf
[6]. http://static.googleusercontent.com/media/research.google.com/en//archive/mapreduce-osdi04.pdf
[7]. http://www.oracle.com/technetwork/database/bi-datawarehousing/twp-indatabase-mapreduce-128831.pdf
[8]. https://www.mapr.com/blog/5-google-projects-changed-big-data-forever
[9]. http://shodhganga.inflibnet.ac.in/bitstream/10603/10203/16/16_publications.pdf