IJCSE International Journal of Computer Sciences and Engineering Open Access

Review Paper Vol-6, Special Issue-8, Oct 2018 E-ISSN: 2347-2693

HIPI: A REVIEW ON HADOOP MAP REDUCE FRAMEWORK USING IMAGE PROCESSING IN BIGDATA

[1*]P.S. Vijayalakshmi, [2]K. Gomathy, [3]C. Kumuthini,

[1,2,3]SNMV College of Arts and Science, Dr.N.G.P Arts and Science College,Tamilnadu,India

Available online at: www.ijcseonline.org

Abstract- Nowadays, Big data is growing vey faster in the world. *Big data is* the *large* volume of *data* that consists of both structured and unstructured on a day-to-day basis. But it's not the amount of *data*. Big data is the data which includes sensor data, biometric data, Geo-spatial, Healthcare, power grid, transport, search engine and in Social networks. Hadoop process large amounts of data, in parallel, clusters of commodity hardware in a reliable and fault-tolerant manner. In this paper we review the Image processing using Map reduce technique with the help of HIPI (the image processing Tool).

Keywords- Hadoop, Map reduce, Big data, Image Processing, HIPI

I. INTRODUCTION

Photo uploads are totally 300 million per day in the face book [1].For Image processing In Big data uses several tools. Here we are expressing Hadoop Map reduce using HIPI.

1.1 Hadoop

Hadoop is an open source software framework, java based and processing for large data sets in the distributed environment[3].Hadoop framework includes following four modules:

- **Hadoop Distributed File System (HDFS™):** A distributed file system that provides high-throughput
- **Map Reduce:** This is YARN-based system for parallel processing of large data sets.
- **Hadoop Common:**It is a collection Java libraries and utilities for supporting Hadoop modules.
- **Hadoop YARN:** This is a framework for job scheduling and cluster resource management.

1.2 MapReduce

Hadoop **Map Reduce** is used for process large amounts of data. The term Map Reduce performs two different types of tasks[2].

The Map Task: In this task,it takes input data and converts the data. Then the individual elements are broken down into tuples (key/value pairs).

- **The Reduce Task:** The Reduce task gets the output from a map task as input and combines those data tuples into a smaller set of tuples. If the map task is over, the reduce task is performed[4].

1.3Hadoop Distributed File System

HDFS uses a master/slave architecture .The Master consists of a single **Name Node** that manages the file system metadata and one or more slave **Data Nodes** that store the actual data[5].A file in an HDFS namespace is split into several blocks and those blocks are stored in a set of Data Nodes. The Name Node determines the mapping of blocks to the Data Nodes. The Data Nodes takes care of read and write operation with the file system. They also take care of block creation, deletion and replication based on instruction given by Name Node.

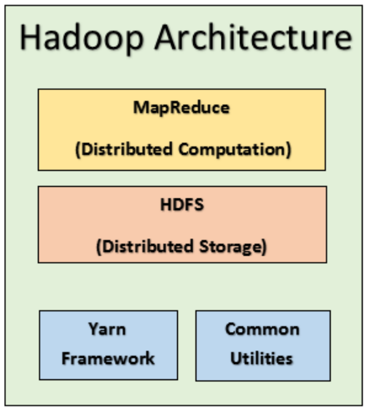

Fig: 1 Hadoop Architecture

II.HIPI (Hadoop Image Processing Interface)

HIPI (Hadoop Image Processing Interface) is an API library designed to provide efficient and high-throughput image processing in the Apache Hadoop Map Reduce parallel programming framework[6]. It also provides support for OpenCV.

This is designed to be used with the Apache Hadoop Map Reduce . HIPI is used for better performance image processing. It is functioning with Map Reduce style parallel programs executed on a cluster environment. Large collection of images on the Hadoop Distributed File System (HDFS) and it is available for distributed processing.

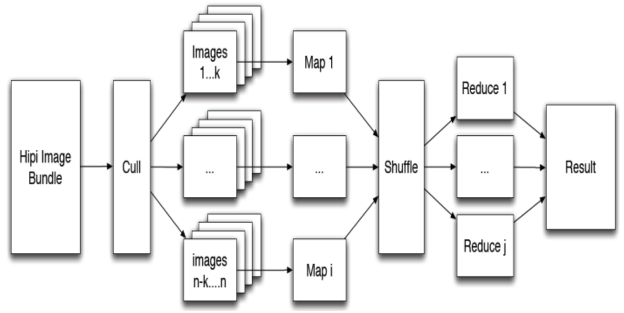

Fig-2 HIPI Architecture

The input object to a HIPI program is a HipiImageBundle (HIB). A HIB is a collection of images in a single file on the HDFS.
Culling is the first stage in HIPI program. It allows filtering the images in a HIB based on a variety of user-defined conditions in the spatial resolution to the image metadata[7]. This is done by the Culler class. Images are culled are not fully decoded, it is processing the saving processing time.
The HibInputFormat class, at last individual images are presented to the Mapper objects derived from the HipiImage abstract base class with an associated HipiImageHeader object.
The following providesa guide to setting up HIPI on your system and writing your first MapReduce/HIPI program.

2.1. Setup Java
HIPI is written in Java and has been tested with Java 7 and 8. Check your version of Java with the following command:

```
$> java -version
Java version "1.8.0_45"
Java(TM) SE Runtime Environment (build 1.8.0_45-b14)
Java HotSpot(TM) 64-Bit Server VM (build 25.45-b02, mixed mode)
```

2.2. Setup Hadoop
HIPI works with a standard installation of the Apache Hadoop Distributed File System (HDFS) and Map Reduce. HIPI has been tested with Hadoop version 2.7.1.Verify that the main Hadoop script is reachable from your path: $> which hadoop /usr/local/bin/hadoop

2.3. Setup Gradle
The HIPI distribution uses the Gradle build automation system to manage compilation and package assembly. HIPI has been tested with Gradle version 2.5. Install Gradle on your system and verify that it is reachable as well: $> which gradle /usr/local/bin/gradle

2.4. Install HIPI
In two ways l HIPI will be installed on the system:
1. HIPI distribution from GitHub and build from source.
2. Download a precompiled JAR .

2.5 Clone the HIPI GitHub Repository
The better wayis to get the latest version of HIPI is by cloning the official GitHub repository and building it along with all of the toolsf. This only takes a few minutes and verifies that your system is properly setup and ready to begin developing your own HIPI applications:

```
$> git clone git@github.com:uvagfx/hipi.git
```

2.6 Build the HIPI Library and Example Programs
Run gradle to build the HIPI library along with all of the tools and example programs:

```
$> cd hipi
$> gradle
:core:compileJava
:core:processResources
:core:classes
:core:jar
:tools:downloader:compileJava
:tools:downloader:processResources
:tools:downloader:classes
:tools:downloader:jar
:tools:dumpHib:compileJava
:tools:dumpHib:processResources
:tools:dumpHib:classes
:tools:dumpHib:jar
...
:install

Finished building the HIPI library along with all tools and examples.

BUILD SUCCESSFUL
Total time: 2.058 secs
```

After the build finishes, to inspect the settings.gradle file in the root directory and the build.gradle files in each directory.

```
$> gradle clean tools:hibImport:jar
:core:clean
...
:core:compileJava
:core:processResources UP-TO-DATE
:core:classes
:core:jar
:tools:hibImport:compileJava
:tools:hibImport:processResources UP-TO-DATE
:tools:hibImport:classes
:tools:hibImport:jar

BUILD SUCCESSFUL
Total time: 1.197 secs
```

HIPI is now installed on the system.

III. First HIPI Program

This is the process of creating a very simple HIPI program that computes the average pixel color over a set of images. First, we need a set of images to work with. Recall that 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 from a collection of images on our local file system located in the directory ~/Sample Images by executing the following command from the HIPI root directory[8]:

```
$> tools/hibImport.sh ~/SampleImages sampleimages.hib
Input image directory: /Users/jason/SampleImages
Output HIB: sampleimages.hib
Overwrite HIB if it exists: false
HIPI: Using default blockSize of [134217728].
HIPI: Using default replication factor of [1].
 ** added: 1.jpg
 ** added: 2.jpg
 ** added: 3.jpg
Created: sampleimages.hib and sample images.hib.dat
```

Note that **import Hib** actually creates two files in the current working directory of the HDFS: sampleimages.hib and sample images.hib.dat. Verify that this is the case with the command: hadoop fs -ls.
We use the handy hib Info tool that comes with HIPI to inspect the contents of this newly created HIB file:

```
$> tools/hibInfo.sh sampleimages.hib --show-meta
Input HIB: sampleimages.hib
Display meta data: true
Display EXIF data: false
IMAGE INDEX: 0
  640 x 480
format: 1
meta: {source=/Users/hipiuser/SampleImages/1.jpg}
IMAGE INDEX: 1
  3210 x 2500
format: 1
meta: {source=/Users/hipiuser/SampleImages/2.jpg}
IMAGE INDEX: 2
  3810 x 2540
format: 1
meta: {source=/Users/hipiuser/SampleImages/3.jpg}
Found [3] images.
```

Next the Gradle, create a source directory hierarchy for the program by executing the following command in the root directory:

```
$> mkdir -p examples/hello
World/src/main/java/org/hipi/examples
```

Next, add a Gradle build task for our new program by creating the file examples/helloWorld/build. gradle with the following contents:

```
jar {
manifest {
attributes("Main-Class": "org.hipi.examples.HelloWorld")
  }
}
```

We also need to update the settings .gradle file in the root directory to tell Gradle about this new build target:

```
include ':core', ':tools:hibImport', ... ':examples:covar',
':examples:helloWorld'
```

Next, create a new Java source file at examples/helloWorld/src/main/java/org/hipi/examples/Hello World.java that contains the following code:

```
package org.hipi.examples;

import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;
public class HelloWorld extends Configured implements Tool
{
public int run(String[] args) throws Exception {
System.out.println("Hello HIPI!");
return 0;
  }
public static void main(String[] args) throws Exception {
ToolRunner.run(new HelloWorld(), args);
System.exit(0);
  }}
```

Every Java program isstart thepublic static void main() method.In MapReduce applications, the main method in our program uses the ToolRunner Hadoop class to call the run () method in this driver class.
Build this very simple program by running the command gradle jar in the examples/helloWorld directory:

```
$> cd examples/helloWorld
$> gradle jar
:core:compileJava UP-TO-DATE
:core:processResources UP-TO-DATE
:core:classes UP-TO-DATE
:core:jar UP-TO-DATE
:examples:helloWorld:compileJava UP-TO-DATE
:examples:helloWorld:processResources UP-TO-DATE
:examples:helloWorld:classes UP-TO-DATE
:examples:helloWorld:jar

BUILD SUCCESSFUL
Total time: 1.191 secs
```

If the build is successful, it will produce the JAR file examples/helloWorld/build/libs/helloWorld.jar directory. Run this program using the following command from within the examples/helloWorld directory:

```
  $> hadoop jar build/libs/helloWorld.jar
  Hello HIPI!
```

Use run() method in HelloWorld.java to initialize and execute a MapReduce job and create stubs for our Mapper and Reducer classes:

```
package org.hipi.examples;

import org.hipi.image.FloatImage;
import org.hipi.image.HipiImageHeader;
import org.hipi.imagebundle.mapreduce.HibInputFormat;

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.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import java.io.IOException;

public class HelloWorld extends Configured implements Tool {

  public static class HelloWorldMapper extends Mapper<HipiImageHeader,
FloatImage, IntWritable, FloatImage> {
    public void map(HipiImageHeader key, FloatImage value, Context context)
      throws IOException, InterruptedException {
    }
  }

  public static class HelloWorldReducer extends Reducer<IntWritable,
FloatImage, IntWritable, Text> {
    public void reduce(IntWritable key, Iterable<FloatImage> values, Context
context)
      throws IOException, InterruptedException {
    }
  }

  public int run(String[] args) throws Exception {
    // Check input arguments
    if (args.length != 2) {
      System.out.println("Usage: helloWorld <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(HibInputFormat.class);
    // Set the driver, mapper, and reducer classes which express the
computation
    job.setJarByClass(HelloWorld.class);
    job.setMapperClass(HelloWorldMapper.class);
    job.setReducerClass(HelloWorldReducer.class);
    // Set the types for the key/value pairs passed to/from map and reduce
layers
job.setMapOutputKeyClass(IntWritable.class);
    job.setMapOutputValueClass(FloatImage.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]));

    // 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 HelloWorld(), args);
    System.exit(0);
  }


}
```

The run() method validate the arguments passed to the program. Create the Hadoop Job object and call setter methods on this object to specify the classes that implement the map and reduce tasks.. The remaining lines of code setup the path to the input file and the output directory and launch the program.

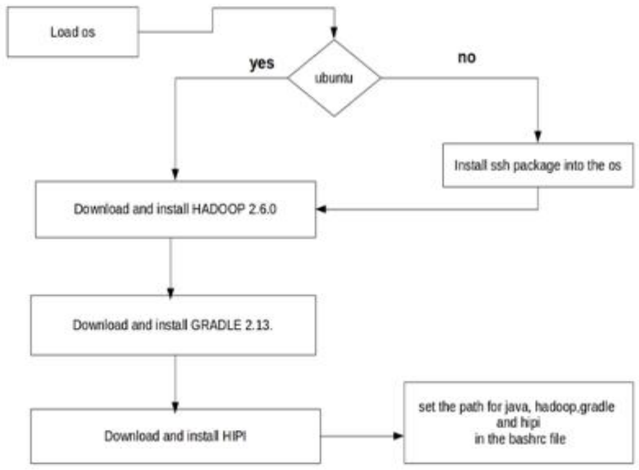
Load os
ubuntu
yes
no
Install ssh package into the os
Download and install HADOOP 2.6.0
Download and install GRADLE 2.13.
Download and install HIPI
set the path for java, hadoop,gradle and hipi in the bashrc file

Fig-3 HIPI Process

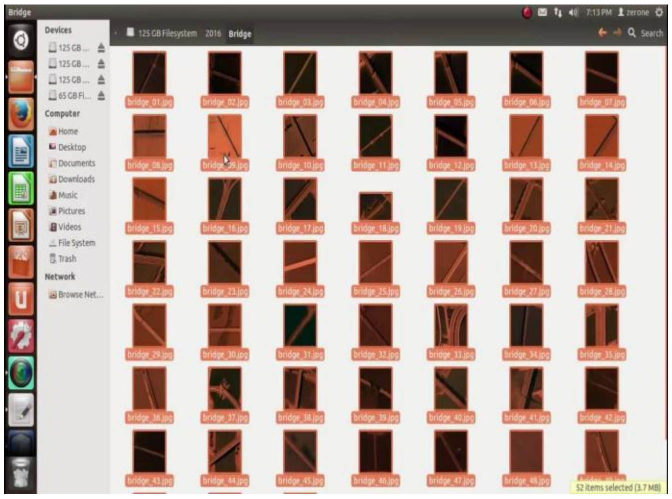

Fig-4 Pictures loaded in HIPI

IV.COMPUTING THE AVERAGE PIXEL COLOR

Now let's add some actual HIPI image processing code to the program. For this example, we will be computing the average RGB value of the pixels in the images in our input HIB. Our mapper will compute the average pixel color over a single image and the reducer will add these averages together and divide by their count to compute the total average pixel color. Because the map tasks are executed in parallel, if our Hadoop cluster has more than one compute node we will perform this entire operation faster than if we were using a single machine. This is the key idea behind parallel computing in Map Reduce. Here is what our map() method looks like

:
```
public static class HelloWorldMapper extends Mapper<HipiImageHeader,
FloatImage, IntWritable, FloatImage> {

    public void map(HipiImageHeader 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.getNumBands() == 3) {

        // Get dimensions of image
        int w = value.getWidth();
        int h = value.getHeight();

        // Get pointer to image data
        float[] valData = value.getData();

        // Initialize 3 element array to hold RGB pixel average
        float[] avgData = {0,0,0};

        // 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++) {
            avgData[0] += valData[(j*w+i)*3+0]; // R
            avgData[1] += valData[(j*w+i)*3+1]; // G
            avgData[2] += valData[(j*w+i)*3+2]; // B
          }
        }

        // Create a FloatImage to store the average value
        FloatImage avg = new FloatImage(1, 1, 3, avgData);

        // Divide by number of pixels in image
        avg.scale(1.0f/(float)(w*h));

        // Emit record to reducer
        context.write(new IntWritable(1), avg);

      } // If (value != null...

    } // map()

  } // HelloWorldMapper
```

The first two arguments of the map() method are a key/value pair (often called a "record" in Map Reduce terminology) that are constructed by the HibInputFormatHibRecordReader classes[9]. In this case, these two arguments are a HipiImageHeader (the "key") and a FloatImage (the "value"), respectively. In HIPI, the first argument of the map() method must always be a HipiImageHeader, but the second argument can be any type that extends the abstract base class HipiImage. Note that this map() method produces a record for each image in the HIB which is sent to the reduce processing stage using the context.write() method. These records consist of an IntWritable (that is always equal to 1) and another HIPI Float Image object that contains the image's computed average pixel value. These records are collected by the Map Reduce framework and become inputs to the reduce() method as an Iterable list of Float Image objects where they are added together and normalized to obtain the final result:

```
  public static class HelloWorldReducer extends Reducer<IntWritable,
FloatImage, IntWritable, Text> {

    public void reduce(IntWritable key, Iterable<FloatImage> values, Context
context)
        throws IOException, InterruptedException {

      // Create FloatImage object to hold final result
      FloatImage avg = new FloatImage(1, 1, 3);

      // Initialize a counter and iterate over IntWritable/FloatImage records
from mapper
      int total = 0;
      for (FloatImage val : values) {
        avg.add(val);
        total++;
      }

      if (total > 0) {
        // Normalize sum to obtain average
        avg.scale(1.0f / total);
        // Assemble final output as string
        float[] avgData = avg.getData();
        String result = String.format("Average pixel value: %f %f %f",
avgData[0], avgData[1], avgData[2]);
        // Emit output of job which will be written to HDFS
        context.write(key, new Text(result));
      }

    } // reduce()

  } // HelloWorldReducer
```

Next, build `helloWorld.jar` and run it using the HIB we created at the beginning:

```
$> gradle jar
:core:compileJava UP-TO-DATE
:core:processResources UP-TO-DATE
:core:classes UP-TO-DATE
:core:jar UP-TO-DATE
:examples:helloWorld:compileJava
:examples:helloWorld:processResources UP-TO-DATE
:examples:helloWorld:classes
:examples:helloWorld:jar

BUILD SUCCESSFUL

Total time: 0.855 secs

$> hadoop jar build/libs/helloWorld.jar sampleimages.hib sampleimages_average
...
```

If everything goes as planned the directory `sampleimages_average` will contain two files:

```
$> hadoop fs -ls sampleimages_average
Found 2 items
```

Whenever a Map Reduce program successfully finishes, it creates the file _SUCCESS in the output directory along with a part-r-XXXXX file for each reduce task. The average pixel value can be retrieved using the cat command:

```
$> hadoop fs -cat sampleimages_average/part-r-00000
1 Average pixel value: 0.321921 0.224995 0.150284
```

V. CONCLUSION

This paper has described HIPI for image processing and vision applications on a Map Reduce framework. It is created with the intent to operate on large sets of images. We provide a format for storing images for efficient access within the Map Reduce pipeline, and simple methods for creating such file. By providing a culling stage before the mapping stage, we give the user a simple way to filter image sets and control the types of images being used in their Map Reduce tasks.

REFERENCES

[1] https://zephoria.com/top-15-valuable-facebook-statistics
[2]Hadoop map reduce framework.http://hadoop.apache.org/mapreduce/.
[3]Customizing input file formats for image processing in hadoop. Arizona State University. Online at:
http://hpc. asu. edu/node/97.
[4]DEAN, J.,AND GHEMAWAT,S.2008. Mapreduce: Simplified data processing on large clusters. Communications of the ACM 51, 1,107–113.
[5] PDF: HIPI: A Hadoop Image Processing Interface for Image-based Map Reduce Tasks. Available from: https://www.researchgate.net/publication/266464321_HIPI_A_Hadoop_Image_Processing_Interface_for_Image-based_MapReduce_Tasks [accessed Jul 18 2018].
[6] http://hipi.cs.virginia.edu/
[7] https://github.com/uvagfx/hipi
[8] http://hipi.cs.virginia.edu/gettingstarted.html
[9] http://hipi.cs.virginia.edu/examples.html