Image Conversion Using Hadoop In Cloud Computing Environment

Aman Hasan Shaik[1], **Himaja Kotturu**[2]
[1] Department of Computer Science, Arkansas State University, Jonesboro, AR, US
[2] Department of Computer Science, Arkansas State University, Jonesboro, AR, US

Abstract - *The number of images being uploaded nowadays to the internet is increasing, with Facebook users uploading over 2.5 billion new photos nearly every month however, applications that make use of this data are not much. To compute this much of huge data for computational resources and storage options we should use Hadoop distributed systems. The conventional approach to transcoding data requires fixed and expensive hardware because of the high-capacity and high definition features of data and transcoding imposes a considerable burden on the computing power as the amount of data increases. So, the proposed platform Hadoop HDFS and the MapReduce framework for distributed parallel processing of image and converting the image database into target format like pdf and jpeg using Cloudera and Virtual box as a platform.*

Keywords: Hadoop, Map Reduce, Cloudera, Virtual Box,

1 Introduction

Cloud computing has accomplished great interest from specialists and the IT business for giving a flexible dynamic IT foundation, QoS ensured computing situations, and configurable programming administrations. Because of these points of interest, many specialist organizations who release Social Network Services (SNS) enables clients to scatter sight and sound articles. SNS and media content suppliers are continually progressing in the direction of giving multimedia rich encounters to end clients. Keeping in mind the end goal to build up a SNS in light of a lot of web-based social networking, versatile mass stockpiling for web-based social networking information made day by day by clients is required. Despite the fact that the capacity to share sight and sound articles makes the Internet more alluring to shoppers, customers and basic systems are not generally ready to stay aware of this developing interest. Interactive media preparing is portrayed by a lot of information, requiring a lot of handling, stockpiling, and correspondence assets, in this manner forcing an impressive weight on the computing infrastructure. The conventional way to deal with transcoding multimedia information requires particular and costly equipment on account of the high-capacity and high definition components of multimedia data. Subsequently, broadly useful gadgets and strategies are not effective, and they have constraints. Here, we apply a cloud computing environment to our Hadoop-based Image Conversion framework. Changes in quality and speed are accomplished by adopting Hadoop Distributed File System (HDFS) for keeping large amount of image information made by various clients, Map Reduce for distributed and parallel processing of image information. This stage is made out of a cloud distributed and parallel information processing platform for storing, distributing and processing social network information.

With the spread of social networking in recent years, a lot of picture information has been gathering. When processing this large amount of information has been constrained to single PCs, computational power and capacity rapidly move toward becoming bottlenecks. On the other hand, handling tasks can ordinarily be performed on a distributed system by dividing the assignment into a few subtasks. The capacity to parallelize tasks takes into account versatile, effective execution of resource applications. The Hadoop Map Reduce system gives a stage to such tasks. While considering operations, for example, face detection, picture grouping and different sorts of processing on images, there are cutoff points on what should be possible to enhance execution of single PCs to make them ready to process data at the size of web-based social networking. Therefore, the advantages of parallel distributed processing of a vast picture dataset by utilizing the computational assets of a cloud computing environment ought to be considered. Furthermore, if computational resources can be secured effectively and generally economically, then cloud computing is appropriate for handling of large picture information with ease and expanded execution. Hadoop, as a framework for preparing substantial quantities of pictures by parallel and distributed computing, seems promising.

2 Cloud Security

Cloud computing security or, more essentially, cloud security refers to a wide arrangement of strategies, advancements, and controls sent to ensure information, applications, and the related framework of cloud computing. It is a sub-area of computer Security, network security, and, all the more extensively, data security.

Cloudcomputing and storage gives users the abilities to store and process their information in third party data centers. Organizations utilize the cloud in an assortment of various administration models (with acronyms, for example, SaaS, PaaS, and IaaS) and organization models (private, public, hybrid, and community). Security concerns related with cloud computing fall into two general classes, security issues confronted by cloud suppliers (associations giving software, platform, or infrastructure as-an administration via cloud) and security issues confronted by their clients (organizations or associations who have applications or store information on the cloud). The responsibility is shared, be that as it may. The supplier must guarantee that their infrastructure is secure and that their customers' information and applications are ensured, while the client must take measures to sustain their application and utilize strong passwords and authentication measures.

Distributed computing security is a fast-growing service that provides the functionalities as IT security. This incorporates shielding critical information from theft, data leakage and deletion.

One of the advantages of cloud services is that you can work at scale and still stay secure. It is like how you right now manage security, however now we have better approaches for conveying security solutions that address new areas of concern. Cloud security does not change the approach on the best way to manage security from counteracting to investigator and restorative activities. But, it does however give you the capacity to play out these activities in a more lithe way.

Your information is secured inside data centers and where a few nations oblige information to be put away in their nation, picking a supplier that has multiple data centers across the World to reach this.

Data centers frequently incorporates certain compliance necessities particularly when storing credit card numbers or health data. Many cloud suppliers offer free third party audit reports to bear witness to that their inner procedure exist and are successful in dealing with the security inside their offices where you store your information.

3 Tools

3.1 Hadoop

Hadoop is an open source, Java-based programming system that supports the processing and capacity of to a great degree extensive informational collections in distributed computing environment. It is a part of the Apache project supported by the Apache Software Foundation.

Hadoop makes it possible to run applications on frameworks with thousands of hardware nodes, and to deal with a huge number of terabytes of information. Its distributed file encourages quick information exchange rates among nodes and enables the framework to keep working in case of node failure. This approach brings down the risk of cataclysmic system failure and unexpected information misfortune, regardless of the possibility that countless wind up noticeably out of commission. Subsequently, Hadoop immediately risen as an establishment for big data processing tasks, for example, business, sales planning and deals arranging, and preparing huge volumes of sensor information, including from web of things sensors.

Hadoop was created by computer researchers Doug Cutting and Mike Cafarella in 2006 to support distribution for the Nutch search tool. It was inspired by Google's MapReduce, a software framework in which an application is separated into various little parts. Any of these parts, which are likewise called frameworks or blocks, can be keep running on any node in a cluster. Following quite a while of improvement inside the open source group, Hadoop 1.0 turned out to be publically accessible in November 2012 as a part of the Apache project supported by the Apache Software Foundation.

Since its initial release, Hadoop has been continuously created and updated. The second emphasis of (Hadoop 2) enhanced resource management and scheduling. It highlights a high-accessibility file system option and support for Microsoft Windows and different segments to extend the system's adaptability for information handling and analytics.

3.2 Map Reduce

MapReduce is a framework utilizing which we can compose applications to process large amounts information, in parallel, on large clusters of product equipment in a reliable manner.

Map Reduce is a processing technique and a program model for distributed computing in view of java. The Map Reduce algorithm contains two essential assignments, namely Map and Reduce. Map takes a set of data and converts it into another set of data, where individual elements are broken down into tuples. Secondly, reduce task, which takes the output from a map as an input and combines those data tuples into a smaller set of tuples. As the succession of the name Map Reduce infers, first the map job is done and then the reduce task.

The advantage of Map Reduce is that it is simple for data processing over various computing nodes. Under the Map Reduce model, the information preparing primitives are called mappers and reducers. Disintegrating an information preparing application into mappers and reducers is infrequently nontrivial. Be that as it may, once we compose an application in the Map Reduce shape, scaling the application to keep running more than hundreds, thousands, or even countless machines in a cluster is simply a setup change. This basic versatility is the thing that has attracted in numerous software engineers to utilize the Map Reduce model.

3.2.1 Algorithm

- Generally MapReduce paradigm depends on sending the computer to where the data resides!

- MapReduce program executes in three phases, namely map stage, shuffle stage, and reduce stage.

Map stage: The map or mapper's job is to handle the input data. For the most part, the input data is as record or index and is storedin the Hadoop document framework (HDFS). The input file is passed to the mapper funtion line by line. The mapper forms the information and makes a several small chunks of data.

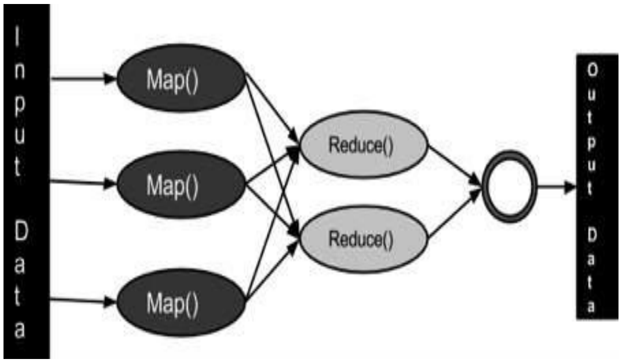
Reduce stage: This stage incorporates the Shuffle stage and the Reduce stage. The Reducer’s job is to process the data that comes from the mapper. After processing, it produces a new set of outputs, which will be stored in the HDFS.

- During a MapReduce work, Hadoop sends the Map and Reduce tasks to the proper servers in the cluster

- The structure deals with every one of the subtle elements of information passing, for example, issuing assignments, confirming task completion, and duplicating information around the cluster between the nodes.

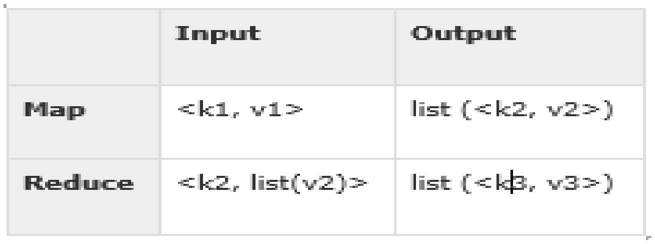
- Most of the computing happens on nodes with information on nearby disks that decreases the network traffic.

- After taking the given tasks, the cluster gathers and decreases the information to frame a proper outcome, and sends it back to the Hadoop server.



3.2.2 Inputs and Outputs (JAVA Perspective)

The MapReduce framework works on <key, value> sets, that is, the framework sees the input to the job as an arrangement of <key, value> pairs and delivers a set of <key, value> pairs as the output of the job, possibly of various sorts.



	Input	Output
Map	<k1, v1>	list (<k2, v2>)
Reduce	<k2, list(v2)>	list (<k3, v3>)

The key and the value classes ought to be in serialized way by the framework and hence, need to implement the Writable interface. Moreover, the key classes need to actualize the Writable-Comparable interface to encourage sorting by the frameowrk. Info and Output sorts of a MapReduce job: (Input) <k1, v1> - > outline > <K2, v2>-> decrease - > <k3, v3> (Output).

3.2.3 Terminology

- **PayLoad** - Applications implement the Map and the Reduce functions, and form the core of the job.

- **Mapper** - Mapper maps the input key/value pairs to a set of intermediate key/value pair.

- **NamedNode** - Node that manages the Hadoop Distributed File System (HDFS).

- **DataNode** - Node where data is presented in advance before any processing takes place.

- **MasterNode** - Node where JobTracker runs and which accepts job requests from clients.

- **SlaveNode** - Node where Map and Reduce program runs.

- **JobTracker** - Schedules jobs and tracks the assign jobs to Task tracker.

- **Task Tracker** - Tracks the task and reports status to JobTracker.

- **Job** - A program is an execution of a Mapper and Reducer across a dataset.

- **Task** - An execution of a Mapper or a Reducer on a slice of data.

- **Task Attempt** - A particular instance of an attempt to execute a task on a SlaveNode.

3.2.4 Compilation and Execution of Process Units Program

Let us assume we are in the home directory of a Hadoop user (e.g. /home/hadoop).

Follow the steps given below to compile and execute the above program.

- **Step 1 :** The following command is to create a directory to store the compiled java classes.

```
$ mkdir units
```

- **Step 2 :** Download Hadoop-core-1.2.1.jar, which is used to compile and execute the MapReduce program. Visit the following link http://mvnrepository.com/artifact/org.apache.hadoop/hadoop-core/1.2.1 to download the jar. Let us assume the downloaded folder is /home/hadoop/.

- **Step 3 :** The following commands are used for compiling the ProcessUnits.java program and creating a jar for the program.

```
$ javac -classpath hadoop-core-1.2.1.jar -d units
ProcessUnits.java

$ jar -cvf units.jar -C units/ .
```

- **Step 4 :** The following command is used to create an input directory in HDFS.

```
$HADOOP_HOME/bin/hadoop fs -mkdir input_dir
```

- **Step 5 :** The following command is used to copy the input file named sample.txtin the input directory of HDFS.

```
$HADOOP_HOME/bin/hadoop fs -put
/home/hadoop/sample.txt input_dir
```

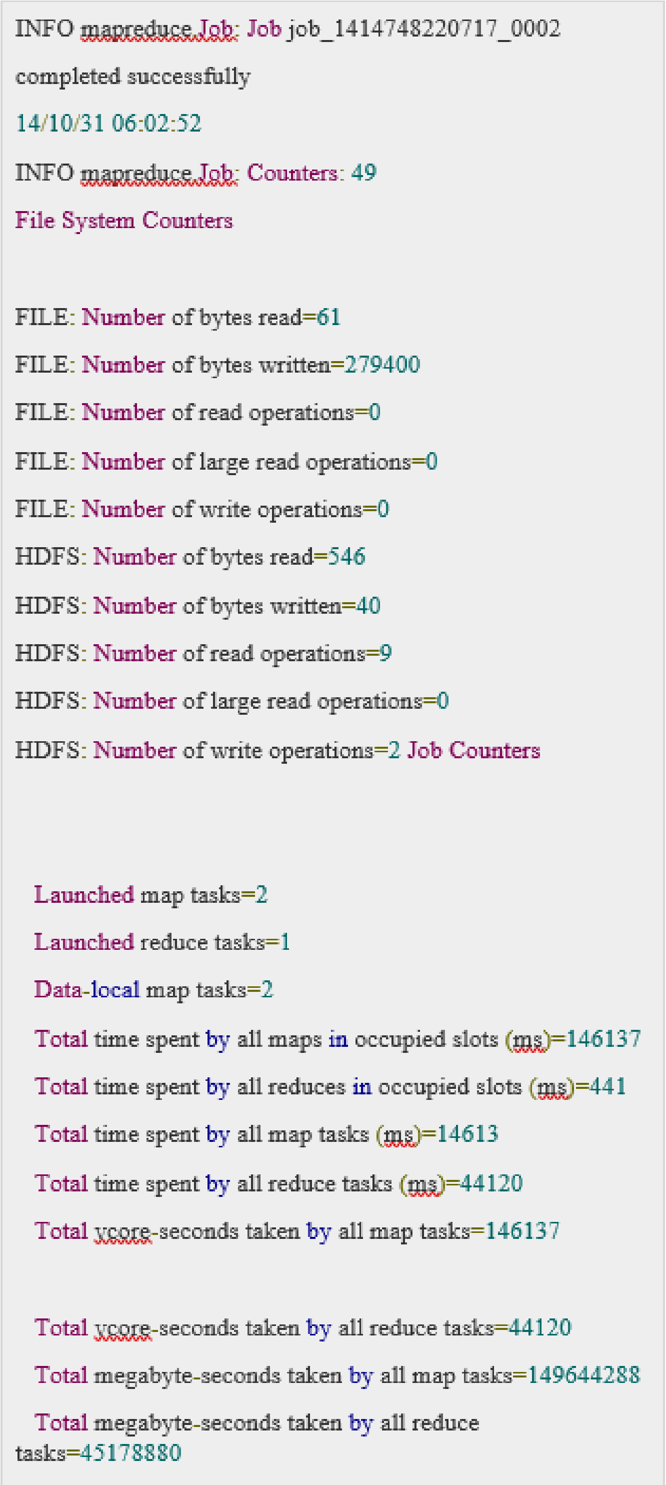
- **Step 6 :** The following command is used to verify the files in the input directory**.**

```
$HADOOP_HOME/bin/hadoop fs -ls input_dir/
```

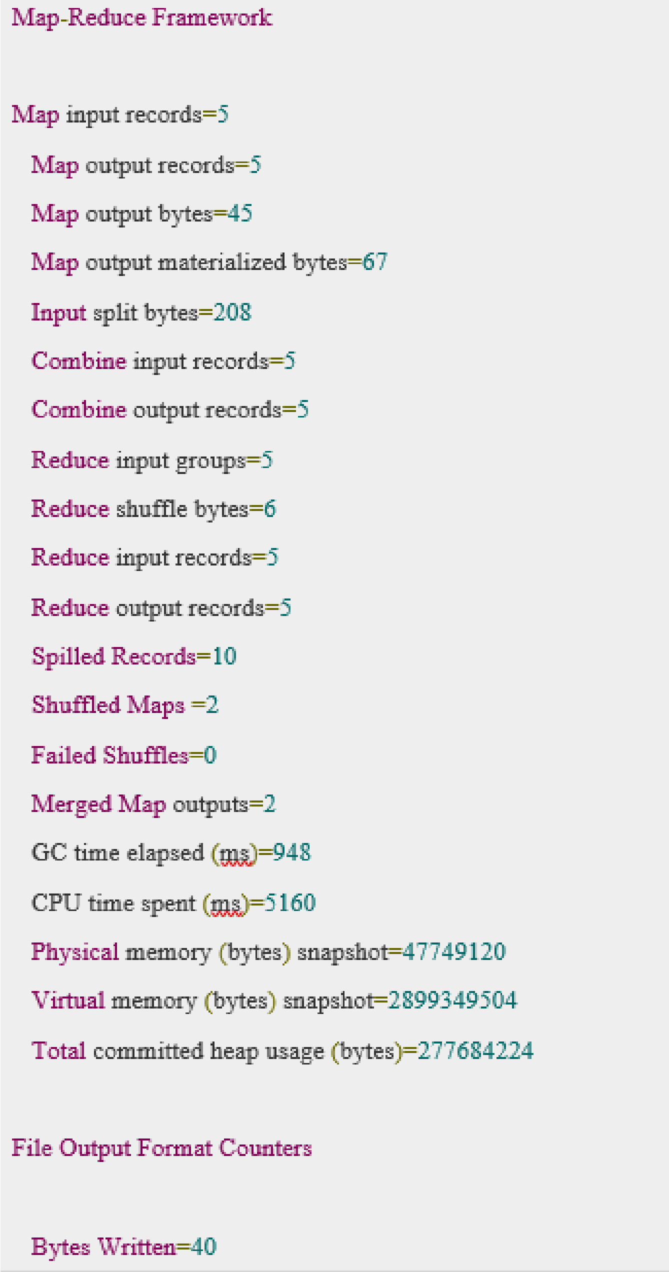
- **Step 7 :** The following command is used to run the Eleunit_max application by taking the input files from the input directory.

```
$HADOOP_HOME/bin/hadoop jar units.jar
hadoop.ProcessUnits input_dir output_dir
```

Wait for a while until the file is executed. After execution, as shown below, the output will contain the number of input splits, the number of Map tasks, the number of reducer tasks, etc.



```
INFO mapreduce.Job: Job job_1414748220717_0002

completed successfully

14/10/31 06:02:52

INFO mapreduce.Job: Counters: 49

File System Counters


FILE: Number of bytes read=61

FILE: Number of bytes written=279400

FILE: Number of read operations=0

FILE: Number of large read operations=0

FILE: Number of write operations=0

HDFS: Number of bytes read=546

HDFS: Number of bytes written=40

HDFS: Number of read operations=9

HDFS: Number of large read operations=0

HDFS: Number of write operations=2 Job Counters



  Launched map tasks=2

  Launched reduce tasks=1

  Data-local map tasks=2

  Total time spent by all maps in occupied slots (ms)=146137

  Total time spent by all reduces in occupied slots (ms)=441

  Total time spent by all map tasks (ms)=14613

  Total time spent by all reduce tasks (ms)=44120

  Total vcore-seconds taken by all map tasks=146137


  Total vcore-seconds taken by all reduce tasks=44120

  Total megabyte-seconds taken by all map tasks=149644288

  Total megabyte-seconds taken by all reduce
tasks=45178880
```



```
Map-Reduce Framework

Map input records=5
   Map output records=5
   Map output bytes=45
   Map output materialized bytes=67
   Input split bytes=208
   Combine input records=5
   Combine output records=5
   Reduce input groups=5
   Reduce shuffle bytes=6
   Reduce input records=5
   Reduce output records=5
   Spilled Records=10
   Shuffled Maps =2
   Failed Shuffles=0
   Merged Map outputs=2
   GC time elapsed (ms)=948
   CPU time spent (ms)=5160
   Physical memory (bytes) snapshot=47749120
   Virtual memory (bytes) snapshot=2899349504
   Total committed heap usage (bytes)=277684224

File Output Format Counters

   Bytes Written=40
```

- **Step 8 :** The following command is used to verify the resultant files in the output folder.

```
$HADOOP_HOME/bin/hadoop fs -ls output_dir/
```

- **Step 9 :** The following command is used to see the output in Part-00000 file. This file is generated by HDFS.

```
$HADOOP_HOME/bin/hadoop fs -cat output_dir/part-00000
```

Below is the output generated by the MapReduce program.

```
1981    34
1984    40
1985    45
```

- **Step 10 :** The following command is used to copy the output folder from HDFS to the local file system for analyzing.

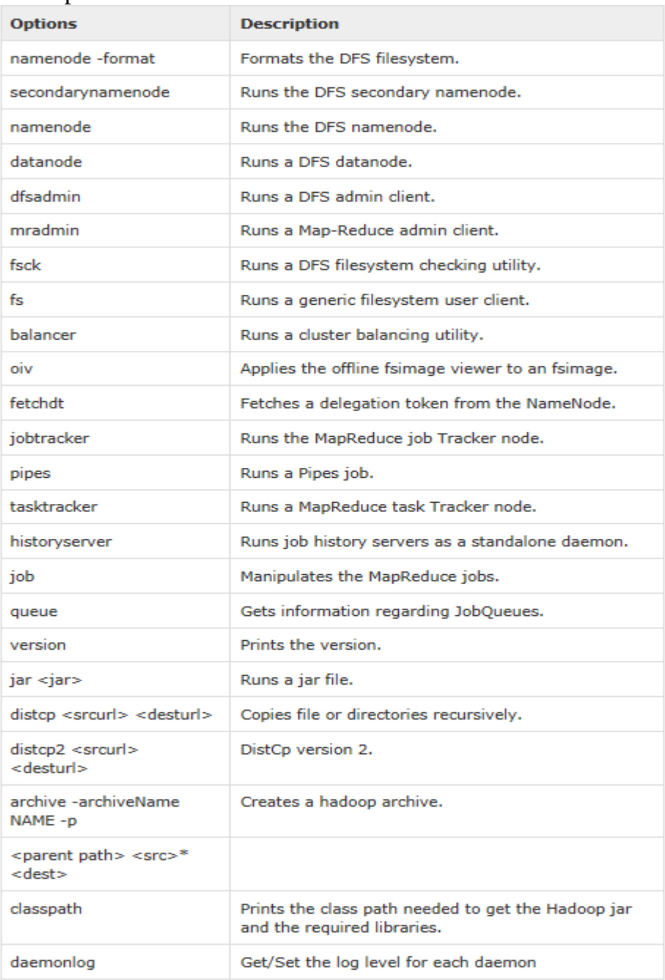
```
$HADOOP_HOME/bin/hadoop fs -cat output_dir/part-00000/bin/hadoop dfs get output_dir /home/hadoop
```

3.2.5 Important Commands

All Hadoop commands are invoked by the **$HADOOP_HOME/bin/hadoop** command. Running the Hadoop script without any arguments prints the description for all commands.

Usage : hadoop [--config confdir] COMMAND

The following table lists the options available and their description.



Options	Description
namenode -format	Formats the DFS filesystem.
secondarynamenode	Runs the DFS secondary namenode.
namenode	Runs the DFS namenode.
datanode	Runs a DFS datanode.
dfsadmin	Runs a DFS admin client.
mradmin	Runs a Map-Reduce admin client.
fsck	Runs a DFS filesystem checking utility.
fs	Runs a generic filesystem user client.
balancer	Runs a cluster balancing utility.
oiv	Applies the offline fsimage viewer to an fsimage.
fetchdt	Fetches a delegation token from the NameNode.
jobtracker	Runs the MapReduce job Tracker node.
pipes	Runs a Pipes job.
tasktracker	Runs a MapReduce task Tracker node.
historyserver	Runs job history servers as a standalone daemon.
job	Manipulates the MapReduce jobs.
queue	Gets information regarding JobQueues.
version	Prints the version.
jar <jar>	Runs a jar file.
distcp <srcurl> <desturl>	Copies file or directories recursively.
distcp2 <srcurl> <desturl>	DistCp version 2.
archive -archiveName NAME -p	Creates a hadoop archive.
<parent path> <src>* <dest>	
classpath	Prints the class path needed to get the Hadoop jar and the required libraries.
daemonlog	Get/Set the log level for each daemon

3.3 Cloudera

Cloudera Inc. is a United States-based software organization that gives Apache Hadoop-based programming, support and administrations, and preparing to business clients.

Cloudera's open-source Apache Hadoop distribution, CDH (Cloudera Distribution Including Apache Hadoop), targets venture class deployments, of that innovation. Cloudera says that over half of its engineering output is given upstream to the different Apache-authorized open source projects (Apache Hive, Apache Avro, Apache HBase, et cetera) that consolidate to frame the Hadoop platform. Cloudera is additionally a sponsor of the Apache Software Foundation.

3.4 Virtual Box

Virtual Box is a cross-stage virtualization application. What does that mean? For a certain something, it introduces on your current Intel or AMD-based PCs, regardless of whether they are running Windows, Mac, Linux or Solaris operating systems. Furthermore, it broadens the abilities of your current PC with the goal that it can run numerous operating systems (inside various virtual machines) in the sometime. Thus, for instance, you can run Windows and Linux on your Mac, run Windows Server 2008 on your Linux server, run Linux on your Windows PC, et cetera, all close by your current applications. You can introduce and keep running the same number of virtual machines as you like - the main pragmatic limits are disk space and memory.

Virtual Box is deceptively basic yet likewise powerful. It can run wherever from little installed frameworks or desktop class machines as far as possible up to datacenter organizations and even Cloud environments.

3.5 Installation on Virtual Machine

Cloudera QuickStart virtual machines (VMs) incorporate all that you have to attempt CDH, Cloudera Manager, Cloudera Impala, and Cloudera Search.

The VM utilizes a bundle based install. This enables you to work with or without Cloudera Manager. Parcels don't work with the VM unless you initially relocate your CDH establishment to utilize parcels.

3.5.1 Prerequisites

• These 64-bit VMs require a 64-bit have OS and a virtualization system that can support a 64-bit OS.

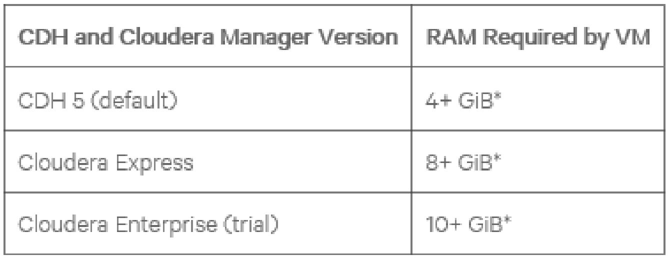
• To utilize a VMware VM, you should utilize a player perfect with WorkStation 8.x or higher:

o Player 4.x or higher

o Fusion 4.x or higher

More seasoned versions of WorkStation can be utilized to make another VM utilizing the same virtual plate (VMDK document), yet a few components in VMware Tools are not accessible.

• The measure of RAM required fluctuates by the run-time choice you pick:



CDH and Cloudera Manager Version	RAM Required by VM
CDH 5 (default)	4+ GiB*
Cloudera Express	8+ GiB*
Cloudera Enterprise (trial)	10+ GiB*

*Minimum memory recommended. In the event that you are running workloads bigger than the illustrations gave, consider allocating extra memory.

3.5.2 Downloading a Cloudera QuickStart VM

Cloudera QuickStart VMs are available as Zip archives in VMware, KVM, and VirtualBox formats. Cloudera recommends that you use 7-Zip to extract these files, when possible. (7-Zip performs well with large files.)

3.5.3 Downloading the VM

Select the hypervisor applicationof your choice. Available options are:

- Oracle VirtualBox(free/opensource)
- VMWare(multiple versions to choose from, some cost money, the "player" is free)
- KVM(free/opensource)

Once your hypervisor application is installed and working on your desktop, select a zip file to downloadwhich matches the hypervisor you've chosen. We offer 1 zip file for each of the above hypervisors.

It is recommended to use the 7-zip application if you are on a Windows desktop, to extract the contents of the downloaded zip file. Linux has an included “unzip” command which should work fine. On Mac OSX, it is recommended to use the “tar” command on large files, for example,

tar xzvf cloudera-quickstart-vm-5.5.0-0-virtualbox.zip

3.5.4 Installing the VM

Using the specific documentation and instructions provided by your hypervisor application, open the extracted file into that hypervisor application.

For example, if you elected to use VirtualBox, you would have downloaded and extracted a *.ovf file from Cloudera. Use the “File -> Import Appliance” menu inside VirtualBox to open your downloaded *.ovf file, or simply double-click on the file itself and VirtualBox should handle it from there.

If you elected to use VMWare, you would have downloaded and extracted a *.vmx file and a *.vmdk file from Cloudera. These two files combine to help VMWare Workstation or Player open the Quickstart VM. See these instructions for further detail.

3.5.5 Information on Accounts

Once you launch the VM, you are automatically logged in as the cloudera user. The account details are:

- username: cloudera
- password: cloudera

The cloudera account has sudo privileges in the VM. The root account password is cloudera.
The root MySQL password (and the password for other MySQL user accounts) is also cloudera.
Hue and Cloudera Manager use the same credentials.

4 Image Conversion

Image is a one among the most essential procedures to address data capably and enough utilized since old conditions. Image is medium for nonverbal correspondence which is justified regardless of a thousand words. Image passes on message quick and adequately with enough open door/open closures for creative Imagination when contrasted with words. As everybody knows, the Internet is a place where everything is about being visual and eye-getting. Computerized images are wherever you look on the Internet-from website pages and e-groups to mechanical research and e-distributing.

With image formats being the essential medium of advanced imaging, the JPEG is without a doubt the most generally utilized.

4.1 JPEG to PDF

Changing over JPEG pictures into PDF offers a more noteworthy number of positive conditions than securing an individual JPEG record itself.

4.1.1 Advantages

The jpeg picture quality is high with small degree of compression and also the format is compatible.

4.1.2 Disadvantages

When printing up JPEG images, the print nature of a image relies on the Pixel measurement. The Pixel measurement for screen and print determination are two distinct things. In this manner, with regards to the printing variable, what you get on screen isn't really what you'll get on paper. In any case, when in the PDF organize, you can print up precisely what you see.

4.2 Image Selection

First Module in my Project is Image selection. In image selection we are selecting a folder of images which we are converting. Any images are selected from the folder for further processing. We will download the images from the cloud platform in which we are working. Input image format can be bmp, gif, jpg, pdf, png, psd, tiff, doc.

4.3 Image Conversion:

After giving the input path, now the image conversion is started on the selected image. The image conversion is done in such way that the choosen image is changed over to the JPEG arrangement or PDF design as we required. We have given two arguments one argument is for input stream and the other is for the output stream. After executing the output document is made and the coveted picture configuration is put away in that file.

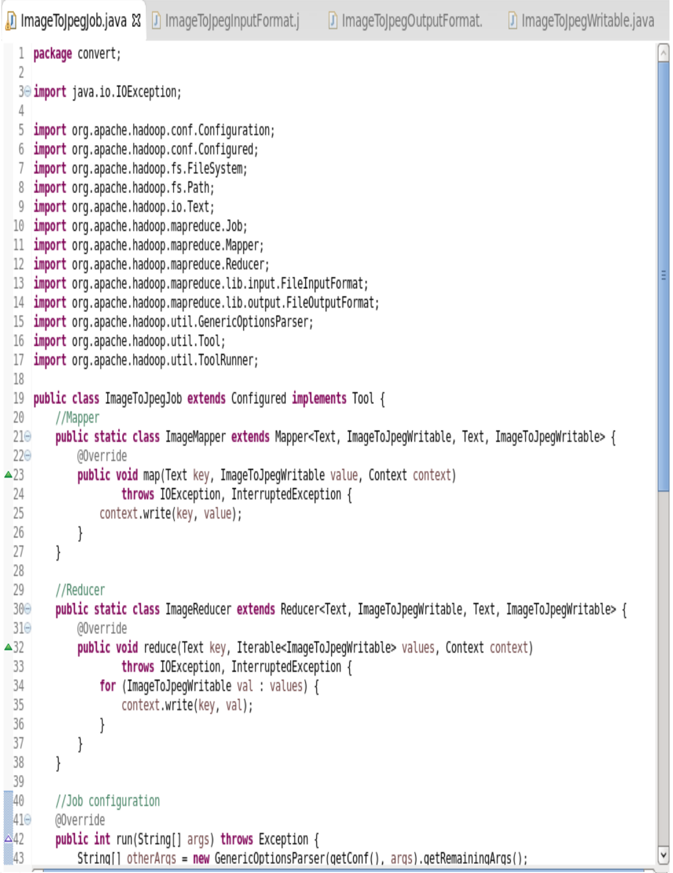
4.4 Store Images on Cloud:

Images which are handled are presently put away on the distributed storage to limit the substantial information storage issue. Presently these output pictures are put away in the cloud MapReduce Framework. Cloud storage is the primary application for web-based social networking, business applications, satellite pictures and photos and video.

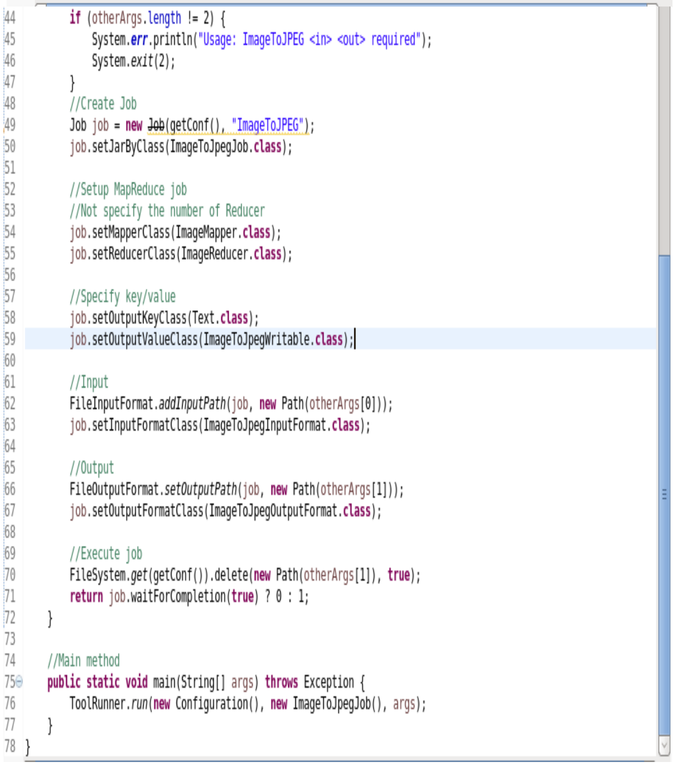
5 Program

5.1 Any Image to Jpeg Format

5.1.1 ImageToJpegJob

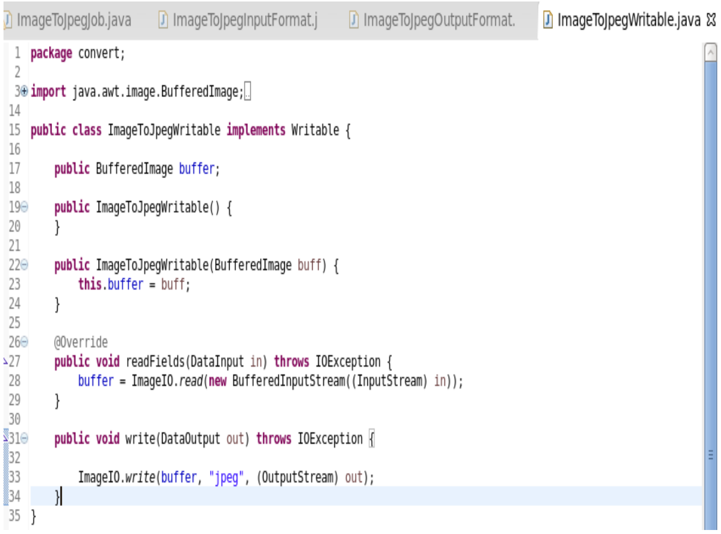


```
package convert;

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Text;
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 org.apache.hadoop.util.GenericOptionsParser;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;

public class ImageToJpegJob extends Configured implements Tool {
    //Mapper
    public static class ImageMapper extends Mapper<Text, ImageToJpegWritable, Text, ImageToJpegWritable> {
        @Override
        public void map(Text key, ImageToJpegWritable value, Context context)
                throws IOException, InterruptedException {
            context.write(key, value);
        }
    }

    //Reducer
    public static class ImageReducer extends Reducer<Text, ImageToJpegWritable, Text, ImageToJpegWritable> {
        @Override
        public void reduce(Text key, Iterable<ImageToJpegWritable> values, Context context)
                throws IOException, InterruptedException {
            for (ImageToJpegWritable val : values) {
                context.write(key, val);
            }
        }
    }

    //Job configuration
    @Override
    public int run(String[] args) throws Exception {
        String[] otherArgs = new GenericOptionsParser(getConf(), args).getRemainingArgs();
```



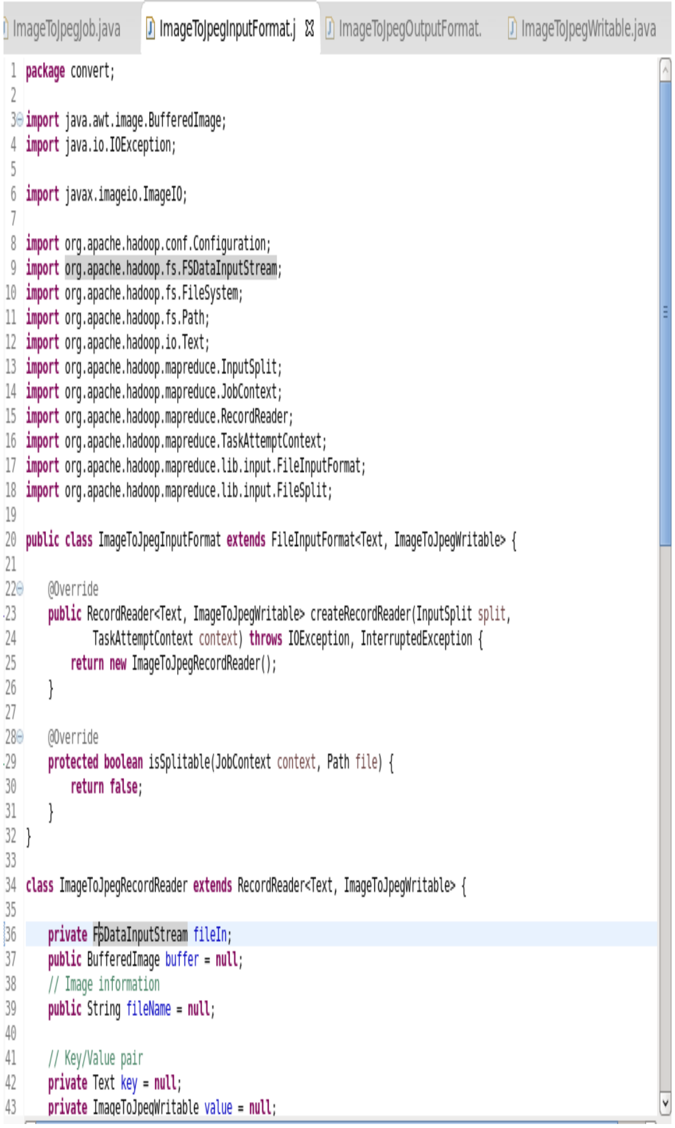
```
        if (otherArgs.length != 2) {
            System.err.println("Usage: ImageToJPEG <in> <out> required");
            System.exit(2);
        }
        //Create Job
        Job job = new Job(getConf(), "ImageToJPEG");
        job.setJarByClass(ImageToJpegJob.class);

        //Setup MapReduce job
        //Not specify the number of Reducer
        job.setMapperClass(ImageMapper.class);
        job.setReducerClass(ImageReducer.class);

        //Specify key/value
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(ImageToJpegWritable.class);

        //Input
        FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
        job.setInputFormatClass(ImageToJpegInputFormat.class);

        //Output
        FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
        job.setOutputFormatClass(ImageToJpegOutputFormat.class);

        //Execute job
        FileSystem.get(getConf()).delete(new Path(otherArgs[1]), true);
        return job.waitForCompletion(true) ? 0 : 1;
    }

    //Main method
    public static void main(String[] args) throws Exception {
        ToolRunner.run(new Configuration(), new ImageToJpegJob(), args);
    }
}
```

5.1.2 ImageToJpegWritable

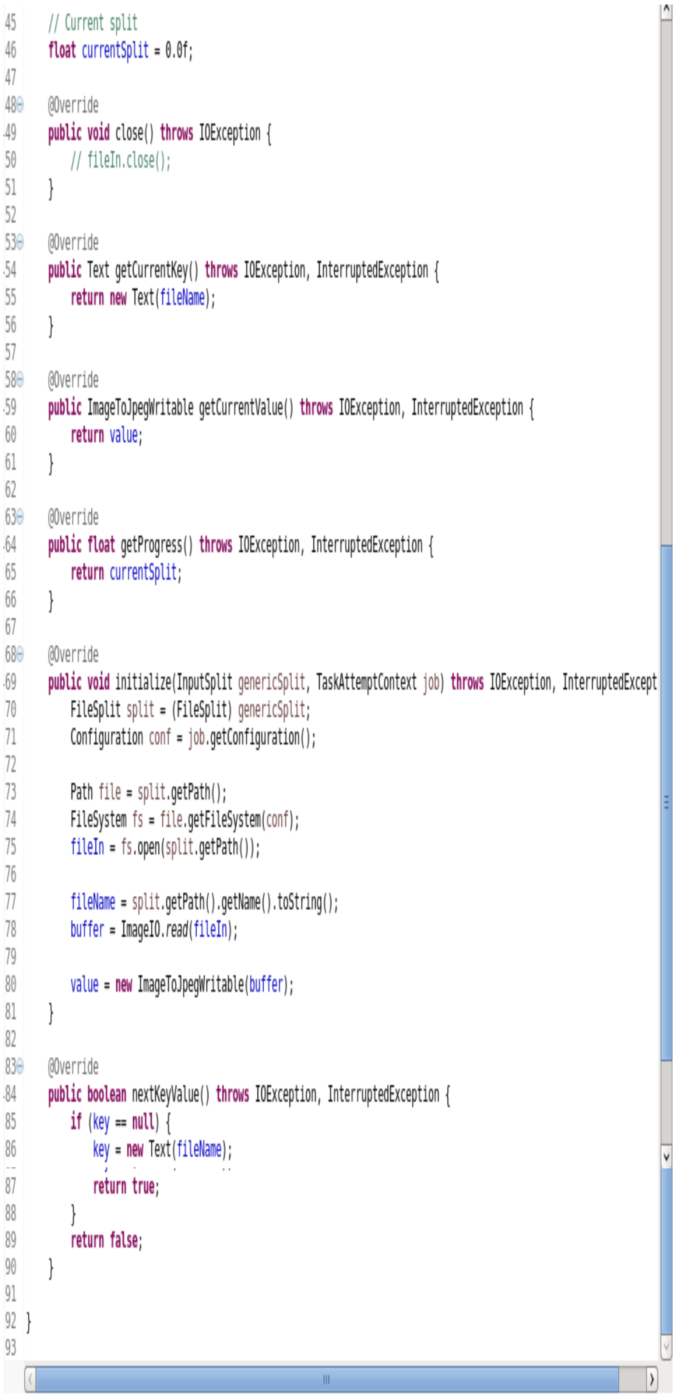


```
package convert;

import java.awt.image.BufferedImage;

public class ImageToJpegWritable implements Writable {

    public BufferedImage buffer;

    public ImageToJpegWritable() {
    }

    public ImageToJpegWritable(BufferedImage buff) {
        this.buffer = buff;
    }

    @Override
    public void readFields(DataInput in) throws IOException {
        buffer = ImageIO.read(new BufferedInputStream((InputStream) in));
    }

    public void write(DataOutput out) throws IOException {

        ImageIO.write(buffer, "jpeg", (OutputStream) out);
    }
}
```

5.1.3 ImageToJpegInputFormat

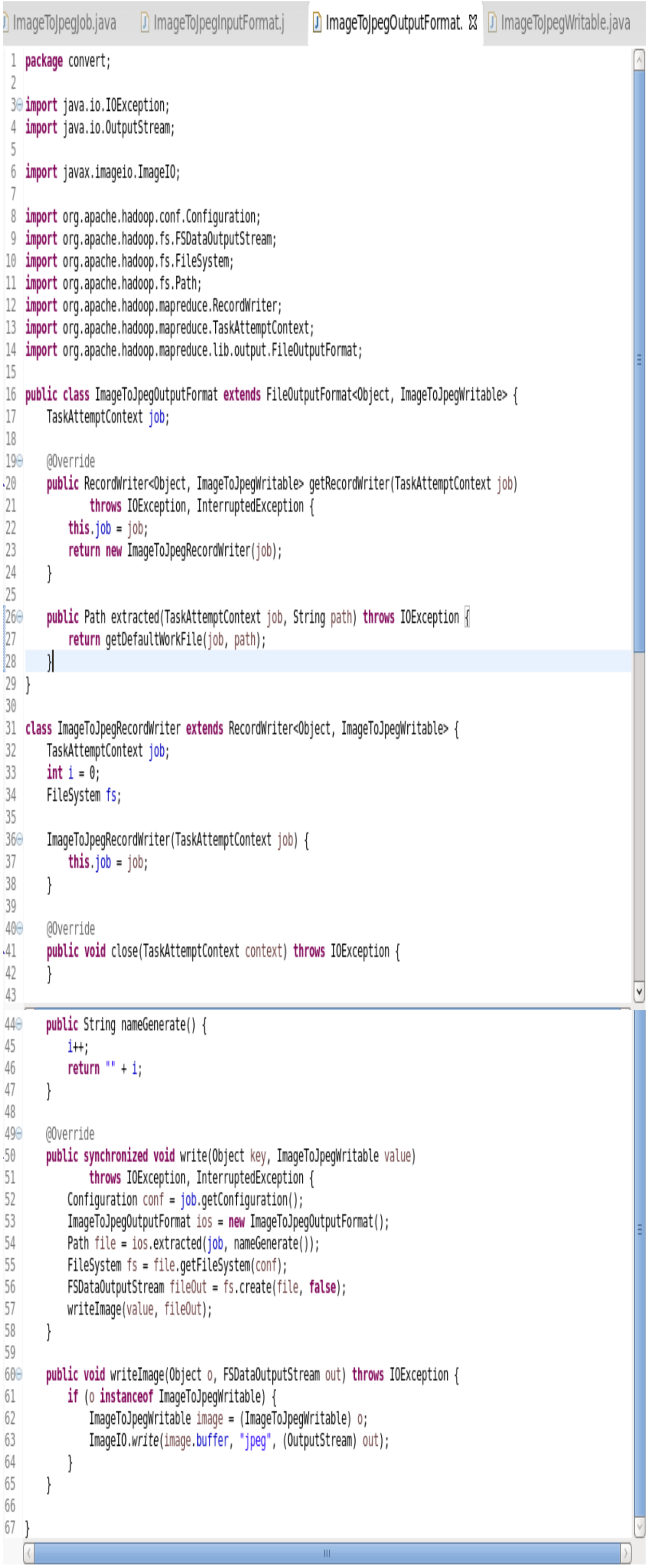


```
package convert;

import java.awt.image.BufferedImage;
import java.io.IOException;

import javax.imageio.ImageIO;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FSDataInputStream;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.InputSplit;
import org.apache.hadoop.mapreduce.JobContext;
import org.apache.hadoop.mapreduce.RecordReader;
import org.apache.hadoop.mapreduce.TaskAttemptContext;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.FileSplit;

public class ImageToJpegInputFormat extends FileInputFormat<Text, ImageToJpegWritable> {

    @Override
    public RecordReader<Text, ImageToJpegWritable> createRecordReader(InputSplit split,
            TaskAttemptContext context) throws IOException, InterruptedException {
        return new ImageToJpegRecordReader();
    }

    @Override
    protected boolean isSplitable(JobContext context, Path file) {
        return false;
    }
}

class ImageToJpegRecordReader extends RecordReader<Text, ImageToJpegWritable> {

    private FSDataInputStream fileIn;
    public BufferedImage buffer = null;
    // Image information
    public String fileName = null;

    // Key/Value pair
    private Text key = null;
    private ImageToJpegWritable value = null;
```



```java
    // Current split
    float currentSplit = 0.0f;

    @Override
    public void close() throws IOException {
        // fileIn.close();
    }

    @Override
    public Text getCurrentKey() throws IOException, InterruptedException {
        return new Text(fileName);
    }

    @Override
    public ImageToJpegWritable getCurrentValue() throws IOException, InterruptedException {
        return value;
    }

    @Override
    public float getProgress() throws IOException, InterruptedException {
        return currentSplit;
    }

    @Override
    public void initialize(InputSplit genericSplit, TaskAttemptContext job) throws IOException, InterruptedExcept
        FileSplit split = (FileSplit) genericSplit;
        Configuration conf = job.getConfiguration();

        Path file = split.getPath();
        FileSystem fs = file.getFileSystem(conf);
        fileIn = fs.open(split.getPath());

        fileName = split.getPath().getName().toString();
        buffer = ImageIO.read(fileIn);

        value = new ImageToJpegWritable(buffer);
    }

    @Override
    public boolean nextKeyValue() throws IOException, InterruptedException {
        if (key == null) {
            key = new Text(fileName);
            return true;
        }
        return false;
    }

}
```

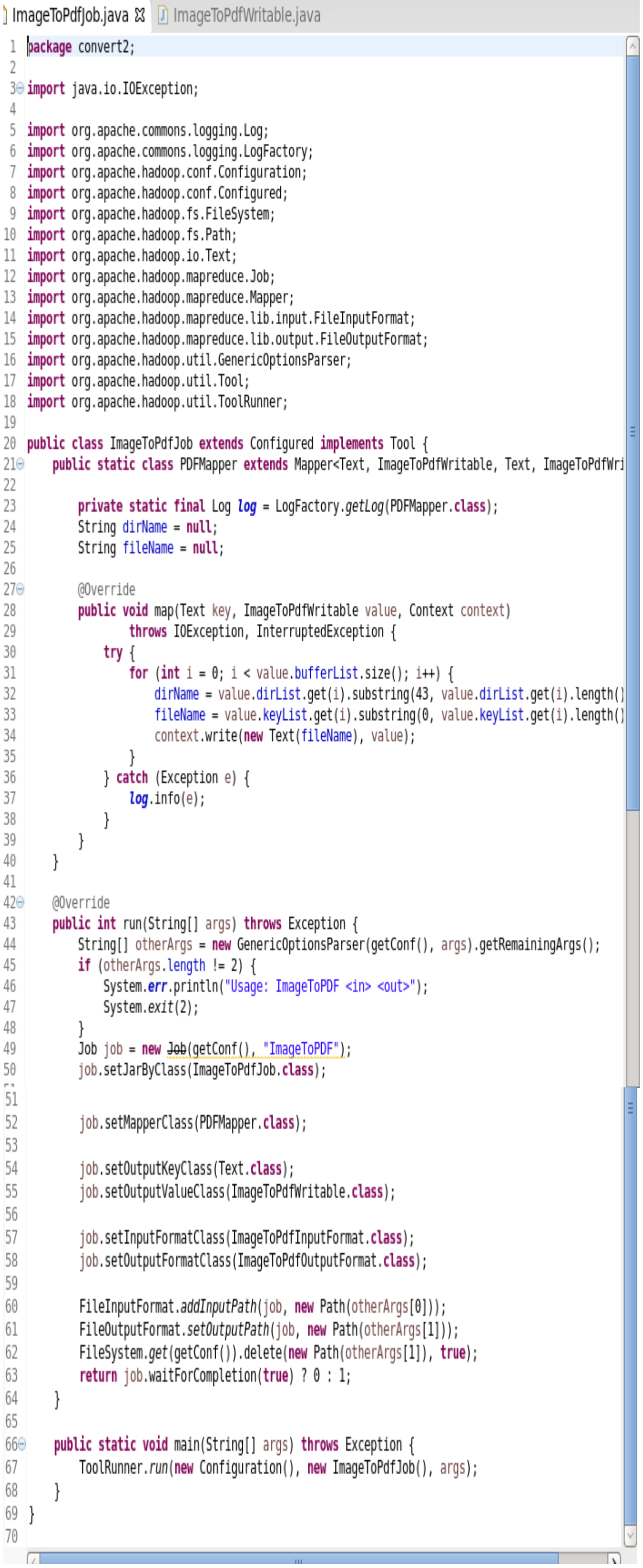
5.1.4 ImageToJpegOutputformat



```java
package convert;

import java.io.IOException;
import java.io.OutputStream;

import javax.imageio.ImageIO;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FSDataOutputStream;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.mapreduce.RecordWriter;
import org.apache.hadoop.mapreduce.TaskAttemptContext;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class ImageToJpegOutputFormat extends FileOutputFormat<Object, ImageToJpegWritable> {
    TaskAttemptContext job;

    @Override
    public RecordWriter<Object, ImageToJpegWritable> getRecordWriter(TaskAttemptContext job)
            throws IOException, InterruptedException {
        this.job = job;
        return new ImageToJpegRecordWriter(job);
    }

    public Path extracted(TaskAttemptContext job, String path) throws IOException {
        return getDefaultWorkFile(job, path);
    }
}

class ImageToJpegRecordWriter extends RecordWriter<Object, ImageToJpegWritable> {
    TaskAttemptContext job;
    int i = 0;
    FileSystem fs;

    ImageToJpegRecordWriter(TaskAttemptContext job) {
        this.job = job;
    }

    @Override
    public void close(TaskAttemptContext context) throws IOException {
    }

    public String nameGenerate() {
        i++;
        return "" + i;
    }

    @Override
    public synchronized void write(Object key, ImageToJpegWritable value)
            throws IOException, InterruptedException {
        Configuration conf = job.getConfiguration();
        ImageToJpegOutputFormat ios = new ImageToJpegOutputFormat();
        Path file = ios.extracted(job, nameGenerate());
        FileSystem fs = file.getFileSystem(conf);
        FSDataOutputStream fileOut = fs.create(file, false);
        writeImage(value, fileOut);
    }

    public void writeImage(Object o, FSDataOutputStream out) throws IOException {
        if (o instanceof ImageToJpegWritable) {
            ImageToJpegWritable image = (ImageToJpegWritable) o;
            ImageIO.write(image.buffer, "jpeg", (OutputStream) out);
        }
    }

}
```

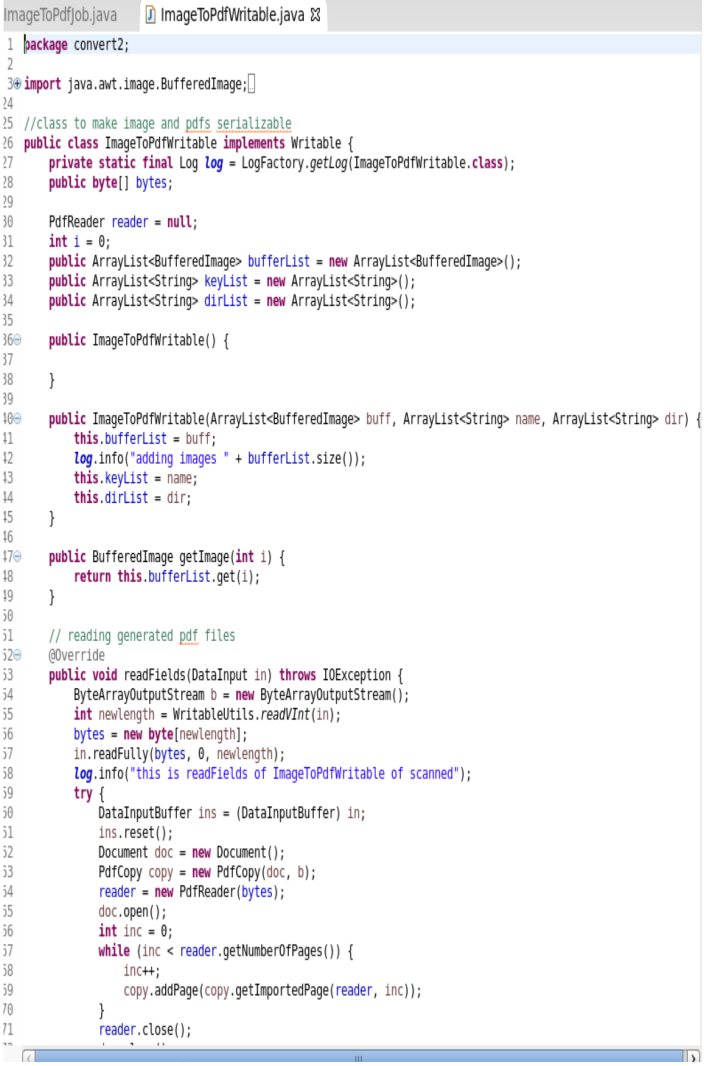
5.2 Any Image to Pdf Format

5.2.1 ImageToPdfJob

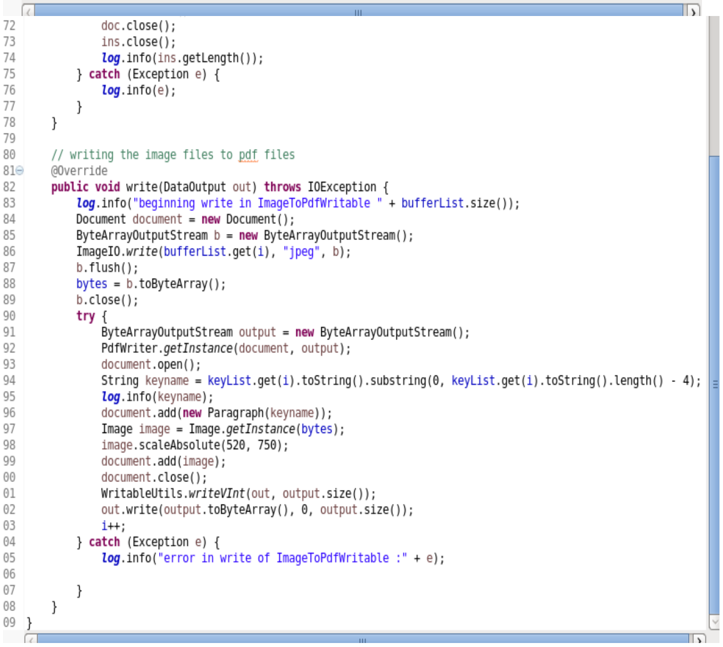


```java
package convert2;

import java.io.IOException;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;

public class ImageToPdfJob extends Configured implements Tool {
    public static class PDFMapper extends Mapper<Text, ImageToPdfWritable, Text, ImageToPdfWri

        private static final Log log = LogFactory.getLog(PDFMapper.class);
        String dirName = null;
        String fileName = null;

        @Override
        public void map(Text key, ImageToPdfWritable value, Context context)
                throws IOException, InterruptedException {
            try {
                for (int i = 0; i < value.bufferList.size(); i++) {
                    dirName = value.dirList.get(i).substring(43, value.dirList.get(i).length()
                    fileName = value.keyList.get(i).substring(0, value.keyList.get(i).length()
                    context.write(new Text(fileName), value);
                }
            } catch (Exception e) {
                log.info(e);
            }
        }
    }

    @Override
    public int run(String[] args) throws Exception {
        String[] otherArgs = new GenericOptionsParser(getConf(), args).getRemainingArgs();
        if (otherArgs.length != 2) {
            System.err.println("Usage: ImageToPDF <in> <out>");
            System.exit(2);
        }
        Job job = new Job(getConf(), "ImageToPDF");
        job.setJarByClass(ImageToPdfJob.class);


        job.setMapperClass(PDFMapper.class);

        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(ImageToPdfWritable.class);

        job.setInputFormatClass(ImageToPdfInputFormat.class);
        job.setOutputFormatClass(ImageToPdfOutputFormat.class);

        FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
        FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
        FileSystem.get(getConf()).delete(new Path(otherArgs[1]), true);
        return job.waitForCompletion(true) ? 0 : 1;
    }

    public static void main(String[] args) throws Exception {
        ToolRunner.run(new Configuration(), new ImageToPdfJob(), args);
    }
}
```

5.2.2 ImageToPdfWritable

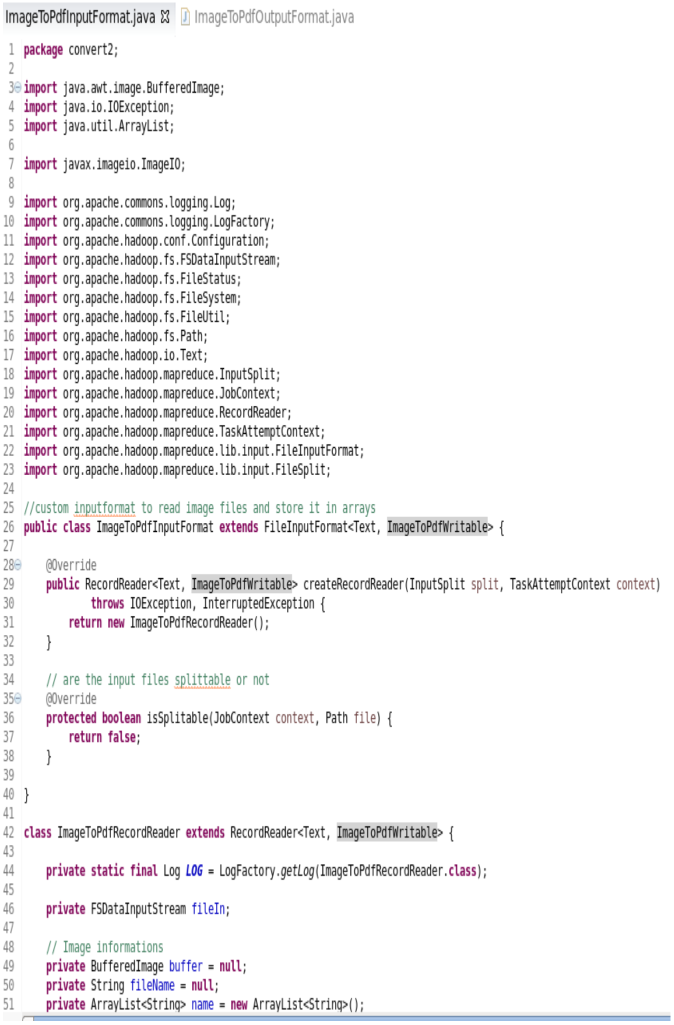


```java
package convert2;

import java.awt.image.BufferedImage;

//class to make image and pdfs serializable
public class ImageToPdfWritable implements Writable {
    private static final Log log = LogFactory.getLog(ImageToPdfWritable.class);
    public byte[] bytes;

    PdfReader reader = null;
    int i = 0;
    public ArrayList<BufferedImage> bufferList = new ArrayList<BufferedImage>();
    public ArrayList<String> keyList = new ArrayList<String>();
    public ArrayList<String> dirList = new ArrayList<String>();

    public ImageToPdfWritable() {

    }

    public ImageToPdfWritable(ArrayList<BufferedImage> buff, ArrayList<String> name, ArrayList<String> dir) {
        this.bufferList = buff;
        log.info("adding images " + bufferList.size());
        this.keyList = name;
        this.dirList = dir;
    }

    public BufferedImage getImage(int i) {
        return this.bufferList.get(i);
    }

    // reading generated pdf files
    @Override
    public void readFields(DataInput in) throws IOException {
        ByteArrayOutputStream b = new ByteArrayOutputStream();
        int newlength = WritableUtils.readVInt(in);
        bytes = new byte[newlength];
        in.readFully(bytes, 0, newlength);
        log.info("this is readFields of ImageToPdfWritable of scanned");
        try {
            DataInputBuffer ins = (DataInputBuffer) in;
            ins.reset();
            Document doc = new Document();
            PdfCopy copy = new PdfCopy(doc, b);
            reader = new PdfReader(bytes);
            doc.open();
            int inc = 0;
            while (inc < reader.getNumberOfPages()) {
                inc++;
                copy.addPage(copy.getImportedPage(reader, inc));
            }
            reader.close();
```

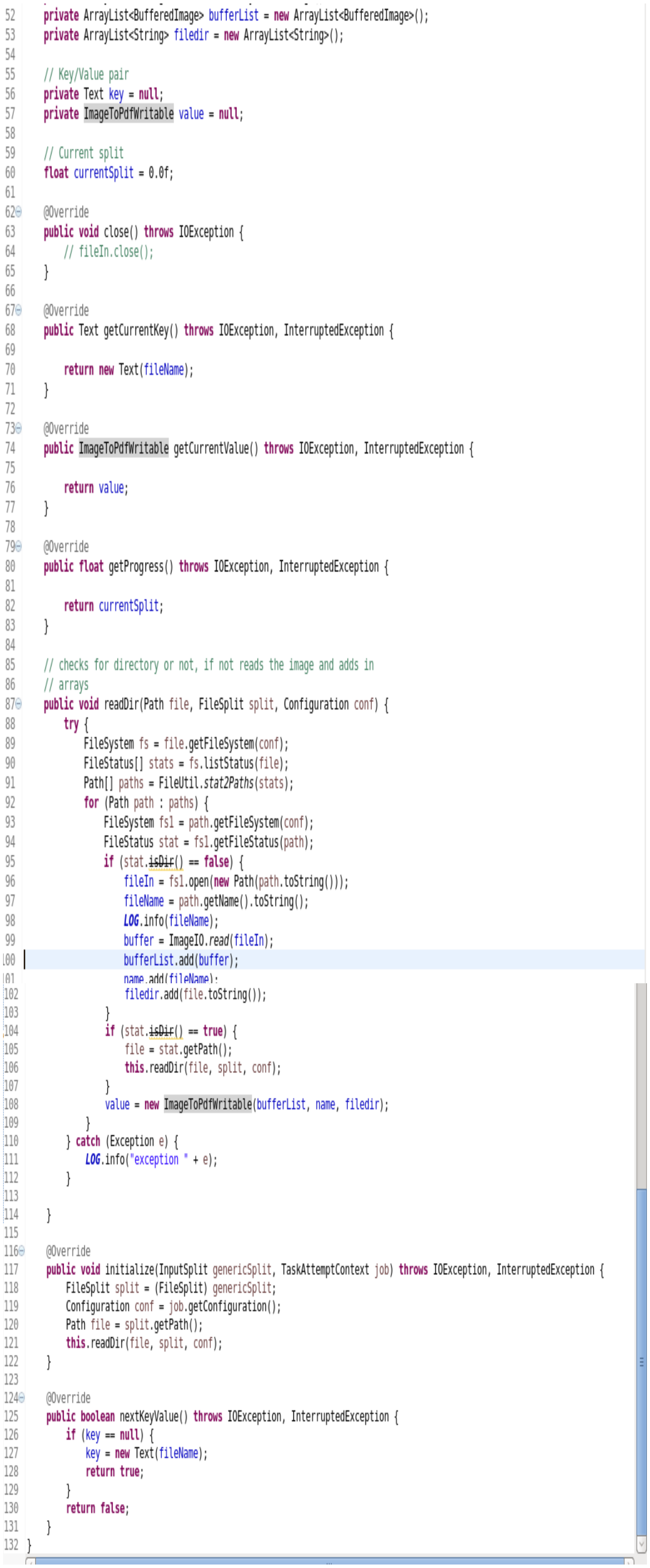


```java
            doc.close();
            ins.close();
            log.info(ins.getLength());
        } catch (Exception e) {
            log.info(e);
        }
    }

    // writing the image files to pdf files
    @Override
    public void write(DataOutput out) throws IOException {
        log.info("beginning write in ImageToPdfWritable " + bufferList.size());
        Document document = new Document();
        ByteArrayOutputStream b = new ByteArrayOutputStream();
        ImageIO.write(bufferList.get(i), "jpeg", b);
        b.flush();
        bytes = b.toByteArray();
        b.close();
        try {
            ByteArrayOutputStream output = new ByteArrayOutputStream();
            PdfWriter.getInstance(document, output);
            document.open();
            String keyname = keyList.get(i).toString().substring(0, keyList.get(i).toString().length() - 4);
            log.info(keyname);
            document.add(new Paragraph(keyname));
            Image image = Image.getInstance(bytes);
            image.scaleAbsolute(520, 750);
            document.add(image);
            document.close();
            WritableUtils.writeVInt(out, output.size());
            out.write(output.toByteArray(), 0, output.size());
            i++;
        } catch (Exception e) {
            log.info("error in write of ImageToPdfWritable :" + e);

        }
    }
}
```

5.2.3 ImagetoPdfInputFormat

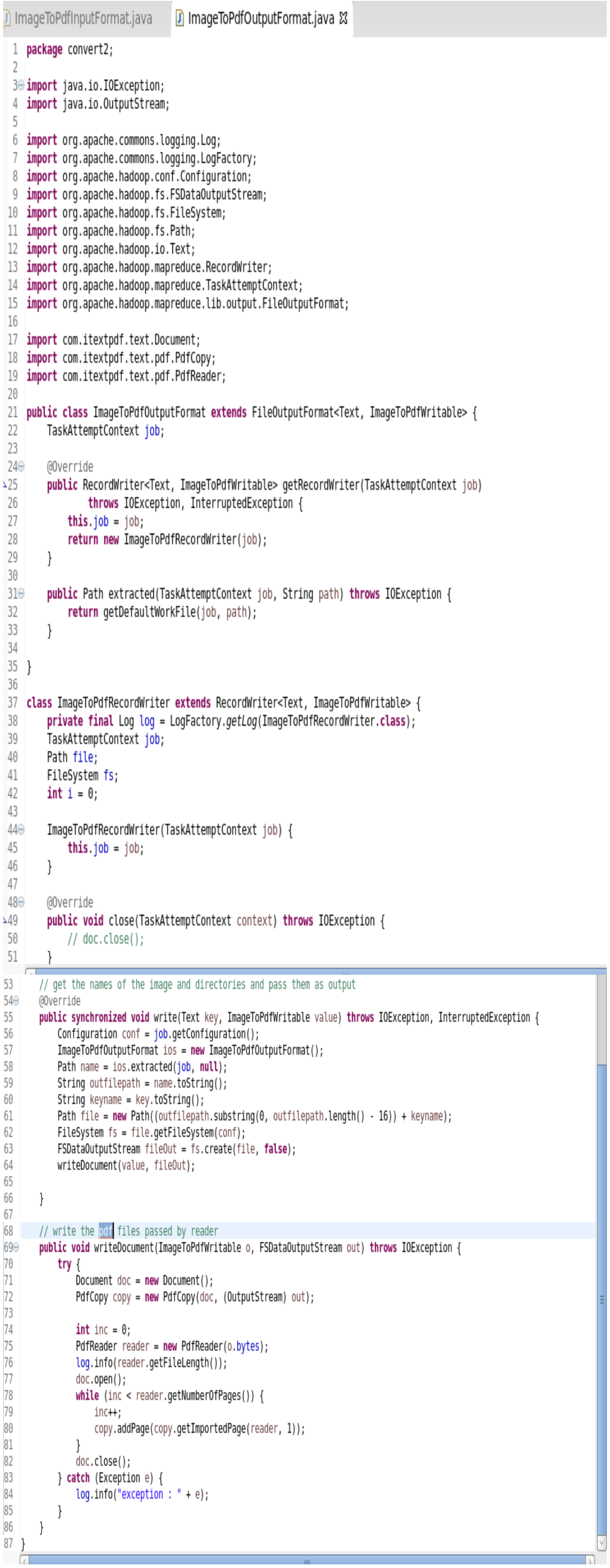


```java
package convert2;

import java.awt.image.BufferedImage;
import java.io.IOException;
import java.util.ArrayList;

import javax.imageio.ImageIO;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FSDataInputStream;
import org.apache.hadoop.fs.FileStatus;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.FileUtil;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.InputSplit;
import org.apache.hadoop.mapreduce.JobContext;
import org.apache.hadoop.mapreduce.RecordReader;
import org.apache.hadoop.mapreduce.TaskAttemptContext;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.FileSplit;

//custom inputformat to read image files and store it in arrays
public class ImageToPdfInputFormat extends FileInputFormat<Text, ImageToPdfWritable> {

    @Override
    public RecordReader<Text, ImageToPdfWritable> createRecordReader(InputSplit split, TaskAttemptContext context)
            throws IOException, InterruptedException {
        return new ImageToPdfRecordReader();
    }

    // are the input files splittable or not
    @Override
    protected boolean isSplitable(JobContext context, Path file) {
        return false;
    }

}

class ImageToPdfRecordReader extends RecordReader<Text, ImageToPdfWritable> {

    private static final Log LOG = LogFactory.getLog(ImageToPdfRecordReader.class);

    private FSDataInputStream fileIn;

    // Image informations
    private BufferedImage buffer = null;
    private String fileName = null;
    private ArrayList<String> name = new ArrayList<String>();
```



```java
    private ArrayList<BufferedImage> bufferList = new ArrayList<BufferedImage>();
    private ArrayList<String> filedir = new ArrayList<String>();

    // Key/Value pair
    private Text key = null;
    private ImageToPdfWritable value = null;

    // Current split
    float currentSplit = 0.0f;

    @Override
    public void close() throws IOException {
        // fileIn.close();
    }

    @Override
    public Text getCurrentKey() throws IOException, InterruptedException {

        return new Text(fileName);
    }

    @Override
    public ImageToPdfWritable getCurrentValue() throws IOException, InterruptedException {

        return value;
    }

    @Override
    public float getProgress() throws IOException, InterruptedException {

        return currentSplit;
    }

    // checks for directory or not, if not reads the image and adds in
    // arrays
    public void readDir(Path file, FileSplit split, Configuration conf) {
        try {
            FileSystem fs = file.getFileSystem(conf);
            FileStatus[] stats = fs.listStatus(file);
            Path[] paths = FileUtil.stat2Paths(stats);
            for (Path path : paths) {
                FileSystem fs1 = path.getFileSystem(conf);
                FileStatus stat = fs1.getFileStatus(path);
                if (stat.~~isDir~~() == false) {
                    fileIn = fs1.open(new Path(path.toString()));
                    fileName = path.getName().toString();
                    LOG.info(fileName);
                    buffer = ImageIO.read(fileIn);
                    bufferList.add(buffer);
                    name.add(fileName);
                    filedir.add(file.toString());
                }
                if (stat.~~isDir~~() == true) {
                    file = stat.getPath();
                    this.readDir(file, split, conf);
                }
                value = new ImageToPdfWritable(bufferList, name, filedir);
            }
        } catch (Exception e) {
            LOG.info("exception " + e);
        }

    }

    @Override
    public void initialize(InputSplit genericSplit, TaskAttemptContext job) throws IOException, InterruptedException {
        FileSplit split = (FileSplit) genericSplit;
        Configuration conf = job.getConfiguration();
        Path file = split.getPath();
        this.readDir(file, split, conf);
    }

    @Override
    public boolean nextKeyValue() throws IOException, InterruptedException {
        if (key == null) {
            key = new Text(fileName);
            return true;
        }
        return false;
    }
}
```

5.2.4 ImageToPdfOutputFormat

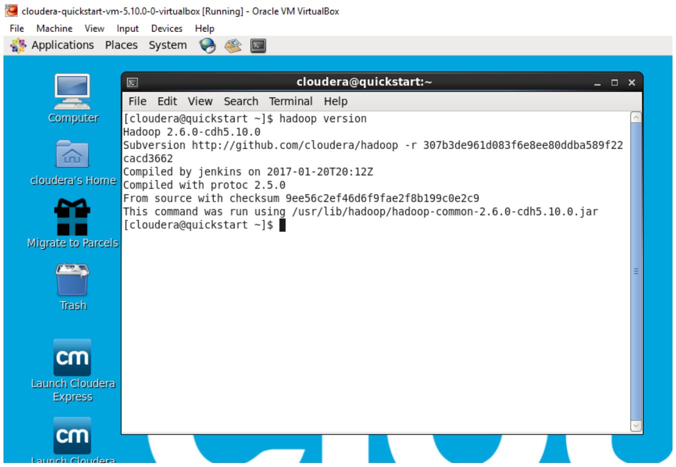


```java
package convert2;

import java.io.IOException;
import java.io.OutputStream;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FSDataOutputStream;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.RecordWriter;
import org.apache.hadoop.mapreduce.TaskAttemptContext;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import com.itextpdf.text.Document;
import com.itextpdf.text.pdf.PdfCopy;
import com.itextpdf.text.pdf.PdfReader;

public class ImageToPdfOutputFormat extends FileOutputFormat<Text, ImageToPdfWritable> {
    TaskAttemptContext job;

    @Override
    public RecordWriter<Text, ImageToPdfWritable> getRecordWriter(TaskAttemptContext job)
            throws IOException, InterruptedException {
        this.job = job;
        return new ImageToPdfRecordWriter(job);
    }

    public Path extracted(TaskAttemptContext job, String path) throws IOException {
        return getDefaultWorkFile(job, path);
    }

}

class ImageToPdfRecordWriter extends RecordWriter<Text, ImageToPdfWritable> {
    private final Log log = LogFactory.getLog(ImageToPdfRecordWriter.class);
    TaskAttemptContext job;
    Path file;
    FileSystem fs;
    int i = 0;

    ImageToPdfRecordWriter(TaskAttemptContext job) {
        this.job = job;
    }

    @Override
    public void close(TaskAttemptContext context) throws IOException {
        // doc.close();
    }
    // get the names of the image and directories and pass them as output
    @Override
    public synchronized void write(Text key, ImageToPdfWritable value) throws IOException, InterruptedException {
        Configuration conf = job.getConfiguration();
        ImageToPdfOutputFormat ios = new ImageToPdfOutputFormat();
        Path name = ios.extracted(job, null);
        String outfilepath = name.toString();
        String keyname = key.toString();
        Path file = new Path((outfilepath.substring(0, outfilepath.length() - 16)) + keyname);
        FileSystem fs = file.getFileSystem(conf);
        FSDataOutputStream fileOut = fs.create(file, false);
        writeDocument(value, fileOut);

    }

    // write the pdf files passed by reader
    public void writeDocument(ImageToPdfWritable o, FSDataOutputStream out) throws IOException {
        try {
            Document doc = new Document();
            PdfCopy copy = new PdfCopy(doc, (OutputStream) out);

            int inc = 0;
            PdfReader reader = new PdfReader(o.bytes);
            log.info(reader.getFileLength());
            doc.open();
            while (inc < reader.getNumberOfPages()) {
                inc++;
                copy.addPage(copy.getImportedPage(reader, 1));
            }
            doc.close();
        } catch (Exception e) {
            log.info("exception : " + e);
        }
    }
}
```

In the class ImageToJpegInputFormat we imported the split class whereas in class ImageToPdfFormat we used the image information as Arraylist.

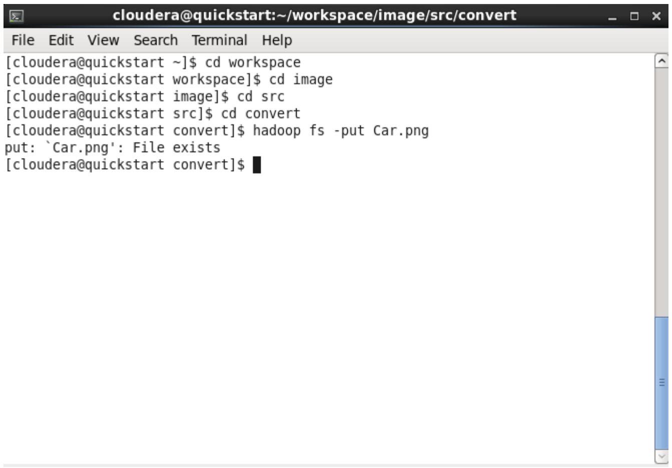
6 Execution

Checking if hadoop is installed on the host, by the below command it shows the version of the hadoop in the system.


cloudera-quickstart-vm-5.10.0-0-virtualbox [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Applications Places System
Computer
cloudera's Home
Migrate to Parcels
Trash
Launch Cloudera Express
Launch Cloudera
cloudera@quickstart:~
File Edit View Search Terminal Help

```
[cloudera@quickstart ~]$ hadoop version
Hadoop 2.6.0-cdh5.10.0
Subversion http://github.com/cloudera/hadoop -r 307b3de961d083f6e8ee80ddba589f22
cacd3662
Compiled by jenkins on 2017-01-20T20:12Z
Compiled with protoc 2.5.0
From source with checksum 9ee56c2ef46d6f9fae2f8b199c0e2c9
This command was run using /usr/lib/hadoop/hadoop-common-2.6.0-cdh5.10.0.jar
[cloudera@quickstart ~]$
```

Saving the file to the Hadoop Distributed File System.


cloudera@quickstart:~/workspace/image/src/convert
File Edit View Search Terminal Help

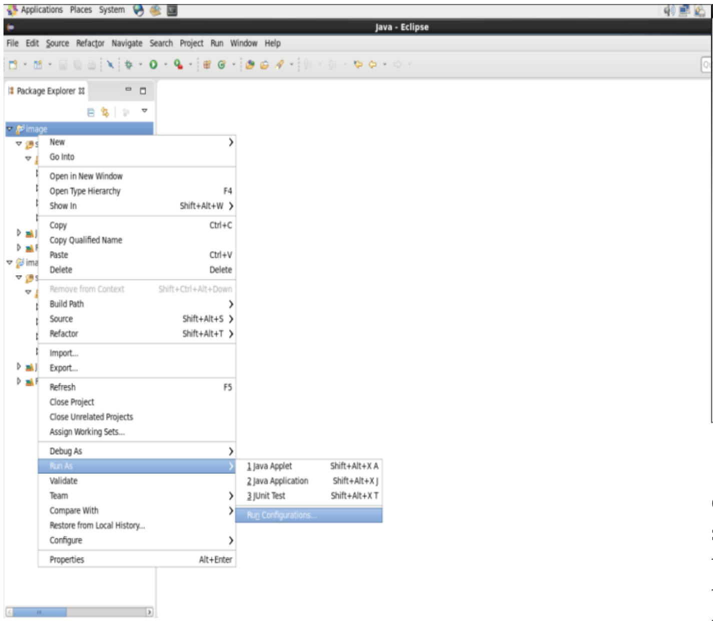
```
[cloudera@quickstart ~]$ cd workspace
[cloudera@quickstart workspace]$ cd image
[cloudera@quickstart image]$ cd src
[cloudera@quickstart src]$ cd convert
[cloudera@quickstart convert]$ hadoop fs -put Car.png
put: `Car.png': File exists
[cloudera@quickstart convert]$
```

The following commands are used for compiling the **ProcessUnits.java** program and creating a jar for the program.

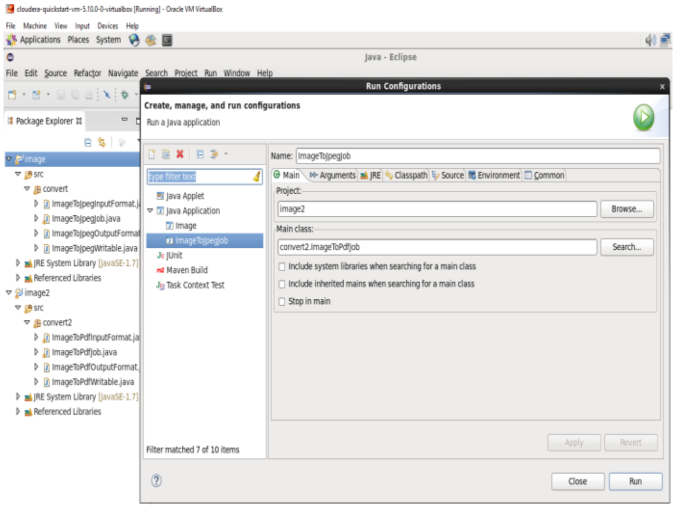
```
$ javac -classpath hadoop-core-1.2.1.jar -d units ProcessUnits.java
$ jar -cvf units.jar -C units/ .
```

Instead of the above commands , we can just run the program in Eclipse itself While executing it in the cloudera.The procedure is explained in detail below.

Select the executing program file, go to RunAs and select Run Configurations…



A window pops up and here select the Project and Main Class in the project. And now we should give the input file tobe executed and should give a path to store the ouput file.



Name: ImageToJpegJob

Project: image2

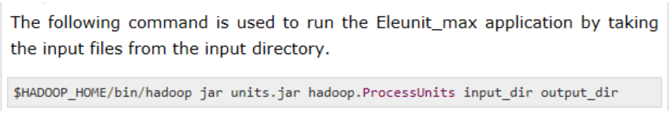
Main class: convert2.ImageToPdfJob

□ Include system libraries when searching for a main class

□ Include inherited mains when searching for a main class

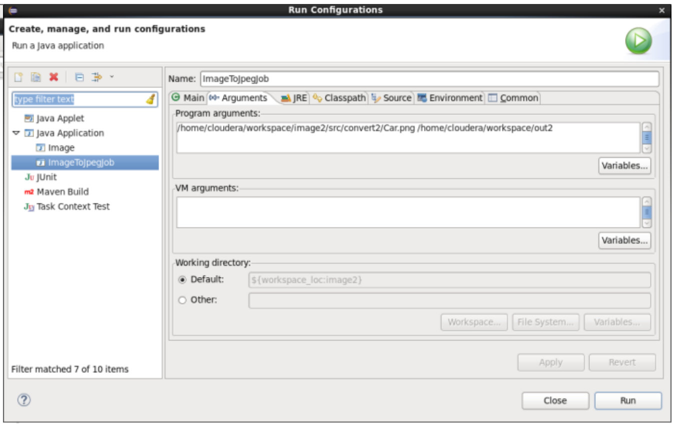
□ Stop in main

Below is the command to execute the program in unix system, but when we excute it in the cloudera manager just give the input and output path in the Arguments tab in the RunConfigurations_window.



The following command is used to run the Eleunit_max application by taking the input files from the input directory.

```
$HADOOP_HOME/bin/hadoop jar units.jar hadoop.ProcessUnits input_dir output_dir
```



Name: ImageToJpegJob

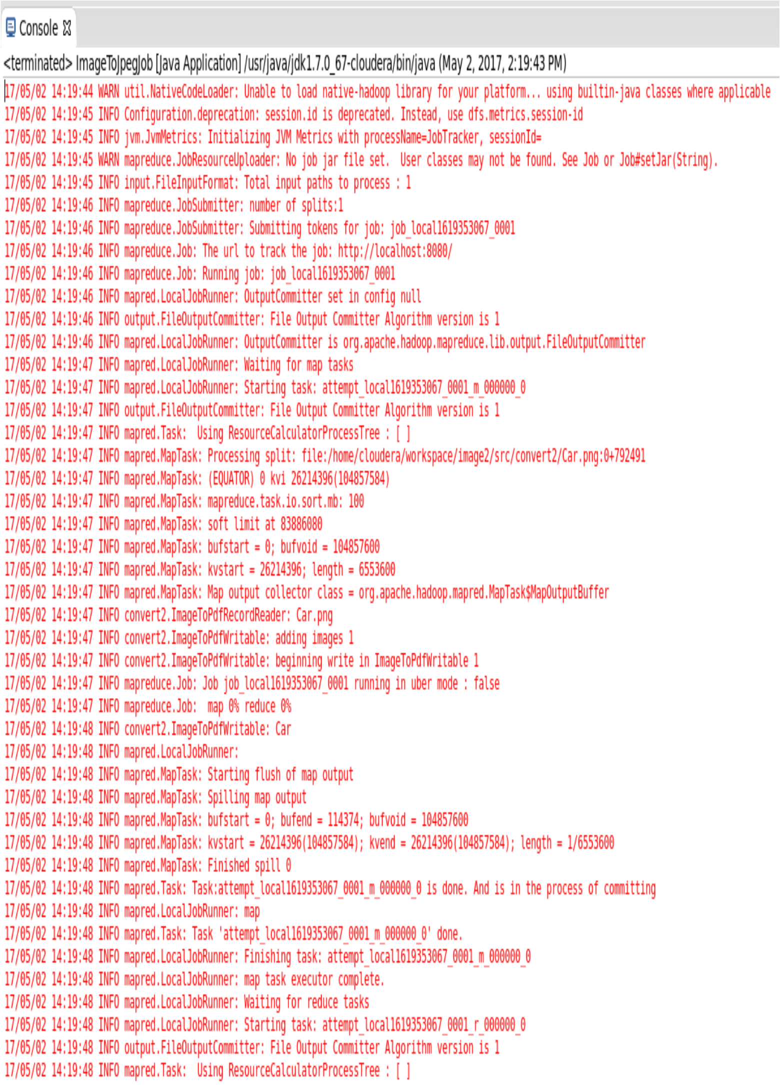
Program arguments: /home/cloudera/workspace/image2/src/convert2/Car.png /home/cloudera/workspace/out2

VM arguments:

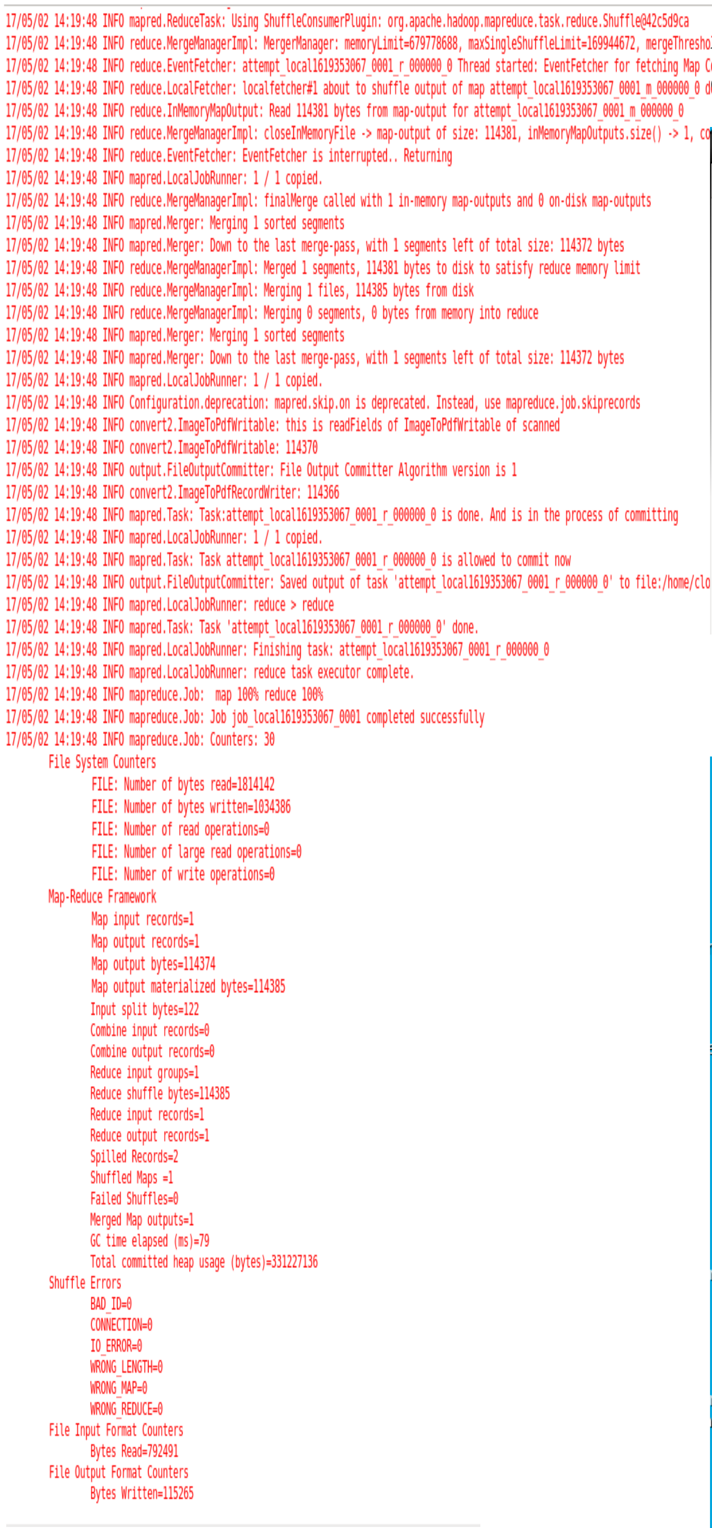
Working directory: ● Default: ${workspace_loc:image2} ○ Other:

Filter matched 7 of 10 items

Now click on run to execute the program. We can see the execution process in the Console Window.A input path selection is taken first and the path is splitted top strings and the file is taken as input. First the mapper task is executed and then the Reducer task is executed, we can see the above process clearly in the console below, first the map 0% reduce 0% indicates that no task has been started.



```
<terminated> ImageToJpegJob [Java Application] /usr/java/jdk1.7.0_67-cloudera/bin/java (May 2, 2017, 2:19:43 PM)
17/05/02 14:19:44 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
17/05/02 14:19:45 INFO Configuration.deprecation: session.id is deprecated. Instead, use dfs.metrics.session-id
17/05/02 14:19:45 INFO jvm.JvmMetrics: Initializing JVM Metrics with processName=JobTracker, sessionId=
17/05/02 14:19:45 WARN mapreduce.JobResourceUploader: No job jar file set.  User classes may not be found. See Job or Job#setJar(String).
17/05/02 14:19:45 INFO input.FileInputFormat: Total input paths to process : 1
17/05/02 14:19:46 INFO mapreduce.JobSubmitter: number of splits:1
17/05/02 14:19:46 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_local1619353067_0001
17/05/02 14:19:46 INFO mapreduce.Job: The url to track the job: http://localhost:8080/
17/05/02 14:19:46 INFO mapreduce.Job: Running job: job_local1619353067_0001
17/05/02 14:19:46 INFO mapred.LocalJobRunner: OutputCommitter set in config null
17/05/02 14:19:46 INFO output.FileOutputCommitter: File Output Committer Algorithm version is 1
17/05/02 14:19:46 INFO mapred.LocalJobRunner: OutputCommitter is org.apache.hadoop.mapreduce.lib.output.FileOutputCommitter
17/05/02 14:19:47 INFO mapred.LocalJobRunner: Waiting for map tasks
17/05/02 14:19:47 INFO mapred.LocalJobRunner: Starting task: attempt_local1619353067_0001_m_000000_0
17/05/02 14:19:47 INFO output.FileOutputCommitter: File Output Committer Algorithm version is 1
17/05/02 14:19:47 INFO mapred.Task:  Using ResourceCalculatorProcessTree : [ ]
17/05/02 14:19:47 INFO mapred.MapTask: Processing split: file:/home/cloudera/workspace/image2/src/convert2/Car.png:0+792491
17/05/02 14:19:47 INFO mapred.MapTask: (EQUATOR) 0 kvi 26214396(104857584)
17/05/02 14:19:47 INFO mapred.MapTask: mapreduce.task.io.sort.mb: 100
17/05/02 14:19:47 INFO mapred.MapTask: soft limit at 83886080
17/05/02 14:19:47 INFO mapred.MapTask: bufstart = 0; bufvoid = 104857600
17/05/02 14:19:47 INFO mapred.MapTask: kvstart = 26214396; length = 6553600
17/05/02 14:19:47 INFO mapred.MapTask: Map output collector class = org.apache.hadoop.mapred.MapTask$MapOutputBuffer
17/05/02 14:19:47 INFO convert2.ImageToPdfRecordReader: Car.png
17/05/02 14:19:47 INFO convert2.ImageToPdfWritable: adding images 1
17/05/02 14:19:47 INFO convert2.ImageToPdfWritable: beginning write in ImageToPdfWritable 1
17/05/02 14:19:47 INFO mapreduce.Job: Job job_local1619353067_0001 running in uber mode : false
17/05/02 14:19:47 INFO mapreduce.Job:  map 0% reduce 0%
17/05/02 14:19:48 INFO convert2.ImageToPdfWritable: Car
17/05/02 14:19:48 INFO mapred.LocalJobRunner:
17/05/02 14:19:48 INFO mapred.MapTask: Starting flush of map output
17/05/02 14:19:48 INFO mapred.MapTask: Spilling map output
17/05/02 14:19:48 INFO mapred.MapTask: bufstart = 0; bufend = 114374; bufvoid = 104857600
17/05/02 14:19:48 INFO mapred.MapTask: kvstart = 26214396(104857584); kvend = 26214396(104857584); length = 1/6553600
17/05/02 14:19:48 INFO mapred.MapTask: Finished spill 0
17/05/02 14:19:48 INFO mapred.Task: Task:attempt_local1619353067_0001_m_000000_0 is done. And is in the process of committing
17/05/02 14:19:48 INFO mapred.LocalJobRunner: map
17/05/02 14:19:48 INFO mapred.Task: Task 'attempt_local1619353067_0001_m_000000_0' done.
17/05/02 14:19:48 INFO mapred.LocalJobRunner: Finishing task: attempt_local1619353067_0001_m_000000_0
17/05/02 14:19:48 INFO mapred.LocalJobRunner: map task executor complete.
17/05/02 14:19:48 INFO mapred.LocalJobRunner: Waiting for reduce tasks
17/05/02 14:19:48 INFO mapred.LocalJobRunner: Starting task: attempt_local1619353067_0001_r_000000_0
17/05/02 14:19:48 INFO output.FileOutputCommitter: File Output Committer Algorithm version is 1
17/05/02 14:19:48 INFO mapred.Task:  Using ResourceCalculatorProcessTree : [ ]
```

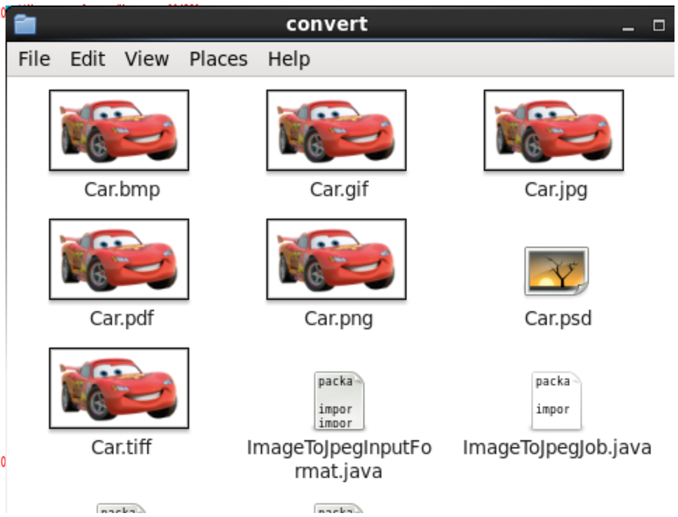


```
17/05/02 14:19:48 INFO mapred.ReduceTask: Using ShuffleConsumerPlugin: org.apache.hadoop.mapreduce.task.reduce.Shuffle@42c5d9ca
17/05/02 14:19:48 INFO reduce.MergeManagerImpl: MergerManager: memoryLimit=679778688, maxSingleShuffleLimit=169944672, mergeThreshold=448653952, ioSortFactor=10, memToMemMergeOutputsThreshold=10
17/05/02 14:19:48 INFO reduce.EventFetcher: attempt_local1619353067_0001_r_000000_0 Thread started: EventFetcher for fetching Map Completion Events
17/05/02 14:19:48 INFO reduce.LocalFetcher: localfetcher#1 about to shuffle output of map attempt_local1619353067_0001_m_000000_0 decomp: 114381 len: 114385 to MEMORY
17/05/02 14:19:48 INFO reduce.InMemoryMapOutput: Read 114381 bytes from map-output for attempt_local1619353067_0001_m_000000_0
17/05/02 14:19:48 INFO reduce.MergeManagerImpl: closeInMemoryFile -> map-output of size: 114381, inMemoryMapOutputs.size() -> 1, co
17/05/02 14:19:48 INFO reduce.EventFetcher: EventFetcher is interrupted.. Returning
17/05/02 14:19:48 INFO mapred.LocalJobRunner: 1 / 1 copied.
17/05/02 14:19:48 INFO reduce.MergeManagerImpl: finalMerge called with 1 in-memory map-outputs and 0 on-disk map-outputs
17/05/02 14:19:48 INFO mapred.Merger: Merging 1 sorted segments
17/05/02 14:19:48 INFO mapred.Merger: Down to the last merge-pass, with 1 segments left of total size: 114372 bytes
17/05/02 14:19:48 INFO reduce.MergeManagerImpl: Merged 1 segments, 114381 bytes to disk to satisfy reduce memory limit
17/05/02 14:19:48 INFO reduce.MergeManagerImpl: Merging 1 files, 114385 bytes from disk
17/05/02 14:19:48 INFO reduce.MergeManagerImpl: Merging 0 segments, 0 bytes from memory into reduce
17/05/02 14:19:48 INFO mapred.Merger: Merging 1 sorted segments
17/05/02 14:19:48 INFO mapred.Merger: Down to the last merge-pass, with 1 segments left of total size: 114372 bytes
17/05/02 14:19:48 INFO mapred.LocalJobRunner: 1 / 1 copied.
17/05/02 14:19:48 INFO Configuration.deprecation: mapred.skip.on is deprecated. Instead, use mapreduce.job.skiprecords
17/05/02 14:19:48 INFO convert2.ImageToPdfWritable: this is readFields of ImageToPdfWritable of scanned
17/05/02 14:19:48 INFO convert2.ImageToPdfWritable: 114370
17/05/02 14:19:48 INFO output.FileOutputCommitter: File Output Committer Algorithm version is 1
17/05/02 14:19:48 INFO convert2.ImageToPdfRecordWriter: 114366
17/05/02 14:19:48 INFO mapred.Task: Task:attempt_local1619353067_0001_r_000000_0 is done. And is in the process of committing
17/05/02 14:19:48 INFO mapred.LocalJobRunner: 1 / 1 copied.
17/05/02 14:19:48 INFO mapred.Task: Task attempt_local1619353067_0001_r_000000_0 is allowed to commit now
17/05/02 14:19:48 INFO output.FileOutputCommitter: Saved output of task 'attempt_local1619353067_0001_r_000000_0' to file:/home/clo
17/05/02 14:19:48 INFO mapred.LocalJobRunner: reduce > reduce
17/05/02 14:19:48 INFO mapred.Task: Task 'attempt_local1619353067_0001_r_000000_0' done.
17/05/02 14:19:48 INFO mapred.LocalJobRunner: Finishing task: attempt_local1619353067_0001_r_000000_0
17/05/02 14:19:48 INFO mapred.LocalJobRunner: reduce task executor complete.
17/05/02 14:19:48 INFO mapreduce.Job:  map 100% reduce 100%
17/05/02 14:19:48 INFO mapreduce.Job: Job job_local1619353067_0001 completed successfully
17/05/02 14:19:48 INFO mapreduce.Job: Counters: 30
	File System Counters
		FILE: Number of bytes read=1814142
		FILE: Number of bytes written=1034386
		FILE: Number of read operations=0
		FILE: Number of large read operations=0
		FILE: Number of write operations=0
	Map-Reduce Framework
		Map input records=1
		Map output records=1
		Map output bytes=114374
		Map output materialized bytes=114385
		Input split bytes=122
		Combine input records=0
		Combine output records=0
		Reduce input groups=1
		Reduce shuffle bytes=114385
		Reduce input records=1
		Reduce output records=1
		Spilled Records=2
		Shuffled Maps =1
		Failed Shuffles=0
		Merged Map outputs=1
		GC time elapsed (ms)=79
		Total committed heap usage (bytes)=331227136
	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=792491
	File Output Format Counters
		Bytes Written=115265
```

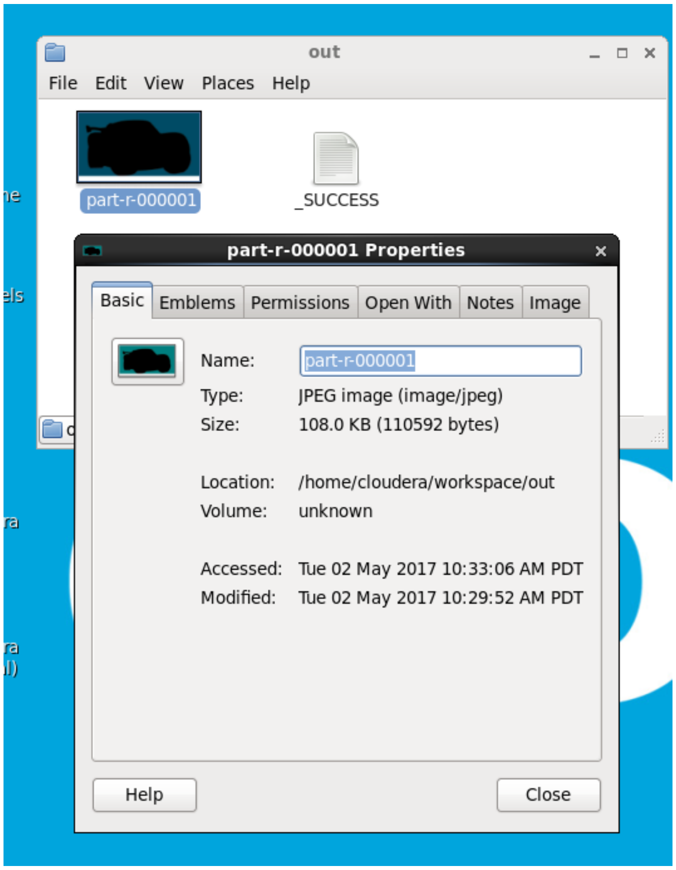
When both the Mapper and Reducer are executed it becomes map 100% reduce 100%, we see this in the below figure.

The counters displayed indicate the number of operations done in the process like, bytes read, written, time to execute,errors occured etc. See the above figure for details.

Below are the input image files in different formats and the Output Jpeg and pdf format files created after program execution.



The outputJPEG file created in the out folder as shown.

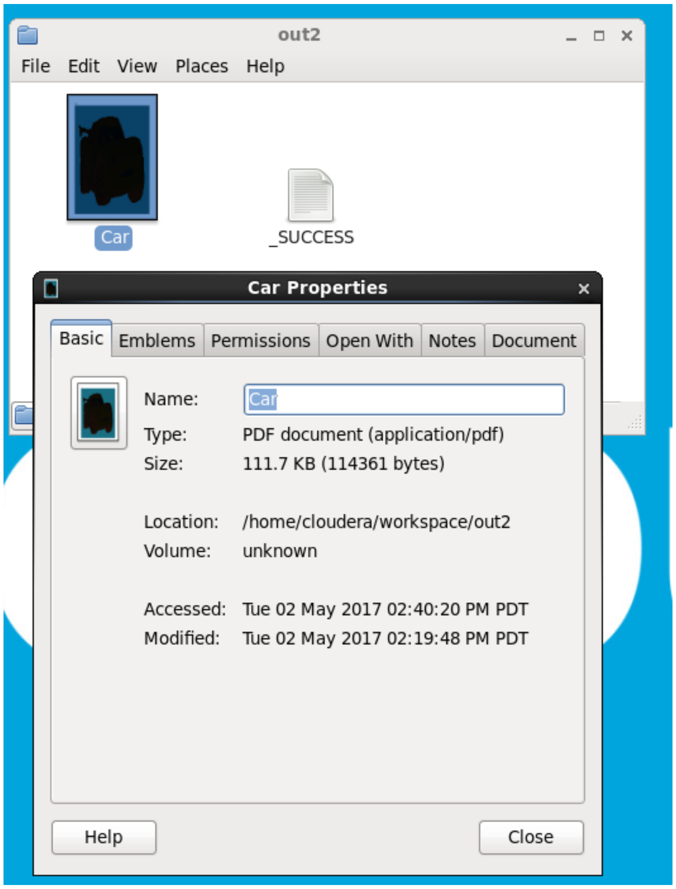

out
File Edit View Places Help
part-r-000001
_SUCCESS
part-r-000001 Properties
Basic Emblems Permissions Open With Notes Image
Help
Close

Name: part-r-000001
Type: JPEG image (image/jpeg)
Size: 108.0 KB (110592 bytes)

Location: /home/cloudera/workspace/out
Volume: unknown

Accessed: Tue 02 May 2017 10:33:06 AM PDT
Modified: Tue 02 May 2017 10:29:52 AM PDT

The output PDFfile is created as shown.

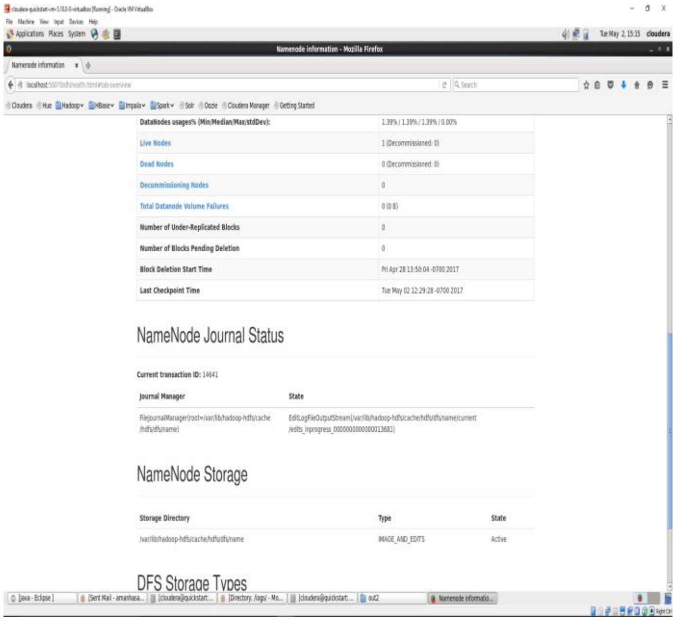


Name: Car
Type: PDF document (application/pdf)
Size: 111.7 KB (114361 bytes)

Location: /home/cloudera/workspace/out2
Volume: unknown

Accessed: Tue 02 May 2017 02:40:20 PM PDT
Modified: Tue 02 May 2017 02:19:48 PM PDT

We are using the URL ‘http://localhost :50070/’ for HDFS. We are using the URL ‘http://localhost :50030/’ for MapReduce.


cloudera-quickstart-vm-5.10.0-0-virtualbox [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Applications Places System
Tue May 2, 15:15 cloudera
Namenode information - Mozilla Firefox
Namenode information
localhost:50070/dfshealth.html#tab-overview
Search
Cloudera Hue Hadoop HBase Impala Spark Solr Oozie Cloudera Manager Getting Started
[java - Eclipse] [Sent Mail - amanhasa... [cloudera@quickstart:... [Directory: /logs/ - Mo... [cloudera@quickstart:... out2 Namenode informatio...

DataNodes usages% (Min/Median/Max/stdDev):	1.39% / 1.39% / 1.39% / 0.00%
Live Nodes	1 (Decommissioned: 0)
Dead Nodes	0 (Decommissioned: 0)
Decommissioning Nodes	0
Total Datanode Volume Failures	0 (0 B)
Number of Under-Replicated Blocks	0
Number of Blocks Pending Deletion	0
Block Deletion Start Time	Fri Apr 28 13:50:04 -0700 2017
Last Checkpoint Time	Tue May 02 12:29:28 -0700 2017

NameNode Journal Status

Current transaction ID: 14641

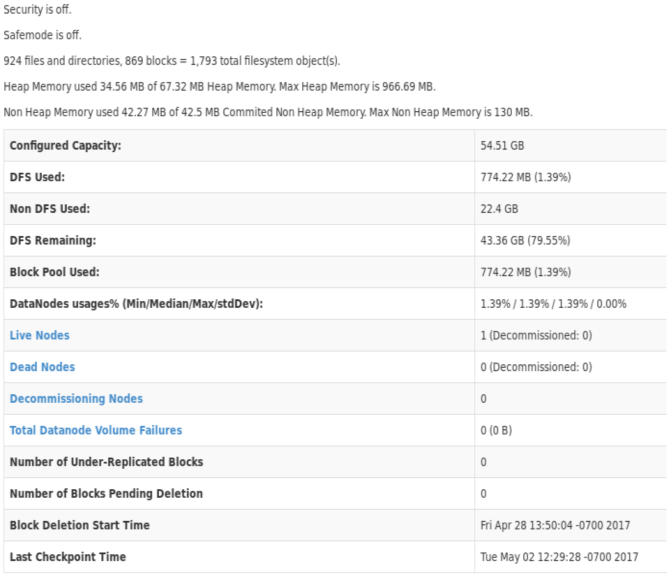
Journal Manager	State
FileJournalManager(root=/var/lib/hadoop-hdfs/cache/hdfs/dfs/name)	EditLogFileOutputStream(/var/lib/hadoop-hdfs/cache/hdfs/dfs/name/current/edits_inprogress_0000000000000013681)

NameNode Storage

Storage Directory	Type	State
/var/lib/hadoop-hdfs/cache/hdfs/dfs/name	IMAGE_AND_EDITS	Active

DFS Storage Types

The console consists of 2 Nodes, one is live node and the other is Node in Service.



Security is off.

Safemode is off.

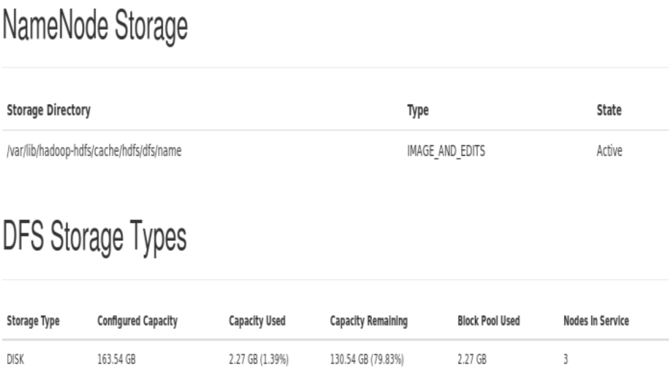
924 files and directories, 869 blocks = 1,793 total filesystem object(s).

Heap Memory used 34.56 MB of 67.32 MB Heap Memory. Max Heap Memory is 966.69 MB.

Non Heap Memory used 42.27 MB of 42.5 MB Commited Non Heap Memory. Max Non Heap Memory is 130 MB.

Configured Capacity:	54.51 GB
DFS Used:	774.22 MB (1.39%)
Non DFS Used:	22.4 GB
DFS Remaining:	43.36 GB (79.55%)
Block Pool Used:	774.22 MB (1.39%)
DataNodes usages% (Min/Median/Max/stdDev):	1.39% / 1.39% / 1.39% / 0.00%
Live Nodes	1 (Decommissioned: 0)
Dead Nodes	0 (Decommissioned: 0)
Decommissioning Nodes	0
Total Datanode Volume Failures	0 (0 B)
Number of Under-Replicated Blocks	0
Number of Blocks Pending Deletion	0
Block Deletion Start Time	Fri Apr 28 13:50:04 -0700 2017
Last Checkpoint Time	Tue May 02 12:29:28 -0700 2017

We can see that image and edits is Active and the Nodes in Service is 3.



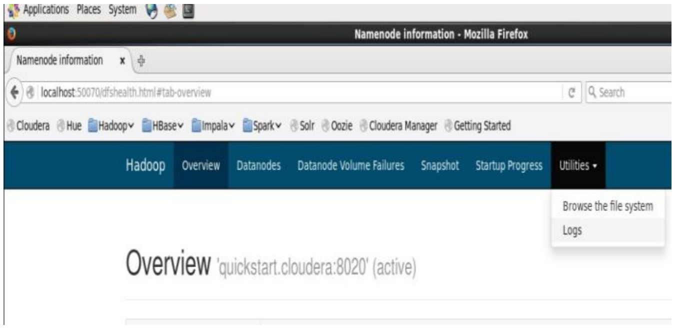
NameNode Storage

Storage Directory	Type	State
/var/lib/hadoop-hdfs/cache/hdfs/dfs/name	IMAGE_AND_EDITS	Active

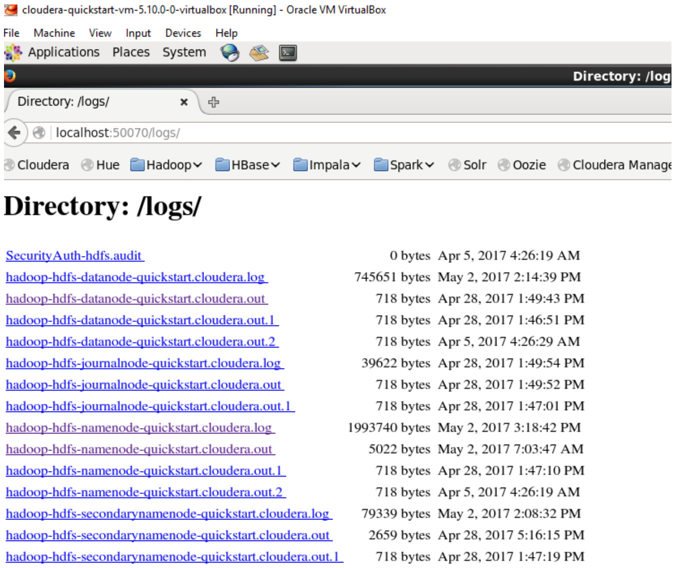
DFS Storage Types

Storage Type	Configured Capacity	Capacity Used	Capacity Remaining	Block Pool Used	Nodes In Service
DISK	163.54 GB	2.27 GB (1.39%)	130.54 GB (79.83%)	2.27 GB	3

Go to the log folder in utilities to see in the directory the files created and saved in the cloud after execution of the project.


Applications Places System
Namenode information - Mozilla Firefox
Namenode information
localhost:50070/dfshealth.html#tab-overview
Search
Cloudera Hue Hadoop HBase Impala Spark Solr Oozie Cloudera Manager Getting Started
Hadoop Overview Datanodes Datanode Volume Failures Snapshot Startup Progress Utilities
Browse the file system
Logs

Overview 'quickstart.cloudera:8020' (active)


cloudera-quickstart-vm-5.10.0-0-virtualbox [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Applications Places System
Directory: /log
Directory: /logs/
localhost:50070/logs/
Cloudera Hue Hadoop HBase Impala Spark Solr Oozie Cloudera Manage

Directory: /logs/

SecurityAuth-hdfs.audit	0 bytes	Apr 5, 2017 4:26:19 AM
hadoop-hdfs-datanode-quickstart.cloudera.log	745651 bytes	May 2, 2017 2:14:39 PM
hadoop-hdfs-datanode-quickstart.cloudera.out	718 bytes	Apr 28, 2017 1:49:43 PM
hadoop-hdfs-datanode-quickstart.cloudera.out.1	718 bytes	Apr 28, 2017 1:46:51 PM
hadoop-hdfs-datanode-quickstart.cloudera.out.2	718 bytes	Apr 5, 2017 4:26:29 AM
hadoop-hdfs-journalnode-quickstart.cloudera.log	39622 bytes	Apr 28, 2017 1:49:54 PM
hadoop-hdfs-journalnode-quickstart.cloudera.out	718 bytes	Apr 28, 2017 1:49:52 PM
hadoop-hdfs-journalnode-quickstart.cloudera.out.1	718 bytes	Apr 28, 2017 1:47:01 PM
hadoop-hdfs-namenode-quickstart.cloudera.log	1993740 bytes	May 2, 2017 3:18:42 PM
hadoop-hdfs-namenode-quickstart.cloudera.out	5022 bytes	May 2, 2017 7:03:47 AM
hadoop-hdfs-namenode-quickstart.cloudera.out.1	718 bytes	Apr 28, 2017 1:47:10 PM
hadoop-hdfs-namenode-quickstart.cloudera.out.2	718 bytes	Apr 5, 2017 4:26:19 AM
hadoop-hdfs-secondarynamenode-quickstart.cloudera.log	79339 bytes	May 2, 2017 2:08:32 PM
hadoop-hdfs-secondarynamenode-quickstart.cloudera.out	2659 bytes	Apr 28, 2017 5:16:15 PM
hadoop-hdfs-secondarynamenode-quickstart.cloudera.out.1	718 bytes	Apr 28, 2017 1:47:19 PM

7 Conclusion

This work demonstrates the pictures of different formats can be given as input. With the proposed algorithm the quality of the picture is fine tuned which had produced better output in the .jpeg format and .pdf format. This output demonstrates that, whatever might be the format of the image, the output can be produced in .jpeg format and .pdf format to give better enhanced quality of image output with less elapsing time and error rate. In this work, the present algorithm has been implemented for the optimizing the result in the map function and reduce function. In future, it is planned to implement some more changes, so that the outcomes can be more exact. This work executes two image conversion formats, where as in the future work, the conversion to other formats can be done using the same technique with few better modifications.

8 Program References

[1] http://stackoverflow.com/questions/10885039/reading-images-from-hdfs-using-mapreduce

[2] http://stackoverflow.com/questions/17425615/reading-large-images-from-hdfs-in-mapreduce

[3] http://stackoverflow.com/questions/16546040/store-images-videos-into-hadoop-hdfs

[4] http://stackoverflow.com/questions/20898073/how-to-convert-sequencefile-in-hadoop-to-image-file-following-code-returns-erro

[5] http://xmodulo.com/convert-jpg-image-file-to-pdf-format-on-linux.html

[6] Create and Execute wordcount in Hadoop, reference for how to execute an Hadoop mapreduce program in virtualbox, https://www.youtube.com/watch?v=VzKGdM4hc74

[7] https://www.youtube.com/watch?v=TWYd0TD8Ops

9 References

[8] https://en.wikipedia.org/wiki/Cloud_computing_security

[9] https://aws.amazon.com/security/introduction-to-cloud-security/

[10] http://searchcloudcomputing.techtarget.com/definition/Hadoop

[11] https://www.tutorialspoint.com/hadoop/hadoop_mapreduce.htm

[12] https://en.wikipedia.org/wiki/Cloudera

[13] https://en.wikipedia.org/wiki/VirtualBox

[14] https://www.virtualbox.org/wiki/VirtualBox

[15] http://www.ijtre.com/images/scripts/2015020914.pdf

[16] https://pdfs.semanticscholar.org/7094/359497f4aa27900266254176496ed94d7984.pdf

[17] http://ccis2k.org/iajit/PDF/Vol.13,%20No.2/7484.pdf

[18] http://www.docsford.com/document/5310154