IJETIE International Journal of Emerging Technology and Innovative Engineering
Volume I, Issue 4, April 2015
ISSN: 2394 - 6598
www.ijetie.org

Image Processing Using Mapreduce With Performance Analysis

Mr. Harish K. Barapatre[1] Mr. Vaibhav Nirgun[2] Mr. Harish Jagtap[3] Mr. Sagar Ginde[4]
Dept. Of Computer Engineering, **University of Mumbai [1234]**
Yadavrao Tasgaonkar Institute of Engineering and Technology[1234]
harishkbarapatre@gmail.com [1] vaibhav.myke1811@gmail.com [2]
jagtapharish8@gmail.com [3] gindesagar@gmail.com [4]

***Abstract*— The traditional approach to transcoding multimedia data requires specific and expensive hardware because of the high-capacity and high definition features of multimedia data and transmoding imposes a considerable burden on the computing infrastructure as the amount of data increases. The proposed module is based on Hadoop HDFS and the MapReduce framework for distributed parallel processing of image database and JAI library for converting the image database into target format and resizing the images also we convert the resizing images into grey scale format. In addition we evaluate the proposed module in terms of processing time under varying experimental conditions on windows platform in cloud environment.**

Index Terms—Hadoop; MapReduce; JAI; HIPI; HDFS; Cygwin;

INTRODUCTION

MDCM is one of the application modules which stores, process and converts the image data into target format using HIPI framework (Hadoop Image processing Interface) and it reduces the size of image data. It also provides the grey scale image using JAI library has been investigated in some studied[18].The amount of images being uploaded to the internet is rapidly increasing, with Facebook users uploading over 2.5 billion new photo severy months [Facebook 2010], however, applications that make use of this data are severely lacking. Current computer vision applications use a small number of input images because there is difficulty in acquiring computational resources and less storage options for large amounts of data [White et al. 2010]. The Hadoop MapReduce platform provides a system for large and computational intensive distributed processing (Dean, 2004), though use of Hadoop system is severely limited by the technical complexities of developing useful applications [White et al. 2010]. To immediately address this, we propose an open-source Hadoop Image Processing Interface (HIPI) that aims to create an interface for computer vision with MapReduce technology. HIPI abstracts the highly technical details of Hadoop's system and is flexible enough to implement much technique in current computer vision literature. Users access multimedia system objects not solely from ancient desktops however additionally from mobile devices, like smart phones and i-pads, whose resources are limited in terms of process, storage, and show capabilities.

Multimedia system process is characterised by large amounts of knowledge, requiring large amounts of process, storage, and communication resources, thereby imposing a substantial burden on the computing infrastructure [1] [18]. The standard approach for transcoding multimedia system needs specific and costly hardware attributable to the high-capacity and high definition options of multimedia system knowledge. Therefore, general purpose devices and ways aren't price effective, and that they have limitations. Recently, transcoding supported cloud computing has been investigated in some studies [1] [2] [3] [18].During this study, we have a tendency to style and implement a multimedia data conversion module supporting MapReduce and HDFS (Hadoop distributed file system) so as to handle the issues mentioned on top of the planned module, it consists of two components. The primary half stores an outsized quantity of image knowledge into HDFS for distributed multiprocessing. The second half processes the hold on image knowledge in HDFS victimisation the MapReduce framework and Java Advanced Imaging (JAI) for changing image knowledge into target formats. We have used the Sequence Files methodology to handle the matter of processing tiny files within the Map operation.

We perform two experiments to demonstrate the proposed module's excellence in without Hadoop and with Hadoop. In the first experiment, we compare the proposed module with a non- Hadoop-based single program running on two different machines. In addition, we verify processing time under different data sizes with non Hadoop and Hadoop based system of the proposed module according to the data sizes of images.

The remainder of this paper is organized as follows. In section 2, we introduce Hadoop HDFS, MapReduce, and JAI. The module architecture and its features are proposed in section 3. In section 4, we describe the implementation module and in section 5 we describe the configuration of Hadoop on windows. The results of the evaluation are presented in section 6. Finally, section 7 concludes this paper with suggestions for future research and Related Work.

International Journal of Emerging Technology and Innovative Engineering
Volume I, Issue 4, April 2015
ISSN: 2394 - 6598
www.ijetie.org

HDFS, MAPREDUCE, JAI, HIPI

A. HDFS

Hadoop is an open source frame work for running applications on large cluster built of commodity hardware. The Hadoop frame work transparently provides applications both reliability and data motion. Hadoop implements a computational paradigm named map/reduce where the application is divided into many small fragments of work, each of which may be executed or re-executed on any node in the cluster. In addition, it provides a distributed file system (HDFS) that stores data on the compute nodes, providing very high aggregate bandwidth across the cluster. Both maps reduce and the Hadoop distributed file systems are designed so that node failures are automatically handled by the frame work.

B. MAPREDUCE

MapReduce frameworks provide a specific programming model and a run-time system for processing and creating large amounts of datasets which is amenable to various real-world tasks [8]. MapReduce framework also handles automatic scheduling, communication, synchronization for processing huge datasets and it has the ability related with fault tolerance. MapReduce programming model is executed in two main steps, called mapping and reducing. Mapping and reducing are defined by mapper and reducer functions that are data processing functions. Each phase has a list of key and value spairs as input and output. In the mapping, MapReduce input datasets and then feeds each data element to the mapper as a form of key and value pairs. In the reducing, all the outputs from the mapper are processed and a final result is created by reducer with merging process. The MapReduce programming model will be actively working with this distributed file system.

C. JAI

JAI is an open-source Java Advanced Image Processing library used for Hipi Image Processing. In JAI Library includes Mapping Function, Reduced Function, and Scaling Function which give the output in target format. In the paper we include JAI library on windows platform also we import many JAR files that support the eclipse and Hadoop plug-in. JAI is an open-source Java library used for image processing [7]. JAI supports various image formats (BMP, JPEG, PNG, PNM, and TIFF) and encoder/decoder functions. In addition, most of the functions related with image conversion are provided through an API, and thus, JAI can be used as a simple framework for image processing.

D. HIPI (Hadoop Image Processing Interface)

HIPI was created to empower researchers and present them with a capable tool that would enable research involving image processing and vision to be performed extremely easily. We modified HIPI with the following goals in mind.

1. Provide an open, extendible library for image processing and computer vision applications in a MapReduce framework.
2. Store images efficiently for use in MapReduce applications.
3. Allow simple filtering of a set of images.
4. Present users with an intuitive interface for image-based operations and hide the details of the MapReduce framework
5. HIPI will set up applications so that they are highly parallelized and balanced so that users do not have to worry about such details.

I. IMAGE PROCESSING MODULE ARCHITECTURE

In this study, we designed and implemented a MapReduce base Image Conversion module in a cloud-computing environment to solve the problem of computing infrastructure overhead. Such overhead increases the burden on the Internet infrastructure owing to the increase in multimedia data shared through the Internet. The traditional approach of transcoding Image Conversion usually involves general-purpose devices and offline-based processes. However, processing is time consuming and requires large computing resources. To solve this problem, we designed Image conversing module that exploits the advantages of cloud computing. The proposed module can resize and convert images in a distributed and parallel manner.

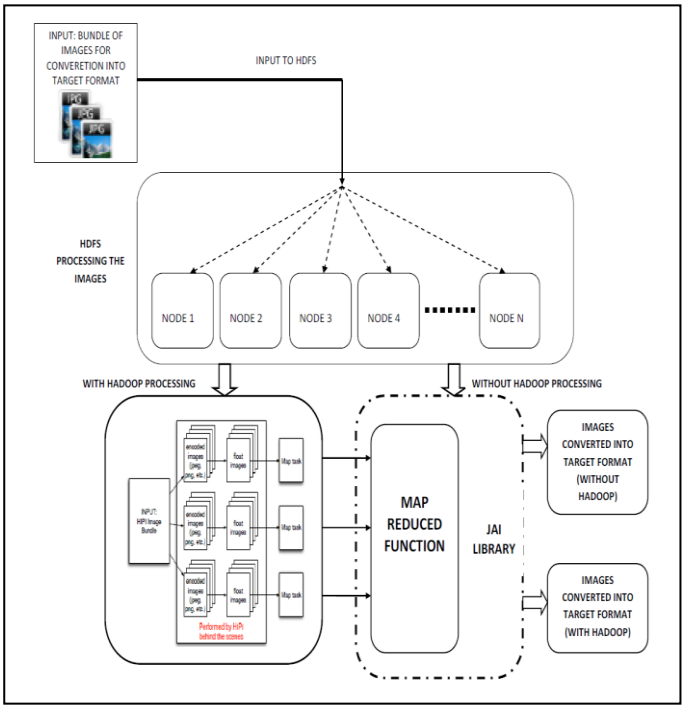

FIGURE-1: IMAGE PROCESSING SYSTEM ARCHITECTURE

IJETIE International Journal of Emerging Technology and Innovative Engineering
Volume I, Issue 4, April 2015
ISSN: 2394 - 6598
www.ijetie.org

The proposed module use HDFS as storage for distributed parallel processing. The image data is distributed in HDFS. For distributed parallel processing, the proposed module uses the Hadoop MapReduce framework. In addition, the proposed module uses the JAI library in Mapper for image resizing and conversion. Figure 1 shows the proposed module architecture.
The proposed module stores image data into HDFS. HDFS automatically distributes the image data to each data node. HIPI framework encoded the images and converted into floating images The Map function processes each float image data in a distributed and parallel manner. The proposed module does not have a summary or construction stage. Thus, there is no need to implement the Reduce function in the proposed module only the Map function is implemented.

II. IMPLEMENTATION OF IMAGE PROCESSING MODULE

Step 1: Download the set of images and converted into float Image bundle using HIPI (input) and initialize the processing start time.
Step 2: The Conversion module reads float value of image data from HIPI using the Record Reader method of the class input Format.
Step 3: Process the image data on HDFS and input format of mapper transforms the image data into sets of Keys (file names) and Values (bytes). Input format passes the sets of Keys and Values to the Mapper.
Step 4: The Mapper processes the image data using the user defined settings and methods for image conversion through JAI library.
Step 5: The conversion module converts the image data into specific formats suitable for a variety of devices such as smart phones, pads and personal computers in a fully distributed manner.
Step 6: The Mapper completes the image into target and passes the results to output format as Key (file name) and Value (byte).
Step 7: The Mapper passes the set of Key and Value to output format. The Record Writer method of the Output Format class writes the result as a file to HDFS.
Step 8: Record Writer method writes the data into the target format also calculate processing time.
Step 9: Calculate the total Processing time with Hadoop.
In this study, the image conversion module was implemented on the basis of Hadoop. However, small chunked files bring problems for the Hadoop MapReduce process. Map tasks usually process a single block of input data at each time instant. If there are many small files, then each Map task processes only a small amount of input data, and as a result, there are many unscheduled Map tasks, each of which imposes extra bookkeeping overhead. Consider a 1-GB file, broken into 16 64-MB blocks, and approximately 10,000 100-KB files. The 10,000 files may require tens or hundreds of times more processing time than an equivalent single-input file.

III. INSTALLING A HADOOP ON WINDOWS

First of all installed JAVA and Eclipse i.e. prerequisite software. After installing the prerequisite software, the next step is to install the Cygwin environment. Cygwin is a set of UNIX packages imported to Microsoft Windows. It is needed to run the scripts supplied with Hadoop because they are all written for the UNIX platform.
To install the Cygwin environment follow these steps:
Download Cygwin installer from http://www.cygwin.com.

1. Run the downloaded file.
2. Keep pressing the 'Next' button until you see the package selection screen. Make sure you select 'openssh'. This package is required for the correct functioning of the Hadoop cluster and Eclipse plug-in.
3. After you selected these packages press the 'Next' button to complete the installation.
4. Set Environment Variables
5. Setup SSH daemon
6. Configure ssh daemon

Open the Cygwin command prompt. Execute the following command:

```
Ssh-host-config
```

When asked if privilege separation should be used, answer ***no***.
When asked if sshd should be installed as a service, answer ***yes.***
When asked about the value of CYGWIN environment variable, enter ***ntsec***.
Start SSH daemon

6.1 Find My Computer icon either on your desktop or in the start-up menu, right-click on it and select **Manage** from the context menu.
6.2 Open **Services and Applications** in the left-hand panel then select the Services item.
6.3 Find the **CYGWIN sshd** item in the main section and right-click on it.
6.4 Select Start from the context menu.

A small window should pop-up indicating the progress of the service start-up. After that window disappears the status of CYGWIN sshd service should change to **Started**. The next step is to Setup authorization key.

7. Download the Hadoop 0.19.1 and unpack the packages and configure the Hadoop with eclipse.
8. **Start the local hadoop cluster**

IJETIE International Journal of Emerging Technology and Innovative Engineering
Volume I, Issue 4, April 2015
ISSN: 2394 - 6598
www.ijetie.org

9. Next step is to launch the newly configured cluster. Close all the windows on the desktop open five Cygwin windows and arrange them in proper sequence as shown in figure-2

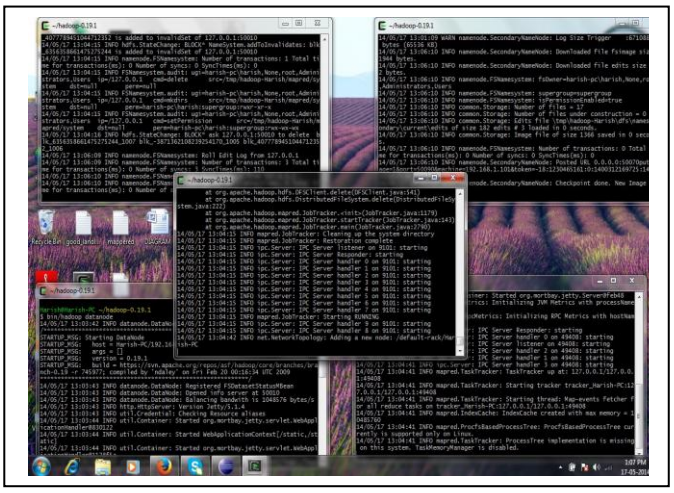

FIGURE-2 LOCAL HADOOP CLUSTER

Start the **namenode** in the first window by executing:
Cd hadoop-0.19.1
bin/hadoop namenode
Start the **secondary namenode** in the second window by executing:
Cd hadoop-0.19.1
bin/hadoop secondarynamenode
Start the **job tracker** the third window by executing:
Cd hadoop-0.19.1
bin/haoop jobtracker
Start the **data node** the fourth window by executing:
Cd hadoop-0.19.1
bin/haoop datanode
Start the **task tracker** the fifth window by executing:
cd hadoop-0.19.1
bin/haoop tasktracker
Now we should have an operational hadoop cluster. If everything went fine your screen, the cluster is running and we can proceed to the next step.

Setup Hadoop Location in Eclipse

Next step is to configure Hadoop location in the Eclipse environment. Launch the **Eclipse** environment. Open Map/Reduce perspective by clicking on the open perspective on ( ), select "Other" from the menu, and then select "**Map/Reduce**" from the list of perspectives. After switching to the *Map/Reduce* perspective, select the **Map/Reduce Locations** tab located at the bottom of the Eclipse environment. Then right click on the blank space in that tab and select **"New Hadoop location...."** from the context menu.

We should see a dialog box i.e. define hadoop location. Fill in the following items, as shown on the figure above.

- Location Name – **localhost**
  - Map/Reduce Master
  - Host – **localhost**
  - Port – **9101**

DFS Master
Check "Use M/R Master Host"
Port – **9100**
User name – **User**

Then press the **Finish** button. After closing the Hadoop location settings dialog we should see a new location in the **"Map/Reduce Locations"** tab. In the **Project Explorer** tab on the left hand side of the Eclipse window, find the **DFS Locations** item. Open it using the "+" icon on its left. Inside, we should see the **localhost** location reference with the blue elephant icon. Keep opening the items below it until we see local host. We can now move on to the next step.

Upload data to HDFS
We are now ready to run the first Map/Reduce project but data is still missing. This section explains how to upload data to the Hadoop Distributed File System (HDFS). Upload Files to HDFS Open a new CYGWIN command window. Execute the following commands in the new CYGWIN window as shown on the figure-2

cd hadoop-0.19.1
bin/hadoop fs -mkdir In
bin/hadoop fs -put *.txt In

When the last of the above commands starts executing, we should see some activity in other Hadoop windows. The result of these commands is a newly created directory -- named **In** -- in the HDFS which contains a set of text files that comes with the Hadoop distribution. Close the Cygwin Window. Verify if the files were uploaded correctly In this section we will check if the files were uploaded correctly.

1. Open the Eclipse environment.
2. Open **DFS locations** folder which is located in the Project Explorer tab of **Map/Reduce** perspective.
3. Open **localhost** folder in **DFS locations** folder.
4. Keep opening HDFS folders until we navigate to the newly created **In** directory
5. When we get to the **in** directory, double-click on the file **LICENCE.txt** to open it.
6. If we see something then the data was uploaded correctly and we can proceed to your Hadoop project.
7. We can now move on to the next step. For Creating and run Hadoop project

International Journal of Emerging Technology and Innovative Engineering
Volume I, Issue 4, April 2015
ISSN: 2394 - 6598
www.ijetie.org

WORKING OF SYSTEM MODULE

Download the multiple images from internet and stored into HDFS for processing images in a distributed manner.

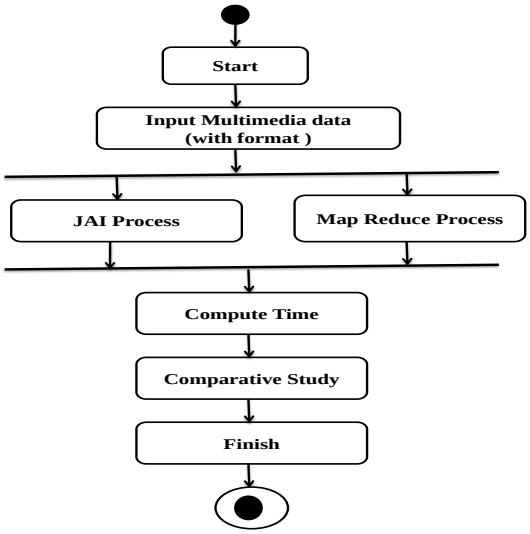

FIGURE-3 WORKING OF DATA CONVERSION MODULE

HIPI interface using the HIPI Image Bundle data type as inputs, we have created an input specification that will distribute images in the HIPI Image Bundle across all map nodes. We distribute images such that we attempt maximize locality between the mapper machines and the machine where the image resides. And a user would have to create InputFormat and RecordReader classes using JAI library. Mapper function conver the image data into target format as shown in figure-3 and compute the processing time with mapper function and JAI library.

IV. EVALUATION

The privileged server used in the experiments for evaluation is a single enterprise scale cluster that consists of 20 computational nodes. Table 1 lists the specifications of the evaluation cluster. Because the structure of the cluster is homogeneous, it provides a uniform evaluation environment.

TABLE I. EVALUATION CLUSTER SPECIFICATIONS

CPU	Intel Xeon 4 Core DP E5506 2.13GHz * 2EA
RAM	4GB Registered ECC DDR * 4EA
HDD	1TB SATA-2 7,200 RPM
OS	WINDOWS-7
Hadoop	0.19.1
JAVA	1.6.1_23
JAI	Java Advanced Imaging JAI 1.1.3
CYGWINE	UNIX TERMINAL
ECLIPSE	ECLIPSE-Europe

Nine data sets were used to verify the performance of the proposed module. The average size of an image files was approximately 19.8 MB. Table II lists the specific information about the data sets used.

TABLE-II IMAGE DATASETS

SIZE (GB)	1	2	4	8	10	20	40
FORMAT	JPG/PNG/TIFF/GIF						
SOURCE	FLICKER						

During the experiment, the following default options in Hadoop were used. The number of block replications was set to 3, and the block size was set to 64 MB. We evaluated the processing Time of the proposed module and optimized it. We planned and executed the experiments. In the First experiment, we calculate the processing time of proposed module with a non-Hadoop-based system and the second experiment we calculate the processing time with Hadoop system. We measured each running time taken in our server using only sequential programing using JAI libraries without MapReduce, respectively. Figure 4 shows the result of the first Experiment with Hadoop and calculate the processing time and resizing the images also converted into the target format.
Figure 5 shows the result of the second Experiment with non-Hadoop based system and calculate the processing time and resizing the images also converted into the target format.
We measured each running time taken in privileged server on windows platform applying only sequential programing using

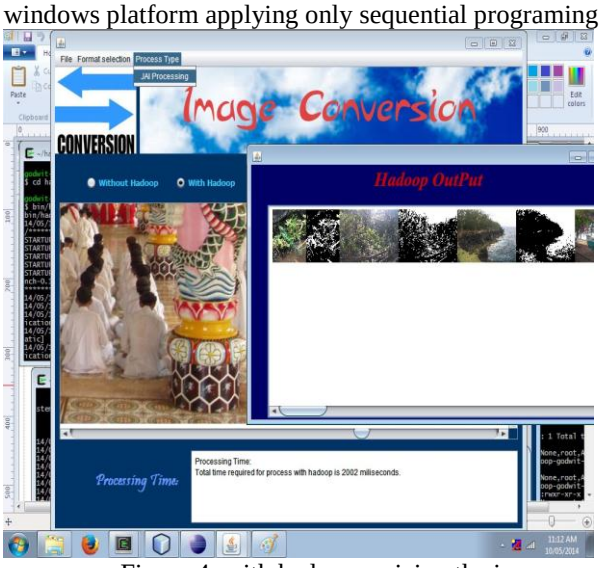

Figure 4: with hadoop resizing the image

IJETIE
International Journal of Emerging Technology and Innovative Engineering
Volume I, Issue 4, April 2015
ISSN: 2394 - 6598
www.ijetie.org

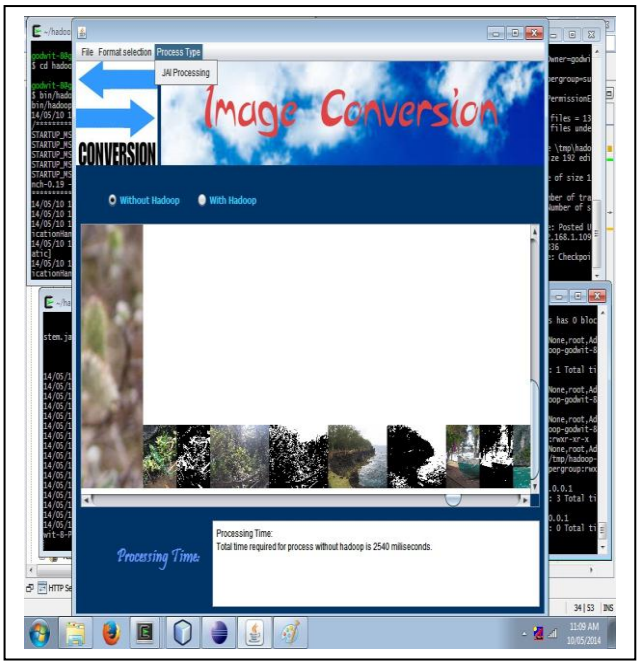

FIGURE 5: NON HADOOP BASED SYSTEM

JAI libraries in without Hadoop and Second experiment we measured processing time applying only JAI with MapReduce Figure 6 shows the result of implementation module

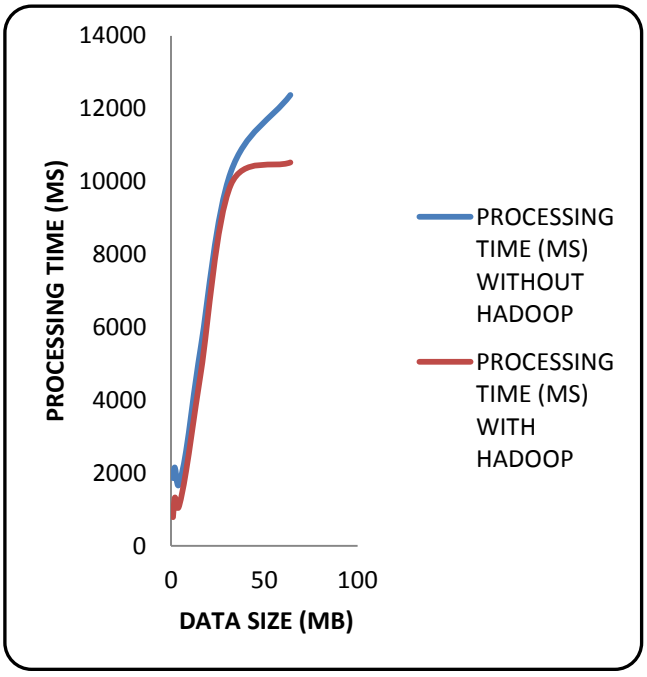

FIGURE 6: EVALUATED PROCESSING TIME FOR WITH HADOOP AND WITHOUT HADOOP

We provided the following data as input and converted into the target format as shown in table III data specification after the execution the system module also we have studied and ploted the graph as shown in figure-6.

TABLE III. INPUT OUTPUT SPECIFICATION

DATA SIZE (MB)	TOTAL NO. IMAGES	SOURCE FORMAT	DESTINATION FORMAT
1	29	JPEG	PNG
2	52	JPEG	BMP
4	21	JPEG	JIF
8	22	JPEG	PNG
16	97	JPEG	JIF
32	182	JPEG	BMP
64	224	JPEG	JPEG

The processing time for the sequential processing without Hadoop always required large as compared to MapReduce when the data is too large. However, after 180 files, the difference between the performances of the two experiments grows when the number of processing files exceeds a certain level; the task of creating Map generates JVM overhead. The option of reusing JVM is a possible solution to reduce overhead created by processing numerous small files on HDFS, as can be seen in the results presented above also we reduce the Burdon of computing power because we processed the data without any hardware, the whole system are designed in JAVA and JAI library using HDFS.

V. CONCLUSIONS & FUTURE WORK

The proposed module is based on Hadoop HDFS and the MapReduce framework on windows for distributed parallel processing of large-scale image data. We redesigned and implemented InputFormat and OutputFormat in the MapReduce framework for image data. We used the JAI library for converting the image format and resizing the images. We exploited the advantages of cloud computing to handle IMAGE processing. We performed two systems for calculating the processing time. In the first experiment, we compared the proposed module with a non-Hadoop-based single program using the JAI library. The proposed module shows better result than the single program. In the second experiment, we changed the *mapred.job.reuse.jvm.num.task* option in the mapred-site.xml file, and we evaluated its processing time. The results of the both experiment show that when the proposed module processes large numbers of small files, it gives the better processing time as compared to non Hadoop based system also it converted the source image into

the target format (JPEG, PNG, BMP, and TIFF). The Planer image method of JAI-Mapper provided the output images as resizing and greyscale format.

Future research should focus not only on image data but also on video data. We plan to implement an integrated multimedia process system and a multimedia share system for SNS in a cloud-computing environment.

ACKNOWLEDGMENT

We would like to give particular thanks to ME guide PROF. ANIL CHHANGANI for his guidance and mentoring throughout this project. His leadership and vision have been excellent models and points of learning for us throughout this process.

REFERENCES

[1]Lee Hyeokju, Kim Myoungjin, Joon Her, and Hanku Lee, Division of Internet & Multimedia Engineering, Konkuk University Seoul Korea in 2011 Ninth IEEE International Conference on "Dependable, Autonomic and Secure Computing"

[2] Lee Hyeokju, Kim Myoungjin, Joon Her, and Hanku Lee, Division of Internet & Multimedia Engineering, Konkuk University Seoul Korea in IEEE ICOIN 2012 "Multimedia cloud environment"

[3]Sun-Moo Kang, Bu-Ihl Kim, Hyun-Sok Lee, Young-so Cho, Jae-Sup Lee, Byeong-Nam Yoon, "A study on a public multimedia sevice provisioning architecture for enterprise networks", Network Operations and Management Symposium, 1998, NOMS 98., IEEE, 15-20 Feb 1998, 44-48 vol.1, ISBN : 0 -7803-4351-4

[4] Wang Hanli,YunShen, Wang Lei, ZhufengKuangtian,Wei Wangand Cheng, Key Laboratory of Embedded System and Service Computing, Ministry of Education, "Large-Scale Multimedia Data Mining Using MapReduce Framework", 2012 IEEE 4th International Conference on Cloud Computing Technology and Science.

[5] B. He and N. K. Govindaraju, "Mars: A MapReduce framework on graphics processors", in PACT'08, 2008, pp. 260- 269.

[6] Y. Shan, B. Wang, J. Yan, Y. Wang, N. Xu, and H. Yang, "FPMR: MapReduce framework on FPGA", in FPGA'10, 2010, pp. 93-102.

[7] S. Papadimitriou and J. Sun, "DisCo: distributed co-clustering with Map-Reduce", in IEEE ICDM'08, 2008, pp. 512-521.

[8] Y. Li, D. J. Crandall, and D. P. Huttenlocher, "Landmark classification in large-scale image collections", in IEEE ICCV'09, 2009, pp. 1957-1964.

[9] L. Kennedy, M. Slaney, and K. Weinberger, "Reliable tags using image similarity: mining specificity and expertise from large-scale multimedia databases", in WSMC'09, 2009, pp. 17-24.

[10] R. Yan, M. O. Fleury, M. Merier, A. Natsev, and J. R. Smith, "Large-scale multimedia semantic concept modeling using robust subspace bagging and MapReduce", in LS-MMRM'09, 2009, pp. 35-42.

[11] B. White, T. Yeh, J. Lin, and L. Davis, "Web-scale computer vision uses MapReduce for multimedia data mining", in MDMKDD'10, 2010, article No. 9.

[12] Hyeokju Lee, Myoungjin Kim, Joon Her, and Hanku Lee, 2011 Ninth IEEE international Conference on Dependable, Autonomic and Secure Computing on "Performance Evaluation of Image Conversion Module Based on MapReduce for Transco ding and Transmoding in SMCCSE"

[13] S. Ghemawat, H. Gobioff and S.T. Leung, "The Google file system," Operating Systems Review (ACM), vol.37, no.5, pp.29 -43, Oct. 20030.

[14] S. Ghemawat, H. Gobioff and S.T. Leung, "The Google file system," Operating Systems Review (ACM), vol.37, no.5, pp.29 -43, Oct. 20030.

[15] http://www.cloudera.com/blog/2009/02/the-small-files-problem/

[16] Hadoop Distributed File System: hadoop.apache.org/hdfs/

[17] Jeffrey Dean, Sanjay Ghemawat, "MapReduce : Simplified Data Processing on large Cluster", OSDI`04 : Sixth Symposium on Operating System Design and Implementation, San Francisco, CA, December, 2004.

[18] Java Advanced Imaging Library: java.sun.com/javase/technologies/desktop/media/jai/