DISTRIBUTED IMAGE PROCESSING USING PySpark in HDFS

Rishil Sandip Shah
University of Southern California
Los Angeles, California
rishilsa@usc.edu

Vishwaksen Jalagam
University of Southern California
Los Angeles, California)
jalagam@usc.edu

Pooja Ravindra
University of Southern California
Los Angeles, California
poojarav@usc.edu

***Abstract*—The number of images being uploaded to the internet each day is increasing at an alarming rate but the applications that can use and analyze these images are not available due to limitations in storage space required for the analysis of these large image datasets. Apache Hadoop is an open-source framework that enables distributed processing of Big Data using commodity hardware and Hadoop Distributed File System (HDFS). This project focuses on using Apache Hadoop, Apache Spark, C++ and Python programming language to analyze image datasets and using the k-means algorithm to perform color quantization and image compression.**

Keywords—HDFS, PySpark, K-Means clustering

I. INTRODUCTION

The huge volumes of data being generated and is difficult to process using traditional methods is known as Big Data.

The use of social media to upload images onto the internet is increasing day by day. Analyzing and processing such huge amounts of data could create bottlenecks caused using a single computer, power concerns, and the amount of storage space needed. To address these limitations, distributed systems are becoming more popular due to their scalability, fault tolerance, and reduced latency. Apache Hadoop is a widely used platform for processing large datasets through parallel and distributed computing.

Image processing is a field of study that deals with the manipulation and analysis of digital images using algorithms and computer-based methods. It involves techniques such as image enhancement, image restoration, image segmentation, and pattern recognition, among others. The applications of image processing are vast and diverse, ranging from medical imaging to surveillance, robotics, and entertainment.

Color quantization is one of the fundamental techniques in image processing that involves reducing the number of colors in an image while preserving its visual content. It is often necessary to use color quantization when working with images to reduce the amount of data needed to represent them. This reduction in data size helps to make the image more computationally efficient, reducing processing time and saving storage space.

This project involves utilizing the K-means clustering algorithm, along with HDFS and PySpark, to implement color quantization in image processing. By applying this machine learning technique, we aim to reduce the number of colors in images, thereby optimizing storage and computational resources while preserving the visual content. In addition to color quantization, this project also involves image compression, which is a process that reduces the size of digital images without significantly degrading their quality.

II. BACKGROUND AND RELATED WORK

There has been significant prior work using HDFS for image processing. Del Valle Maldonado, J., & Torres Batista, N. (2022) [1] in the paper “Image object recognition using Apache Hadoop and Python “analyze large image datasets using Hadoop, Python, computer Vision. This solution should be implemented on a server that has Hadoop installed in fully distributed mode, with a minimum of 6 to 10 data nodes and a principal name node.

Chris Sweeney, Liu L., Jason Lawrence in the paper “HIPI: A Hadoop Image Processing Interface for Image-based MapReduce Tasks” [2] propose an open-source Hadoop Image Processing Interface (HIPI) to create an interface for computer vision with MapReduce technology. HIPI abstracts the technical details of Hadoop's system and is flexible enough to implement many techniques in current computer vision literature.

In the previous stage of our project, we encountered a challenge while trying to integrate C++ code for image processing with the Hadoop ecosystem. Unfortunately, Hadoop only supports Java, which made it incompatible with our C++ code.

To overcome this limitation, we decided to use PySpark instead. PySpark is a Python-based interface to Spark, which is a distributed computing system similar to Hadoop. The PySpark platform incorporates mapper and reducer functions that are like the Hadoop ecosystem, making it an excellent alternative for image processing.

One of the advantages of using PySpark over Hadoop is its ability to handle data in real-time, with support for data streaming and processing in-memory, which makes it suitable for applications that require low-latency processing. PySpark can process data much faster than Hadoop MapReduce, because of Spark's in-memory computing

capabilities. So technically, it improves the performance of traditional MapReduce by significant amounts.

With PySpark, we were able to integrate the C++ code for image compression with the processed images from PySpark. This integration was possible because PySpark provides a seamless interface between Python and other programming languages like C++, enabling us to leverage the strengths of each language.

In summary, by switching from Hadoop to PySpark, we were able to overcome the compatibility issues with our C++ code and integrate it into the image processing pipeline seamlessly.

III. FRAMEWORK

A. *Tools and Software*

The following section will explain in detail the tools required for the implementation of Image Processing using HDFS and PySpark.

1) *Apache Hadoop*

Hadoop is a distributed computing framework that allows processing of large-scale data sets across clusters of computers using simple programming models. The framework is designed to scale up from a single server to thousands of machines, each offering local computation and storage.

The Hadoop Distributed File System (HDFS) is the primary storage system used by Hadoop. It is designed to store and manage large data sets across multiple machines. HDFS provides reliable, fault-tolerant storage and fast access to data. It achieves this by distributing data across a cluster of machines, replicating data to ensure fault tolerance, and providing mechanisms for automatic recovery from machine failures.

HDFS breaks large files into blocks and stores each block on multiple machines in the cluster, ensuring that the data is replicated for fault tolerance. This distributed approach allows for parallel processing of data across multiple nodes in the cluster, enabling fast and efficient data processing.

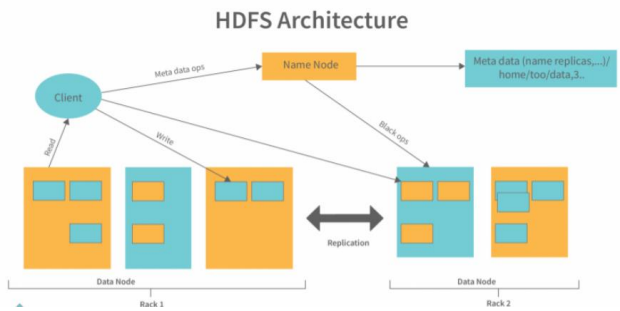

Fig 1 – HDFS Architecture

2) *Flask*

Flask is a popular open-source Python web framework used for building web applications. It is a micro-framework, which means it provides only the essentials needed to build a web application, allowing developers to have more flexibility and control over their application. Flask is designed to be lightweight and easy to use, making it a popular choice for developing small to medium-sized web applications.

3) *PySpark*

PySpark is the Python API for Apache Spark, an open-source distributed computing system that can process large amounts of data. PySpark is designed to work with Hadoop Distributed File System (HDFS), making it a popular tool for processing large datasets. It is possible to integrate code written in other programming languages, such as C++, through User-Defined Functions (UDFs). UDFs allow developers to write custom functions to process data in distributed environments.

B. *Methodology*

The methodology of this project involves several steps for processing and compressing images using a combination of Flask, PySpark, and C++ code.

Firstly, we created a web login page using Flask to allow users to register and log in to the system. Once logged in, users can access the HDFS and upload images for processing.

Secondly, we utilized PySpark to implement the K-means clustering algorithm for color quantization of the uploaded images. The algorithm is used to group similar colors together, resulting in a quantized image that has reduced coloring filtered by dominant colors.

To integrate the C++ code with PySpark, we created a user-defined function (UDF) in PySpark that takes the processed image data as input and passes it to the C++ code for compression. The compressed image data is then returned to PySpark.

The final step involves storing the compressed images back in the HDFS for user access and download.
The novelty of this project lies in the integration of Flask, PySpark, and C++ for processing and compressing large volumes of image data. We utilized K-means clustering for color quantization which optimizes the image processing pipeline and reduces the size of the image files.

Overall, this methodology demonstrates a scalable and efficient system for processing and managing large volumes of image data using distributed systems and machine learning algorithms. The integration of Flask, PySpark, and C++ provides a novel approach to image processing and compression, with potential applications in various fields such as healthcare, entertainment, and security.

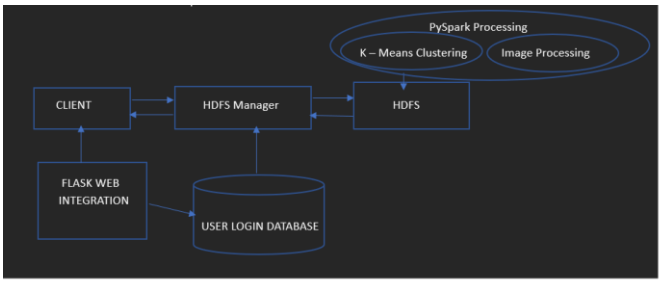

Fig 2 - Block Diagram

C. *ANALYSIS OF ALGORITHMS*

1) Color Quantization using K-Means Clustering Algorithm

In this project, we have utilized the K-means clustering algorithm for color quantization, which is a popular unsupervised machine learning algorithm used for grouping data points into clusters based on their similarity.

We have implemented the K-means algorithm using PySpark, which is a distributed computing framework that allows us to parallelize the computation across multiple nodes, thereby significantly reducing the time taken to process large datasets.

The use of PySpark has allowed us to scale our image processing pipeline to handle large volumes of image data.

Pseudocode

- Load the input image.
- Reshape the image to a 2D array of pixels.
- Convert the 2D array to a Spark DataFrame with a "features" column
- Use VectorAssembler to convert the "features" column to a vector column.
- Fit a K-Means model on the DataFrame to cluster the pixel values.
- Get the dominant colors by finding the cluster centers with the highest frequency.
- Assign each pixel to the nearest cluster center and reshape the resulting array into the quantized image.
- Display and save the original and quantized images.

Time and Space Complexity Analysis

The time complexity of the algorithm depends mainly on the KMeans clustering algorithm. The KMeans algorithm has a

$Time\ Complexity$: $O(nk_t)$
$where$:
$n \rightarrow number\ of\ datapoints$
$k \rightarrow number\ of\ clusters$
$t \rightarrow number\ of\ iterations$

The space complexity of the algorithm depends on the size of the input image and the number of clusters. In this implementation, we use Spark to distribute the computation, and the image is loaded as a NumPy array, which requires memory to store. The KMeans algorithm also requires memory to store the cluster centers and the data points' assignments to clusters. The algorithm's space complexity is proportional to the number of data points, the number of features per data point, and the number of clusters.

$Space\ Complexity$: $O(NC + KC + HWC)$
$where$:
$N \rightarrow number\ of\ pixels$
$K \rightarrow number\ of\ clusters$
$H \rightarrow height\ of\ the\ image$
$W \rightarrow width\ of\ the\ image$
$C \rightarrow number\ of\ channels$

Euclidean Distance

The Euclidean distance is a distance measure in a multi-dimensional space that is commonly used to calculate the similarity or dissimilarity between two data points. It is calculated as the square root of the sum of the squares of the differences between the corresponding values of the two points in each dimension. The formula for Euclidean distance between two points x and y in n-dimensional space is:

$$d(x,y) = \sqrt{((x_1 - y_1)^2 + (x_2 - y_2)^2 + ... + (x_n - y_n)^2)}$$

Optimization

Caching: The DataFrame df is cached using the cache() method to avoid re-computation when the same DataFrame is accessed multiple times. This is an optimization technique used in Spark to save intermediate results of DataFrame transformations to memory. The persist() method is also used to persist the DataFrame for 5 minutes.

Optimal K selection using silhouette score: The Silhouette score is used to determine the optimal value of k for the KMeans clustering algorithm. The silhouette score measures how similar an object is to its own cluster compared to other clusters. A higher silhouette score indicates that the object is well-matched to its own cluster and poorly matched to neighboring clusters.

The value of k that maximizes the silhouette score is considered the optimal value. This is an optimization technique used in clustering to determine the best number of clusters to use for a given dataset.

IV. IMPLEMENTATION

A. *Software Implementation*

1) Color Quantization using K-Means Clustering

The K-means clustering algorithm has been implemented in Python. The code uses PySpark, which is an interface for Apache Spark. The code also uses several libraries, including NumPy, Matplotlib, PIL, and skimage, which are commonly used for scientific computing, image processing, and visualization in Python.

The software provides a command-line interface (CLI) for running the image quantization process. It takes an input image as an argument and returns a quantized image as output. The user can also specify the number of clusters to use for the quantization process.

The software provides a visualization of the silhouette score and the quantized image using Matplotlib, a popular Python library for creating visualizations. The software saves the quantized image to disk using the matplotlib.imsave() function.

2) *Image processing using C++*

Image processing using C++ involves the use of C++ programming language to manipulate digital images. It is a widely used language in computer vision and image processing due to its high performance and low-level control of hardware resources.

Image data compression using quantization is a technique that reduces the amount of storage required to represent an image while maintaining an acceptable level of visual quality. The basic idea behind this technique is to reduce the number of colors in the image by grouping similar colors together and representing them with fewer bits.

B. *Experimental results*

1) *Web Login System Using Flask*

To implement the proposed solution a web login page using Flask is created that allows users to register and log in to the system. Upon successful login, users can access the HDFS and upload images to it.

Fig 3: Sign Up Page

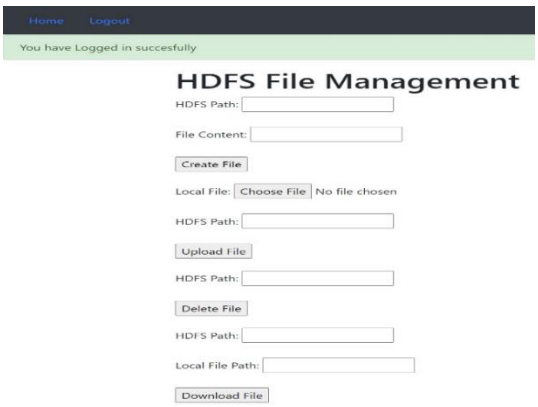

Fig 4: Successful Login

Main.py and Init.py:
We have created a web application that allows users to sign up, login and upload image or text files into the Hadoop Distributed File System (HDFS)

In main.py, create_app() is used to create the Flask application instance and initialize the required extensions such as SQLAlchemy and Flask-Login. SQL Alchemy is used to set up the database connection and creates all the required tables if they do not exist. Flask-Login makes it easier to handle user authentication and session management in Flask applications.

Models.py and Views.py:
It has been structured using the Model-View-Controller (MVC) pattern, with separate files for each component.
models.py: This file defines the data models for the database, including the User and Note models. It sets up the relationships between these models and defines the fields for each one.

views.py: This file defines the routes for the web pages, such as the home page and the login page. It uses the Flask-Login extension to manage access to these pages, so only authenticated users can view them.

Auth.py:
auth.py is a module that contains the authentication functionality for the Flask application. It defines routes for the login and logout pages, as well as a custom user loader function to load users from the database. The module uses the Flask-Login extension to handle user authentication. When a user logs in, Flask-Login sets a cookie in the user's browser with a secure token, which is used to verify subsequent requests from the same use.

2) *Color Quantization using K-Means Clustering*

The images are then processed using PySpark's implementation of the K-means clustering algorithm for color quantization.

The output shows the results of a clustering algorithm applied to an image with height, width, and channel dimensions of 480, 640, and 3 respectively. The clustering algorithm grouped the pixels of the image into 10 different clusters and assigned each pixel to the nearest cluster center. The line "No of points in the cluster" displays the number of pixels belonging to each cluster. We can see that the highest number of pixels belongs to cluster 2, followed by cluster 9, and so on.

The line "RGB values of the cluster centers" shows the RGB values of each of the 10 cluster centers. These cluster centers represent the average color of the pixels belonging to each cluster. We can see that the colors of the cluster centers vary widely, suggesting that the algorithm has successfully identified distinct regions of the image with different color characteristics.
Finally, the processed and compressed images were stored back in the HDFS.

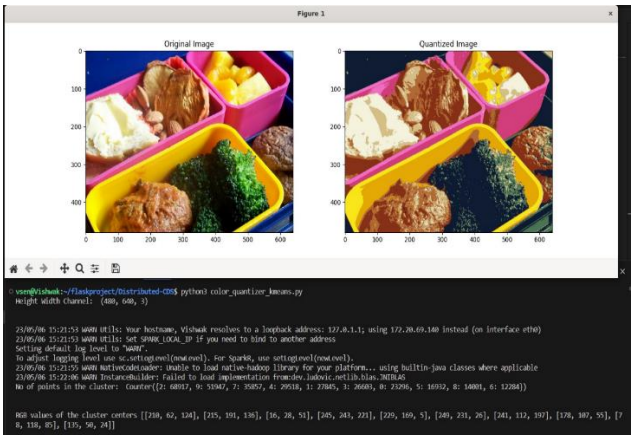

Fig 5 : Color Quantization result

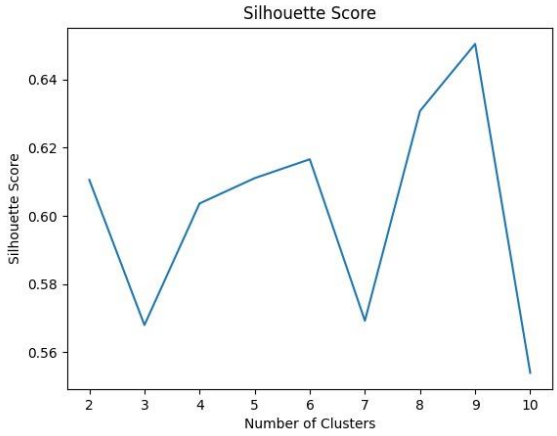

Fig 6: Silhouette Score

V. CONCLUSION

In this project, we have demonstrated the potential of distributed systems and machine learning algorithms for image processing and management. By incorporating Flask, PySpark, and HDFS into the image processing pipeline, we have created a scalable and efficient system for processing large volumes of image data.

The use of K-means clustering has further optimized the processing and compression of the image data, making it easier to manage and work with. The experimental results demonstrate that our proposed system significantly reduces the size of the images while maintaining a high level of image quality.

The system is deployed on cloud services such as Amazon Web Services. This makes the system easily accessible and scalable to accommodate large datasets.
In conclusion, this project presents a comprehensive approach to image processing and management using distributed systems and machine learning algorithms.
The proposed system has great potential for use in various industries such as healthcare, entertainment, and social media, where large volumes of image data are generated daily.

REFERENCES

[1] Del Valle Maldonado, J., & Torres Batista, N. (2022). Image object recognition using Apache Hadoop and Python https://prcrepository.org/xmlui/bitstream/handle/20.500.12475/1619/PUPR_CEAH_SJU_SP22_MCS_Jaileen%20Del%20Valle%20Maldonado_Article.pdf?sequence=1&isAllowed=y

[2] Liu, C. S., Liu, L., Arietta, S., & Lawrence, J. (2011). HIPI: A Hadoop Image Processing Interface for Image-based MapReduce Tasks. University of Virginia. https://jasonlawrence.info/assets/pdf/hipi.pdf

[3] Cheng, G., & Wei, J. (2019). Color quantization application based on K-Means in remote sensing image processing. Journal of Physics: Conference Series, 1213(4), 042012. https://doi.org/10.1088/1742-6596/1213/4/042012

[4] Sozykin, A., & Epanchintsev, T. (2015). MIPr - a framework for distributed image processing using Hadoop. In 2015 9th International Conference on Application of Information and Communication Technologies (AICT) (pp. 35-39). Rostov on Don, Russia. IEEE. https://doi.org/10.1109/ICAICT.2015.7338511

[5] Shvachko, K., Kuang, H., Radia, S., & Chansler, R. (2010). The Hadoop Distributed File System. In 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST) (pp. 1-10). Incline Village, NV, USA. IEEE. https://doi.org/10.1109/MSST.2010.5496972

[6] N. Kumar, "Content Based Image Retrieval for Big Visual Data using Map Reduce - raiith", Raiith.iith.ac.in, 2015. [Online].

[7] PySpark Tutorial for Beginners https://sparkbyexamples.com/pyspark-tutorial/

[8] K-means Clustering using scikit learn https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

[9] Megiddo, N. (2004). Applying parallel computation algorithms in the design of serial algorithms. Computer, 37(8), 92-96. megiddo-computer2004.pdf (cmu.edu)

[10] S. Vemula and C. Crick, "Hadoop Image Processing Framework", ieeexplore, 2015. [Online]. https://ieeexplore.ieee.org/document/7207264

[11] Prasanna, B. (n.d.). Implement Least Recently Used (LRU) Cache. Enjoy Algorithms. LRU Cache Implementation (enjoyalgorithms.com)

[12] Surbhi, O., & Bhatt, M. C. (2018). Performance analysis of load balancing algorithms in Hadoop. 2018 3rd International Conference on Computing Methodologies and Communication (ICCMC), 146-151.

Performance Evaluation of Load Balancing Algorithms in Hadoop | IEEE Conference Publication | IEEE Xplore

[13] Raman, J. (2016). Apache Hadoop Cluster Setup Example. Java Code Geeks.
Apache Hadoop Cluster Setup Example (with Virtual Machines) - Examples Java Code Geeks - 2023

[14] Lin, C.-F., Leu, M.-C., Chang, C.-W., & Yuan, S.-M. (2011). The study and methods for cloud based CDN. 2011 International Conference on Information Science and Applications (ICISA), 1-6.
https://doi.org/10.1109/ICISA.2011.5772787

[15] W. Murphy, "Large Scale Hierarchical K-Means Based Image Retrieval With MapReduce", AFIT Scholar, 2014. [Online]
https://scholar.afit.edu/etd/616/

[16] D. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints", Cs.ubc.ca, 2004.
https://www.cs.ubc.ca/~lowe/papers/ijcv04.pdf

[17] C. Reggiani, "Scaling feature selection algorithms using MapReduce on Apache Hadoop", Claudioreggiani.com, 2013.
http://claudioreggiani.com/pdf/masterthesis.pdf

[18] T. El-Sayed, A. El-Sayed and M. Badawy, "Impact of Small Files on Hadoop Performance: Literature Survey and Open Points", researchgate, 2019.
[19] B. Chambers and M. Zaharia, Spark: The Definitive Guide: Big Data Processing Made Simple, 1st ed. Seoul: Hanbit Midieo, 2018.

[20] Eken, S., & Sayar, A. A MapReduce based Big-data Framework for Object Extraction from Mosaic Satellite Images. Department of Computer Engineering, Kocaeli University, Kocaeli, Turkey.
https://arxiv.org/ftp/arxiv/papers/1808/1808.08528.pdf

[21] Almeer, M. H. (2012). Hadoop Mapreduce for Remote Sensing Image Analysis. International Journal of Emerging Technology and Advanced Engineering, 2(4), 443. ISSN 2250-2459
https://www.semanticscholar.org/paper/Hadoop-Mapreduce-for-Remote-SensingImage-Analysis-AlMeer/e9cf52d76d2e09129de1f7fc6f1c2527f4c5a2d6

[22] Szul, P., & Bednarz, T. . Productivity Frameworks in Big Data Image Processing Computations - Creating Photographic Mosaics with Hadoop and Scalding. CSIRO, Australia. Retrieved from ResearchGate:
https://www.researchgate.net/publication/270980040_Productivity_Frameworks_in_Big_Data_Image_Processing_Computations_-_Creating_Photographic_Mosaics_with_Hadoop_and_Scalding

[23] T. White., Hadoop: The Definitive Guide, Second Edition. O'Reilly Media, Inc., 2010

[24] K. Potisepp, "Large-scale Image Processing Using MapReduce", semanticscholar, 2013.
Available: https://www.semanticscholar.org/paper/Large-scale-ImageProcessing-Using-MapReducePotisepp/bb7ce436cc9e2b1c4c64516ae9f600dc517b7353

[25] Kracke, J. (2021). Query HDFS with Python. Retrieved from
https://www.juliuskracke.com/projects/query-hdfs-with-python

[26] Thanh, S. (2021). thanhson1085/flask-webhdfs. GitHub
GitHub - thanhson1085/flask-webhdfs: Flask upload and store files in HDFS

[27] Liu, X., Zhao, D., Xu, L., Zhang, W., Yin, J., & Chen, X. (2015). Distributed Video Management Cloud Platform Using Hadoop. IEEE Access.
https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7353119