Distributed Computing and Image Processing for Autonomous Driving Systems

Tejaswa Gavankar
Department of Computer Engineering and Information Technology
College of Engineering Pune (COEP)
Pune, India
tejaswaljg14.comp@coep.ac.in

Aditi Joshi
Department of Computer Engineering and Information Technology
College of Engineering Pune (COEP)
Pune, India
joshiaditi14.comp@coep.ac.in

Shantanu Sharma
Department of Computer Engineering and Information Technology
College of Engineering Pune (COEP)
Pune, India
shantanus14.comp@coep.ac.in

***Abstract*— in an autonomous driving system, the field of view spans multiple cameras placed around a car driven through numerous driving scenarios. Sensor data is received by the analyzing unit at a high velocity, also the camera provides over millions of images for a small drive of about half a mile. Also not all the images captured by the cameras are capable of being analyzed as some of them might have to be discarded on accounts of high noise levels or lack of lighting. A simple example of this is when pictures clicked on burst mode often have more throwaways than the ones which can be utilized. So, it is important for the analyzing unit to make a series of decisions before even starting the feature extraction process. Efficient processing of a high volume of images is therefore a challenge which autonomous systems such as the driving system face. Given the multiple cameras present on autonomous cars, providing high resolution pictures through varying driving scenarios, the objective is to process and analyze this huge dataset efficiently. This paper shall demonstrate the power of distributed computing in image processing algorithms and analysis of incredibly large datasets using a distributed approach. This paper gives a statistical proof of concept of how implementing a distributed parallel programming paradigm can improve autonomous systems such as the driving system which deal with high volumes of images.**

Keywords— *Computer Vision, OpenCV, Image Processing, Distributed Computing.*

I. INTRODUCTION

Driver negligence is one the leading causes of road accidents worldwide. More than 12 million people lose their lives every year globally. Light motor vehicles account for 22 percent of the total road accidents in India alone in 2016. With the ever-increasing number of vehicles, the use of computer guided systems for safety is the way forward.

An automated solution can be achieved through Computer Vision. The field of view spans multiple cameras placed around a car driven through numerous driving scenarios. Through image processing using OpenCV the first series of decisions like lane detection and switching, traffic light and road sign detection will be made. Given the multiple cameras, high resolution, and varying driving scenarios, the objective will also be to process and analyze this huge dataset efficiently. The paper shall demonstrate the power of distributed computing in order to enable efficient preprocessing and then subsequent interpolation through feature extraction and analysis of incredibly large datasets.

II. DATASETS

Datasets Considered:

KITTI Vision Benchmark Suite, Oxford RobotCar Database, comma.ai, University Guanajuato, Udacity, Cityscapes, PeRL.

The sensitivity of Image processing algorithms, it was essential that a dataset is chosen that is shot on an appropriate resolution with the kind of attributes that are to be evaluated. Also the dataset must be labeled for us to test and categorise them through the distributed file system.

Oxford Robotic drove a 10-kilometer route through the central Oxford twice a week from November 2014 to December 2015. Such a vast period of data collection resulted in datasets produced in every possible weather conditions. The cameras mounted on their car provided 20 million of images along with information from LIDAR and GPS sensors.

'comma.ai' decided to release 7.5 hours of driving data along with labels concerning vehicle state. Udacity released 223 gigabytes of data containing frames of 70 minutes videos in total recorded while driving around Mountain View. The advantage of this dataset is its diversity—videos for sunny and overcast weather conditions are included. With every frame comes information about car position, current speed, steering angle, throttle or brake.

Fig. 1. Oxford Robotic's Rear Camera Screen Grab.

A dataset that could be most relevant to India is by Hayet Guzman from *University Guanajuato, Mexico*. They realized that most of available road datasets lack several features like abundant potholes or speed bumpers—elements that are typical of developing countries. During the summer of 2014 they grabbed stereo video and provided it along with GPS and acceleration data. However, this will not be used in the

978-1-5386-5323-4/18/$31.00 ©2018 IEEE

initial stages since the organized system of roads of developed countries are easier for analysis.

Cityscapes, was released in the 2016. It provides around 25 000 photos taken in 50 cities. Every pixel of each photo is annotated to belong to one of 30 defined classes such as roads, sidewalks, vehicles, traffic signs or people. Photos were taken at daylight in different months of the year.

PeRL Ford Campus Vision and Lidar Dataset consists of time-registered data from sensors mounted on the vehicle, collected while driving the vehicle around the Ford Research campus and downtown Dearborn, Michigan during November-December 2009. The vehicle path trajectory in these datasets contains several large and small-scale loop closures, which should be useful for testing Mapping algorithms.

KITTI is the dataset developed by Karlsruhe University of Technology. KITTI is a broadly recognized benchmark for testing different algorithms. Every frame of those videos comes with annotations to car speed, acceleration, GPS position, steering angles. The datasets are captured by driving around the mid-size city of Karlsruhe, in rural areas and on highways. Up to 15 cars and 30 pedestrians are visible per image.

Fig. 2. Screen grab of a car at a traffic signal along with information regarding the particular datset from KITTI Database

Given that KITTI has various external environments, metadata, is open source and has a lot of data available combined with a standard mounted camera view (not fisheye, oblong lenses), this implementation shall be carried out with the KITTI Vision Benchmark Suite as our dataset for distributed computing.

III. RELEVANT TECHNOLOGIES

A. *MapReduce Model and Image Processing*

The MapReduce is a programming model and an associated implementation for processing and generating large data sets. Every problem which is to be solved using the mapreduce paradigm involves of two parts, the map function and the reduce function. The *'Map'* function takes an input pair and produces a key value pair which is served as input to the '*Reduce'* function. Popularized by Google the MapReduce function helped them tackle problems related to web crawling, web request logs and graph structure of webpages.etc.

Mapreduce has found great use in processing text documents which have large amounts of raw data. Social networking sites like Facebook and Twitter have been using Apache Hadoop to implement the MapReduce paradigm to process the huge amounts of user data in an efficient manner. Analysis of structured data such as log files and SQL tables has taken a giant leap with the computationally intensive distributed processing environment and has helped organisation process data which is high in volume and velocity.

Today, the amount of images each individual deals with has increased exponentially. The common use of selfie sticks and GoPro cameras to shoot hours of time lapse videos, 360 degree photographs and burst mode images have made every user deal with large image data sets. Textual data has taken a back seat to store memories and even every day happenings. Technology has done well to catch-up with this shift as images and video footage (rapid frames of images) too have been made valid input data for the MapReduce Paradigm.

Unstructured data like images, videos and audio recordings are also now stored and processed in a distributed fashion. Instagram uses Amazon S3 to store the images. Apache Hadoop has introduced the Hadoop S3A client which offers high performance IO against Amazon S3 object stores. It helps in directly reading S3 objects thus making processing of object storage openly available to all developers.

B. *Hadoop Image Processign Interface (HIPI)*

HIPI lets you efficiently process images on a Hadoop cluster. It's needed because HDFS can't handle large numbers of files, so it provides a way of bundling images together into much bigger files, and unbundling them on the fly as you process them.

HIPI is basically an image processing library designed to be used with the Apache Hadoop MapReduce parallel programming framework. It facilitates efficient and high-throughput image processing with MapReduce style parallel programs typically executed on a cluster. HIPI provides a solution for how to store a large collection of images on the Hadoop Distributed File System (HDFS) and make them available for efficient distributed processing. HIPI also provides integration with OpenCV

The primary input object to a HIPI program is a HipiImageBundle (HIB). A HIB is a collection of images represented as a single file on the HDFS. The first processing stage of a HIPI program is a culling step that allows filtering the images in a HIB based on a variety of user-defined conditions like spatial resolution or criteria related to the image metadata. The images that survive the culling stage are assigned to

individual map tasks in a way that attempts to maximize data locality. Finally, individual images are presented to the Mapper as objects derived from the HipiImage abstract base class along with an associated HipiImageHeader object. These classes provide a number of useful functions like cropping, color space conversion, and scaling. The records emitted by the Mapper are collected and transmitted to the Reducer according to the built-in MapReduce shuffle algorithm that attempts to minimize network traffic. Finally, the user-defined reduce tasks are executed in parallel and their output is aggregated and written to the HDFS.

C. *Image Processing Software*

The implementation of this project is on OpenCV through Hadoop. The only real possible choices were OpenCV and MATLAB, and OpenCV was selected since it is the most comprehensive open source library for computer vision and has more functions for computer vision. In general C++ based OpenCV code runs faster than Matlab code (interpreted since it is Java based) and is also tweakable to improve efficiency. Due to the high level nature of Matlab, it consumes a lot of system resources. Thus OpenCV is more viable for real time constraints in a system like autonomous driving.

IV. HIPI IMPLEMENTATION

The choice of master slave configuration was based on the two important criteria:

- Master nodes should be expensive machines which allow high computation speeds even though they do not provide high amount of storage capabilities, as the main function of the master will be of splitting the data and then again recompiling it once the individual nodes return the results to the partial problems.
- Slave nodes should be cheap machines which have higher storage facilities and not necessarily high computation capabilities, as the main function of the slave nodes will be of providing storage facilities and executing trivial computation processes on the acquired data.

The inexpensive nature of slave machines allow distributed computing environments with high scalability and fault tolerance abilities, as the data can be replicated on more than a single slave machine and thus, making the overall system more reliable.

HIPI lets you efficiently process images on a Hadoop cluster. It's needed because HDFS can't handle large numbers of files, so it provides a way of bundling images together into much bigger files, and unbundling them on the fly as you process them.

HIPI is basically an image processing library designed to be used with the Apache Hadoop MapReduce parallel programming framework. It facilitates efficient and high-throughput image processing with MapReduce style parallel programs typically executed on a cluster. HIPI provides a solution for how to store a large collection of images on the Hadoop Distributed File System (HDFS) and make them available for efficient distributed processing. HIPI also provides integration with OpenCV

The primary input object to a HIPI program is a HipiImageBundle (HIB). A HIB is a collection of images represented as a single file on the HDFS.

The first processing stage of a HIPI program is a culling step that allows filtering the images in a HIB based on a variety of user-defined conditions like spatial resolution or criteria related to the image metadata.

The images that survive the culling stage are assigned to individual map tasks in a way that attempts to maximize data locality. Finally, individual images are presented to the Mapper as objects derived from the HipiImage abstract base class along with an associated HipiImageHeader object. These classes provide a number of useful functions like cropping, color space conversion, and scaling.

The records emitted by the Mapper are collected and transmitted to the Reducer according to the built-in MapReduce shuffle algorithm that attempts to minimize network traffic. Finally, the user-defined reduce tasks are executed in parallel and their output is aggregated and written to the HDFS.

Single Machine Configuration	
Machine	Details
Personal Computer	Intel CORE i5-8250U 1.6GHz 8GB RAM Total storage 500GB OS Ubuntu 17

Fig. 3. Configuration details of a single machine system

Master / Slave configuration	
Machine	Configuration
Master	Intel CORE i5-8250U 1.6GHz 4GB RAM Total storage 500GB
Slave 1	Intel CORE i7-7500U 2.70GHz 8GB RAM Total storage 300GB OS Ubuntu 14
Slave 2	Intel CORE i5-6200U 2.30GHz 4GB RAM Total storage 300GB OS Ubuntu 14
Slave 3	Intel CORE i3-8250U 1.6GHz 8GB RAM Total storage 500GB OS Ubuntu 17

Fig. 4. Configuration details of the cluster

V. RESULTS

To quantify the difference between the traditional and MapReduce implementation of image processing algorithms, the execution times were compared of programs which calculated the average pixel color (RGB) value of multiple images in our data set.

Size	Single Node	Single Machine	MultiNode Cluster
0.5	*1.74*	*0.06*	*1.88*
32	*3.68*	*2.65*	*8.98*
92	*5.69*	*4.19*	*10.33*
243	*13.73*	*10.5*	*18.66*
313	*24.89*	*22.9*	*26.13*
619	*43.3*	*37.5*	*43.24*
1024	*124.22*	*120.23*	*135.13*
5120	*INT_MAX*	*INT_MAX*	*355.12*
10240	*INT_MAX*	*INT_MAX*	*765.21*
15360	*INT_MAX*	*INF*	*1282.44*

Fig. 5. Time Comparison of same process on different configurations.

Firstly, the average running time of the program was tested following the Map reduce paradigm and compared it to the time it took for a simple java program to calculate the average pixel color in the pictures for different sizes of the data set. Both were run on a single machine (limited storage) and the criteria for comparison were the execution times.

Second, as the size of the data set grew and a single machine did not have storage capabilities to store images, the computation was shifted to multi node cluster setup where the storage facilities of 3 machines along with the master machine was harnessed, allowing us to process large data sets.

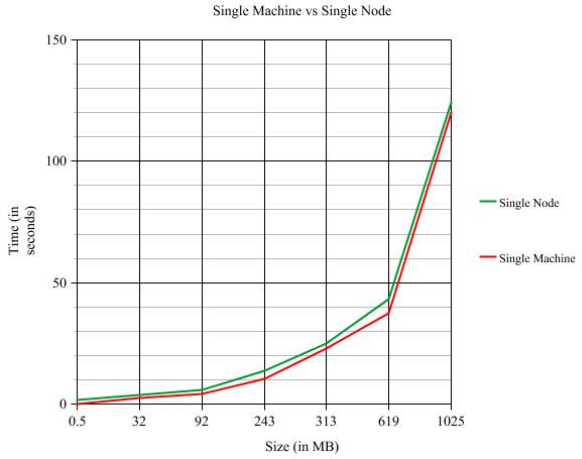

Fig. 6. Graph for single machine vs single node times.

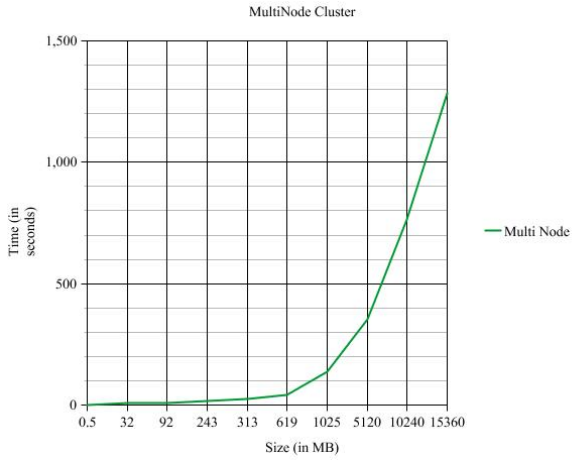

Fig. 7. Graph for times obtained ona multinode cluster.

VI. CALCULATIONS

Speed-up calculation:
Formula used:

$$Speed\ Up = \frac{Time\ taken\ by\ smaller\ system\ to\ execute}{Time\ taken\ by\ larger\ system\ to\ execute}$$

where T is task executed on the two systems

Time taken by single machine (traditional approach) for 1 GB: 120.23 seconds

Time taken by MultiNode cluster (MapReduce paradigm) for 1 GB: 135.13 seconds
Speed up when considering 1GB dataset:

$$\frac{120.23}{135.13} = 0.889 < N$$

(As N =4), thus for small datasets there is a sub-linear speed up.

Time taken by single machine (traditional approach) for 15 GB: Not defined (Infinity)

Time taken by MultiNode cluster (MapReduce paradigm) for 15 GB: 1282.44

Speed up when considering 15GB dataset:

$$\frac{\infty}{1282.44} > N$$

Thus for larger data sets *there is a linear speedup.*

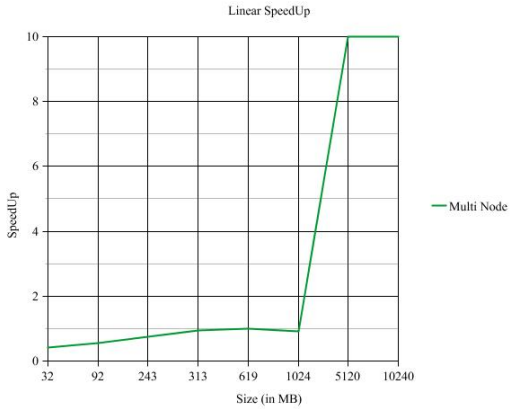

Fig. 8. Graph for linear SpeedUp calculated.

VII. GRAPH INFERENCES

1) Initially the map reduce paradigm due to the overheads of job execution took longer than the traditional approach.

2) As the size of the data set grew the execution time of the two programs became similar and comparable.

3) Further when the size was increased beyond 500 MB, the map reduce model proved to be much more efficient,

4) No hard coding was required in the MapReduce model whereas the traditional approach forced us to hard code folder and file names.

But as the size exceeded 1 GB the single node HDFS too wasn't able to handle the size and thus a distributed MultiNode platform was required (increasing the size of the HDFS by a scale of almost 4).

The MapReduce model allowed us to process the images in a distributed fashion and allowed us massive scale-up. We tested the distributed system for sizes > 1GB and the cluster was able to process them.

VIII. Hadoop Implementation for Autonomous Driving Systems

RGB computations helped us to segregate datasets on the basis of luminance which can be calculated using the formula:

*Luminance = (0.2126 * R + 0.7152 * G + 0.0772 * B)*

The luminance gives an indication of the brightness of the data set which in turn will be a measure of the quality of the data set. Since the image processing modules are implemented taking into consideration some amount of uniformity and quality of the images. For execution purposes, this method of eliminating data sets was an efficient and an effective way.

The modules implemented in this project have a common step of converting videos/images to grayscale format before processing, this step has been shifted to a distributed environment to achieve a similar speed up as achieved in the RGB computation. The incorporation of OpenCv function (RGB2Gray) made the images ready to be used for the python scripts.

IX. Lane Detection and Departure

Lanes on the road can be used to identify whether overtaking is permitted and alert the driver if he is shifting between lanes unintentionally. Such cases can occur when the driver has his sight of vision away from the road

The first step towards these goals is the detection of the lanes. The lanes were identified through Canny Edge Detection which proved to provide on average the most coherent outputs. Following this a Region of interest was calculated. Since, the camera in is fixed at a position the current lane can be defined to fall within the triangle in the area enclosed by a triangle with the vertices:

$[(0, Y_{max}), (X_{max}/2, Y_{max}/2), (X_{max}, Y_{max})]$

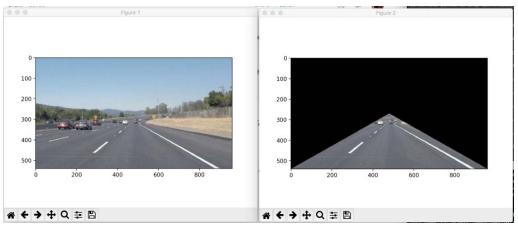

Fig. 9. Selecting a Region of Interest

For detecting the lines from these edges in the region of interest we use Hough transform. Hough transform returns two pairs of coordinates, the points that represent the start and ends of the line. The detected lines are rendered back onto the image itself by looping over all the line coordinates and drawing them on a blank image.

The two lines detected are the left and right lanes. These are non parallel straight lines that if virtually extended will converge somewhere above the horizon. We can monitor the fluctuation in this point of convergence and if it crosses a certain threshold in its movement in the horizontal axis, issues a warning for lane departure.

$$\text{Right lane} : l_1 : y = m_1x + c_1$$
$$\text{Left lane} : l_2 : y = m_2x + c_2$$

Hence the point of convergence, O will have the coordinates:

$$(X_0, Y_0) = \left[\frac{(c_2 - c_1)}{(m_1 - m_2)}, \frac{(m_1c_2 - m_2c_2)}{(m_1 - c_1)}\right]$$

Lane departure will be detected when there is movement in this point of convergence. We are only concerned with the movement in the x axis. Beyond a certain threshold (30% of the image width), the system triggers a warning.

$$\left[\left(x < \left(x_0 - \frac{tolerance}{2}\right)\right) OR \left(x > \left(x_0 + \frac{tolerance}{2}\right)\right)\right]$$

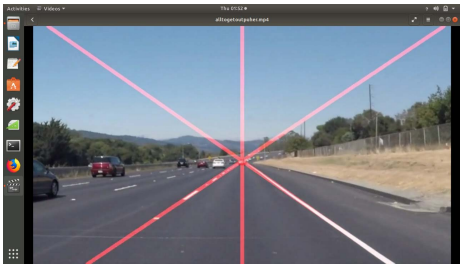

Fig. 10. Point of Intersection and detected lanes

X. Vehicle and Traffic Sign Detection

A trained Haar cascade classifier is used that would detect not a particular car model, but an indiscriminate rear end of a car. Haar features are first trained to obtain a representation to be used latter for real time object detection.

There are two types of training samples negative and positive. Negative samples correspond to non-object images. Positive samples correspond to images with detected objects. Once we have the pictures, we need to crop them so that only our desired object is visible. The best results come from positive images that look exactly like the ones you'd want to detect the object in, except that they are cropped so only the object is visible. Set of negative samples must be prepared manually, whereas set of positive samples is created using OpenCV create samples utility. Negative samples are taken from arbitrary images. These images must not contain detected objects.

Fig. 11. Vehicle detection (Rear ends of automobiles)

A CNN is used for training and testing the traffic sign classifier. SSD (Single Shot MultiBox Detector) only needs an input image and ground truth boxes for each object during training. Default boxes are precomputed fixed size bounding boxes that closely match the distribution of original ground truth boxes. In convolutional fashion, we evaluate a small set of default boxes of different aspect ratios at each location in several feature maps with different scales. For each default box we predict both the shape offsets and confidences for all object categories.

During training we need to determine which default boxes correspond to ground truth detection and train the network accordingly. These default boxes are actually matched to the ground truth boxes in a way such that their IoU (Intersection over Union) ratio also known as the Jaccard ratio is greater than a fixed threshold (Eg. > 0.5). Non Maximum Suppression is the technique used to prune most of the bounding boxes that will be generated during the forward pass of SSD at inference time. Among boxes with predicted probability greater than confidence threshold (Eg. 0.9) and boxes exceeding IoU Threshold (Eg.0.2), the box with highest confidence is selected .This helps to remove noisy images and retain only the most likely predictions.

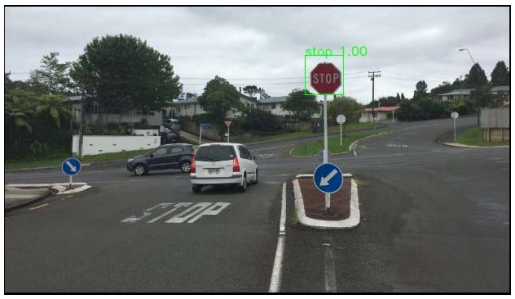

Fig. 12. Stop Sign detection at an intersection.

XI. CONCLUSION

From the statistics and calculations in this paper, it can be inferred that a distributed approach to image processing is a viable and efficient approach. Considering the increasing quality of video cameras which provide users with 4K recordings, storage issues are bound to crop up and thus processing of such unstructured data present in such a high amount of volume will require systems which can parallelize tasks. Also with increase in heterogeneity of data, this approach can also extend to processing of data which is a raw combination of images and videos. The paper also describes that how Hadoop Image Processing Interface (HIPI) provides an extensive range of capabilities which give us way forward in processing images using a distributed architecture

XII. REFERENCES

[1] Jeff Dean, Sanjay Ghemawat. “Map Reduce: Simplified Data Processing on Large Clusters” Google, Inc.
[2] Richard Szeliski – Computer Vision: Algorithms and Applications (Springer, 2010).
[3] I. Demir, A. Sayar – “Hadoop Optimization for Massive Image Processing: Case Study Face Detection” in International Journal for Computers Communication and Control.
[4] S. Vemula, C. Crick – “Hadoop Image Processing Framework” in 2015 IEEE International Conference on Big Data.
[5] S. Arsh, A. Bhatt, P. Kumar – “Distributed Image Processing using Hadoop and HIPI” in 2016 International Conference on Advances in Computing, Communications and Informatics.
[6] T. Epanchintsev, A. Sozykin – “Processing Large Amounts of Images on Hadoop with OpenCV”.
[7] S. Liu,J. Tang, C. Wang, Q. Wang, Gaudiot – “Implementing a Cloud Platform for Autonomous Driving”.
[8] Salman Zubair Toor - "Managing Applications and Data in Distributed Computing Infrastructure".
[9] Dhinkaran R, Prasanthi K, Pradeep Kumar - "Distributed image processing using HIPI".
[10] Bernanrd Alala,Waweru Mwangi, George Okeyo - "Image representation using RGB color space".
[11] Mohammed H Almer - "Cloud Hadoop MapReduce for Remote Sensing Image Analysis - 2009".