Report for Step 1

**a. Research**

**b. Approach**

A video consists of very fast moving images.The idea is to use a traffic cam video as an input and split it into component images, using FFmpeg framework. Each image shall consist of vehicles whose license plates can be read using the openALPR (an open source Automatic License Plate Recognition library). The make,model and other details are extracted and sent over to Azure.

**c. Challenges**

* Converting video to individual component images.
* Parallelization of images to make them compatible with hadoop
* Capturing license plate data using openALPR
* Data cleansing and processing

**d. Solution**

**e. Future Scope**

**Installations and Setup**

The project aims at solving 2 important questions related to vehicular traffic:

1. Spotting and reporting stolen cars
2. Calculating and projecting the environmental effects of air pollution generated by vehicles at any given location at a specified time
3. FFmpeg
4. OpenALPR
5. Azure Stream Analytics
6. Inotify
7. Python
8. Java
9. C#
10. MapReduce

To begin running the project, we run the shell file and the input video through the command line.

./vidmapred.sh <video file name>

vidmapred.sh is a shell file that contains all the commands that need to execute consecutively. It contains commands to create input & output folders, run hadoop jobs and provide updates.

The input to the project is in the form of a video generated by a cctv camera that monitors traffic on a busy city street.

Frame rate (expressed in frames per second or fps) is the rate at which consecutive images, called frames are displayed in an animated display. Ideally, films are shot at 24 frames per second.

We consider the traffic video to consist of 24 frames per second for the project.

This video is processed via FFmpeg software.

FFmpeg is a free software framework that produces libraries and programs for handling multimedia data. FFmpeg is highly portable. It compiles and runs on various platforms such as Linux, Mac OS X, Microsoft Windows, Solaris etc. under a wide variety of build environments, machine architectures and configurations.

FFmpeg splits the input video into individual frames that constitute the video. It is a codec library that handles video format conversions (binary to video data and vice-versa).

The number of frames per second that the video is split can be controlled and can vary from 1 fps(frame per sec) to 30 fps.

The syntax for running the ffmpeg framework is in the following format:

ffmpeg [global\_options] {[input\_file\_options] -i input\_url} ... {[output\_file\_options] output\_url} .

The syntax used in the project is:

ffmpeg -i $VIDEO\_FILE -r 2/1 $MAPRED\_IMG\_DIR/%07d.png

‘ffmpeg’ reads from an arbitrary number of input "files” specified by the -i option, and writes to an arbitrary number of output "files", which are specified by a plain output url. Anything found on the command line which cannot be interpreted as an option is considered to be an output url.

The above syntax uses $VIDEO\_FILE as an input. 2/1 signifies number of frames per second. $MAPRED\_IMG\_DIR is the output folder.The individual images that are extracted from the video are stored in this folder.

The %07d signifies that the images that are extracted will be in the order of ‘0000000’ numbers. For example, the first image will be stored as 0000001.png. This format ensures that the images extracted will not run out of memory to be stored.

After the FFmpeg stage, the input video is converted into a bunch of images.

In the next stage, the processing begins on the images. The approach is to process all the images using mapreduce.

Mapreduce is suited for this project since the size of the images generated by the video processing is big data.

24 hours video = 24\*60\*60 = 86400 seconds video

24 frames per second\* 86400 seconds = 2,073,600 frames or images

1 image ~ 350 kb

2,073,600 \* 350 =725,760,000 kb ~ 725 gb of data per day

Also, since input to the mapper should be in the form of consecutive lines of text, all the image names are extracted and collected to form a text file. Each line in this file consists of one image name.

The Mapper is written using Python. After passing the above mentioned file to Hadoop, it splits the file into individual lines, each representing an image file name. These lines (image file names) are passed to the mapper one by one.

The mapper reads each line as a file name of one image from the video sequence. In other words, we have passed one image to the mapper (in the form of its file name). Next, the mapper passes this file name to the OpenALPR command, which it calls using the subprocess package. In other words, OpenALPR now processes a single image from the video sequence. Since the mappers can be parallelized, multiple mapper instances will each process a different image frame, and so the overall speed of processing is increased multiple times.

OpenALPR is an open source Automatic License Plate Recognition library written in C++ with bindings in C#, Java, Node.js and Python.

This library analyzes images to identify license plates. It also enables the users to select from a range of countries to help identify the country of issuance of the license plate.The output of this stage is a text representation of license plate characters.

The library reads license plate numbers for each image passed as an input. The mapper result is the license plate number and confidence. Confidence is the probability with which the accuracy of license plate that is read is determined.

The Reducer takes the output from the mapper, processes it to only select license plates from the file, and provides uncluttered information.

This output is stored in the format ‘License\_plate\_number’ ‘Confidence:’ ‘Numerical\_value\_of\_Confidence’

For example: -BD 792346 confidence: 86.0561

References:

<https://www.ffmpeg.org/ffmpeg-formats.html>

<http://doc.openalpr.com/>

Mapreduce described in brief:

* The python Mapreduce extracts the car license info and outputs a text file (- <license-plate-number> confidence : 89>) which is sent to Java Mapreduce for obtaining Make, model and other info. (Ideally, an API would fetch make, model, but we have simulated that in our Java code. Also, Python is chosen to mapreduce the frame to text license number extraction as it is much lighter.)
* The Java Mapreduce eliminates the duplicate license numbers and fetches the car info. The output json is sent to the C# Event-hub listener that sends the files to the Azure stream analytics job.