ABSTRACT

In parallel computing, an embarrassingly parallel workload or problem (also called perfectly parallel or pleasingly parallel) is one where little or no effort is needed to separate the problem into a number of parallel tasks. This is often the case where there is little or no dependency or need for communication between those parallel tasks, or for results between them. Thus, these are different from distributed computing problems that need communication between tasks, especially communication of intermediate results. They are easy to perform on server farms which lack the special infrastructure used in a true supercomputer cluster. In this project, the perceptual visibility of an image that is degraded by atmospheric haze is enhanced. The server shares the workload among the clients which perform hazing technique independently. There are many haze removal techniques of which DARK CHANNEL PRIOR Technique which is a SINGLE IMAGE HAZE REMOVAL technique is used here as an application of parallel computing. Using this prior with the haze imaging model, it can directly estimate the thickness of the haze and recover a high-quality haze free image. When CPU (without parallel computing) is used for computation of hazed images it takes a very long time to generate the dehazed images as output but when both CPU and GPU are used parallely to perform computation of images together then it shows a drastic change in time for processing of images. In this project Embarrassingly Parallel Computations over the Web using XAMPP stack for Haze Removal (EPCWebX) using one node (computer), the time taken is same as that of computational time of CPU and as the no. of nodes (computers) increase and are run parallely, the computational time slowly decreases and tries to meet the computational time of CPU+GPU. When the number of nodes are 4 then this method surpasses the computational time of the CPU+GPU.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

1.1 Problem Statement

This project is an application for performing parallel computations by splitting a single problem into a number of independent sub problems and solving them parallely with n number of available clients. This concept is applied to dehaze the hazed images using DARK CHANNEL PRIOR Technique which is a SINGLE IMAGE HAZE REMOVAL technique.

1.2 Objective

Images of outdoor scenes are usually degraded by the turbid medium in the atmosphere. So, the main objective of this project is to accurately determine which areas are hazy and dehaze the hazy areas and to complete all rendering in a reasonable amount of time. Here each client would independently perform their tasks without communicating with other tasks.

1.3 Scope

The dark channel prior method consists of a process called soft matting which uses Preconditioned Conjugate Gradient(PCG) algorithm which takes about 10-20 sec to process a 600*400 pixel image on a PC with a 3.0GHz Intel Pentium 4 processor. The dark channel prior method can unveil the details and recover vivid color information even in very dense haze regions. But it is a kind of statistic, which may not work for some particular images. When the scene objects are inherently similar to atmospheric light and no shadow is cast on them , the dark channel prior is invalid. EPCWebX method will underestimate the transmission for these objects.

1.4 Approach

In this project, "SINGLE IMAGE APPROACH" is used which relies on a statistical assumption. The kind of technique used in Single Image approach is dark channel prior which is based on statistical assumption that-"in most of the non-sky patches, atleast one color channel has very low intensity at some pixels. In other words, the minimum intensity in such a patch should have a very low value." This assumption is used to estimate TRANSMISSION. Another

assumption is that-“a portion of the scene is dominated by airlight.” This assumption is used to estimate “AIRLIGHT”.

1.5 Motivation

Hadoop could be used as a solution for parallel computations. But in Hadoop systems, softwares’ (like Apache Spark) should be installed for parallel computations which would make it system dependent. In the project it has been implemented in a new way so that the computations are system independent. Hence this project is done without using Hadoop.

1.6 Challenges

- One of major challenges while dealing with embarrassingly parallel computing is synchronization between clients. This scenario boils down to the situation that once an image is selected for processing, no other client should process the same image.
- Load imbalance- Different amounts of works across processors and the different speeds of the processors cause load imbalance.
- Communication cost between server and client.
- The parallel speedup of any program is limited by the time needed for any sequential portions of the program to be completed.

2. LITERATURE REVIEW

2.1. Parallel Programming for the Web

Parallel hardware is today's reality and language extensions that ease exploiting its promised performance flourish. For most mainstream languages, one or more tailored solutions exist that address the specific needs of the language to access parallel hardware. Yet, one widely used language is still stuck in the sequential past: JavaScript, the lingua franca of the web. Existing solutions do not transfer well to the world of JavaScript due to differences in programming models, the additional requirements of the web, like safety, and to developer expectations. To address this River Trail - a new parallel programming API designed specifically for JavaScript was proposed and it satisfies the needs of the web. To prove that this approach is viable, a prototype JIT compiler in Firefox was implemented that shows an order of magnitude performance improvement for a realistic web application. (Parallel Programming for the Web Stephan Herhut Richard L. Hudson Tatiana Shpeisman Jaswanth Sreeram Intel Labs {stephan.a.herhut, rick.hudson, tatiana.shpeisman, jaswanth.sreeram}@intel.com)

2.2. Web-based parallel computing infrastructure

The Internet, best known by most users as the World-Wide-Web, continues to expand at an amazing pace. A new infrastructure is proposed to harness the combined resources, such as CPU cycles or disk storage, and make them available to everyone interested. This infrastructure has the potential for solving parallel supercomputing applications involving thousands of cooperating components. This approach is based on recent advances in Internet connectivity and the implementation of safe distributed computing embodied in languages such as Java. A prototype of a global computing infrastructure, called SuperWeb, is develpoed that consists of hosts, brokers and clients. Hosts register a fraction of their computing resources (CPU time, memory, bandwidth, disk space) with resource brokers. Client computations are then mapped by the broker onto the registered resources. An economic model is examined for trading computing resources, and several technical challenges associated with such a global computing environment are discussed. (A. D. Alexandrov, M. Ibel, K. E. Schauser and C. J. Scheiman, "SuperWeb: towards a global Web-based parallel computing infrastructure," Proceedings 11th International Parallel Processing Symposium, Genva, Switzerland, 1997, pp. 100-106.)

2.3. Design issues in building Web-based parallel programming environments

The recent advances in Internet connectivity and Web technologies for building Web-based parallel programming environments (WPPEs) that facilitate the development and execution of parallel programs on remote high-performance computers are exploited. A Web browser running on the user's machine provides a user-friendly interface to server-site user accounts and allows the use of parallel computing platforms and software in a convenient manner. The user may create, edit, and execute files through this Web browser interface. This new Web-based client-server architecture has the potential of being used as a future front-end to high-performance computer systems. The design and implementation of several prototype WPPEs that are currently in use at the Northeast Parallel Architectures Center and the Cornell Theory Center are discussed. These initial prototypes support high-level parallel programming with Fortran 90 and High Performance Fortran (HPF), as well as explicit low-level programming with Message Passing Interface (MPI). The lessons learned during the development process and outline the tradeoffs of various design choices in the realization of the design are detailed. Special concentration on providing server-site user accounts, mechanisms to access those accounts through the Web, and the Web-related system security issues are provided. (K. Dincer and G. C. Fox, "Design issues in building Web-based parallel programming environments," Proceedings. The Sixth IEEE International Symposium on High Performance Distributed Computing (Cat. No.97TB100183), Portland, OR, USA, 1997, pp. 283-292.)

2.4. Internet-based Parallel Computing using Java

Java offers the basic infrastructure needed to integrate computers connected to the Internet into a seamless parallel computational resource: a flexible, easily-installed infrastructure for running coarse grained parallel applications on numerous, anonymous machines. Ease of participation is seen as a key property for such a resource to realize the vision of a multiprocessing environment comprising thousands of computers. Javelin, a Java-based infrastructure for global computing is presented. The system is based on Internet software technology that is essentially ubiquitous: Web technology. Its architecture and implementation require participants to have access only to a Java-enabled Web browser. The security constraints implied by this, the resulting architecture, and current implementation are presented. The Javelin

architecture is intended to be a substrate on which various programming models may be implemented. Several such models are presented: A Linda Tuple Space, an SPMD programming model with barriers, as well as support for message passing. Experimental results are given in the form of micro-benchmarks and a Mersenne Prime application that runs on a heterogeneous network of several parallel machines, workstations, and PCs.(Christiansen, Bernd O. et al. "Javelin: Internet-based Parallel Computing using Java." Concurrency - Practice and Experience 9 (1997): 1139-1160.)

2.5. Embarrassingly Parallel computation

The term "Embarrassingly parallel" is first found in the literature in a 1986 book on multiprocessors by MATLAB creator Cleve Moler who claims to have invented the term. An alternative term "Pleasingly parallel" has gained some use, perhaps to avoid the negative connotations of embarrassment in favour of a positive reflection on the parallelizability of the problems:- "Of course there is nothing embarrassing about these programs at all." The usage of the term "Embarrassingly" may derive from the idiom "an embarrassment of riches":- which means an abundance or overabundance of something which is good. These are different from distributed computing problems that need communication between tasks, especially communication of intermediate results. They are easy to perform on server farms which lack special infrastructure used in a true supercomputer cluster. The opposite of embarrassingly parallel problems are inherently serial problems, which cannot be parallelized at all.

There are several forms of parallel computing : bit-level, instruction-level and task-level parallelism. Parallel computing is closely related to concurrent computing – they are frequently used together, and often conflated, though the two are distinct. It is possible to have parallelism without concurrency (bit-level parallelism) and concurrency without parallelism (such as multi-tasking by time sharing on a single core CPU). In parallel computing, a computational task is simply broken down into several, very similar subtasks that can be processed independently and whose results are combined afterwards upon completion. In contrast, in concurrent computing, the various process often do not address related tasks, when they do, as is typical in distributed computing, the separate tasks may have a varied nature and often required some inter process communication during execution. Historically parallel computing was used for scientific computing and the simulation of scientific problems, particularly in the natural and engineering

sciences, such as meteorology. This led to the design of parallel hardware and software as well as high performance computing.

2.6. Image De-Hazing

Image dehazing is a task which is performed parallelly in this project. Robby T. Tan (2008) has introduced an automated method that only requires a single input image. Two observations are made based on this method, first, clear day images have more contrast than images afflicted by bad weather; and second, airlight whose variant mostly depends on the distance of objects to the observer tends to be smooth. Tan develops a cost function in the framework of Markov random fields based on these two observations. The results have larger saturation values and may contain halos at depth discontinuities. Tarel et al. (2009) have demonstrated algorithm for visibility restoration from a single image that is based on a filtering approach. The algorithm is based on linear operations and needs various parameters for adjustment. It is advantageous in terms of its speed. This speed allows visibility restoration to be applied for real-time applications of dehazing. They also proposed a new filter which preserves edges and corner as an alternate to the median filter. The restored image may be mot good because there are discontinuities in the scene depth. Yu et al. (2010) have proposed a new fast dehazing method based on the atmospheric scattering model. The atmospheric scattering model is simplified earlier to visibility restoration. First they acquire a coarse approximation of the atmospheric veil and then the coarser estimation is smoothed using a fast bilateral filtering approach that preserving edges. The complexity of this method is only a linear function of the number of input image pixels and this thus permits a very fast implementation. Fang et al. (2011) have discussed a new fast haze removal algorithm from multiple images in uniform bad weather conditions is proposed which bases on the atmospheric scattering model. The basic idea is to establish an over determined system by forming the hazy images and matching images taken in clear days so that the transmission and global airlight can be acquired. The transmission and global airlight solved from the equations are applied to the local hazy area. The discussed algorithm reduces haze effectively and achieves accurate restoration. He et al. (2011) have proposed a simple but effective image prior dark channel prior to remove haze from a single input image. The dark channel prior is a type of statistics of outdoor haze-free images. In most of the non-sky patches, at least one color channel (RGB) has very low intensity at some pixels

(called dark pixels). These dark pixels provide the estimation of haze transmission. They can directly evaluate the thickness of the haze using this prior with the haze imaging model and get a high-quality haze-free image. The dark channel prior does not work efficiently when the surface object is similar to the atmospheric light. Long et al. (2012) have presented a fast and physical-based method. Based on the dark channel prior, they can easily extract the global atmospheric light and roughly estimate the atmospheric veil with the dark channel of the input haze image. Then refine the atmospheric veil using a low-pass Gaussian filter. In most cases, the approach can achieve good results. But, when the images have dense and heterogeneous haze, the results obtained will have color distortion especially in the bright regions and loss of details.

2.7. Parallel De-Hazing

Haze is an atmospheric phenomenon that fogs the visibility of the scenes. Removing the haze has been an important issue in image processing technologies. Many image dehazing technologies with evolution algorithms have been proposed to remove the fog in the image. However, these algorithms usually are compute-intensive. A parallel hybrid evolution algorithm based on GPU to enhance the computational performance. In traditional evolution algorithms, the calculation of fitness function occupies the most of the computation time. In the proposed method, it is implemented GPU by using CUDA framework to reduce the computational load. The experiment results show that the proposed method can remove the haze efficiently and successfully. (C. Hung, R. Yan and H. Wang, "Parallel image dehazing algorithm based on GPU using fuzzy system and hybrid evolution algorithm," 2016 17th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), Shanghai, 2016, pp. 581-583.)

Image dehazing based on dark channel has good performance and has been widely applied in computer vision. But with high computational complexity, it's difficult to apply this algorithm into real-time image dehazing. In order to realize the real-time HD image dehazing, a GPU-accelerated parallel computing implementation is proposed to optimize the image dehazing algorithm based on dark channel. A parallel computing implementation based on transposition is presented to optimize the mean filter in guided filter. Texture memory is used to accelerate the process of min filter. Reduction algorithm is used to calculate the value of atmospheric light of a haze image at high speed. Experiment results indicate that, under the premise of zero accuracy

loss, the algorithm can deal with the 1080p with 38 frames per second and the speedup up to 51x to meet the requirements of real-time HD image dehazing.(X. Wu, R. Wang, Y. Li and K. Liu, "Parallel Computing Implementation for Real-Time Image Dehazing Based on Dark Channel," 2018 IEEE 20th International Conference on High Performance Computing and Communications; IEEE 16th International Conference on Smart City; IEEE 4th International Conference on Data Science and Systems (HPCC/SmartCity/DSS), Exeter, United Kingdom, 2018, pp. 1-5.)

Video dehazing is an important preprocessing task to improve the visibility of various UAV videos for further processing, such as target recognition and tracking. State-of-the-art Guided Filter based method employs dark channel prior to infer the direct scene transmission parameter and refines it using guided image filter. The quality of the dehazing result from this method is satisfactory except some artificial effect in large sky area. Meanwhile, the computational efficiency is not very high and the real-time performance is not reached. A parallel framework for video dehazing algorithm is proposed to accelerate the dehazing computation on UAV ground station. Firstly parallel O(1) complexity local minimum filter is employed to get the initial dark channel image, which is further refined by parallel Joint Recursive Bilateral Filter. Combined with the atmosphere parameter which is obtained by histogram based estimation, the dehazing result is finally achieved. The proposed method is evaluated on multi-core UAV ground station using C++ programming language with OpenMP compiler directive. Experimental results show that the proposed method outperforms available Guided Filter based method and has a real-time performance.(Y. Cheng, W. Niu and Z. Zhai, "Video dehazing for surveillance unmanned aerial vehicle," 2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC), Sacramento, CA, 2016, pp. 1-5.)

3. TECHNICAL REQUIREMENTS REVIEW

The software and libraries required for the project are described below.

3.1 XAMPP STACK

XAMPP stands for Cross-Platform, Apache, MySQL and PHP and is the preferred choice when it comes to deploying localhost web applications. It provides a solid and reliable foundation for building web application.

3.2 MySQL Server

MySQL server is a SQL complaint server, in other words it is a relational model database server. It is very popular because it is free. It was developed by Sun and moved to Oracle when Oracle acquired Sun. Oracle continued improving it. The latest version is 5.7.
The relational model is a way to organize data in tables, where each table as a primary key (which is unique) and rows can relate to each other using that key.

3.3 PHP

The PHP Hypertext Preprocessor (PHP) is a programming language that allows web developers to create dynamic content that interacts with databases. PHP is basically used for developing web based software applications. PHP will work with virtually all database software, including Oracle and Sybase but most commonly used is freely available MySQL database.

3.4 Javascript

Javascript is a lightweight, interpreted programming language. It is designed for creating network centric applications. It is complimentary to and integrated with Java. Javascript is very easy to implement because it is integrated with HTML. It is open and cross-platform.

3.5 FFmpeg

FFmpeg is the leading multimedia framework, able to stream, filter, decode ,encode, transcode,mux,demux and play pretty much anything that humans and machines have created. It supports the most obscure ancient formats up to the cutting edge. No matter if they were

designed by some standards committee, the community or a corporation. It is also highly portable: FFmpeg compiles, runs, and passes testing infrastructure FATE across Linux, Mac OS X, Microsoft Windows, the BSDs, Solaris, etc. under a wide variety of build environments, machine architectures, and configurations.

3.6 Libraries

1) Opencv

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in commercial products. OpenCV library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc.

2) Javascript utility library

All Bootstrap's JavaScript files depend on util.js and it has to be included alongside the other JavaScript files. If you're using the compiled (or minified) bootstrap.js, there is no need to include this—it's already there.

util.js includes utility functions and a basic helper for transitionEnd events as well as a CSS transition emulator. It's used by the other plugins to check for CSS transition support and to catch hanging transitions.

Util.js contains a number of utility functions to help you update your web pages with javascript data (such as might be returned from the server). DWR's focus is not on DOM based utilities, thus these utilities are limited in scope. There are several other libraries (JS templating

tools, jQuery, etc.) that are more robust in this area that you may want to explore if util.js does not meet your needs.

4. SYSTEM ARCHITECTURE AND DESIGN

4.1 SYSTEM ARCHITECTURE

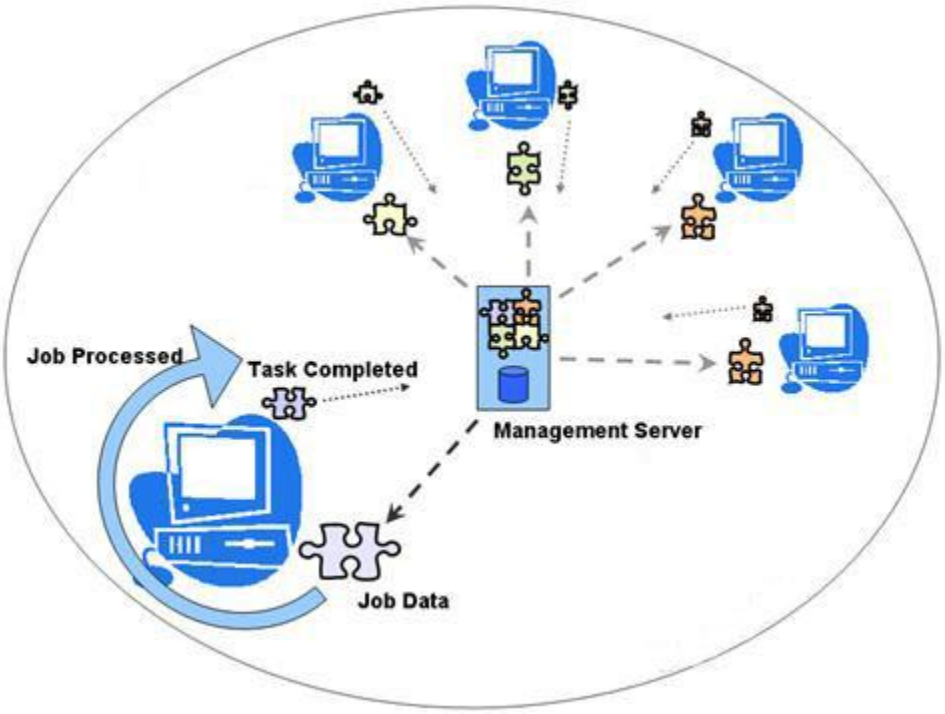

Fig: 4.1 System Architecture

A computation that can be divided into a number of completely independent parts, each of which can be executed by a separate processor. Parallel computing is the use of two or more processors (cores, computers) in combination to solve a single problem. The programmer has to figure out how to break the problem into pieces, and has to figure out how the pieces relate to each other. For example, a parallel program to play chess might look at all the possible first moves it could make. Each different first move could be explored by a different processor, to see how the game would continue from that point. At the end, these results have to be combined to figure out which is the best first move. Actually, the situation is even more complicated, because if the program is looking ahead several moves, then different starts can end up at the same board position. To be efficient, the program would have to keep track of this, so that if one processor had already evaluated that position, then others would not waste time duplicating the effort. This is how must parallel chess-playing systems work, including the famous IBM Deep Blue machine that beat Kasparov.

4.2 SYSTEM DESIGN

4.2.1 Data Flow Diagram

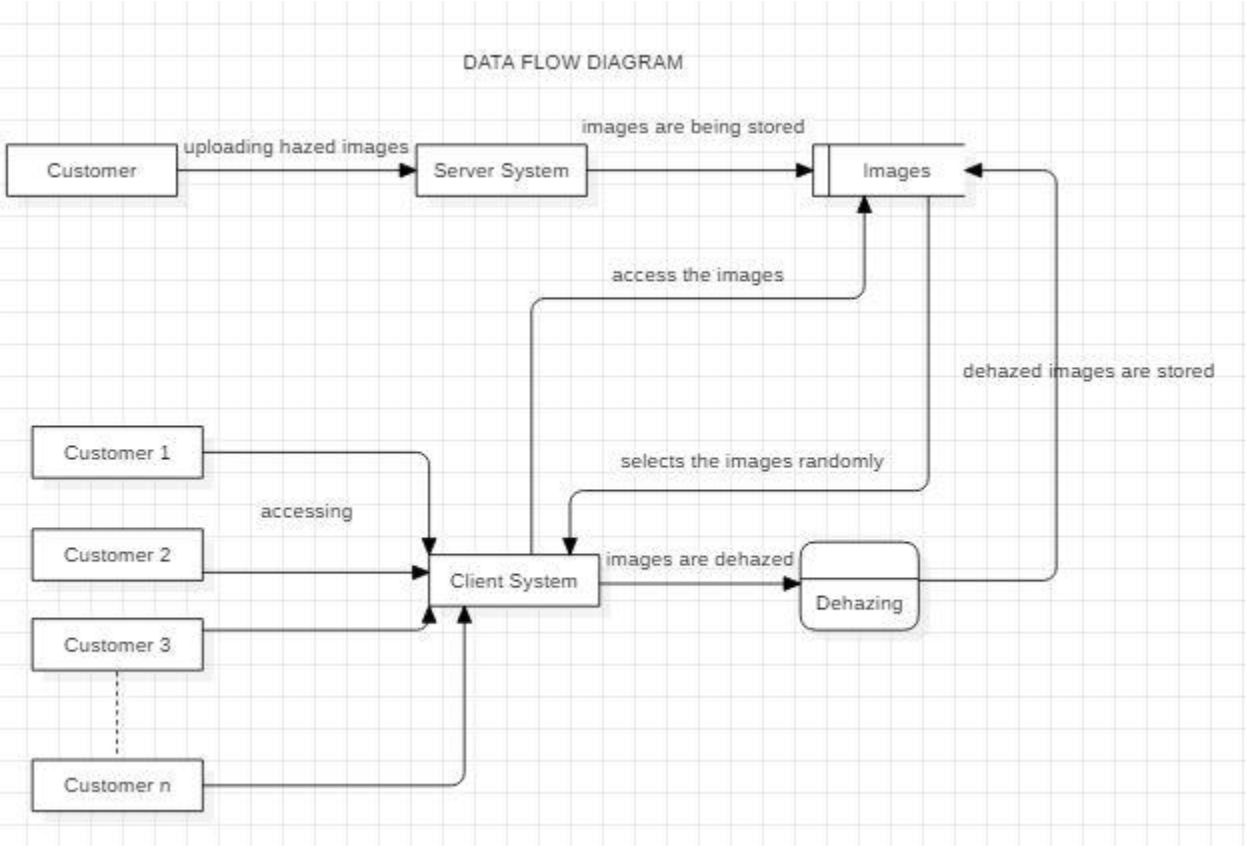

Fig: 4.2 Data Flow Diagram for Parallel Computation of Image Dehazing

A customer has to initially upload the hazed images to the server system and then those images are stored in the database and then the client systems are used for processing the images from the database and dehazed images are stored back into the database and then the customer downloads back it in a zip file.

4.2.2 Class Diagram

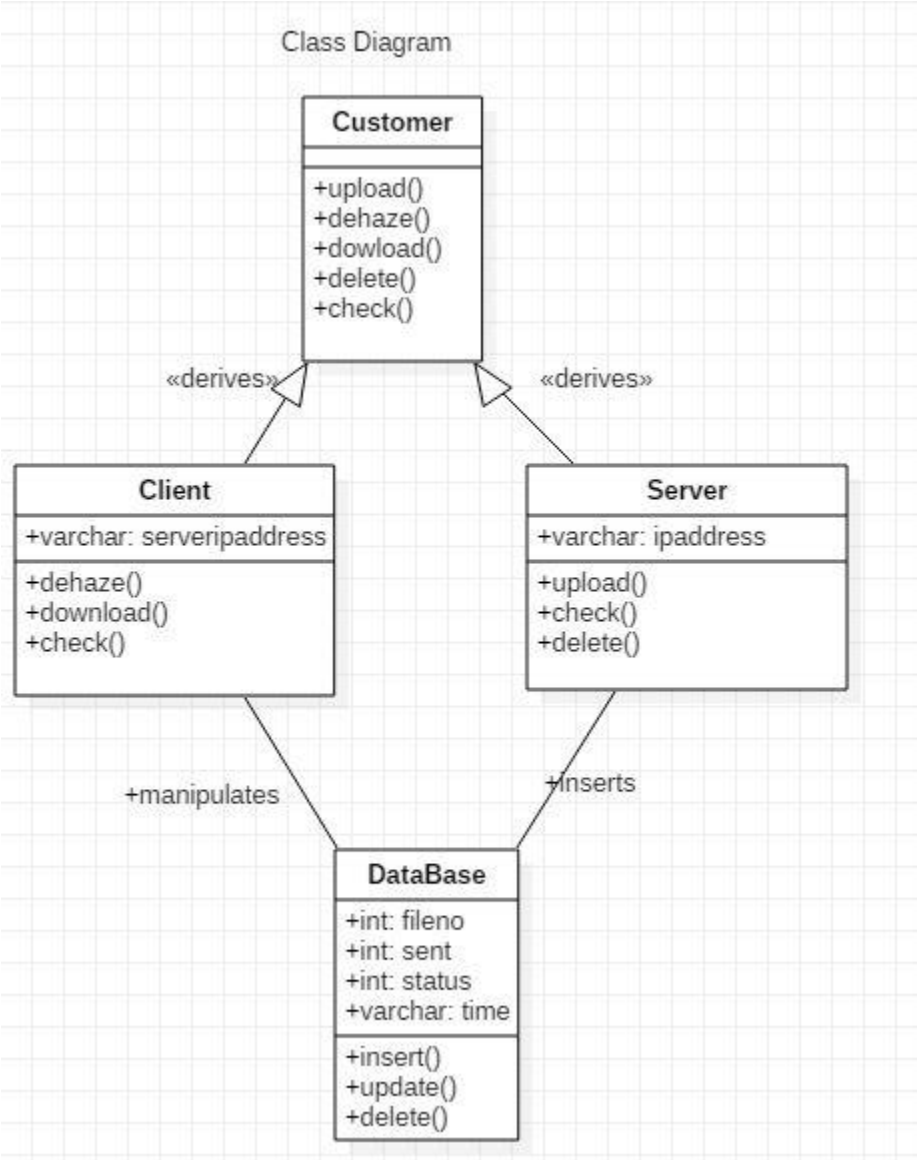

Fig: 4.3 Class Diagram for Parallel Computation of Image Dehazing

The class diagram tells how the classes are interlinked with each other and how the operations of each class are used. By observing the class diagram an abstract view of the whole system is got.

4.2.3 Sequence Diagram

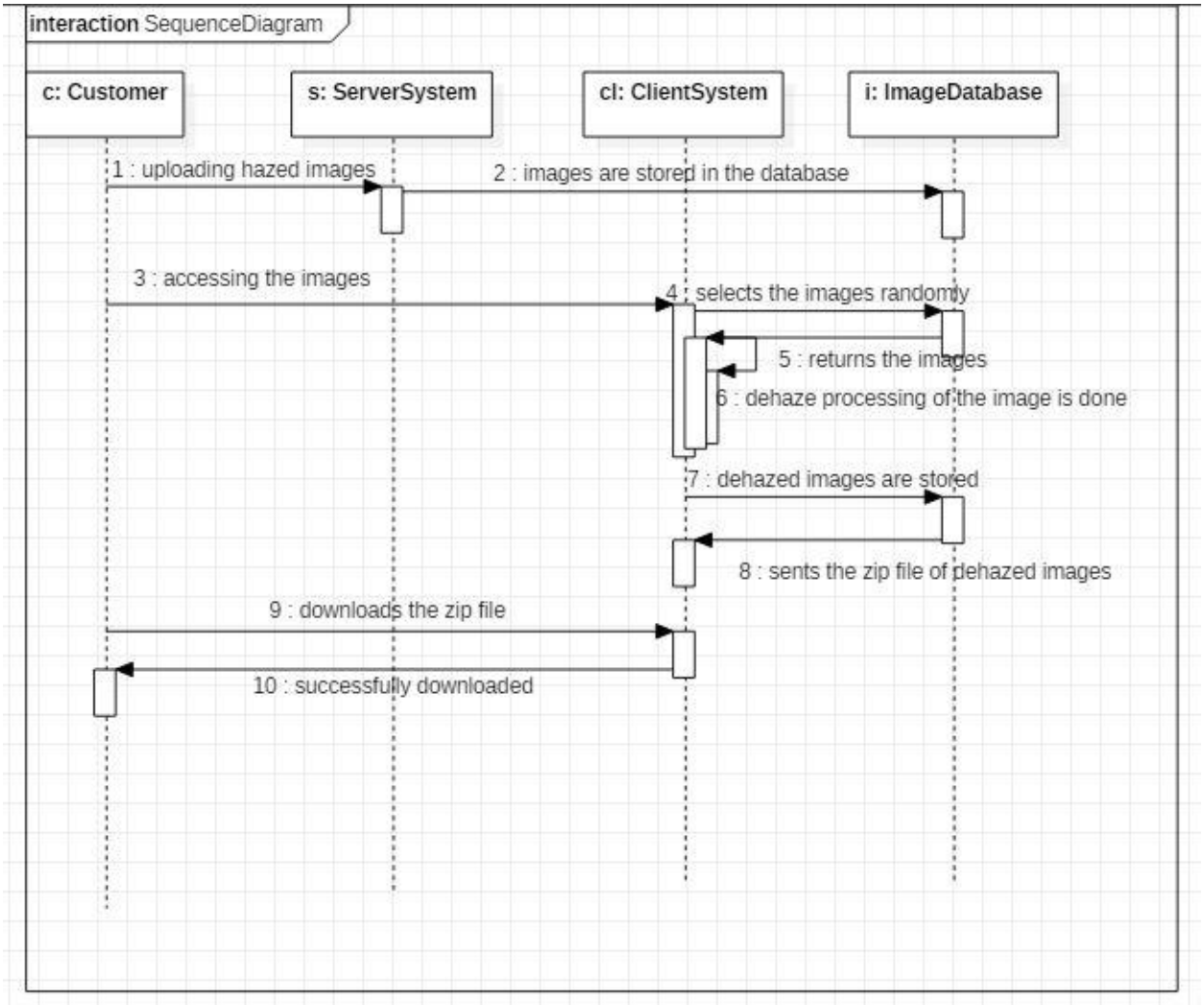

Fig: 4.4 Sequence Diagram for Parallel Computation of Image Dehazing

The sequence diagram tells about the procedure of how the flow of the project is done and also how the customer goes with the flow of the process.

4.2.4 Collaboration Diagram

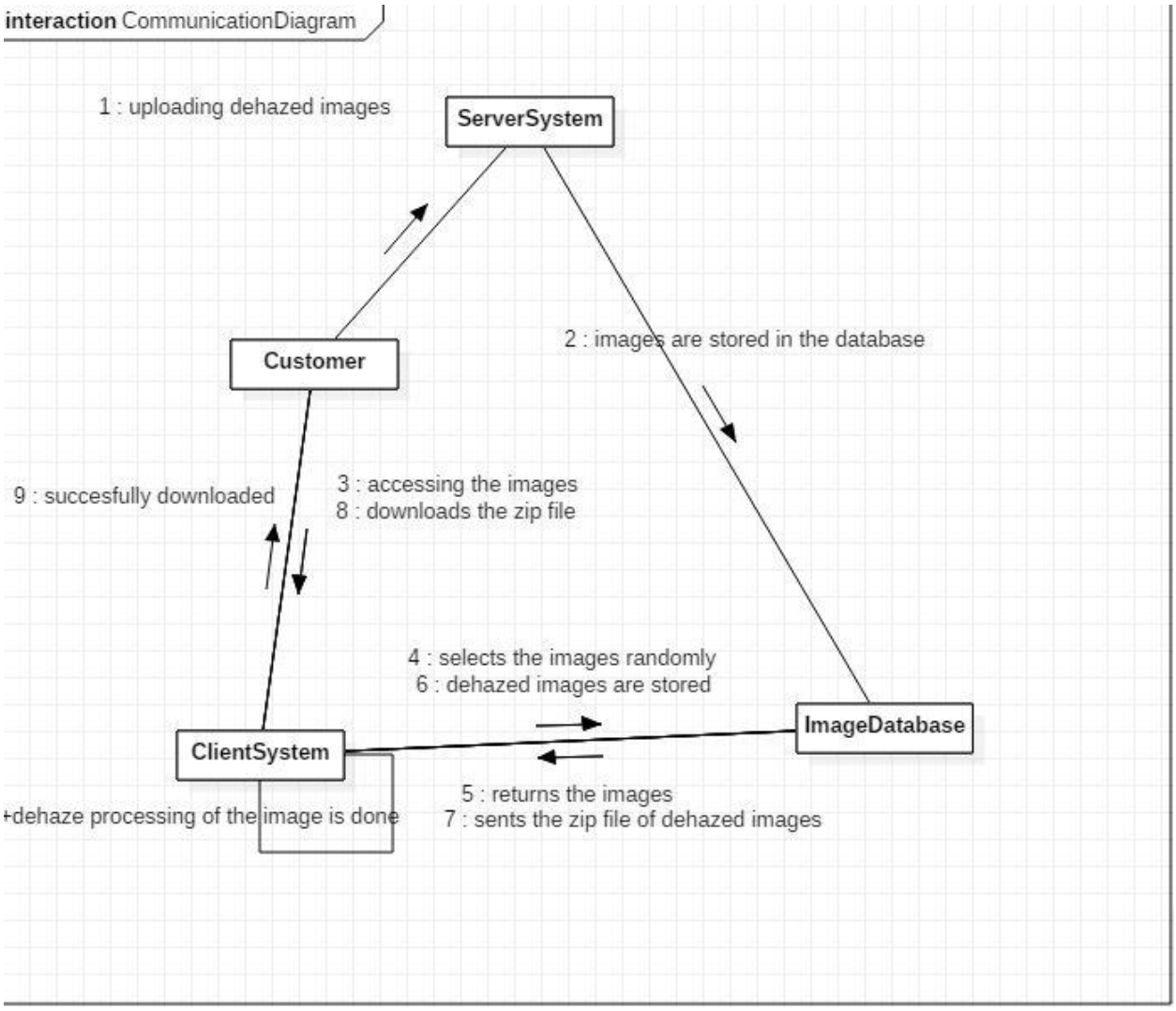

Fig: 4.5 Collaboration Diagram for Parallel Computation of Image Dehazing

4.2.5 Object Diagram

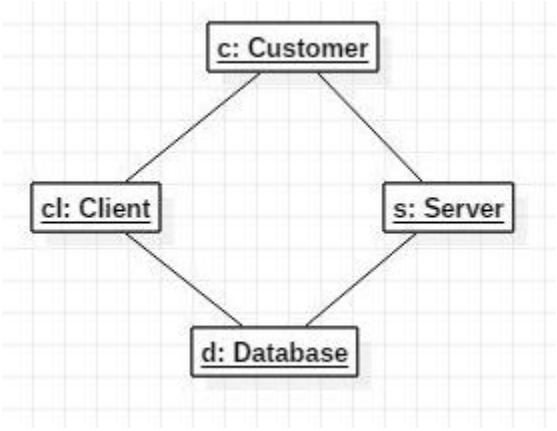

Fig: 4.6 Object Diagram for Parallel Computation of Image Dehazing

4.2.6 Use case Diagram

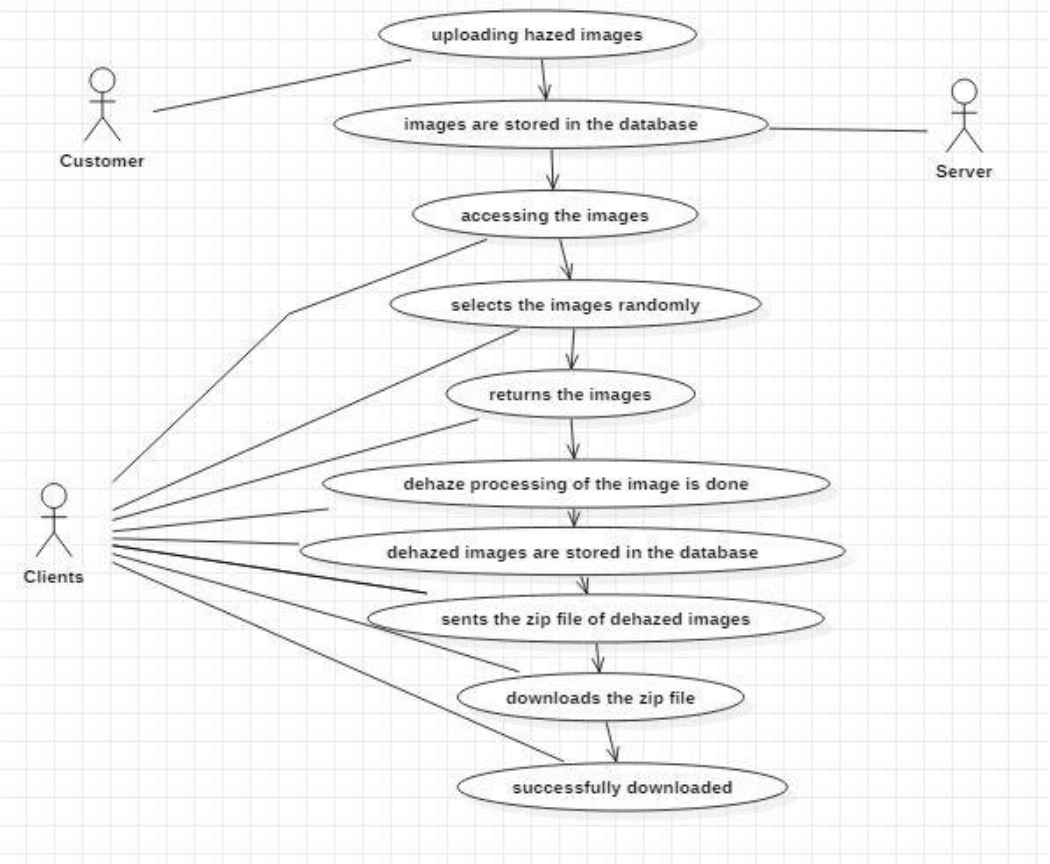

Fig: 4.7 Use case Diagram for Parallel Computation of Image Dehazing

4.2.7 Activity Diagram

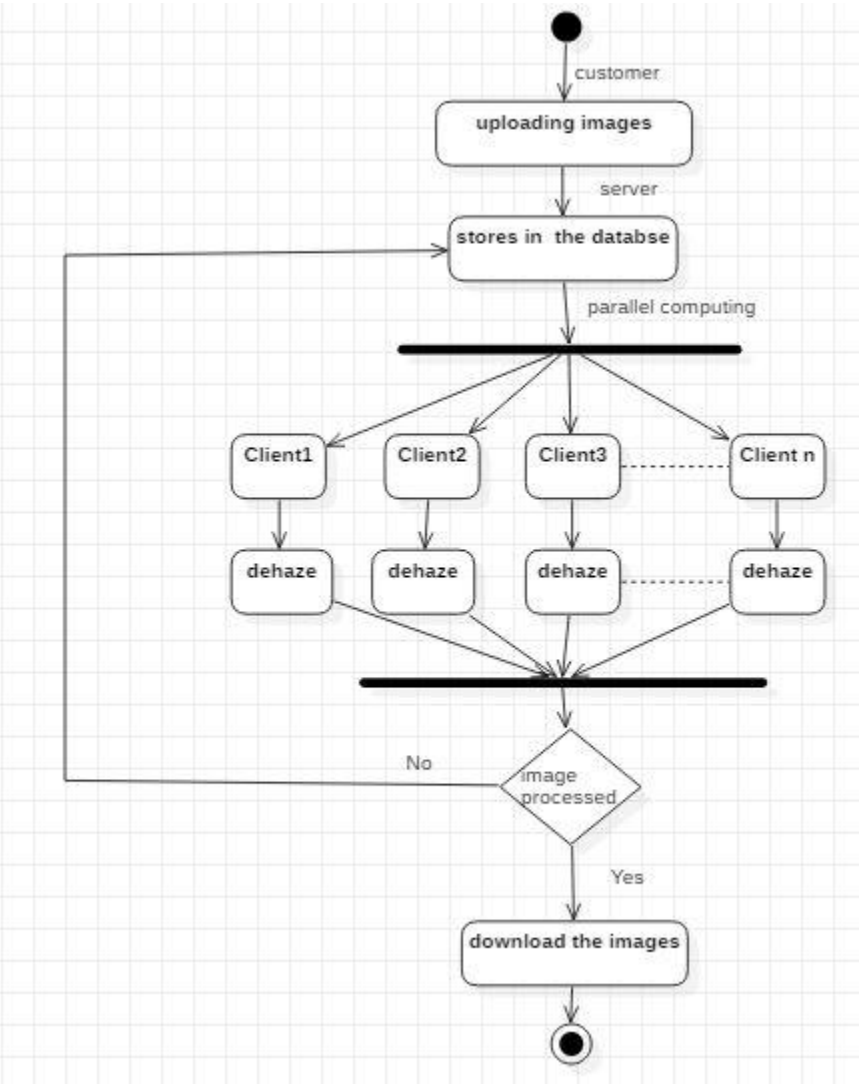

Fig: 4.8 Activity Diagram for Parallel Computation of Image Dehazing

4.2.8 Database Design

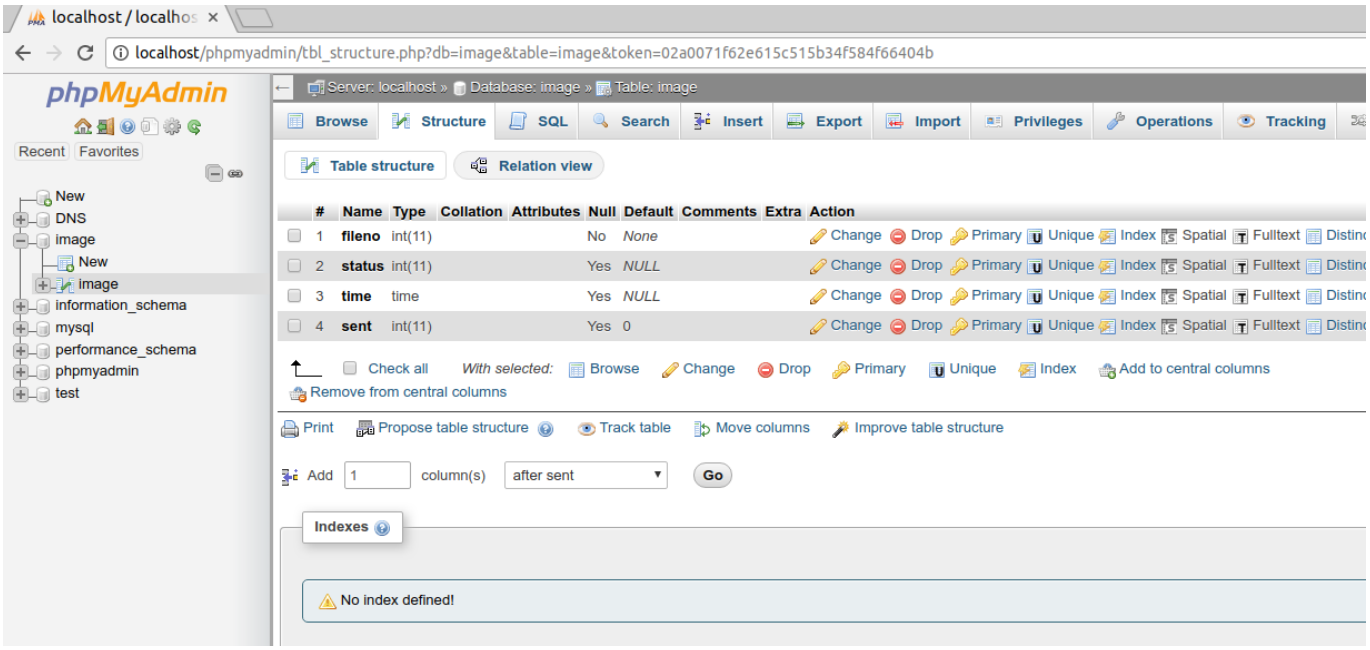

Fig: 4.9 Database Design for Parallel Computation of Image Dehazing

The image tells about the database that has been used. It consists of fileno which tells the image names, status which tells whether the image is processed successfully or not, time which tells exactly at which time the image is stored and sent tells whether the image is successfully taken into processing stage or not.

5. IMPLEMENTATION

The dark channel prior haze removal technique is parallelly computed by clients.

Haze Removal Using Dark Channel Prior

Estimating the transmission

It is first assumed that the atmospheric light **A** is given. Further assume that the transmission in a local patch Ω(**x**) is constant. The patch's transmission is denoted as $\tilde{t}(\mathbf{x})$. Taking the min operation in the local patch on the haze imaging Equation I(x) = J(x)t(x) + A(1 − t(x)),

$$\min_{\mathbf{y}\in\Omega(\mathbf{x})}(I^c(\mathbf{y})) = \tilde{t}(\mathbf{x}) \min_{\mathbf{y}\in\Omega(\mathbf{x})}(J^c(\mathbf{y})) + (1-\tilde{t}(\mathbf{x}))A^c. \quad (1)$$

Notice that the min operation is performed on three color channels independently. This equation is equivalent to:

$$\min_{\mathbf{y}\in\Omega(\mathbf{x})}(\frac{I^c(\mathbf{y})}{A^c}) = \tilde{t}(\mathbf{x}) \min_{\mathbf{y}\in\Omega(\mathbf{x})}(\frac{J^c(\mathbf{y})}{A^c}) + (1 - \tilde{t}(\mathbf{x})). \quad (2)$$

Then, the min operation among three color channels on the above equation and obtain:

$$\min_c(\min_{\mathbf{y}\in\Omega(\mathbf{x})}(\frac{I^c(\mathbf{y})}{A^c})) = \tilde{t}(\mathbf{x}) \min_c(\min_{\mathbf{y}\in\Omega(\mathbf{x})}(\frac{J^c(\mathbf{y})}{A^c})) + (1 - \tilde{t}(\mathbf{x})). \quad (3)$$

According to the dark channel prior, the dark channel Jdark of the haze-free radiance J tend to be zero:

$$J^{dark}(\mathbf{x}) = \min_c(\min_{\mathbf{y}\in\Omega(\mathbf{x})}(J^c(\mathbf{y}))) = 0. \quad (4)$$

As A^c is always positive, this leads to:

$$\min_c(\min_{\mathbf{y}\in\Omega(\mathbf{x})}(\frac{J^c(\mathbf{y})}{A^c})) = 0 \quad (5)$$

Putting Equation (3) into Equation (4), we can estimate the transmission t ˜simply by:

$$\tilde{t}(\mathbf{x}) = 1 - \min_c(\min_{\mathbf{y}\in\Omega(\mathbf{x})}(\frac{I^c(\mathbf{y})}{A^c})). \quad (6)$$

In fact, $\min_c(\min_{\mathbf{y}\in\Omega(\mathbf{x})}(\frac{I^c(\mathbf{y})}{A^c}))$ is the dark channel of the normalized haze image $\frac{I^c(\mathbf{y})}{A^c}$. It

directly provides the estimation of the transmission. As mentioned before, the dark channel prior is not a good prior for the sky regions. Fortunately, the color of the sky is usually very similar to the atmospheric light A in a haze image

$$\min_c(\min_{\mathbf{y}\in\Omega(\mathbf{x})}(\frac{I^c(\mathbf{y})}{A^c})) \to 1, \text{ and } \tilde{t}(\mathbf{x}) \to 0 \qquad (7)$$

in the sky regions. Since the sky is at infinite and tends to has zero transmission, the Equation (6) gracefully handles both sky regions and non-sky regions. There is no need to separate the sky regions beforehand.

In practice, even in clear days the atmosphere is not absolutely free of any particle. So, the haze still exists in distant objects. Moreover, the presence of haze is a fundamental cue for human to perceive depth [3, 13]. This phenomenon is called aerial perspective. If re- move the haze thoroughly, the image may seem unnatural and the feeling of depth may be lost. So optionally keep a very small amount of haze for the distant objects by introducing a constant parameter ω ($0<\omega$ 1) into Equation

$$\tilde{t}(\mathbf{x}) = 1 - \omega \min_c(\min_{\mathbf{y}\in\Omega(\mathbf{x})}(\frac{I^c(\mathbf{y})}{A^c})). \qquad (8)$$

The nice property of this modification is to adaptively keep more haze for the distant objects. The value of ω is application-based. It is fixed to 0.95 for all results reported. It is reasonably good but contains some block effects since the transmission is not always constant in a patch. In the next subsection, this map is refined using a soft matting method.

Soft Matting

The haze imaging Equation $I(x) = J(x)t(x) + A(1 - t(x))$, has a similar form with the image matting equation. A transmission map is exactly an alpha map. Therefore, apply a soft matting to refine the transmission. Denote the refined transmission map by t(x). Rewriting t(x) and t ˜(x) in their vector form as t and ˜t, minimize the following cost function

$$E(\mathbf{t}) = \mathbf{t}^T\mathbf{L}\mathbf{t} + \lambda(\mathbf{t} - \tilde{\mathbf{t}})^T(\mathbf{t} - \tilde{\mathbf{t}}). \qquad (9)$$

where L is the Matting Laplacian matrix , and λ is a regularization parameter. The first term is the smooth term and the second term is the data term.

The (i,j) element of the matrix L is defined as:

$$\sum_{k|(i,j)\in w_k} (\delta_{ij} - \frac{1}{|w_k|}(1+(\mathbf{I}_i-\mu_k)^T(\Sigma_k+\frac{\varepsilon}{|w_k|}U_3)^{-1}(\mathbf{I}_j-\mu_k))) \qquad (10)$$

where Ii and Ij are the colors of the input image I at pixels i and j, δij is the Kronecker delta, μk and Σk are the mean and covariance matrix of the colors in window wk, U3 is a 3×3 identity matrix, ε is a regularizing parameter, and |wk| is the number of pixels in the window wk. The optimal t can be obtained by solving the following sparse linear system:

$$(L + \lambda U)t = \lambda \tilde{t} \qquad (11)$$

where U is an identity matrix of the same size as L. Here, set a small value on λ (10−4) so that t is softly constrained by ˜t.

Levin's soft matting method has also been applied by Hsu to deal with the spatially variant white balance problem. In both Levin's and Hsu's works, the ˜t is only known in sparse regions and the matting is mainly used to extrapolate the value into the unknown region. The soft matting is used to refine a coarser ˜t which has already filled the whole image.

Recovering the Scene Radiance

With the transmission map, it can recover the scene radiance according to Equation (1). But the direct attenuation term J(x)t(x) can be very close to zero when the transmission t(x) is close to zero. The directly recovered scene radiance J is prone to noise. Therefore, restrict the transmission t(x) to a lower bound t0, which means that a small amount of haze are preserved in very dense haze regions. The final scene radiance J(x) is recovered by

$$\mathbf{J}(\mathbf{x}) = \frac{\mathbf{I}(\mathbf{x}) - \mathbf{A}}{\max(t(\mathbf{x}), t_0)} + \mathbf{A}. \qquad (12)$$

A typical value of t0 is 0.1. Since the scene radiance is usually not as bright as the atmospheric light, the image after haze removal looks dim. So, increase the exposure of J(x) for display.

Estimating the Atmospheric Light

In most of the previous single image methods, the atmospheric light A is estimated from the most haze-opaque pixel. For example, the pixel with highest intensity is used as the atmospheric light and is furthered refined. But in real images, the brightest pixel could be on a white car or a white building. As discussed in the dark channel of a haze image approximates the haze denseness well . The dark channel is used to improve the atmospheric light estimation. First the top 0.1% brightest pixels in the dark channel are picked. These pixels are most hazeopaque. Among these pixels, the pixels with highest intensity in the input image are selected as the atmospheric light. These pixels may not be brightest in the whole image. This simple method based on the dark channel prior is more robust than the ”brightest pixel” method. This is used to automatically estimate the atmospheric lights for all images shown in the project.

6. EVALUATION AND RESULTS

6.1 Test Case-1: Setup of the Project and Server Home Page

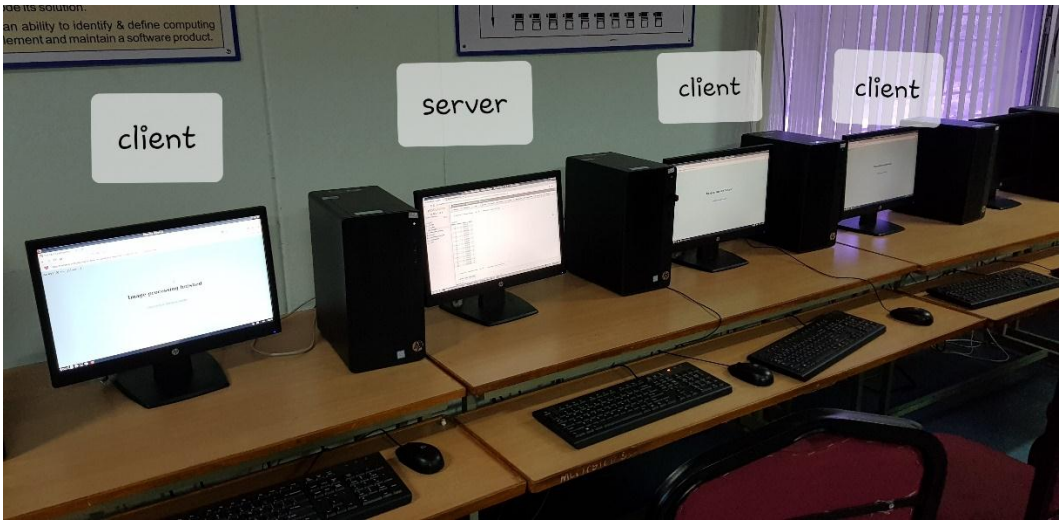

Fig: 6.1 Setup of the Project

Fig: 6.2 Server Home Page

The Server Home page is used to upload unhazed images to the server which in turn are stored in the database with respective to the upload time of the image and status of the images.

6.2 Test Case-2: Preprocessing Intimation Page

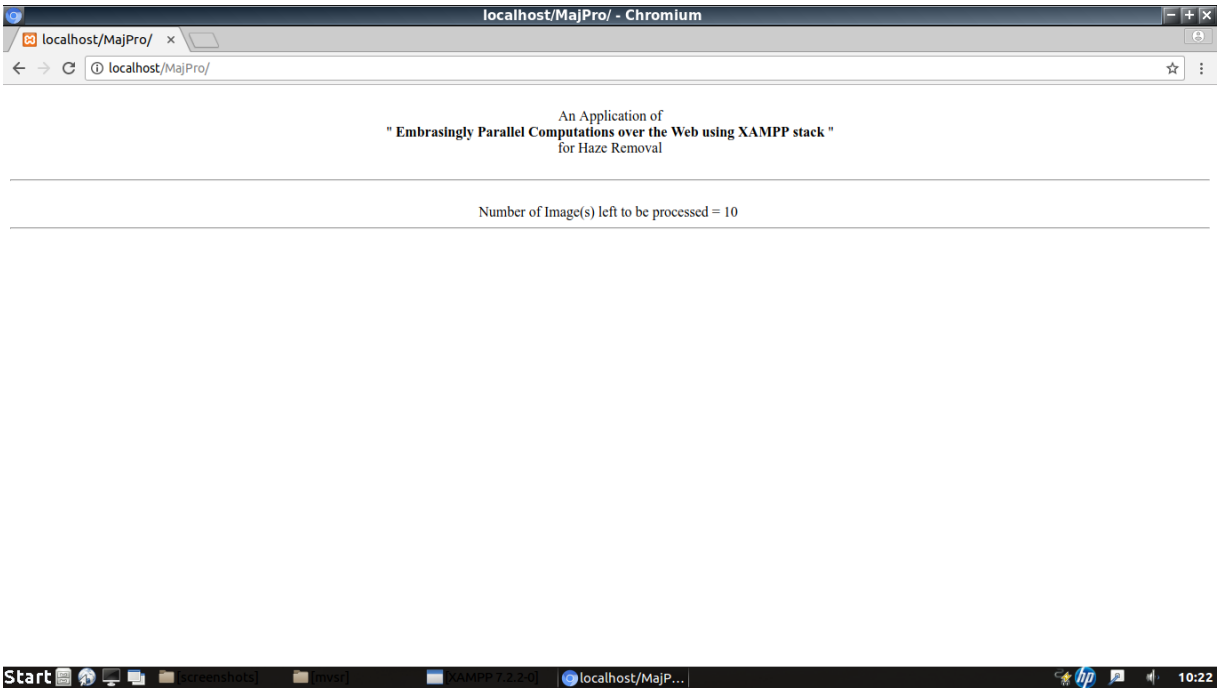

Fig: 6.3 Preprocessing Intimation Page

This page tells us about how many images are yet to be processed as it keeps on tracking the status of the images from the database.

6.3 Test Case-3: Client Home Page

Fig: 6.4 Client Home Page

The Client Home page is used by other clients to run process parallelly and it automatically takes images from the database for the processing.

6.4 Test Case-4: Dehazing Process

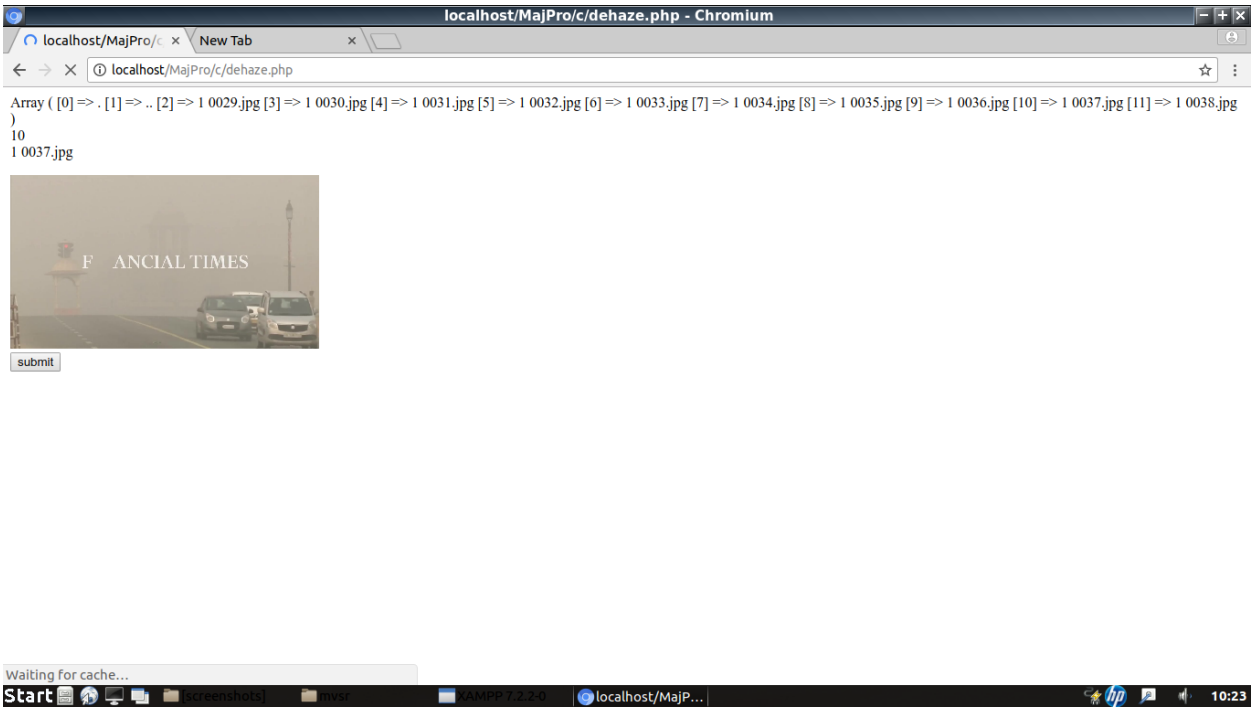

Fig: 6.5 Dehazing process

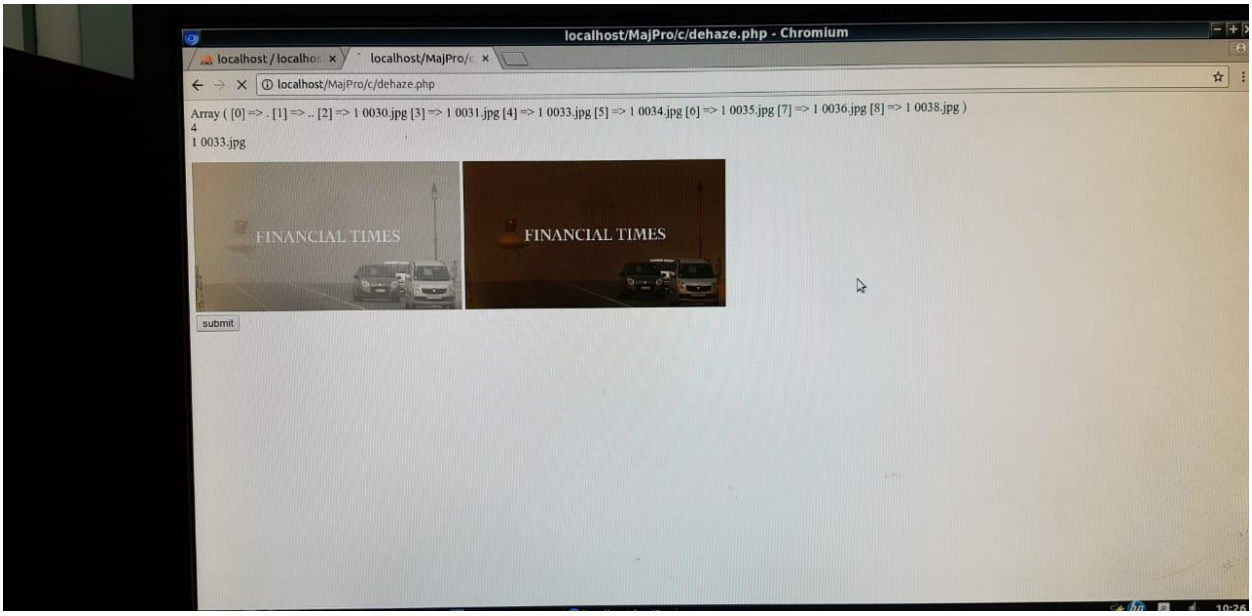

Fig: 6.6 Dehazed image

The image is dehazed in this process and stored back in the database and is ready for the customer to download the images.

6.5 Test Case-5: View of Stored images in Database

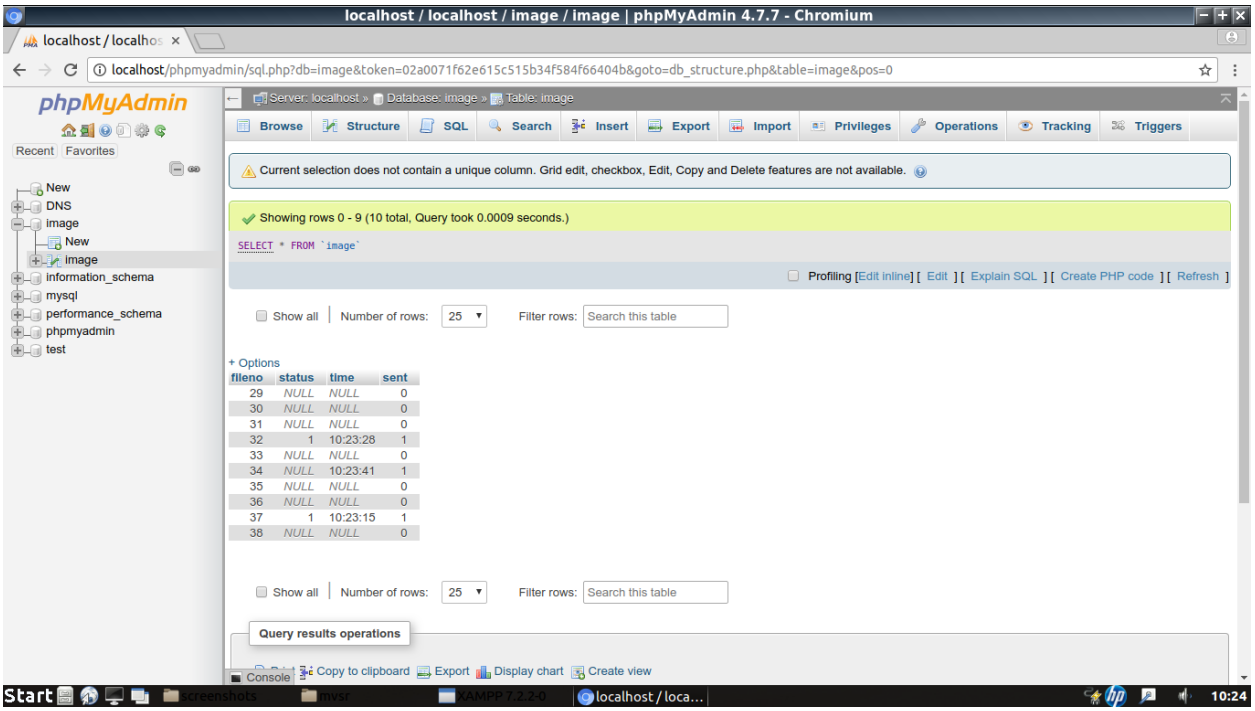

Fig: 6.7 View of Stored images in Database

The images are stored in the database with status and sent values so that it can check if all images are processed and completed successfully or not.

6.6 Test Case-6: Image Process Status

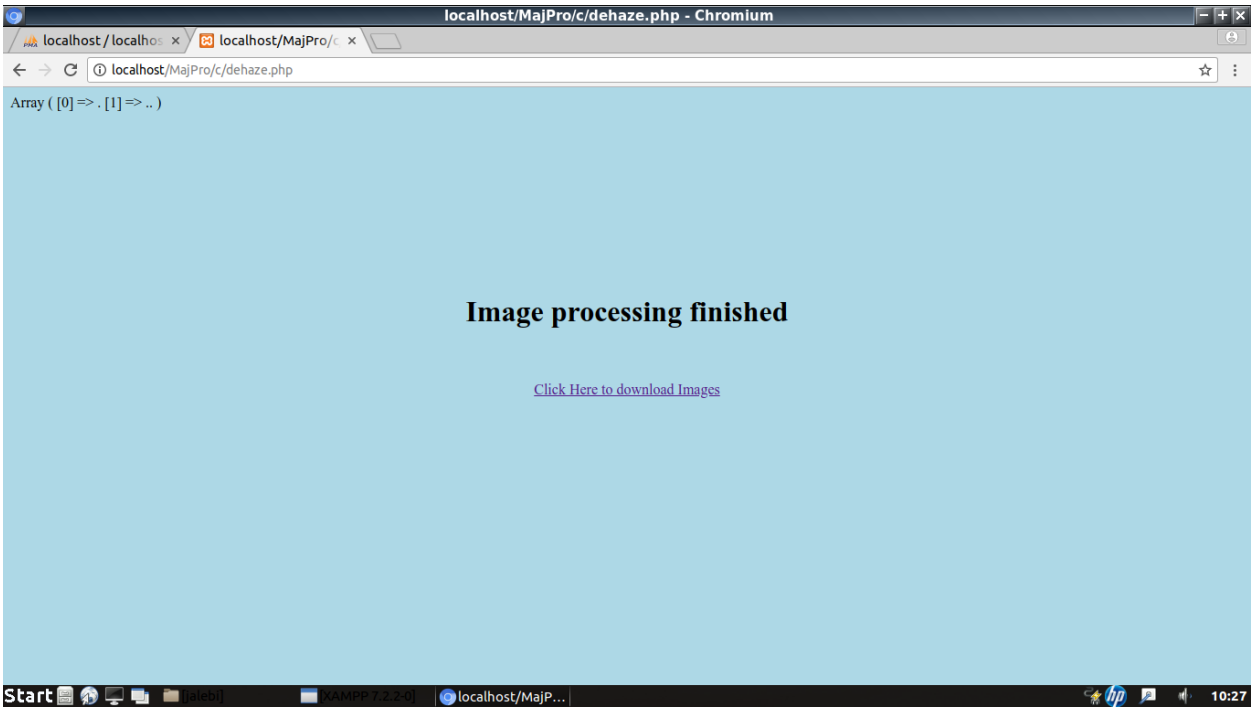

Fig: 6.8 Image Process Status

The status of the hazed images is known here that is whether the images are still processing or whether the process is finished.

6.7 Test Case-7: Downloading and Deletion Phase

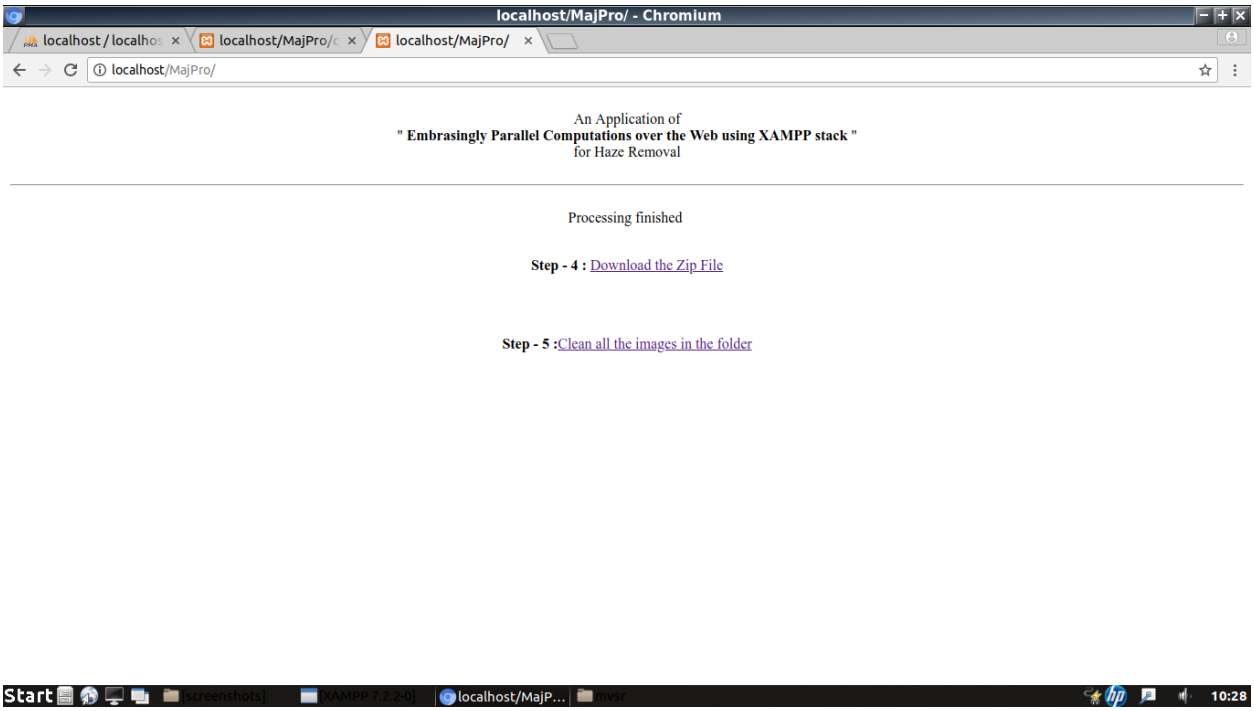

Fig: 6.9 Downloading and Deletion Phase

This phase is used for downloading the zip file of all the dehazed images and from here the folder and database can be cleaned.

6.8 Test Case-8: Dehazed images to video

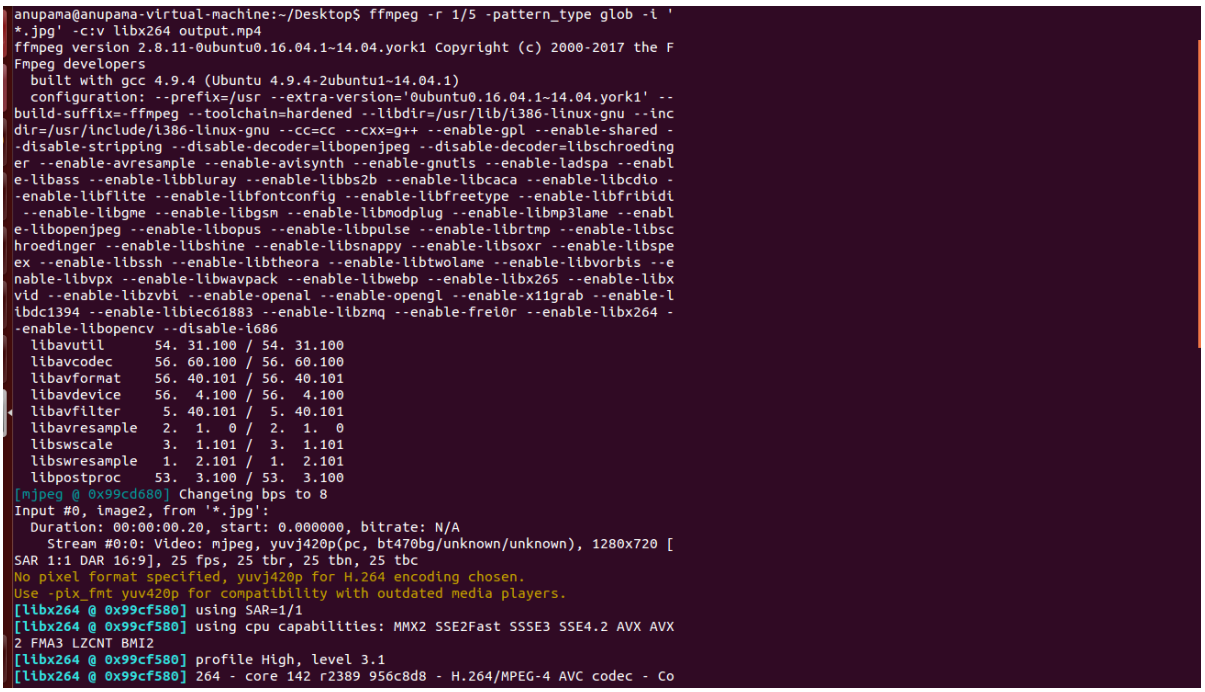

Fig: 6.10 images to video-1

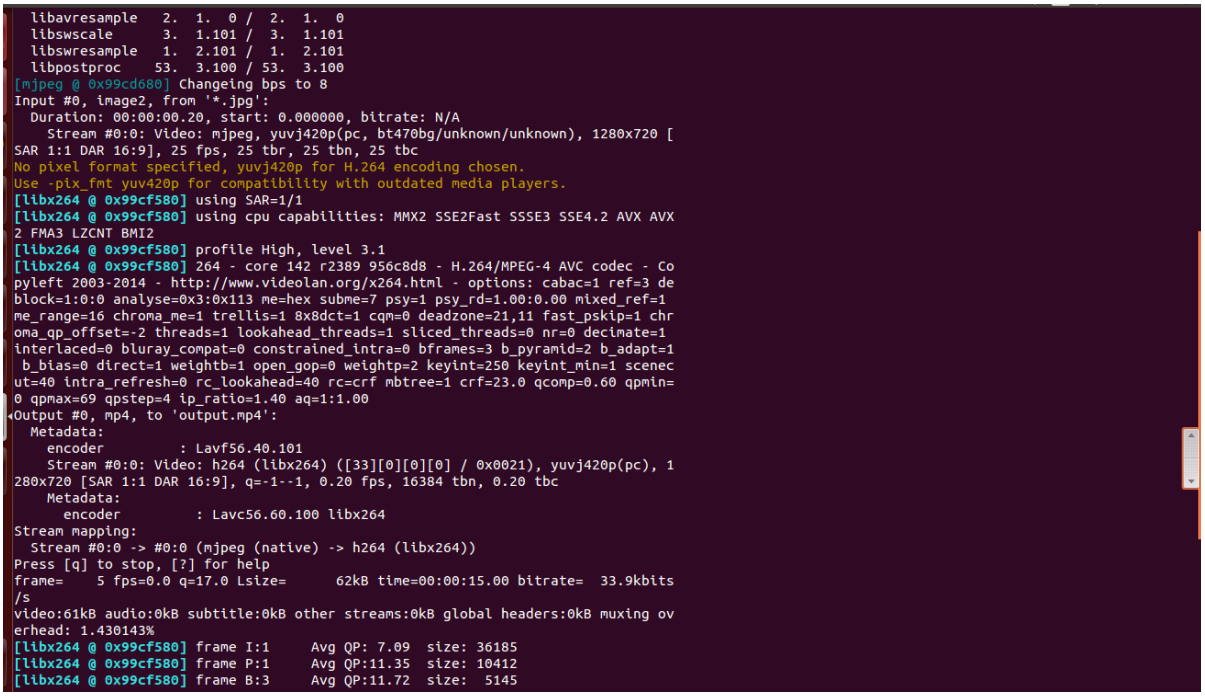

Fig: 6.11 images to video-2

```
interlaced=0 bluray_compat=0 constrained_intra=0 bframes=3 b_pyramid=2 b_adapt=1
 b_bias=0 direct=1 weightb=1 open_gop=0 weightp=2 keyint=250 keyint_min=1 scenec
ut=40 intra_refresh=0 rc_lookahead=40 rc=crf mbtree=1 crf=23.0 qcomp=0.60 qpmin=
0 qpmax=69 qpstep=4 ip_ratio=1.40 aq=1:1.00
Output #0, mp4, to 'output.mp4':
  Metadata:
    encoder         : Lavf56.40.101
    Stream #0:0: Video: h264 (libx264) ([33][0][0][0] / 0x0021), yuvj420p(pc), 1
280x720 [SAR 1:1 DAR 16:9], q=-1--1, 0.20 fps, 16384 tbn, 0.20 tbc
    Metadata:
      encoder         : Lavc56.60.100 libx264
Stream mapping:
  Stream #0:0 -> #0:0 (mjpeg (native) -> h264 (libx264))
Press [q] to stop, [?] for help
frame=    5 fps=0.0 q=17.0 Lsize=      62kB time=00:00:15.00 bitrate=  33.9kbits
/s
video:61kB audio:0kB subtitle:0kB other streams:0kB global headers:0kB muxing ov
erhead: 1.430143%
[libx264 @ 0x99cf580] frame I:1     Avg QP: 7.09  size: 36185
[libx264 @ 0x99cf580] frame P:1     Avg QP:11.35  size: 10412
[libx264 @ 0x99cf580] frame B:3     Avg QP:11.72  size:  5145
[libx264 @ 0x99cf580] consecutive B-frames: 20.0%  0.0%  0.0% 80.0%
[libx264 @ 0x99cf580] mb I  I16..4: 32.3% 59.9%  7.8%
[libx264 @ 0x99cf580] mb P  I16..4:  1.3% 11.5%  0.7%  P16..4: 10.5%  6.2%  4.4%
  0.0%  0.0%    skip:65.4%
[libx264 @ 0x99cf580] mb B  I16..4:  0.1%  2.0%  0.1%  B16..8: 15.4%  4.3%  2.0%
  direct: 3.1%  skip:72.9%  L0:40.2% L1:49.0% BI:10.8%
[libx264 @ 0x99cf580] 8x8 transform intra:64.4% inter:64.1%
[libx264 @ 0x99cf580] coded y,uvDC,uvAC intra: 56.5% 24.2% 10.0% inter: 7.7% 5.2
% 0.8%
[libx264 @ 0x99cf580] i16 v,h,dc,p: 40% 42% 16%  2%
[libx264 @ 0x99cf580] i8 v,h,dc,ddl,ddr,vr,hd,vl,hu: 24% 38% 31%  1%  1%  1%  2%
  1%  1%
[libx264 @ 0x99cf580] i4 v,h,dc,ddl,ddr,vr,hd,vl,hu: 37% 44%  9%  1%  2%  1%  4%
  1%  1%
[libx264 @ 0x99cf580] i8c dc,h,v,p: 64% 22% 12%  2%
[libx264 @ 0x99cf580] Weighted P-Frames: Y:0.0% UV:0.0%
[libx264 @ 0x99cf580] ref B L0: 98.1%  1.9%
[libx264 @ 0x99cf580] ref B L1: 98.1%  1.9%
[libx264 @ 0x99cf580] kb/s:19.85
anupama@anupama-virtual-machine:~/Desktop$
```

Fig: 6.12 images to video-3

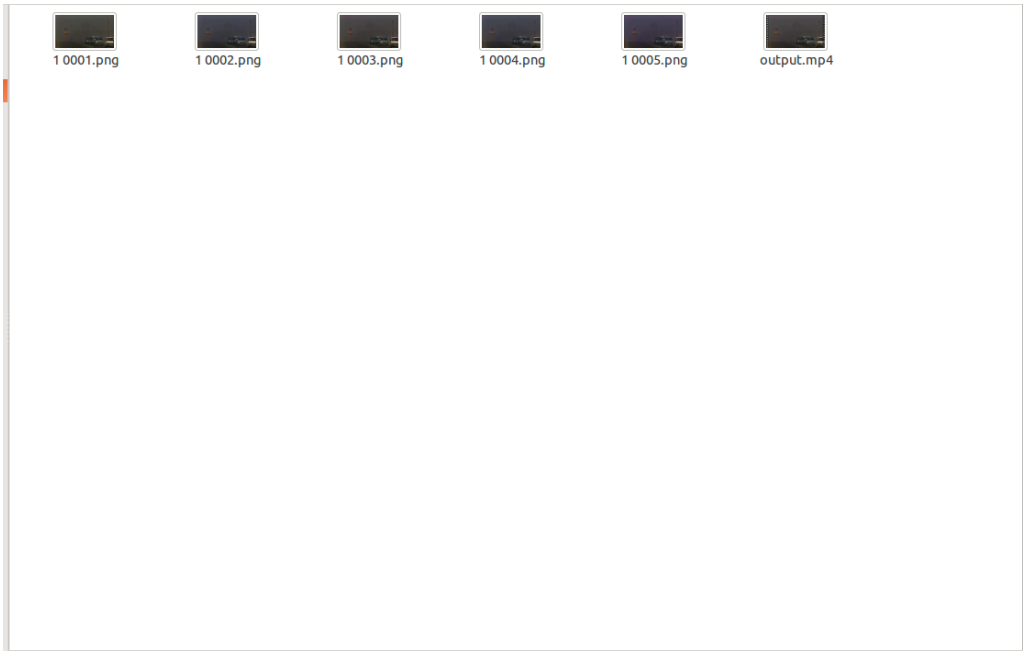

Fig: 6.13 images to video-4

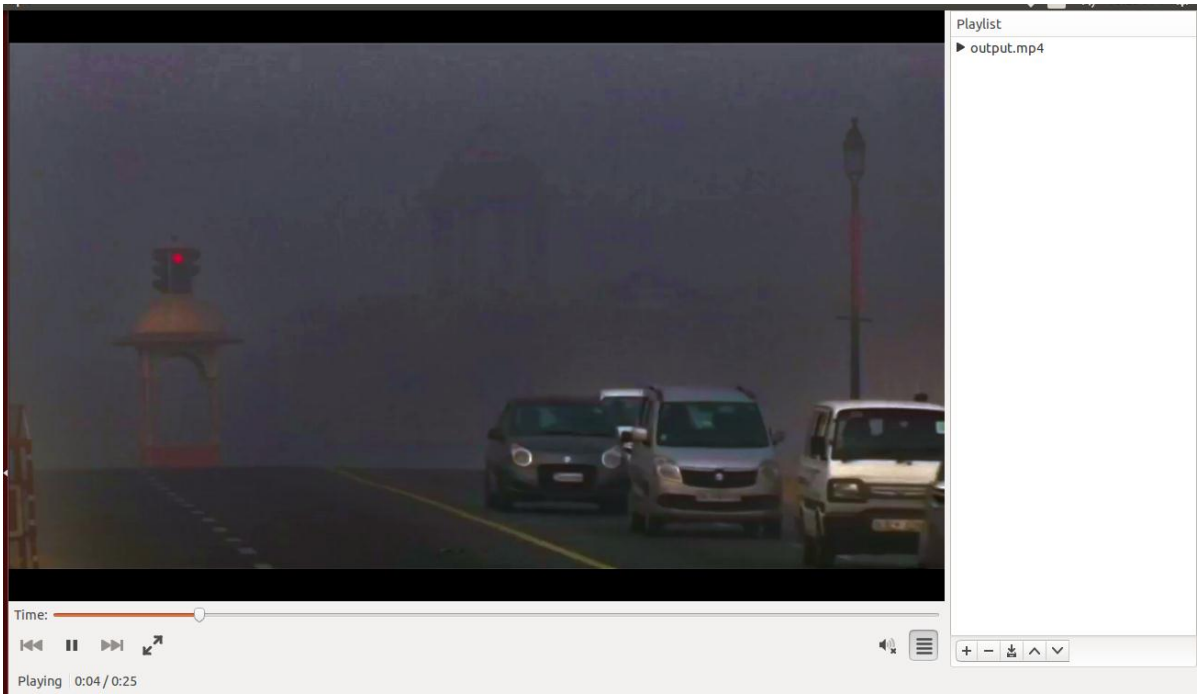

Fig: 6.14 Demonstration of the video file

6.9 Experimental Results

The following hardware and software setup was used for experimentation of the proposed solution:-

Hardware Configuration

Processor : 4x Intel(R) Core(TM) i3-7100 CPU @ 3.90GHz
Memory : 8060MB (1575MB used)

Software Platform

Kernel : Linux 4.13.0-36-generic (x86_64)
Compiled : #40~16.04.1-Ubuntu SMP Fri Feb 16 23:25:58 UTC 2018
Distribution : Ubuntu 16.04.4 LTS

For the experiment, the performance of the dehazing algorithm is compared on a set of images extracted from a video in the following cases:

1. Without parallel computing on a CPU
2. With parallel computing on a CPU with GPU (CUDA-NVIDIA & a third party tool)
3. With parallel computing using EPXWebX over various CPUs

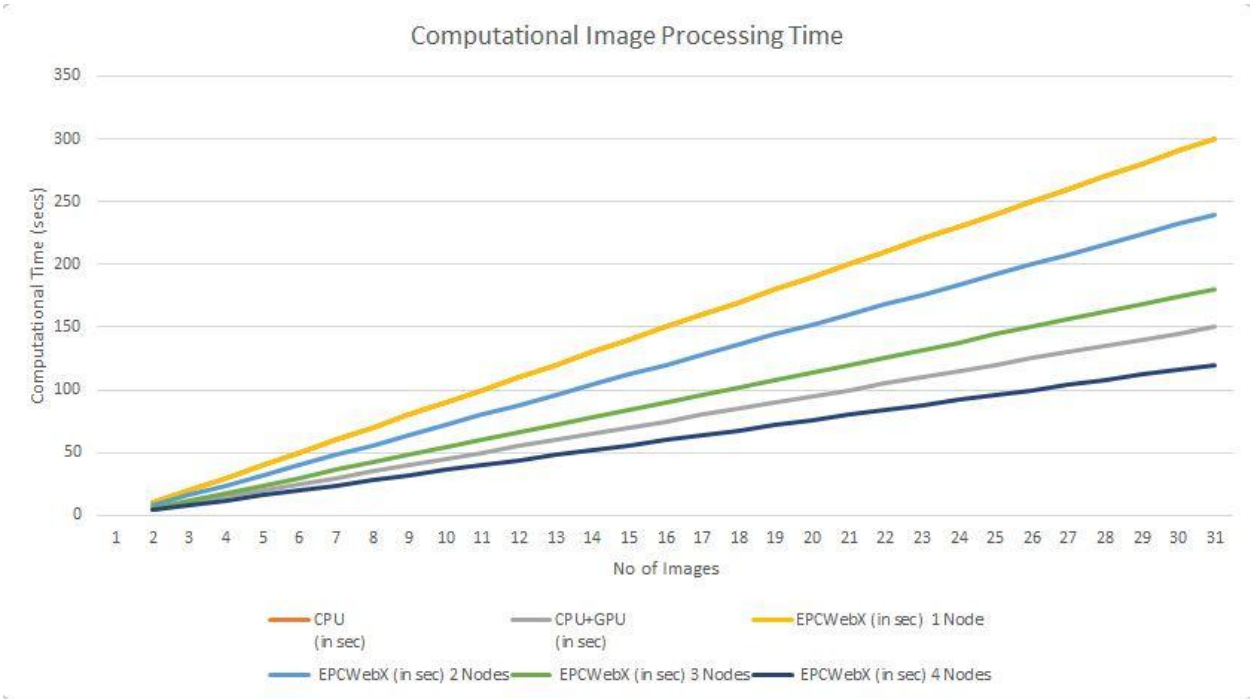

Fig: 6.15 Computational Image Processing Time

In the above graph it is seen that when CPU(without parallel computing) for computation of images is used, it takes a very long time to generate the output but when both CPU and GPU are used in parallel to perform computation of images together then it shows a drastic change in time for processing of images. In this project when EPCWebX method is used with one node (computer), it shows the same result as that of the computational time of CPU. As the no. of nodes (computers) increase and are run in parallel, the computational time slowly decreases and tries to meet the computational time of CPU+GPU. When the number of nodes are 4 then this method surpasses the computational time of the CPU+GPU running in parallel.

7. CONCLUSION & FUTURE ENHANCEMENTS

7.1 Conclusion

Parallel computing is a type of computation in which many calculations or execution of processes are carried out simultaneously. Large problems can often be divided into smaller ones, which can then be solved at a time. This project is an application for performing parallel computations by splitting a single problem into a number of independent sub problems and solving them parallely with n number of available clients. This concept is applied to dehaze the hazed images using DARK CHANNEL PRIOR Technique which is a SINGLE IMAGE HAZE REMOVAL technique.

When CPU (without parallel computing) is used for computation of hazed images it takes a very long time to generate the dehazed images as output but when both CPU and GPU are used parallely to perform computation of images together then it shows a drastic change in time for processing of images. In this project Embarrassingly Parallel Computations over the Web using XAMPP stack for Haze Removal (EPCWebX) using one node (computer), the time taken is same as that of computational time of CPU and as the no. of nodes (computers) increase and are run parallely, the computational time slowly decreases and tries to meet the computational time of CPU+GPU. When the number of nodes are 4 then this method surpasses the computational time of the CPU+GPU.

7.2 Future Enhancements

The idea of massive parallelism permeates virtually all unconventional models of computation proposed till date through examples such as DNA computing, quantum computing or reaction–diffusion computers. Even a model that is mainly of theoretical interest, like the accelerating machine, can be thought of as deriving its power from doubling the number of processing units (operating in parallel) at each step. Using a better language and separating the design of parallelization from the runtime can create wonders. Many industrial problems that require optimum solutions at lowest computational cost may be taken up for designing software packages containing hybridization of the evolved heuristics. It can also be used to solve multi objective optimization problems and nonlinear optimization problems.

REFERENCES

1. Single Image Haze Removal using Dark Channel Prior, by Kaiming He, Jian Sun, and Xiaoou Tang, in **CVPR** 2009 (**Oral, Best Paper Award**).
2. Guided Image Filtering, by Kaiming He, Jian Sun, and Xiaoou Tang, in **ECCV** 2010 (**Oral**).
3. Single Image Haze Removal using Dark Channel Prior, by Kaiming He, Jian Sun, and Xiaoou Tang, in **TPAMI** 2011 (**Spotlight Paper**).
4. Guided Image Filtering, by Kaiming He, Jian Sun, and Xiaoou Tang, n **TPAMI** 2013.
5. R. Fattal. Single image dehazing. In SIGGRAPH, pages 1–9, 2008.
6. E. Hsu, T. Mertens, S. Paris, S. Avidan, and F. Durand. Light mixture estimation for spatially varying white balance. In SIGGRAPH, pages 1–7, 2008
7. J. Kopf, B. Neubert, B. Chen, M. Cohen, D. Cohen-Or, O. Deussen, M. Uyttendaele, and D. Lischinski. Deep photo: Model-based photograph enhancement and viewing. SIGGRAPH Asia, 2008.
8. A. Levin, D. Lischinski, and Y. Weiss. A closed form solution to natural image matting. CVPR, 1:61–68, 2006
9. S. G. Narasimhan and S. K. Nayar. Chromatic framework for vision in bad weather. CVPR, pages 598–605, 2000.
10. S. G. Narasimhan and S. K. Nayar. Vision and the atmosphere. IJCV, 48:233–254, 2002.
11. S. G. Narasimhan and S. K. Nayar. Contrast restoration of weather degraded images. PAMI, 25:713–724, 2003.
12. S. G. Narasimhan and S. K. Nayar. Interactive deweathering of an image using physical models. In Workshop on Color and Photometric Methods in Computer Vision, 2003.
13. S. K. Nayar and S. G. Narasimhan. Vision in bad weather. ICCV, page 820, 1999
14. A. J. Preetham, P. Shirley, and B. Smits. A practical analytic model for daylight. In SIGGRAPH, pages 91–100, 1999.

15. Y. Y. Schechner, S. G. Narasimhan, and S. K. Nayar. Instant dehazing of images using polarization. CS. Shwartz, E. Namer, and Y. Y. Schechner. Blind haze separation. CVPR, 2:1984–1991, 2006.VPR, 1:325, 2001.
16. R. Tan. Visibility in bad weather from a single image. CVPR, 2008.
17. M. van Herk. A fast algorithm for local minimum and maximum filters on rectangular and octagonal kernels. Pattern Recogn. Lett., 13:517–521, 1992
18. OpenCV JS Haze https://ganwenyao.github.io/opencv_js/openCV_js_demo2_hazeRemove/hazeRemove.html
19. Dehaze - http://www.jiansun.org/papers/Dehaze_CVPR2009.pdf
20. Canvas to Image - https://www.askingbox.com/tutorial/send-html5-canvas-as-image-to-server
21. PHP Multiple File Upload - https://makitweb.com/multiple-files-upload-at-once-with-php/
22. JavaScript Tutorial - https://www.w3schools.com/js/
23. SQL DataBaseTutorial - ttps://www.w3schools.com/sql/sql_create_db.asp
24. Single Image Haze Removal - http://kaiminghe.com/cvpr09/
25. PHP Multiple File Upload - https://makitweb.com/multiple-files-upload-at-once-with-php/
26. OpenCV JS - https://github.com/ganwenyao/opencv_js/tree/master/docs
27. Util JS - http://directwebremoting.org/dwr/documentation/browser/util/index.html
28. OpenCV JS Tutorial - https://docs.opencv.org/3.4/d5/d10/tutorial_js_root.html
29. OpenCV JS Tutorial - https://scotch.io/tutorials/introduction-to-computer-vision-in-javascript-using-opencvjs
30. FFmpeg Tutorial - https://linuxize.com/post/how-to-install-ffmpeg-on-ubuntu-18-04/

APPENDIX

A. Source Code

A.1. Code Deployed over Server System

A.1.1. index.php

```
<center>

<br>An Application of<br>
" <b>Embrasingly Parallel Computations over the Web using XAMPP
stack</b> "<br>
for Haze Removal<br>

<br><hr><center><br>

<?php

$page = $_SERVER['PHP_SELF'];
$sec = "5";
header("Refresh: $sec; url=$page");
$dir = "c/in/";
$a = scandir($dir); //Sort in ascending order
//print_r($a);

$dir2 = "c/out/";
$b = scandir($dir2); //Sort in ascending order
//print_r($b);

if(count($a)<=2 && count($b)>2)
{
echo "Processing finished <br><br> ";
/*$page = $_SERVER['PHP_SELF'];
$sec = "10";
header("Refresh: $sec; url=$page");*/
?>

<p><b>Step - 4 : </b><a href="zip.php">Download the Zip
File</a></p><br><br>
<p><b>Step - 5 :</b><a href="delete.php">Clean all the images in the
folder</a></p>

<?php
die();
}

if(count($a)<=2)
{
```

```
?>

<p><b>Step - 1 : Extract all the Frames from a Fog Video</b><br>
(Hint: You may use the following free converter)<br>
https://www.dvdvideosoft.com/products/dvd/Free-Video-to-JPG-
Converter.htm</p><br>

<p><b>Step - 2 : </b><a href="in.html">Click here to UPLOAD ALL the
extracted FRAMES</a></p><br>

<?php
}
else {


echo "Number of Image(s) left to be processed = ".(count($a)-2);

$dbHost     = 'localhost';
        $dbUsername = 'root';
        $dbPassword = '';
        $dbName     = 'image';


        //Create connection and select DB
        $db = new mysqli($dbHost, $dbUsername, $dbPassword, $dbName);

        // Check connection
        if($db->connect_error){
            die("Connection failed: " . $db->connect_error);
            }
            $dir = "c/in/";

/*$a = scandir($dir);

            for ($x = 2; $x < count($a); $x++) {
           $i = $a[$x];
                $i = (int)substr($i, 2, 4);
       $sql = "INSERT INTO `image` (`fileno`, `status` ) VALUES ('$i',
NULL)";
       mysqli_query($db,$sql);
      }
      */

}
//echo $i;
?>

<br><hr>
```

A.1.2. in.php

```
<?php
//include 'db.php';
$dbHost     = 'localhost';
        $dbUsername = 'root';
        $dbPassword = '';
        $dbName     = 'image';


        //Create connection and select DB
        $db = new mysqli($dbHost, $dbUsername, $dbPassword, $dbName);

        // Check connection
        if($db->connect_error){
            die("Connection failed: " . $db->connect_error);
            }

if(isset($_POST['submit'])){
 // Count total files
 $countfiles = count($_FILES['file']['name']);

 // Looping all files
 for($i=0;$i<$countfiles;$i++){
   $filename = $_FILES['file']['name'][$i];
   //$status=0;
   //$insert ="INSERT INTO `image` (`fileno`, `status`, `time`) VALUES
('$x', NULL, CURRENT_TIMESTAMP)";
   // Upload file

move_uploaded_file($_FILES['file']['tmp_name'][$i],'c/in/'.$filename);


 }
 $dir = "c/in/";

$a = scandir($dir);

            for ($x = 2; $x < count($a); $x++) {
           $i = $a[$x];
                $i = (int)substr($i, 2, 4);
       $sql = "INSERT INTO `image` (`fileno`, `status` ) VALUES ('$i',
NULL)";
       mysqli_query($db,$sql);
     }

/*
ignore_user_abort(1); // run script in background
set_time_limit(0); // run script forever
$interval=60*1; // do every 15 minutes...
do{
```

```
   sleep($interval); // wait 15 minutes
}while(true);
*/
}
header("Location: .");
?>
```

A.1.3. db.php

```
<?php
$dbHost     = 'localhost';
$dbUsername = 'root';
$dbPassword = '';
$dbName     = 'image';


        //Create connection and select DB
$db = new mysqli($dbHost, $dbUsername, $dbPassword, $dbName);

        // Check connection
if($db->connect_error){
die("Connection failed: " . $db->connect_error);
}
?>
```

A.1.4. delete.php

```
<html>
<head>
<meta http-equiv="refresh" content="2; URL=http://localhost/MajPro/">
<meta name="keywords" content="automatic redirection">
</head>
<body>
<h1 style="text-align:center;">Deleting Files</h1>
<p style="text-align:center;">redirecting in 2 seconds</p>

</body>
</html>



<?php
        $dbHost     = 'localhost';
        $dbUsername = 'root';
        $dbPassword = '';
        $dbName     = 'image';
```

```
        //Create connection and select DB
        $db = new mysqli($dbHost, $dbUsername, $dbPassword, $dbName);

        // Check connection
        if($db->connect_error){
            die("Connection failed: " . $db->connect_error);
            }
$sql = "delete from image";
mysqli_query($db,$sql);

    $files = glob('c/out/*'); //get all file names
foreach($files as $file){
    if(is_file($file))
    unlink($file); //delete file
}
//header("Location: index.php");
?>
```

A.2. Code Deployed over Client System

A.2.1. index.php

```
<html>

<br>
<br>
<br>
<br>
<br>
<br>
<br>
<br>
<br>
<br>
<center><h1> <a href="dehaze.php">Click here to ENROLL this Machine
into Haze Removal Process</a></h1></center>
</html>
```

A.2.2. dehaze.php

```
<?php
//include 'db.php';
$dbHost     = 'localhost';
$dbUsername = 'root';
$dbPassword = '';
$dbName     = 'image';
```

```
        //Create connection and select DB
$db = new mysqli($dbHost, $dbUsername, $dbPassword, $dbName);

        // Check connection
if($db->connect_error){
die("Connection failed: " . $db->connect_error);
}
$dir = "in/";

$a = scandir($dir);
print_r($a);
loop:
if(count($a)<=2)
{
//$page = $_SERVER['PHP_SELF'];
//$sec = "10";
//header("Refresh: $sec; url=$page");
//header("Location: opt/lampp/htdocs/MajPro/index.php/ ");
?>
<!DOCTYPE html>
<html>
<body bgcolor="#ADD8E6">
<br>
<br>
<br>
<br>
<br>
<br>
<br>
<br>
<br>
<br>
<br>

<center><h1>   Image processing finished </h1></center>
<br>
<br>
<center><a href="finish.php">Click Here to download
Images</a></center>
</body>
</html>

<?php
die();

}

$i = rand(2,count($a)-1);
$x = $a[$i];
$x = (int)substr($x, 2, 4);
```

```
$sql = "SELECT sent from image where fileno = '$x'";
$return = mysqli_query($db,$sql);
$row = $return->fetch_assoc();
//echo $return;
if ($row['sent'] == '0')
{
echo "<br>".$i."<br>";
echo $a[$i]."<br>";
$sq = "UPDATE image SET sent = '1',time = CURRENT_TIMESTAMP() WHERE
fileno = '$x'";
mysqli_query($db,$sq);
}
else {
      $sq = "select time from image where fileno = '$x'";
      $ti = mysqli_query($db,$sq);

      $ro = $ti->fetch_assoc();
      $mro = (int)substr($ro['time'], 3, 5);
      $sro = (int)substr($ro['time'], 6, 9);
      $sumro = $mro*60+$sro;
      $dt = new DateTime("now", new DateTimeZone('Asia/Kolkata'));
   $d = $dt->format('H:i:s');
$ctm = (int)substr($d, 3, 5);
      $cts = (int)substr($d, 6, 9);
      $sumc = $ctm*60+$cts;
      $thres = $sumc - $sumro;
      if($thres < 10)
      goto loop;
}
?>

<script src="lib/utils.js"></script>
<script async="" src="lib/opencv.js" id="opencvjs"></script>

<body onload="hazeRemoveExecuteCode()">

<textarea rows="18" cols="100" id="hazeRemoveTestCode"
spellcheck="false" style="display:none">
function quickSort(s, l, r, k) {
    if (l &lt; r) {
        let i = l, j = r;
        let x = s[l];
        while (i &lt; j) {
            while (i &lt; j &amp;&amp; s[j][2] &lt; x[2])
                --j;
            if (i &lt; j)
            { s[i++] = s[j]; }
            while (i &lt; j &amp;&amp; s[i][2] &gt;= x[2])
                ++i;
            if (i &lt; j)
                s[j--] = s[i];
        }
```

```
        s[i] = x;
        if (i &gt; k) {
            quickSort(s, l, i - 1, k);
        }
        if (k &gt; i) {
            quickSort(s, i + 1, r, k - i);
        }
    }
}

function Producedarkimg(image, w) {
    // Get channel min image
    let min = 255;
    let radius = Math.round((w - 1) / 2);
    let row = image.rows;
    let col = image.cols;
    let darkImg = new cv.Mat(row, col, cv.CV_8UC1);
    let B, G, R;

    for (let i = 0; i &lt; row; ++i) {
        for (let j = 0; j &lt; col; ++j) {
            R = image.ucharPtr(i, j)[0];
            G = image.ucharPtr(i, j)[1];
            B = image.ucharPtr(i, j)[2];
            min = min &gt; R ? R : min;
            min = min &gt; B ? B : min;
            min = min &gt; G ? G : min;
            darkImg.ucharPtr(i, j)[0] = min;
            min = 255;
        }
    }

    // Get window min image
    let rowMinImg = new cv.Mat(row, col, cv.CV_8UC1);
    let darkFilterImg = new cv.Mat(row, col, cv.CV_8UC1);

    for(let j = 0; j &lt; row; ++j) {
        L = [];
        min = 255;
        for (let i = 0; i &lt; radius; ++i) {
            if (darkImg.ucharPtr(j, i)[0] &lt; min)
                min = darkImg.ucharPtr(j, i)[0];
            rowMinImg.ucharPtr(j, i)[0] = min;
        }
        for (let i = 1; i &lt; col; ++i) {
            if (i &gt;= w) {
                rowMinImg.ucharPtr(j, i - radius - 1)[0] =
darkImg.ucharPtr(j, L.length &gt; 0 ? L[0] :i - 1)[0];
            }
            if (darkImg.ucharPtr(j, i)[0] &gt; darkImg.ucharPtr(j, i -
1)[0]) {
                L.push(i - 1);
```

```
                if (i == w + L[0])
                    L.shift();
            }
            else {
                while (L.length &gt; 0) {
                    if (darkImg.ucharPtr(j, i)[0] &gt;=
darkImg.ucharPtr(j, L[L.length - 1])[0]) {
                        if (i == w + L[0])
                            L.shift();
                        break ;
                    }
                    L.pop();
                }
            }
        }
        rowMinImg.ucharPtr(j, col - radius - 1)[0] =
darkImg.ucharPtr(j, L.length &gt; 0 ? L[0] : col - 1)[0];
        min = 255;
        for (let i = col - 1; i &gt;= col - radius; --i) {
            if (darkImg.ucharPtr(j, i)[0] &lt; min)
                min = darkImg.ucharPtr(j, i)[0];
            rowMinImg.ucharPtr(j, i)[0] = min;
        }
    }

    for(let j = 0; j &lt; col; ++j) {
        L = [];
        min = 255;
        for (let i = 0; i &lt; radius; ++i) {
            if (rowMinImg.ucharPtr(i, j)[0] &lt; min)
                min = rowMinImg.ucharPtr(i, j)[0];
            darkFilterImg.ucharPtr(i, j)[0] = min;
        }
        for (let i = 1; i &lt; row; ++i) {
            if (i &gt;= w) {
                darkFilterImg.ucharPtr(i - radius - 1, j)[0] =
rowMinImg.ucharPtr(L.length &gt; 0 ? L[0] : i - 1, j)[0];
            }
            if (rowMinImg.ucharPtr(i, j)[0] &gt; rowMinImg.ucharPtr(i
- 1, j)[0]) {
                L.push(i - 1);
                if (i == w + L[0])
                    L.shift();
            }
            else {
                while (L.length &gt; 0) {
                    if (rowMinImg.ucharPtr(i, j)[0] &gt;=
rowMinImg.ucharPtr(L[L.length - 1], j)[0]) {
                        if (i == w + L[0])
                            L.shift();
                        break;
                    }
```

```
                    L.pop();
                }
            }
        }
        darkFilterImg.ucharPtr(row - radius - 1, j)[0] =
rowMinImg.ucharPtr(L.length &gt; 0 ? L[0] : row - 1, j)[0];
        min = 255;
        for (let i = row - 1; i &gt;= row - radius; --i) {
            if (rowMinImg.ucharPtr(i, j)[0] &lt; min)
                min = rowMinImg.ucharPtr(i, j)[0];
            darkFilterImg.ucharPtr(i, j)[0] = min;
        }
    }
    rowMinImg.delete();
    darkImg.delete();
    return darkFilterImg;
}

function getatmospheric_light(image, darkimg)
 {
    // Get A
    let row = darkimg.rows;
    let col = darkimg.cols;
    let darksize = row * col;
    let topsize = Math.ceil(darksize / 1000);
    let A = Array(3);
    let allpixels = Array(row * col);

    for (let i = 0; i &lt; row; i++)
       for (let j = 0;j &lt; col;j++)
            {
                let Pixel = [i, j, darkimg.ucharPtr(i, j)[0]];
                allpixels[i * col + j] = Pixel;
            }

    // Sort k maximum number
    let tmp = allpixels[0];
    allpixels[0] = allpixels[allpixels.length - 1];
    allpixels[allpixels.length - 1] = tmp;
    quickSort(allpixels, 0, allpixels.length - 1, topsize);

    let R, G, B, sum;
    let max = 0, maxi, maxj, x, y;
    for (let i = 0; i &lt; topsize; i++) {
       x = allpixels[i][0];
       y = allpixels[i][1];
       R = image.ucharPtr(x, y)[0];
       G = image.ucharPtr(x, y)[1];
       B = image.ucharPtr(x, y)[2];
       sum = R * 0.299 + G * 0.587 + B * 0.114;
       if (max &lt; sum) {
           max = sum;
```

```
            maxi = x;
            maxj = y;
        }
    }
    A[0] = image.ucharPtr(maxi, maxj)[0];
    A[1] = image.ucharPtr(maxi, maxj)[1];
    A[2] = image.ucharPtr(maxi, maxj)[2];
    return A;
}

function getTransmission_dark(image, darkimg, A)
{
    let avg_A = (A[0] + A[1] + A[2]) / 3.0;
    let w = 0.95;
    let row = darkimg.rows;
    let col = darkimg.cols;
    let transmission = new cv.Mat(row, col, cv.CV_32FC1);
    for (let i = 0; i &lt; row; i++)
        for (let j = 0; j &lt; col; j++)
            transmission.floatPtr(i, j)[0] = 1 - w *
(darkimg.ucharPtr(i, j)[0] / avg_A);
    let gray = new cv.Mat();
    let trans = new cv.Mat();
    cv.cvtColor(image, gray, cv.COLOR_RGB2GRAY, 0);
    guidedFilter(transmission, gray, trans, 6*11, 0.001);
    gray.delete();
    transmission.delete();
    //cv.GaussianBlur(transmission, transmission, new cv.Size(11 ,11),
0);
    return trans;
}

function recover(image, trans, A, radius)
{
    let row = image.rows;
    let col = image.cols;
    let tx = trans.floatPtr(radius, radius)[0];
    let t0 = 0.1;
    let finalimg = cv.Mat.zeros(row, col, cv.CV_8UC3);
    let val = 0;
    for (let i = 0; i &lt; 3; i++)
        for (let k = 0; k &lt; row; k++)
            for(let l = 0; l &lt; col; l++) {
               tx = trans.floatPtr(k, l)[0];
               tx = tx &gt; t0 ? tx : t0;
               val = Math.round((image.ucharPtr(k, l)[i] - A[i]) / tx
+ A[i]);
               val = val &lt; 0 ? 0 : val;
               finalimg.ucharPtr(k, l)[i] = val &gt; 255 ? 255 : val;
            }
    return finalimg;
}
```

```
function guidedFilter(source, guided_image, output, radius, epsilon)
{
    let guided = guided_image.clone();

    let source_32f = new cv.Mat();
    let guided_32f = new cv.Mat();
    source.convertTo(source_32f, cv.CV_32F);
    guided.convertTo(guided_32f, cv.CV_32F);

    let let_Ip = new cv.Mat();
    let let_I2 = new cv.Mat();
    cv.multiply(guided_32f, source_32f, let_Ip);
    cv.multiply(guided_32f, guided_32f, let_I2);


    let mean_p = new cv.Mat();
    let mean_I = new cv.Mat();
    let mean_Ip = new cv.Mat();
    let mean_I2 = new cv.Mat();
    let win_size = new cv.Size(2 * radius + 1, 2 * radius + 1);
    cv.boxFilter(source_32f, mean_p, cv.CV_32F, win_size);
    cv.boxFilter(guided_32f, mean_I, cv.CV_32F, win_size);
    cv.boxFilter(let_Ip, mean_Ip, cv.CV_32F, win_size);
    cv.boxFilter(let_I2, mean_I2, cv.CV_32F, win_size);

    let cov_Ip = new cv.Mat();
    let var_I = new cv.Mat();
    cv.subtract(mean_Ip,  mean_I.mul(mean_p, 1), cov_Ip);
    cv.subtract(mean_I2,  mean_I.mul(mean_I, 1), var_I);

    for (let i = 0; i &lt; guided_image.length; ++i)
        for (let j = i + 1; j &lt; guided_image.length; ++j)
          var_I.floatPtr(i, j)[0] += epsilon;

    let a = new cv.Mat();
    let b = new cv.Mat();
    cv.divide(cov_Ip, var_I, a);
    cv.subtract(mean_p, a.mul(mean_I, 1), b);

    let mean_a = new cv.Mat();
    let mean_b = new cv.Mat();
    cv.boxFilter(a, mean_a, cv.CV_32F, win_size);
    cv.boxFilter(b, mean_b, cv.CV_32F, win_size);
    cv.add(mean_a.mul(guided_32f, 1), mean_b, output)

    guided.delete(); source_32f.delete(); guided_32f.delete();
let_Ip.delete(); let_I2.delete();
    mean_p.delete(); mean_I.delete(); mean_Ip.delete();
mean_I2.delete(); cov_Ip.delete();
    var_I.delete(); a.delete(); b.delete(); mean_a.delete();
mean_b.delete();
```

```
}

let src = cv.imread("hazeRemoveCanvasInput");
let dark = Producedarkimg(src, 7);
let A = getatmospheric_light(src, dark);
let trans = getTransmission_dark(src, dark, A);
let dst = recover(src, trans, A, 7);

cv.imshow("hazeRemoveCanvasOutput", dst);
src.delete(); dst.delete(); trans.delete(); dark.delete();

prepareImg();

document.getElementById("myForm").submit();



</textarea>

<p class="err" id="hazeRemoveErr"></p>

<canvas id="hazeRemoveCanvasInput" style="width:25%"></canvas>

<canvas id="hazeRemoveCanvasOutput" style="width:25%"></canvas>

<form id="myForm" method="post" action="out.php" >
<input name="imgname" type="hidden" value='<?php echo $a[$i]; ?>'>
<input id="inp_img" name="img" type="hidden" value="">

<button>submit</button>
</form>

<script>

function hazeRemoveExecuteCode() {
    let hazeRemoveText =
document.getElementById("hazeRemoveTestCode").value;
    try {
        eval(hazeRemoveText);
        document.getElementById("hazeRemoveErr").innerHTML = " ";
    } catch(err) {
        document.getElementById("hazeRemoveErr").innerHTML = err;
    }
}

loadImageToCanvas("in/<?php echo $a[$i]; ?>",
"hazeRemoveCanvasInput");
let hazeRemoveInputElement =
document.getElementById("hazeRemoveInput");
hazeRemoveInputElement.addEventListener("change",
hazeRemoveHandleFiles, false);
function hazeRemoveHandleFiles(e) {
```

```
    let hazeRemoveUrl = URL.createObjectURL(e.target.files[0]);
    loadImageToCanvas(hazeRemoveUrl, "hazeRemoveCanvasInput");
}
function onReady() {
    document.getElementById("hazeRemoveTryIt").disabled = false;
}
if (typeof cv !== 'undefined') {
    onReady();
} else {
    document.getElementById("opencvjs").onload = onReady;
}

function prepareImg() {
   var canvas = document.getElementById('hazeRemoveCanvasOutput');
   document.getElementById('inp_img').value = canvas.toDataURL();
}
</script>
</body>
```

A.2.3. test.php

```
<?php
 session_start();
      //Put session start at the beginning of the file



$dbHost     = 'localhost';
$dbUsername = 'root';
$dbPassword = '';
$dbName     = 'image';


        //Create connection and select DB
$db = new mysqli($dbHost, $dbUsername, $dbPassword, $dbName);

        // Check connection
if($db->connect_error){
die("Connection failed: " . $db->connect_error);
}
$i = $_SESSION['k'];

$sq = "UPDATE image SET status = '1' WHERE fileno = '$i'";
mysqli_query($db,$sq);
header("Location: dehaze.php");
?>
```

A.2.4. out.php

```
<?php
```

```
session_start();
      //Put session start at the beginning of the file
$i =  $_POST['imgname'];
$i = (int)substr($i, 2, 4);
$_SESSION['k'] = $i;
$img = $_POST['img'];
$img = str_replace('', '', $img);
$img = str_replace(' ', '+', $img);
$data = base64_decode($img);
$file = 'out/'.$_POST["imgname"];
$file = substr($file, 0, -4);
                //Removing .jpg extension

$file = $file.".png";                              //Adding .png
extension

if (file_put_contents($file, $data)) {
   echo "<p>The canvas was saved as $file.</p>";

$f = "in/".$_POST["imgname"];
if (!unlink($f))
  {
  echo ("Error deleting $f");
  }
else
  {
  echo ("Deleted $f");
  }

header("Location: test.php");

} else {
   echo "<p>The canvas could not be saved.</p>";
}
?>
```