Distributed Video Processing Architecture

A Project Report submitted for a Project done
towards partial fulfillment of the
Advanced Operating Systems Course

by

Paladugu, Abhinay and Suman, Ashish
under the guidance of
Dr. Linda M Null

to the
DEPARTMENT OF COMPUTER SCIENCE AND MATHEMATICS
THE PENNSYLVANIA STATE UNIVERSITY HARRISBURG
MIDDLETOWN, PENNSYLVANIA-17057
Acknowledgements

We would like to express our special gratitude to our supervisor Dr. Linda M Null who gave us this golden opportunity to do this challenging project on the Distributed Video Processing Architecture under her guidance. Thank you for supporting us throughout the duration of this project and helping us with suggestions whenever we got stuck. It helped us in performing a significant amount of research and we learned many new things that would definitely help us in future.

We would also like to thank our fellow students in COMP512 class who helped us by providing critical feedback that was much required. We would also like to thank Shashwata Mandal who tried to break our code.
Contents

List of Figures .................................................. 3
List of Tables ..................................................... 4

1 Introduction ..................................................... 1
  1.1 Background and Motivation ................................... 1
  1.2 Problem Statement ............................................ 1
  1.3 Goals .......................................................... 2
  1.4 Organization of The Report ................................. 2

2 Overall General Design ......................................... 3
  2.1 Architecture ................................................ 3
     2.1.1 Server ............................................... 4
     2.1.2 Upload Client ........................................... 4
     2.1.3 Active Client ........................................... 4
  2.2 Walk-through ............................................... 4
  2.3 Challenges Faced ........................................... 5
     2.3.1 Space usage on server .................................. 5
     2.3.2 Packet Drops ........................................... 5
     2.3.3 Server Fault Tolerance ................................. 5
     2.3.4 Client Fault Tolerance ................................. 5

3 Proof of Concept ................................................. 7
  3.1 Image Processing Model ..................................... 7
  3.2 Use Cases .................................................. 8

4 Results and Analysis ............................................ 9

5 Web Application ................................................ 10
  5.1 Webpages .................................................. 10

6 Conclusion and Future Work ..................................... 12
  6.1 Conclusion ................................................ 12
  6.2 Future Work ................................................ 12

References ....................................................... 13
List of Figures

2.1 Relation Between Active Client, Upload Client and Server and the messages are given in the table 2.1 ......................................................... 3

3.1 Video being broken into frames .................................................. 7
3.2 A frame of the video containing a zebra ....................................... 8

5.1 index.html .............................................................................. 10
5.2 uploadVideo.html .................................................................... 11
5.3 Finallanding.ejs - Found .......................................................... 11
List of Tables

2.1 Messages and their contents ................................................................. 3
4.1 Results when our architecture was run under different conditions ........ 9
Chapter 1

Introduction

1.1 Background and Motivation

In this digital era, there are a lot of important tasks that we leave for computers to perform without much human intervention. Though computers have classically been receiving inputs in the form of symbols, this conception has changed in the last few years. For us human beings, one of the main sensory input is what we see. We are now trying to make computers understand what they see and make inferences based on it. This change has lead to an exponential increase in the amount of study happening in the field of video processing. Processing a video is a computationally intensive task which takes a significant amount of time even on modern day powerful computers. For example, if we have a video of 25 frames per second and each frame has resolution 720 * 480, this equates to around 26 million operations per second. Since this itself is so computationally intensive we can think how intensive it would be to process a higher frame rate video.

Because of the heavy computation requirement for video processing, the first thing that might come into someone’s (possibly a computer scientist) mind would be what would happen if this workload is distributed. What if we break the video into frames and then process these frames separately on a different machine and then clip them back together. Can we use the divide and conquer approach i.e. can we process each of these frames in a parallel setting over the network. Our claim is that yes we can do this using a distributed network of computers in a client-server architecture. If one system gets a video input, we would then break the video file into parts and send them over a network to the worker nodes who will the process these parts and send back an output. The overall processing time in this distributed setup should be lower compared to a single machine doing all the work.

1.2 Problem Statement

As suggested in the introduction, the main problem that we are trying to solve is to reduce the amount of time taken to process a video by designing a distributed system which will perform parallel processing of video frames in order to achieve a considerable gain in video
processing speed. By speed, we mean to process all the frames of the video as quickly as possible. This system will prove to be advantageous if there is a time constraint for processing a video. It may find possible applications in various field including traffic footage processing, video processing for scientific studies, computer vision and augmented reality.

Systems available today are capable of achieving this but these systems must have very huge computing power. As this computation power decreases, the lack of computation is mitigated by frame dropping or the overall processing time increases. Therefore, rather than a single system processing all the frames, if the work can be distributed among multiple nodes using client-server architecture, the processing time can decrease significantly.

To formalize this statement, the main goals of the architecture, probable input, and the intended output is discussed in the next section.

1.3 Goals

Below are the goals we achieved:

1. Object detection in a video.
2. The overall video processing time over the distributed system is lower when compared to the time taken while processing the same video on a single machine.

The application takes as input

1. A video file
2. The object to be detected as a string

Once a video is uploaded to the server, it will break down the video into frames and distribute the frames among the connected worker nodes. The server will also inform the worker nodes about the target object to be detected. The object detection model is set up on the worker nodes once it connects to the server. The worker nodes use this model to predict if the object lies in the image currently held by it. If a node detects the target object successfully, it replies back to the server with the frame number in which the object was detected. The server then stops all other nodes from processing any further frames and outputs the result. The system then moves on to processing the next video uploaded.

1.4 Organization of The Report

The rest of the report is organized as follows. Chapter 2 talks about the architecture of the project. Chapter 3 lists our Proof of concept and various use cases. Chapter 4 compares the time taken with number of clients connected. Chapter 5 shows our website structure. Chapter 6 concludes this report and provides future directions for work.
Chapter 2

Overall General Design

2.1 Architecture

The Distributed Video Processing System uses client-server architecture as shown in the Fig. 2.1

![Diagram of client-server architecture]

**Fig. 2.1** Relation Between Active Client, Upload Client and Server and the messages are given in the table 2.1

<table>
<thead>
<tr>
<th>Message Number</th>
<th>Message Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Video and Object to be detected</td>
</tr>
<tr>
<td>2</td>
<td>Result if the object has been found in any frame of the video</td>
</tr>
<tr>
<td>3</td>
<td>One frame of the video</td>
</tr>
<tr>
<td>4</td>
<td>Result if the Object is present in the frame sent in 3</td>
</tr>
</tbody>
</table>

**Table 2.1** Messages and their contents
2.1.1 Server

The server acts as the central control unit and manages the overall functioning of the system. All the data being uploaded and processed is stored on the server. The various tasks of the server includes breaking down the uploaded video into frames, sending frames of the video to the active clients and sending the final result to the upload client. The server is hosting our website using a Node.js framework [2].

2.1.2 Upload Client

An Upload client is a node which has a video that is to be processed. It is a JavaScript enabled browser that uploads a video to the server and then requests for the detection of a specific object in the video. It then waits for the final result.

2.1.3 Active Client

An Active Client is a node which posses computational power and is willing to lend its computational resources for overall functioning of the system. One of our main goals while designing the system was to make clients as independent as possible. The client can be any digital device which has a JavaScript enabled web browser. No application installation is required on the clients. This property makes this a very good architecture for an Internet-based public volunteer computing project.

2.2 Walk-through

Lets now discuss step by step functioning of the system. Once the server is up and running, multiple devices with processing power can connect to the server. These clients are termed as Active Clients. At this stage the active clients and the server are waiting for a job to be submitted. When a connected client uploads a video to be processed along with the information about the object to be detected, a job is created for the system to perform. This job is added to the list of active jobs and the processing of the video starts. First the uploaded video is broken down into frames and the video is deleted after this to conserve server space. The server then distributes the frames among the clients which are currently connected to the server. The server also sends the object detection model along with the frames. A client which receives the object detection model, caches the same in its browser cache and starts processing the frame. Once a client finishes processing a frame it sends back the result to the server. After that the client receives next available frame for processing. If a client detects the target object in any frame, it return a success message to the server along with the frame numbers in which the object was detected. If the server receives a success message from any client it replies back to the upload client the frame in which the object was detected. Otherwise once all frames are processed, the server sends a message to the client stating that the target object was not detected in the video uploaded. The system then starts processing the next job in the job queue.
2.3 Challenges Faced

2.3.1 Space usage on server

The following data is stored on the server:

1. The uploaded video file
2. The frames generated from the video

As we know storing a video file can be space intensive and in a server with limited storage capacity this can be huge issue. As there is no limit on the length of the video being uploaded on the server, a client may upload a large size video file which can result in a space crunch on the server. To handle this issue we have designed the system such that it deletes the video file as soon as the video is successfully broken down into frames.

The number of generated frames is dependent on the frame rate of the video file. For example a 2 minute video with frame rate of 30fps will generate total of 3600 frames. As this number will be much higher for a higher frame rate video, this will occupy large amount of space on the server. As we can generally observe if an object exists in a video, it is spread over several frames. So to detect a target object in a video file, we do not need to process each and every frame of the video. The space usage on the server for storing the video frames can be reduced by controlling the number of frames generated. The system is currently configured to extract 30 frames from each second of the video. This setting can be easily changed from the configuration file.

2.3.2 Packet Drops

*What if a frame is lost while transmission?* There is a high chance of frames being dropped during transmission. We therefore decided to use a HyperText Transfer Protocol (HTTP) based transmission which uses TCP as a transport layer. TCP detects any packet lost during transmission and performs retransmission of the lost packet to ensure reliable messaging.

2.3.3 Server Fault Tolerance

*What if the server crashes while processing a video?* There is a possibility that the server may crash while processing a video. The system is designed to store the uploaded videos and the frames generated from these videos in non-volatile storage. We are also using a *sqlite* database to store other information like the list of active clients, the information about the frames the active clients are currently processing, details of all the ongoing jobs and the final results for different jobs. These measures ensures that when the server comes back up again, all the data and important information are recovered and the incomplete jobs restart as if the server never crashed in the first place. Thus we have fault tolerance against server crashes and failures.

2.3.4 Client Fault Tolerance

*What if the client crashes while processing a frame or after processing a frame and before submitting the result?* Since we are processing frames from a video, we can observe that
if an object exists in the video it will be present in multiple frames. So even if few of the client fail while processing frames, there is a negligible effect on the overall functioning of the system. To achieve even more client fault tolerance, we can increase the number of frames per second we extract from the video. As stated before, the current system configuration is to extract 30 frames from each second of the video. This configuration provides a good balance between fault tolerance and space required to store these frames.

What if the client crashes after uploading a video for processing? If a client uploads a video for processing and crashes while waiting for the results, it can reconnect to the server and use the Check Last Request button on the homepage of the application to get the results for the last request it submitted.
Chapter 3

Proof of Concept

To demonstrate our architecture, we implemented a simple object detection task as a proof of concept. Object detection here means that the user provides the name of an object to be detected in a video and the application returns the first occurrence of that object.

Once the server receives a video and the name of the object to be detected it splits the video into frames as depicted in Fig 3.1.

![Diagram of video frames](image)

**Fig. 3.1** Video being broken into frames

After this split is done, the object detection model is run on each frame in a first-in-first-out sequence i.e. the frame that was first created is sent for processing first and then the next and so on. Please note that this sequence is only valid for the distribution of frames but not for the order in which the results are received. Whichever frame gets processed first and returns a success response regarding the object being detected is then returned to the upload client that requested the job. If the object is not found even after all the frames are processed we return a "object is not found" response to the upload client.

3.1 Image Processing Model

As a proof of concept we have used a pre-trained image processing model called Mobilenet based on tensorflow.js[1]. Mobilenet[3] provides a list of objects that can be classified so we used only these objects. Mobilenet returns the objects present in the video with the probability of detection.
3.2 Use Cases

We selected a nature video from YouTube and trimmed it so that the number of frames wouldn’t be overwhelming. In general the architecture can handle any number of frames but we trimmed the video just for demonstration purposes. We searched for a zebra in the video as test case 1 and then for an elephant as test case 2. A zebra is present in the video but an elephant is not and we got correct results for both the test cases. One of the frames from the video which has a zebra is given in Fig 3.2

![A frame of the video containing a zebra](image)

**Fig. 3.2** A frame of the video containing a zebra
Chapter 4

Results and Analysis

As mentioned in Chapter 3 we have used a wildlife video to detect a zebra and the results are tabulated in Table 4.1. We used different types of systems for testing like wireless mobile devices, wired personal computers (SUNLab Machines), etc. The server in all the cases was running on a wireless Personal Computer.

- Size of the video file: 17 MB
- Length of the video: 15 seconds
- Total Frames Generated: 313
- Time taken by the algorithm on a single machine: 1 minute 30 seconds

<table>
<thead>
<tr>
<th>Number of nodes</th>
<th>Total Time Taken</th>
<th>Found in Frame</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 Wired PC's</td>
<td>44.21 seconds</td>
<td>313</td>
</tr>
<tr>
<td>7 Wired PC's</td>
<td>58.51 seconds</td>
<td>312</td>
</tr>
<tr>
<td>6 Wired PC's</td>
<td>56.23 seconds</td>
<td>312</td>
</tr>
<tr>
<td>4 Wired PC's and 3 Wireless Mobile Devices</td>
<td>1.0019 minutes</td>
<td>313</td>
</tr>
<tr>
<td>2 Wireless PC’s and 3 Wireless Mobile Devices</td>
<td>2.50 minutes</td>
<td>313</td>
</tr>
</tbody>
</table>

Table 4.1  Results when our architecture was run under different conditions

On analyzing the results we obtained, it is clear that the distributed architecture works better when compared to a single machine running the same algorithm. We can also observe that the time taken to process the same video slightly increases if we use few wireless devices along with the wired devices as clients. This difference is because of the increase in latency in wireless networks. In fact if we use all mobile devices as clients the processing time is much higher. So we can conclude that the throughput of the application is based on the bandwidth and latency of the underlying network infrastructure.
Chapter 5

Web Application

5.1 Webpages

The server hosts a number of web pages which provides the end user a clean and intuitive interface to interact with the application. The web application is designed to be responsive and thus it works seamlessly on all devices including laptops, mobile phones, and tablets. Below is the list of web pages available to the end user along with few screenshots of the web application.

1. index.html - The home page of the website.

![Fig. 5.1 index.html](image)

2. processor.html - The page where a client lands when it is waiting for a job.

3. uploadVideo.html - The page where a client can upload a video and the object to be detected.
4. ObjDect.ejs - The page where the clients run the image processing algorithm.

5. results.html - The page where the upload client waits for results.

6. Finallanding.ejs - The page where the final result is displayed to the user.

7. team.html and about.html - Pages containing information about the team and the project.
Chapter 6

Conclusion and Future Work

6.1 Conclusion

This project was a great learning experience for us. We learned a lot about distributed systems and programming in general. We got hands-on experience with the implementation of various distributed systems concepts we learned in class. The simplicity of the application we designed makes it fairly easy to use for personal and commercial purposes.

6.2 Future Work

Since video processing is an integral part of many modern day applications like traffic monitoring, weather forecast, and scientific studies, there is an ever-growing demand for faster and more efficient video processing applications. The distributed architecture of our application makes it well suited for implementation in a wide range of use-cases. In the future, adding some of the below-mentioned features will further increase the scope of this architecture:

1. Support for real-time processing of streaming video like traffic camera footage.
2. Security features to handle sensitive data.
3. Further reduce the throughput of the application.
References


