Video Ingress System for Surveillance Video Querying

by

Aaron Sipser

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2020

Author .....
Department of Electrical Engineering and Computer Science
May 12, 2020

Certified by .....
Michael R. Stonebraker
Adjunct Professor
Thesis Supervisor

Accepted by .....
Katrina LaCurts
Chair, Master of Engineering Thesis Committee

Video Ingress System for Surveillance Video Querying

by

Aaron Sipser

Submitted to the Department of Electrical Engineering and Computer Science
on May 12, 2020, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Police departments struggle to review surveillance footage in an efficient manner. Finding a person matching certain characteristics requires manually scrolling through video feeds, potentially wasting hundreds of valuable man-hours solving crimes. SurvQ is a video query system which automates this process. Many existing systems are either too computationally intensive or only work on one type of camera. SurvQ, instead, focuses on a real time ingest system which supports arbitrary video sources (body cameras, dash cams, CCTV) at scale. It then uses a combination of cheap object detection, on-demand analysis, and priority ranking to efficiently analyze relevant video. We found this approach could scale to several hundred cameras in real time, suitable for the use-case of an entire police department in West Lafayette, IN.

Thesis Supervisor: Michael R. Stonebraker
Title: Adjunct Professor

Acknowledgments

Thank you very much to the Mikes from MIT (Mike Stonebraker and Mike Cafarella). They constantly gave advice and provided direction to the project. I also want to thank my best friend, Zachary Collins, for working with me on this project. The system wouldn't be anywhere where it is without his partnership and friendship.

Contents

1 Introduction **13**

2 Related Work **17**

2.1 Video Search 17

2.2 Trigger Systems 18

3 System Architecture **19**

3.1 Video Ingress Module 20

3.2 Feature Extraction Module 21

3.3 User Interface Module 22

3.4 Central Database 23

3.4.1 Geospatial Support 23

3.4.2 Communication 23

4 Design and Performance **25**

4.1 Video Ingest Pipeline 25

4.2 Feature Extraction 27

4.2.1 YOLO as a Base Model 28

4.2.2 YOLO Priority System 28

4.2.3 Parallelizing Feature Models 29

4.3 Database Triggers 29

5 Future Work **33**

5.1 Migrating to a Distributed Database 33

5.2 Offloading Trigger Communication 33
5.2.1 Adding Additional Feature Extraction Models 34
5.2.2 Adding Features to Priority System 34

6 Conclusion **37**

List of Figures

3-1 High Level Overview 20
3-2 Video Ingress Module 20
3-3 Feature Extraction Module 21
3-4 User Interface Module 22

4-1 60K Records Insertion Time (no triggers firing) vs # Active Investigations. 31
4-2 60K Records Insertion Time (triggers firing) vs # Active Investigations. 31

List of Tables

4.1 Time to process 60 second video at various quality, averaged over 5 trials. 27
4.2 Video properties of interest to law enforcement. 27

Chapter 1

Introduction

Surveillance footage is a surprisingly powerful tool for police task forces. In 2010, the number of CCTV cameras in Britain was roughly 60,000 [6]. This figure has grown significantly since then, with some studies putting numbers as high as 5 million [8]. In addition to CCTV footage, police forces use a multitude of camera types (body cameras, dash cameras, mobile video) in their investigations. Reviewing this footage requires a police officer to scroll through it manually, trying to find a specific suspect. This is a painstakingly manual process that can waste hundreds of police officer man-hours for just an individual case [11, 22].

We address this problem with Surveillance Querying (SurvQ), a system that provides real time video feature processing and analysis at scale. It allows police officers to take raw video footage and "search" through it for individuals matching certain characteristics. It processes video in a fraction of the time it would take a human. Furthermore, SurvQ stores previous user queries and runs them on new video as it enters the system. If the new video matches a previous query made by the user, SurvQ notifies that user. This allows officers to keep track of ongoing events, and keep tabs on unsolved crimes. We designed our system for the use case of a police department in West Lafayette which has on the order of 200 CCTV cameras. West Lafayette Police are interested in roughly 31 characteristics [4], such as race, gender, and clothing color. Currently SurvQ supports a small subset of these features but has the infrastructure to add more as they are developed.

In order to operate in real time and at scale, SurvQ had to address machine learning, video processing, and infrastructural challenges. Many of the thirty features require individual models, which need to be integrated with each other to provide valuable information. These models need to run in parallel to maintain performance, but also combine their results as if one is run after the other. Additionally, we had finite computational resources with a single GPU machine. Using a priority system and on-demand analysis approach allowed SurvQ to meet its strict real time goals.

Another challenge for SurvQ's real time objective was the heavy amounts of video data entering the system simultaneously. This video data could come in a multitude of formats and sources. This led to several video processing challenges as internally all video is stored as 1-minute MP4 files, so we needed a mechanism to convert video and store it in under the time it took to record it.

All real-time goals had to be held in the context of the Lafayette use-case. Many parts of SurvQ had to be scaled horizontally because providing more computational resources was less effective than parallelization. We landed on using Kubernetes [7] as an infrastructural solution which could easily scale and maintain all the separate modules of the system.

Beyond the challenges for SurvQ to operate effectively and in real time, it needed an interface to interact with the above systems. This interface posed many challenges on its own, both from implementation and design standpoints. All of the data for the project was stored in a SQL database, but its main user audience, police forces, have no knowledge of the language. Furthermore, most police forces use technology written many years ago [9, 22], making it difficult to create a user interface that would be intuitive. We landed on creating a simplistic interface focused on capturing user intent and converting it into SQL queries and triggers.

This paper addresses the challenges in designing such a video ingest system by presenting an implementation and analysis of the system. Beyond the video ingest pipeline, the paper discusses the techniques and infrastructure to support running many machine learning algorithms concurrently. The result of these pieces is a system which can support the West Lafayette use-case of analyzing several hundred cameras

in real time.

Chapter 2

Related Work

2.1 Video Search

Building systems to process video at scale is a well-research domain. Systems such as BlazeIt [19], NoScope [20], and Focus [18] each use neural networks to preprocess footage. BlazeIt and Focus run efficient and computationally cheap object detectors on a subset of incoming footage. They then run expensive models upon user query, on a smaller subset of filtered video. SurvQ uses a similar multi-stage approach using an open source object detector before running custom algorithms or additional models. SurvQ, however, defers all video processing until user query time. SurvQ operates on a lower computational footprint, and cannot run object detection on all incoming video. Additionally, BlazeIt is tailored for analysts comfortable with SQL, as demonstrated by its custom FrameQL language to perform video analytic queries. SurvQ's audience are users with zero programming knowledge, who need a custom user interface to search through video. NoScope specializes in optimizing video search for fixed-angle cameras, such as CCTV feeds, while SurvQ is built to be operational on all video sources (body cam, dash cam, CCTV, etc).

2.2 Trigger Systems

Much research into active databases has been done, as can be seen by Postgres [24], Ariel [15] and Starbust [25]. These systems describe an event-driven architecture using database triggers. Ode [21] defines a language O++ for database applications to subscribe to database state changes. SurvQ has a need for such a trigger system to support standing queries for ongoing investigations.

One key element we were aware of in SurvQ was performance of triggers, and how they would scale in the West Lafayette use case. Timer-Driven Triggers [17] checks for database changes at a fixed interval, not upon every record insertion. This sacrifices latency for performance, but may be necessary if scaling to an extremely large quantity of triggers. TriggerMan [16] is a database extension designed to resolve common scale issues with database triggers through clever use of caching and relying on the fact that many triggers in a database will share similar structure. Sentinel [12] uses temporal events to filter trigger actions based on event time. Our goal was to make our trigger system operate in real time for the West Lafayette use-case on a native Postgres 11 instance.

Chapter 3

System Architecture

SurvQ is a modular system, consisting of a Video Ingress, Feature Extraction, and a User Interface (UI) module. Each module connects to a central relational database. All communication between modules is done via the central database. This was a core principle of the project to reduce complexity. The Video Ingress module takes all incoming video streams, segments them into 1-minute MP4 clips, and uploads them to a cloud storage container. It also inserts metadata about the clips into the central database, as well as sends a message to the Feature Extraction module that there is new video to be run. The Feature Extraction module downloads the video onto machine learning servers, and runs our models on it. All results from the models are stored back into the central database. The UI queries the database for the results of the video feature extraction, and places database triggers for notification when new relevant information is available. A high level overview of the entire system can be seen in Figure 3-1.

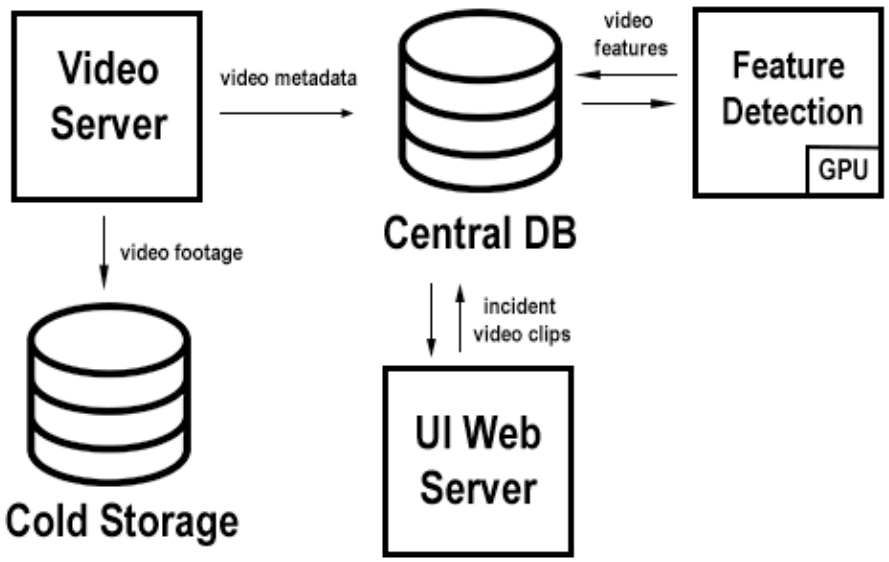

Figure 3-1: High Level Overview

3.1 Video Ingress Module

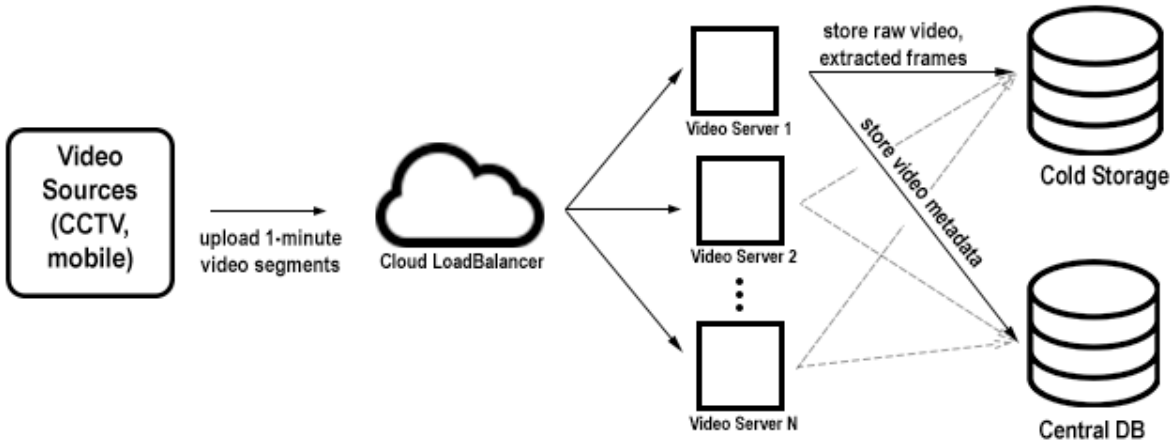

Figure 3-2: Video Ingress Module

Figure 3-2 has a more indepth view of the Video Ingress module. We assume that all ingress sources upload their video in 1 minute long clips. The mobile and laptop video sources come from applications which we built. These applications allow users to record video and upload it to our system. Both laptop and mobile applications

automatically split recorded video into minute long segments, and convert video to MP4. Video which comes from sources we did not have control over (CCTV cameras) is manually split into 1 minute clips, and then uploaded to the video servers.

A Kubernetes load balancer allows us to horizontally scale our video servers. It takes incoming video being uploaded to SurvQ and routes it to an available video server. This lets us to scale our video servers to meet incoming demand.

Upon receiving a video segment, a video server performs several operations. First, it converts it to MP4 if the video format is different. Then, it extracts video frames at a subsampled rate (1 frame / second) from the video for machine learning models. The video frames, as a set of images, and the raw video footage, is uploaded to a GCP container. Simultaneously, video servers insert video metadata to the central database. Video metadata consists of the link to the video in cloud storage, its location, a timestamp, and an ID of the device the video came from. Lastly, video servers notify the Feature Extraction module that new video is available to be analyzed. This is done by inserting a record linking to the video metadata into a specific table that feature extraction servers poll for updates.

3.2 Feature Extraction Module

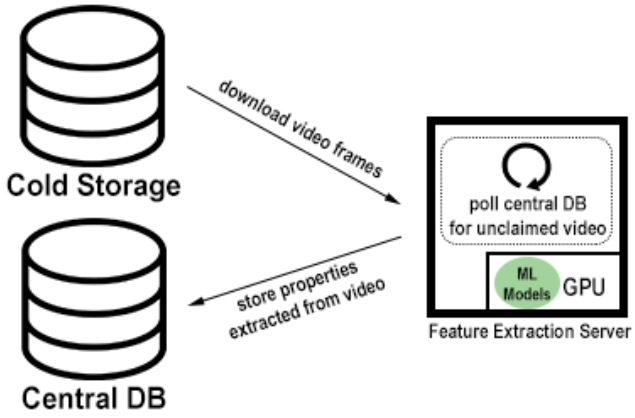

Figure 3-3: Feature Extraction Module

Figure 3-3 shows how the Feature Extraction module works in more detail. Each

machine learning server is equipped with a GPU to run models. Each server polls the central database at a fixed interval for new video to analyze. The polling mechanism uses a priority system which ranks unprocessed video clips based on user interest, described further in section 4.2.2. This is needed when too much video is being uploaded to the system for the machine learning servers to keep up in real time. After selecting the video with highest priority, a server marks the video clip as claimed in the central database. This is done to prevent other servers from processing the same video. It then downloads the frames of the video, and runs the machine learning models on the images. Feature extraction is significantly faster when run on subsampled frames, and doesn't lose noticeable accuracy. After running the models, all the extracted properties from the video frames are stored back in the central database.

3.3 User Interface Module

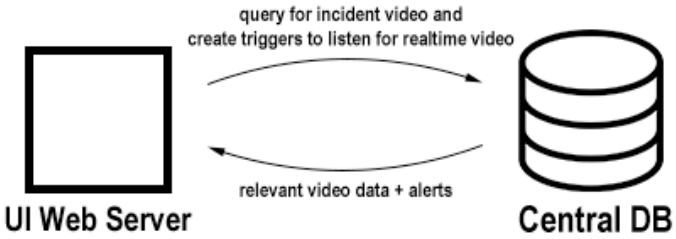

Figure 3-4: User Interface Module

The User Interface communicates with the central database, as demonstrated in Figure 3-4. The UI allows a police officer to convert an incident report into an investigation. In an investigation, he can search video based on the suspect appearance. Beyond just characteristics, an officer can filter video temporally through a query time range, and geographically through selecting a region on a map. This map displays all active cameras in SurvQ's database, so an officer knows exactly which cameras

he will be searching through. Upon creating an investigation, the web server queries the database for all video clips which have frames that match features specified by the user. It additionally places a trigger on the database, which fires if the feature extraction servers insert any new records which match on the spatiotemporal and characteristic query.

3.4 Central Database

The central database is a key figure in SurvQ. The database used is PostgreSQL, which is the one of the most popular open source relational databases on the market [1]. A relational database is a good pick for SurvQ due to the structure it provides via a schema. This structure enables relational databases to be faster at querying and insertion than noSQL databases [14]. Postgres additionally supports complex queries at a scale needed for SurvQ's use-case in West Lafayette, making it a good fit for the project.

3.4.1 Geospatial Support

Postgres supports geospatial queries through an extension called PostGIS [10], which is key in filtering videos to run feature extraction. This is done by representing each video segment in the database as geospatial points, using their latitude and longitude. So, a user's region of interest as specified in the user interface can be easily converted into a geospatial filter which eliminates any video clips which were recorded outside the query region.

3.4.2 Communication

Unlike ViCiBaF [26] and LaS-VPE [27] which make extensive use of message brokers for video analysis, we found Postgres fast enough to handle both data storage and communication. We used Postgres for two main forms of communication in SurvQ. One used database triggers to alert services via websockets. The other was a polling

system where updates to specific tables would be tracked. Polling was useful when messages would come in a reliable stream, for example in video ingress. Websocket communication was used in cases where messages needed to be sent to a wide range of users at an irregular interval. The main use for this was delivering notifications to the user interface that new video was available for their investigation. These notifications could come in a flood, or come very rarely, making websockets a good choice.

One concern that needed to be analyzed was the latency triggers and polling put on the database. Polling has a negligible effect, as the polling rate was once every few seconds. Postgres can support thousands of queries per second [4], so the additional cost of polling was minimal. However, triggers may place a large burden on a database, particularly on tables with high traffic. We found that triggers would support the real time use-case, with analysis in section 4.2.2.

Chapter 4

Design and Performance

As stated, SurvQ's system was designed around a use-case of several hundred live cameras uploading data in real time. This let us put exact bounds to how fast the various modules needed to be to meet time constraints. Below is an analysis of the design goals and performance of the ingest pipeline, feature extraction, and database triggers.

4.1 Video Ingest Pipeline

SurvQ's Ingest Pipeline had several goals which guided its design.

1. **Support video from any source.** Police forces have video coming from body cameras, dash cameras, and light pole cameras. Each of these videos comes in a different format, and quality.

2. **Internally maintain consistency between all stored video.** This involved converting each incoming video if it didn't match the internal format.

3. **Operate in real time.** Even with all cameras active in the Lafayette use-case, SurvQ's ingest pipeline shouldn't fall behind in processing footage.

In order to support video from any source, SurvQ automatically converts any incoming video to an internal format, MP4, when storing it in our cloud storage

servers. All video with proprietary formats are converted separately to our internal format. A key reason to use MP4 as the internal video format is the support in web browsers for MP4 video. The majority of modern web browsers support a limited set of video formats, and all include MP4 as one [5].

Evaluating if SurvQ could operate in real time is difficult to test, as video from different sources comes in various quality and framerate. In order to get an accurate benchmark, all video was assumed to arrive in the internal MP4 format, and in the same video quality. We used a mobile application developed on Android to simulate a camera uploading video in real time. We tested processing time for several different quality videos, and compared the results with video quality of CCTV footage.

In order for SurvQ to run in real-time, no video server could take longer to process video than the length of the video clip. For example, if a minute long clip was uploaded, SurvQ had to process it in under a minute. Video is coming continuously in the same interval, so as soon as it takes longer to process video than the interval length it is impossible to keep up with the demand, regardless of the number of video servers. Once processing time is less than the interval length, each optimization to reduce processing time leads to a reduction of the number of video servers needed to handle incoming load. If video is processed in half the interval length, than the number of necessary video servers is $O\left(\frac{\#\text{ active video sources}}{2}\right)$. This is because one video server could process two camera's videos before receiving more video to process. Each video server needs roughly one CPU to operate. So, the total necessary video cameras and CPU cores can be described by $O\left(\frac{\#\text{ active video sources}}{\text{processing time}}\right)$. If this number is greater than the number of the cameras then SurvQ cannot operate in real time.

There were several optimizations possible which enabled us to lower the video processing time. First, video servers needed to split each video segment into a set of subsampled frames, and to upload both the frames and the original video to the server. We parallelized this process, reducing processing time to whichever task was individually longer. Lowering the video file size had the greatest impact on performance. This was done through lowering the quality of the video and its frame rate.

Unfortunately, we didn't have access to modify the camera's frame rate but changing the camera's quality had a dramatic impact on the processing time as can be seen in Table 4.1 below.

Video Quality	Segment File Size	Processing Time
1080p	230 Mb	24.5s
720p	79.4 Mb	10.5s
480p	39.58 Mb	4.9s

Table 4.1: Time to process 60 second video at various quality, averaged over 5 trials.

The file size of Lafayette CCTV video after being manually converted to MP4 was on average 15.5 Mb. So, the lowest quality setting on the mobile application serves as a reasonable upper bound on the number of video servers needed for the Lafayette use-case. One video server would be able to support at least 10 mobile cameras at 480p simultaneously. Thus, to support the Lafayette use-case of 200 CCTV cameras would take at most 20 video servers.

4.2 Feature Extraction

The Feature Extraction Module's high level goal was simple. Provided CCTV footage, identify a set of features from the Lafayette police department. This set of 31 features is shown in Table 4.2 below.

White	Black	Hispanic	Asian
Male	Female	Tattoos	Beard
Bald	Hair color	Sandals	Shoes
Boots	Jeans	Pants	Shorts
T-shirt	Baseball hat	Jacket	Tall
Shorts	Walking	Running	Motorcycle
Bicycle	Truck	Passenger car	Skateboard
Smoking	Backpack	Headphones	

Table 4.2: Video properties of interest to law enforcement.

4.2.1 YOLO as a Base Model

Not all of these features can be detected by the same model, so the feature extraction module needs to be able to support running multiple models at once. Additionally, their results need to be combined cleverly. For example, given one image with two people, one model might output "one short and one tall" and the other can output "one male and one female". Detecting whether the female is tall or short requires combining the models. This was done by using a popular open-source object detector, YOLO [23], as a basis for other models to build off of. Given an image, YOLO marks the bounding boxes of all people, cars, etc. Then, as long as each model pairs its output with the bounding box of the object it is parsing, all features can be associated with specific objects in the image. Currently we only have this process working for one model, which detects colors of jacket and pants. However, following this system will allow us to integrate other models with ease.

4.2.2 YOLO Priority System

Ideally, we can run all feature extraction models on video as it enters the system. Currently, running YOLO and the color detector on minute-long video takes around 15 seconds on a machine with a NVIDIA Tesla K80 GPU. While CPU and RAM are quite cheap, GPUs are an expensive commodity. So, unlike the video ingress pipeline, we cannot scale the machine learning servers horizontally until we meet incoming demand. We used a different approach with feature extraction by running our models at query time. We initially let feature extraction servers lay dormant until users create an investigation. Each investigation is associated with a time and location, and can serve as a spatiotemporal filter for video to run feature extraction. While West Lafayette may have 200 cameras, an individual investigation likely will only want to run on a select few CCTV feeds, which is only a few hours of footage. This spatiotemporal filter narrows down video from potentially years of footage to a handful of hours. We then run feature extraction on this video. The tradeoff is that a user may have to wait after creating an investigation for results to populate.

While this approach is useful, it falls short in several areas. A user mistake can easily overload the system by selecting too wide an area, or too long a time range. This causes the feature extraction servers to run on too much video, and prevents it from working on video from other users' queries. Another problem is that if no investigations have been created, the feature extraction servers will be sitting dormant while they could be processing video. The solution to both these problems is a priority system to rank individual video clips as they enter the system, and as investigations are created. Video clips which are relevant to multiple investigations would be given higher priority. If there were no active investigations, the feature extraction models would passively process available videos.

Priority was assigned linearly. All incoming video which don't match any current investigation's time and location are assigned a priority of 0. Each matching investigation increases the clips priority by one. If a new investigation is created, all clips which match the spatiotemporal filter have their priority increased. We find this approach fully utilized the GPU resources by keeping the feature extraction model active, while always running on the most relevant video available.

4.2.3 Parallelizing Feature Models

The current machine learning environment runs on a single machine. We can achieve a linear speedup from parallelizing feature models. Two GPU machines running the color model would process twice as much video in the same time. To ensure each model would not run on duplicate video, each feature extraction server uses a test-and-set approach when claiming video to run on. This guarantees that a given video clip will be claimed by at most one server.

4.3 Database Triggers

The largest strain on our Postgres database came from our use of triggers. To provide users with real time feedback on investigation updates, we place one trigger on the database for each active investigation. Every investigation trigger runs after a feature

extraction server finishes processing a video clip. The investigation trigger queries for all processed frames of that video clip, and fires a notification message to the UI if the extracted features match the investigation criteria (time, location, characteristics).

For a given video clip, feature extraction servers insert
O (# detected people per frame $\times$ # total frames) records. Lafayette CCTV footage has roughly 5 people per frame, and 60 frames per clip = 300 records per video clip. The Lafayette use case was projected to have 200 active cameras, producing video clips at one minute intervals, which equals 60,000 feature records entering the database per minute. In order for the database triggers to support that, they had to be able to process the incoming load in under the insertion rate. So, inserting 60K records had to take less than 60 seconds. This time would depend on the number of active investigations. This makes sense, as each active investigation has its own trigger. Figures 4-1 and 4-2 demonstrate that SurvQ can easily support 100 concurrent investigations, a number suitable for West Lafayette police [13].

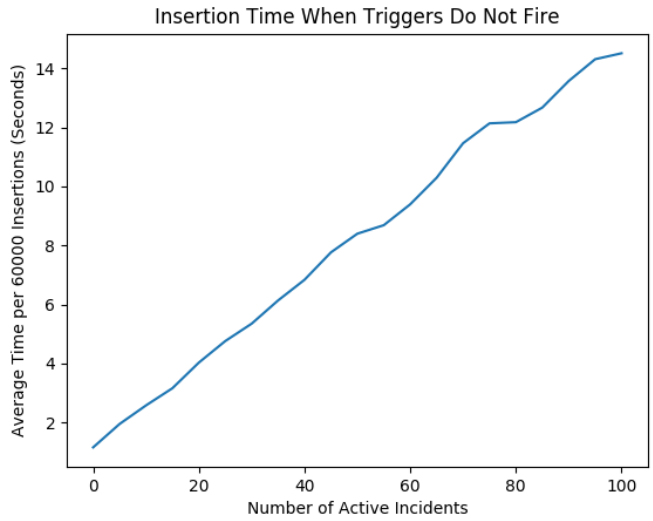

Figure 4-1: 60K Records Insertion Time (no triggers firing) vs # Active Investigations.

Figure 4-2: 60K Records Insertion Time (triggers firing) vs # Active Investigations.

Chapter 5

Future Work

5.1 Migrating to a Distributed Database

Given a much larger use case of 50,000 cameras, one can expect there to be over 10 million inserts into the database per minute from feature extraction models. At some point the load on the central database from the sheer number of inserts into the system will become a new bottleneck to SurvQ. Native Postgres cannot be scaled horizontally, unlike video processing servers. Fortunately, with the use of Citus [3], a Postgres extension, we can shard our Postgres database across multiple computers. Citus enables us to scale our database horizontally. The transition to Citus will be easy as it uses Postgres and runs on our existing infrastructure of Kubernetes. This transition isn't necessary, however, until the current bottlenecks in GPU resources are resolved.

5.2 Offloading Trigger Communication

While our current trigger workload easily runs within time constraints for real time use, it will eventually overwhelm our system. As stated above, if we increase the number of video feeds in SurvQ, we will increase the number of inserts into the central database. Specifically, we would strictly increase inserts into a table which has triggers that run on each insert. Additionally, we assumed SurvQ would not run

more than several hundred investigations concurrently. As shown in Figures 4-1 and 4-2, the number of active investigations is proportional to the latency of an insert. If there are dramatically more investigations than expected, or we expand to a use case with many more feeds, SurvQ will be bottlenecked from trigger processing.

Triggers are used in SurvQ as a form of communication (sending notification updates to the UI). By migrating to a system built for such communication, such as Kafka, we can remove the majority of triggers from the database. While Kafka adds significant complexity to SurvQ, it is designed for exactly this purpose. It can handle workloads orders of magnitude greater than persistent storage systems, such as Postgres [2].

5.2.1 Adding Additional Feature Extraction Models

Currently the feature extraction servers are the biggest bottleneck to SurvQ. As we add more models which need to be run, this bottleneck will only get worse. To improve this we will need additional computational resources, in addition to an improved priority system.

5.2.2 Adding Features to Priority System

While West Lafayette police are interested in 31 features, a given incident will only filter on several. For example, an investigation may specify three features: white, female, in a red shirt. Only a subset of the feature models need to be run for that investigation. Using the above priority system, we can incorporate an investigation's features of interest to run video only on the models which extract those features. To do this, we maintain a separate priority ranking for each model. Upon creation of an investigation, the priority for the corresponding feature models is increased as before. One difference is that if multiple feature extraction models are sharing the same GPU, some models will have to stay dormant. In the current system, if there are no video clips with elevated priority, the color model will passively run on video in the system. However, if another feature model has video with elevated priority,

then the color model will impact its performance via resource competition. Models would only be able to run passively if there will be no other models active.

Chapter 6

Conclusion

We present SurvQ, a real-time video monitoring system that is capable of concurrently handling hundreds of video feeds. It is composed of three core modules: video ingress, feature extraction, and a user interface. Video ingress can support video of all types (body cameras, CCTV, mobile) and of any format. Feature extraction uses state of the art object detection algorithms in coordination with machine learning models to extract features of people from video frames. SurvQ only currently supports a subset of the desired 31 features, and has the infrastructure to add more easily. To use GPU resources effectively, models run on-demand with a priority system. These modules are integrated with a user interface designed for police investigators. While SurvQ was built for analyzing surveillance video, we see broad applications of feature detection at scale. Video analysis is necessary in almost every industry, and we hope to expand SurvQ further beyond its current use-case.

Bibliography

[1] 2019 open source database report. https://dzone.com/articles/2019-open-source-database-report-top-databases-pub.

[2] Benchmarking apache kafka: 2 million writes per second (on three cheap machines). https://engineering.linkedin.com/kafka/benchmarking-apache-kafka-2-million-writes-second-three-cheap-machines.

[3] Citus data, inc. https://www.citusdata.com.

[4] Fermi estimates on postgres performance. https://www.citusdata.com/blog/2017/09/29/what-performance-can-you-expect-from-postgres/.

[5] Html video. https://www.w3schools.com/html/html5_video.asp.

[6] Investigation: A sharp focus on cctv. https://www.wired.co.uk/article/investigation-a-sharp-focus-on-cctv.

[7] Kubernetes. https://kubernetes.io/.

[8] One surveillance camera for every 11 people in britain, says cctv survey. https://www.telegraph.co.uk/technology/10172298/One-surveillance-camera-for-every-11-people-in-Britain-says-CCTV-survey.html.

[9] Police embracing tech that predicts crimes. https://www.cnn.com/2012/07/09/tech/innovation/police-tech/index.html.

[10] Postgis. https://postgis.net/.

[11] The rise of cctv surveillance in the us. https://www.bbc.com/news/magazine-22274770.

[12] S Chakravarthy, E Anwar, L Maugis, and D Mishra. Design of sentinel: an object-oriented dmbs with event-based rules. *Information and Software Technology*, 36(9):555 – 568, 1994.

[13] Troy Greene. Investigative interview. Private Communication.

[14] Cornelia Gyorödi, Robert Gyorödi, and Roxana Sotoc. A comparative study of relational and non-relational database models in a web- based application. *International Journal of Advanced Computer Science and Applications*, 6(11), 2015.

[15] E. N. Hanson. The design and implementation of the ariel active database rule system. *IEEE Transactions on Knowledge and Data Engineering*, 8(1):157–172, 1996.

[16] Eric Hanson, Chris Carnes, Lan Huang, Mohan Konyala, Lloyd Noronha, Sashi Parthasarathy, J. Park, and Albert Vernon. Scalable trigger processing. *Proceedings 15th International Conference on Data Engineering (Cat. No.99CB36337)*, pages 266–275, 03 2001.

[17] Eric N. Hanson and Lloyd X. Noronha. Timer-driven database triggers and alerters: Semantics and a challenge. *SIGMOD Rec.*, 28(4):11–16, December 1999.

[18] Kevin Hsieh, Ganesh Ananthanarayanan, Peter Bodik, Paramvir Bahl, Matthai Philipose, Phillip B. Gibbons, and Onur Mutlu. Focus: Querying large video datasets with low latency and low cost, 2018.

[19] Daniel Kang, Peter Bailis, and Matei Zaharia. Blazeit. *Proceedings of the VLDB Endowment*, 13(4):533–546, Dec 2019.

[20] Daniel Kang, John Emmons, Firas Abuzaid, Peter Bailis, and Matei Zaharia. Noscope: Optimizing neural network queries over video at scale. *Proc. VLDB Endow.*, 10(11):1586–1597, August 2017.

[21] B. F. Lieuwen, N. Gehani, and R. Arlein. The ode active database: trigger semantics and implementation. In *Proceedings of the Twelfth International Conference on Data Engineering*, pages 412–420, 1996.

[22] Gerry Palmer. Detective interview. Private Communication.

[23] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015.

[24] Michael Stonebraker and Lawrence A. Rowe. The design of postgres. *SIGMOD Rec.*, 15(2):340–355, June 1986.

[25] J. Widom. The starburst active database rule system. *IEEE Transactions on Knowledge and Data Engineering*, 8(4):583–595, 1996.

[26] Kai Yu, Yang Zhou, Da Li, Zhang Zhang, and Kaiqi Huang. A large-scale distributed video parsing and evaluation platform. *Intelligent Visual Surveillance*, page 37–43, 2016.

[27] Weishan Zhang, Liang Xu, Pengcheng Duan, Wenjuan Gong, Xin Liu, and Qinghua Lu. Towards a high speed video cloud based on batch processing integrated with fast processing. *2014 International Conference on Identification, Information and Knowledge in the Internet of Things*, pages 28–33, 2014.