A Real-time Analytics Pipeline for Scalable Smart Video Surveillance.

Joseph Honour - 130291538

Newcastle University

Author Note

Completed as part of my undergraduate BSc in Computer Science with Industrial Placement (G401), supervised by Matt Collison.

Abstract

Computer vision has been a large area of research in recent years, devising methodologies to understand and act on events seen within video streams. With the wide deployment of CCTV and IP cameras in the past years, the opportunity for smart video analysis is available, however smart video surveillance remains a feature reserved for new hardware systems. To enable the benefits of smart video analysis using existing hardware we need to overcome and standardize the process of combining analysis frameworks with existing hardware, while enabling scalability through the use of Cloud computing.

This paper proposes an extendable and scalable framework that is able to provide an end-to-end video processing pipeline for smart video surveillance. The framework is built on core technologies that are open source, reliable and scalable. Leveraging; OpenCV, Apache Kafka, Apache Flink, Apache Spark, Neo4J and Terraform, the framework provides object detection and tracking, scalability, activity detection, event classification, data representation and system deployment respectively.

The design approach adopted allows extensibility at every opportunity, so the framework can be adapted for a multitude of use cases, with a proposed use case based on the Abbey Road crossing in London shown within this paper. Finally, the framework will be hosted publicly allowing new avenues to be explored by the community, with avenues of exploration suggested at the end of this paper.

Declaration

“I declare that this dissertation represents my own work, except where otherwise stated.”

Acknowledgments

This is my acknowledgments.

Table of Contents

[Abstract 2](#_Toc510019915)

[Declaration 3](#_Toc510019916)

[Acknowledgments 4](#_Toc510019917)

[1. Introduction 8](#_Toc510019918)

[1.1 Motivation 8](#_Toc510019919)

[1.2 Aim 9](#_Toc510019920)

[1.3 Objectives 9](#_Toc510019921)

[1.4 Paper Structure 10](#_Toc510019922)

[2. Background and Literature Review 11](#_Toc510019923)

[2.1 Video Processing Methodologies and their Adoption 11](#_Toc510019924)

[2.1.1 Object Detection 12](#_Toc510019925)

[2.1.2 Object Tracking Techniques 14](#_Toc510019926)

[2.1.3 Behavior and Activity Analysis 15](#_Toc510019927)

[2.1.4 Event Classification 15](#_Toc510019928)

[2.3 Distributed Computing and the Cloud 16](#_Toc510019929)

[2.3.1 Parallel Computing 17](#_Toc510019930)

[2.3.2 Distributed Messaging 18](#_Toc510019931)

[2.3.3 Cloud Computing 19](#_Toc510019932)

[2.4 Existing Technologies and Approaches 20](#_Toc510019933)

[3. Proposed Framework 21](#_Toc510019934)

[3.1 System Architecture 21](#_Toc510019935)

[3.1.1 Pre-Processing 22](#_Toc510019936)

[3.1.2 Activity Analysis 22](#_Toc510019937)

[3.1.3 Event Analysis 22](#_Toc510019938)

[3.1.4 Data Storage 23](#_Toc510019939)

[3.1.5 Sub System Communication 23](#_Toc510019940)

[3.1.6 Cloud Based Architecture 24](#_Toc510019941)

[3.1.7 Improvements 24](#_Toc510019942)

[3.1.8 Limitations 24](#_Toc510019943)

[3.2 System Implementation 25](#_Toc510019944)

[3.2.1 Sub-System Communication 26](#_Toc510019945)

[3.2.2 Video Pre-Processing 28](#_Toc510019946)

[3.2.3 Activity Analysis 29](#_Toc510019947)

[3.2.4 Event Classification 30](#_Toc510019948)

[3.2.5 Event Notifications 30](#_Toc510019949)

[3.2.6 Data Storage 31](#_Toc510019950)

[3.2.7 Data Interfacing 32](#_Toc510019951)

[3.2.8 Infrastructure Deployments 32](#_Toc510019952)

[4. Use Case Evaluation 34](#_Toc510019953)

[4.1 Aim 34](#_Toc510019954)

[4.2 Scenario 34](#_Toc510019955)

[4.3 Implementation Procedure 34](#_Toc510019956)

[4.3.1 Configuring Video Pre-Processing 35](#_Toc510019957)

[4.3.2 Configuring Activity Analysis 36](#_Toc510019958)

[4.3.3 Configuring Event Analysis 36](#_Toc510019959)

[4.3.4 Data Persistence 36](#_Toc510019960)

[4.4 Deployment Procedure 37](#_Toc510019961)

[4.4.1 Apache Kafka Deployment 37](#_Toc510019962)

[4.4.2 Neo4J Deployment 37](#_Toc510019963)

[4.4.3 Video Pre-Processing Deployment 37](#_Toc510019964)

[4.4.4 Apache Flink Deployment 37](#_Toc510019965)

[4.4.5 Apache Spark Deployment 38](#_Toc510019966)

[4.5 Evaluation 38](#_Toc510019967)

[4.5.1 Produced Data 38](#_Toc510019968)

[4.5.2 Ease of Deployment 39](#_Toc510019969)

[5. Discussion and Conclusion 44](#_Toc510019970)

[5.1 Summary 44](#_Toc510019971)

[5.2 Objectives Met 44](#_Toc510019972)

[5.2.1 Objective One 44](#_Toc510019973)

[5.2.2 Objective Two 44](#_Toc510019974)

[5.2.3 Objective Three 45](#_Toc510019975)

[5.3 Discussion 45](#_Toc510019976)

[5.2 Future Development 45](#_Toc510019977)

[References 46](#_Toc510019978)

# Introduction

## 1.1 Motivation

In the United Kingdom, an estimated that over 1.85 million surveillance cameras record video streams every day, with each person being caught on camera an average of 68 times per day [1]. Fueling the mass deployment of surveillance equipment is its capability to deter criminal activity [2] coupled with enablement of event causality [3]. As a result of mass deployment, CCTV is estimated to have prevented hundreds of thousands of crimes per year [4], providing evidence for thousands of criminal investigations, including being available in 96% of homicide investigations [5]. However, with an estimated 5000 years of footage produced every day it has also been estimated that CCTV is only used in 0.5% of crimes that are recorded [2], [6], and while physical CCTV cameras are an active deterrent to criminal activities, there remains a limited approach to responding to the data produced from CCTV cameras.

Video surveillance is traditionally monitored in one of two ways; passively where it is only investigated in response to an event, or actively where it is viewed live by a single or team of people. Active monitoring and video investigation are time consuming processes, which enable a multi-million pound industry to form around them [7]. As the chance of an event of interest occurring within a video stream decreases, users tend to adopt a more passive approach, which has led to developments that bridge the gap between passive and active monitoring. Sensor driven active monitoring has produced a more efficient approach to video analysis, where movement sensors or external stimuli, trigger a prompt for active monitoring of footage. Within an industrial setting, dedicated hardware has been integrated with sensor-driven active monitoring with increasingly more sophisticated computer vision being used to trigger active monitoring.

With the development of Smart Cities, the capacity of smart video stream analysis needs to be able to scale to data volumes never encountered before within this domain. Large scale applications being with them new considerations; minimizing data transfer, distributing work to enable greater processing throughput, while providing analysis results in a quick enough time to be valuable. To support this capacity and existing video surveillance deployments, a base framework is proposed. It provides common functionality for video processing by default, while allowing extensions to be developed, enabling its adoption in the widest variety of use cases. Furthermore, distributed technologies provide a base to meet the demands of even the largest surveillance networks.

## 1.2 Aim

Propose an extendable and scalable framework for video stream analysis with object, activity and event classification.

## 1.3 Objectives

1. To research existing video processing techniques and available software packages in order to support current computer vision techniques.
2. Develop a framework that provides a minimum viable product of object, activity and event detection, while being scalable and extensible.
3. Develop a use case that will allow the evaluation of the proposed real-time video processing framework.
4. Using the use case defined in objective four, evaluate the frameworks ability to support existing video processing techniques, while meeting bespoke user requirements.
5. Compare and contrast the performance of the proposed framework against existing CCTV technologies and approaches.

## Paper Structure

This will be my paper structure.

# Background and Literature Review

Computer vision applications are often deployed with a variety of desired outcomes, including facial recognition, optical character recognition (OCR), medical image analysis, automatic number plate recognition (AMPR), traffic analysis and footfall analysis. Within computer vision domains, analysis is usually implemented with a pipelined architecture, with a focus on pattern recognition.

## 2.1 Video Processing Methodologies and their Adoption

The exploration of theoretical stages within generic video processing are well documented. Ko et al [8], provides an industrial case setting, with the goal of providing surveillance information to government agencies, that can be used as a basis for common core functionality required of smart surveillance systems. Within this setting functionality can be seen to include background modelling, object segmentation, object classification, object tracking, with the outcome of person identification with behavior and activity analysis (Figure 1). This provides an avenue of interrogation within the realm of smart surveillance systems. An alternative approach is to use black box models for pattern recognition, such as deep learning techniques, that are able to integrate multiple phases of the pipeline described, within the layers of the classification model [9]. Typically, modern video analysis pipelines adopt a combination approach that uses black box models for object detection, object tracking, behavior analysis and event classification but allows for limited interrogation of the intermediate steps.



Figure 1: Adaptation of video processing within a general framework for automated visual surveillance system [8]. This shows common stages involved in providing computer vision applications in the context of automated surveillance systems.

2.1.1 Object Detection is the base of all computer vision applications, providing the ability to accurately identify objects within a frame. Work within this field has relied upon being able to interpret the combinations of pixels correctly to identify an object. The most common method of doing this is using a Haar feature-based cascade classifier [10]. This machine learning approach works by showing a classifier a multitude of images, with some containing the object you wish to detect. Then, by applying features to the image that allow accurate pattern recognition of the object, we can train the classifier to identify the unique signature of the desired object when it is present. From here, given a trained classifier, we extract the features from the images that identify the object, where each feature is a single value obtained by subtracting the sum of pixels under the white rectangle from the sum of pixels under the black rectangle (Figure 2). The composition of the identifying features of an object enable the final model to identify if, and where, an object is present to a degree of accuracy.

Figure 2: Left, an example of Haar cascade features. Right, the adoption of Haar cascade features in detecting a face [11].

Further to this, we are able to improve the performance of detections using an algorithm called Adaboost [12], which allows ranking of object detection features based on their error rate when attempting to successfully identify the object.

In more recent years, with the improvement of Graphics Processing Units (GPU) hardware, advanced detection techniques have been created upon Deep Learning models [9]. Further to this, object detection techniques often utilize motion and background separation methods, enabling them to provide insight into events and activities occurring [13]. Utilizing the foundation of accurate object detection, objects can be given identity between frames.

2.1.2 Object Tracking Techniques enable the observation of object movement vectors, along with the monitoring of object interaction patterns. Work within the object tracking domain build predictive models with online data, meaning they adapt at runtime to provide improved predictions. Algorithms within this domain often model an objects location as a set of positions that each could contain the objects location based on its previous known location [14], [15], [16], with the most common tracking techniques described in Table 1. Given successful employment of object tracking, we can now observe the behaviors of individual objects through a video stream.

|  |  |  |  |
| --- | --- | --- | --- |
| Approach | Overview | Pros | Cons |
| Multiple Instance Learning | Tracks an object by treating its location as a set of positions (‘bags’) that each could contain the objects location based on its previous location. This gives the learning algorithm the responsibility of removing the ambiguity of the exact object location and predicting which instance in each bag is most correct [14]. | Works well when object partially occluded due to its bag representation. | The tracker cannot handle full occlusion of objects well. |
| Median Flow | Tracks an object by selecting a variety of points within the object space and then computes the trajectory for this object, both forwards and backwards in time. This means the tracker is able to compare both trajectories and make accurate predictions of the objects final location [15]. | Works well when an objects trajectory is predictable. | The tracker becomes less reliable under fast or unpredictable motion. |
| Tracking-Learning-Detection | Breaks the tracking problem up into: Tracking, learning and detection. The tracker then follows an object between frames with the detector correcting the tracker based on all observed appearances. The learning estimates the error of the detector and updates it to avoid these errors in future [16]. | Works well under long periods of occlusion. | The tracker does not perform well under large full rotations and can create many false positives. |

Table 1: A comparison of three common approaches to object tracking. This aims to show the diversity in object tracking, while illuminating the costs and benefits to each approach.

2.1.3 Behavior and Activity Analysis allow the assignment of context to object movements, enabling more advanced processing techniques down-stream in the video processing pipeline, such as event classification. Within computer surveillance it is often desirable to identify endangering or suspicious activities [8]. Active work within this field is varied, with successful modelling of behaviors being produced using Markov Models [17], [18], along with the more computationally intense techniques architected with Neural Networks [8]. Markov Model techniques have identified behaviors to precision rates above the 90th percentile [19], however they struggle with noise in the data which can cause accuracy to drop. Other popular methods of behavior analysis include [8]:

* Dynamic Time Warping: a technique for comparing the similarity between two sequences of events. This allows the calculation of the probability that a shown behavior corresponds to a previously known behavior pattern.
* Finite State Machines: models’ behavior as a finite set of states, with transitions between related states. This allows for analysis on real-time video, watching for movements that trigger a tracked object to progress to a different state. A comparison between the transition path through the state machine and known behavior traits enables the identifying of witnessed behaviors.

The understanding of activities within a video stream provides a deep insight into what is transpiring, allowing the modelling of the video context through time.

2.1.4 Event Classification builds upon activity analysis, giving a method for identifying unusual data points within the context of the video stream. Work in this field focuses on building a model to represent the current state of the data, and then compares new data points to the model calculating how far the point deviates from the existing data set. Multiple models are able to provide this, frequently built around clustering techniques [20].

Further to this, the adaptation of One Class Support Vector Machines have provided anomaly detection for short term observed behaviors within video streams [19]. They are able to detect whether a short-term series of points is outside of normal observed behaviors, and provide an abnormal threshold capability that can distinguish which data points require more in-depth anomaly analysis.

Recently, with the wide adoption of Neural Networks, many methods have emerged built around this core architecture that can provide high accuracy detections, at the cost of requiring high performance hardware [21]. Based around learning the distribution of features within the data, the models can detect when a data point falls outside of the normal distribution and classify it as anomalous. A Restricted Boltzmann Machine is a Neural Network model for the learning of distributions and, when multiple are stacked together, they build a Deep Belief Network [21]. However, Deep Belief Networks bring with them a large computational overhead and therefore may not be appropriate for a generic video processing framework.

## 2.3 Distributed Computing and the Cloud

Video stream data is rich and complex and therefore raises significant challenges in data handling and processing when attempting to retain real-time requirements. In order to provide actionable intelligence, analysis not only has to have the functional requirement of being correct, but it must also meet the non-functional requirement of being produced within a time frame that allows actions to be taken with significance.

2.3.1 Parallel Computing allows the concurrent processing of data streams, enabling analysis on large quantities of data, also known as throughput. Within the realm of video processing, there is a need to perform large amounts of calculations on individual frames, in order to provide services ranging from initial object detection to tracking and analysis. As more video streams are added to a system for analysis, there is a need for these expensive operations to scale, therefore we must be able perform these operations in parallel over a distributed set of machines. Current openly available Frameworks, enable the distribution of work within a cluster of machines, vastly increasing the throughput of the overall system (Table 2).

|  |  |
| --- | --- |
| Framework | Functional Details |
| Apache Storm | Apache Storm runs on a distributed cluster of machines, where a topology is defined representing the series of processing stages to apply to each event. A topology is comprised of bolts, with a bolt representing an individual processing stage on an event. This architecture gives Apache Storm power to process billions of events across a distributed set of machines in a single day [22]. |
| Apache Flink | Apache Flink works by expressing data processing as a series of tasks to apply to each event as a directed graph. the sending of messages between stages in the dataflow pipeline make use of buffers, with the backpressure from these buffers being used to control the throughput of the pipeline. This allows Apache Flink to provide throughputs of up to 80 million events per second [23]. |
| Apache Spark | Apache Spark was originally designed for distributed batch processing, but has since been extended for stream processing. It processes events in small micro-batches in order to run them in real-time. It represents data as a resilient distributed dataset that can have operations applied to it [24]. |

Table 2: An overview of popular available frameworks for distributed computing, showing a brief insight into the architecture of each system.

As these technologies offer similar services, performance comparisons between them exist. Under high workloads, it has been found that Apache Flink out performs other systems with regards to latency, however Apache Spark is able to provide the largest processing throughput [25]. Nevertheless, these systems are in their infancy of development and therefore full conclusion about the final performance of each system are hard to make.

2.3.2 Distributed Messaging provides the ability to decouple the different stages of processing, allowing for the addition of new features without effecting existing stages of a processing pipeline. As well as decoupling, video processing analysis relies on creating an immutable log of objects, behaviors and anomalies detected within a given video stream. This immutable log enables the understanding of the causality of events; which is a core component of video processing analytics. Further to this, a distributed messaging layer should be able to scale to multiple video stream inputs without effecting real-time performance requirements. Providing these capabilities is Apache Kafka [26].

Apache Kafka is a distributed streaming platform, which lets applications publish and subscribe to topics of messages. It stores the messages published in a fault tolerant way, partitioning information across multiple machines within a configured cluster. To communicate with Apache Kafka a client performs a high-performance, language agnostic, TCP protocol, meaning any system can speak with it. Further to its core functionality, Apache Kafka is also extremely fast, outperforming traditional messaging systems when handling the producing and consuming of messages between decoupled systems (Figure 5).

Figure 5: A comparison of Apache Kafka’s producer performance (Left), and its consumer performance (Right) against competitor messaging systems [27].

2.3.3 Cloud Computing enables a user to rent the computational power they require, scaling up when demand is high, and reducing when systems become idle. This elasticity is a highly desirable feature when proposing a framework that needs to cater to a large variety of projects and requirements. This enables the distributed processing systems discussed to have resources allocated to them at runtime, when demand is high on an individual stage of the pipeline more machines can be created and added to meet non-functional requirements. Further to this, machines can be destroyed if they are not required, reducing running costs of the system.

The benefits of Cloud computing do not come without risks however; with virtualized hardware exact performance metrics may be unknown until runtime. As a client of a Cloud provider, you are also at the mercy of their availability; if their service becomes unavailable, you are incapable of fixing it yourself. Further considerations to make can include, data confidentiality, software licensing costs and runtime renting costs [28].

## 2.4 Existing Technologies and Approaches

Computer vision is often most successful in batch analysis with extensive post processing, however smart video surveillance requires real-time event classification. This relies upon the computer assuming the active monitoring role rather than to prompt a human to analyses the footage, which is beyond the scope of most existing applications. Table 3 provides a summary of existing technologies that use computer vision applications within the video processing pipeline. It is seen however, that these are often designed for an individual use case or are developed under closed licensing and therefore cannot be built upon.

|  |  |
| --- | --- |
| Product | Functionality Details |
| Nest [29] | Alerts users in real-time, via an app, to event and motion detection events seen on camera. Provides event and anomaly detection and real-time alerts when these occur. However, it requires specialized hardware, is not extendable for development. |
| A Video Analysis Framework for Surveillance System [30] | Gives a novel framework approach to online video analysis using .NET 2.0. Provides object detection, event detection with extensibility to insert new functionality within the application. It is limited by its lack of distributed computing; therefore, it will not be able to scale to all user requirements. Further to this, the product is developer in .NET, requiring a Microsoft workstation to run. |
| DiVA [31] | Gives a distributed video processing framework that uses a database as a message source allowing components to communicate agnostic of technologies adopted. Provides object detection with the goal of detecting object abandonment and removal. Extensibility is provided through the communication of modules/algorithms occurring at a database level. However, it uses a singular database to communicate, rather than a distributed messaging system. It also requires fixed hardware meaning it currently does not make use of Cloud Computing. |

Table 3: A list of current technologies in the market that aim to provide smart CCTV.

# Proposed Framework

## 3.1 System Architecture

Proposed is a scalable framework that supports common video processing techniques by default, while being open for extension to enable domain specific modifications (Figure 6). It adopts a streaming architecture to provide real-time analytics on data generated from the raw video input. Streaming based technologies allow for operations to be applied to individual events as they occur, rather than periodically querying data that is generated. This provides the benefit of real-time feedback as video footage is being transmitted. This does not limit traditional batch processing, as the data is still streamed to a data store as the real-time analytics are being applied, meaning offline querying, that may be too time consuming or expensive do be performed in real-time, is still supported within the framework. Thus, we support traditional batch applications while providing real-time feedback to the user. In order to achieve this the framework design is modular, with modules subscribing to data streams within the distributed messaging system, allowing for compartmentalized operations to be performed agnostic of other systems acting on the same stream.



Figure 6: The proposed video processing architecture, showing the avenues of communication in the distributed system.

3.1.1 Pre-Processing of the raw video input allows for a fine grained control of the volume of data entering the downstream processing stages of the pipeline. This is the first stage of the pipeline, and is the interfacing component between existing camera hardware and the analytics framework deployed. The module is responsible for annotating the video, and extracting all necessary information from each video frame, such as object locations and tracking information, which can then be entered into the analytics pipeline through the messaging system. Using a technique known as Edge Computing [32], applying filtering and detection on the raw video feed to control the amount of data entering the downstream processing stages, the networks capacity is maximized. Thus, this increases the throughput of the pipeline as less unnecessary data arrives at the messaging system. This also allows the quality of data to be maintained by the time it reaches downstream processing services, degraded or faulty readings can be determined before they have a chance to reach the core analytics stages of the pipeline.

3.1.2 Activity Analysis is then performed on the annotated data published to the distributed messaging system. This is done over a distributed processing framework, allowing for parallel identification of activities. Making use of pattern matching techniques, the service consumes the stream of video annotations produced by the pre-processing stage and compares the processed events with its internal model of activity signatures. When an activity is identified it can then be published onto the messaging service for downstream services to analyze.

3.1.3 Event Analysis then engages with the information produced from upstream stages to identify clusters of behavior, looking for events that deviate from the normal. This is supported through the use of an unsupervised cluster detection methodology by default. This enables a user to use event classification techniques without understanding or building complex reasoning models. However, as the service is extendable and modular it can further support complex reasoning models, but this is not an out of the box behavior as it cannot be made generic for all available use cases. If an event of interest occurs the service has the capacity to send notifications to the appropriate user, allowing for further investigation.

3.1.4 Data Storage occurs as the data is streamed between services. The service consumes all messages being sent and persists them to a database. Performed in real-time, the service aims to provide enablement of user interfacing, allowing further services to query the data flowing through the entire pipeline at all stages, and semantically reason about it. This enables offline analysis of the data by users, along with querying and exploration of data produced by the pipeline, agnostic of data producing services.

3.1.5 Sub System Communication is completed through a distributed messaging layer, decoupling systems and maintaining a flexible and extensible application. Communication can then be achieved by using a common Application Program Interface (API) between systems and the messaging layer. The decoupled nature of the sub systems means they can be deployed independently, allowing the most appropriate tool to be used for each area of processing, with individual resource allocations for different parts of the system. This enables users to deploy infrastructure on a per-service basis, giving a fine-grained level of control to avoid over or under allocation of resources to a task. A messaging layer adds complexity to the framework, as systems have the added overhead of indirectly communicating with each other, which is a noticeable cost when considering this approach.

3.1.6 Cloud Based Architecture provides a service that allows clients to rent hardware rather than buying it upfront, giving a flexible way of managing infrastructure depending on specific performance requirements, and the processing framework looks to make use of this. Each individual processing service runs over a distributed set of machines, enabling throughput and latency targets to be met, and with Cloud architectures hardware can be dynamically allocated as required to scale to meet any non-functional requirements. Though, the deployment can be made to a local environment, and considerations can be made as to the most appropriate production infrastructure on a per use case basis [28].

3.1.7 Improvements to previously seen work [30] [31] have been made by making use of Cloud Computing infrastructure, while enabling clusters of machines to distribute work for a single task, rather than just distributing work between different stages of the processing pipeline. The framework identifies the success of existing work; taking a modular approach to design, communicating through a shared messaging layer, allowing extensibility to meet individual requirements. This approach hopes to overcome the challenges of scaling, while maintaining the success of modular design seen in previous work, enabling large scale computer vision applications to become possible.

3.1.8 Limitations can be found with a distributed based approach, the network latency of the infrastructure heavily affects distributed processing, as each node in a cluster must communicate with its counterparts to organize and distribute work. This can drastically reduce the performance of the proposed framework if deployed onto a degraded network environment. Mitigating this, the frameworks deployment to Cloud infrastructure should allow for dynamic network configuration to meet individual requirements, coupled with intense processing happening at the edge of the Cloud. Accompanying this feature, Cloud deployments should be made smooth and approachable enabling users with little infrastructure experience to deploy applications to production.

Further to this, Cloud infrastructure can become expensive as network usage is charged to the user along with the rented computing power of the machines. To combat this, the framework is not linked to a single Cloud provider, or to the Cloud at all, giving the freedom of choice to the adopting user.

## 3.2 System Implementation

The proposed system architecture is implemented as shown in Figure 7, producing the data processing pipeline in Figure 8. The sub systems are written independently and do not share dependencies, allowing them to be modified and extended without the need for modifications to other systems. Accompanying this, support projects have been created allowing the automated deployment of all required distributed processing frameworks to Amazon Web Services (AWS).



Figure 7: The proposed video processing framework implementation, distributed computing technologies (Apache Kafka, Apache Flink, Apache Spark) are adopted to provide high throughput, low latency, processing within a Cloud environment.



Figure 8: The data flow through the application, showing how each processing stage makes use of Apache Kafka to read input and produce outputs.

3.2.1 Sub-System Communication is provided using Apache Kafka. Apache Kafka [26] is chosen due to its ability to scale to support millions of messages per second, along with its low latency [27]. This enables projects of any size to adopt the framework and allows the framework room to scale as demand increases (Figure 9, Figure 10). This data displays how, even Apache Kafka configurations that sacrifice some asynchronous performance for reliability of message storage, still support hundreds of thousands of messages per second. This enables data integrity guarantees to be met while continuing to meet real-time requirements of video processing systems. Further to this, Apache Kafka can act as an efficient buffer between systems allowing for asynchronous communication, reducing time spent waiting for message responses. Buffering also allows slower downstream processes to not impede the processing progress of upstream processes that are able to operate at a faster rate. Enablement of extending the pipeline with slower processing stages is then achieved without effecting the existing throughput of the pipeline as Apache Kafka handles back pressure of any processing stage. Apache Kafka is implemented through its deployment as a standalone system within the framework, making use of Terraform cloud deployment tools discussed within 3.2.8. Once deployed, the individual processing stages are able to publish and subscribe to their appropriate message queues, enabling the input and output streaming of data. With Apache Kafka at the heart of the video processing framework, the individual video processing components are able to communicate with efficient buffering.

Figure 9: Apache Kafka throughput in messages per second [33].

Figure 10: Apache Kafka latency in milliseconds [33].

3.2.2 Video Pre-Processing is implemented as a Python package offering configurable video processing techniques for object detection and tracking. Its responsibility is to process the raw video input stream, and convert it to video annotations that are consumable by downstream processing services. The module is built around a processing pipeline interface, allowing the user to choose which processing techniques are applied to each frame of the video input, and where the results of the processing stages are sent. Video inputs can be accepted from file and from a directly connected USB camera (Figure 9). Further to this, person and car detection is provided using the OpenCV library, with tracking of all detected objects enabled through a Kernalized Correlation Filters model. This dynamic configuration allows for the package to be extended and configured easily to individual domains. The output of the pre-processing stage is sent to Apache Kafka by default, with the annotated video shown on screen. This gives the user the choice to see the annotation information being generated and sent to downstream processes, which aids in visualization of generated data.



Figure 9: An example configuration of the video pre-processing package. This displays how the use of a builder pattern can be adopted to enable easy building of custom processing stages.

3.2.3 Activity Analysis is delivered as an Apache Flink task, that can be executed on any deployed Apache Flink cluster. The implemented task is able to apply its processing operations over all the machines in the cluster, allowing it to operate on multiple input records concurrently. The task offers distributed activity detections for standing, walking and running people, along with the detection of parked and moving cars. Reading the data produced from the pre-processing service via Apache Kafka, the service adopts Apache Flink’s advanced pattern matching technology to look for simultaneous movement events typical of known activity behaviors. The pattern matching library observes a stream and is able to provide data windowing functionality, in which it can detect a defined pattern of events. The service is tuned through configuration files read in at task runtime, allowing for activity identification to be tuned to provide precise results for particular camera installations and viewing angles. Furthermore, the service is made extendable through the Flink pattern matching API, and new patterns can be developed to allow the identification of new behaviors. Once the task is configured it can be submitted to any Apache Flink cluster, enabling it to utilize the deployed distributed architecture when processing incoming data streams.

3.2.4 Event Classification adopts the core libraries provided by the Apache Spark ML library to provide unsupervised event classification. The service reads data produced from the activity analysis service and then uses an unsupervised K-Means clustering model to assign it a cluster of origin and calculate how far the instance deviates from the cluster center. This model is adaptive, as the data is streamed through the system it is able to adjust cluster locations to provide accurate classifications even if data patterns change. The service is tunable through command line parameters, that enable the configuration of the number of clusters that the model should attempt to identify. Once it calculates the events cluster and distance from the cluster center, it published this information to Apache Kafka. This is implemented as an Apache Spark job, and can therefore be submitted to any deployed Apache Spark architecture allowing it to make use of distributed processing techniques. Apache Spark provides data streaming in the form of micro batch processing, which adds an artificial latency to processing, which should be considered if further downstream processes require the output of this service in a low latency manor.

3.2.5 Event Notifications are enabled through a task running on Apache Flink, reading data produced from the event classification service via Apache Kafka, notifications can be sent for events assigned to a specific cluster or for events that deviate from their assigned cluster by a set amount. These rules can be configured at runtime through configuration files, allowing for dynamic business domain rules to be created and changed as more insight into data is achieved. If an event is deemed to require a notification, the current system supports the sending of an email which contains the events unique ID, along with its assigned cluster and deviation score. This enables immediate feedback to the interested party when an event of interest occurs. The email also contains a link to the user interfacing component, allowing the user all the information they require to investigate the event.

3.2.6 Data Storage is achieved through an Apache Flink task that consumes messages from all Apache Kafka topics and writes them to a database. As each service publishes to its output topic, the data storage task consumes the messages along with the other downstream processes. Neo4J is the chosen database by default, offering clustered storage with fast access, it meets all non-functional requirements of the framework. The storage service then, for each message, is able to create a node within the Neo4J database instance. Once a node has been created, the data storage service creates relationships between nodes to show the flow of data relations (Figure 11). This allows for easy data interpretation and semantical reasoning, as the data is modelled through its relationships.



Figure 11: The relationships between entities created, stored within Neo4J. An observed object can be seen within many video frames, while an activity can be observed over many observed objects. Each observed activity has an event classification, which is assigned to an event cluster.

3.2.7 Data Interfacing is provided through the Neo4J packaged interface. This interface provides easy tools for data querying and exploration, accompanied by a visualization tool (Figure 12). This was chosen due to its ease of use, and the fact Neo4J is already being used as our data storage provider. The interface is made available to a user through a website, and offers functionality to export query results and capture images of the data represented as a graph. This allows for the data to be quickly interpreted by non-technical persons, as well as supporting complex graph querying algorithms, such as shortest path finding, which may allow further business insight into the relationships between nodes within the database.



Figure 12: An example query result in the Neo4J dashboard. This shows the clear data visualization capacity of Neo4J. The returned graph is also interactive so you can move and expand data from your original query, giving you power of exploration.

3.2.8 Infrastructure Deployments are made simple in the configured framework through Terraform scripting. Terraform is an Infrastructure as Code language, meaning you can define your required deployment of machines as code and then execute the configuration, which will create the desired infrastructure. The base framework has scripts to define Apache Kafka, Apache Flink, Apache Spark and Neo4J deployments. These scripts create minimum viable clusters for all distributed technologies (Table 4) which also enable extension to create more machines as a solution needs to scale. The Terraform scripts deploy all machines to AWS currently, as this is one of the most popular cloud providers, and therefore has been chosen as the standard for out of the box behavior of the framework. These scripts have been developed separately to any single required system within the framework, and therefore a client who is already running any of the infrastructure is not forced to use this scripting technology. This enables existing clients to adopt the parts of the base framework that are required, without forcing the adoption of the entire framework. Decoupling in this manor aims to achieve a modular adoption process, where a client can configure and deploy services independently, without requiring the pipeline to be completely redeployed if only a single module was updated.

|  |  |
| --- | --- |
| Framework | Deployed Infrastructure |
| Apache Kafka | 3 EC2 t2.micro Ubuntu instances   * 2 EC2 instances run Kafka brokers * 1 EC2 instance runs Zookeeper |
| Apache Flink | 3 EC2 t2.micro Ubuntu instances   * 2 EC2 instances run a Flink Task Manager * 1 EC2 instance runs a Flink Job Manager |
| Apache Spark | 3 EC2 t2.micro Ubuntu instances   * 2 EC2 instances run Spark worker nodes * 1 EC2 instance runs a Spark master node |
| Neo4J | 1 EC2 t2.medium Ubuntu instance   * The instance runs the Neo4J database |

Table 4: A table to show the deployed AWS infrastructure for each required service that is delivered by the framework out of the box.

# Use Case Evaluation

The analysis pipeline described above constitutes a combination of core video processing functionality that enables extension and flexibility to meet a broad range of client use cases. Presented below is a practical use case showing the application of our analysis pipeline to a specific context.

## 4.1 Aim

In order to determine the success of the proposed framework, and evaluate the main aim of presenting an extendable and scalable framework for video stream analysis with object, activity and event classification, the framework must be shown to work end-to-end. The use case aims to show that the different stages or processing are able to successfully interact with each other, while consuming real-time video produced from a currently deployed surveillance camera.

## 4.2 Scenario

Abbey Road in London, pictured on the front of the famous Beetles album Abbey Road, is a busy road crossing and a popular destination for tourists and locals attempting to recreate the album cover. Due to this, it is under live surveillance at all times and has high volumes of pedestrian and vehicle traffic. This presents an opportunity to detect and track cars and people within the live video stream, infer activities of walking, standing and detect real-time insights of events occurring at the crossing. As this is an IP camera deployed to a street in a city, it also aims to show that the services are able to extract and work with video footage created by existing surveillance systems.

## 4.3 Implementation Procedure

In order to configure the base framework, each service requires tuning in order to work for the chosen business domain.

4.3.1 Configuring Video Pre-Processing is done by defining an appropriate processing pipeline within the provided python package. For our single video feed, this is configured to take a single video feed input produced by the deployed surveillance camera. The framework then standardizes the size of each frame, adjusting it to a 400px/400px aspect ratio. This allows the downstream processes within the pre-processing package to work on a smaller sample of pixels, increasing performance. The next stage in the pre-processing pipeline is to take each frame of the video and perform person and car detection, building a bounding box around each identified object. This information is then passed to an object tracker, that is able to distinguish newly detected objects from ones it is currently tracking, adding new objects to its list of tracked objects. Once object locations are identified, the pipeline is configured to send this information to an Apache Kafka cluster, with connection details provided at runtime. Along with this, the annotated video is configured to be displayed to screen, allowing easy visualization of the data being sent to downstream processes (Figure 13).



Figure 13: The local video output of the deployed video processing framework identifying people and cars within the “Abbey Road” real-time video stream.

4.3.2 Configuring Activity Analysis requires little modification to the base framework, will the framework already supporting standing, walking and running, along with parked, and driving activities. This means the package was simply tuned through configuration files to create a sliding window to look for each activity that was appropriate. Along with this the displacement of each object for each activity had to be tuned, so that it accurately correlated objects movements to the correct activity being performed. This task could then be submitted to a running Apache Flink cluster, once deployed.

4.3.3 Configuring Event Analysis is completed through the provided Apache Spark task. The task takes command line parameters to define the number of clusters the K-Means unsupervised model will use, and during this use case it was set to 5. The task then consumed the activity analysis data stream from Apache Kafka, producing a stream of anomaly scores containing the assigned cluster and distance from the cluster center of each activity. This stream was then consumed by an Apache Flink task that sends an email to the user containing an events id, along with its assigned cluster and distance from cluster center when an event had a distance greater than a desired threshold, which was tuned when the service was running. The email also contained a link to the deployed Neo4J instance that was being used for data persistence.

4.3.4 Data Persistence was provided through the deployment of the data storage task, which takes all the Apache Kafka streams and stores them as nodes within Neo4J. The task was then configured to create relationships between the nodes (Figure 11) to allow for reasoning with regards to event causality. This also provided the data interfacing tool through the Neo4J dashboard, and gave visualization and querying capabilities over the produced data.

## 4.4 Deployment Procedure

To deploy the infrastructure required to run the analytics pipeline, the pipeline makes available a series of Terraform scripts. This enables the smooth, reproducible, deployment and management of the production infrastructure.

4.4.1 Apache Kafka Deployment is required before any other stage, as it is the single communication broker between all services. In order to deploy Apache Kafka, the built in Terraform scripts were executed, outputting the dynamic connection details required for all services to talk to Apache Kafka.

4.4.2 Neo4J Deployment enables the storage of all data flowing through the Apache Kafka cluster. The Neo4J database was deployed onto a single machine and its connection credentials were made available to the data storage service, which will be deployed once the Apache Flink infrastructure is running.

4.4.3 Video Pre-Processing Deployment is responsible for parsing real-time video, with the adoption of Edge Computing techniques discussed, this stage needs to be deployed as close as possible to camera source. This service was therefore deployed to AWS region Ireland, as this is the closest region with the hardware requirements necessary to run the service. The service was then started, processing the raw video stream, and sending the annotated object location and tracking data to the deployed Apache Kafka cluster.

4.4.4 Apache Flink Deployment is responsible for producing a cluster of machine capable of running all required Apache Flink task. The required Apache Flink tasks include activity analysis, event alerting and data persistence. The Apache Flink cluster was deployed through the execution of the default scripts. Once the cluster was deployed, each task was started in order, being passed the Apache Kafka credentials of the deployed Apache Kafka cluster, allowing them to communicate through the distributed messaging layer.

4.4.5 Apache Spark Deployment is used to run the event classification service. The Apache Spark cluster was deployed using the default Terraform scripts, with the event classification job being submitted to the cluster for execution on successful deployment.

With the modular deployment process completed, the video analytics pipeline was running successfully over the raw video footage.

## 4.5 Evaluation

With the aim of presenting the pipeline as a working end-to-end entity, to evaluate the system we must certify that the systems interact together successfully, and that information produced by the system is sustainable and presented to the user in an intuitive way that enables understanding. Furthermore, the system must be approachable in its deployment process enabling the complexity of Cloud computing infrastructure to be abstracted away from the adopting user.

4.5.1 Produced Data from the deployed pipeline is made available through Neo4J in real-time. A common use case from the deployment of the pipeline, is to be able to visualize the causality of a particular event. This is the process in which the downstream events contributing to an upstream event, such as the classification of a point to a particular cluster, can be visualized and interrogated.

To support this, the data is stored in an accessible state through the use of Neo4J, allowing for overall visualizations of the running pipeline to be produced, as shown in Figure 14. Further to the ability to collect meta data on the events being captured, the activities that have been identified can also be visualized within 3Dimensional space, allowing for interactive exploration of data formations over time (Figure 15). If we are the interested in a specific event

If we then evaluate the pipelines availability to track down a specific data point, to simulate a user receiving an event notification that they need to investigate. As you can see from Figure 16 we can visualize an event within its data space, in order to gain context as to where the event is located with respect to other data processed by the pipeline. From this point, we can take the unique ID displayed and login to the running Neo4J instance for further investigation.

Within Neo4J queries can be executed over the dataset to find a particular node that meets a criterion, which are written in Cypher Query Language. On execution of the query, Neo4J returns an interactive graph (Figure 17) showing the corresponding data node. We can then make use of the graph database functionality, and expand the nodes relationships to nodes around it. This enables the user to clearly see the anomaly score and cluster associated with this node, while also being able to explore other nodes connected to any of the nodes in question. Enabling the user to explore causality of events through the use or relational modelling within the graph database, we have met the aim of not only proving the sub systems are capable of interacting, but that they produce tangible information that is in a format the user can visualize and explore to gain insight.

4.5.2 Ease of Deployment was achieved through the use of Terraform scripting. This required only a small manual input to configure some initial SSH connectivity settings within the AWS Management Console, before being able to deploy the complete infrastructure in stages. Deployment was smooth due to its separation from any individual task to run. Deploying separate infrastructure machine clusters meant that any problems were compartmentalized and easy to track down, without having to attempt redeployment of the entire system. This also required no knowledge of the underlying machine allocations to execute, as the scripts output connection information on successful completion and therefore a user is able to perform the submission of tasks and management of infrastructure from each distributed systems user interface. Concluding, this enables a user to quickly deploy infrastructure at scale, while still allowing for extension and management of machines through the provided scripts if a more advanced setup is required.



Figure 14: The proposed video processing framework detecting activities within the given video stream.



Figure 15: A plot to show the average displacement of events by the event type, in their observed x position. This shows the clustering of event types in the video stream, and where they are usually found. Isolating an event for investigation from within the displayed graph.



Figure 16: Isolating an event for investigation from within the displayed graph.



Figure 17: Finding the event for investigation in the data store (blue) now allows us to observe the anomaly score given to the event (red), along with the frames in which it was observed (pink). Along with this, we can see the cluster that the anomaly belongs to, and if wanted, can expand the cluster to see other associated events.

# Discussion and Conclusion

## 5.1 Summary

Within this paper and the supporting work, a proposed real-time video processing pipeline has been architected, with the aim of being extendable and scalable. An initial implementation has been produced, along with a use case evaluation of how to adopt the pipeline for a bespoke domain. This has included the deployment procedure and the capabilities of the pipeline for data integration.

## 5.2 Objectives Met

In order to deem the success of the work completed a concluding point is made against each of the initial objectives, enabling a true evaluation of the final projects success.

5.2.1 Objective One: To research existing video processing techniques and available software packages in order to support current computer vision techniques.

During the project we have successfully identified existing technologies, evaluating their capability to provide modular design and extensibility, while observing their limitations with regards to scalability and development opportunity. Further to this, we can conclude on their influence of the proposed analytics pipeline defined within this paper. The modularity previously seen is carried forward to the final analytics pipeline described within this paper, with the pipeline providing a clean separation of technologies and functionality through its supported processing stages. Furthermore, the pipeline improves over previously seen work through its adoption of distributed messaging and processing, enabling users to meet high throughput and low latency requirements, that may not have been met using existing approaches.

5.2.2 Objective Two: Develop a framework that provides a minimum viable product of object, activity and event detection, while being scalable and extensible.

Delivered as part of this work is a framework that ships with object detection and tracking techniques, that are able to convert raw video into video annotations that can be consumed by downstream processes. Downstream processes are independent of each other and consist of activity detection that supports standing, walking, running, parked and driving activities. Event detection is unsupervised and provided in the form a K-Means clustering model. All services are extendable, as shown through the evaluated use case, displaying how modifications and tuning can be made to modules within the framework to meet bespoke use cases. The framework is built upon distributed technologies; Apache Kafka, Apache Flink, Apache Spark, allowing for large scale applications to be supported. This combination of features allows me to conclude that this objective was fully met.

5.2.3 Objective Three: Develop a use case that will allow the evaluation of the proposed real-time video processing framework.

The use case adopted…

## 5.3 Discussion

This will be my discussion.

## 5.2 Future Development

This will be a look into areas of future development.

References

[1] T. Reeve, “How many cameras in the UK? Only 1.85 million, claims ACPO lead on CCTV,” *CCTV Image magazine*, 2011. [Online]. Available: http://www.securitynewsdesk.com/2011/03/01/how-many-cctv-cameras-in-the-uk/.

[2] B. C. Welsh and D. P. Farrington, “Public area CCTV and crime prevention: An updated systematic review and meta-analysis,” *Justice Quarterly*, vol. 26, no. 4. pp. 716–745, 2009.

[3] J. M. Caplan, L. W. Kennedy, and G. Petrossian, “Police-monitored CCTV cameras in Newark, NJ: A quasi-experimental test of crime deterrence,” *J. Exp. Criminol.*, vol. 7, no. 3, pp. 255–274, 2011.

[4] S. J. McLean, R. E. Worden, and M. S. Kim, “Here’s Looking at You: An Evaluation of Public CCTV Cameras and Their Effects on Crime and Disorder,” *Crim. Justice Rev.*, vol. 38, no. 3, pp. 303–334, 2013.

[5] S. Yard, “CCTV in Homicide Investigations,” 2010. [Online]. Available: https://goo.gl/oS5Tgn. [Accessed: 15-Nov-2017].

[6] E. L. Piza, J. M. Caplan, and L. W. Kennedy, “Analyzing the Influence of Micro-Level Factors on CCTV Camera Effect,” *J. Quant. Criminol.*, vol. 30, no. 2, pp. 237–264, 2014.

[7] S. Germain, “A prosperous ‘business’: The success of CCTV through the eyes of international literature,” *Surveill. Soc.*, vol. 11, no. 1–2, pp. 134–147, 2013.

[8] T. Ko, “A survey on behavior analysis in video surveillance for homeland security applications,” *Appl. Imag. Pattern Recognit. Work. 2008. AIPR ’08. 37th IEEE*, pp. 1–8, 2008.

[9] Y. Lecun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553. pp. 436–444, 2015.

[10] P. Viola and M. Jones, “Robust real-time object detection,” *Int. J. Comput. Vis.*, vol. 57, no. 2, pp. 137–154, 2001.

[11] OpenCV, “Facial Detection with Haar Cascades in OpenCV,” 2018. [Online]. Available: https://docs.opencv.org/3.3.0/d7/d8b/tutorial\_py\_face\_detection.html. [Accessed: 30-Jan-2018].

[12] P. a Viola and M. J. Jones, “Fast and Robust Classification using Asymmetric AdaBoost and a Detector Cascade,” *Adv. Neural Inf. Process. Syst.*, no. December, pp. 1311–1318, 2001.

[13] Y. Zhu, N. M. Nayak, and a K. Roy-Chowdhury, “Context-Aware Activity Recognition and Anomaly Detection in Video,” *Sel. Top. Signal Process. IEEE J.*, vol. 7, no. 1, pp. 91–101, 2013.

[14] B. Baben and S. Belongie, “Visual tracking with online Multiple Instance Learning,” *2009 IEEE Conf. Comput. Vis. Pattern Recognit.*, pp. 983–990, 2009.

[15] Z. Kalal, K. Mikolajczyk, and J. Matas, “Forward-backward error: Automatic detection of tracking failures,” in *Proceedings - International Conference on Pattern Recognition*, 2010, pp. 2756–2759.

[16] Z. Kalal, K. Mikolajczyk, and J. Matas, “Tracking-learning-detection,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 34, no. 7, pp. 1409–1422, 2012.

[17] J. Kröckel and F. Bodendorf, “Intelligent Processing of Video Streams for Visual Customer Behavior Analysis,” *ICONS 2012, Seventh Int. Conf. Syst.*, no. c, pp. 163–168, 2012.

[18] T. T. Z. T. T. Zin, P. T. P. Tin, T. Toriu, and H. Hama, “A Markov Random Walk Model for Loitering People Detection,” *Intell. Inf. Hiding Multimed. Signal Process. (IIH-MSP), 2010 Sixth Int. Conf.*, 2010.

[19] P. Antonakaki, D. Kosmopoulos, and S. J. Perantonis, “Detecting abnormal human behaviour using multiple cameras,” *Signal Processing*, vol. 89, no. 9, pp. 1723–1738, 2009.

[20] M. Goldstein and S. Uchida, “A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data,” *PLoS One*, vol. 11, no. 4, 2016.

[21] M. Längkvist, L. Karlsson, and A. Loutfi, “A review of unsupervised feature learning and deep learning for time-series modeling,” *Pattern Recognit. Lett.*, vol. 42, no. 1, pp. 11–24, 2014.

[22] A. Toshniwal *et al.*, “Storm@twitter,” in *Proceedings of the 2014 ACM SIGMOD international conference on Management of data - SIGMOD ’14*, 2014, pp. 147–156.

[23] P. Carbone, S. Ewen, S. Haridi, A. Katsifodimos, V. Markl, and K. Tzoumas, “Apache Flink: Unified Stream and Batch Processing in a Single Engine,” *Data Eng.*, vol. 36, pp. 28–38, 2015.

[24] M. Zaharia *et al.*, “Apache Spark: a unified engine for big data processing,” *Commun. ACM*, vol. 59, no. 11, pp. 56–65, 2016.

[25] S. Chintapalli *et al.*, “Benchmarking streaming computation engines: Storm, flink and spark streaming,” in *Proceedings - 2016 IEEE 30th International Parallel and Distributed Processing Symposium, IPDPS 2016*, 2016, pp. 1789–1792.

[26] Apache, “Apache Kafka,” 2018.

[27] J. Kreps, N. Narkhede, and J. Rao, “Kafka: a Distributed Messaging System for Log Processing,” 2011.

[28] M. Armbrust, A. Fox, R. Griffith, A. Joseph, and RH, “Above the clouds: A Berkeley view of cloud computing,” *Univ. California, Berkeley, Tech. Rep. UCB* , pp. 07–013, 2009.

[29] Nest, “No Title,” 2017. [Online]. Available: https://nest.com/uk/cameras/nest-cam-indoor/overview/. [Accessed: 24-Nov-2017].

[30] N. Suvonvorn, “A video analysis framework for surveillance system,” *2008 IEEE 10th Work. Multimed. Signal Process.*, pp. 867–871, 2008.

[31] J. C. SanMiguel, J. Bescós, J. M. Martínez, and Á. García, “DiVA: A Distributed Video Analysis Framework Applied to Video-Surveillance Systems,” in *International Workshop on Image Analysis for Multimedia Interactive Services (WIAMIS)*, 2008, pp. 207–210.

[32] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, “Edge Computing: Vision and Challenges,” *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, 2016.

[33] J. Kreps, “Apache Kafka Performance Benchmark,” 2014. [Online]. Available: https://engineering.linkedin.com/kafka/benchmarking-apache-kafka-2-million-writes-second-three-cheap-machines.