# Conference Submission

## Architecture



Figure 1: The proposed video processing framework architecture.

The system architecture shows how Edge Computing (Shi, Cao, Zhang, Li, & Xu, 2016) is adopted at the camera source to pre-process the video stream, in order to minimise the data transferred to the Cloud through Apache Kafka. Once Apache Kafka receives the pre-processed video data, more complex data analysis can be computed against it through Apache Flink. This analysis is then written back to Apache Kafka. Apache Spark is then utilised to provide distributed machine learning on the data within Apache Kafka, in a real-time capacity. The results are once again written back to Apache Kafka. Finally, a process writes all data within Kafka to a data store which can be queried through a web-based user interface.

## System Evaluation

The base framework uses Apache Foundation services in order to provided distributed real-time processing, enabling even the largest of clients to make use of the framework. This means all contributing source code within the framework is Open Source allowing the greatest transparency and extensibility when adopting this video processing framework.

### Performance

As the different services within the proposed framework communicate through Apache Kafka, Apache Kafka is critical to the performance of the application. Apache Kafka was chosen due to its high throughput and low latency, along with its reliability (Kreps, Narkhede, & Rao, 2011). Benchmarked on a moderately configured cluster, Apache Kafka provides promising results with regards to load handling and speed of execution (Kreps, 2014).

Figure 2: The throughput of Apache Kafka on a moderately powered multi-node cluster.

During benchmarking, Apache Kafka enabled up to 2.5 million messages to be sent from source per second, which would support even the largest video networks looking at adding real-time analytics to their video streams.

With each of the messages sent through Apache Kafka, we also care about the time it takes from producing the messages to processing it. This is known as latency, and is a key requirement for real-time systems. Kafka has extremely low latency, giving a median score of a 2ms processing time (Figure 3), and is therefore suitable for even the fastest real-time systems.

Figure 3: The latency of Apache Kafka from production of a message to consumption of a message.

Alongside Apache Kafka, Apache Flink and Apache Spark are used for distributed processing of messages, and are therefore key technologies within the system. They are both able to meet extremely performant requirements (Chintapalli et al., 2016), with Apache Spark able to provide a higher throughput of events and Apache Flink giving a much lower latency on a per record processed event. This means when extending the system, the base framework provides implementation capabilities that can meet both throughput and latency requirements, giving the freedom of choice to the implementing user.

### Limitations

The main limitation of distributed processing is the network latency of any deployed system, as each node in a cluster must communicate with its counterparts to organise and distribute work. This can drastically reduce the performance of the proposed framework if deployed onto a degraded network environment. However, this is mitigated by the frameworks deployment to Cloud infrastructure, with intense processing happening at the edge of the Cloud.

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. The framework does however provide ‘Infrastructure as Code’ scripts for Amazon Web Services (AWS).

## Use Case

The proposed framework delivers core video processing functionality that enables extension and flexibility to meet a broad range of client use cases. Presented below is a conceptual use case showing the frameworks successful adoption to a specific domain, providing core insights and analytics into the video footage in real-time.

### Scenario

The Abbey Road in London, pictured on the front of the famous Beetles album “Abbey Road”, is a popular destination for tourists and locals attempting to recreate the album cover for themselves. Due to this, it is under live surveillance by a multitude of video cameras at all times. This presents an opportunity to gain real-time analytics of the activities occurring at the crossing in real-time.

To achieve this the proposed framework is deployed to detect cars and people within the live video stream, extrapolate activities to track walking, standing and people attempting to recreate the album cover photo. It will analyse the activity log for anomalies, alerting the user upon discovery an anomalous event.



Figure 3: The famous Beetles cover “Abbey Road” (<https://www.google.co.uk/search?q=beatles+abbey+road&safe=off&rlz=1C5CHFA_enGB750GB750&source=lnms&tbm=isch&sa=X&ved=0ahUKEwi5rJ_ehO_ZAhUMCsAKHTLaA9IQ_AUICigB&biw=1280&bih=612#imgrc=sJtcKFJFhoc5cM:>)

In order to evaluate the proposed frameworks adoption success, we will focus on the performance and extensibility of the framework, showcasing its flexibility to support bespoke client domains.

### Implementation

As the proposed architecture diagram shows (Figure 1), Edge Computing is adopted in order to maintain real-time requirements and limit the stream of data from the video input source. This enables the detection and tracking of people and vehicles to occur at, or near, the camera source drastically reducing the data load sent over the network.

When identifying cars and people within the video feed, the base framework allows the use of Haar Cascades (Viola & Jones, 2001). This is a common, high-performant, detection methodology that provides an acceptable accuracy rate in identifying objects (Figure 4). The Python package providing core video processing has people and car detection built-in and therefore makes adoption for this use case simple.

Between frames, tracking needs to occur in order to give people and vehicles a persistent identity, enabling the computation of further server-side analytics in identifying object movement patterns. Providing this is a Kernalized Correlation Filters (KCF) tracker (Henriques, Caseiro, Martins, & Batista, 2015), which 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. It looks at overlapping positive regions identified to possibly contain the person or cars location to provide an accurate final decision on the objects location.

The base framework enables video stream processing in the form of a Python application, making use of OpenCV for performant video analysis tools (“OpenCV Library,” 2018).



Figure 4: The deployed video processing framework identifying people and cars within the “Abbey Road” real-time video stream.

Once the video stream processing system was deployed and correctly capturing people and cars, the next phase is to allow it to communicate with the Cloud-based services. This can be enabled through a deployed Apache Kafka cluster on AWS. The video stream processing application publishes messages containing the locations of objects to the cluster which can then be consumed by server-side services to provide more insightful analytics.

In order to gain a deeper awareness into the video stream, and enable the objective of identifying people performing the “Abbey Road” album cover, an activity analysis service was needed. The base framework uses Apache Flink for this capability, and has built in behaviour for detecting walking, standing and running activities. The framework can then be extended to not only detect people standing, but to identify four people standing, with the correct distribution between them, that would be considered an impersonation of the album cover. As Apache Flink comes with advanced pattern matching, this is a simple extension to make to the framework. Configuration files are used to provide fine grained tuning of the activity models, letting the user easily tweak the deployed activity identification models to their specific object tracking data feed.

The activity analysis service can be deployed on a distributed cluster in AWS using the frameworks pre-defined deployment scripts. The output of the service is the locations of the activities it has identified along with the activity type identifier, which are written to the Apache Kafka cluster.

Given the knowledge of the activities being performed, and their locations, an anomaly detection service can be utilised to identify unusual activities within the video feed, and alert the user when something that deviates from the norm is detected. Providing this service, the base framework makes use of Apache Spark to deploy a distributed K-Means unsupervised (<http://theory.stanford.edu/~sergei/papers/vldb12-kmpar.pdf>) machine learning model. This service can run on AWS in its own multi-node cluster. The model is fed the normalised activity locations, along with a dimension for the activity type, and is then able to calculate how far an activity deviates from the centre of common activities. This permits the detection of not only activities performed outside of their normal locations within the video stream, but also activities of an unusual type. This offers the flexibility to identify the album cover activity as an anomaly if it is not being performed often, however if there is a series of video that has frequent similar activities, the model can adjust itself to detect that as the normal, giving it the ability to adapt to changing behaviours.

The service then outputs the deviation of a given activity from the normal to Apache Kafka, which can then be consumed by a simple service that sends an alert to the owner of the camera that either an anomalous action has been performed or that an action of a specific type has been seen, allowing the captured video to be investigated. If the anomaly detection service is configured to weight the “Abbey Road” activity as anomalous then this can aid in the aggregation of all video footage of people performing the action.

As the base framework comes with deployment scripts written in Terraform for all the discussed technologies; Apache Flink, Apache Spark, Apache Kafka, the deployment of the entire application to AWS is simple. The scripts require minimum setup, and one click deployments of individual services, meaning we could include new services within the processing pipeline as and when they are configured.

### Performance

As this is a single camera use case, where we produce a message for each frame, containing the objects we are tracking, we only require a throughput of around 10-15 messages per second. As shown in the performance section, the throughput of the technologies chosen within the base framework allow hundreds of thousands of messages to be processed per second, enabling our real-time requirements with no modification to the base framework.

### Conclusion

In conclusion, we can see through this use case that the framework can accommodate bespoke deployments with only small modifications to domain specific functionality. Further to this, it is able to meet real-time requirements through its use of distributed technologies, without the adopting user having to deal with complex infrastructure deployments, allowing them to focus on their unique application.

To extend the framework, more modules can be developed that offer other areas of important functionality. These can include facial recognition, as well as the application of more advanced Deep Learning technologies to provide more accurate classifications of activities and anomalies.

## References

Chintapalli, S., Dagit, D., Evans, B., Farivar, R., Graves, T., Holderbaugh, M., … Poulosky, P. (2016). Benchmarking streaming computation engines: Storm, flink and spark streaming. In *Proceedings - 2016 IEEE 30th International Parallel and Distributed Processing Symposium, IPDPS 2016* (pp. 1789–1792). https://doi.org/10.1109/IPDPSW.2016.138

Henriques, J. F., Caseiro, R., Martins, P., & Batista, J. (2015). High-speed tracking with kernelized correlation filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *37*(3), 583–596. https://doi.org/10.1109/TPAMI.2014.2345390

Kreps, J. (2014). Apache Kafka Performance Benchmark. Retrieved from https://engineering.linkedin.com/kafka/benchmarking-apache-kafka-2-million-writes-second-three-cheap-machines

Kreps, J., Narkhede, N., & Rao, J. (2011). Kafka: a Distributed Messaging System for Log Processing. Retrieved from http://notes.stephenholiday.com/Kafka.pdf

OpenCV Library. (2018). Retrieved from https://opencv.org/

Shi, W., Cao, J., Zhang, Q., Li, Y., & Xu, L. (2016). Edge Computing: Vision and Challenges. *IEEE Internet of Things Journal*, *3*(5), 637–646. https://doi.org/10.1109/JIOT.2016.2579198

Viola, P., & Jones, M. (2001). Robust real-time object detection. *International Journal of Computer Vision*, *57*(2), 137–154. https://doi.org/http://dx.doi.org/10.1023/B:VISI.0000013087.49260.fb