On-demand Serverless Video Surveillance with Optimal Deployment of Deep Neural Networks

Unai Elordi[1,2], Luis Unzueta[1], Jon Goenetxea[1], Estíbaliz Loyo[1], Ignacio Arganda-Carreras[2,3,4] and Oihana Otaegui[1]

[1]Vicomtech, Basque Research and Technology Alliance (BRTA), San Sebastian, Spain
[2]Basque Country University (UPV/EHU), San Sebastian, Spain
[3]Ikerbasque, Basque Foundation for Science, Bilbao, Spain
[4]Donostia International Physics Center (DIPC), San Sebastian, Spain

Keywords: Video Surveillance, Serverless Computing, Deep Neural Networks Optimizations.

Abstract: We present an approach to optimally deploy Deep Neural Networks (DNNs) in serverless cloud architectures. A serverless architecture allows running code in response to events, automatically managing the required computing resources. However, these resources have limitations in terms of execution environment (CPU only), cold starts, space, scalability, etc. These limitations hinder the deployment of DNNs, especially considering that fees are charged according to the employed resources and the computation time. Our deployment approach is comprised of multiple decoupled software layers that allow effectively managing multiple processes, such as business logic, data access, and computer vision algorithms that leverage DNN optimization techniques. Experimental results in AWS Lambda reveal its potential to build cost-effective on-demand serverless video surveillance systems.

1 INTRODUCTION

Serverless computing is a cloud-native platform that hides server usage from developers and runs developer code on-demand, automatically scaled, and billed only for the time the code is running (Castro et al., 2019), under the scope of the Function-as-a-Service (FaaS) paradigm. It represents an evolution in cloud computing, which matches better the original expectations for being treated as a utility (Ishakian et al., 2018). Its two key features are cost (pay-as-you-go billing with millisecond granularity) and elasticity (scaling from zero to "infinity"). It allows developers to concentrate on providing a piece of code (function) to be executed by the serverless computing platform and to delegate all their operational complexity and scalability to the cloud provider without requiring a high level of cloud computing expertise.

Most relevant cloud providers, such as Amazon, IBM, Microsoft, and Google, have already released serverless computing platforms, which are gaining popularity due to their simplicity and economic advantages. However, all these advantages over "serverfull" architectures come at the expense of some limitations of the current stateless platforms (Ishakian et al., 2018), namely:

- The stateless nature of functions, which prevents them to be executed, relying on the serverless platform runtime to maintain the state between invocations to optimize performance.
- The lack of access to GPUs (despite very recent initiatives to solve this limitation (Kim et al., 2018)), which prevents deployed algorithms from making use of high parallelization capabilities within function instances.
- Cold" starts, i.e., additional latencies that occur when the serverless function is invoked for the first time, due to the required setting-up of containers (part of the core capability of serverless platforms) and bootstrapping.
- Scalability limits, i.e., despite their high scalability capabilities they cannot scale up to "infinity.
- Space constraints for the deployed program, i.e., the main code, its dependencies, and the required resources (e.g., data files).

Elordi, U., Unzueta, L., Goenetxea, J., Loyo, E., Arganda-Carreras, I. and Otaegui, O.
On-demand Serverless Video Surveillance with Optimal Deployment of Deep Neural Networks.
DOI: 10.5220/0010344807170723
In *Proceedings of the 16th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP 2021) - Volume 4: VISAPP*, pages 717-723
ISBN: 978-989-758-488-6

These limitations are especially relevant to build an on-demand video surveillance system (VSS) with computer vision algorithms that involve the deployment of Deep Neural Networks (DNNs), as their complexity is significantly higher for hardware platforms with limited computational resources (Bianco et al., 2018). Thus, very recently important efforts are being done towards the goal of optimizing DNNs, such as new methodologies (Elordi et al., 2018) (Frankle and Carbin, 2019), new microprocessor classes (e.g., Intel's VPUs (Intel, 2019) and Google's TPUs (Google, 2019) and new software tools for DNN model optimization included in deep learning frameworks (e.g., TensorFlow (Google, 2020), PyTorch (Facebook, 2019) and OpenVINO (Intel, 2020)). However, since both DNN optimization techniques and serverless architectures are still at early stages, few works have yet focused on the optimal deployment of DNNs on serverless platforms, tackling simultaneously the characteristics of both components.

Our main motivation is to help building cost-effective on-demand VSSs, leveraging (1) the latest advances of DNN optimization techniques for inference purposes along with (2) tailored deployment strategies to make the most of current FaaS architectures. Although this paper is focused on optimal DNN deployment in serverless environments, our approach considers the security and privacy measures to preserve the biometric data on VSS environments (Biometrics Institute, 2020).

This work represents a step forward in distributed computational VSS infrastructures and the Video-Surveillance-as-a-Service (VSaaS) paradigm (Limna and Tandayya, 2016). We have taken AWS Lambda (Baird et al., 2017) as the baseline to design our methodology.

2 SERVERLESS VSS PLATFORM

The FaaS platforms are materialized in function instances, which have two stages (Baird et al., 2017). The first stage begins when the FaaS function is invoked for the first time, creating an isolated runtime environment with the necessary resources. This process takes additional time to be completed and, consequently, this stage is called the *cold start* stage. When the container initialization is finished, the remaining function instances are executed concurrently. This second stage is called the *warm* stage.

Wrong management of resources in the initialization process and the concurrent instances could drastically increase the cold start and serverless execution (warm stage) time. Therefore, the key is to identify strategies to minimize processing time in both stages for a good quality of service. In the following, we summarize the current performance strategies presented in the literature (Baird et al., 2017) (Bardsley et al., 2018).

- Concise function logic if 3rd-party dependencies are required, avoid using open-source packages. Since their general-purpose and 3rd-party interdependency nature, open-source packages include more functionalities than required and, thus, can cause a significant slowdown in cold start time and increase processing time.
- Third-party dependencies: limit the space and the use of third-party libraries to match the serverless function storage limitations.
- Resource management: limit the reinitialization of local variables on every instance. Instead, use global/static variables or singleton patterns to handle the application scope variables.
- Allocated function memory: finding the trade-off between the configuration of computing resources and execution cost can be the key to optimal serverless execution.
- Language agnostic advice: the interpreted programming languages achieve faster initial invocation time, while compiled languages perform best in the warm stage.
- Keep the container in the warm state: make preconfigured periodical calls to serverless functions to avoid changing to a cold stage.

Although these strategies are available for general serverless architectures, the complexity of DNN models (Bianco et al., 2018) requires a deep analysis of DNN model deployment to cope with the serverless platform limitations. With that purpose, we present a FaaS architecture with tailored DNN optimization strategies to maximize inference efficiency.

The proposed serverless architecture is illustrated in Figure 1, together with the lifecycle of the processing pipeline, where each processing task is numbered from 1 to 11. This pipeline contains two main components: the initialization process (from step 1 to 7) and the on-demand invocation task (from step 8 to 11). The event controller shown in the architecture represents the event-triggering design of FaaS platforms (see Figure 1). In this context, each input-image source triggers an event to the FaaS function. In terms of security, the images are stored in a Virtual Private Cloud (VPC). Also, the image data is encrypted.

Following the serverless strategies described above, the designed FaaS function references the resources in the global scope (software layers) and the processing workload relies on the handler function.

3 INITIALIZATION PROCESS

When the serverless function is invoked for the first time (cold start), the initialization process begins, and the warm-up process initializes the runtime execution container along with the software layers (step 2-3). Then, DNN models are downloaded to the runtime container (step 5-6). Finally, the DNN models are loaded along with library initializations.

FaaS functions should be simple and concise. Besides, FaaS architectures are based on ephemeral storage, which means data will be erased when the function finishes. Based on these requirements, we decoupled the functionalities into three layers which are shared across serverless function instances:

- *Deep Learning (DL) layer:* in charge of handling DNN workload operations such as model loading and inference processing, along with pre- and post-processing low-level image operations for Computer Vision (CV).
- *High-Level Algorithm (HLA) layer:* containing the library for complex CV pipelines such as face detection, face recognition and body pose detection, supported by several DNN models for inference processing.
- *Business Logic (BL) layer:* which provides utilities deal with for accessibility to I/O operations, communications, and business logic algorithms.

Ideally, an optimal initialization process will preserve the accuracy and the inference latency of DNN models (Bianco et al., 2018) under FaaS limitations such as storage size, memory consumption, and computing resources. Model compression techniques, such as pruning and quantization (Han et al., 2015), reduce the size of the DNN files, and therefore the required amount of memory to load the compressed model, lessening the cold start delay. These compression techniques are especially relevant when several DNN models are loaded into a layer, because of the rigorous constraint of storage size on FaaS platforms. However, these techniques require special attention in the deployment, since too much compression could affect the accuracy of the models (Liu et al., 2018).

FaaSification consists in changing the execution runtime from monolithic architectures (code processed in the same execution unit) to FaaS architecture. Based on (Spillner et al., 2017), this process depends on the Atomic Units (AU), which depending on the level of complexity (3rdparty dependencies, inter-function dependencies, etc) is classified as shallow (AU: functions or method), medium (AU: lines of code) and deep (AU: instructions). Since the complexity of the DNN processing algorithm lies in three functions (load model, DNN inference, post-processing), our approach is deployed following a shallow FaaSification supported by HLA and DL layers.

4 ON-DEMAND INVOCATION TASKS

After the first invocation of a serverless function instance, the system verifies that all resources are in the warm state and ready for DNN inference. Next, several instances of the handler function are triggered by the input data. These handler instances execute a set of CV algorithms that involve multiple DNN inferences (step 10). Supported by HLA and DL layer to process CV tasks and DNN inferences, when the FaaS function finishes, the BL layer encodes the algorithm output in the preferred output (step 11).

The serverless platform capabilities to offload the computation across several instances could leverage an impressive DNN inference throughput. However, the virtualization nature of this platform hardware resources relies on a bottleneck, especially, when CV processing tasks require to process many DNN models at the same pipeline.

Based on the analysis of the computational complexity of DNN models (Bianco et al., 2018), choosing the ones that lower inference time while preserving the accuracy is crucial for this type of architecture. Moreover, vectorization programming libraries such as Single Input Multiple Output (SIMD) instructions and multi-threading-block libraries provide extra processing power.

The assigned memory to each FaaS function instance plays an important role in performance optimization because more memory per function means more resources for the handler function, but also a higher price per execution. On the contrary, FaaS instances are billed by function execution time, so less time per function means a lower price. Thus, a good trade-off between allocated memory and function execution cost becomes an essential strategy.

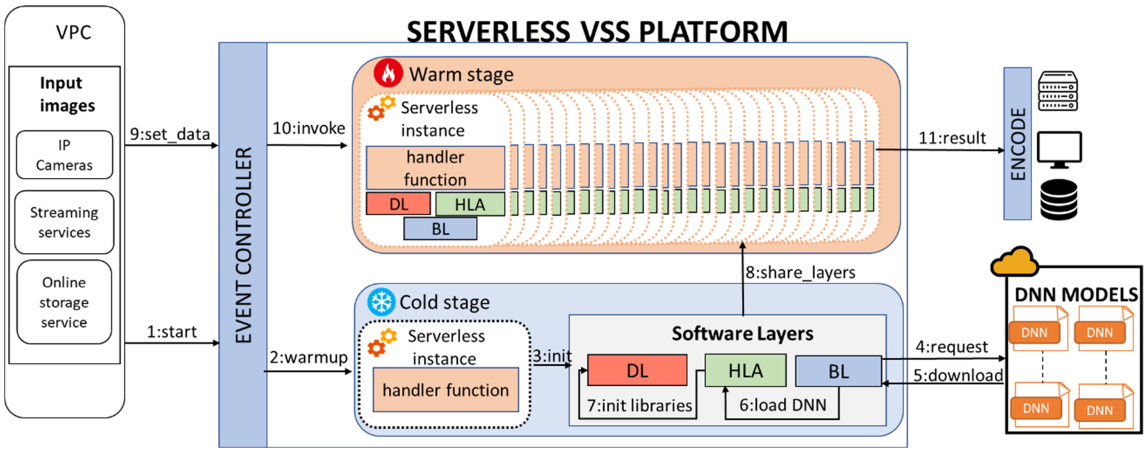

Figure 1: The proposed serverless video surveillance system architecture.

5 VSS CASE STUDY AND EXPERIMENTS

We evaluate the potential of our approach in the following case study: a VSS that periodically receives (every few minutes) images acquired by several surveillance cameras to detect human presence and recognize registered individuals in uncontrolled environments.

Deep Learning models are complex and require a huge amount of processing power. A Deep Learning model inference lies in Matrix-multiplication, regularization, and the number of weights. So, choosing the most optimal DNN with minimum latency and maximum accuracy is a crucial strategy. More specifically, in this VSS we deploy four DNNs trained for the following purposes:

- *Camera coverage detection (CM):* a MobileNet v1-based image classifier to detect whether the image comes from a camera that has been covered (by a hand, a sticker, etc.) or not.
- *Human body points detection (BP):* an OpenPose-based regression model with MobileNet v1 as the backbone to detect people's body landmarks.
- *Human face landmarks detection (FL):* a classic convolutional design-based regression model that localizes both eyes, nose tip, and mouth corners in a cropped facial image.
- *Human face reidentification (FR):* a MobileNet v2-based facial feature extractor for reidentification purposes. The extracted facial features are compared with the registered ones to determine whether they correspond to registered individuals.

The table 1 shows the performance parameters of the selected models.

Table 1:Selected DNN model parameters.

NAME	Complexity (GFlops)	AVG Precission (Mp)	AVG Precission(%)
CM	0.569	4.24	70.9
HP	15.435	4.099	42.8
FL	0.021	0.191	92.95
FR	0.588	1.107	99.47

We have taken AWS Lambda as a baseline to design and test our methodology. The source code is written in Python language. We used OpenVINO as DNN framework and OpenCV for CV algorithms. Also, we used the AWS boto3 library for I/O operations. Since video surveillance environments manage biometric data, to preserve the security of user privacy, we stored all images in a Virtual Private Cloud (VPC) along with an encrypted Amazon S3 storage service. We also have given the minimum and only necessary permissions to the handler lambda function. Finally, we monitor function calls with Amazon X-Ray. The low-economic impact to process 10,000 images with different batch sizes per request and memory configurations per function is shown in Figure 2. Notice the minimum memory to support the VSS application logic is 704MB. The cost calculation is based on the following equation:

$$\text{cost} = nr*((0.0009765625*am)*(0.001*ru(rd,m)*mcc+mrc \qquad (1)$$

Where:

- nr: number of requests in a month
- am: allocated memory (in MB)
- rd: request duration (in ms)
- ru: round up operation to the nearest M multiple (m=100 ms)
- mcc: monthly compute charges (0.0000166667 USD/GB-s)
- mrc: monthly request charges (0.0000002 USD/request)

This cost experiment evinces that more images per request involve cost-saving, especially when the allocated memory function is higher. However, despite the cost fluctuation of the first three configurations (704MB to 1536MB) being negligible, the price evolution of the remaining configurations is increased from 13.38% (2048MB) to 61%. (3008MB). Despite the AWS Lambda free tier offers 1 million Amazon Lambda function instances, these function instances are activated with images (put request) coming from S3 online storage service. This S3 free tier offers 2000 put requests (images) in a month. So, for this experiment, the Amazon free tier is discarded because it represents only the 1% of the experiment.

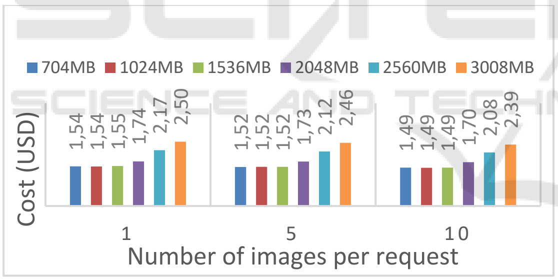

Figure 2: Average cost to process 10K images with VSSH in AWS Lambda. The horizontal axis represents image batch size per request (1,5,10) and the colour bars represent the allocated memory per function, from 704MB to 3008MB. AWS free tier is not included in this experiment.

Figures 3 and 4 analyse the influence of the cold start delays according to local and global scope resource management strategies. In the local scope strategy, all initialization process is executed in the handler function while the global scope initializes all resources before the first handler function. Each figure contains three different lines which represent container setup time (blue), function runtime init time (red), and function code execution total time (green). The container setup is the time delay to create an isolated image container. In the function runtime init we evaluate the time delay of the serverless function resources (loading external resources, classes initializations, loading 3rdparty libraries, code downloading). Finally, the function code execution calculates the total execution time of a serverless cold instance.

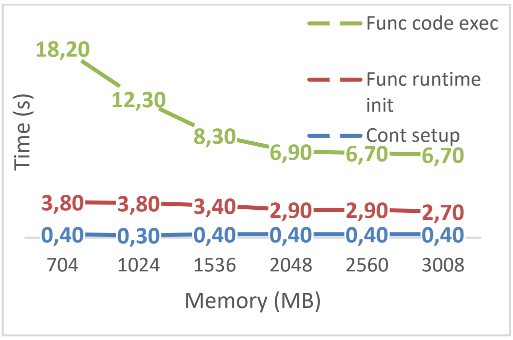

Figure 3: Cold start time analysis of the global scope strategy according to the amount of allocated memory per function (from 704MB to 3008MB).

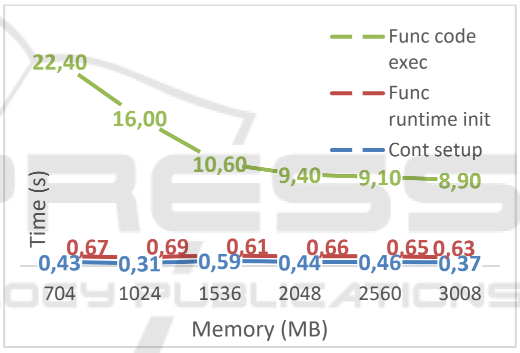

Figure 4: Cold start time analysis of the local scope strategy according to the amount of allocated memory per function (from 704MB to 3008MB).

This analysis of the resource management strategies reveals that initializing the resources in the global scope improves the performance of the cold start delays (about 2-4 seconds difference). Also, the increased time delay of the global scope in function runtime initialization is due to the DNN models, code, and libraries are loaded in this step.

As it was expected, as far as the allocated memory per function instance increases, the cold-start time delay is reduced in both scope strategies. This time reduction is especially visible when the allocated memory is between 704 and 1536MB. In contrast, the container setup's minimal time variations reveal that the FaaS container initialization does not depend on the allocated memory per function instance.

To analyse the cost-worthiness of serverless computing deployment, Figure 5 unveils that our FaaS architecture leverages an outstanding performance with an important time saving from hours that would be needed with an off-the-shelf PC to minutes (our approach). Also, the influence of the allocated memory per function instance is shown in Figure 3, where the reduction of the processing time is very significant, especially between the 704MB and 1536MB configurations.

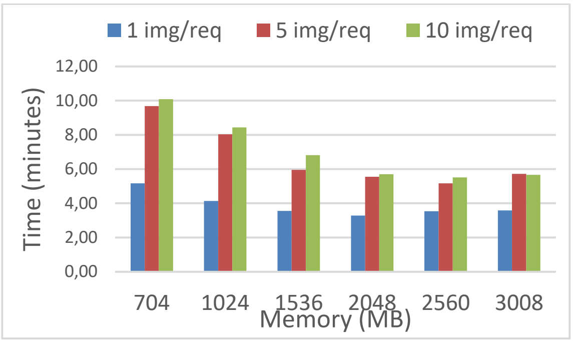

Figure 5: Total times to process 10K images with the VSS in AWS-Lambda. The colour bars represent image batch size.

Considering the economic and the time performance analysis shown in Figures 2, 3, and 4, we conclude that the optimal allocated memory per function remains on 1536MB. Also, as observed in Figure 3, the optimal way to achieve the maximum processing throughput is processing one image per each FaaS instance.

6 DISCUSSION AND CONCLUSION

The FaaS platform environment offers a suitable distributed execution model to provide parallel processing at a high scale. Nevertheless, the resource limitations of this platform collide with the DNN complex environment.

To overcome this challenge, we have presented a methodology to optimally deploy several DNN models to FaaS platforms supported by the latest CV techniques to maximize the DNN processing performance at minimum cost.

We have also evaluated a VSS case study supported by experimental results that reveal an outstanding performance improvement of our serverless architecture. Furthermore, we conclude that the major bottleneck lies in the processing of each FaaS function, while the influence of the memory allocation per function is visible in the processing speed. Nevertheless, there is a large room for improvement in reducing the DNN complex environment, while the bottleneck could be addressed by analysing the possibilities of distributing the DNN processing into multi-tenant systems.

ACKNOWLEDGEMENTS

This work has been partially supported by the program ELKARTEK 2019 of the Basque Government under project AUTOLIB.

REFERENCES

Baird, A., Huang, G., Munns, C., and Weinstein, O. (2017). *Serverless Architectures with AWS Lambda: Overview and Best Practices* [White paper].

Bardsley, D., Ryan, L., and Howard, J. (2018). Serverless Performance and Optimization Strategies. In *Proceedings of the IEEE International Conference on Smart Cloud (SmartCloud)*. pages 19–26.

Bianco, S., Cadene, R., Celona, L., and Napoletano, P. (2018). Benchmark analysis of representative deep neural network architectures. *IEEE Access*, 6:64270–64277.

Biometrics Institute. (2020). The Three Laws of Biometrics. https://www.biometricsinstitute.org/the-three-laws-of-biometrics/

Castro, P., Ishakian, P., Muthusamy, V., and Slominski, A. (2019). The server is dead, long live the server: Rise of serverless computing, overview of current state and future trends in research and industry. *arXiv preprint arXiv:1906.02888.*

Elordi, U., Unzueta, L., Arganda-Carreras, I., and Otaegui, O. (2018). How can deep neural networks be generated efficiently for devices with limited resources? In *Proceedings of the International Conference on Articulated Motion and Deformable Objects*, pages 24–33.

Facebook. (2019). PyTorch torch.optim. https://pytorch.org/docs/stable/optim.html

Frankle, J. and Carbin, M. (2019). The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Google (2019). Cloud TPU. https://cloud.google.com/tpu

Google (2020). TensorFlow model optimization. https://www.tensorflow.org/lite/performance/modeloptimization

Han, S., Mao, H., and Dally, W. J. (2015). Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. *arXiv preprint arXiv:1510.00149.*

Intel (2019). Intel Movidius VPU. https://www.movidius.com/solutions/vision-processing-unit.

Intel (2020). OpenVINO toolkit. https://software.intel.com/en-us/openvino-toolkit

Ishakian, V., Muthusamy, V., and Slominski, A. (2018). Serving deep learning models in a serverless platform. In *Proceedings of the IEEE International Conference on Cloud Engineering (IC2E)*, pages 257–262.

Kim, J., Jun, J., Kang, D., Kim, D., and Kim, D. (2018). GPU enabled serverless computing framework. In *Proceedings of the Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*, pages 533–540.

Limna, T. and Tandayya, P. (2016). A flexible and scalable component-based system architecture for video surveillance as a service, running on infrastructure as a service. *Multimedia Tools and Applications*, 75(4):1765–1791.

Liu, Z., Sun, M., Zhou, T., Huang, and G., Darrell, T. (2018). Rethinking the value of network pruning. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Spillner, J., Mateos, C., and Monge, D. A. (2017). Faaster, better, cheaper: The prospect of serverless scientific computing and HPC. In *Proceedings of the Latin American High Performance Computing Conference*, pages 154–168.