Video Processing with Serverless Computing: A Measurement Study

Miao Zhang
Computing Science School
Simon Fraser University
mza94@sfu.ca

Yifei Zhu
Computing Science School
Simon Fraser University
yza323@sfu.ca

Cong Zhang
University of Science and Technology of China
congz@ustc.edu.cn

Jiangchuan Liu
Computing Science School
Simon Fraser University
jcliu@cs.sfu.ca

ABSTRACT

The growing demand for video processing and the advantages in scalability and cost reduction brought by the emerging serverless computing have attracted significant attention in serverless computing powered video processing. However, how to implement and configure serverless functions to optimize the performance and cost of video processing applications remains unclear. In this paper, we explore the configuration and implementation schemes of typical video processing functions deployed to the serverless platforms and quantify their influence on the execution duration and monetary cost from a developer's perspective. Our measurement reveals that memory configuration is non-trivial. Dynamic profiling of workloads is necessary to find the best memory configuration. Moreover, compared with calling external video processing APIs, implementing these services locally in serverless functions can be competitive. We also find that the performance of video processing applications could be affected by the underlying infrastructure. Our work provides guidelines for further *function-level* optimization and complements the existing measurement studies for both serverless computing and video processing.

CCS CONCEPTS

• **Networks → Network measurement**; • **Computer systems organization → Cloud computing**; • **Information systems →** *Multimedia information systems.*

KEYWORDS

Video Processing, Serverless Computing, Serverless Functions

ACM Reference Format:
Miao Zhang, Yifei Zhu, Cong Zhang, and Jiangchuan Liu. 2019. Video Processing with Serverless Computing: A Measurement Study. In *29th ACM SIGMM Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV '19), June 21, 2019, Amherst, MA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3304112.3325608

1 INTRODUCTION

The proliferation of video streaming services [3] and the recent advances in computer vision algorithms boost the demand for video processing [14]. Videos either need to be transformed into different formats to guarantee smooth playback on different devices and in different network conditions (i.e., processing for delivery) or need to be analyzed automatically so that the information in it can be extracted to help further decision making (i.e., processing for understanding). Unfortunately, video processing for both delivery and understanding is inherently resource-intensive, which means it is always costly and time-consuming [4]. For example, the video object detector [16], which powers the winning entry of ImageNet VID challenges 2017 [11], processes only 1.2 frames per second on an Nvidia Tesla K40 GPU. To accelerate the video processing speed and reduce costs, previous studies have focused on migrating video processing tasks to the resource-rich cloud [5, 10, 12]. Yet constrained by the operation and pricing model of the prevalent infrastructure as a service (IaaS) cloud, users inevitably need to reserve and manage cloud resources by themselves and pay for the idle server time. Further, these heavy-weight cloud server instances cannot scale up or down flexibly to meet the changing workloads, especially for those that require near real-time processing.

As a new execution model, serverless computing is emerging to overcome the weaknesses of traditional cloud computing models [7]. It offloads the task of server provisioning and management from developers to platforms. In the serverless platform, developers only need to break up application codes into a collection of stateless functions and set events to trigger their executions. Platforms are responsible for handling every trigger and scaling precisely with the size of workloads. The light-weight virtualization techniques used in serverless computing, represented by containerization [1], further enable function instances to spin up or down in milliseconds. As the rapid evolution of serverless computing, it has been implemented in commercial offerings and open source projects, such as AWS Lambda [1], Google Cloud Functions (GCF) [2], and Apache OpenWhisk [3]. In addition, commercial serverless offerings charge users at a fine-grained timescale (100ms) and almost achieve the long-promised "pay-as-you-go" pricing strategy.

Given the significant advantages in scalability, startup delay, and pricing, industry experts and researchers from academia have shown great interests in applying serverless computing in video processing. For example, existing studies [2, 4] have shown the ability of serverless computing in executing massively parallel functions to speed up video processing. Industry practices also prove that serverless computing is an attractive choice of building scalable and cost-effective video processing applications [4,5]. However, these

[1]https://aws.amazon.com/lambda/
[2]https://cloud.google.com/functions/
[3]https://openwhisk.apache.org
[4]https://aws.amazon.com/solutions/case-studies/vidroll/
[5]https://aws.amazon.com/solutions/case-studies/netflix-and-aws-lambda/

Table 1: Pricing Schemes of Serverless Computing Platforms (beyond free tiers)$^{\alpha}$

	Symbol	AWS Lambda	GCF
Price per invocation	I	\$0.0000002	\$0.0000004
Memory (CPU)	M (P)	{128(p), ..., 3008(23.5p)}	{128(200), 256(400), 512(800),1024(1400), 2048(2400)}
Price per 100ms	C	$10^{-10} * 16.28M$	$10^{-10} * (2.44M + 10P)$

$^{\alpha}$: The unit of memory size is MB, the unit of CPU is MHz and the unit of price is US dollar; p is unknown to users.

attempts are still in their infancy. How to implement and configure serverless functions to optimize the performance and cost of video processing applications remains unclear.

In this paper, we conduct extensive measurements on two leading serverless computing platforms (AWS Lambda and GCF) to study the impact of user-controllable knobs on the performances of video processing applications from a developer's perspective. Specifically, we quantify the impact of function resource allocation (e.g., memory size), function implementation schemes (local implementation or external APIs), and platform selection (AWS Lambda or GCF) on typical video processing applications (e.g., transcoding, face detection). As an argument for platform selection, we also identify the effect of underlying infrastructures on applications' performance. Our primary measurement insights include:

1. Memory configuration for video processing functions deployed on serverless computing platforms is non-trivial, which is highly related to the workload types. The largest memory size does not always lead to the most beneficial result. Dynamic profiling of the workloads' demand for resources is necessary to find the best memory configuration.
2. To perform complex video processing tasks in serverless functions, calling external APIs is much more expensive than using local implementations. For example, running a pre-trained model to perform face detection locally can achieve comparable accuracy and latency to external APIs at a much lower cost.
3. The performance of video processing functions is platform dependent. In particular, AWS Lambda has more advantages than GCF in terms of execution duration and monetary cost under the same configuration and implementation scheme. However, the performance of AWS Lambda is affected by the underlying infrastructure and is not as stable as that of GCF.

2 BACKGROUND

Modern applications deployed in the cloud tend to be increasingly complicated and resource-intensive, which makes application monitoring and workload forecasting progressively challenging. Consequently, providing a cloud computing execution model that is not merely elastic but *autoscaling* accurately based on the workload is necessary [6]. Serverless computing emerges as such an execution model where providers are responsible for the high availability and scalability of each function to meet different workload needs. It is capable of launching thousands of parallel stateless functions hosted in light-weight containers rather than heavy-weight virtual machines (VMs) within a short period. In the serverless computing platform, an invocation for a newly deployed function is served by a newly created container (*cold-start*). Most providers also choose to reuse already activated containers to serve recurring invocations (*warm-start*) to reduce the overhead incurred by initializing a new container. Another significant advantage of the serverless platform is the "pay-as-you-go" pricing strategy. Providers usually charge for the resources consumption and execution duration of functions. In this paper, we focus on AWS Lambda and GCF because they are leading companies in this field. A summary of the pricing schemes of these two platforms is shown in Table 1. For AWS Lambda, the price per 100ms is proportional to the memory size, and the CPU quota is also allocated linearly in proportion to it. Although the memory size increment in the pricing list of AWS Lambda is 64MB, we find that the memory size can be actually set to any integer between 128MB to 3008MB via AWS CLI. For GCF, the price per 100ms is related to the memory size and CPU quota. The total price of both platforms for one function configured with m MB memory executing t seconds can be calculated by $C(m) * ceil(t/10) + I$, where $C(m)$ is the corresponding price per 100ms for memory size m MB, I is the price per invocation.

Given the advantages of serverless computing, efforts have been made to use it for video processing. Based on AWS Lambda, ExCamera [4] achieves a massively parallel, cloud-based video processing framework that can be used as the backend for interactive video processing applications. Sprocket [2] orchestrates serverless functions in video processing pipelines and exploits intra-video parallelism to achieve low latency. Both studies aim at employing massively parallel function instances for video processing acceleration. Recent measurement works [9, 13] on serverless computing platforms systematically explore the underlying resource allocation mechanisms at the platform side. In our measurement study, we focus on examining the impact of the user-controllable factors, including function resource configurations and function implementation schemes, on latency and cost reduction of standalone video processing functions from the developers' perspective.

3 METHODOLOGY

We invoke serverless functions directly via the provider's command line interface (AWS CLI or gcloud CLI) and pass in event data when necessary. Since our purpose is to study the implementation and configuration of standalone functions, rather than test the scalability of the serverless platform, we invoke functions serially instead of concurrently.

Measurement functions. We implement a measurement function template for video processing tasks and show its logic in Figure 1. Since serverless functions are designed to be stateless, shared remote storage is usually used to share data between different functions [8]. We adopt the cloud object storage such as Amazon S3 or

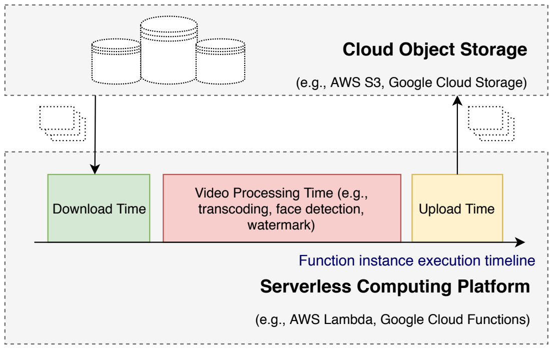

Cloud Object Storage
(e.g., AWS S3, Google Cloud Storage)
Download Time
Video Processing Time (e.g., transcoding, face detection, watermark)
Upload Time
Function instance execution timeline
Serverless Computing Platform
(e.g., AWS Lambda, Google Cloud Functions)

Figure 1: The measurement function template.

Google Cloud Storage (GCS) as the shared remote storage due to its widespread usage and reasonable I/O performance [7]. There are three subtasks in the template: downloading a video chunk from the cloud storage, performing the video processing task (e.g., transcoding or face detection) on the downloaded chunk, and uploading the processed video chunk back to the cloud storage. All measurement functions are implemented based on this template and only differ in the video processing section. In addition to executing subtasks, measurement functions are also responsible for subtask execution timing and collection of runtime information.

Runtime. We choose Python as our measurement function runtime since it is the most commonly used language runtime in AWS Lambda according to New Relic [6]. As GCF currently only supports Python3.7 runtime, unless otherwise noted, we configure Python3.7 as the function runtime to make fair cross-platform comparisons.

Workloads. Video processing includes not only simple video conversion tasks (e.g., compression, transcoding, editing) but also more complicated analytic tasks (e.g., scene recognition, face detection). We create a set of representative video processing functions to simulate different types of workloads. In particular, we report the results of the *transcoding* and *face detection* functions. Nonetheless, our observations apply to other video processing functions as well.

Transcoding functions use `ffmpeg` to transcode a video chunk downloaded from the cloud storage (S3 or GCS) to another bitrate and upload the newly generated video chunk to the cloud storage. *Face detection* functions are implemented to detect the presence of faces in a 6-second video chunk of the 720p Spider-Man trailer [7]. The frame sampling rate in our measurement is 8 frames/sec. The interaction between the cloud storage and the serverless function works in the same sequence as in the transcoding case. We compare three face detection methods. (1) Pre-trained MTCNN model to detect faces. MTCNN is a deep cascaded multi-task framework used for face detection and alignment [15]. We deployed an MTCNN model to the AWS Lambda platform with `tensorflow` and `opencv` libraries (Lambda-MTCNN). However, we failed to deploy it to GCF platform since the `tensorflow` library has not been well supported by python3.7. (2) Amazon Rekongnition Image API [8]. It is a deep learning powered image recognition service. We called its `DetectFaces` API within an AWS Lambda function to process video frames (Lambda-AAPI). (3) Google Cloud Vision API [9]. We called its `face detection` API within a GCF function to perform the face detection task (GCF-GAPI).

[6]https://blog.newrelic.com/product-news/aws-lambda-state-of-serverless/
[7]https://www.youtube.com/watch?v=OjgO_-Zk7bQ
[8]https://aws.amazon.com/rekognition/image-features/

Evaluation metric. From the developers' perspective, the most important concerns are the task execution time and the money to be paid. We thus adopt function execution duration and monetary cost as the performance metrics. The function execution duration is not the end-to-end time, but the time it takes for the function code to execute. This time is usually reported by the serverless platform and is rounded up to the nearest 100 ms as the billed time.

Underlying system information peeping. The measurement functions collect information about underlying infrastructures while performing video processing tasks. We adopt the method proposed by [13]: the measurement functions obtain VM identifications by checking the `proc` filesystem and distinguish between *cold-start* and *warm-start* instances by checking the existence of *InstanceID*.

4 MEASUREMENT RESULT

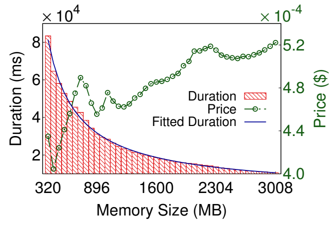
From the developers' perspective, there are mainly two ways to control function's performance: resources configuration and function implementation schemes. In this section, we examine the effects of these two controlling knobs on the functions' execution duration and monetary cost. Most of our measurements were completed between November 2018 and February 2019. Unless otherwise specified, we use the average of 10 *warm-start* invocations as the result of one measurement to mitigate randomness.

4.1 Function Configuration

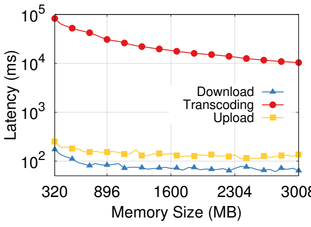
Figure 2 shows the results of the *transcoding* and *face detection* (Lambda-MTCNN) functions with different memory configurations deployed on AWS Lambda. We start to plot the figures from 320MB (448MB) because it is the minimum memory size needed for the *transcoding* (*face detection*) function to run successfully. As one can see from Figure 2a and Figure 2c, the function execution duration does not decrease proportionally with the increase of memory size. The relationship between execution duration and memory size can be fitted by a power-law distribution: $y = 15665000x^{-0.9119}$ for *transcoding*; $y = 14950000x^{-0.7171}$ for *face detection*. This indicates that there will not be a significant decrease in execution duration after increasing the memory size to a certain extent. Figure 2b and Figure 2d show the execution latencies of the three subtasks. The download and upload latencies are much smaller than the video processing latency, which follows the same pattern as the total execution duration. The download and upload latencies experience a slight decrease with smaller memory sizes and then fluctuate at a relatively stable value. This is because, for small memory sizes, the I/O performance is affected by the function memory (CPU) [13].

GCF uses a pricing strategy similar to AWS Lambda, charging for CPU and memory usage, but only provides five alternative memory and CPU configurations. We plot the execution duration and cost of *transcoding* functions deployed with these configurations and compare them with their AWS Lambda counterparts in Figure 3. For a fair cross-platform comparison, the results shown in the figure are the averages of 24 measurements over the day, and the error bars are used to represent the standard deviation. We start to plot

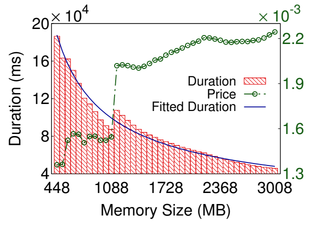
[9]https://cloud.google.com/vision/



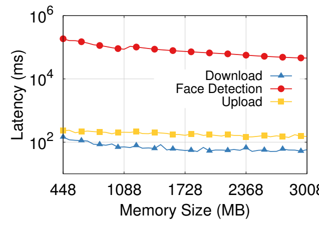
(a) Transcoding cost.



(b) Subtasks of transcoding.



(c) Face detection cost.



(d) Subtasks of face detection.

Figure 2: *Transcoding* and *face detection* performance with different memory sizes.

from 512MB because 128MB and 256MB are too small to run the function successfully. According to this figure, choosing a larger memory size is better for GCF since the execution duration can be improved a lot with a little extra money. Overall, AWS Lambda has more advantages regarding execution duration and monetary cost, but it experiences more intense performance fluctuations when the memory size is small.

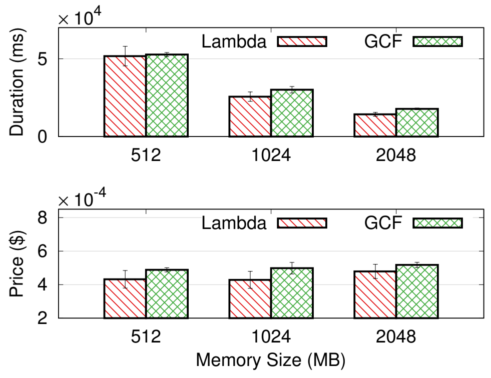


Figure 3: Comparison of different serverless platforms.

4.2 Function Implementation Scheme

Current serverless computing platforms have many limits on the computing and storage resources. For example, the deployment package size (250 MB, unzipped) and local disk size (512MB) of AWS Lambda may prohibit it from being used for running deep learning algorithms with large models. Furthermore, considering the convenience in management and development, combining serverless functions with external API services has become a trend. Since invoking external APIs requires extra latency and money for API accessing, we are interested to know the execution duration and cost comparison between running tasks locally or through external API invoking. We explore three *face detection* methods: Lambda-MTCNN, Lambda-AAPI and GCF-GAPI (see §3 for detail).

Figure 4a displays the execution duration of three *face detection* functions. The execution duration of API-based methods (Lambda-AAPI and GCF-GAPI) stays almost unchanged as the memory size rises. This is consistent with intuition since the most time-consuming work (face detection) is done by external APIs. However, Lambda-MTCNN, which consumes local resources to detect faces, shows an obvious latency decline and reaches a value very close to that of Lambda-AAPI when the memory size rises to 2048MB. Although unshown in the figure, when the memory size reaches 3008MB, the latency of Lambda-MTCNN (45815.4ms) is smaller than that of Lambda-AAPI (62375ms). It is worth noting that for the video we tested, Lambda-AAPI achieves the highest face detection recall (99.11%), followed by Lambda-MTCNN (90.27%). GCF-GAPI has the worst detection recall (71.93%) although it achieves the lowest latency. We hypothesize that Google Vision API is designed to sacrifice accuracy for efficiency. Overall, Figure 4a reveals that serverless-based *face detection* function implementation is comparable to API-based implementations in terms of execution duration.

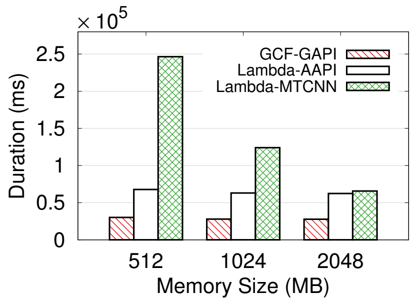
Figure 4b shows the monetary cost of different methods. For API-based methods, the total cost consists of the API invocation cost and the serverless function execution cost. We plot these two costs in stacked bars (API invocation costs are stacked above serverless function execution costs). Both Amazon Rekognition API and Google Cloud Vision API use tiered pricing schemes. Their amortized API invocation costs are illustrated by the heights of the bars. The minimum and maximum API invocation costs under the tiered pricing schemes are illustrated by error bars. We observe that API invocation costs contribute more than 90% of the total costs. This indicates that calling external API services is much more expensive than executing serverless functions. This observation can be further supported by the cost of Lambda-MTCNN (labeled as MTCNN in the figure), which only contains serverless function execution cost and is significantly cheaper than API-based methods. Overall, Figure 4b reveals that serverless-based implementation of the *face detection* function is more effective than API-based implementations in terms of monetary cost.

We plot the processing latency of each frame in the test video chunk in Figure 4c. We observe that the processing latency is related to the content of the frame. For example, the processing latency of the first frame is longer than the tenth frame in all methods. This can be explained by the fact that there are multiple human faces in the first frame and no face in the tenth frame. We also find that increasing the memory size configured for the Lambda-MTCNN function narrows the gap between the processing latencies of frames. This suggests that we can achieve a stable latency by providing more resources for the serverless-based implementation.

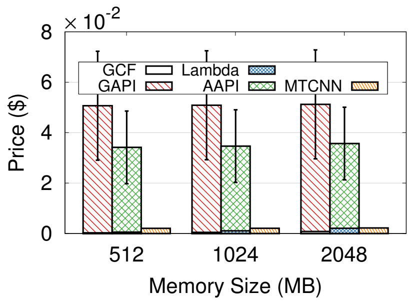
5 DISCUSSION

5.1 What We Learn?

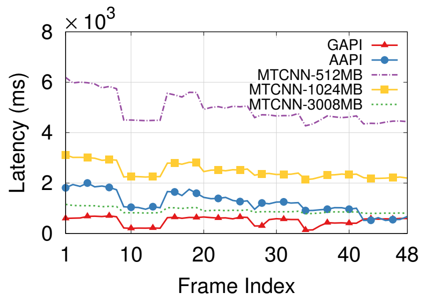
Memory configuration. The memory configuration for serverless functions is non-trivial, especially for a serverless platform



(a) ***Face detection* execution duration.**



(b) Monetary cost.



(c) Video frame processing latency.

Figure 4: Comparison of different implementations of the *face detection* function.

like AWS Lambda, which has 2881 configuration options. Although the relationship between execution duration and memory size can be fitted by a power-law distribution for AWS Lambda functions, different workloads have different fitting parameters (e.g., *transcoding* and *face detection*). This indicates that we need to profile the workload dynamically first and then choose an appropriate memory size based on the final optimization goal (latency or cost or both). For example, in order to minimize the execution duration of a given budget, users can first obtain the corresponding relationship between price and memory size by profiling the workload and then choose the maximum memory size that does not exceed the memory size corresponding to the budget.

Implementation scheme. Integrating serverless functions with other cloud services (such as Amazon Rekognition) can break up the limits on serverless functions, enabling them to handle more complex video processing tasks. Despite this, running pre-trained models in serverless functions to perform video analytic tasks can obtain advantages of latency and monetary cost.

Platform Selection. So far, for all video processing tasks, AWS Lambda functions outperform GCF functions under the same configuration and implementation scheme in terms of execution duration and monetary cost. Does this mean that AWS Lambda is better than GCF in all respects? We will answer this question in §5.2.

5.2 Insights into System Factors

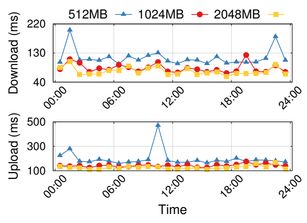
During our experiments, we find that if we call the same function deployed with AWS Lambda at different times of the day, we can observe apparent differences in execution duration. To further confirm that this is an accidental or a common phenomenon, we conduct measurements once an hour to see how the performance of the *transcoding* function deployed with the serverless platforms changes in one day and show the results in Figure 5.

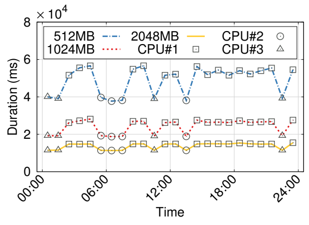
Figure 5a shows the I/O time variation of AWS Lambda. Increasing the memory size from 1024MB to 2048MB does not improve the download and upload latencies much, which indicates that for larger memory sizes, CPU power is not the bottleneck for I/O tasks. We plot the variation of total duration in Figure 5b (video processing time follows the same pattern). We observe that as the memory size decreases, the execution duration changes more and more dramatically. For example, the longest execution duration of 512MB (56640ms) is 1.5 times as much as the shortest execution duration (37735ms), and the increment is even larger than the average execution duration of 2048MB (13754ms). Considering that we sequentially invoke the function and AWS Lambda assigns dedicated VMs for one tenant [13], it cannot be caused by the resources competition with other co-located function instances. We check the information of VMs where function instances are hosted and find that the obvious changes in performance over time mainly caused by the underlying heterogeneous infrastructure. There are three types of CPUs in our experiments: CPU#1 (E5-2666 v3 @ 2.90GHz), CPU#2 (E5-2676 v3 @ 2.40GHz) and CPU#3 (E5-2686 v4 @ 2.30GHz). We plot the CPU types of VMs hosting the corresponding function instances in Figure 5b. As can be observed, different CPU types correspond to different performance levels. We also find that the abnormal rise in Figure 2c is caused by underlying infrastructure. When the memory size is less than 1152MB, the CPU type of VMs hosting Lambda-MTCNN function instances is CPU#2, after that the CPU type of VMs is CPU#1. Therefore, we hypothesize that if all Lambda-MTCNN function instances are assigned the same type of VM, the function execution duration in Figure 2c will continue to decrease as the memory size increases, like that of the *transcoding* function (shown in Figure 2a). Overall, it is fair to say that the influence of the heterogeneous underlying infrastructures on performance cannot be ignored.

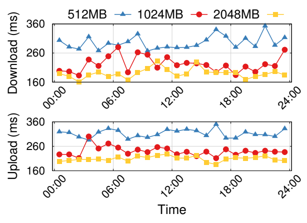
Compared with AWS Lambda, the performance of functions deployed with GCF is more stable, especially for small memory sizes. According to the CPU information we can get, we do not find obvious performance differences between function instances hosted on different VMs [10]. We also find that GCF tends to launch new function instances aggressively for functions with small memory sizes rather than reuse existing ones as stated in [13].

To further investigate the stability changes in a more coarse time granularity, we plot the changes in execution duration of the *transcoding* task within one week in Figure 6. For AWS Lambda, the performance shows more drastic changes as the memory size decreases. Although unshown in the figure, the execution duration of 3008MB remains basically unchanged within one week. The changes do not show a daily periodic pattern but are closely related to the heterogeneous underlying infrastructures. This convinces

[10]We cannot get *CPU Model names* but can get the values of *CPU family* and *Model*. We thus suppose VMs with different values of *CPU family* and *Model* must have different CPUs.


(a) AWS Lambda I/O time.


(b) AWS Lambda duration.


(c) GCF I/O time.

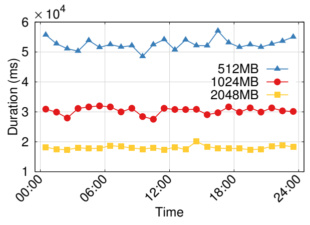

(d) GCF duration.

Figure 5: Changes in execution time for the *transcoding* function in one day (30/01/2019).

us that the scheduling and allocation of VMs are random and independent of time. In contrast, GCF exhibits a more stable execution duration regardless of the memory size.

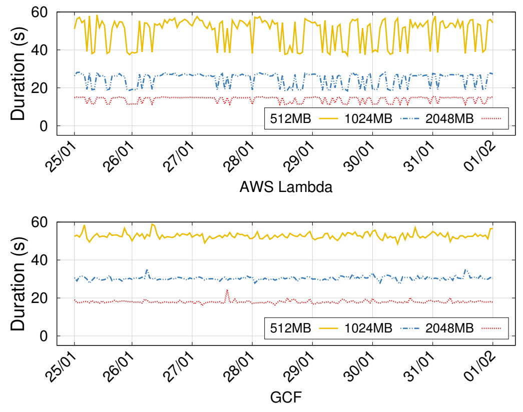


Figure 6: *Transcoding* function execution duration changes within one week (25/01/2019-31/01/2019) on different serverless platforms.

6 CONCLUSION AND FUTURE DIRECTIONS

In this paper, we examined the influence of configuration and implementation schemes on the performance of typical video processing serverless functions from the developers' perspective. Our work provided guidelines for future *function-level* optimization for developers and complements the existing measurement studies on both video processing and serverless computing. Powering video processing tasks with the emerging serverless computing architecture is still in its infancy. There are a few exciting directions to follow:

Serverless cost-efficiency optimization. Our measurement reveals that the memory configuration for cost-efficient serverless functions is non-trivial. The best memory configuration is influenced by the task type or even the video content. More work is needed to design an efficient and adaptive system to find the best configuration for serverless functions in video processing pipelines.

Serverless deep learning. Although deep learning has achieved great success in video analytics, deploying large models to the current serverless platforms still faces many challenges (e.g., small storage, no support for GPU). To achieve the potential of serverless computing, running a set of concurrent functions that each runs a small and dedicated model in the serverless platform can be a viable alternative for running a large model on GPU-based infrastructures.

Serverless edge computing. An undiscussed aspect of serverless computing in this paper is the combination with edge infrastructures. While executing function codes closer to users facilitates customized video processing and improves the response speed, edge resource is usually limited compared to the cloud. The trade-off between performance and cost in serverless edge computing will be an important issue in future research.

ACKNOWLEDGMENTS

This work is supported by a Canada Technology Demonstration Program (TDP) Grant and a Canada NSERC Discovery Grant. The work of C. Zhang is supported by the Fundamental Research Funds for the Central Universities (WK2150110015).

REFERENCES

[1] I. E. Akkus, R. Chen, I. Rimac, M. Stein, K. Satzke, A. Beck, P. Aditya, and V. Hilt. 2018. SAND: Towards High-Performance Serverless Computing. In *USENIX ATC*.

[2] L. Ao, L. Izhikevich, G. M. Voelker, and G. Porter. 2018. Sprocket: A Serverless Video Processing Framework. In *ACM SoCC*.

[3] Cisco. 2018. Cisco Visual Networking Index: Forecast and Trends, 2017-2022. https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-741490.html

[4] S. Fouladi, R. S. Wahby, B. Shacklett, K. Balasubramaniam, W. Zeng, R. Bhalerao, A. Sivaraman, G. Porter, and K. Winstein. 2017. Encoding, Fast and Slow: Low-Latency Video Processing Using Thousands of Tiny Threads. In *USENIX NSDI*.

[5] G. Gao and Y. Wen. 2016. Morph: A fast and scalable cloud transcoding system. In *ACM MM*.

[6] J. M. Hellerstein, J. Faleiro, J. E. Gonzalez, J. Schleier-Smith, V. Sreekanti, A. Tumanov, and C. Wu. 2018. Serverless Computing: One Step Forward, Two Steps Back. *arXiv preprint arXiv:1812.03651* (2018).

[7] E. Jonas, Q. Pu, S. Venkataraman, I. Stoica, and B. Recht. 2017. Occupy the cloud: Distributed computing for the 99%. In *ACM SoCC*.

[8] A. Klimovic, Y. Wang, C. Kozyrakis, P. Stuedi, J. Pfefferle, and A. Trivedi. 2018. Understanding ephemeral storage for serverless analytics. In *USENIX ATC*.

[9] W. Lloyd, S. Ramesh, S. Chinthalapati, L. Ly, and S. Pallickara. 2018. Serverless computing: An investigation of factors influencing microservice performance. In *IEEE IC2E*.

[10] H. Ma, B. Seo, and R. Zimmermann. 2014. Dynamic scheduling on video transcoding for MPEG DASH in the cloud environment. In *ACM MMSys*.

[11] R. Olga, D. Jia, S. Hao, K. Jonathan, S. Sanjeev, M. Sean, H. Zhiheng, K. Andrej, K. Aditya, B. Michael, C. B. Alexander, and F. Li. 2015. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision* 115, 3 (2015), 211–252.

[12] K. Sembiring and A. Beyer. 2013. Dynamic resource allocation for cloud-based media processing. In *ACM NOSSDAV*.

[13] L. Wang, M. Li, Y. Zhang, T. Ristenpart, and M. Swift. 2018. Peeking behind the curtains of serverless platforms. In *USENIX ATC*.

[14] H. Zhang, G. Ananthanarayanan, P. Bodik, M. Philipose, P. Bahl, and M. J. Freedman. 2017. Live Video Analytics at Scale with Approximation and Delay-Tolerance. In *USENIX NSDI*.

[15] K. Zhang, Z. Zhang, Z. Li, and Y. Qiao. 2016. Joint face detection and alignment using multitask cascaded convolutional networks. *IEEE Signal Processing Letters* 23, 10 (2016), 1499–1503.

[16] X. Zhu, Y. Wang, J. Dai, L. Yuan, and Y. Wei. 2017. Flow-guided feature aggregation for video object detection. In *IEEE ICCV*.