Migrating Large Deep Learning Models to Serverless Architecture

Dheeraj Chahal
TCS Research
Mumbai, India
d.chahal@tcs.com

Ravi Ojha
TCS Research
Mumbai, India
ravi.ojha@tcs.com

Manju Ramesh
TCS Research
Mumbai, India
manju.ramesh1@tcs.com

Rekha Singhal
TCS Research
Mumbai, India
rekha.singhal@tcs.com

***Abstract*—Serverless computing platform is emerging as a solution for event-driven artificial intelligence applications. Function-as-a-Service (FaaS) using serverless computing paradigm provides high performance and low cost solutions for deploying such applications on cloud while minimizing the operational logic. Using FaaS for efficient deployment of complex applications, such as natural language processing (NLP) and image processing, containing large deep learning models will be an advantage. However, constrained resources and stateless nature of FaaS offers numerous challenges while deploying such applications.**
In this work, we discuss the methodological suggestions and their implementation for deploying pre-trained large size machine learning and deep learning models on FaaS. We also evaluate the performance and deployment cost of an enterprise application, consisting of suite of deep vision preprocessing algorithms and models, on VM and FaaS platform. Our evaluation shows that migration from monolithic to FaaS platform significantly improves the performance of the application at a reduced cost.

***Index Terms*—serverless, FaaS, cloud, AI**

I. Introduction

Contrary to training a deep neural network model, which is time consuming and may run from few hours to days, model inference is performed in real-time with stringent throughput and latency requirements as defined in the Service Level Agreements (SLAs). These requirements become more stringent and challenging for the applications using series of models for inferencing such as invoice processing for enterprise process automation framework. Deploying deep learning algorithms and models in production for inference requires high performance machines to achieve high throughput and low latency [1]. However, maintaining a high performance cluster for inference incurs high cost and sometimes economically non-viable. Serverless computing is emerging as a preferred choice for low cost and high performance deployment of machine learning and deep learning applications [2].

The price model of serverless platform is based on the execution time instead of resource allocation time.The pay-per-request model of serverless computing provides cost effective application deployment. Additionally, it abstracts away all challenges and efforts required in managing the infrastructure and allowing developers to focus on application code, logic and hence making serverless computing a preferred choice for enterprises. The availability of high-end elastic infrastructure on cloud provides ease of scalability in a serverless deployment. The high availability and auto-scaling of FaaS function provides smooth performance for large workloads.

Despite several benefits and popularity, serverless architecture can not be used for all products and applications. Cloud vendors provide detailed guidelines, best practices and cost models for using FaaS platform [3]–[6]. Unfortunately, the methodology and techniques provided by cloud vendors are given for general workloads and may not be cost effective and efficient for all kinds of products and workloads. The onus is on developers and architects to use the FaaS resources optimally based on their application specific requirements and workloads.

Deploying machine learning and neural network based applications for training models or inference on serverless architectures offers numerous challenges. Serverless architecture is stateless and FaaS platform offered by popular vendors such as Amazon Web Service (AWS) Lambda [1], Microsoft Azure function [2], IBM OpenWhisk [3], Google cloud function [4] etc. provide limited local storage and memory for frameworks, packages, libraries and models. Application APIs using large frameworks, libraries and models can not be accommodated in this limited space. Moreover, lifetime of FaaS instance is limited before it stops abruptly. Restarting FaaS instance for a new request results in high latency due to cold start and hence risk violating SLAs. Deploying such applications using a serverless architecture is daunting but important.

One challenging and motivating use case for a serverless architecture is compute intensive and data intensive stateful applications that use large and complex deep learning models for information extraction from images. These applications require uploading documents and images that trigger an event. Images are sliced and information is extracted from different entities such as text block, tables, handwritten or printed text, boxes and lines. This information further stored in high level relational schema. Serverless computing is suitable for this kind of task for its pay-per-request and run on-demand properties.

In this work, we propose a methodology to deploy relatively large models for inference and also study the cost effectiveness

[1]https://aws.amazon.com/lambda/
[2]https://azure.microsoft.com/en-in/services/functions/
[3]https://www.ibm.com/in-en/cloud/functions
[4]https://cloud.google.com/functions

978-1-7281-7735-9/20/$31.00 ©2020 IEEE
DOI 10.1109/ISSREW51248.2020.00047

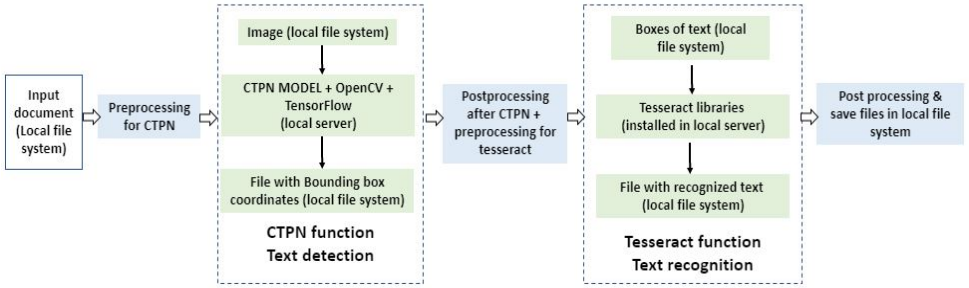

Fig. 1. Abstract view of On-premise deployment of Deep Reader application OCR API

to inference such models in an application. We look at various optimizations that we can apply while migrating such applications to a serverless architecture. More specifically, our main contributions are as follows:

- Proposing methodology and optimizations to migrate a vision algorithm based application containing a suite of models from on-premise deployment to a serverless architecture on cloud.
- We study the cost and performance of serverless architecture against on-premise deployment and virtual machine on cloud.
- We study the effect of using multiple cloud services such as VM instances and Elastic File System (EFS) on the performance of the FaaS for implementing large models.

In this work, we also highlight the advantages and disadvantages of FaaS. We evaluate our work on most commonly used serverless platform AWS Lambda [7] but the approach is generic which can be applied to similar cloud services offered by other vendors.

The rest of the paper is structured as follows. In section II we discuss state-of-the-art in serverless architecture domain. In-house application used for this work is discussed in section III. Our methodology and architecture to deploy this application using multiple cloud services on serverless architecture is discussed in section IV. The experimental setup and evaluation is discussed in section V. We discuss our observation and experience in section VI followed by conclusion in section VII.

II. Related work

In one of the works, SPEC RG cloud group discussed performance related challenges that arise in FaaS and also outlined some of the research areas that require immediate attention of the research community [8]. In [2], detailed study on cost reduction by migrating application to FaaS is discussed.

Dakkak et al. developed a tool called TrIMS [9] to address drawbacks of FaaS. The tool provides a persistent model store for multiple device types, set of application APIs and container technologies for integration with FaaS and DL frameworks. Zhang et al. also introduced a tool called MArk [10] for using a serverless architecture for ML inference. It exploits the stateless nature of inference serving and uses serverless instances for bursty traffic that is hard to predict. Perron et al. developed a query engine called Starling [11] that is built on cloud function service to provides interactive query latency at low cost by overcoming the FaaS challenges .

Lee et al. [12] in their study explored performance of partitioned tasks in a serverless computing environment. In this work they demonstrated comparison of cost and resource utilization of serverless computing and virtual machines.

In a similar work [13], an approach for migrating document processing FinTech application from monolithic architecture to the serverless implementation is presented. Another study [14], evaluates the suitability of a serverless computing platform for the inferencing of large neural network models. However, both these techniques are valid for small models that can be loaded from the FaaS storage into memory.

Although serverless computing is not suitable for training process in an AI application due to time, compute and memory intensive nature of the job but has proved to be useful for inference and prediction [15] [16]. Recent work [17] is about slimming AI libraries and dynamically loading models in to temporary runtime memory. Our work extends this approach further for large models that can't fit in FaaS local storage. Additionally, we study and compare the cost and performance of FaaS based deployment against VM instance based deployment on cloud. We believe that our work might influence the trend of cost effective large AI model deployment on cloud.

III. Our Application and use case

In this section we discuss the architecture of our in-house application called Deep Reader [18] that we used for our experiments. Deep Reader is an image and PDF document processing application used to extract and identify text, relationships among the different text blocks and then create a meta relational model. It has multiple state-of-the-art vision algorithms and large deep learning model embedded inside.

The major APIs used in Deep Reader includes Optical Character Recognition (OCR) API to extract words, sentences and text blocks along with coordinates, type extractor API to identify the simple and semantic data type of the words, relationship extractor API to identify the spatial relationship

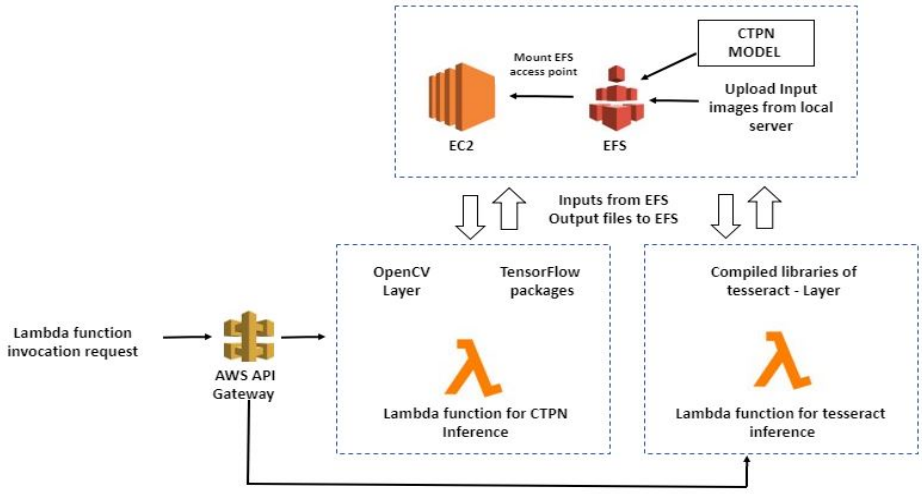

Fig. 2. Architecture for FaaS deployment of Deep Reader OCR API

between the different entities in a PDF document or image and table Detection API for identifying the tables in the image, and get the data in the tables to convert to CSV. As shown in Figure 1, the OCR API is used for image detection and recognition. The CTPN model [19] and Tesseract [20] OCR engine in the OCR API are used for the text detection and recognition. Tesseract OCR engine is specifically used for text recognition. In Deep Reader, batch Tesseract mode is used where multiple text boxes are processed together in one invocation. To implement Tesseract as a Lambda function, the compiled Tesseract binaries and configuration files are added as a layer. Then this layer is used in the Lambda function.

In this work our focus is on accelerating the Deep Reader vision APIs using serverless architecture. Among all these APIs, the OCR API is the most time consuming and takes more than 70% of the total processing time of an image and contains large CTPN model and Tesseract engine. Also, OCR API is used for processing almost all types of documents while the other APIs are used infrequently depending upon the input document or image. Our objective is to deploy and accelerate the OCR API successfully.

IV. Deployment Methodology

In this section we discuss our technique for deploying the on-premise Deep Reader application to AWS Lambda with focus on complex OCR API. The architecture for deploying Deep Reader using AWS Lambda serverless platform is as shown in Figure 2. The required packages including TensorFlow [5], OpenCV [6] and Deep Reader code is zipped locally and uploaded on AWS Lambda. In order to accommodate the packages in the storage space limit of 250 MB available on Lambda, we trimmed both TensorFlow and OpenCV by removing all unnecessary libraries that are not required by Deep Reader but are part of TensorFlow and OpenCV. Also, input images that need to be processed are preloaded in the AWS EC2 instance. All pre-processing required for the OCR API and post-processing is done in one FaaS function. The output generated at various stages in OCR API is stored in EFS that is used by other APIs or lambda functions.

The size of CTPN model we used is 275 MB and hence a challenge for loading model along with other packages in '/tmp' of Lambda which has a limit of 250 MB. For deploying CTPN model we created EFS access points on AWS EC2 instance. We used EFS in general purpose mode as it is useful for Lambda workloads and provides high performance at low cost [21]. Mounting EFS system adds minimum latency as it is mounted when Lambda execution environment is created. We have observed a latency of few milliseconds for the first invocation of the function and subsequent invocations have no latency as the Lambda function is already warm. In addition to the availability of large persistent storage, another advantage of using EFS is its availability across multiple Lambda functions. Hence, the output written by one function can be accessed by all other functions.

We applied few optimizations to address serverless architecture constraints

1. Trimming TensorFlow framework and libraries One of the constraints in serverless architecture is the total physical space available for packages and libraries that are required during a function execution. The AI and deep learning models, frameworks or libraries are complex and too large to fit in this space. One solution that we adopted was to eliminate the python libraries that were not required by our APIs. In addition to that, TensorFlow and OpenCV frameworks required by our APIs contain libraries that are not used and hence eliminated. In our implementation, we are able to see more than 50% reduction in the size to

[5]https://www.tensorflow.org/
[6]https://opencv.org/

117MB as compared to the original size of more than 250MB.

2. Loading input data The cost of using a Lambda function is proportional to the total memory reserved and the processing time. Reserving a high memory in Lambda function for uploading data directly from the client application for processing would result in higher costs. There exists multiple ways to load input data into the Lambda function memory for processing using storage service on cloud. In one of the previous works on economics and architecture of serverless computing, AWS S3 storage service was used for high performance and low cost [2]. Recently introduced EFS [7] support for AWS Lambda, uses the network file system version 4 protocol and provides higher IOPS and lower latency as compared to S3. We preload all images from the client application that were needed to be processed into EFS on EC2 instance. An event is triggered to load large models and input data, used in various APIs from EFS to Lambda function memory. The intermediate data generated by any of the APIs in the function is stored in EFS and later loaded and consumed by other APIs in another functions.

The following steps are used to deploy application APIs on AWS Lambda:

- All packages and the application code is zipped in a folder on a local machine.
- Create server.yml file on local machine and list libraries or packages in the yml file that are not required.
- Load required models and input data (e.g. images for batch processes) on EC2 instance while zipped folder in Lambda storage by invoking the yml file. Unzip the folder in '/tmp' of the Lambda storage.
- On the first invocation of the Lambda instance, load models in Lambda RAM from EC2 instance using EFS access point.

V. Experiments

As discussed earlier, one of the objectives of this study is to evaluate the performance and cost of a serverless platform and compare it with an on-premise and cloud VM deployment. For this study, we used AWS EFS, AWS Lambda for Function-as-a-Service and AWS EC2 instances. We opted for inexpensive EC2 instance, as it is required only for creating EFS access point to store large models.

A. Comparison of local vs EC2 VM vs Lambda response time and throughput

In this experiment, we compare the response time of OCR API on on-premise bare metal machines, virtual machines on cloud (EC2 instances) and AWS Lambda. We use *md5.12xlarge* EC2 instance with 24 cores, while on on-premise, we use 24 core machine with the hyper threading on. We increase the Lambda memory as the workload increases and maximum available memory of 3GB is allocated for a batch of 1000 images. Also, we record response time of

[7]https://docs.aws.amazon.com/lambda/latest/dg/services-efs.html

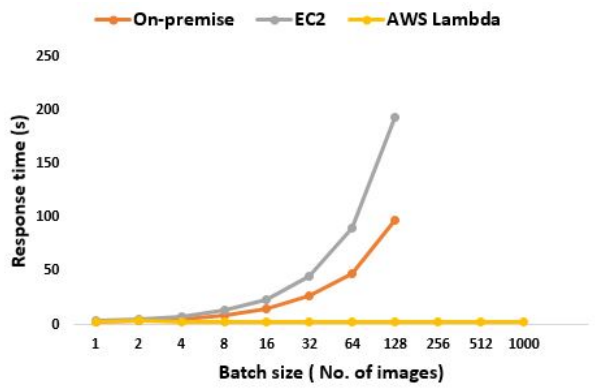

Fig. 3. On-premise, EC2 virtual machine, AWS Lambda response time comparison with increasing workload

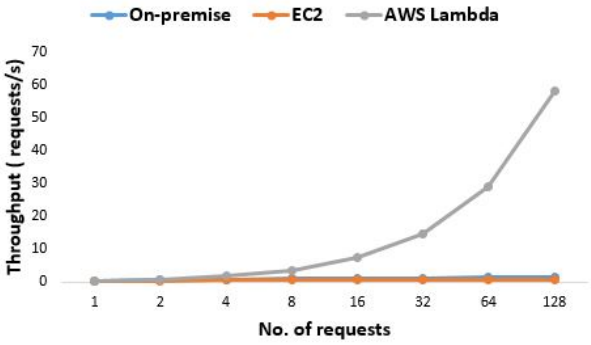

Fig. 4. On-premise, EC2 virtual machine, AWS Lambda throughput comparison with increasing workload

Lambda when the function is warm. We observe the response time for concurrent execution of the batch. As shown in Figure 3, we observe the Lambda response time slightly lower than EC2 VM and on-premise deployment for smaller workloads (up to a batch of 16 images). As we increase the workload further, we observe a rise in the response time for on-premise and EC2 VM while the Lambda response time is constant. This is due to the auto scaling available in Lambda. Lambda can scale up or down instances without the knowledge of the developers. Figure 4 shows the change in the throughput for on-premise, EC2 and Lambda. While throughput of EC2 and on-premise deployment does not change much, but increases with workload for Lambda due to auto scaling.

B. Cold start and warm start with concurrency

In these set of experiments, we study the effect of cold start on the response time with an increasing concurrency. The effect of cold start in a serverless architecture on latency is well known and documented. In addition to this, we study the effect of cold start and warm start on latency in conjunction with an increasing concurrency. In our experiments we observe the latency in processing the batches of different sizes at a time interval of 15 minutes resulting in cold start for each invocation of Lambda function, and also at the interval of

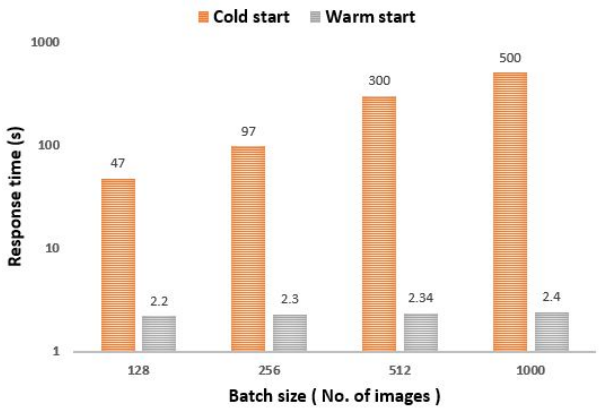

Fig. 5. Cold and warm start effect with concurrency

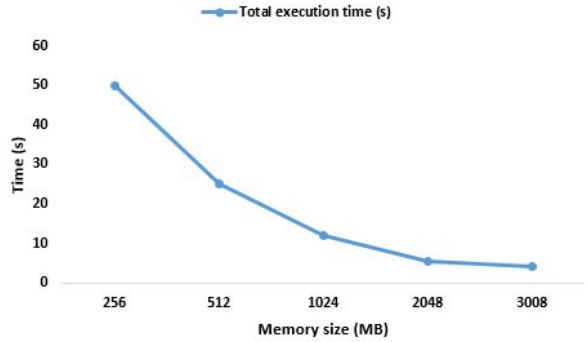

Fig. 6. Effect of memory size on execution time

2 minutes in which case Lambda function is warm for each invocation for a batch. As shown in Figure 5, we observe that for each batch of input images to process, cold start results in large response time as compared to the warm start for the same set of images. Another important conclusion that we can make from this experiment is that cold start time is also dependent on the concurrency or batch size. As the batch size increases, the time to cold start the function also increases and the gap between cold start and warm start time widens. This is because execution environment creation time increases as size of data being loaded in the RAM increases with increase in batch size.

TABLE I
AWS LAMBDA MEMORY COST

Memory (MB)	Price per 100ms
128	$0.0000002083
512	$0.0000008333
1024	$0.0000016667
1536	$0.0000025000
2048	$0.0000033333
3008	$0.0000048958

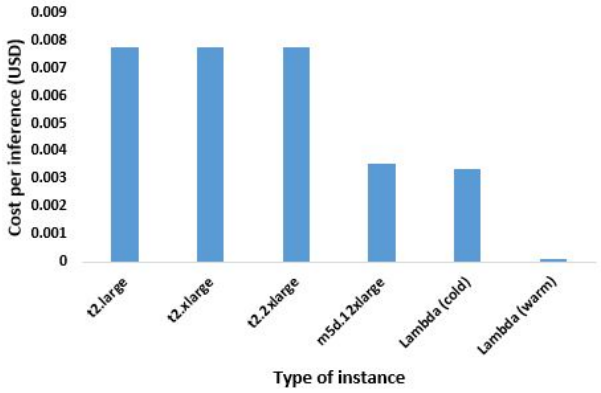

Fig. 7. EC2 vs Lambda cost analysis

C. Effect of Lambda memory size on response time

We observe that memory allocated to Lambda can affect the response time. In our experiment, we increase the Lambda memory in steps and recorded total execution time of the batch of 64 images. Figure 6 shows that as we increase the memory allocated to Lambda, total execution time of the batch decreases. This is due to the fact that compute resources are allocated as per the memory allocation i.e. 128 MB of Lambda function has half the CPU power as compared to 256 MB. The maximum CPU power is available at the maximum memory allocation but the cost increases proportionally as shown in Table I.

D. Cost comparison of VM on cloud and Lambda

One of the major advantages of using serverless architecture is the cost reduction due to pay-per-request model. We did a cost comparison of four different types of AWS EC2 instances and AWS Lambda. Figure 7 shows the cost per inference for using four different types of instances namely *t2.large, t2.xlarge, t2.2xlarge, md5.12xlarge* containing 2, 4, 8, 24 cores respectively and Lambda with 3008 MB memory. For calculating the cost per inference of VM instances, we assume that instances are up for approximately 600 seconds to run our workload of 1000 images. The VM instance up time includes time to boot up instances, authorization, setting up code base with all the necessary files being moved from local to EC2 and finally bringing down the instance after post processing of logs. Also, we calculated Lambda cost for warm start and cold start that includes time to create execution environment. Lambda cost also includes the EFS expenses. We see that the cost per inference in AWS Lambda for cold start is comparable to largest instance we chose but for warm start it is far less than any VM machines while delivering higher throughput (Figure 4) and lower response time (Figure 3) when compared with the same largest instance.

VI. DISCUSSION

Our approach of using EFS with Lambda for deploying application containing large models proved to be promising.

Although, there is a marginal increase in the cost per inference for using EFS along with Lambda but still cheaper than a VM on cloud. Also, the performance metrics such as throughput and response time achieved using EFS with Lambda are better than using one large VM. This is primarily due to auto-scaling in FaaS. However, a cold start results in a very high response time. Considering this cold start issue, FaaS may not be a good option when the requests in real time are sparse resulting in function cold start each time.

Estimating the cost of VM usage is easy as it is proportional to the total time for which a particular VM is acquired. Contrary to this, cost prediction for a serverless architecture usage is tedious as it depends on usage of resources which changes with varying workloads.

From these experiments, we can conclude that the deployment methodology and architecture we used for our application using a serverless platform improves the performance significantly with lower costs. Also, this approach can be used for deploying large deep learning models for inference using EFS.

VII. Conclusion and future direction

Numerous tools and methodologies have been developed to deploy applications from different domains using serverless architecture. However, there are still many open challenges that needs to be addressed such as deploying large packages, libraries, models for inference etc. using serverless architecture platforms. In this work, we have proposed methodology that can be used to deploy deep learning applications containing large models for inference. We presented performance comparison of FaaS with on-premise deployment of one such application. Also, we presented cost analysis using AWS Lambda service for batch inference in large deep neural network model based application.

Our on-going work is focused on extending our approach and evaluation to different deep learning frameworks such as MXNet and PyTorch and other different types of convolution and recurrent neural network models. We are also evaluating serverless platform offering from multiple vendors. Our future research is focused on building a tool for optimal and cost effective use of FaaS resources.

References

[1] D. Chahal, R. Ojha, S. R. Choudhury, and M. Nambiar, "Migrating a recommendation system to cloud using ml workflow," in *Companion of the ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 1–4. [Online]. Available: https://doi.org/10.1145/3375555.3384423

[2] G. Adzic and R. Chatley, "Serverless computing: Economic and architectural impact," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2017. New York, NY, USA: Association for Computing Machinery, 2017, p. 884–889. [Online]. Available: https://doi.org/10.1145/3106237.3117767

[3] (2020) The AWS Lambda. [Online]. Available: https://aws.amazon.com/lambda/

[4] (2020) The IBM Functions. [Online]. Available: https://cloud.ibm.com/functions/

[5] (2020) The Azure serverless. [Online]. Available: https://azure.microsoft.com/en-us/solutions/serverless/

[6] (2020) The Google cloud. [Online]. Available: https://cloud.google.com/serverless

[7] J. Scheuner and P. Leitner, "Function-as-a-service performance evaluation: A multivocal literature review," 2020.

[8] E. van Eyk, A. Iosup, C. L. Abad, J. Grohmann, and S. Eismann, "A spec rg cloud group's vision on the performance challenges of faas cloud architectures," in *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 21–24. [Online]. Available: https://doi.org/10.1145/3185768.3186308

[9] A. Dakkak, C. Li, S. G. D. Gonzalo, J. Xiong, and W. mei W. Hwu, "Trims: Transparent and isolated model sharing for low latency deep learninginference in function as a service environments." *arXiv: Distributed, Parallel, and Cluster Computing*, 2018.

[10] C. Zhang, M. Yu, W. Wang, and F. Yan, "Mark: Exploiting cloud services for cost-effective, slo-aware machine learning inference serving," in *Proceedings of the 2019 USENIX Conference on Usenix Annual Technical Conference*, ser. USENIX ATC '19. USA: USENIX Association, 2019, p. 1049–1062.

[11] M. Perron, R. Castro Fernandez, D. DeWitt, and S. Madden, "Starling: A scalable query engine on cloud functions," in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 131–141. [Online]. Available: https://doi.org/10.1145/3318464.3380609

[12] H. Lee, K. Satyam, and G. Fox, "Evaluation of production serverless computing environments," in *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, 2018, pp. 442–450.

[13] A. Goli, O. Hajihassani, H. Khazaei, O. Ardakanian, M. Rashidi, and T. Dauphinee, "Migrating from monolithic to serverless: A fintech case study," in *Companion of the ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 20–25. [Online]. Available: https://doi.org/10.1145/3375555.3384380

[14] V. Ishakian, V. Muthusamy, and A. Slominski, "Serving deep learning models in a serverless platform," *2018 IEEE International Conference on Cloud Engineering (IC2E)*, pp. 257–262, 2018.

[15] I. Baldini, P. C. Castro, K. S.-P. Chang, P. Cheng, S. J. Fink, V. Ishakian, N. Mitchell, V. Muthusamy, R. M. Rabbah, A. Slominski, and P. Suter, "Serverless computing: Current trends and open problems," in *Research Advances in Cloud Computing*, 2017.

[16] I. Baldini, P. C. Castro, K. S. Chang, P. Cheng, S. J. Fink, V. Ishakian, N. Mitchell, V. Muthusamy, R. M. Rabbah, A. Slominski, and P. Suter, "Serverless computing: Current trends and open problems," *CoRR*, vol. abs/1706.03178, 2017. [Online]. Available: http://arxiv.org/abs/1706.03178

[17] A. Christidis, S. Moschoyiannis, C. Hsu, and R. Davies, "Enabling serverless deployment of large-scale ai workloads," *IEEE Access*, vol. 8, pp. 70 150–70 161, 2020.

[18] D. Vishwanath, R. Rahul, G. Sehgal, S. , A. Chowdhury, M. Sharma, L. Vig, G. Shroff, and A. Srinivasan, *Deep Reader: Information Extraction from Document Images via Relation Extraction and Natural Language*, 06 2019, pp. 186–201.

[19] Z. Tian, W. Huang, H. Tong, P. He, and Y. Qiao, "Detecting text in natural image with connectionist text proposal network," vol. 9912, 10 2016, pp. 56–72.

[20] R. Smith, "An overview of the tesseract ocr engine," in *Ninth international conference on document analysis and recognition (ICDAR 2007)*, vol. 2. IEEE, 2007, pp. 629–633.

[21] Aws efs. [Online]. Available: https://aws.amazon.com/blogs/compute/using-amazon-efs-for-aws-lambda-in-your-serverless-applications/