Serverless Inference of Sequential Neural Networks

Mohan Rao Divate Kodandarama [1]

Abstract

Serverless Functions / Cloud Functions provide several key benefits such as 1. Functions can be triggered in response to events, i.e. it adopts an event-based programming model. 2. Provisioning and scalability issues are not the concern of the application. 3. The serverless platform adopts the pay-per-use model, hence prevent wastage of application resources that happens either by over-provisioning and hurting budget or under-provisioning and hurting throughput and/or latency.

Sequential Neural Networks are widely used for modelling temporal sequences. They are used in several applications ranging from text classification to audio super-resolution. In this paper, we study the viability of building and inference system for sequential neural network models on serverless computing platform. We identify sequential neural network models suitable for deployment on serverless computing platform, propose a serverless system architecture. Further, we evaluate it with respect to traditional GPU based deployment in terms of throughput and cost.

1. Introduction

Serverless computing is a computational model in which the cloud provider runs the server, and dynamically manages the allocation of machine resources. It comes with several advantages few of which are listed below -

1. No server management is necessary.
2. Pay-per-use model.
3. Serverless architectures are inherently scalable.

Several cloud vendor offer serverless computing platform which is also referred to as function as a service (FaaS). Most prominent ones are AWS Lambda offered by amazon and cloud functions offered by google. For this project we have used AWS Lambda.

To use AWS lambda the user has to first write code to create lambda function. As of today aws lambda supports several different programming languages ranging from Python to JavaScript. Further, AWS Lambda supports all third party libraries. Once the Lambda function is loaded, the next step is to set event sources to monitor (e.g. S3 Bucket, HTTP endpoints). Lambda function would be triggered automatically when an event occurs.

In the past decade, there has been a sharp increase in the number of application utilizing machine learning and Neural Networks. Sequential neural networks are a class of neural networks used to model temporal sequences. Any application that utilizes these models consists of the following two stages of machine learning: *training* and *inference*. There has been tremendous amount of research for training machine learning model which has let to the development of several tools to expedite the process. In contrast, there has been very little research in the area of deployment of trained machine learning models. Further, existing inference systems (e.g. Tensorflow Serving, Clipper) requires server management.

To utilize the benefits of serverless computing and to address the limitations of existing inference systems (Prediction serving systems), we propose novel inference system for sequential neural networks that have several attractive features such as auto-scaling and pay-per-use model.

2. Motivation and Challenges

Serverless computing platform offers several advantages enumerated in the above section. Additionally,this platform relieves the users from the struggles of complex cluster management and configuration tools for running even simple applications. However, to fully utilize the potential of serverless computing platform for machine learning applications, we first need to address a few challenges -

1. The microvm's currently offered by the serverless platform have several resource limitations. For example,

[*]Equal contribution [1]Department of Computer Science, University of Wisconsin-Madison, Madison, USA. Correspondence to: Mohan Rao Divate Kodandarama <divatekodand@wisc.edu>.

Proceedings of the 35^{th} International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018.

AWS lambda currently limits the size of the deployment package to 50MB and the amount of RAM to 3008MB. However, deep neural networks are compute and memory intensive. We plan to explore model parallelism/data parallelism to address this problem.

2. Serverless functions are stateless and do not possess persistent storage to store training data. We plan to address this problem by maintaining the data on central persistent storage.

3. Current deep learning models consist of several millions of parameters. However, serverless functions have only a limited amount of memory.

4. Stateless nature of serverless functions stymies distributed computing.

5. Due to the above-mentioned challenges, most of the prior work has only explored utilizing serverless platform for either embarrassingly parallel applications or small traditional machine learning models.

3. Related Work

Serverless Computing: One Step Forward, Two Steps Back (Hellerstein et al., 2018) is one of the first paper that studies the advantages of serverless computing and also addresses critical gaps in first-generation serverless computing. It identifies the applications suitable for first-generation serverless computing, provides case studies to demonstrate the limitations of serverless computing for Big Data and Distributed computing settings. Further, it also provides recommendations for future generation serverless computing platforms.

PyWren (Jonas et al., 2017) proposed a serverless system developed in python with AWS Lambda. The novel idea in the paper was to use a common lambda function to fetch a serialized function stored in AWS S3, invoke the function, and write back the results to S3. PyWren addresses more general form of distributed computational model by using Redis Cluster for data shuffle operations. Similar to PyWren, we utilize Redis Cluster data shuffle operations and also for transferring data between lambda functions.

TensorFlow (Abadi et al., 2015) is an end-to-end open source platform for machine learning. It adapts the model of a computational graph in which operations are represented as nodes and edges represent the flow of tensors. The user first defines the computational graph and training is carried out as a series of forward and backward passes through the graph.

Tensorflow serving (Olston et al., 2017) is a flexible, high-performance serving system for machine learning models. It provides high-performance prediction API to simplify deploying new algorithms and experiment with new models without modifying frontend applications. Unlike Tensorflow serving our approach does not require server management.

Pytorch (Contributors) is yet another Deep learning framework that focuses on both usability and speed. Unlike initial versions of tensorflow, it does not restrict the computational graph to be static. It provides an imperative and Pythonic programming style that supports code as a model, makes debugging straight forward and is consistent with other popular scientific computing libraries, while remaining efficient and supporting hardware accelerators such as GPUs.

Parameter Server (Li et al., 2014) is a framework for distributed training of machine learning models. It distributed both data and workloads over worker nodes, while the globally shared parameters are maintained in server nodes. Unlike, Parameter Server (Li et al., 2014), in our approach worker nodes, do not store data since serverless functions are transient. Also, parameters are stored as a part of the library in the lambda layers.

Clipper (Crankshaw et al., 2017) is a general-purpose low-latency prediction serving system built on top of existing machine learning frameworks such as Tensorflow and Pytorch. It uses techniques such as caching, batching and adaptive model selection to reduce prediction latency and improve prediction throughput. However, it does not modify the underlying machine learning framework. Unlike Clipper, our approach distributes the computations involved in the model across several compute units.

Serving deep learning models in a serverless platform (Ishakian et al., 2017) evaluates the suitability of serverless computing environment for serving large neural network models. The paper carries out extensive experiment in AWS Lambda environment using MxNet deep learning framework and concludes that the inference latency for such a system can be withing an acceptable range but longer delays due to cold starts can skew the latency distribution and hence risk violating more stringent SLAs.

SerFer [5] proposed an architecture for deploying convolutional neural networks on serverless computing platform. It utilizes model parallelism by splitting the input image and processing each split in parallel using multiple serverless functions. Our work is an extension of Serfer [5] : we propose an architecture for both serving sequential neural network models on serverless computing platform.

4. Technical Contributions

Our key contributions are listed below -

1. Identify sequential neural network models suitable for serverless deployment.

2. A serverless architecture for deploying sequential neu-

ral networks.

3. Evaluation of the proposed system with respect to performance and cost.

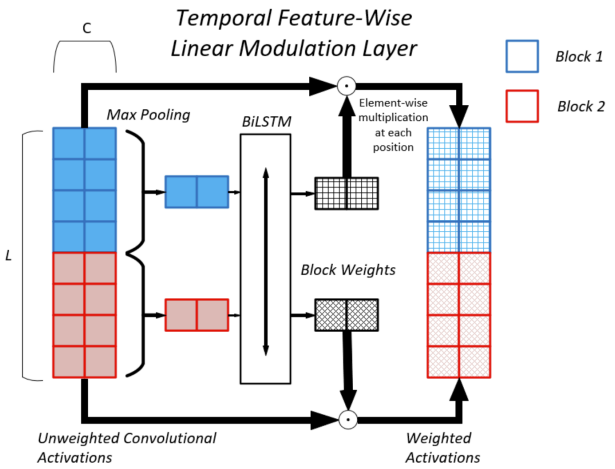

Figure 1. The TFiLM layer utilizes both convolutional and recurrent neural networks. In the above, T = 8, C = 2, B = 2, and a pooling factor of 2.
Figure from (Birnbaum et al., 2019)

4.1. Model

Neural networks are a class of machine learning model that have shown substantial progress in many applications ranging from image classification to language modelling. Convolutional neural networks and sequential neural networks are major categories of neural networks. Convolutional neural networks are dominated by convolution operation and are hence easy to parallelize. However, most widely used sequential neural networks (e.g. LSTM, GRU) have recurrent relation in which the input to a particular stage depends on the output of the previous stage. Hence, there is no straight forward way to parallelize such sequential models.

To overcome this, we model temporal sequences using convolutions neural networks with Temporal FiLM layers (Birnbaum et al., 2019). This model combines the strength of both convolutions and recurrent neural networks. More formally, it uses a recurrent neural network to alter the activations of a convolutional model which expands the receptive field of convolutional sequence models. Further, convolution operations can be easily parallelized and the length of the input to recurrent neural networks are typically very small.

4.1.1. Temporal FiLM

In this section, we briefly review Temporal FiLM (Temporal Feature-Wise Linear Modulation) (Birnbaum et al., 2019) layer. Temporal Feature-Wise Linear Modulation (TFiLM) captures long-range input dependencies in sequential inputs by modulating the activations of convolutional model by using long range information captured by a recurrent neural network.

Given $F \in R^{T \times C}$, a Tensor of activations from a convolutional layer, a TFiLM layer takes as input F and applies the following series of transformations. First, F is split along the time axis into B blocks to produce $F^{blk} \in R^{B \times T/B \times C}$. Next, we compute the affine transformation parameters using an RNN. These parameters are then used to modulate each of the blocks separately. The entire process described in Figure 1.

4.1.2. Text Classification Model

For all the experiments in this project, we use a simple text classification model shown in Figure 2. It consists of 3 convolutional layers with TFiLM layer in between them. The final layer outputs the class of the input text.

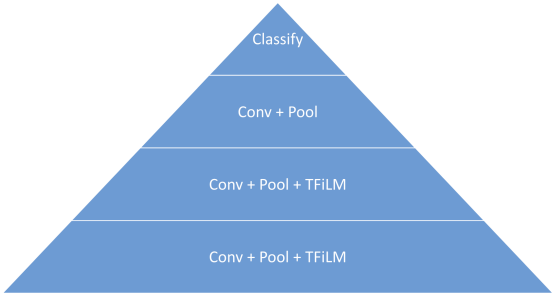

Figure 2. Simple Text Classification Model

4.2. Serverless System Architecture

The architecture of the proposed serverless inference system for the text classification model described in Figure 2 consists of three major components show in Figure 3.

1. **Master Node:** The user submits the query (Text) to the Master Node which splits the query and uploads each of the splits to the elasticache storage which triggers the first set of lambda functions.

2. **Elasticache:** Since serverless functions are transient, all the intermediate activations are stored in Elasticache stoarge. Further, each lambda function reads its input from elasticache and writes the result back to elasticache. In our implmentation, we used Redis in-memory database as elasticache storage.

3. **Lambda Functions:** Each lambda function implements a certain layer of the sequential neural network.

The proposed system consists of the following three different lambda functions -

(a) *LAMBDA0:* Implements convolution operations. Hence several LAMBDA0's are used to parallelize the convolution operation.

(b) *LAMBDA1:* Implements TFiLM layer.

(c) *LAMBDA(CLASSIFY):* Implements the final classifier layer.

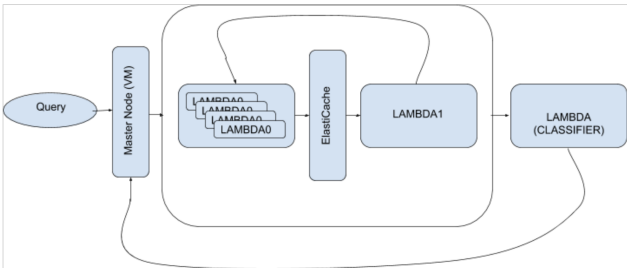

Figure 3. Architecture of Inference System
LAMBDA0 implements Conv-layers, LAMBDA1 implements Temporal FiLM layer

4.2.1. WORKFLOW

The workflow to process a single query is listed below -

1. User submits a query (Text) to the Master Node.
2. Master Node divides the input into several splits and uploads them to elasticache.
3. Several copies of LAMBDA0's are invoked and the computed results are written back to elasticache.
4. LAMBDA1 is triggered and the computed results are written back to elasticache.
5. Steps 3-4 are repeated as many number of times as the number of layers in the sequential neural network model.
6. Final write to elasticache triggers LAMBDA(CLASSIFIER) that returns back the result to the master node.

5. Experiments

For all the experiment in this project, we use services offered by AWS. In our implementation the master node is AWS t2.xlarge instance. For GPU benchmarking, AWS p2.xlarge (NVIDIA Tesla K80 GPU) instance was used.

5.1. Throughput

In this section, we compare throughput of the propose system to that of a GPU based deployment. As we see from Figures 4 and 5 the throughput of GPU based system saturates with as the input size increases. In contrast, the proposed system shows linear scaling of throughput with the input size. To achieve the same behaviour in a GPU based system, one has to use a load balance and multiple nodes with GPU's which leads to increased cost during idle time.

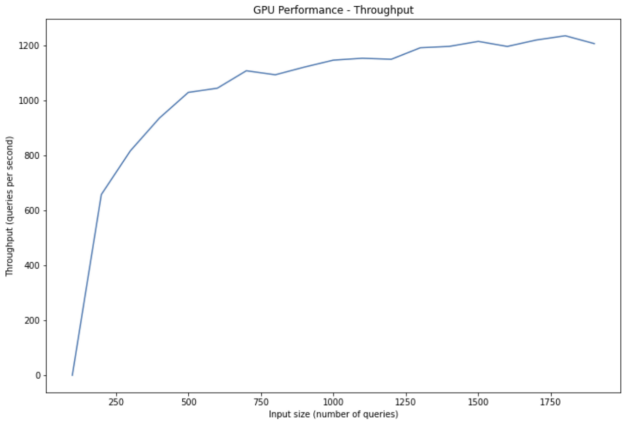

Figure 4. Throughput scaling characteristics of GPU based deployment

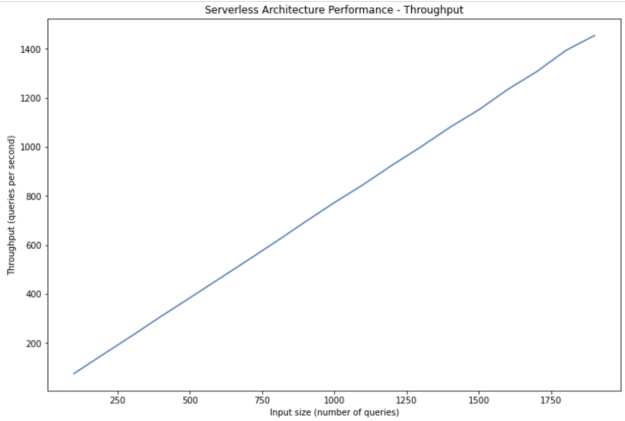

Figure 5. Throughput scaling characteristics of Serverless deployment

5.2. Latency

5.3. Cost

1. *GPU System Cost:* The cost of AWS p2.xlarge instance is about $0.9 per hour. Assuming 50% utilization, the

total number of queries processed would be 1.8 million queries.

2. *Serverless System Cost:* AWS Lambda costs about \$0.2 for million invocations and each query takes about 10 invocations. Hence, AWS Lambda costs about \$3.6. In addition the master node and elasticache (cache.m5.large) cost about \$0.188 and \$0.156. Hence the total cost would be \$3.944.

6. Conclusion and Future Work

Serverless computing platform can be used for inference of sequential models with throughput comparable to that of a GPU based system. Further, the proposed system enjoys the benefit of auto-scaling throughput and eliminated the need for server management. However, the cost of such a system is approximately four times the cost of the GPU based system. Hence ideal use case for this system are applications with non uniform traffic and large idle times.

A possible extension to this work is porting of framework like tensorflow to work on serverless computing platform.

6.1. Software and Data

For additional project resources visit GitHub.

Acknowledgements

We would like to thank Prof. Theodoros (Theo) Rekatsinas for the providing the directions towards selecting sequential models suitable for serverless platform, guiding us throughout the process and suggesting us improvements.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL http://tensorflow.org/. Software available from tensorflow.org.

Birnbaum, S., Kuleshov, V., Enam, Z., Koh, P. W. W., and Ermon, S. Temporal film: Capturing long-range sequence dependencies with feature-wise modulations. In Wallach, H., Larochelle, H., Beygelzimer, A., dAlché-Buc, F., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 32*, pp. 10287–10298. Curran Associates, Inc., 2019. URL http://papers.nips.cc/paper/9217-temporal-film-capturing-long-range-sequence-pdf.

Contributors, P. Pytorch. https://pytorch.org/.

Crankshaw, D., Wang, X., Zhou, G., Franklin, M. J., Gonzalez, J. E., and Stoica, I. Clipper: A low-latency online prediction serving system. In *Proceedings of the 14th USENIX Conference on Networked Systems Design and Implementation*, NSDI'17, pp. 613–627, Berkeley, CA, USA, 2017. USENIX Association. ISBN 978-1-931971-37-9. URL http://dl.acm.org/citation.cfm?id=3154630.3154681.

Hellerstein, J. M., Faleiro, J. M., Gonzalez, J. E., Schleier-Smith, J., Sreekanti, V., Tumanov, A., and Wu, C. Serverless computing: One step forward, two steps back. *CoRR*, abs/1812.03651, 2018. URL http://arxiv.org/abs/1812.03651.

Ishakian, V., Muthusamy, V., and Slominski, A. Serving deep learning models in a serverless platform. *CoRR*, abs/1710.08460, 2017. URL http://arxiv.org/abs/1710.08460.

Jonas, E., Pu, Q., Venkataraman, S., Stoica, I., and Recht, B. Occupy the cloud: Distributed computing for the 99%. In *Proceedings of the 2017 Symposium on Cloud Computing*, SoCC '17, pp. 445–451, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-5028-0. doi: 10.1145/3127479.3128601. URL http://doi.acm.org/10.1145/3127479.3128601.

Li, M., Andersen, D. G., Park, J. W., Smola, A. J., Ahmed, A., Josifovski, V., Long, J., Shekita, E. J., and Su, B.-Y. Scaling distributed machine learning with the parameter server. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pp. 583–598, Broomfield, CO, October 2014. USENIX Association. ISBN 978-1-931971-16-4. URL https://www.usenix.org/conference/osdi14/technical-sessions/presentation/li_mu.

Olston, C., Li, F., Harmsen, J., Soyke, J., Gorovoy, K., Lao, L., Fiedel, N., Ramesh, S., and Rajashekhar, V. Tensorflow-serving: Flexible, high-performance ml serving. In *Workshop on ML Systems at NIPS 2017*, 2017.