Serving distributed inference deep learning models in serverless computing

Kunal Mahajan
Meta
kunalmahajan@fb.com

Rumit Desai
Meta
rumitdesai@fb.com

***Abstract*—Serverless computing (SC) in an attractive win-win paradigm for cloud providers and customers, simultaneously providing greater flexibility and control over resource utilization for cloud providers while reducing costs through pay-per-use model and no capacity management for customers. While SC has been shown effective for event-triggered web applications, the use of deep learning (DL) applications on SC is limited due to latency-sensitive DL applications and stateless SC. In this paper, we focus on two key problems impacting deployment of distributed inference (DI) models on SC: resource allocation and cold start latency. To address the two problems, we propose a hybrid scheduler for identifying the optimal server resource allocation policy. The hybrid scheduler identifies container allocation based on candidate allocations from greedy strategy as well as deep reinforcement learning based allocation model.**

***Index Terms*—Serverless Computing, Distributed Inference, Deep Learning, Cold start**

I. Introduction

With abundance of existing data, continuous new data generation, and training larger deep learning models, the models have high performance [1] at the expense of requiring bigger machines in terms of CPU/GPU and memory. The individual server hardware limitations necessitate distributed inference models.

Distributed Inference models are created for inference models that can not fit on one physical machine [2]. The distributed learning model creates three major problems for the developers. Developers need to configure, deploy, and manage the resource allocations. They need to fine-tune the model's performance based on deployments. They need to optimize the capacity utilization and minimize monetary costs for running them on the cloud. All these problems divert developers from their forte of developing and improving the models.

Serverless Computing (SC) is an emerging cloud services paradigm, promising greater flexibility and reduced cost through pay-per-use model for customers and simultaneously allowing higher resource utilization for cloud providers [3], [4]. While previous work has proposed using SC for inference models, serving distributed inference models through SC and the performance, cost tradeoffs associated has not been discussed [5], [6]. In this paper, we explore the problem space of enabling distributed inference models on serverless computing and propose a deep reinforcement learning based hybrid scheduler to maximize performance and increase resource utilization for the cloud provider.

II. Background

A. Serverless Computing

In Serverless Computing, developers provide a set of functions, event triggers to execute the functions and memory requirement to the cloud provider. The pricing is dependent on how long the function runs, how many times it is invoked and how much memory it consumes. SC is ideal for event triggered applications as well as exploiting compute parallelism.

B. Distributed Inference Models

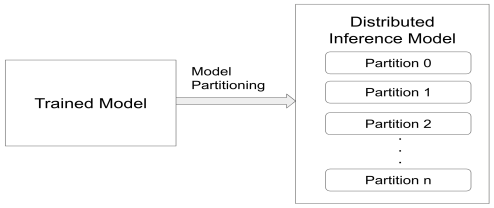

Fig. 1. Model Partitioning

Figure 1 illustrates how a distributed inference model is created from a trained model. A distributed inference model consists of multiple parts obtained from partitioning the trained model [2], [7]. The number of partitions is dependent upon the model developers and based on performance benchmarking of the model [8], [9]. Typically all the partitions consists of different embeddings/data.

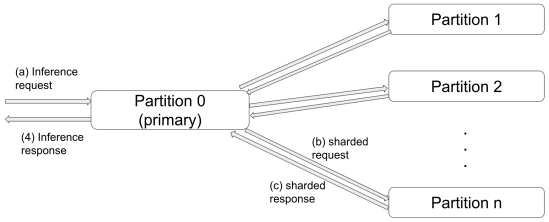

Fig. 2. Inference request processing

An inference request arrives at the primary partition, denoted by partition 0 in Figure 2. The primary partition splits the request across other partitions and waits for the other partition's responses. Upon receiving all the response, the primary partition will construct and return the inference response. Each partition is deployed in a dedicated container. To serve an

inference request, all the partitions are executed. A partition can be memory or compute intensive.

III. Problem space

To enable distributed inference models on serverless compute, developers will need to provide all the model partitions along with their memory requirements and set up the event triggers to process the inference request and obtain inference response. The implementation details of the setup, contributing to the serving latency, has been addressed by related work [10] and is not in the scope of this paper. Each partitioned model is executed in a memory-bound container following the Model-as-a-Service paradigm [11]. The amount of memory determines the amount of CPU available to the container [12]. In this section, we focus on two main problems: resource allocation and deploying recurrently trained DI models, both impacting cloud provider's capacity utilization and serving latency.

A. DI models Resource Allocation in Serverless

The computing capacity provided for each serverless container by the cloud provider is dependent on two resource dimensions: memory and CPU [12]. The container allocations on the server cannot exceed the maximum memory and CPU available on the server. Each container will consist of a user-defined memory size along with cloud-provider computed CPU allotment. Given a set of containers and a set of servers, container allocation on servers boils down to a bin-packing problem. The solution of the bin-packing problem will allow the cloud providers to obtain optimal resource utilization, effectively increasing number of containers that can be executed and increasing revenue. However, solving bin-packing problem is computationally and time intensive. For a cloud provider offering event triggered container execution in serverless architecture, adding bin-packing solve latency is not feasible for performance. Ideally, container allocation decisions should be relatively instantaneous after the event trigger and provide the optimal resource allocation.

B. Cold start latency of deploying recurrently trained DI models

One way to increase the accuracy and performance of ML models is by increasing the training dataset [1]. As services generate data continuously, the new data combined with old data leads to a larger dataset and models are retrained with this larger dataset either online or offline. The retrained models are essentially update of weights without any major changes to the feature space on which the model is built. For serving inference, the retrained models are deployed periodically over a time interval (could be couple of hours to months) that is determined based on the amount of new data and weight changes.

The new model version will be booted up on a new container. Depending on the traffic pattern of the requests, both the new version and old version executing at the same time serving different requests. For instance, assuming uniformly distributed incoming requests, say 100 requests/sec with 10 seconds inference serving latency (where the serving latency includes time to download container, container start up time and serving logic execution), both the old model version and new model will serve 1000 requests in the 10 seconds window immediately after new model version is deployed. The old model versions can be subjected to warm starts, thereby reducing serving latency. However, new model versions will incur cold start latency. This latency is exacerbated for DI models as the model partition sizes increase considerably. The difference in cold start and warm start latencies create unpredictable performance of serving requests, impacting business metrics, such as revenue and service-level agreements (SLAs).

IV. Model similarity evaluation

Previous works have shown that content similarity can be exploited by the file systems, such as Docker [13] and IPFS [14], to reduce the container start up time. Given the fact that the retrained models are weight updates without major changes in the feature space, we evaluate the similarity across two versions of recurrently trained distributed inference models. We used deep learning multi-task multi-label (MTML) models [15] for model similarity experiment. In order to compute the similarity we have considered 2 different versions of the recurrently trained model and divided each version into file blocks of size ranging from 32 kB to 1024 kB. Figure 3 describes the similarity percentage of model file block chunks for multiple model files across different block sizes. Dividing models into smaller block sizes results into higher similarity, up to 33%. Deploying the models with higher similarity on the same server can minimize the file blocks to be downloaded on the server, thereby reducing container start up latency.

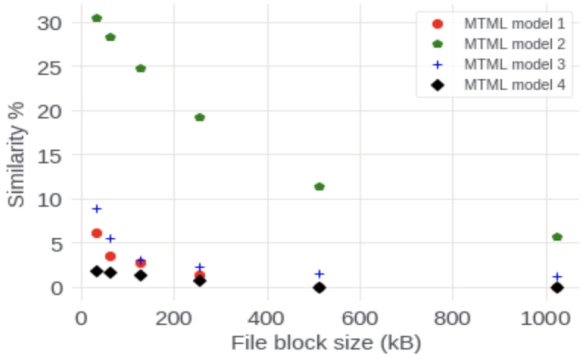

Fig. 3. Model similarity evaluation

V. Hybrid Scheduler

To enable DI models in Serverless, the cloud provider needs to optimize resource allocation while minimizing latency for serving the inference request. Resource allocation is a time-intensive process requiring periodic evaluation of multiple server parameters (cpu units, memory size, network bandwidth and their respective time-varying utilization), network topology, switch bandwidth, over-subscription ratio, etc. Moreover,

the increasing adoption of novel hardware using ASICs and GPU makes evaluations between server parameters difficult. While, minimizing latency for serving an inference request is time-critical as it requires evaluation of content similarity and existing container placements directly impacting container start up latency. We propose a hybrid scheduler, as shown in Figure 4 to tackle the time-dependency tradeoff.

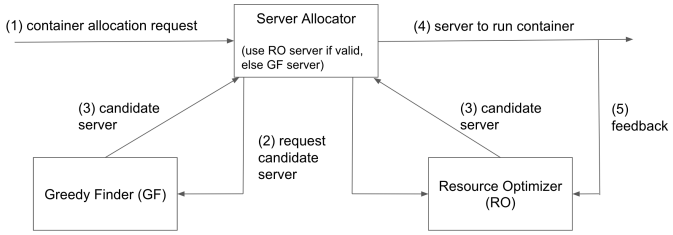

Fig. 4. Hybrid Scheduler

The hybrid scheduler consists of three components: server allocator, resource optimizer and greedy finder. The processing flow for the scheduler is as follows: (1) The server allocator receives the request to boot up a container. (2) Server Allocator requests a candidate server from both the greedy finder and the resource optimizer in parallel. (3) The resource optimizer consists of a deep reinforcement learning model, recurrently trained over time, receiving the server allocation request and providing the candidate server. The greedy finder returns the candidate server by identifying the first server that can accommodate the container. (4) Upon receiving both the candidate server, Server Allocator will prioritize using RO's candidate server if it is a valid server i.e. server has capacity to boot up the requested container. Otherwise, greedy finder's candidate server will be used. (5) Server Allocator will provide the feedback to Resource Optimizer if the RO's candidate server was used for container placement.

VI. Related work

The area of research focused on enabling machine learning in serverless computing can be divided in two categories: inference and training. A common line of work for learning explores deploying various machine learning models on serverless and measuring monetary cost and performance for serving inference requests [5], [6], [16], [17]. For training, the research has focused on identifying learning stages, such as hyperparameter search, and using data parallelism to exploit concurrent execution offered by SC [18], [19]. The work has explored design and implementation of serverless framework for training distributed machine learning, predictive analytics, and distributed double machine learning [20]–[23]. Another area of research has explored minimizing cold start latency and use of fast shared storage across serverless containers, both directly impacting performance of serverless compute [10], [24].

References

[1] A. Ng, "Machine learning yearning," 2017. [Online]. Available: http://mlyearning.com/

[2] K. Bhardwaj, C.-Y. Lin, A. Sartor, and R. Marculescu, "Memory- and communication-aware model compression for distributed deep learning inference on iot," *ACM Transactions on Embedded Computing Systems*, 2019.

[3] H. Shafiei, A. Khonsari, and P. Mousavi, "Serverless computing: A survey of opportunities, challenges and applications," 2021.

[4] K. Mahajan, D. Figueiredo, V. Misra, and D. Rubenstein, "Optimal pricing for serverless computing," in *IEEE Global Communications Conference (GLOBECOM)*, 2019.

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

[6] Z. Tu, M. Li, and J. Lin, "Pay-per-request deployment of neural network models using serverless architectures," in *Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*, 2018.

[7] P. Sun, Y. Wen, N. B. Duong Ta, and S. Yan, "Towards distributed machine learning in shared clusters: A dynamically-partitioned approach," in *IEEE International Conference on Smart Computing (SMARTCOMP)*, 2017.

[8] M. Yu, Z. Jiang, H. C. Ng, W. Wang, R. Chen, and B. Li, "Gillis: Serving large neural networks in serverless functions with automatic model partitioning," in *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*, 2021.

[9] J. Jarachanthan, L. Chen, F. Xu, and B. Li, "Amps-inf: Automatic model partitioning for serverless inference with cost efficiency," in *50th ACM International Conference on Parallel Processing (ICPP)*, 2021.

[10] Z. Jia and E. Witchel, "Boki: Stateful serverless computing with shared logs," in *ACM SIGOPS 28th Symposium on Operating Systems Principles*, 2021.

[11] H. Liu, Q. Gao, J. Li, X. Liao, H. Xiong, G. Chen, W. Wang, G. Yang, Z. Zha, D. Dong, D. Dou, and H. Xiong, "Jizhi: A fast and cost-effective model-as-a-service system for web-scale online inference at baidu," in *27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2021.

[12] "Aws lambda computing power," https://docs.aws.amazon.com/lambda/latest/operatorguide/computing-power.html, 2022, accessed: 2022-03-06.

[13] "Docker layer architecture," https://docs.docker.com/get-started/overview//, 2022, accessed: 2022-03-12.

[14] K. Mahajan, S. Mahajan, V. Misra, and D. Rubenstein, "Exploiting content similarity to address cold start in container deployments," in *15th ACM International Conference on Emerging Networking EXperiments and Technologies (CoNEXT)*, 2019.

[15] Y. Huang, W. Wang, L. Wang, and T. Tan, "Multi-task deep neural network for multi-label learning," in *IEEE International Conference on Image Processing*, 2013.

[16] Y. Wu, T. T. A. Dinh, G. Hu, M. Zhang, Y. M. Chee, and B. C. Ooi, "Serverless model serving for data science," *arXiv preprint arXiv:2103.02958*, 2021.

[17] A. Christidis, S. Moschoyiannis, C.-H. Hsu, and R. Davies, "Enabling serverless deployment of large-scale ai workloads," *IEEE Access*, 2020.

[18] J. Carreira, P. Fonseca, A. Tumanov, A. Zhang, and R. Katz, "A case for serverless machine learning," in *Workshop on Systems for ML and Open Source Software at NeurIPS*, 2018.

[19] L. Feng, P. Kudva, D. Da Silva, and J. Hu, "Exploring serverless computing for neural network training," in *IEEE 11th international conference on cloud computing (CLOUD)*, 2018.

[20] F. Xu, Y. Qin, L. Chen, Z. Zhou, and F. Liu, "λdnn: Achieving predictable distributed dnn training with serverless architectures," *IEEE Transactions on Computers*, 2021.

[21] H. Wang, D. Niu, and B. Li, "Distributed machine learning with a serverless architecture," in *IEEE Conference on Computer Communications (INFOCOM)*, 2019.

[22] A. Bhattacharjee, Y. Barve, S. Khare, S. Bao, A. Gokhale, and T. Damiano, "Stratum: A serverless framework for the lifecycle management of machine learning-based data analytics tasks," in *USENIX Conference on Operational Machine Learning (OpML)*, 2019.

[23] M. S. Kurz, "Distributed double machine learning with a serverless architecture," in *Companion of the ACM/SPEC International Conference on Performance Engineering*, 2021.

[24] K. Mahajan, *Next Generation Cloud Computing Architectures: Performance and Pricing*. Columbia University, 2021.