SLA-aware Workload Scheduling Using Hybrid Cloud Services

Dheeraj Chahal, Surya Palepu, Mayank Mishra, Rekha Singhal
TCS Research, Mumbai, India
{d.chahal, surya.palepu, mishra.m, rekha.singhal}@tcs.com

ABSTRACT

Cloud services have an auto-scaling feature for load balancing to meet the performance requirements of an application. Existing auto-scaling techniques are based on upscaling and downscaling cloud resources to distribute the dynamically varying workloads. However, bursty workloads pose many challenges for auto-scaling and sometimes result in Service Level Agreement (SLA) violations. Furthermore, over-provisioning or under-provisioning cloud resources to address dynamically evolving workloads results in performance degradation and cost escalation.

In this work, we present a workload characterization based approach for scheduling the bursty workload on a highly scalable serverless architecture in conjunction with a machine learning (ML) platform. We present the use of Amazon Web Services (AWS) ML platform SageMaker and serverless computing platform Lambda for load balancing the inference workload to avoid SLA violations. We evaluate our approach using a recommender system that is based on a deep learning model for inference.

CCS CONCEPTS

• **General and reference** → **Performance**; • **Computer systems organization** → **Cloud computing**.

KEYWORDS

Serverless, ML Platform, AWS Lambda, AWS SageMaker, Load balancing

ACM Reference Format:
Dheeraj Chahal, Surya Palepu, Mayank Mishra, Rekha Singhal. 2021. SLA-aware Workload Scheduling Using Hybrid Cloud Services. In *Proceedings of the 1st Workshop on High Performance Serverless Computing (HiPS '21), June 25, 2021, Virtual Event, Sweden.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3452413.3464789

1 INTRODUCTION

Many organizations are migrating their Artificial Intelligence (AI) application workloads to cloud due to the availability of cost-effective large infrastructure. Additionally, popular cloud vendors provide ML platforms for migrating these workloads to cloud such as AWS SageMaker [1], Microsoft Azure ML [2], Google Cloud Platform (GCP) [3], etc. These platforms provide APIs for data preprocessing, model training, deployment, and inference. The inference service is implemented on these platforms under a strict SLA [1]. The auto-scaling feature with platforms provides elasticity to dynamically scale up and scale down resources and deliver desired latency and throughput as defined in SLA. The auto-scaling feature solves the problem of under-provisioning and performance degradation by spawning new resources. Also, it makes deployment cost-effective by downscaling the resources when they are underutilized.

A bursty workload results in stress on instance resources resulting in SLA violations. To mitigate the effect of bursty workloads on SLAs, new VMs are spawned using auto-scaling features. However, it takes several minutes to instantiate a new ML instance resulting in an SLA violation during this period.

In this work, we envision the use of a serverless platform for scheduling the bursty workload in conjunction with ML platforms. Serverless architecture is emerging as a preferred paradigm for deploying deep learning workloads [2]. Although, serverless platforms also suffer from cold start problem, however, latency due to this is in few seconds. Furthermore, a serverless platform can be a viable solution for serving bursty workloads due to high scalability and a cost-effective pay-per-use cost model.

We present an approach to use an ML platform which uses dedicated virtual machines (VMs) as endpoints for serving deep models in conjunction with a serverless platform for serving the bursty traffic. Our approach is based on the characterization of the workload to distribute the load optimally between the ML platform and serverless instances. Succinctly, our contribution is as follows:

- We propose the use of ML platform with serverless architecture available as cloud services, to balance the bursty workload.
- We present a methodology for cost-effective scheduling of inference workload on these cloud services using the workload characterization technique.

We use the AWS ML Platform SageMaker and the serverless platform Lambda [4] for this study. However, we believe that this approach is also applicable to cloud services offered by other cloud vendors.

The rest of the work is presented as follows. Related work is discussed in section 2. We discuss our use-case and its deployment on cloud in section 3. Our methodology for workload characterization is discussed in section 4. Experimental setup and results are discussed in 5 followed by a conclusion in 6.

HiPS '21, June 25, 2021, Virtual Event, Sweden

ACM ISBN 978-1-4503-8388-2/21/06…$15.00
https://doi.org/10.1145/3452413.3464789

[1] https://aws.amazon.com/sagemaker/
[2] https://azure.microsoft.com/en-in/services/machine-learning/
[3] https://cloud.google.com/solutions/ai
[4] https://aws.amazon.com/lambda/

2 RELATED WORK

Cloud services have been explored extensively for SLA-aware and cost-effective Machine Learning (ML) inference serving. Zhang et al. [10] developed a scalable and cost-effective SLA-aware ML inference serving system in AWS. Bhattacharjee et al. [2] presented a highly scalable and cost-effective framework for serverless deep learning inference. Deep leaning model inference was also studied by Ishakian et al. in [7].

In our previous works, we have presented an approach for migrating ML model-based recommender systems to ML platform [3]. The use of serverless architecture for deep learning model inference is presented in our work [4]. In this work, we combine both these techniques to build an SLA-aware job scheduling methodology.

In a work similar to ours, Noval et al. [8] and Gunasekaran et al. [5] have presented frameworks called FEAT and Spock respectively exploiting cloud functions for auto-scaling and scheduling. These techniques are based on feedback control scaling or predictive analytics. We extend their work further by using our on-premise workload characterization technique for SLA-aware scheduling.

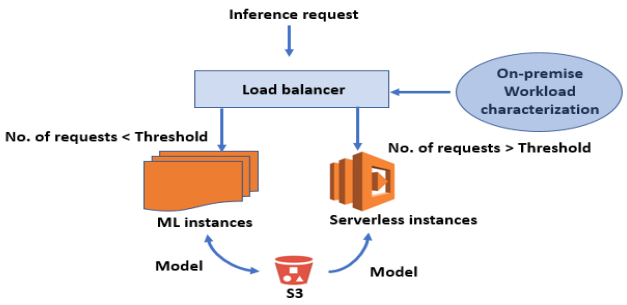

Figure 1: Overview of the load balancing architecture

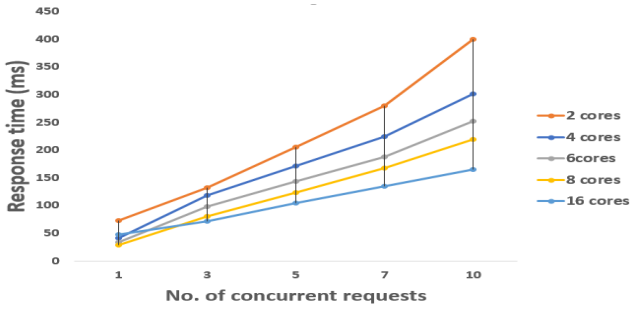

Figure 2: On-premise characterization of the model inference workload

3 OUR USE-CASE AND ITS CLOUD DEPLOYMENT

In this section, we briefly introduce the use-case for this study followed by a discussion on its deployment on ML platform and serverless architecture as shown in Figure 1.

3.1 Use-case

To evaluate our approach, we use an in-house recommender system called NISER [6]. NISER uses a Graph Neural Network (GNN) [9] based model and makes recommendations for the next product based on the user's actions, like product or item clicks in the past. The training data set used by NISER consists of past sessions of the item or product clicks by the user. During inference, the sequence of products clicked in the current session is used to recommend top n products to the user.

3.2 ML Platform based deployment

We use ML platform SageMaker APIs for deploying our NISER model for inference. The training data is saved on AWS storage S3 and downloaded by train API of SageMaker for training. On completion of training, the model is saved back on S3. During deployment, the model is downloaded from the AWS S3 and deployed using deploy API on the chosen ML instance. NISER model is hosted using the PyTorch model server that runs inside SageMaker endpoint. The predict API is used for model inference.

3.3 Serverless based deployment

We use AWS Lambda as a serverless platform for deploying our deep learning NISER model. The pre-trained model is saved on AWS storage service S3. When the first request is received by the Lambda function, the model is loaded from the S3 into its memory. Concurrent requests are served by spawning an equal number of Lambda instances. The configuration of each instance is based on the characterization of the workload. We use the CPU version of PyTorch which requires only 200 MB space and hence satisfies the storage constraint of the Lambda.

A detailed deployment scheme of a recommender system on SageMaker and Lambda is available in [3] [4] respectively.

4 OUR IMPLEMENTATION

In this section, we discuss the approach for implementing SLA-aware job scheduling. We first characterize the inference workload on-premise in a controlled environment by varying the configuration of the hardware resources. The on-premise profiling serves multiple purposes as follows

- Identifying the most appropriate configuration of the ML platform instance such as the number of cores and memory required
- Identifying the maximum number of requests that can be served by model servers without violating the SLAs
- Finding the optimal configuration for the serverless instance

Using the characterization data of the workload, we understand the maximum number of requests served by the model on the given VM instance without SLA violations. If the number of requests exceeds the threshold that might result in SLA violations, we redirect the additional requests to the serverless platform. Additionally, Identifying the appropriate configuration for a given workload avoids over-provisioning or under-provisioning of the resources and hence results in the cost-effective deployment without affecting the performance.

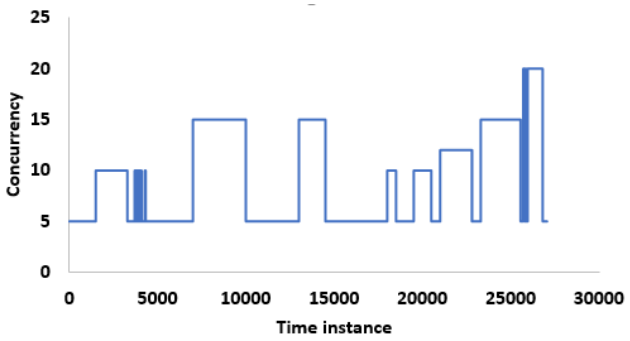

Figure 3: Bursty workload used for methodology evaluation

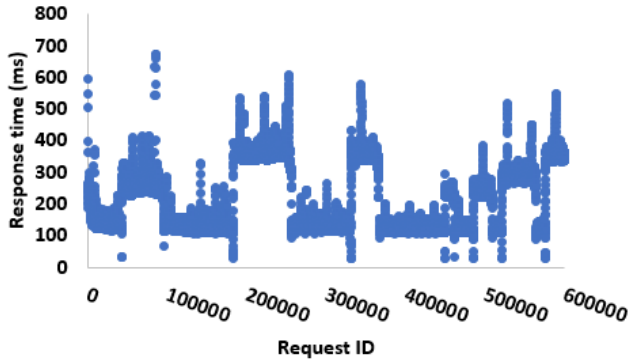

Figure 4: Response time of the inference requests using 2 core SageMaker ML instance without load balancing on Lambda

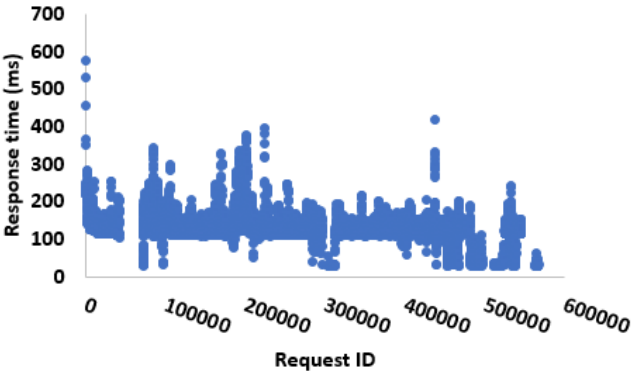

Figure 5: Response time of the inference requests using 2 core SageMaker ML instance with Lambda instances

5 EXPERIMENTAL RESULTS

In this section, we discuss the experiments that we have conducted to evaluate the efficacy of our technique. The on-premise model is deployed on an Intel Xeon 1359v2@2.45 GHz machine with 28 physical cores configured with 256 GB of memory. For our experiments with SageMaker and Lambda, we use the SageMaker Jupyter notebook instance as a load generator. The SageMaker, Lambda, and load generator instances are allocated from only ap-south-1 geography.

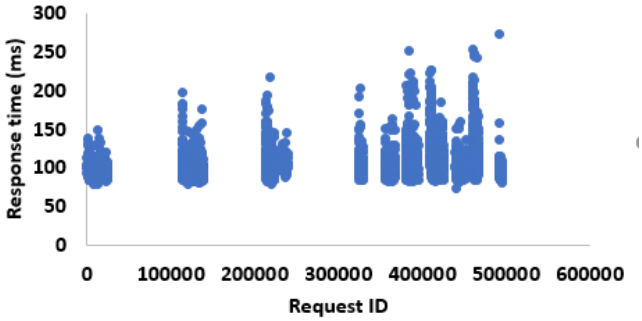

Figure 6: Response time of the inference requests served by Lambda instances

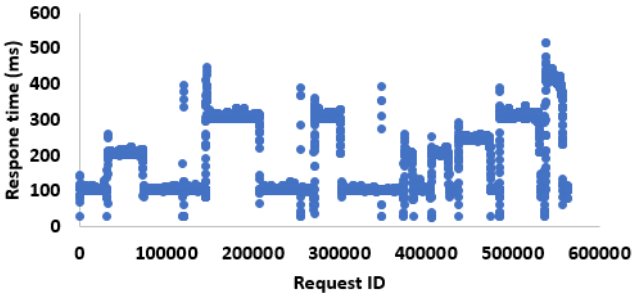

Figure 7: Response time of the inference requests using 8 core SageMaker ML instance without load balancing on Lambda

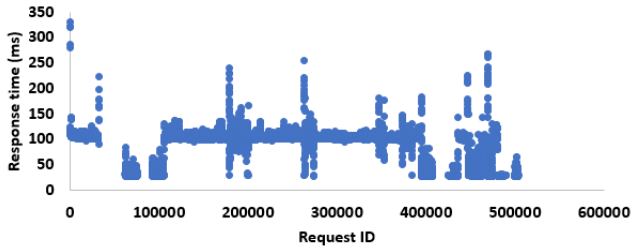

Figure 8: Response time of the inference requests using 8 core SageMaker ML instance with Lambda instances

5.1 On-premise workload characterization

Figure 2 shows the on-premise characterization of the inference workload. This Figure shows the response time of the inference requests as we increase the concurrency and the capacity of the

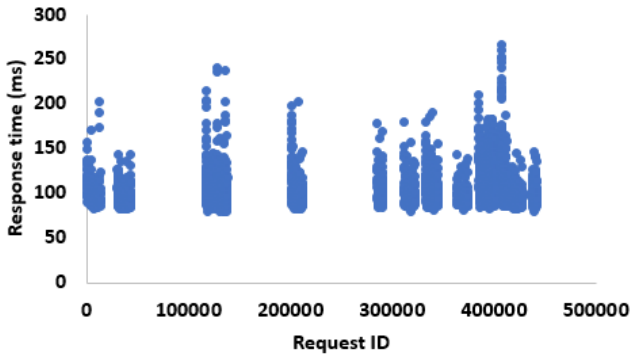

Figure 9: Response time of the inference requests served by Lambda instances

on-premise instance. One important conclusion that we can draw from this data is that for one request i.e. concurrency level one, we see significant improvement in the response time as we increase the number of cores from 2 to 4. However, an increasing number of cores further does not provide any improvement in the response time. Hence, each serverless instance that serves one inference request at a time shall be configured with 4 cores. Although the maximum number of cores available per instance on AWS Lambda for 10 GB memory is 6, however, based on the on-premise observations, we keep the number of cores restricted to 4 on the Lambda instance. This results in an approximately 35% reduction in the cost without compromising the performance. This is because the cost of using Lambda is directly proportional to the amount of memory and hence cores reserved in each instance.

Also, using the characterization data in Figure 2, we can find the maximum number of concurrent requests that can be served by an ML instance without violating the response time SLA. For example, if the model is deployed on a ml.c5.2xlarge machine (8 cores) and SLA defined response time is a maximum of 150 ms, then not more than 5 concurrent requests can be served by the instance. Any additional inference requests must be diverted to Lambda.

5.2 Scheduling on ML instance and serverless instance

In this experiment, we use a sample bursty workload where the number of concurrent requests varies between 5 and 20 (Figure 3), and the maximum allowed response time for an inference request is 200ms. As a first step, using characterization data in Figure 2, we see the maximum number of requests that can be served by any ML instance without violating response time constraint. We then deploy our recommender system model on ml.c5.large (2 core) ML instance of SageMaker that can serve a maximum of 5 concurrent requests with a response time less than 200 ms. As shown in Figure 4, when we run the highly variable workload on this ML instance, response time varies a lot violating the SLA. However, using our load balancer when we redirect the requests resulting in concurrency more than 5 to Lambda, SLA violations are reduced significantly as shown in Figure 5. Also, the requests that are sent to Lambda are served under SLA response time limits of 200 ms as shown in Figure 6.

In another experiment with the same sample inference workload, we change the response time requirements to 150ms. Again, based on the workload characterization, we observe that ml.c5.2xlarge (8 cores) SageMaker ML instance can serve 5 concurrent requests with a response time less than 150 ms. Figure 7 shows the response time increases significantly for any short burst of more than 5 concurrent requests. However, when we use Lambda instances for serving all requests above concurrency 5, we see a response time of less than 150 ms for the requests served by ML instance (Figure 8) and Lambda (Figure 9).

This study shows that on-premise workload characterization can be a useful technique for cost-efficient and SLA-aware inference workload scheduling on ML platform like SageMaker and serverless platform like Lambda.

6 CONCLUSION

We envision the use of a serverless platform as an alternative to the auto-scaling feature in VM instances on cloud. Our preliminary results show that the use of a serverless platform with dedicated instances for load balancing short-duration bursty workloads results in cost reduction without compromising on performance. In the current implementation, we see some SLA violations due to the cold start of the Lambda. Using provisional concurrency and predictive analysis to address this challenge is a work in progress. Our future work is focused on implementing this technique in a production environment for larger workloads.

REFERENCES

[1] Mohammad S Aslanpour, Adel N Toosi, Raj Gaire, and Muhammad Aamir Cheema. 2020. Auto-scaling of Web Applications in Clouds: A Tail Latency Evaluation. In *2020 IEEE/ACM 13th International Conference on Utility and Cloud Computing (UCC)*. IEEE, 186–195.

[2] A. Bhattacharjee, A. D. Chhokra, Z. Kang, H. Sun, A. Gokhale, and G. Karsai. 2019. BARISTA: Efficient and Scalable Serverless Serving System for Deep Learning Prediction Services. In *2019 IEEE International Conference on Cloud Engineering (IC2E)*. 23–33. https://doi.org/10.1109/IC2E.2019.00-10

[3] Dheeraj Chahal, Ravi Ojha, Sharod Roy Choudhury, and Manoj Nambiar. 2020. Migrating a Recommendation System to Cloud Using ML Workflow. In *Companion of the ACM/SPEC International Conference on Performance Engineering* (Edmonton AB, Canada) *(ICPE '20)*. Association for Computing Machinery, New York, NY, USA, 1–4. https://doi.org/10.1145/3375555.3384423

[4] D. Chahal, R. Ojha, M. Ramesh, and R. Singhal. 2020. Migrating Large Deep Learning Models to Serverless Architecture. In *2020 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. 111–116. https://doi.org/10.1109/ISSREW51248.2020.00047

[5] Jashwant Raj Gunasekaran, Prashanth Thinakaran, Mahmut Taylan Kandemir, Bhuvan Urgaonkar, George Kesidis, and Chita Das. 2019. Spock: Exploiting serverless functions for slo and cost aware resource procurement in public cloud. In *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*. IEEE, 199–208.

[6] Priyanka Gupta, Diksha Garg, Pankaj Malhotra, Lovekesh Vig, and Gautam Shroff. 2019. NISER: Normalized Item and Session Representations to Handle Popularity Bias. arXiv:1909.04276 [cs.IR]

[7] V. Ishakian, V. Muthusamy, and A. Slominski. 2018. Serving Deep Learning Models in a Serverless Platform. In *2018 IEEE International Conference on Cloud Engineering (IC2E)*. 257–262. https://doi.org/10.1109/IC2E.2018.00052

[8] J. H. Novak, S. K. Kasera, and R. Stutsman. 2019. Cloud Functions for Fast and Robust Resource Auto-Scaling. In *2019 11th International Conference on Communication Systems Networks (COMSNETS)*. 133–140. https://doi.org/10.1109/COMSNETS.2019.8711058

[9] Matteo Tiezzi, Giuseppe Marra, Stefano Melacci, Marco Maggini, and Marco Gori. 2020. A Lagrangian Approach to Information Propagation in Graph Neural Networks. *arXiv preprint arXiv:2002.07684* (2020).

[10] Chengliang Zhang, Minchen Yu, Wei Wang, and Feng Yan. 2019. Mark: Exploiting cloud services for cost-effective, slo-aware machine learning inference serving. In *2019 {USENIX} Annual Technical Conference ({USENIX}{ATC} 19)*. 1049–1062.