Efficient Inference Pipeline Serving by Bi-Dimensional Autoscaling

Mehran Salmani*, Kamran Razavi†, Max Mühlhäuser†, Lin Wang‡, Boris Koldehofe*
*TU Ilmenau, †Technical University of Darmstadt, ‡Paderborn University

Abstract

In recent years, there has been a rise in using multiple deep learning (DL) models chained together, producing intelligent pipelines called inference pipelines. Inference pipelines utilize the scalable cloud resources to predict user queries. Furthermore, they are user-facing, and users expect a timely response to their queries, and long delays negatively impact their experience; this timeliness expectation is addressed by the system guaranteeing an end-to-end latency service level objective (SLO). Inference pipelines receive a fluctuating workload over time due to a variable number of users sending their inference queries. As a result, it is critical to dynamically provision the right resources of the distributed cloud infrastructure to meet the SLOs cost-efficiently. Most current approaches apply horizontal scaling, i.e., deploying more instances of the DL model and load-balancing among them, to meet SLOs of inference pipelines under fluctuating workload. Recent approaches have employed vertical scaling to provide a roughly zero-delay scalability compared to horizontal scaling, which has the cold-start issue for the extra instances. However, vertical scaling is limited by the available local resources and supports fewer queries using the same amount of resources as horizontal scaling. In this work, we propose Biscale to address the problem of efficiently retaining the latency SLO of inference pipelines under workload fluctuation by combining vertical and horizontal scaling. In particular, Biscale uses vertical scaling to react quickly to increased workload demand to retain the SLO and switches to horizontal scaling when the workload stabilizes for cost efficiency. Our preliminary result shows that Biscale has up to $11\times$ fewer SLO violations than horizontal scaling alone, and up to 75% fewer CPU cores than vertical scaling alone.

Index Terms

Deep Learning Serving, Resource Autocaling, Distributed Systems

I. Introduction

Deep learning (DL) has enabled high-quality services in various application domains like computer vision, audio processing, machine translation, and chatbots [1], [2]. Trained DL models are served on the distributed infrastructures, like cloud or edge datacenters, to predict user queries, which is known as inference services. Many modern inference services interconnect DL models to form intelligent pipelines [3]–[5] such as a video analytics service comprised of an object detection stage followed by a specific classification of the detected objects, or an audio translation pipeline including audio processing and machine translation stages.

DL inference services are online resource-intensive services where users sending queries expect responsiveness; because of this, providers specify a latency service level objective (SLO) within which the system must respond to user queries. Further, inference services face a fluctuating workload over time [6]. Effective DL model serving (inference serving) is key to providing latency SLO-aware and efficient DL inference under the fluctuating workload.

Vertical and horizontal scaling are two primary mechanisms adapted in the literature to meet the latency SLO without over-provisioning resources. Vertical scaling changes the local machine resources (CPU cores) allocated to the running inference service instance (container). In-place vertical scaling [7], which means changing instance resources without restarting/rescheduling, enables scaling a service instance according to its varying resource requirement almost immediately. Horizontal scaling works by changing the number of service instances (replicas) in the cluster and load-balancing the incoming user queries across them [4], [5], [8].

Vertical and horizontal scaling come with their advantages and disadvantages for inference services. Vertical scaling offers quick reaction time to workload fluctuation, but with limited scalability for two reasons: 1. The maximum amount of resources that one can allocate to the service depends on the available free resources of the local machine, and 2. The performance boost received from vertical scaling for inference services (more CPU threads to process a query) depends on the parallelization degree of the DL model computation and the Amdahl's

© 2025 by the authors. – Licensee Technische Universität Ilmenau, Deutschland.

DOI: 10.22032/dbt.67108 *DOI (proceedings): 10.22032/dbt.66316*

law [9], which means increasing CPU cores (and subsequently, number of threads) from for example 1 to 8 does not provide 8× improvement on the processing latency, but less. Horizontal scaling is more efficient and can provide a higher throughput, but suffers from the cold-start issue [10] that leads to the violation of latency SLO during the scaling until the instances are up and the DL models are loaded.

In this paper, we introduce Biscale, an inference pipeline serving system that combines the two scaling mechanisms to benefit from their advantages and overcome their limitations. To combine the two, we aim to answer the following questions: When should we apply which mechanism? How to use each mechanism? Furthermore, as we consider inference pipelines, how does the interdependency of pipeline stages affect the decisions for the subsequent stages? We answer these questions in section II.

II. BISCALE

Biscale aims to combine horizontal and vertical scaling for inference pipelines to meet latency SLOs. In the following, we first discuss the challenges of combining horizontal and vertical scaling for inference pipelines, then we formulate the bi-dimensional autoscaling problem and introduce the system design.

A. Challenges

The challenges of designing a bi-dimensional autoscaling for SLO-aware inference pipelines include:

Pipeline dependency: Scaling decisions at a stage of an inference pipeline impact the downstream stages. Employing an independent autoscaler for each stage can lead to end-to-end latency SLO violations and resource inefficiencies. An autoscaler must find the new overall system configuration with respect to the external variables (workload), internal pipeline structure, and latency SLO, in real time.

Batch size: Batching is another decision in conjunction with scaling at a stage that affects the downstream stages. Batching refers to aggregating multiple queries and processing them as one bulkier query. Increasing batch size improves the throughput but also increases the processing and queuing latency. Batching for pipeline stages shall be made considering the query arrival rate and the pipeline dependency to retain the latency SLO.

When to use which scaling mechanism: It is critical to decide when to use which mechanism to benefit from the responsiveness of vertical scaling and efficiency of horizontal scaling, as early activation of horizontal scaling to save costs violates SLOs and solely relying on vertical scaling leads to resource inefficiency.

How to switch to horizontal scaling: When deciding to switch to horizontal scaling to save costs, should we bring more new 1-core instances or fewer new m-core instances? Since our purpose for applying horizontal scaling is cost-efficiency, we should choose the configuration with the fewest overall CPU cores.

B. Problem Formulation

To achieve our goals of retaining latency SLO cost-efficiently under fluctuating workload, we present an Integer Program (IP) formulation as follows:

$$\begin{aligned} \min \quad & \sum_{s \in S} n_s \times c_s \\ \text{subject to} \quad & \sum_{s \in S} l_s(b_s, c_s) + q_s(b_s) \leq SLO \\ & \lambda \leq h_s(b_s, c_s) \times n_s, \forall s \in S \\ & b_s, c_s, n_s \in \mathbb{Z}^+, \forall s \in S \end{aligned} \tag{1}$$

The first line of the Equation 1 is the objective function to minimize the overall cost, i.e., number of instances (n_s) multiplied by CPU cores per instance (c_s) for all the stages (S). The second line of the formulation is a constraint to ensure that the end-to-end latency (sum of queuing and processing latency of all stages) is below the SLO. The third line ensures that the throughput for every stage ($h_s(b_s, c_s) \times n_s$) is greater than the current query arrival rate (λ). Activating horizontal scaling when the query arrival rate is increasing results in violating the throughput constraint of the equation for the duration of cold-start and neglecting to use horizontal scaling when the workload is stable leads to cost inefficiency.

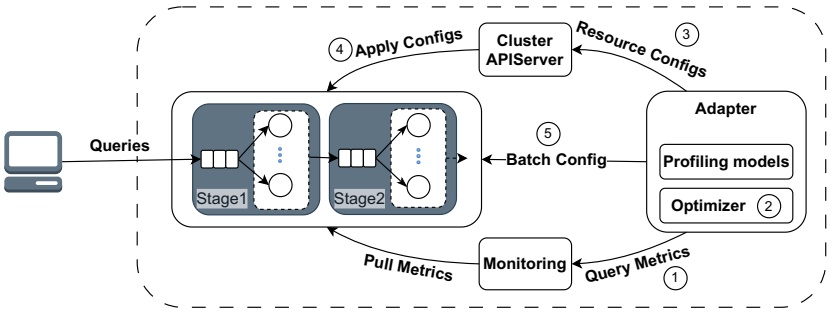


Fig. 1: System Design of Biscale

C. Approach and System Design

We explain how Biscale addresses the challenges in subsection II-A:

When to use which scaling mechanism: We use vertical scaling whenever the query arrival rate increases compared to the current system throughput, or when we forecast (using a time-series forecaster) an increase in the query arrival rate in near future, to quickly support the queries with the increased throughput achieved by the processing speedup. Also, whenever the query arrival forecasting indicates no increase shortly, we conclude that the workload will be stable and switch to horizontal scaling to benefit from its cost efficiency without SLO violations.

How to switch to horizontal scaling: Following the Amdahl's law, the speedup (L) achieved for query processing by increasing the number of CPU cores (c) can be calculated by $L(c) = \frac{1}{(1-p)+\frac{p}{c}}$ where $0 \leq p \leq 1$ is the parallelization degree (the closer to 1, the more parallelizable the task is). With one instance having one CPU core, $L(1) = 1$. With an instance having two CPU cores, $L(2) = \frac{2}{2-p} \leq 2$, which shows having two 1-core instances has a better speedup compared to having one 2-core instance, i.e., $L(2) \leq 2 \times L(1)$. This argument can be generalized for any c to show that having 1-core instances provides the highest speedup. For this reason, when switching to horizontal scaling, we bring extra 1-core instances up and update the existing instances' CPU cores to 1.

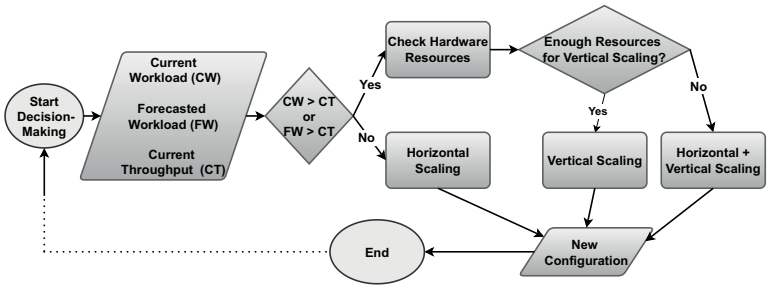
Pipeline dependency, scaling, and batching: As we discussed in subsection II-A, the scaling and batching decision in a pipeline stage affects the subsequent stages. We address this by using an optimizer aware of the pipeline structure and the performance of every stage to find the new configurations in one shot for all stages, considering the SLO.

Figure 1 shows the system design of Biscale. The Monitoring component pulls the query arrival rate per second and stores it. The adaptation starts with the Adapter component querying the Monitoring to receive the current query arrival rate and a history of arrival rates from the past minute. It then forecasts the maximum query arrival in 10 seconds in the future using an LSTM time-series forecaster. Based on the conditions in Figure 2a, it decides whether to use horizontal or vertical scaling. Then, the relevant optimizer for either vertical scaling or horizontal scaling, by using the performance profiles of the DL models on different CPU and batch configurations and the problem formulation, finds the new configurations for the number of instances, CPU per instance, and batch size, for all stages. Then the new configurations regarding resource allocation (changing scales) are applied through a cluster API server; batch size configurations are installed directly in the stages queues.

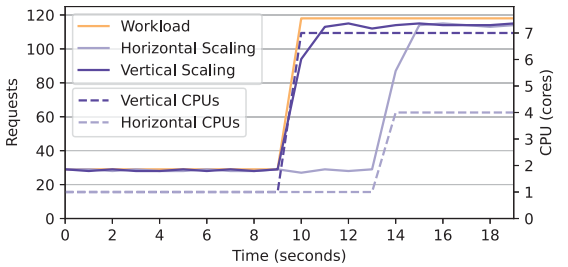
D. Preliminary Result

We ran an experiment to show the pros and cons of vertical and horizontal scaling and how the combination can improve inference serving. The results of this experiment are in Figure 2b. In this experiment, we send ≈ 30 queries per second (QPS) for 10 seconds, where an instance with one CPU core can sustain the throughput. After 10 seconds, we increase the workload to ≈ 120 QPS. Vertical scaling can quickly (in ≈ 250 ms) increase the instance scale from one CPU core to seven CPU cores and absorb the burst (about $0.25 \times (120 - 30) \approx 23$ queries violate the latency SLO). Horizontal scaling needs only three more instances with one CPU core, but it takes more than three seconds to spawn the extra instances; during this period, more than $3 \times (120 - 30) = 270$ queries violate the latency SLO.

This experiment illustrates that if we use vertical scaling to increase the CPU cores from one to seven when the workload increases from 30 QPS to 120 QPS, and switch to horizontal scaling to have four 1-core instances when the workload becomes stable, we will have $11\times$ less SLO violations than horizontal scaling during the burst, and 75% less CPU cores compared to vertical scaling when the workload is stable.



(a) Transition



(b) The reaction time to workload changes and the number of CPU cores for vertical and horizontal scaling.

Fig. 2

III. Conclusion and Future Work

In this paper, we investigated the advantages and disadvantages of horizontal and in-place vertical scaling to motivate proposing a mechanism that combines them to enable efficient and SLO-aware inference pipeline serving. The combination provides a better mechanism than vertical scaling since it does not have the hardware availability and model parallelizability limitations of vertical scaling. Also, it outperforms horizontal scaling since it does not suffer from the cold-start delays as horizontal scaling.

Our approach spans only CPU inferencing, as there is native support of in-place vertical scaling [11] for CPU resources, while GPUs lack this feature. CUDA MPS [12] could be enabled to allocate only a share of the GPU to the model, but it does not support in-place updating of the share. We leave the investigation of integrating GPUs into the system to have a heterogeneous adaptive environment for inference pipelines as a future work.

References

[1] N. Maslej, L. Fattorini, R. Perrault, V. Parli, A. Reuel, E. Brynjolfsson, J. Etchemendy, K. Ligett, T. Lyons, J. Manyika, J. C. Niebles, Y. Shoham, R. Wald, , and J. Clark, "The AI index 2024 annual report," 2024.

[2] M. Salmani, S. Ghafouri, A. Sanaee, K. Razavi, M. Mühlhäuser, J. Doyle, P. Jamshidi, and M. Sharifi, "Reconciling high accuracy, cost-efficiency, and low latency of inference serving systems," in *Proceedings of the 3rd Workshop on Machine Learning and Systems*, 2023, pp. 78–86.

[3] S. Ghafouri, K. Razavi, M. Salmani, A. Sanaee, T. L. Botran, L. Wang, J. Doyle, and P. Jamshidi, "[Solution] IPA: Inference Pipeline Adaptation to achieve high accuracy and cost-efficiency," *Journal of Systems Research*, vol. 4, no. 1, 2024.

[4] K. Razavi, M. Luthra, B. Koldehofe, M. Mühlhäuser, and L. Wang, "FA2: Fast, accurate autoscaling for serving deep learning inference with SLA guarantees," in *2022 IEEE 28th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2022, pp. 146–159.

[5] D. Crankshaw, G.-E. Sela, X. Mo, C. Zumar, I. Stoica, J. Gonzalez, and A. Tumanov, "InferLine: Latency-aware provisioning and scaling for prediction serving pipelines," in *Proceedings of the 11th ACM Symposium on Cloud Computing*, 2020, pp. 477–491.

[6] A. Khare, D. Garg, S. Kalra, S. Grandhi, I. Stoica, and A. Tumanov, "SuperServe: Fine-Grained inference serving for unpredictable workloads," in *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*, 2025, pp. 739–758.

[7] K. Razavi, S. Ghafouri, M. Mühlhäuser, P. Jamshidi, and L. Wang, "Sponge: Inference serving with dynamic SLOs using in-place vertical scaling," in *Proceedings of the 4th Workshop on Machine Learning and Systems*, 2024, pp. 184–191.

[8] F. Romero, Q. Li, N. J. Yadwadkar, and C. Kozyrakis, "INFaaS: Automated model-less inference serving," in *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, 2021, pp. 397–411.

[9] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *Proceedings of the April 18-20, 1967, spring joint computer conference*, 1967, pp. 483–485.

[10] A. Joosen, A. Hassan, M. Asenov, R. Singh, L. Darlow, J. Wang, Q. Deng, and A. Barker, "Serverless cold starts and where to find them," in *Proceedings of the Twentieth European Conference on Computer Systems*, 2025, pp. 938–953.

[11] T. K. Authors, "In-place resource resize for Kubernetes Pods," https://kubernetes.io/blog/2023/05/12/in-place-pod-resize-alpha/, 2023, accessed on 30.01.2024.

[12] N. Corporation, "CUDA MPS," https://docs.nvidia.com/deploy/mps/index.html, 2021, accessed on 13.05.2025.