ScaleServe: A Scalable Multi-GPU Machine Learning Inference System and Benchmarking Suite

Ali Jahanshahi
ajaha004@ucr.edu
University of California, Riverside
USA

Marcus Chow
mchow009@@ece.ucr.edu
University of California, Riverside
USA

Daniel Wong
danwong@ucr.edu
University of California, Riverside
Riverside, California, USA

ABSTRACT

We present, ScaleServe, a scalable multi-GPU machine learning inference system that (1) is built on an end-to-end open-sourced software stack, (2) is hardware vendor-agnostic, and (3) is designed with modular components to provide users with ease to modify and extend various configuration knobs. ScaleServe also provides detailed performance metrics from different layers of the inference server which allow designers to pinpoint bottlenecks.

We demonstrate ScaleServe 's serving scalability with several machine learning tasks including computer vision and natural language processing on an 8-GPU server. The performance results for ResNet152 shows that ScaleServe is able to scale well on a multi-GPU platform.

ACM Reference Format:
Ali Jahanshahi, Marcus Chow, and Daniel Wong. 2022. ScaleServe: A Scalable Multi-GPU Machine Learning Inference System and Benchmarking Suite. In *The 14th Workshop on General Purpose Processing Using GPU (GPGPU'22), April 3, 2022, Seoul, Republic of Korea.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3530390.3532735

1 INTRODUCTION

As machine learning (ML) accuracy improves near human-level performance, the demand for deploying trained models into production pipelines has emerged. Inference systems as a cloud solution, a.k.a Inference-as-a-service, have been developed to provide high-throughput inference service on multi-GPU systems. Such commercial examples include Triton Inference Server, Tensorflow-serving, TorchServe, and RayServe. Examples of academic solutions include Clipper [1], and INFaaS [3].

To the best of our knowledge, no existing inference system solutions are GPU vendor-agnostic nor provide open-source end-to-end orchestration of the software and hardware components of the inference system, both critical aspects which allows an open-platform for accelerated inference server research. For example, Nvidia's Triton Inference Server is a widely used inference system solution, but is not hardware vendor-agnostic as it utilizes CUDA-specific functionality and requires closed-source backends such as NVIDA's deep learning library (cuDNN) or NVIDIA driver. In fact, the majority of existing inference servers mainly target Nvidia-based products leading to the lack of end-to-end open-source inference serving solutions. With diverse emerging multi-GPU products coming from vendors, such as AMD and Intel, it is imperative that inference servers are designed to be hardware vendor-agnostic and provide end-to-end open-source software stacks.

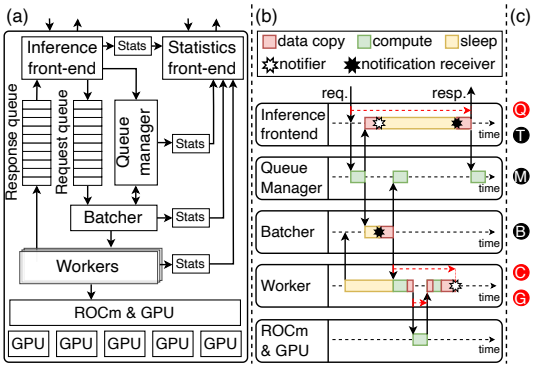

Figure 1: a) ScaleServe design and components (Sec.2.1). b) Request/response life-cycle in ScaleServe (Sec.2.2). c) Real-time statistics provided by each component (red circles correspond to the latency shown by the red arrows) (Sec.2.3).

While ML inference benchmarking exist, most notably with MLPerf inference benchmarking suite [2], this benchmarking suite provides very few system-level statistic and mainly focus on inference-level performance. Thus, there is a need to expose the inference system's system-level statistic to the designers so that they are able to fine-tune the server configurations and software runtime to utilize the system's resources efficiently. In this work, we aim to solve the aforementioned limitations by introducing ScaleServe, a highly configurable multi-GPU inference system that is end-to-end open-source, easily configurable and extendable, and provides detailed component-level online performance statistics.

2 SCALESERVE MULTI-GPU INFERENCE SERVER

2.1 Design and Components

Figure 1(a) shows the architecture of ScaleServe . consisting of:

Inference Front-end (per model): a multi-threaded process responsible for accepting the asynchronous gRPC requests from clients and sending back the inference result (response) to them.

Request/Response Queues (per model): queues are shared memory segments for storing request's (response's) data to be served (sent to the client). The queue size (number of requests accepted for serving) is a configurable knob as it contributes to the performance of the server.

GPGPU'22, April 3, 2022, Seoul, Republic of Korea

ACM ISBN 978-1-4503-9348-5/22/04.
https://doi.org/10.1145/3530390.3532735

This work was supported in part by NSF Grant 2047521 and 1955650.

Table 1: Statistics provided by ScaleServe's components. For each statistics min, max, mean, and percentile (25th, 50th, 75th, 90th, 95th, and 99th) metrics are provided.

Symbol	Component	Description
Q	Inf. Front-end	Queue latency (request arrival to response reply)
T	Inf. Front-end	Thread pool utilization
M	Que. Manager	Request/response queue utilization
B	Batcher	Batch size
C (∈ Q)	Worker	Pre-/post-processing + data copies (= CPU time)
G (∈ Q)	Worker	Inference latency on GPU

Queue Manager (per model): responsible for managing request/response queues by providing a thread-safe interface for tracking occupied and available queue slots.

Workers (scalable/configurable): gets a batch of requests from Batcher and performs pre-processing, inference, and post-processing on a batch of requests. The number of worker threads is another configurable knob.

Batcher (per model): provides a thread-safe interface for worker threads to request for a batch of requests. Batcher copies the requests' data from the request queue to make the batch and pass it to the worker. After making the batch, Batcher notifies the queue manager that the request data are being processed and the slot is available again for accepting new requests.

Statistics Front-end (global): each component of ScaleServe stores its statistics in a shared memory at run-time (shown by *Stats* in Figure 1(a). If requested by a client, the front-end pulls the requested statistics from components' associated statistics shared memory. Statistics Front-end provides a gRPC endpoint for requesting statistics from the server.

2.2 Request/response life-cycle in ScaleServe

Figure 1(b) illustrates the life-cycle of a request. When a request arrives, a thread from thread pool of the Inference Front-end is assigned to serve the request. The thread, acquires a free request slot from the request queue through the Queue Manager. Then, the request data is copied to the response queue slot to be served. The thread sleeps until notified by a worker thread to collect the results, releases the response queue slot through Queue Manager, and send the inference result back to the client.

At the same time, waiting workers are notified by the Batcher for a new batch. The worker performs pre-processing, inference, post-processing, copies the results to their designated resp. queue slots, and finally notify the Inference Front-end thread to collect the inference results. Batcher gets notified on every request queue slot allocation by Inference Front-end so that it keeps track of incoming request to make a batch based on its batching policy, batch size, and batch polling interval.

2.3 Real-time statistics provided by ScaleServe

Each component in ScaleServe, launches an independent, observer thread to periodically collect and store stats at run-time. The statistics front-end (upon user request), pulls the requested data from its corresponding shared-memory region, calculates the requested metric(s), and responds the result back to the user. Figure 1(c) shows the statistics each component of ScaleServe provides. Table 1 shows more detailed description of statistics.

2.4 Configuration knobs in ScaleServe

Table 2 lists the available knobs, their description, and the granularity at which they can be configured. Global means the knob will be set for all models being served in ScaleServe. Per model means it can be configured per model being served by ScaleServe.

Table 2: Inference server configuration knobs exposed to designers in ScaleServe and their configuration granularity.

Component	Knob	Configuration gran.
Inf. Front-end	Thread pool size	Global
Que. Manager	Request/response queues' size	Global
Batcher	Batch polling policy	Per model
Worker	Number of worker threads	Per model
Worker	Batching policy	Per model
Worker	Batch size	Per model

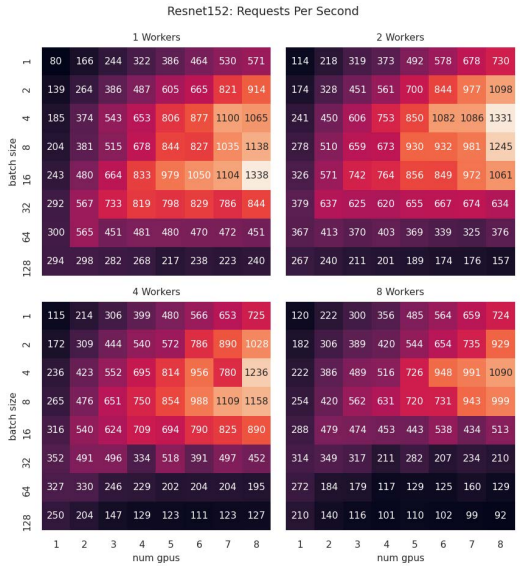

Figure 2: Server RPS for ResNet152 with respect to batch size, number of gpus, and workers per gpu.

3 CASE STUDY: RESNET152

We deployed ScaleServe on a server featuring 8 AMD MI50 GPUs, AMD EPYC 7302 16-Core Processor, 512 GB RAM, Ubuntu 18.04 LTS with kernel 5.4.0, and Intel 10G X550T network card. To demonstrate the performance and scalability of ScaleServe, we performed a design space exploration on the configuration knobs exposed by ScaleServe. Figure 2 shows the server performance with variety of configurations; scaling resources such as GPU or worker threads, as well as scaling batch size for ResNet152 which is being used widely in variety of computer vision tasks including object classification and detection. In this experiment, we fixed thread pool size to 50, req./resp. queues' size to 256 which means the inference front-end no further accepts requests as long as there are 256 requests in the server to be processed, for batch polling policy, we randomly pick a number from an interval (example [1, 5] msec) and poll the req. queue for a new batch, batching policy fixed to relaxed/dynamic to simulate the real-world scenario.

REFERENCES

[1] Daniel Crankshaw, Xin Wang, Guilio Zhou, Michael J Franklin, Joseph E Gonzalez, and Ion Stoica. 2017. Clipper: A Low-Latency Online Prediction Serving System. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*.

[2] Vijay Janapa Reddi and et al. 2020. Mlperf inference benchmark. In *ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*.

[3] Francisco Romero, Qian Li, Neeraja J Yadwadkar, and Christos Kozyrakis. 2021. INFaaS: Automated Model-less Inference Serving. In *2021 USENIX Annual Technical Conference (USENIX ATC)*.