Deep Learning Inference Service at Microsoft

Jonathan Soifer, Jason Li, Mingqin Li, Jeffrey Zhu, Yingnan Li, Yuxiong He, Elton Zheng, Adi Oltean, Maya Mosyak, Chris Barnes, Thomas Liu, and Junhua Wang, *Microsoft*

https://www.usenix.org/conference/opml19/presentation/soifer

This paper is included in the Proceedings of the 2019 USENIX Conference on Operational Machine Learning (OpML '19).

May 20, 2019 • Santa Clara, CA, USA

ISBN 978-1-939133-00-7

Open access to the Proceedings of the 2019 USENIX Conference on Operational Machine Learning is sponsored by USENIX.

Deep Learning Inference Service at Microsoft

Jonathan Soifer, Jason Li, Mingqin Li, Jeffrey Zhu, Yingnan Li, Yuxiong He,
Elton Zheng, Adi Oltean, Maya Mosyak, Chris Barnes, Thomas Liu, Junhua Wang
Microsoft

Abstract

This paper introduces the Deep Learning Inference Service, an online production service at Microsoft for ultra-low-latency deep neural network model inference. We present the system architecture and deep dive into core concepts such as intelligent model placement, heterogeneous resource management, resource isolation, and efficient routing. We also present production scale and performance numbers.

1 Introduction

Over the past couple of years, many services across Microsoft have adopted deep neural networks (DNN) to deliver novel capabilities. For example, the Bing search engine uses DNNs to improve search relevance by encoding user queries and web documents into semantic vectors, where the distance between vectors represents the similarity between query and document [6, 7, 9]. However, due to the computational complexity of DNNs, application-embedded inference and off-the-shelf micro-service offerings don't meet the necessary scale, performance, and efficiency requirements for many of Microsoft's critical production services. These services receive hundreds of thousands calls per second and are often constrained to single-digit millisecond latency budgets. DNNs authored across a spectrum of operating systems and frameworks must be provisioned efficiently on heterogeneous datacenter hardware, such as CPU, GPU, and FPGA. With rapid innovations in DNN architectures, the system must be extensible and agile by supporting fast model validation, deployment, and proper version control. Deep Learning Inference Service (DLIS) is a dedicated platform to address these requirements, and now serves as the inference backend for many teams across Microsoft such as web search, advertising, and Office intelligence. At present, DLIS is handling three million inference calls per second, served from tens of thousands of model instances, and deployed in more than 20 data centers world-wide.

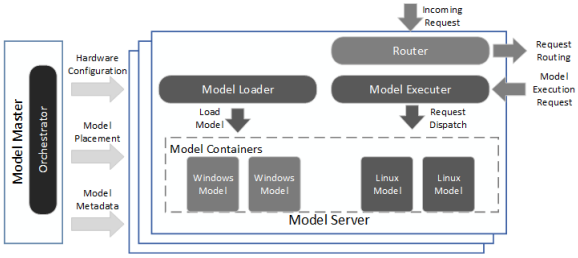

Figure 1: DLIS Architecture

2 System Overview

Figure 1 shows an overview of DLIS and its key components. Model Master (MM) is a singleton orchestrator responsible for intelligently provisioning model containers onto one or more servers by factoring in model requirements and hardware resources. Model Servers (MS) are the server unit and can number in the thousands. They have two roles: routing and model execution. MS receives an incoming request from a client and efficiently routes it to another MS hosting an instance of the requested model. The MS receiving the request from the routing server then executes the request with low-latency. These three core functionalities of provisioning, routing, and model execution will be discussed in detail in sections 3, 4, and 5. In addition to the features discussed in this paper, MS is flexible. It runs on both Windows and Linux and supports multiple orchestrators outside of MM. These include YARN and Kubernetes [1, 3].

3 Intelligent Model Placement

The performance of different DNN models varies across hardware. For example, convolutional neural network models are most performant on GPU, while recurrent neural network models often achieve lower latency on FPGA or CPU [5, 8, 10]. DLIS needs to understand different models' requirements and place them efficiently onto matching hardware. This neces-

sitates an intelligent model placement system in the Model Master.

Model Placement. MM has a global view of all servers and their respective hardware and resource availability, which includes CPU instruction sets, number of CPU cores, amount of memory, and number of GPUs, among others. MM is aware of a model's estimated resource usage through a validation test run prior to model deployment. To host an instance of a model, servers must satisfy the following constraints: they must meet the hardware requirements of the model, they must have available resources to host at least one instance, and they must be spread across a certain number of fault domains. Placement is multi-tenant and dynamic. Instances can be hosted with other instances of the same model or a different model. Further, MM reads resource usage at runtime and can decide to move instances to different servers at any time.

Diverse Hardware Management. Specialized hardware such as GPU and FPGA requires proper configuration and management. To support this, DLIS uses a special model called a machine configuration model (MCM). MCMs configure servers at regular intervals. For example, an MCM may run every ten minutes, installing a GPU driver, resetting GPU clock speed, and verifying overall GPU health.

4 Low-Latency Model Execution

DNNs are computationally complex. Different levels of optimization are required to achieve low-latency serving. DLIS supports both system- and model-level optimizations [10]. This section describes the system optimizations, while model optimizations are outside the scope of this paper.

Resource Isolation and Data Locality. For low-latency serving in a multi-tenant environment, data access is localized to take advantage of different cache layers, while resource isolation is used to ensure that model instances do not interfere with each other. To achieve this, MS isolates model instances in containers. Linux models are run in Docker containers, while Windows models are run in custom containers under job objects [2]. DLIS enforces resource isolation in the form of processor affinity, NUMA affinity (when hardware supports it), and memory restrictions. Processor affinity allows model-critical data to stay in the nearest processor caches. NUMA affinity guarantees that a model doesn't have to cross memory banks. Memory restrictions ensure that the model never needs to access disk. Together, they ensures that model instances localize data access with minimal interference from other instances.

Server-to-Model Communication. Container-based isolation leads to a need for efficient communication between server and model. To support this, Linux models are wrapped in custom infrastructure to enable efficient communication over UDP. Windows models are wrapped in custom infrastructure to enable efficient communication over a shared-memory queue. The shared-memory queue provides inter-process communication latencies of less than a few hundred microseconds.

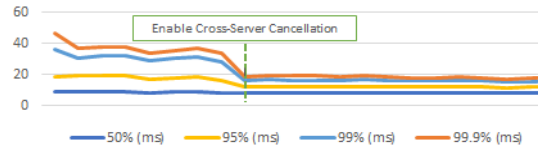

Figure 2: Latency Before and After Enabling Cross-Server Cancellation

5 Efficient Routing

Traffic patterns to model inference at Microsoft come with unique challenges. First, there is frequent burst traffic - many requests in the span of a few milliseconds. In extreme scenarios, each request may be a batch with hundreds of sub-requests. Such burst traffic can lead to many requests being enqueued on the same server. Next, tail model latency is often very near performance SLA. These challenges necessitate MS to route requests with minimal overhead.

Backup Requests and Cross-Server Cancellation. With frequent burst traffic, it is hard to accurately predict each server's load. To compensate, MS router supports backup requests which serves as a second chance if the first request has a risk of missing SLA. Backup requests can be either statically configured (for example, sending a backup request after 5ms) or dynamically configured (for example, sending a backup request at the 95th-percentile model latency). For many low-latency scenarios, backup requests alone are not enough. For example, say an SLA is 15ms, current 95th-percentile model latency is 10ms, and average model latency is 8ms. If backup requests are configured to send at 10ms, the request will almost certainly timeout. However, if the backup request is sent earlier (say at 2ms), the system's load will effectively be doubled. To solve this, MS router supports backup requests with cross-server cancellation [4]. In this mode, MS will send backup requests earlier. When a server dequeues the request, it will notify the other server to abandon that request. For our scenarios, backup requests at 2ms with cross-server cancellation provides the best latency improvement with the least amount of extra computation. With these optimizations, MS routing overhead is less than 1.5ms. Figure 2 shows the nearly 2x latency drop after cross-server cancellation is enabled for a model.

6 Conclusion

We have presented DLIS. It is serving millions of inference calls per second across tens of thousands of model instances. These models run on varying hardware with low overhead and are supporting many production Microsoft services.

References

[1] Apache hadoop yarn. https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html. 2018.

[2] Docker. https://www.docker.com/. Accessed: 2019-2-12.

[3] Kubernetes. https://kubernetes.io/. Accessed: 2019-2-12.

[4] Jeffrey Dean. Achieving rapid response times in large online services. https://research.google.com/people/jeff/latency.html, 2012. Accessed: 2019-2-12.

[5] Jeremy Fowers, Kalin Ovtcharov, Michael Papamichael, Todd Massengill, Ming Liu, Daniel Lo, Shlomi Alkalay, Michael Haselman, Logan Adams, Mahdi Ghandi, Stephen Heil, Prerak Patel, Adam Sapek, Gabriel Weisz, Lisa Woods, Sitaram Lanka, Steven K. Reinhardt, Adrian M. Caulfield, Eric S. Chung, and Doug Burger. A configurable cloud-scale dnn processor for real-time ai. In *Proceedings of the 45th Annual International Symposium on Computer Architecture*, ISCA '18, pages 1–14, Piscataway, NJ, USA, 2018. IEEE Press.

[6] Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry Heck. Learning deep structured semantic models for web search using clickthrough data. In *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*, pages 2333–2338. ACM, 2013.

[7] Hamid Palangi, Li Deng, Yelong Shen, Jianfeng Gao, Xiaodong He, Jianshu Chen, Xinying Song, and Rabab Ward. Deep sentence embedding using long short-term memory networks: Analysis and application to information retrieval. *IEEE/ACM Transactions on Audio, Speech and Language Processing (TASLP)*, 24(4):694–707, 2016.

[8] Daniel Strigl, Klaus Kofler, and Stefan Podlipnig. Performance and scalability of gpu-based convolutional neural networks. In *Proceedings of the 2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing*, PDP '10, pages 317–324, Washington, DC, USA, 2010. IEEE Computer Society.

[9] Hongfei Zhang, Xia Song, Chenyan Xiong, Corby Rosset, Paul Bennett, Nick Craswell, and Saurabh Tiwary. Generic intent representation in web search. In *submission*, 2019.

[10] Minjia Zhang, Samyam Rajbhandari, Wenhan Wang, and Yuxiong He. Deepcpu: Serving rnn-based deep learning models 10x faster. In *Proceedings of the 2018 USENIX Conference on Usenix Annual Technical Conference*, USENIX ATC '18, pages 951–965, Berkeley, CA, USA, 2018. USENIX Association.