No DNN Left Behind: Improving Inference in the Cloud with Multi-Tenancy

Amit Samanta†, Suhas Shrinivasan†, Antoine Kaufmann, Jonathan Mace
Max Planck Institute for Software System
asamanta,sshriniv,antoinek,jcmace@mpi-sws.org

† student author

Motivation

With the rise of machine learning, inference on deep neural networks (DNNs) has become a core building block on the critical path for many cloud applications. Inference, in contrast to training, is a low-latency online task that makes predictions on-demand using a trained DNN.

In the cloud, inference workloads bear similarity to other online applications like databases, web services, and microservices. DNNs are typically hosted separately from application logic, and accessed via RPC. We identify the following desirable properties for inference workloads:

Latency Inference workloads need stable average and tail latency, regardless of workload volume. Infrequent or sporadic workloads should not suffer from high cold-start latency.

Elasticity Online workloads are inherently unpredictable, and may rapidly transition from low-volume to high-volume. Workload fluctuations should be handled transparently.

Cost-Effectiveness Users should not have to significantly over-provision resources. Moreover, costs should be consistent regardless of the workload volume.

With these desirable properties in mind, we find that existing approaches to deploying DNNs for inference fall significantly short. The conventional approach, based on provisioning containers or virtual machines, suffers from over-provisioning, slow auto-scaling, high cold-start latencies, and mismatched pricing abstractions [1,3,6]. This forces users to compromise on consistent latency, elasticity, or cost-efficiency, depending on workload characteristics.

Multi-Tenancy

In this work, we propose to elevate DNN inference to be a first class cloud primitive provided by a shared multi-tenant system, akin to cloud storage [4] and cloud databases [2]. Multi-tenant systems are only justifiable for *core datacenter functionality*, where there is a common need for the functionality across many different tenants and workloads. We believe that DNN inference is sufficiently important and prevalent to justify a specialized multi-tenant system.

With this approach, users no longer provision per-workload resources. Instead, the cloud provider or system operator provisions resources for the system as a whole, and runs long-lived system processes that receive and execute requests from different tenants concurrently.

By sharing resources, fluctuations in demand can be amortized across tenants, and we avoid over-provisioning and wasting resources. By sharing processes, workload spikes can be absorbed by re-distributing load, and workloads with long periods of inactivity do not suffer from cold-start latency. And lastly, since cloud providers maintain control over the system and its resources, they can more closely align the pricing and system abstractions (i.e., by charging on a per-request basis).

System Overview

Our system architecture bears similarity to many other shared systems, such as filesystems [4] and databases [2]. High-level meta-operations are handled by a logically centralized controller. Model hosting and inference is handled by worker processess, spread across many machines.

The lifecycle from a user's perspective is to (1) upload a trained DNN to the system, then (2) send inference requests. The system performs inference when requests are received, and transparently scales based on the workload demand.

Internally, the system distributes models to one or more workers. Inference requests are routed to whichever workers host the model. Workers host models from many tenants simultaneously, and multiplex execution across different models. If a model is in high demand, it is replicated onto multiple workers, and co-located models may be migrated elsewhere to reduce contention. Replication achieves both fault tolerance and elasticity. For cost-efficiency, we focus particular attention on how workers handle inference requests.

Key Challenges

Security Shared systems execute the requests of different tenants within the same, shared processes. Thus, users are no longer separated by rigid OS or VM boundaries. Nonetheless, we must ensure security between different tenants' workloads. Addressing security entails finding high-level abstractions and interfaces that generalize across many workloads and tenants, but are sufficiently restrictive to achieve security isolation.

Performance Isolation The system must prevent performance interference between different tenants. For example, it shouldn't be possible for one tenant to adversely impact the performance of another tenant's workload, by submitting a high volume of requests. However, shared systems cannot rely on OS mechanisms for isolation between tenants, and instead must address performance and resource management at the application level. Performance isolation has been difficult to achieve for many existing multi-tenant systems, with difficulties arising due to unpredictable workload fluctuations, unpredictable request costs, and strict latency requirements [8]. As such, many existing systems still suffer performance problems due to insufficient resource management [7].

Opportunities

The goal of our system is to achieve the desired latency, elasticity, and cost-efficiency properties, while addressing the security and performance isolation challenges. As mentioned, this has been challenging for many prior systems. However, for DNN inference, we observe several fundamental properties about the workload that we believe makes the challenges far more tractable.

Predictable Structure Our key insight comes from the underlying structure of DNNs: they are an inherently predictable computation, lacking control flow and pointers. It suffices to think of a DNN as a stateless, deterministic, black-box function. For any DNN, we can analytically determine the number of flops required to compute the output from the input. Due to hardware variations, we prefer a more pragmatic approach based on measurement. In our preliminary experiments, we are able to reliably predict inference latencies for a range of different DNNs.

Optimization-Based Scheduling We exploit DNN predictability to do a much better job of request scheduling, both at request admission, and at finer granularity within the system. Predictable latencies enable the system to react to workload fluctuations much more quickly and enable higher quality scheduling decisions. Instead of heuristic-based best-effort scheduling, we can confidently optimize an objective across all pending requests, such as minimizing average execution latency.

Multi-Resource Scheduling Worker processes host many models simultaneously, and alternate service between different tenants. In the worst case, each request may require loading and executing a model that isn't currently loaded. This introduces additional resource costs, such as the need to copy the model from a remote machine or cold storage. Similarly, if we use hardware accelerators, then models need to be copied from host memory to device memory. Overall, the total inference latency depends on a combination of execution latency (CPU, GPU, accelerator) and transfer latency (PCI, disk, network). As mentioned above, execution latency for DNNs is inherently predictable. However, so too is transfer latency, since the memory footprint of a DNN is also fixed and known a priori. This leads to a multi-resource scheduling problem that, while similar to work in other domains [5, 7], has some unique constraints: (i) each resource is individually predictable; (ii) resources are consumed one-at-a-time; and (iii) scheduling decisions can be interposed before each resource. These constraints present an opportunity for high quality, fine-grained scheduling decisions.

Memory Management Not all inference requests incur memory transfer overheads because of caching opportunities at each level. The typical memory footprint of a DNN is in the tens or hundreds of MBs; in contrast, today's web servers often exceed 1TB of main memory, current-generation GPUs have up to 32GB device memory, and current-generation TPUs have 64GB device memory. It is much more efficient to swap between cached models than to reload from scratch each time.

Conclusion

In this work, we will present our (as yet unnamed) system for multi-tenant DNN inference serving. Multi-tenancy enables cost-efficient DNN inference with consistent performance across a range of workloads. Moreover, DNN inference is an ideal candidate for multi-tenancy, because of its predictable resource requirements, which address the challenge of performance isolation.

References

[1] Google Cloud ML Engine Pricing: More about prediction costs. Retrieved January 2019 from https://cloud.google.com/ml-engine/docs/pricing#more_about_prediction_costs.

[2] Fay Chang et al. Bigtable: A distributed storage system for structured data. In *OSDI*, 2006.

[3] Abdul Dakkak et al. TrIMS: Transparent and Isolated Model Sharing for Low Latency Deep LearningInference in Function as a Service Environments. *arXiv preprint arXiv:1811.09732*, 2018.

[4] Sanjay Ghemawat et al. The Google File System. In *SOSP*, 2003.

[5] Ali Ghodsi et al. Multi-Resource Fair Queueing for Packet Processing. In *SIGCOMM*, 2012.

[6] Vatche Ishakian et al. Serving deep learning models in a serverless platform. In *IC2E*, pages 257–262, 2018.

[7] Jonathan Mace, Peter Bodik, Rodrigo Fonseca, and Madanlal Musuvathi. Retro: Targeted Resource Management in Multi-Tenant Distributed Systems. In *NSDI*, 2015.

[8] Jonathan Mace, Peter Bodik, Madanlal Musuvathi, Rodrigo Fonseca, and Krishnan Varadarajan. 2DFQ: Two-Dimensional Fair Queuing for Multi-Tenant Cloud Services. In *SIGCOMM*, 2016.