Letting the Cloud Serve DNN Inferences with Ruthless Efficiency

Reza Karimi
rkarimi@emory.edu
Emory University
Atlanta, GA

Anthony Simpson
asimpson@mpi-sws.org
MPI-SWS
Saarbrücken, Germany

Antoine Kaufmann
antoinek@mpi-sws.org
MPI-SWS
Saarbrücken, Germany

Ymir Vigfusson
ymir@mathcs.emory.edu
Emory University
Atlanta, GA

Jonathan Mace
jcmace@mpi-sws.org
MPI-SWS
Saarbrücken, Germany

INTRODUCTION

The success of deep neural networks (DNNs) in computer vision, natural language processing, speech recognition and other domains has led to the rapid growth of systems and platforms to support deep learning (DL) and enable its wider use. In a cloud and data center settings, DL workloads can broadly be divided into training tasks and inferences. Training is a compute intensive batch task that constructs a DNN using large quantities of data; *training* bears similarity to other batch tasks like data analytics jobs and faces similar challenges. In contrast, *inference* is a low-latency, online task that generates predictions on-demand using a trained DNN; inference bears similarity to online applications like databases, web services, and microservices, and is often just one piece of a broader end-to-end application. DNNs are typically hosted separately from application logic and accessed via remote procedure call (RPC).

The need for speed. Owing to the ecosystem of platforms, libraries, and runtimes used to develop, train, and deploy DNNs, current hosted DNN inference engines, or *model serving* services, have inherited unnecessary bloat. Time-critical inference requests for a model may require a large container or an entire virtual machine (VM) to be spun up, overhead that eclipses the relatively lightweight operation of running an inference on a model loaded into a CPU or GPU. For example, researchers evaluating the feasibility of DNNs in serverless applications measured cold-start times of up to 12 seconds for 100MB models [3]. These overheads translate into latency for end-users and costs for model providers, problems that stymie the growth of hosted machine learning applications.

Our contributions. We set out tackle the challenges of making model inference services fast and cost-effective. We have built Clockwork: a model serving system that provides lightweight DNN inference on its compute resources as a multi-tenant cloud primitive. Instead of depending on bulky associated environments for running each inference, Clockwork treats the DNN models as first-class citizens. Clockwork exploits the observation that DNN inference has predictable duration and deploys a centralized scheduler to provide fast and cost-efficient inference operations with consistent performance. The Clockwork design maintains isolation and provides latency SLOs to operators. The system ensures fair and graceful degradation of inference service across models when load exceeds system capacity, and heralds capacity warnings sooner than in feedback loops driven by SLO violation rates.

MULTI-TENANT MODEL SERVING GOALS

Machine learning models are increasingly being applied to solve problems in interactive settings, and now sit on the critical path of end-user requests. To accommodate such applications, model serving systems have grown while focusing predominantly on maximizing the throughput for individual models (such as DNNs) operating at large scale, for example by leasing a dedicated Google virtual machine with a TPU. Recent research, however, has started to examine model serving where multiple pre-trained models need to be served. In this scenario, multi-model serving is an online task with typically unpredictable workloads that can experience temporary bursts and fluctuations in demand. Multi-model serving strives for several goals:

(G1) To satisfy strict latency SLOs (on the order of milliseconds);
(G2) To maximize throughput by minimizing wasted resources (e.g., by packing models together), and
(G3) To drop requests early and in a fair manner across models when SLOs cannot be met because of limited compute resources.

The first two goals underscore the higher-level objective of making model serving cost-effective. In settings whereby one model does not exhibit enough demand to fully utilize an entire machine, as studied in recent work, the resources can be shared to mitigate costs – scenarios include cloud model hosting, edge model hosting, and on-premise hosting.

CHALLENGES

Yet resource sharing makes goals G1-G3 difficult to meet for several key reasons.

Foregone performance isolation. The system must prevent performance interference between different tenants. However, shared systems cannot rely on OS mechanisms for isolation between tenants, instead must address isolation at application level.

Low-demand workloads. Many workloads are intermittent or sporadic, or never see sufficient demand to warrant dedicated resources and are thus subjected to cold-start set-up times. Existing solutions from cloud providers are insufficient for low-demand workloads, as they suffer from high latency and unacceptable dollar costs, and these workloads have not been studied at adequate depth in recent literature to our knowledge.

Security concerns. Shared systems execute requests of different tenants within the same, shared processes. Thus, users are no longer separated by rigid OS or VM boundaries. Hence, we must ensure

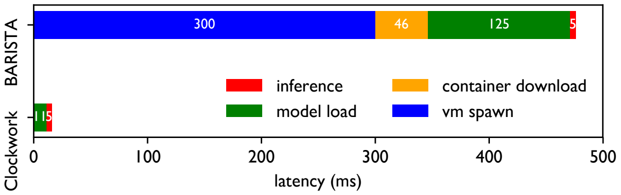

Figure 1: *Breakdown of worst-case model inference times with a cold-start (no model loaded into GPU memory) for Clockwork and numbers reported for BARISTA [2] on the Resnet-50 DNN benchmark.*

security between different tenants’ workloads. Resource sharing further opens the doors for side-channel attacks whose risks must be considered, assessed, and assumed.

OPPORTUNITIES

DNN inference has predictable performance. Unlike workloads from other domains, DNN inference has highly predictable resource consumption patterns, and requests have a predictable duration once scheduled. In other multi-tenant systems, performance isolation is difficult primarily because resource requirements are unpredictable and vary widely from request to request, and once a request is admitted it runs to completion. DNN inference does not face this challenge, because inference is a fundamentally predictable computation. Experiments with our TVM [1] -based runtime show 99th percentile latencies not exceeding 2% of the mean for a range of off-the-shelf DNNs. This determinism stems from the structure of DNNs, they are a fixed sequence of mathematical operations. A priori, we can quantify the exact number of flops required by each layer of the DNN. Moreover, DNNs are predictable as they do not contain control flow elements. DNNs that accept variable-sized or batched inputs also vary deterministically based on input size.

Models likely to be reused can be cached. Once a model is trained, the model weights are immutable and identical between inference requests. Therefore, not all inference requests must incur memory transfer overheads if the DNN weights are cached in main memory. The memory footprint of a DNN is in the tens/hundreds of MBs; in contrast, today’s web servers often exceed 1TB of main memory, present GPUs have up to 32GB device memory, and present TPUs have 64GB device memory. Swapping between cached models is thus more efficient than to reload trained models from scratch.

Inference scheduling can be highly optimized. We can exploit the predictable DNN inference latency together with predictable transfer times between host memory and device memory to vastly improve request scheduling, both at request admission, and at finer granularity within the system. Instead of heuristic-based best-effort scheduling, we can confidently optimize an objective across all pending requests, such as minimizing average execution latency, and to ensure fair resource sharing across tenants.

CLOCKWORK SYSTEM DESIGN

The Clockwork system architecture is similar to existing systems such as shared filesystems and databases. Meta-operations are handled by a logically centralized controller. DNN 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.

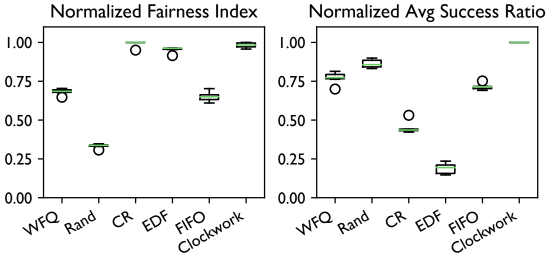

Figure 2: *Fairness and average success rate of different scheduling policies compared to the Clockwork scheduling algorithm.*

At the heart of Clockwork is an optimization algorithm for scheduling requests. Our algorithm expands Fair-EDF [5] to multiple workers, supporting arbitrary-length jobs and considering DNN-model affinity, explicitly trading off the number of completed jobs with *fairness* – the variation in the proportion of jobs completed by each tenant out of those they requested. We also exploit batching, explicitly bundling together inferences for the same model instance and thus avoiding model start-up costs.

EVALUATION RESULTS

Clockwork is fast. To illustrate the potential speed-up from our approach, Figure 1 shows the breakdown of worst-case performance for the state-of-the-art model serving system BARISTA [2] and Clockwork on a standard benchmark (Resnet-50) when serving an inference for a model not yet loaded into memory. Once the model weights have been received by the node, Clockwork loads the model and completes an inference in 17ms, compared to 478ms for BARISTA (28× slower) due to an amalgam of overheads. To mitigate overheads, BARISTA attempts to predict inferences through time-series analysis and proactively spawn up appropriate VMs, containers and models before they are needed. Clockwork, in contrast, curbs overheads and focuses its efforts on scheduling inference tasks in a manner that further reduces the already meager model load-up times.

Clockwork is fair. On a set of different simulated workload scenarios that vary model request rates, client burstiness, SLOs and DNN models, the Clockwork algorithm outperformed other scheduling policies on both Jain’s fairness index [4] and the proportion of jobs that finished prior to their deadline (Figure 2).

CONCLUSION

Our system, Clockwork, provides DNN inference as a cloud primitive by implementing a shared multi-tenant system. Clockwork’s ability to serve models very quickly — our implementation over TVM serves ResNet-50 inference query from cold start in only 17ms — clears obstacles for applying machine learning on the critical path of requests and paves the way for highly efficient model serving.

REFERENCES

[1] TVM: End to End Deep Learning Compiler Stack. https://tvm.ai/, Last accessed on 08-08-2019.

[2] Anirban Bhattacharjee, Ajay Dev Chhokra, Zhuangwei Kang, Hongyang Sun, Aniruddha Gokhale, and Gabor Karsai. Barista: Efficient and scalable serverless serving system for deep learning prediction services. *arXiv preprint arXiv:1904.01576*, 2019.

[3] Vatche Ishakian, Vinod Muthusamy, and Aleksander Slominski. Serving deep learning models in a serverless platform. In *2018 IEEE International Conference on Cloud Engineering (IC2E)*, pages 257–262. IEEE, 2018.

[4] Raj Jain, Arjan Durresi, and Gojko Babic. Throughput fairness index: An explanation. In *ATM Forum contribution*, volume 99, 1999.

[5] Yuhan Peng and Peter Varman. Fair-EDF: a latency fairness framework for shared storage systems. In *11th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 19)*, 2019.