TrIMS: Transparent and Isolated Model Sharing for Low Latency Deep Learning Inference

Abdul Dakkak*, Cheng Li*, Simon Garcia de Gonzalo*, Jinjun Xiong†, Wen-Mei Hwu*
{dakkak, cli99, grcdgnz2, w-hwu}@illinois.edu, jinjun@us.ibm.com
*University of Illinois Urbana-Champaign, †IBM Research Yorktown

Motivation

Deep neural networks (DNNs) have become pervasive within low latency Function as a Service (FaaS) prediction pipelines, but suffers from two major sources of latency overhead: 1) the round-trip network latency between FaaS container and a remote model serving process; 2) Deep Learning (DL) framework runtime instantiation and model loading from storage to CPU or GPU memory. While persistent model serving schemes solve the latter, they do so by eternally persisting models within memory — introducing resource waste and increases cost.

We propose TrIMS, a multi-tier software caching layer on top of FaaS worker machines to solve this problem. TrIMS enables sharing DL models across all levels of the memory hierarchy in the cloud environment — GPU, CPU, local storage, and remote storage — it does so while maintaining the isolation constraints, minimizing model-loading overhead, decreasing end-to-end inference latency, and increasing hardware resource utilization.

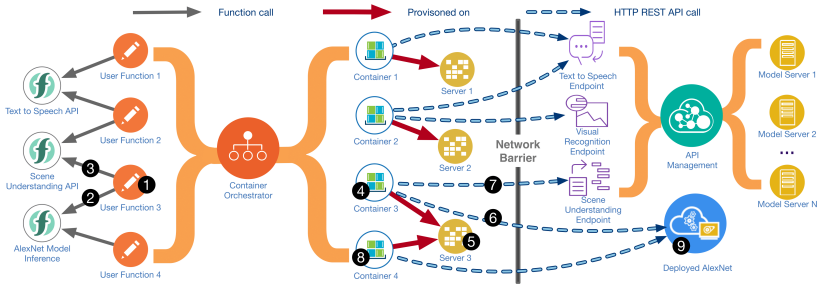

Figure 1: An example of using DL inference in the cloud. (1) application code calls functions from their (2) deployed model or a (3) model catalog. The code is then provisioned onto a (4) container running on (5) server by the cloud provider. The code performed API calls to (6) perform AlexNet inference and (7) the scene understanding API. (9) AlexNet is deployed by users through the cloud provider's cloud deployment mechanism. To avoid the network latency, a common practice is to collocate the model within the deployed functions or the application pipelines. However, collocating the model within the deployed functions requires the model needs to be loaded for the first function invocation, introducing latency overhead

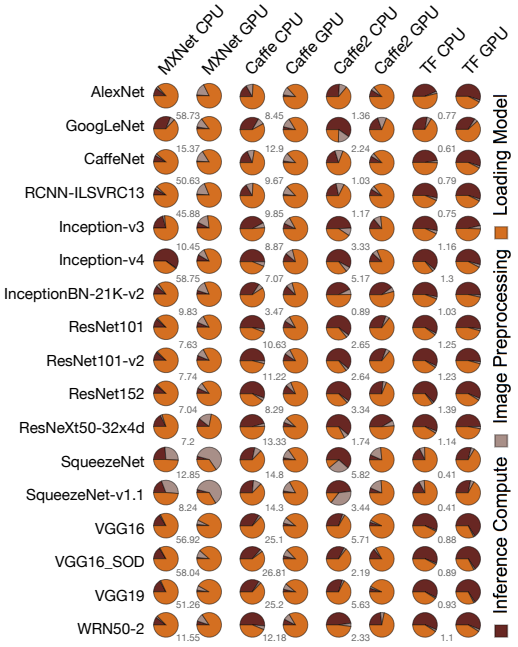

Figure 2: Percentage of time spent in model loading, inference computation, and image preprocessing for "cold start" online DL inference (*batch size* = 1) using CPU and GPU for MXNet, Caffe, Caffe2, and TensorFlow on an IBM S822LC with Pascal GPUs. The speedup of using GPU over CPU for the inference compute alone is shown between the pie charts. Inference time for all frameworks is dominated by model loading except for small models, such as SqueezeNet, where the model size is a few megabytes.

Design

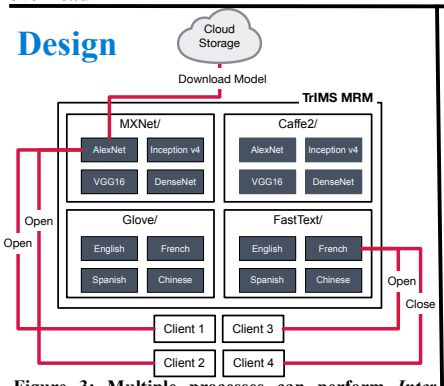

Figure 3: Multiple processes can perform *Inter Process Communication (IPC)* requests to the TrIMS Model Resource Manager (MRM) server; for example Client1, Client2, and Client3 are performing an Open request, while Client4 is performing a Close request. TrIMS's MRM is responsible for loading and managing the placement of the models in GPU memory, CPU memory, or local disk.

Conclusion

TrIMS offers advantages to both cloud providers and users - enabling cloud providers to over-provision hardware resources and decreasing TCO, thus users seeing reducing latency and cost of inference. It is a generic memory sharing technique for applications that

- span multiple processes, maintaining isolation between users
- require large amount of constant data resources to be in situ on the CPU or GPU

Evaluation

Name	CPU	GPU	Memory	GPU Memory	Cached Reads	Buffered Disk Reads
System 1	Intel Core i9-7900X	TITAN Xp P110	32 GB	12 GB	8 GB/sec	193.30 MB/sec
System 2	Intel Xeon E5-2698 v4	Tesla V100-PCIE	256 GB	16 GB	10 GB/sec	421.30 MB/sec
System 3	IBM S822LC Power8 w/ NVLink	Tesla P100-SXM2	512 GB	16 GB	27 GB/sec	521.32 MB/sec

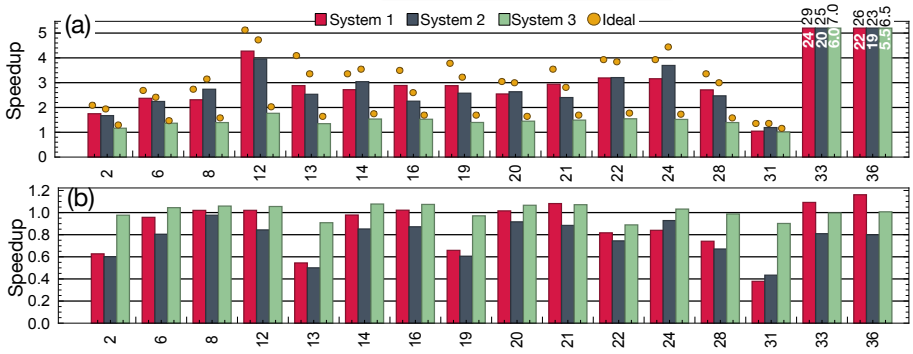

Figure 4: A representative sample of the image classification models are chosen and are run on the above systems to achieve (a) the best case end-to-end time --- when the model has been pre-loaded in GPU memory --- and (b) the worst case end-to-end time --- when the model misses both the CPU and GPU persistence and needs to be loaded from disk. The speedups are normalized to end-to-end running time of the model without TrIMS. The yellow dots show the ideal speedup; the speedup achieved by removing any I/O and data-transfer overhead --- keeping only the framework initialization and compute. For models 33 and 36, the achieved speedup is shown on the bar (white) and the ideal speedup is shown on top of the bar (black).

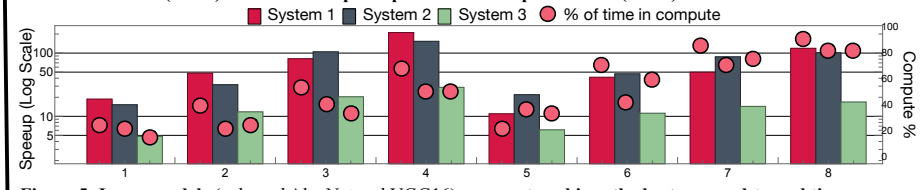

Figure 5: Large models (enlarged AlexNet and VGG16**) are run to achieve the best case end-to-end time --- when the model has been pre-loaded in GPU memory. The speedups are normalized to the end-to-end running time of the model without TrIMS. The red dots show the percentage of time spent performing the compute. We see linear speedup until the inference becomes compute bound.**