Model Registration

User

Register a model

Model Repository

Today’s Inference Serving

Invoke a Model

User(s)

Today’s Inference Serving

Select a model

- ResNet
- DenseNet

Framework

TVM

Optimizer

TensorRT

Hardware architecture

- CPUs
- GPUs
- FPGAs
- ASICs

Users must select the right model-variant
Diverse application requirements

Example: Face Recognition

Object Detection

Face Recognition

Accuracy

Latency

Social Media

Latency

Cost

Navigation for visually impaired person

Applications evolve over time

The performance penalty can be up to 100x
Today’s Inference serving

Invoke a Model

User(s) -> Inference -> Today’s Inference Serving

Model-less Inference serving

Prediction task,
App Reqs
User(s) -> Inference -> Model-less Inference Serving

Prediction task:
“Face Recognition”

App Reqs:
Target accuracy
Target latency

Select a model
∑-ResNet  ∑-DenseNet
∑-SqueezeNet  ∑-Inception

Framework
PyTorch
Caffe2
mxnet

Hardware architecture
CPUs  GPUs  FPGAs  ASICs

Optimizer
TensorRT  TVM

Optimizer
TensorRT  TVM
**Easy-to-use:** Automatically and efficiently select a model and hardware

**Cost Efficient:** Share the hardware as well as models across users
INFaaS provides a model-less API to inference queries that abstracts (a) Model Selection and (b) Resource Provisioning from users.

INFaaS is open-source!

https://stanford-mast.github.io/INFaaS/
Goals & Requirements

Ease-of-use: Automatically select a model and hardware

Challenges

• Novice and expert users
• Diverse user requirements
• Large search space
• Decision overhead
INFaaS overview

**Query:**
- Recognize face in 200ms, >70% accuracy

**Result:**
- "Homer Simpson"

**Register Model**
- ONNX
- .pbtxt

**Front-End**
- Controller
  - Dispatcher
  - Model-Variant Selection Policy
  - VM-Autoscaler
  - Model Registrar

**INFaaS**

**Metadata Store**
- Variant-Profiler
- Variant-Generator

**Model Repository**

**Worker**
- Hardware Executor
- Model-Autoscaler
  - Model-Variant Selection Policy
- Monitoring Daemon

**Controller**
- Dispatcher
  - Model-Variant Selection Policy

"OK"
INFaas overview

Users register models
INFaas overview

Users register models

1. The user submits a query using INFaas' user API
Front-end

- **Goal**: need to map query requirements to models and resources
  - Affects user API, metadata organization, model-variant selection, and autoscaling

- **Challenge**: needs to be intuitive
The *model-less* abstraction

Registered model → bert-pytorch-cpu, resnet50-tf-cpu, resnet50-caffe2-gpu
The *model-less* abstraction

Prediction task

Registered model

- translation
- face-detection

- bert-pytorch-cpu
- resnet50-tf-cpu
- resnet50-caffe2-gpu
The *model-less* abstraction

Prediction task

Registered model

Latency target

Accuracy target

Translation

Face-detection

bert-pytorch-cpu

resnet50-tf-cpu

resnet50-caffe2-gpu
INFaas overview

0. Users register models

1. The user submits a query using INFaas’ user API

2. The Controller selects a model-variant, then selects a worker to process the query
INFaas overview

Users register models

1. The user submits a query using INFaas’ user API

2. The Controller selects a model-variant, then selects a worker to process the query

3. The query proceeds to run on the variant’s target hardware platform
Users register models

The user submits a query using INFaaS’ user API

The Controller selects a model-variant, then selects a worker to process the query

The query proceeds to run on the variant’s target hardware platform

Upon completion, the result is returned to the user
Ease-of-use and cost efficiency

INFaaS removes the system configuration burden and improves ease-of-use
Goals & Requirements

Ease-of-use: Automatically select a model and hardware

Autoscaling: allocate just enough resources, meet SLOs, minimize cost

Challenges

• Novice and expert users
• Diverse user requirements
• Large search space
• Decision overhead

• Query load and pattern changes
• Heterogeneous hardware & models
• Scalability
Existing systems

• **Static provisioning** (TensorFlow Serving, Triton Inference Server)
  - based on peak load
• Meet SLOs but expensive
• Waste resources at low load
Existing systems

- **Static provisioning** (TensorFlow Serving, Triton Inference Server)
  - based on peak load
  - Meet SLOs but expensive
  - Waste resources at low load
- **Replica-only** (Clipper, SageMaker, AI Platform) - replicate individual variants
  - Lower costs but high start-up latency
  - Fail to leverage heterogeneous resources / variants
Autoscaling

3 types of scaling
- Model-horizontal scaling
- Model-vertical scaling -> (Our contribution)
- VM-autoscaling

Division of responsibility
3 types of scaling
- Model-horizontal scaling
- Model-vertical scaling
  
  \(\rightarrow\) (Our contribution)
- VM-autoscaling

Division of responsibility
Model-Autoscaler at each worker

• **Goal:** Decide the *type* and *number* of model-variants to meet the load and requirements, while minimizing cost
• Formulate as an Integer Programming problem

\[
\text{min Cost(action)} = \text{Hardware Cost} + \lambda \text{ Loading Latency}
\]

\{load, unload\} variant instances
Model-Autoscaler at each worker

- **Goal:** Decide the *type* and *number* of model-variants to meet the load and requirements, while minimizing cost
- Formulate as an Integer Programming problem

\[
\text{min Cost(action)} = \text{Hardware Cost} + \lambda \text{Loading Latency}
\]

**Constraints:**

1. With the chosen scaling action, INFaaS supports the incoming query load.
2. The newly-loaded instances satisfy applications’ SLOs.
3. Do not exceed the total system resources.
4. The number of running instances is non-negative.
Model-Autoscaler at each worker

- Respond to changes in load and meet SLOs by: 1) model-horizontal scaling and 2) model-vertical scaling

Worker

Model-Autoscaler

- Load (Reqs/sec)
- SLO violations

Σ-CPU
Model-Autoscaler at each worker

- Respond to changes in load and meet SLOs by: 1) model-horizontal scaling and 2) model-vertical scaling
Model-Autoscaler at each worker

- Respond to changes in load and meet SLOs by: 1) **model-horizontal scaling** and 2) **model-vertical scaling**

**Worker**

- **Model-Autoscaler**
  - Load (Reqs/sec)
  - SLO violations

**Σ-Inferentia batch 1**

**Σ-CPU**

**Model-vertical scaling (upgrading)**
Evaluation

- **Baselines:**
  - **CLIPPER\(^+\)** (Clipper, TIS and TFS): preloaded and persisted beefy variants
  - **CLIPPER\(^+\)\(_{\text{GPU}}\) and CLIPPER\(^+\)\(_{\text{CPU}}\)**
  - **SM\(^+\)** (InferLine, SageMaker, and AI Platform): model-horizontal scaling, replicated *light-weight* variants on/across worker
  - **SM\(^+\)\(_{\text{GPU}}\) and SM\(^+\)\(_{\text{CPU}}\)**
Evaluation

- We deployed INFaaS on AWS EC2
  - GPU worker has 1 NVIDIA V100 GPU
  - Inferentia worker has 1 AWS Inferentia accelerator
  - Controller / CPU worker / client are CPU-only machines
How well does INFaaS scale with load?

INFaaS achieved load while minimizing cost.
How well does INFaaS scale with load?

- INFaaS reduced cost by 3x by leveraging CPU/Inferentia variant;
  - if limited to CPU/GPU variants, still 1.7x cheaper
Putting it all together

• Real workload: Twitter trace (diurnal pattern + spikes)

• Compared to CLIPPER\textsuperscript{+} and SM\textsuperscript{+}:
  • 1.1x, 1.3x higher throughput versus CLIPPER\textsuperscript{+}, SM\textsuperscript{+}
  • 1.6x, 2.5x fewer SLO violations compared to CLIPPER\textsuperscript{+}, SM\textsuperscript{+}
  • 1.23x lower cost by leveraging CPU, GPU, Inferentia machines

INFaaS achieved high performance, better resource utilization, lower SLO violations, and reduced cost
Conclusion

Query:
Recognize face in 200ms, >70% accuracy

Result:
“Homer Simpson”

Register Model

“OK”

https://stanford-mast.github.io/INFaaS/

Contact us:
{faromero,qianl15,neerajay,kozyraki}@stanford.edu