Clipper
A Low-Latency Online Prediction Serving System

Daniel Crankshaw
Stanford NetSeminar
February 1, 2018
TensorFlow: A system for large-scale machine learning

Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Martin J. Barham, Xiaodong Liu, Marc Lelann, Shen Xiaozhou, Jeff Chord, and Xiaoqiang Zheng

Google Brain

Abstract

distributed, datarich training requires a new kind of machine learning system that can scale out to a large cluster of machines and still remains easy to use. TensorFlow satisfies these requirements using a new framework built on top of the distributed computation system developed for MapReduce/D Joe.

Project Adam: Building an Efficient and Scalable Deep Learning Training System

Trishul Chilimbi, Yutaka Sueue, Johnson Apacible, Karthik Kalyanaraman

Microsoft Research

ABSTRACT

Large deep neural networks are fit using stochastic gradient descent methods that minimize a loss function constructed as a sum of losses over training samples. The cost of training these models is dominated by data loading and computational costs. A variety of techniques have been developed to speed up the computation, but the cost of data loading has remained high. Project Adam proposes a new pipeline for deep learning training that can significantly speed up training on large-scale systems.

1. INTRODUCTION

A key problem in modern data analysis is discovery of effective representation for sensory inputs—images, audio, video, text, etc. While performance of conventional, handcrafted features has plateaued in recent years, new developments in deep convolutional architectures have led to performance gains.

A key problem in modern data analysis is discovery of effective representation for sensory inputs—images, audio, video, text, etc. While performance of conventional, handcrafted features has plateaued in recent years, new developments in deep convolutional architectures have led to performance gains.

GraphX: A Graph Processing System for Distributed Systems

Joseph E. Gonzalez, Reynold S. Xin, Anukul Das, Daniel Cranmer, Michael J. Franklin, Ion Stoica

UC Berkeley AirLab

Abstract

GraphX is a high-level API for graph processing in distributed systems. It is based on the concept of vertex programs, which are user-specified functions to be executed on graph vertices. The system provides a virtual machine for executing these programs in parallel over the graph.

GraphLab: A New Framework For Parallel Machine Learning

Yucheng Low, Carnegie Mellon University

Joseph Gonzalez, Carnegie Mellon University

Aapo Klouza, Carnegie Mellon University

Joseph M. Hellerstein, UC Berkeley

Abstract

GraphLab is a new framework for parallel machine learning that is designed to be expressive and efficient. It uses a combination of techniques, including MapReduce, to allow users to express complex algorithms in a high-level language. The framework also provides abstractions for parallelizing both the main algorithm and the data partitions.

Caffe: Convolutional Architecture for Fast Feature Embedding

Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, Trevor Darrell

SUBMITTED TO ACM MULTIMEDIA 2014 OPEN SOURCE SOFTWARE COMPETITION

UC Berkeley EECS, Berkeley, CA 94702

Abstract

Caffe is an open source convolutional architecture for fast feature embedding. It provides a comprehensive framework for defining and training deep learning models, and is designed to be easy to use. The framework supports a wide range of architectures, from simple feedforward nets to complex convolutional nets.
Big Data

Learning

Training

Complex Model
Learning Produces a Trained Model

Query

Model

"CAT"

Decision
Big Data Training Learning

Model

Query Decision

Application
Prediction-Serving for interactive applications

Timescale: ~10s of milliseconds
Prediction-Serving Raises New Challenges
Prediction-Serving Challenges

Support low-latency, high-throughput serving workloads

Large and growing ecosystem of ML models and frameworks
Support low-latency, high-throughput serving workloads

Models getting more complex

- 10s of GFLOPs [1]

Deployed on critical path

- Maintain SLOs under heavy load

Using specialized hardware for predictions

Google Translate

Serving

140 billion words a day\(^1\)

82,000 GPUs running 24/7

Big Companies Build One-Off Systems

**Problems:**

- Expensive to build and maintain
- Highly specialized and require ML and systems expertise
- Tightly-coupled model and application
- Difficult to change or update model
- Only supports single ML framework
Support low-latency, high-throughput serving workloads

Large and growing ecosystem of ML models and frameworks
Large and growing ecosystem of ML models and frameworks

Fraud Detection

Content Rec.

Personal Asst.

Robotic Control

Machine Translation

Netflix

theano

Dato

Create

Caffe

TensorFlow

scikit-learn

mxnet

VW

Spark

KALDI
Large and growing ecosystem of ML models and frameworks

**Difficult to deploy and brittle to manage**

Varying physical resource requirements
But most companies can’t build new serving systems...
Use existing systems: Offline Scoring
Use existing systems: Offline Scoring

Big Data → Training → Model → Scoring → Datastore

Batch Analytics
Use existing systems: Offline Scoring

Look up decision in datastore

Low-Latency Serving
Use existing systems: Offline Scoring

Look up decision in datastore

Problems:

- Requires full set of queries ahead of time
  - Small and bounded input domain
- Wasted computation and space
  - Can render and store unneeded predictions
- Costly to update
  - Re-run batch job
Prediction-Serving Challenges

Support low-latency, high-throughput serving workloads

Large and growing ecosystem of ML models and frameworks
How does Clipper address these challenges?
Clipper Solutions

- Simplifies deployment through layered architecture
- Serves many models across ML frameworks concurrently
- Employs caching, batching, scale-out for high-performance serving
Clipper Decouples Applications and Models

Applications

Predict

RPC/REST Interface

Feedback

Clipper

RPC

Model Container (MC)

RPC

MC

RPC

MC

RPC

MC

RPC

Model Container (MC)

RPC

Caffe

RPC

MC

RPC

MC

RPC

MC
Clipper Architecture

Applications

Predict ↑

RPC/REST Interface

Observe ↑

Clipper

Model Selection Layer

Improve accuracy through bandit methods and ensembles, online learning, and personalization

Model Abstraction Layer

Provide a common interface to models while bounding latency and maximizing throughput.

RPC

Model Container (MC)

RPC

MC

RPC

MC

RPC

MC
Clipper Architecture

Applications

Predict

RPC/REST Interface

Observe

Clipper

Selection Policy

Model Selection Layer

Caching

Model Abstraction Layer

Adaptive Batching

RPC

Model Container (MC)

RPC

MC

Caffe

RPC

MC

scikit-learn
Clipper Implementation

Applications

Predict  \[\uparrow\]

RPC/REST Interface

Observe

Clipper

Core system: 5000 lines of Rust

RPC:

• 100 lines of Python
• 250 lines of Rust
• 200 lines of C++
Provide a common interface to models while...

Model Abstraction Layer

Caching
Adaptive Batching

RPC ↑

Model Container (MC)

RPC ↑

RPC ↑

RPC ↑

Model Abstraction Layer

Caching
Adaptive Batching

RPC ↑

Model Container (MC)

RPC ↑

RPC ↑

RPC ↑

Model Abstraction Layer

Caching
Adaptive Batching

RPC ↑

Model Container (MC)

RPC ↑

RPC ↑

RPC ↑

Model Abstraction Layer

Caching
Adaptive Batching

RPC ↑

Model Container (MC)

RPC ↑

RPC ↑

RPC ↑

Model Abstraction Layer

Caching
Adaptive Batching

RPC ↑

Model Container (MC)

RPC ↑

RPC ↑

RPC ↑

Model Abstraction Layer

Caching
Adaptive Batching

RPC ↑

Model Container (MC)

RPC ↑

RPC ↑

RPC ↑

Model Abstraction Layer

Caching
Adaptive Batching

RPC ↑

Model Container (MC)

RPC ↑

RPC ↑

RPC ↑

Model Abstraction Layer

Caching
Adaptive Batching

RPC ↑

Model Container (MC)

RPC ↑

RPC ↑

RPC ↑

Model Abstraction Layer

Caching
Adaptive Batching

RPC ↑

Model Container (MC)

RPC ↑

RPC ↑

RPC ↑

Model Abstraction Layer

Caching
Adaptive Batching

RPC ↑

Model Container (MC)

RPC ↑

RPC ↑

RPC ↑

Model Abstraction Layer

Caching
Adaptive Batching

RPC ↑

Model Container (MC)

RPC ↑

RPC ↑

RPC ↑

Model Abstraction Layer

Caching
Adaptive Batching

RPC ↑

Model Container (MC)

RPC ↑

RPC ↑

RPC ↑

Model Abstraction Layer

Caching
Adaptive Batching

RPC ↑

Model Container (MC)

RPC ↑

RPC ↑

RPC ↑

Model Abstraction Layer

Caching
Adaptive Batching

RPC ↑

Model Container (MC)

RPC ↑

RPC ↑

RPC ↑

Model Abstraction Layer

Caching
Adaptive Batching

RPC ↑

Model Container (MC)

RPC ↑

RPC ↑

RPC ↑

Model Abstraction Layer

Caching
Adaptive Batching

RPC ↑

Model Container (MC)

RPC ↑

RPC ↑

RPC ↑

Model Abstraction Layer

Caching
Adaptive Batching

RPC ↑

Model Container (MC)

RPC ↑

RPC ↑

RPC ↑

Model Abstraction Layer

Caching
Adaptive Batching

RPC ↑

Model Container (MC)

RPC ↑

RPC ↑

RPC ↑

Model Abstraction Layer

Caching
Adaptive Batching

RPC ↑

Model Container (MC)

RPC ↑

RPC ↑

RPC ↑

Model Abstraction Layer

Caching
Adaptive Batching

RPC ↑

Model Container (MC)

RPC ↑

RPC ↑

RPC ↑

Model Abstraction Layer

Caching
Adaptive Batching

RPC ↑

Model Container (MC)

RPC ↑

RPC ↑

RPC ↑

Model Abstraction Layer

Caching
Adaptive Batching

RPC ↑

Model Container (MC)

RPC ↑

RPC ↑

RPC ↑

Model Abstraction Layer

Caching
Adaptive Batching

RPC ↑

Model Container (MC)

RPC ↑

RPC ↑

RPC ↑

Model Abstraction Layer

Caching
Adaptive Batching

RPC ↑

Model Container (MC)

RPC ↑

RPC ↑

RPC ↑

Model Abstraction Layer

Caching
Adaptive Batching

RPC ↑

Model Container (MC)

RPC ↑

RPC ↑

RPC ↑

Model Abstraction Layer

Caching
Adaptive Batching

RPC ↑

Model Container (MC)

RPC ↑

RPC ↑

RPC ↑

Model Abstraction Layer

Caching
Adaptive Batching

RPC ↑

Model Container (MC)

RPC ↑

RPC ↑

RPC ↑

Model Abstraction Layer

Caching
Adaptive Batching

RPC ↑

Model Container (MC)

RPC ↑

RPC ↑

RPC ↑

Model Abstraction Layer

Caching
Adaptive Batching

RPC ↑

Model Container (MC)

RPC ↑

RPC ↑

RPC ↑

Model Abstraction Layer

Caching
Adaptive Batching

RPC ↑

Model Container (MC)

RPC ↑

RPC ↑

RPC ↑

Model Abstraction Layer

Caching
Adaptive Batching

RPC ↑

Model Container (MC)

RPC ↑

RPC ↑

RPC ↑

Model Abstraction Layer

Caching
Adaptive Batching

RPC ↑

Model Container (MC)

RPC ↑

RPC ↑

RPC ↑

Model Abstraction Layer

Caching
Adaptive Batching

RPC ↑

Model Container (MC)

RPC ↑

RPC ↑

RPC ↑

Model Abstraction Layer

Caching
Adaptive Batching

RPC ↑

Model Container (MC)

RPC ↑

RPC ↑

RPC ↑

Model Abstraction Layer

Caching
Adaptive Batching

RPC ↑

Model Container (MC)

RPC ↑

RPC ↑

RPC ↑

Model Abstraction Layer

Caching
Adaptive Batching

RPC ↑

Model Container (MC)

RPC ↑

RPC ↑

RPC ↑

Model Abstraction Layer

Caching
Adaptive Batching

RPC ↑

Model Container (MC)

RPC ↑

RPC ↑

RPC ↑

Model Abstraction Layer

Caching
Adaptive Batching

RPC ↑

Model Container (MC)

RPC ↑

RPC ↑

RPC ↑

Model Abstraction Layer

Caching
Adaptive Batching

RPC ↑

Model Container (MC)

RPC ↑

RPC ↑

RPC ↑

Model Abstraction Layer

Caching
Adaptive Batching

RPC ↑

Model Container (MC)

RPC ↑

RPC ↑

RPC ↑

Model Abstraction Layer

Caching
Adaptive Batching

RPC ↑

Model Container (MC)

RPC ↑

RPC ↑

RPC ↑

Model Abstraction Layer

Caching
Adaptive Batching

RPC ↑

Model Container (MC)

RPC ↑

RPC ↑

RPC ↑

Model Abstraction Layer

Caching
Adaptive Batching

RPC ↑

Model Container (MC)

RPC ↑

RPC ↑

RPC ↑

Model Abstraction Layer

Caching
Adaptive Batching

RPC ↑

Model Container (MC)

RPC ↑

RPC ↑

RPC ↑

Model Abstraction Layer

Caching
Adaptive Batching

RPC ↑

Model Container (MC)

RPC ↑

RPC ↑

RPC ↑

Model Abstraction Layer

Caching
Adaptive Batching

RPC ↑

Model Container (MC)

RPC ↑

RPC ↑

RPC ↑

Model Abstraction Layer

Caching
Adaptive Batching

RPC ↑

Model Container (MC)

RPC ↑

RPC ↑

RPC ↑

Model Abstraction Layer

Caching
Adaptive Batching

RPC ↑

Model Container (MC)

RPC ↑

RPC ↑

RPC ↑

Model Abstraction Layer

Caching
Adaptive Batching

RPC ↑

Model Container (MC)

RPC ↑

RPC ↑

RPC ↑

Model Abstraction Layer

Caching
Adaptive Batching

RPC ↑

Model Container (MC)

RPC ↑

RPC ↑

RPC ↑

Model Abstraction Layer

Caching
Adaptive Batching

RPC ↑

Model Container (MC)

RPC ↑

RPC ↑

RPC ↑

Model Abstraction Layer

Caching
Adaptive Batching

RPC ↑

Model Container (MC)

RPC ↑

RPC ↑

RPC ↑

Model Abstraction Layer

Caching
Adaptive Batching

RPC ↑

Model Container (MC)

RPC ↑

RPC ↑

RPC ↑

Model Abstraction Layer

Caching
Adaptive Batching

RPC ↑

Model Container (MC)

RPC ↑

RPC ↑

RPC ↑

Model Abstraction Layer

Caching
Adaptive Batching

RPC ↑

Model Container (MC)
Common Interface $\rightarrow$ Simplifies Deployment:

- Evaluate models using original code & systems
Container-based Model Deployment

Implement Model API:

class ModelContainer:
    def __init__(model_data)
    def predict_batch(inputs)
Container-based Model Deployment

Implement Model API:

```python
class ModelContainer:
    def __init__(model_data)
    def predict_batch(inputs)
```

- Implemented in many languages
  - Python
  - Java
  - C/C++
Container-based Model Deployment

*Model implementation packaged in container*

**Model Container (MC)**

```python
class ModelContainer:
    def __init__(model_data)
    def predict_batch(inputs)
```
Container-based Model Deployment

Clipper

RPC

Model Container (MC)

RPC

MC

RPC

MC

RPC

MC

RPC

Model Container (MC)

RPC

Caffe

RPC

MC

RPC

MC

RPC

MC

RPC

Model Container (MC)
Common Interface → Simplifies Deployment:

- Evaluate models using original code & systems
- Models run in separate processes as Docker containers
  - Resource isolation
Common Interface → Simplifies Deployment:

- Evaluate models using original code & systems
- Models run in separate processes as Docker containers
  - Resource isolation
  - Scale-out

Problem: frameworks optimized for **batch processing** not **latency**
**Batching** to Improve Throughput

- **Why batching helps:**
  - A single page load may generate many queries

- **Optimal batch depends on:**
  - hardware configuration
  - model and framework
  - system load

- Hardware Acceleration
- Helps amortize system overhead
Adaptive Batching to Improve Throughput

Why batching helps:

A single page load may generate many queries

Hardware Acceleration

Helps amortize system overhead

Optimal batch depends on:

- hardware configuration
- model and framework
- system load

Clipper Solution:

Adaptively tradeoff latency and throughput…

- Inc. batch size until the latency objective is exceeded (Additive Increase)
- If latency exceeds SLO cut batch size by a fraction (Multiplicative Decrease)
Throughput
(Queries Per Second)

Tensor Flow Conv. Net (GPU)

Batch Size
Tensor Flow Conv. Net (GPU)

Throughput (Queries Per Second)

Latency (ms)

Better

Batch Size

38
Tensor Flow Conv. Net (GPU)

Throughput (Queries Per Second)

Latency (ms)

Optimal Batch Size

Deadline

P99

Batch Size
<table>
<thead>
<tr>
<th>Method</th>
<th>Throughput (QPS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>No-Op</td>
<td>8963</td>
</tr>
<tr>
<td>Random Forest (SKlearn)</td>
<td>317</td>
</tr>
<tr>
<td>Linear SVM (PySpark)</td>
<td>7206</td>
</tr>
<tr>
<td>Linear SVM (SKLearn)</td>
<td>1920</td>
</tr>
<tr>
<td>Kernel SVM (SKLearn)</td>
<td>203</td>
</tr>
<tr>
<td>Log Regression (SKLearn)</td>
<td>1921</td>
</tr>
</tbody>
</table>

Better

No Batching
Throughput (QPS)
**Throughput (QPS)**

**P99 Latency (ms)**

*20 ms is Fast Enough*

---

**Better**

- **Throughput (QPS):** Adaptive vs. No Batching
- **P99 Latency (ms):** No-Op vs. Other Models (Random Forest, Linear SVM, Kernel SVM, Log Regression)
### Throughput (QPS)

<table>
<thead>
<tr>
<th>Batch Size</th>
<th>Adaptive</th>
<th>No Batching</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>48386</td>
<td>8963</td>
</tr>
<tr>
<td>20000</td>
<td>22859</td>
<td>29350</td>
</tr>
<tr>
<td>40000</td>
<td>7206</td>
<td>1920</td>
</tr>
<tr>
<td>60000</td>
<td>48934</td>
<td>197203</td>
</tr>
</tbody>
</table>

### P99 Latency (ms)

<table>
<thead>
<tr>
<th>Batch Size</th>
<th>Adaptive</th>
<th>No Batching</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>20000</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>40000</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>60000</td>
<td>20</td>
<td>20</td>
</tr>
</tbody>
</table>

### Batch Size

<table>
<thead>
<tr>
<th>Batch Size</th>
<th>No-OP</th>
<th>Random Forest (SKlearn)</th>
<th>Linear SVM (PySpark)</th>
<th>Linear SVM (SKLearn)</th>
<th>Kernel SVM (SKLearn)</th>
<th>Log Regression (SKLearn)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1304</td>
<td>490</td>
<td>632</td>
<td>1318</td>
<td>3</td>
<td>1224</td>
</tr>
</tbody>
</table>
Overhead of decoupled architecture

Applications

Predict

RPC/REST Interface

Feedback

Clipper

RPC

MC

Caffe

MC

MC

RPC

RPC

RPC

RPC

MC
Overhead of decoupled architecture

Applications

Predict ↑

RPC/REST Interface

Feedback

Clipper

RPC ↑

RPC ↑

RPC ↑

RPC ↑

MC

MC

MC

MC

Caffe

•••

Applications

Predict ↑

RPC Interface

TensorFlow-Serving

45
Overhead of decoupled architecture
Overhead of decoupled architecture

Model: AlexNet trained on CIFAR-10

Throughput (QPS)

TensorFlow Serving

Clipper

Better

P99 Latency (ms)

TensorFlow Serving

Clipper

Better
Clipper Architecture

Applications

Predict

RPC/REST Interface

Observe

Clipper

Improve accuracy through **bandit methods and ensembles, online learning, and personalization**

Provide a **common interface** to models while **bounding latency** and maximizing throughput.

Model Abstraction Layer

Model Selection Layer

RPC

Model Container (MC)

RPC

MC

RPC

MC

RPC

MC

RPC

MC

RPC

MC
Clipper

Improve accuracy through **bandit methods and ensembles, online learning, and personalization**

Model Selection Layer

**Periodic retraining**

**Experiment with new models and frameworks**
Selection Policy: Estimate confidence

Version 2

Version 3

“CAT”
“CAT”
“CAT”

Policy

“CAT”
CONFIDENT
Selection Policy: Estimate confidence

“CAT”
“MAN”
“CAT”
“SPACE”

Policy

“CAT”
UNSURE
Selection Policy: Estimate confidence

Better Error Rate

ImageNet

<table>
<thead>
<tr>
<th>Policy</th>
<th>Top-5 Error Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>ensemble</td>
<td>0.0586</td>
</tr>
<tr>
<td>4-agree</td>
<td>0.0469</td>
</tr>
<tr>
<td>5-agree</td>
<td>0.0327</td>
</tr>
</tbody>
</table>

Better selection policy: Estimate confidence
Selection Policy: Estimate confidence

Better

Top-5 Error Rate

0.0 0.2 0.4

ensemble

ImageNet

4-agree

5-agree

0.0586

0.0469

0.3182

0.0327

0.1983

width is percentage of query workloads
Selection policies supported by Clipper

- Exploit multiple models to estimate confidence
- Use multi-armed bandit algorithms to learn optimal model-selection online
- Online personalization across ML frameworks

*See paper for details*
Research Prototype to Active Open-Source Project
Status of the project

- Threw away the research prototype and rewrote the system from scratch in C++
- First released in May 2017 with a focus on usability
- Development has focused on performance improvements, support for cloud deployments, and increased support for common machine learning frameworks
  - Supports native deployments on Kubernetes and a local Docker mode
  - First class support for several common machine learning frameworks: PyTorch, TensorFlow, Scikit-Learn, PySpark, R
- 18 contributors
  - Including 7 from outside Berkeley
Project Goals

Community-owned platform for model deployment and serving

Research platform for continued investigation into model serving and management in low-latency settings
Development Roadmap: Spring 2018

- More explicit support for distributed cloud deployments
  - Support for running Clipper in replicated and partitioned modes for high-availability and scalability

- Performance Optimizations
  - A recent research fork introduced several performance optimizations that are in the process of being ported back to the mainline code base

- Metrics and Monitoring infrastructure using Prometheus
  - Comprehensive monitoring is critical for production infrastructure

- Increased community outreach
  - We want community feedback on what is working, and what is not!
Research Roadmap

- Support for increasingly complex machine-learning applications
  - InferLine: Serving multi-model pipelines on heterogeneous hardware
- Serving RL policies and integration with Ray
- Closing the loop between training and serving
  - Online model evaluation and monitoring to detect problems ASAP
  - Automatically decide when and how to retrain models
  - Integration of online learning and bandit algorithms with offline batch training
Serving Multi-Model Pipelines with InferLine
(Work In Progress)
Ensembles can improve accuracy

Faster inference with prediction cascades

If confident then return

Slow but accurate model

Pre-trained DNN

If face detected

Else

Task-specific model

Face detector

If object detected

Object detector

Faster development through model-reuse

Model specialization

How to efficiently support serving arbitrary model pipelines?
Trends in Applied Machine Learning

• Specialized model \(\rightarrow\) composing multiple models to render a single decision
  • More complex applications being made

![Diagram of machine learning models and ensemble](image_url)

- ResNet
- Inception
- Kernel SVM
- Logistic Regression

Ensemble of Neural Features
Model Composition Patterns

• Ensembles
Model Composition Patterns

• Multi-armed bandits
Model Composition Patterns

- Neural Features/Models as transformers

Diagram showing the flow from Model 1 to Model m.
Model Composition Patterns

• Cascades

Ref: Xin Wang et al, IDK Cascades: Fast Deep Learning by Learning not to Overthink
Model Composition Patterns

• E.g. ensemble of Cascades

```
  p_{11}  p_{21}
 /
 Model_{11}  Model_{21}
 /
 p_{12}  p_{22}
 /
 Model_{12}  Model_{22}
```
Model Serving Considerations

![Graph showing throughput and latency as functions of batch size. The graph illustrates how latency increases with batch size, while throughput saturates. The Latency SLO is indicated by a dashed line.](image-url)
## Global Reasoning: Motivating Example

Latency SLO = 300ms

<table>
<thead>
<tr>
<th>Batch size</th>
<th>throughput</th>
<th>Latency(ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>40</td>
<td>80</td>
</tr>
<tr>
<td>5</td>
<td>80</td>
<td>150</td>
</tr>
<tr>
<td>10</td>
<td>100</td>
<td>180</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Batch size</th>
<th>throughput</th>
<th>Latency(ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>100</td>
</tr>
<tr>
<td>5</td>
<td>20</td>
<td>150</td>
</tr>
<tr>
<td>10</td>
<td>30</td>
<td>200</td>
</tr>
</tbody>
</table>
Global Reasoning: Motivating Example

<table>
<thead>
<tr>
<th>Batch size</th>
<th>throughput</th>
<th>Latency(ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>40</td>
<td>80</td>
</tr>
<tr>
<td>5</td>
<td>80</td>
<td>150</td>
</tr>
<tr>
<td>10</td>
<td>100</td>
<td>180</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Batch size</th>
<th>throughput</th>
<th>Latency(ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>100</td>
</tr>
<tr>
<td>5</td>
<td>20</td>
<td>150</td>
</tr>
<tr>
<td>10</td>
<td>30</td>
<td>200</td>
</tr>
</tbody>
</table>

Latency SLO = 300ms
Global Reasoning: Motivating Example

Latency SLO = 300ms

<table>
<thead>
<tr>
<th>Batch size</th>
<th>throughput</th>
<th>Latency(ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>40</td>
<td>80</td>
</tr>
<tr>
<td>5</td>
<td>80</td>
<td>150</td>
</tr>
<tr>
<td>10</td>
<td>100</td>
<td>180</td>
</tr>
</tbody>
</table>

bs₁ = 10  →  throughput₁ = 100
Global Reasoning: Motivating Example

Latency SLO = 300ms

<table>
<thead>
<tr>
<th>Batch size</th>
<th>throughput</th>
<th>Latency(ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>40</td>
<td>80</td>
</tr>
<tr>
<td>5</td>
<td>80</td>
<td>150</td>
</tr>
<tr>
<td><strong>10</strong></td>
<td><strong>100</strong></td>
<td><strong>180</strong></td>
</tr>
</tbody>
</table>

bs₁ = 10 → throughput₁ = 100  
bs₂ = 1 → throughput₂ = 10  
Pipeline throughput = 10
### Global Reasoning: Motivating Example

Latency SLO = 300ms

<table>
<thead>
<tr>
<th>Batch size</th>
<th>throughput</th>
<th>Latency(ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>40</td>
<td>80</td>
</tr>
<tr>
<td>5</td>
<td>80</td>
<td>150</td>
</tr>
<tr>
<td>10</td>
<td>100</td>
<td>180</td>
</tr>
</tbody>
</table>

bs₁ = 10 → throughput₁ = 100
bs₂ = 1 → throughput₂ = 10
Pipeline throughput = 10

<table>
<thead>
<tr>
<th>Batch size</th>
<th>throughput</th>
<th>Latency(ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>100</td>
</tr>
<tr>
<td>5</td>
<td>20</td>
<td>150</td>
</tr>
<tr>
<td>10</td>
<td>30</td>
<td>200</td>
</tr>
</tbody>
</table>

bs₂ = 10 → throughput₂ = 30
bs₁ = 1 → throughput₁ = 40
Pipeline throughput = 30
Model Performance/Cost

<table>
<thead>
<tr>
<th>GPU</th>
<th>Cost ($/hr)</th>
<th>Cloud</th>
</tr>
</thead>
<tbody>
<tr>
<td>K80</td>
<td>0.90</td>
<td>AWS</td>
</tr>
<tr>
<td>P100</td>
<td>1.46</td>
<td>GCP</td>
</tr>
<tr>
<td>V100</td>
<td>3.06</td>
<td>AWS</td>
</tr>
</tbody>
</table>

Resnet 152

![Graph showing throughput against batch size for Resnet 152 on K80 (2014)]
Model Performance/Cost

<table>
<thead>
<tr>
<th>GPU</th>
<th>Cost ($/hr)</th>
<th>Cloud</th>
</tr>
</thead>
<tbody>
<tr>
<td>K80</td>
<td>0.90</td>
<td>AWS</td>
</tr>
<tr>
<td>P100</td>
<td>1.46</td>
<td>GCP</td>
</tr>
<tr>
<td>V100</td>
<td>3.06</td>
<td>AWS</td>
</tr>
</tbody>
</table>

Resnet 152

Throughput (qps) vs Batch size
Model Performance/Cost

<table>
<thead>
<tr>
<th>GPU</th>
<th>Cost ($/hr)</th>
<th>Cloud</th>
</tr>
</thead>
<tbody>
<tr>
<td>K80</td>
<td>0.90</td>
<td>AWS</td>
</tr>
<tr>
<td>P100</td>
<td>1.46</td>
<td>GCP</td>
</tr>
<tr>
<td>V100</td>
<td>3.06</td>
<td>AWS</td>
</tr>
</tbody>
</table>

Graph showing throughput vs. batch size for ResNet 152 on different GPUs and clouds.
Model Configuration: Control Parameters

- Batch size
  - Exploiting internal parallelism of the model

- Physical resource allocation
  - GPU allocation (Usually decision is either 0 or 1)
  - CPU allocation

- Replication factor
  - Exploiting external parallelism
Problem Statement

- Per-model configuration requires global reasoning
  - Each pipeline component contributes latency
  - Each model affects pipeline balance \rightarrow\ pipeline throughput
- Combinatorial number of pipeline configurations
  - per-model parameters: batch size, replication factor
  - In the number of model types
  - In the number of resource types
- \textbf{How do we capture and provision a well-balanced cost-efficient inference pipeline?}
System Goals

- **Well-balanced** inference pipeline
  - For heterogeneous models
  - On heterogeneous hardware
- **Cost-efficient** pipeline provisioning
  - Minimize cost to achieve desired latency and throughput SLO
- **Expressive/general** for arbitrary pipelines & constraints:
  - Conditional/probabilistic, bandits, cascades
  - Application throughput targets, cost budget, latency SLO
InferLine Architecture

- Driver.py
- Sample queries
- Constraints
  - SLO
  - Throughput
  - Cost

Per model conf
- Batch size
- #replicas
- Resource bundle

Driver.py
Sample queries
Constraints
  - SLO
  - Throughput
  - Cost

Profiler
Optimizer
Clipper

m1 m2
m3 m4

GPU

Per model profiles
pipeline lineage
PICL
QP
PICL Compiler

GPU
InferLine Architecture

- Driver.py
- Sample queries
- Constraints
  - SLO
  - Throughput
  - Cost

- Profile pipeline
- Profile models
- Sample queries per model
- Pipeline lineage per model profiles
- PICL expressions
- PICL Compiler
- QP
- Clipper
- GPU

Per model conf:
- Batch size
- #replicas
- Resource bundle

82
InferLine Architecture

- Driver.py
- Sample queries
- Constraints
  - SLO
  - Throughput
  - Cost

Per model conf
- Batch size
- #replicas
- Resource bundle
Conclusion

- Challenges of serving increasingly complex models trained in variety of frameworks while meeting strict performance demands

- Clipper adopts a container-based architecture and employs prediction caching and latency-aware batching

- Ongoing efforts on a workload-aware optimizer to optimize the deployment of complex, multi-model pipelines

 crankshaw@berkeley.edu

http://clipper.ai

https://github.com/ucbrise/clipper
GPU Cluster Scaling

![Graph showing throughput and latency vs. number of replicas for different network speeds and configurations.](image-url)