**Timescale:** minutes to days

**Systems:** offline and batch optimized

*Heavily studied ... major focus of the AMPLab*
Big Data → Training → Big Model

→ Query → Decision → Application

Learning → Inference
Big Data Training

Learning

Big Model

Inference

Query

Decision

Application

Timescale: ~20 milliseconds

Systems: online and latency optimized

Less studied …
Big Data

Learning

Training

Big Model

Feedback

Inference

Query

Decision

Application
Big Data Training Application

Learning

Inference

Timescale: hours to weeks

Systems: combination of systems

Less studied ...

Application

Feedback
Big Data

Big Model

Training

Application

Decision

Query

Learning

Inference

Adaptive
(~1 seconds)

Responsive
(~10ms)

Feedback

Responsive (~10ms)

Adaptive (~1 seconds)
Prediction Serving Challenges

- Complexity of deploying new models
  - New applications or products \((0 \rightarrow 1 \text{ models})\).
  - New data, features, model family: \((N \rightarrow N+1 \text{ models})\).
  - Why is it hard: Frameworks not designed for low-latency serving, frameworks have different APIs, different resource requirements, and different costs.

- System Performance
  - Need to ensure low-latency predictions, scalable throughput. Deploying a new model can’t degrade system performance.

- Model or Statistical Performance
  - Model Selection: Which models to use?
  - When to deploy a new model?
  - How to adapt to feedback?
  - At a meta-level: what are the right metrics for measuring model performance?
LASER: A Scalable Response Prediction Platform for Online Advertising

Agarwal et al. 2014
LASER Overview

- Top-down system design enforced by company organizational structure
- Picked a model (logistic regression) and built the system based on that choice
- Force data-scientists to use this model, express features in specialized configuration language
- Result: *System and model family are tightly coupled*

\[ p_{ijt} = \frac{1}{1 + \exp(-s_{ijt})} \]

\[ s_{ijt} = \omega + s_{ijt}^{1,c} + s_{ijt}^{2,c} + s_{ijt}^{2,\omega} \]
Addressing Deployment Complexity

- **Fixed Model Choice:** Can be hardcoded into system, no need for API to specify model
- **Configuration language:** specify feature construction in JSON-based configuration language
  - Restricts feature transformations to be built from component library
  - Allows for changes in pipeline without service restarts or code modification
  - Allows easy re-use of common features across an organization
  - Similar to PMML, PFA
- **Language details**
  - **Source:** translate data to numeric feature vectors
  - **Transformer:** Vector-to-vector transformations (transform, aggregate)
  - **Assembler:** Concatenates all feature pipelines together into single vector
Addressing System Performance

- **Precompute second-order interaction terms**
  - The LASER logistic regression model includes second order interaction terms between user and campaign features:
    \[ s_{ij,t}^{2,c} = x'_i A c_j + \ldots \]

- **Don’t wait for delayed features**
  - Features can be delayed by slow DB lookup, expensive computation
  - **Solution:** Substitute expected value for missing features and degrade accuracy, not latency
  - **Solution:** Cache precomputed scalar products in PRC, save overhead of re-computing features and dot products which are lazily evaluated
Addressing Model Performance

- Decompose model into slowly-changing and quickly-changing components
  - Fast retraining of warm-start (quickly-changing) component of model without cost of full retraining

\[ s_{ijt} = \omega + s_{ijt}^{1,c} + s_{ijt}^{2,c} + s_{ijt}^{2,\omega} \]

- Explore/Exploit with Thompson Sampling
  - Sometimes serve ads with low empirical mean but high-variance
  - Draw sample from posterior distribution over parameters and use sample to predict CTR instead of mode
  - In practice, hold \( \Theta_c \) fixed and sample from \( \Theta_w \)
Some Takeaways from LASER

- System performance is paramount in the broader application context
  - Slow page load has much larger impact on revenue than poor ad-recommendation
- AUC/accuracy is not always the most useful model performance metric
- The more assumptions you can make about your tools (software, models) the more tricks you can play (config language, shared features, warm-start/cold-start decomposition)
  - Safe for LASER to make these assumptions because they are enforced through extra-technological methods
  - Similar to some of the design choices we saw in Borg last week
Clipper
A Low-Latency Online Prediction Serving System

Daniel Crankshaw,
Xin Wang
Giulio Zhou
Michael Franklin,
Joseph E. Gonzalez
Ion Stoica
Goals of Clipper

- **Design Choice:** *General purpose, easy to use* prediction serving system
  - Generalize to many *different ML applications* (contrast to LASER which was designed to address LinkedIn’s ad-targeting needs)
  - Generalize to *many frameworks/tools* for a single application
    - Don’t tie the hands of data scientists developing models
  - Make it simple for a *data-scientist* to deploy a new model into production
- Given these design choices, maximize system and model performance using *model-agnostic* techniques
Clipper **Generalizes** Models Across ML Frameworks

- Fraud Detection
- Content Rec.
- Personal Asst.
- Robotic Control
- Machine Translation

Clipper

- theano
- Dato
- Caffe
- TensorFlow
- scikit-learn
- KeystoneML
- Create
- VW
- mxnet
- KALDI
Clipper Architecture

Applications

Predict ↑

RPC/REST Interface

Observe ↓

Clipper

theano

Dato

Keystone

Caffe

TensorFlow

dmlc

mxnet

scikit

learn

VW

Create

KALDI
Clipper Architecture

Predict ⬆️

RPC/REST Interface

Observe ⬇️

Clipper

RPC ⬆️

Model Wrapper (MW)

KeystoneML

RPC ⬆️

MW

Caffe

RPC ⬆️

MW

RPC ⬆️

MW

RPC ⬆️

MW

...
Clipper Architecture

Applications

Predict → RPC/REST Interface → Observe

Clipper

Model Selection Layer

Improve accuracy through ensembles, online learning and personalization

Model Abstraction Layer

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

RPC → Model Wrapper (MW) → Keystne

RPC → MW → Caffe

RPC → MW → TF

RPC → MW → scikit-learn
Clipper Architecture

Applications

Predict

RPC/REST Interface

Observe

Clipper

Selection Policy

Model Selection Layer

Caching

Model Abstraction Layer

Adaptive Batching

RPC

Model Wrapper (MW)

Keystone

RPC

MW

Caffe

RPC

MW

scikit-learn
<table>
<thead>
<tr>
<th>Model Wrapper (MW)</th>
<th>Caching</th>
<th>Adaptive Batching</th>
</tr>
</thead>
<tbody>
<tr>
<td>KeystoneML</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Caffe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TensorFlow</td>
<td></td>
<td></td>
</tr>
<tr>
<td>scikit-learn</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Provide a common interface to models while...
Common Interface → Simplifies Deployment:

- Evaluate models using original code & systems
- Models run in separate processes (Docker containers)
  - Resource isolation
Common Interface $\Rightarrow$ Simplifies Deployment:
- Evaluate models using original code & systems
- Models run in separate processes
  - Resource isolation
  - Scale-out

**Problem:** frameworks optimized for **batch processing** not **latency**
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:
be as slow as allowed…

- Inc. batch size until the latency objective is exceeded (Additive Increase)
- If latency exceeds SLO cut batch size by a fraction (Multiplicative Decrease)
Adaptive Batching to Improve Throughput

25.5x throughput increase
Clipper Architecture

Applications

Predict

RPC/REST Interface

Observe

Clipper

Selection Policy

Model Selection Layer

Caching

Model Abstraction Layer

Adaptive Batching

RPC

Model Wrapper (MW)

Keystone

RPC

MW

Caffe

RPC

MW

RPC

MW

RPC

MW

RPC

MW
Goal:

Maximize accuracy through bandits and ensembles, online learning, and personalization.

Incorporate feedback in real-time to achieve:

- **robust predictions** by combining multiple models & frameworks
- **online learning** and **personalization** by selecting and personalizing predictions in response to feedback
Model Selection Policy

Improves prediction **accuracy** by:
- Incorporates real-time **feedback**
- Estimates **confidence** of predictions
- Determines how to combine multiple **predictions**
  - e.g., choose best, average, ...
  - enables frameworks to **compete**
Cost of Ensembles

Increased Load

- **Solutions:**
  - **Caching** and **Batching**
  - **Model Selection** prioritizes frameworks for load-shedding

Stragglers

- e.g., framework fails to meet SLO
- **Solution:** **Anytime** predictions
  - Selection policy must select/combine from available predictions
  - e.g., built-in ensemble policy substitutes expected value
Limitations of Clipper

- Clipper does not address offline model retraining

- By treating deployed models as black boxes, Clipper forgoes the opportunity to optimize prediction execution of the models themselves or share computation between models

- Only performs coarse-grained tradeoffs of accuracy, robustness, and performance.
TensorFlow Serving

- Recently released open-source prediction-serving system from Google
- Companion to TensorFlow deep-learning ML framework
- Easy to deploy *TensorFlow Models*
- System automatically manages the lifetime of deployed models
  - Watches for new versions, loads and transfers requests to new models automatically
- System does not address model performance, only system performance (through batching)
TensorFlow Serving Architecture

Applications

Predict

RPC/REST Interface

TensorFlow-Serving

Prediction Batching

New model version trained

V1

V2

V3

V2

RETIRED
TensorFlow Serving Architecture

Applications

Predict ↑  ↓  RPC/REST Interface

TensorFlow-Serving

Prediction Batching

V1  V2  V3

RETIRED

TensorFlow  TensorFlow  TensorFlow
Other Prediction-Serving Systems

- **Turi**
  - Company co-founded by Joey, Carlos Guestrin, and others to serve predictions from models (primarily) trained in the GraphLab Create framework
  - Not open-source
  - Recently acquired by Apple

- **Oryx**
  - Developed by Cloudera for serving Apache Spark Models
  - Implementation of Lambda Architecture with Spark and Spark Streaming to incrementally maintain models
  - Open source

- **PredictionIO**
  - Open-source Apache Incubating project, the company behind the project was recently acquired by Salesforce
  - Built on Apache Spark, Hbase, Spray, ElasticSearch