Distributed Machine Learning with a Serverless Architecture

Hao Wang¹, Di Niu², Baochun Li¹
¹University of Toronto, ²University of Alberta
What is machine learning?
Deep Learning
Machine Learning

Numerical optimization → Gradients

- Input: $x_1, x_2, x_3, \ldots, x_M$
- Hidden Layer: $z_1, z_2, \ldots, z_D$
- Output: $y$
ML Workflow

Objective
Budget

Resource
Reservation

Objective
Data

Model
Design

Datasets
...

Model
Tuning

Convergence
Loss rate
...

Training &
Evaluation

Resource
Reservation

Model
Design

Datasets
...

Model
Tuning

Convergence
Loss rate
...

Training &
Evaluation
Our Key Insights

• Most current ML training jobs are data parallel

• Model quality and resource investment have a nonlinear relation

• ML training is inevitably a trial-and-error process
# Distributed ML Infrastructure

<table>
<thead>
<tr>
<th></th>
<th>IaaS</th>
<th>PaaS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Pricing</strong></td>
<td>Per hour</td>
<td>Per hour</td>
</tr>
<tr>
<td><strong>Maintanance</strong></td>
<td>By users</td>
<td>By providers</td>
</tr>
<tr>
<td><strong>Examples</strong></td>
<td>AWS EC2, Google Cloud Compute ...</td>
<td>Azure ML Studio, Google Cloud ML Engine ...</td>
</tr>
</tbody>
</table>

Diagram showing cloud services from Google Cloud, Amazon Web Services, and Microsoft Azure.
<table>
<thead>
<tr>
<th></th>
<th>IaaS</th>
<th>PaaS</th>
<th>Serverless</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Pricing</strong></td>
<td>Per hour</td>
<td>Per hour</td>
<td>Per call</td>
</tr>
<tr>
<td><strong>Maintainance</strong></td>
<td>By users</td>
<td>By providers</td>
<td>By providers</td>
</tr>
<tr>
<td><strong>Examples</strong></td>
<td>AWS EC2</td>
<td>Azure ML Studio</td>
<td>AWS Lambda</td>
</tr>
</tbody>
</table>
Serverless Computing?

- Only input and output, no intermediate states
Go Serverless?

Pro:
1. Flexible concurrency
2. Instant response
3. Easy to deploy

Con:
1. Execution model is too simple
2. Runtime limitations (~15min)
3. Communication overhead
ML Training on Serverless?

- MapReduce on Serverless Cloud (PyWren, [SoCC’17])
- Video processing on Serverless Cloud (Sprocket [SoCC’18])
Stochastic Gradient Descent (SGD)

\[ \theta_j = \theta_j + \alpha (y^i - h_\theta(x^i)) x^i_j \]
Mini-batch SGD

\[ \theta_j = \theta_j + \frac{a}{b} \sum_{k=i}^{i+b-1} (y^k - h_\theta(x^k))x^k_j \]
Parameter Server

- Model replicas on workers
- Servers update parameters

Li, Mu, et al. Scaling distributed machine learning with the parameter server. OSDI'14
\[ \theta_j = \theta_j + \frac{a}{b} \sum_{k=i}^{i+b-1} (y^k - h_{\theta}(x^k))x^k_j \]
ML Training on Lambda

Input Samples

\[
\begin{align*}
x_1 & \quad \text{Layer } L_1 \\
x_2 & \quad \text{Layer } L_2 \\
x_3 & \quad \text{Layer } L_3 \\
1 & \quad \text{Layer } L_4
\end{align*}
\]

\[ h_{W,b}(x) \]

Func.

Func.

Func.

Func.

KV Storage

Amazon S3

or

redis
Toy Example

• **Workload**
  - A logistic regression model

• **AWS Lambda**
  - 20 functions
  - 150 functions
  - X functions (dynamic # of func.)
  - S3 storage

• **EC2 c5.2xlarge**
  - 8 CPUs, 16GB mem
  - Local storage
Toy Example

- Loss value v.s. training time
- Loss value v.s. monetary cost

![Graph showing loss value vs. time and cost for different scenarios](image-url)
## Toy Example

<table>
<thead>
<tr>
<th>X functions</th>
<th>Loss Value</th>
<th>Time (s)</th>
<th>Cost ($)</th>
</tr>
</thead>
<tbody>
<tr>
<td>20 functions</td>
<td>0.009725</td>
<td>237.40</td>
<td>0.019</td>
</tr>
<tr>
<td>8-core EC2</td>
<td>0.009779</td>
<td>307.87</td>
<td>0.029</td>
</tr>
<tr>
<td>150 functions</td>
<td>0.009699</td>
<td>50.04</td>
<td>0.031</td>
</tr>
<tr>
<td>X functions</td>
<td>0.009761</td>
<td>202.55</td>
<td>0.019</td>
</tr>
</tbody>
</table>

- Slowest, no cheap
- Fastest, expensive
- Fast, cheap

- The first epoch: 120 functions
- The last epoch: 10 functions
- Intermediate epochs: 20 functions
Challenges

• Functions on Serverless
  - Limitation on performance and deployment

• Dynamic Resource Provisioning
  - Speed v.s. cost (given a budget, how fast could be?)
Siren

- Hybrid Synchronous Parallel (HSP)
- Experience-Driven Resource Scheduler
Architecture

- User-Defined Model
- API Libs

Stateless Functions

- code
- package

Cloud

- resource scheme
- function status

Local Client

- action
- states

Scheduler

DRL Agent

Function Manager
Enforce Parallelism on Siren
Synchronous or Asynchronous

Synchronous training

Asynchronous training
Hybrid Synchronous Parallel (HSP)

- Fetching input
- Computing
- Updating parameters

$f_{t,i}$, mini-batch

$\rightarrow$ time

epoch $t$ $\rightarrow$ epoch $t+1$
Experience-Driven Scheduler
## Toy Example - Find the X

<table>
<thead>
<tr>
<th></th>
<th>Loss Value</th>
<th>Time (s)</th>
<th>Cost ($)</th>
</tr>
</thead>
<tbody>
<tr>
<td>20 functions</td>
<td>0.009725</td>
<td>237.40</td>
<td>0.019</td>
</tr>
<tr>
<td>8-core EC2</td>
<td>0.009779</td>
<td>307.87</td>
<td>0.029</td>
</tr>
<tr>
<td>150 functions</td>
<td>0.009699</td>
<td>50.04</td>
<td>0.031</td>
</tr>
<tr>
<td>X functions</td>
<td>0.009761</td>
<td>202.55</td>
<td>0.019</td>
</tr>
</tbody>
</table>
Deep Reinforcement Learning

Agent

- Features
- Policy parameters $\theta$

Environment

- Stateless Functions
- Reward $r_t$
- Policy $\pi(a_t|s_{t-1}, \theta)$
- Action $a_t$

State $s_{t-1}$
## State

\[ s_t = (t, \ell_t, P_t, P_{t,F}, P_{t,C}, P_{t,U}, u_t, w_t, b_t) \]

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t )</td>
<td>the epoch index of the training workload</td>
</tr>
<tr>
<td>( P_{t,F} )</td>
<td>the time period for function ( f_{t,i} ) fetching input data</td>
</tr>
<tr>
<td>( P_{t,C} )</td>
<td>the time period for function ( f_{t,i} ) computing gradients</td>
</tr>
<tr>
<td>( P_{t,U} )</td>
<td>the time period for function ( f_{t,i} ) updating parameters</td>
</tr>
<tr>
<td>( P_t )</td>
<td>the whole time period of the epoch ( t )</td>
</tr>
<tr>
<td>( \ell_t )</td>
<td>the loss value achieved at the end of the epoch ( t )</td>
</tr>
<tr>
<td>( b_t )</td>
<td>the remaining budget at epoch ( t )</td>
</tr>
<tr>
<td>( u_t )</td>
<td>the average memory utilization observed in epoch ( t )</td>
</tr>
<tr>
<td>( w_t )</td>
<td>the average CPU utilization observed in epoch ( t )</td>
</tr>
</tbody>
</table>
**Action**

| \( n_t \) | the number of concurrent functions in epoch \( t \) |
| \( m_t \) | the memory size of each function in epoch \( t \) |

\[
\mathbf{a}_t = (n_t, m_t) \quad n_t, m_t \in \mathbb{Z}^+
\]

\( n_t \times m_t \) choices \( \sim 138,000 \) actions on AWS

Approximating with Gaussian distribution

\[
\pi(\mathbf{a} | \mathbf{s}_{t-1}, \theta) = \frac{1}{\sigma(\mathbf{s}, \theta)\sqrt{2\pi}} \exp\left(-\frac{(\mathbf{a} - \mu(\mathbf{s}, \theta))^2}{2\sigma(\mathbf{s}, \theta)^2}\right)
\]

Policy

\[\pi(a_t | s_{t-1}, \theta)\]
At each epoch \( t \),

\[
\begin{align*}
    r_t &= -\beta P_t, \quad t = 1, \ldots, T - 1
\end{align*}
\]

At the final epoch \( T \),

\[
    r_T = \begin{cases} 
    -\beta P_T + C & \text{if } \ell_T \leq \mathcal{L} \text{ and } b_T \geq 0, \\
    -\beta P_T - C & \text{otherwise.}
    \end{cases}
\]

A constant as the final reward/penalty

Reach the expected loss value, or use up all budget
Maximize cumulative discounted reward:

\[
\max \sum_{t=1}^{T} \gamma^t r_t, \quad \gamma \in (0, 1]
\]

Policy gradient:

\[
\nabla_{\theta} \mathbb{E}_\pi \left[ \sum_{t=1}^{T} \gamma^t r_t \right] = \mathbb{E}_\pi [ \nabla_{\theta} \ln \pi(a | s, \theta) q_\pi(s, a)]
\]
DRL

Agent

Features

Policy parameters $\theta$

State $s_{t-1}$

Reward $r_t$

Policy

$\pi(a_t|s_{t-1}, \theta)$

Action $a_t$

Environment

Stateless Functions
Cloud Workflow

User-Defined Model

API Libs

code package

Stateless Functions

Cloud

Step 1

resource scheme

Step 2

Stateless Functions

function status

Step 3

Step 4

Action

Scheduler

DRL Agent

Function Manager

Step 5

Local Client

states
Evaluation

• Simulation: OpenAI Gym

• Testbed: AWS Lambda + AWS EC2

• 44.3% \downarrow on job completion time
Simulation - overview

- **Workload**: mini-batched SGD algorithms
- **Goal**: DRL agent v.s. Grid search (# of functions)
Simulation - grid search

- GS-300
- GS-200
- GS-100
- GS-50

Total rewards vs Number of functions:

- GS-300
- GS-200
- GS-100
- GS-50

Time (s) vs Budget ($):

- GS: 10.03%
- Siren: 12.87%
- GS: 36%
# Simulation

<table>
<thead>
<tr>
<th>Method</th>
<th>Function #</th>
<th>Cost ($)</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grid Search</td>
<td>828</td>
<td>299.89</td>
<td>2452.3</td>
</tr>
<tr>
<td>SIREN</td>
<td>652 – 892</td>
<td>299.92</td>
<td>1569.5</td>
</tr>
<tr>
<td>Grid Search</td>
<td>482</td>
<td>199.67</td>
<td>2816.9</td>
</tr>
<tr>
<td>SIREN</td>
<td>355 – 597</td>
<td>199.73</td>
<td>2454.4</td>
</tr>
<tr>
<td>Grid Search</td>
<td>138</td>
<td>99.99</td>
<td>4979.7</td>
</tr>
<tr>
<td>SIREN</td>
<td>56 – 258</td>
<td>99.82</td>
<td>4480.4</td>
</tr>
<tr>
<td>Grid Search</td>
<td>3000</td>
<td>47.76</td>
<td>Fail</td>
</tr>
<tr>
<td>SIREN</td>
<td>1293 – 2995</td>
<td>49.82</td>
<td>Fail</td>
</tr>
</tbody>
</table>
Simulation - DRL training

Number of functions

Epoch of ML training

Total rewards

Iteration

Siren-300
Siren-200
Siren-100
Testbed

• Siren on AWS Lambda v.s. MXNet on EC2
  - m4.large: 2 vCPU, 8GB memory, $0.1/hr
  - m4.xlarge: 4 vCPU, 16GB memory, $0.2/hr
  - m4.2xlarge: 8 vCPU, 32GB memory, $0.4/hr

• Workload
  - LeNet on MNIST
  - CNN on movie review
  - Linear Classification on click-through prediction dataset
Testbed - Siren and EC2 on LeNet

- **Cost ($)**
  - m4.large: 0.02
  - m4.xlarge: 0.04
  - m4.2xlarge: 0.06

- **Time (s)**
  - 50
  - 100
  - 150
  - 200
  - 250

- **Number of EC2 instances**
  - 2
  - 4
  - 6
  - 8
  - Siren

The diagram shows the comparison of time and cost for different instances and Siren.
Testbed - DRL training

- # of functions: 500, 1000
- Memory (MB): 200, 400, 600, 800
- Training epoch of LeNet: 2, 4, 6, 8
- Total rewards: -100, 0, 100
- Iteration: 0, 100, 200, 300
Testbed - time v.s. cost

![Graph showing the relationship between time (s) and cost ($) for EC2 and Siren. The graph plots time (s) on the y-axis and cost ($) on the x-axis. The data points indicate a general trend where increased cost is associated with decreased time.]
Testbed - given the same cost

- LeNet
- CNN
- Linear Classification

Time (s)

- m4.2xlarge
- Siren
Conclusion

• **Siren**: Distributed Machine Learning with a Serverless Architecture
  - Hybrid Synchronous Parallel (HSP)
  - Experience-Driven Resource Scheduler

• **Evaluation**
  - Simulation & Testbed
  - 44.3% ↓ on job completion time
Q&A  Thank You