Cirrus:
A Serverless Framework for End-to-end ML Workflows

Joao Carreira, Pedro Fonseca, Alexey Tumanov, Andrew Zhang, Randy Katz

Berkeley
University of California

Purdue
University

Georgia Tech
Machine Learning
End-to-end ML workflows

- Modern end-to-end ML workflows are complex
End-to-end ML workflows

- Modern end-to-end ML workflows are complex
- ML workflows consist of 3 heterogeneous stages
End-to-end ML workflows

- Modern end-to-end ML workflows are complex
- ML workflows consist of 3 heterogeneous stages

Dataset
Preprocessing
End-to-end ML workflows

- Modern end-to-end ML workflows are complex
- ML workflows consist of 3 heterogeneous stages
End-to-end ML workflows

- Modern end-to-end ML workflows are complex
- ML workflows consist of 3 heterogeneous stages
End-to-end ML workflows

- Modern end-to-end ML workflows are complex
- ML workflows consist of 3 heterogeneous stages

ML workflows are **interactive** and **iterative**
Provisioning ML workflows is challenging

- Hard to accurately estimate resource demands of each stage
- Data scientists have limited systems expertise

- Complex infrastructure management *detracts from ML work*
- **Resource waste** due to overprovisioning of resources
Serverless computing

Code
$f(x)$

Input
AWS S3

Output
Serverless computing

Code
\( f(x) \)

Input

\( \lambda \)

Output
Serverless computing benefits

**Tight provisioning of resources**
- Fine-grained billing
- Fine-grained resources
- High elasticity

**Simplifying infrastructure management**
- Automatic resource configuration / provisioning / maintenance
Challenges of serverless

- Small local memory and storage
- Limited lambda package size
- Lack of fast shared storage
- Low bandwidth and no P2P communication
- Short-lived and unpredictable launch times
Existing approaches

Serverless Frameworks

Machine Learning Frameworks
Existing approaches

Serverless Frameworks

- **PyWren**
  - Limit. Pkg size
  - Download dependencies from S3
  - High-latency communication through S3
  - Unpred. launch

Machine Learning Frameworks

- **faast.js**
  - Stragglers
Existing approaches

**Serverless Frameworks**

- **PyWren**
  - Download dependencies from S3
  - High-latency communication through S3
  - Stragglers

- **faast.js**
  - Limit. Pkg size
  - No fast storage
  - Unpred. launch

**Machine Learning Frameworks**

- **TensorFlow**
  - Unable to launch runtimes in lambdas

- **Spark**
  - Small mem.
  - No ring/tree reduces
  - No P2P comm.
  - Unpred. launch
  - Precludes MPI
Cirrus: a framework for serverless end-to-end ML workflows
Cirrus: design principles

1) Addressing serverless challenges

- Low memory
- Limited package size
- No P2P communication
- No fast storage
- Short lifetimes and unpredictable launch

- Ultra-lightweight runtime + data prefetching
- High-perf. data store (parameter-server and KV)
- Robust handling of lambda termination
2) Achieving benefits for end-to-end ML

- Tight provisioning of resources
- Per-stage fine-grained variable agile scalability
- Simplifying infrastructure management
- High-level API supports end-to-end ML
Cirrus architecture (client side)

Dashboard

Python API

Client frontend
- Preproc.
- Training
- Tuning

Create/Stop Task

Client backend
- Task Scheduler
- Lambda Manager

Data scientist

Client side (stateful)
Cirrus Dashboard

Loss vs. Time

Current Cost: $0.04
($0.00028/sec)
Num Lambdas: 57
Mem Usage: 126 MBs

Show All
Show Best Five
Show Worst Five
Chose line: 3
learning_rate: 0.2

Kill Line: 3
Kill All
Cirrus Dashboard

Loss vs. Time

Current Cost: $0.04
($0.00028/sec)

Num Lambdas: 57

Mem Usage: 126 MBs

Rate 0.200000

(373.0047, 0.7388219)

Chose line: 3

learning_rate: 0.2

KILL LINE: 3
KILL ALL
Cirrus architecture (server side)

Cirrus runtime
- Data Iterator API
- Minibatch Buffer
- Sparse LR
- Mat. Fact.
- LDA

Data store client API

Data store
- PS API
- Key-value API
  - SGD
  - Adagrad
  - Momentum
  - Models Key-values

Server side (stateless)

- put
- get
- put/get

Cirrus runtime
- put
- get
- put/get key
Cirrus evaluation

1. Cirrus provides benefits by specializing both for serverless and end-to-end ML.

2. We show that Cirrus outperforms a state-of-the-art serverless system: PyWren.
Evaluation setup

1. Deployment: AWS Lambdas (3GB of mem.)
2. Benchmark: async. distributed SGD Sparse Logistic Regression task
3. Dataset: Criteo Dataset (a dataset of display ads)
4. PyWren:
   a. Baseline: iterative synchronous SGD training using AWS S3 to store gradients and model
   b. + 3 incremental optimizations
5. Cirrus: 2 modes (with/without prefetching)
Cirrus outperforms vanilla serverless

Synchronous SGD training suffers from stragglers
Multiple SGD iterations on each lambda invocation

Asynchronous SGD

Cirrus outperforms vanilla serverless
Cirrus outperforms vanilla serverless

Test Loss

Sparse gradients and training data prefetching

PyWren

+ Reuse Lambdas + Async.

+ Sparse Grad. + Pref.
Cirrus outperforms vanilla serverless

Test Loss

- PyWren
- + Reuse Lambdas + Async.
- + Sparse Grad. + Pref.
- + Redis

Replace AWS S3 with high-performance store (Redis)

+700x updates/sec
Cirrus outperforms vanilla serverless

Cirrus without training data prefetching
Cirrus outperforms vanilla serverless

Test Loss

Cirrus with training data prefetching
1. End-to-end ML workflows:
   a. time-consuming infrastructure management
   b. resource overprovisioning
2. Cirrus -- serverless end-to-end ML framework:
   a. simplify deployment of ML workflows
   b. per-stage provisioning of resources
3. Cirrus outperforms existing serverless solutions by specializing for serverless and ML
Thank you!

github.com/ucbrise/cirrus  @jccarreira