Ray: A Distributed Framework for Emerging AI Applications

Review by **Ross Tooley**

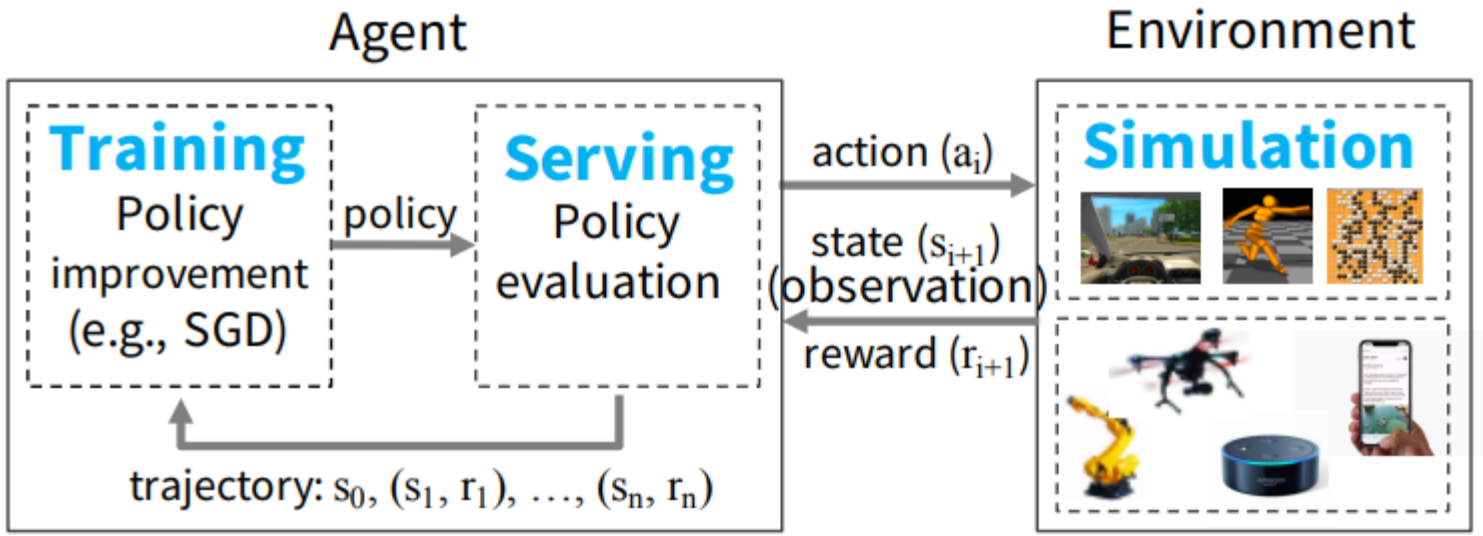
2020/10/15

Background

Distributed Execution Engine

- Handles the complexity of running on a distributed system.

Reinforcement Learning



TensorFlow

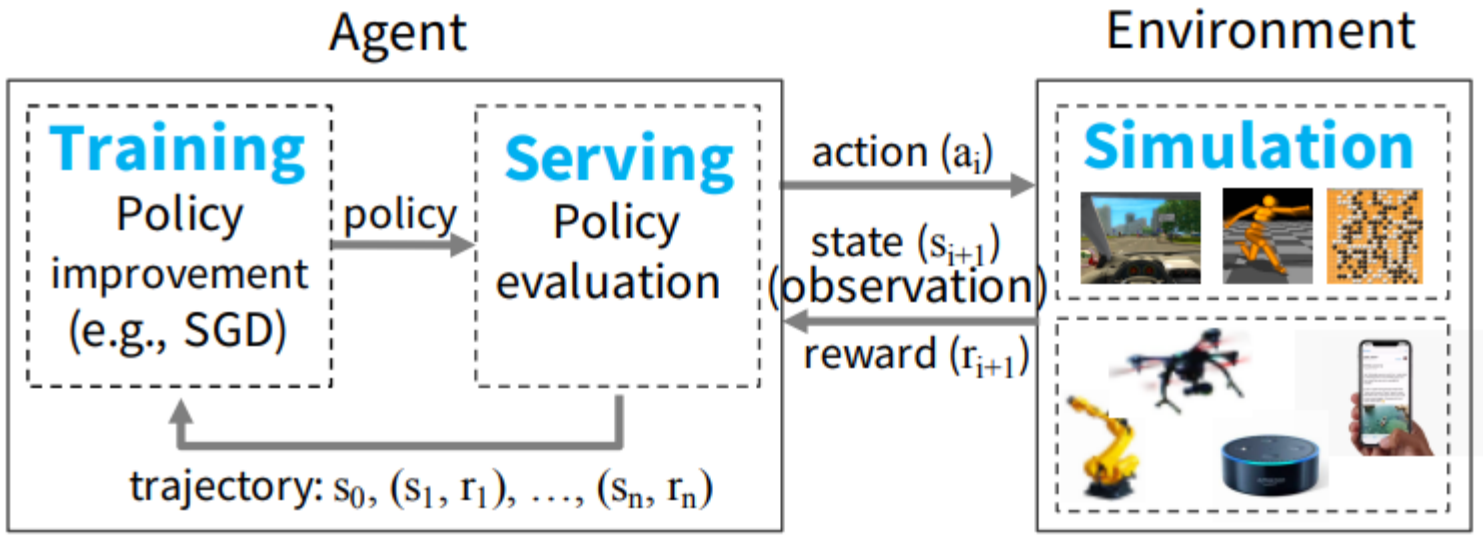
HOROVOD

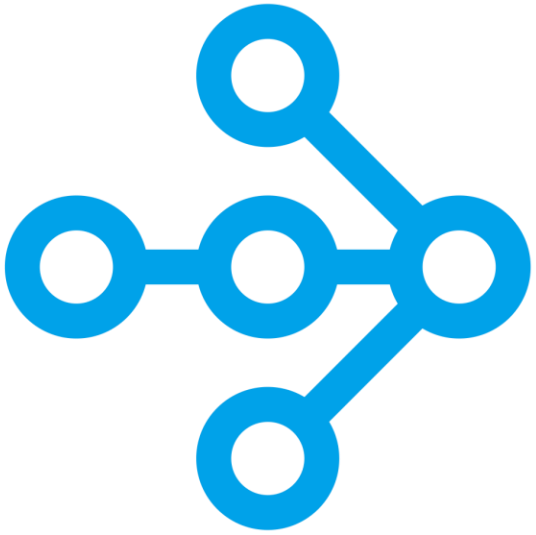
Clipper

APACHE Spark™

DASK

Reinforcement Learning

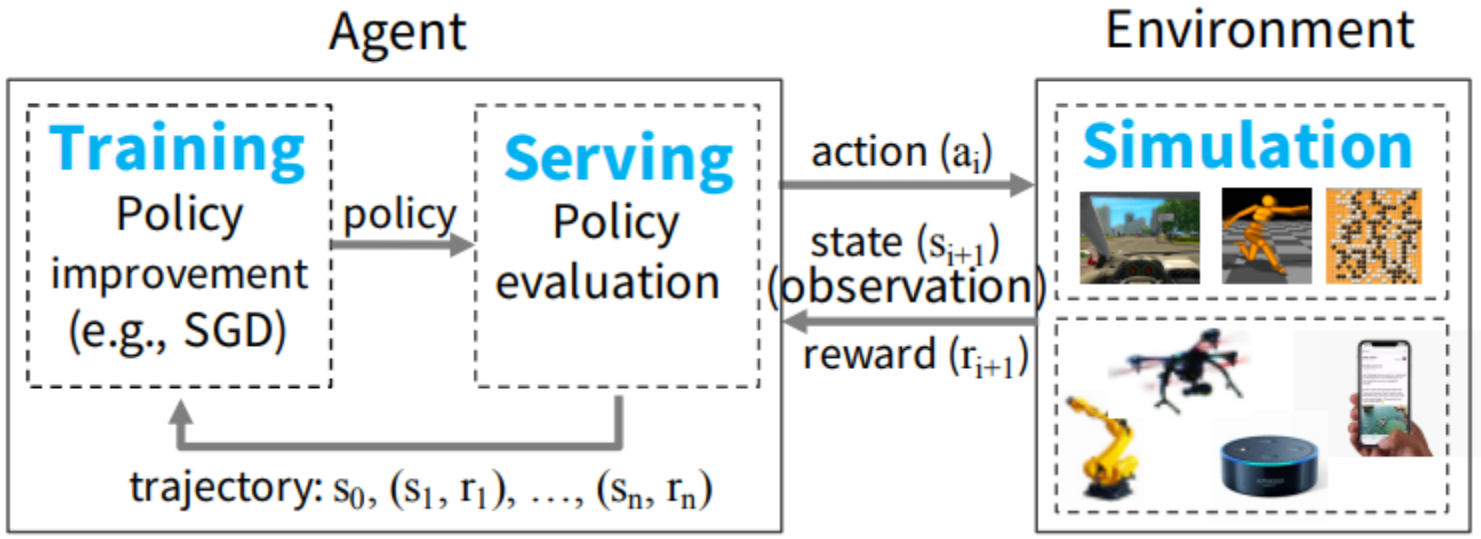




RAY

Motivation

- Tight-coupling
- Dynamic scheduling
- Heterogenous hardware



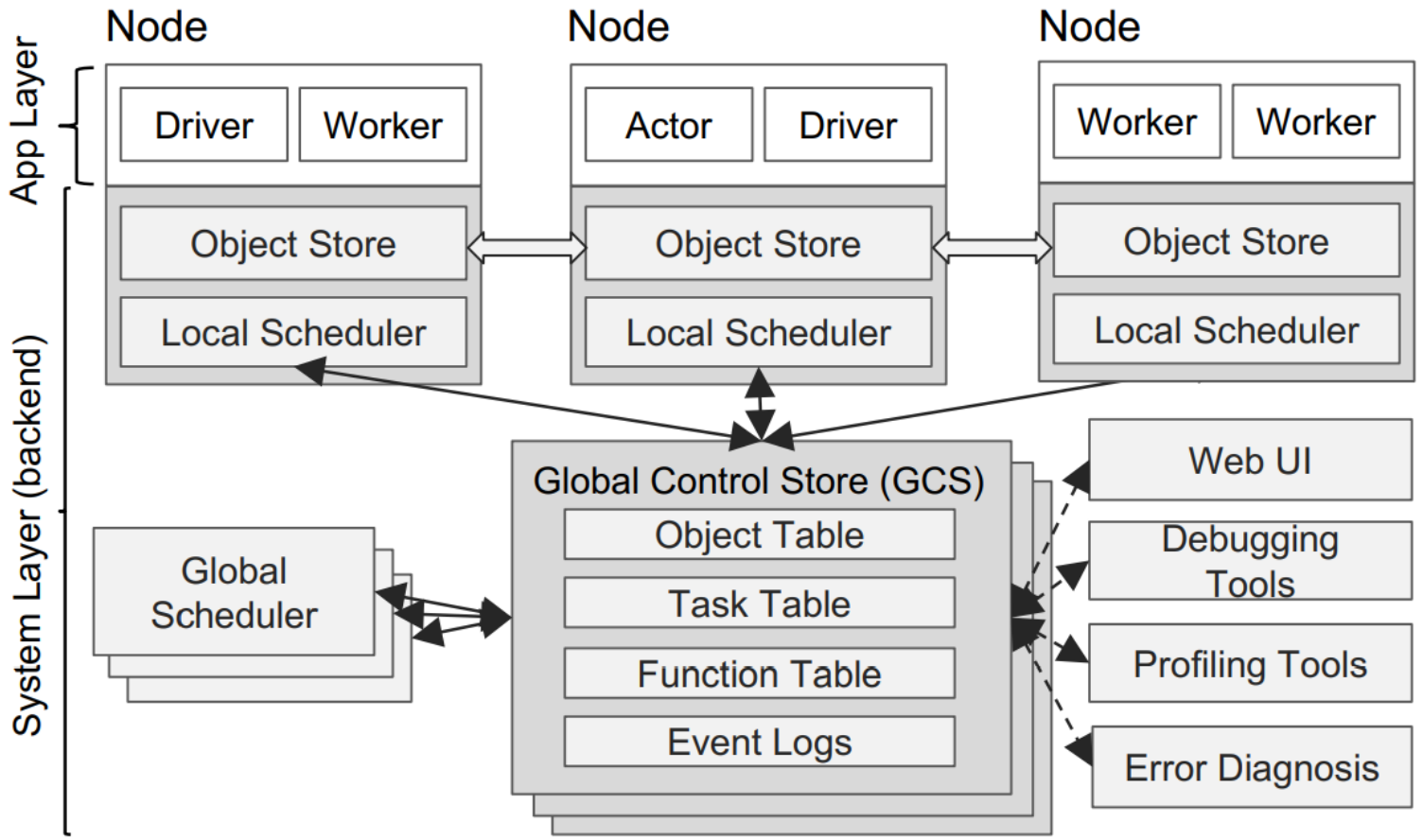
Key Features

API

- Two types of computational unit

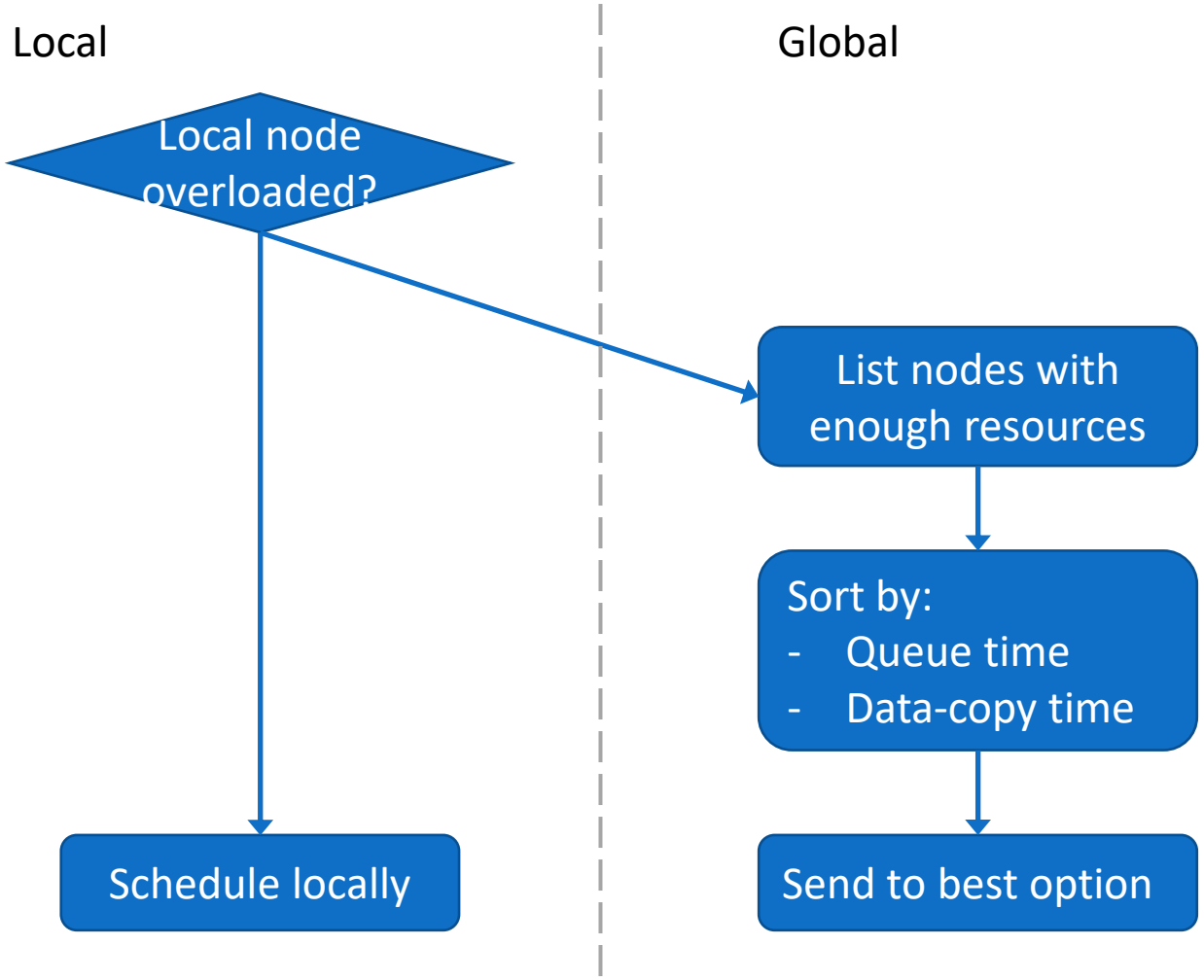
Task	Actor
Borrowed from Spark, CIEL, etc.	Borrowed from Akka, Rx, etc.
Remote	Remote
Stateless	Stateful
e.g. Simulation	e.g. Training
Fine-grained load-balancing	Low-overhead updates
Efficiently fault-tolerant	

Architecture



- Object Store provides *shared memory* per node
- GCS uses *sharding* to scale-out

Bottom-up Scheduler



- Reduces bottleneck at Global Scheduler
- Supports Data Locality

Results

Building Blocks

Training

- Equals Horovod throughput, 10% off TensorFlow. (Distributed SGD)

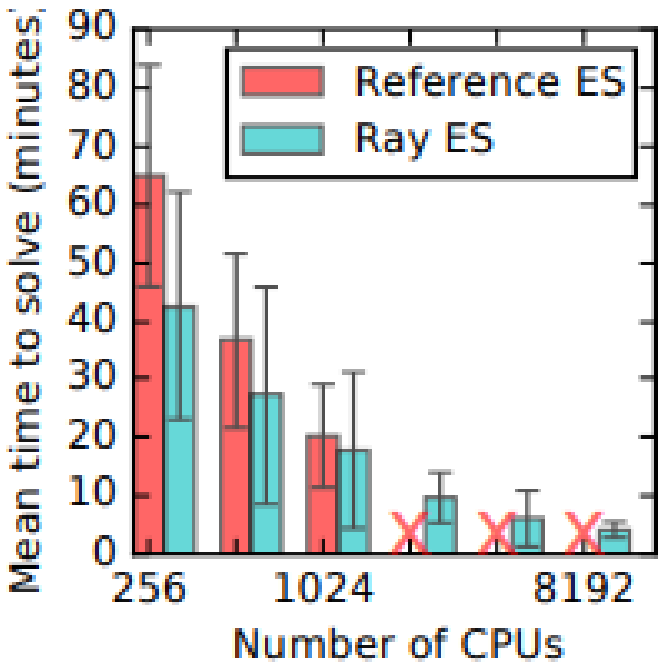
Serving

- Order of magnitude faster throughput than Clipper. (Embedded simulation, same machine, fully-connected NN)

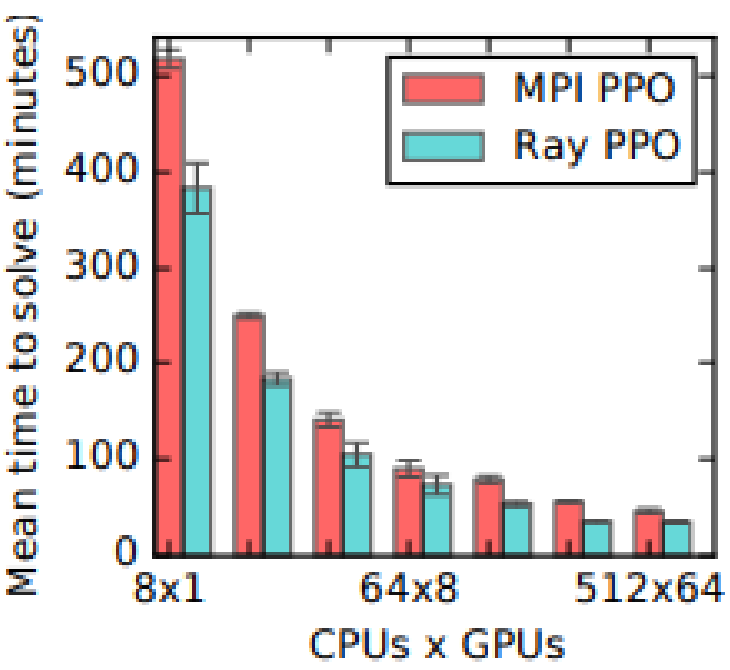
Simulation

- 1.8x throughput of MPI. (Pendulum-v0)

RL Applications



(a) Evolution Strategies



(b) PPO

Review

Impressive Systems Tech

- GCS and Bottom-up Scheduler are contributions in their own right.
  - Horizontally-scalable dynamic scheduling
  - Data-locality support
- Could be applied to other ‘dynamically-scheduled’ frameworks
- Though dynamically-scheduled frameworks are rare
  - Can lead to high data copying, so may be slow in worst case
  - RL restricts the applications, but may not always avoid this

‘Scalability! But at what COST?’

Scalability! But at what COST?

Frank McSherry Unaffiliated
Michael Isard Unaffiliated*
Derek G. Murray Unaffiliated†

Abstract

We offer a new metric for big data platforms, COST, or the Configuration that Outperforms a Single Thread. The COST of a given platform for a given problem is the hardware configuration required before the platform outperforms a competent single-threaded implementation. COST weighs a system’s scalability against the overheads introduced by the system, and indicates the actual performance gains of the system, without rewarding systems that bring substantial but parallelizable overheads.

We survey measurements of data-parallel systems recently reported in SOSP and OSDI, and find that many systems have either a surprisingly large COST, often hundreds of cores, or simply underperform one thread for all of their reported configurations.

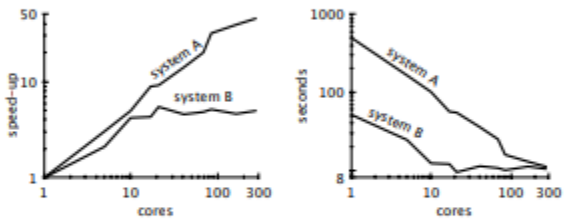


Figure 1: Scaling and performance measurements for a data-parallel algorithm, before (system A) and after (system B) a simple performance optimization. The unoptimized implementation “scales” far better, despite (or rather, because of) its poor performance.

While this may appear to be a contrived example, we will argue that many published big data systems more closely resemble system A than they resemble system B.

Is Ray useful?

- Fits a niche: RL applications with (short) embedded simulations.
  - More efficient than existing frameworks and bespoke solutions

- But, is this a common pattern?
  - Otherwise it is just yet another Distributed Framework
  - Lacking funding of Spark / TensorFlow to add features and optimisations

Since publication

- Integration with many existing frameworks

- New RL libraries use Ray as a pluggable backend
  - RLlib
  - RLgraph