HI!

CS 744: RAY

Shivaram Venkataraman

Fall 2020

ADMINISTRIVIA

- Assignment Grades? → by mid / late next week
- Project proposal aka Introduction (10/16)

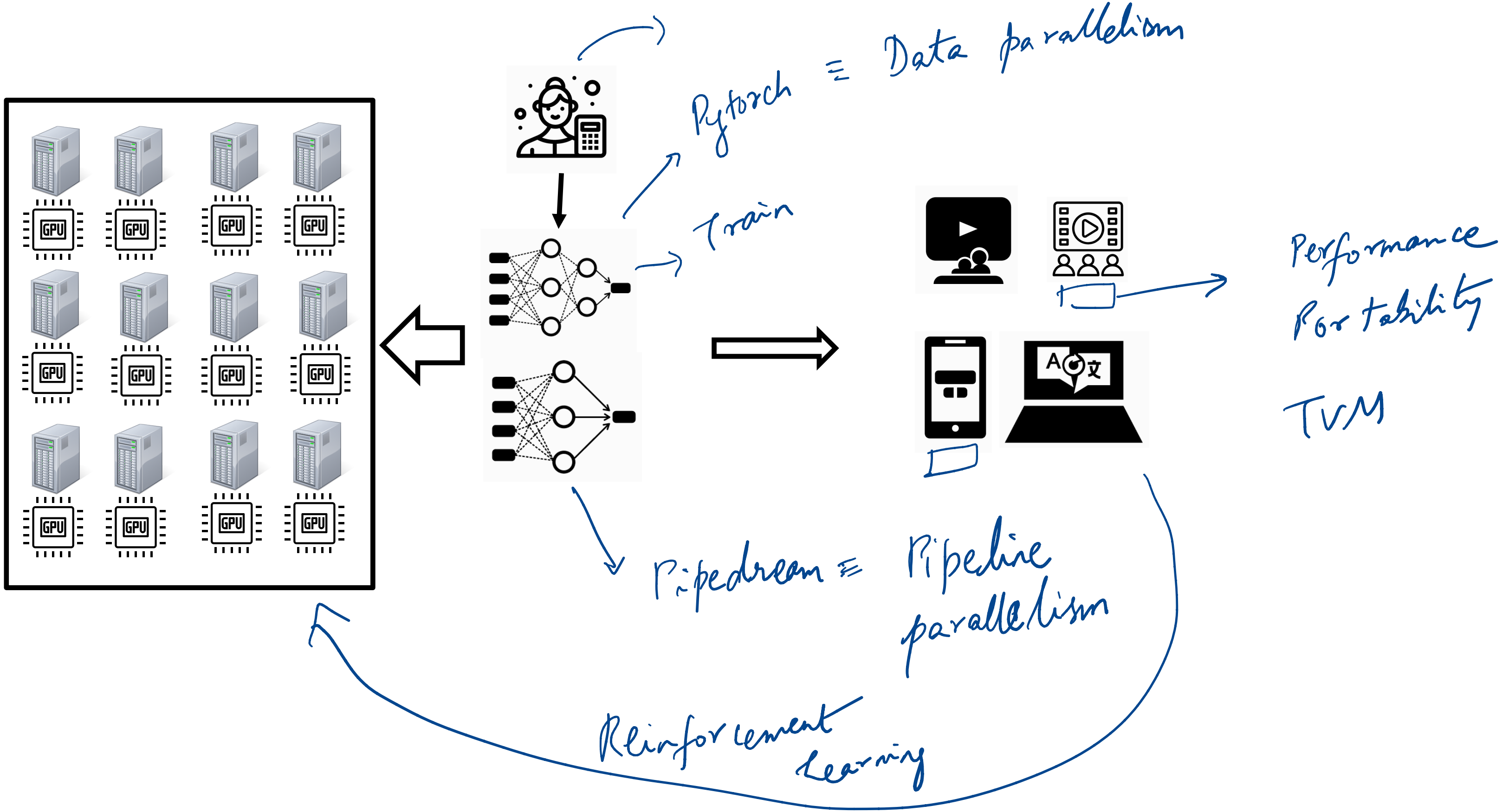
  Introduction

  Related Work

  Timeline (with eval plan)

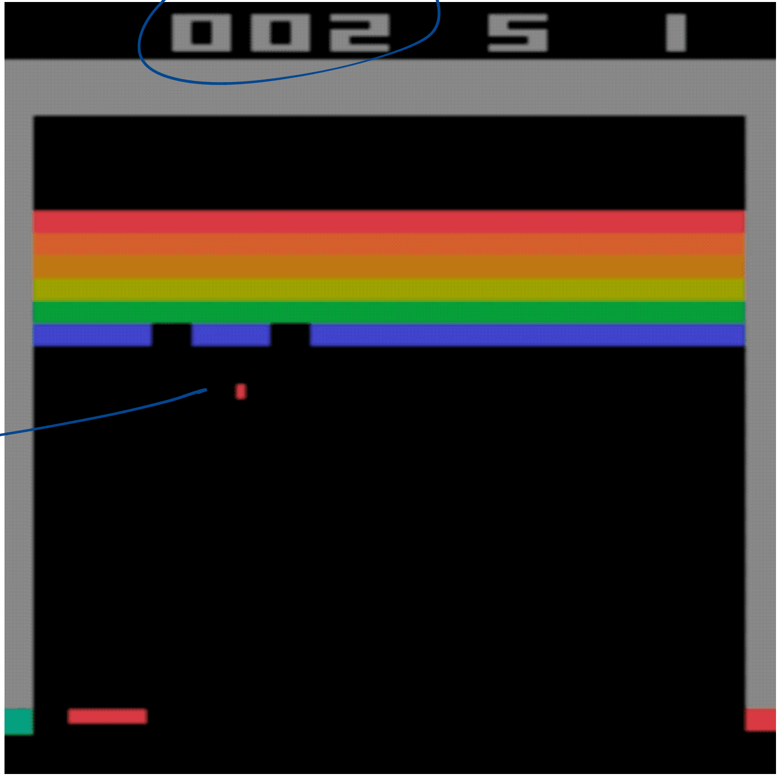
- Midterm: Oct 22 → early next week.

MACHINE LEARNING: STACK



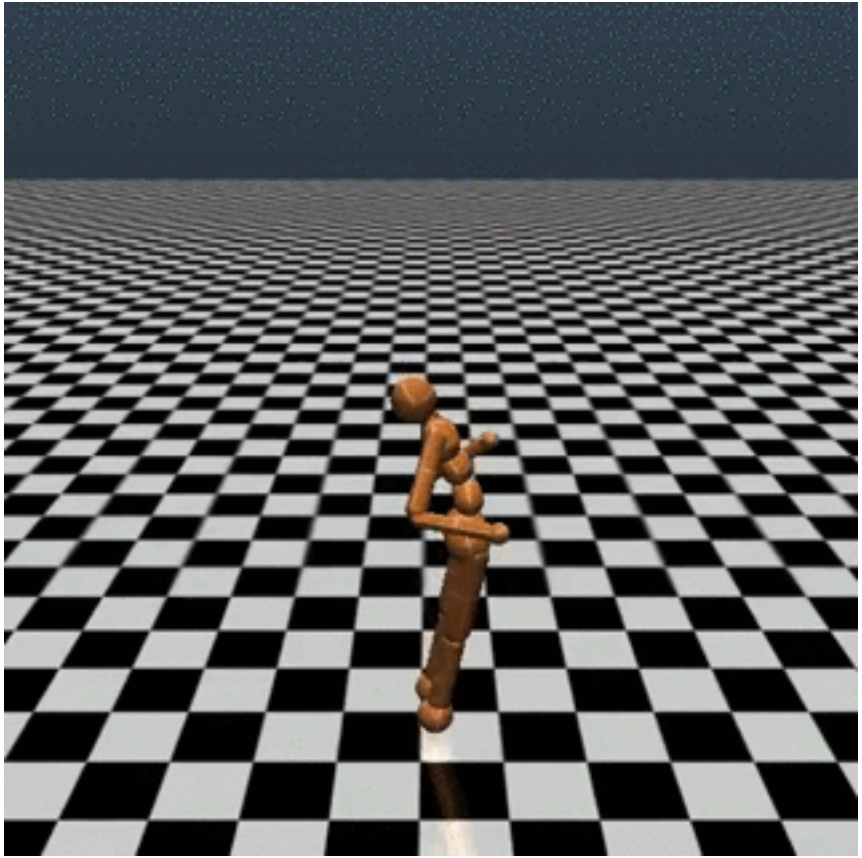
REINFORCEMENT LEARNING

Reward



Agent controlled by algorithm

Not just supervised learning

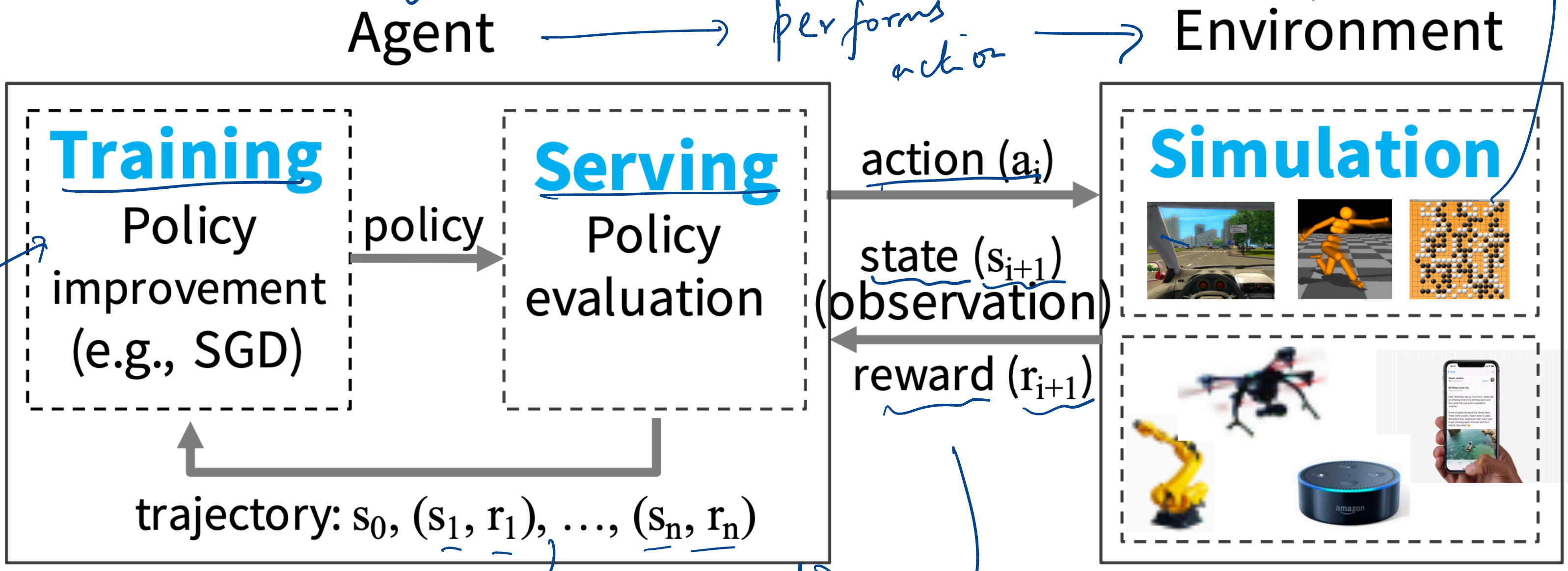


RL SETUP

Anything that performs an action

road/traffic

board/game state

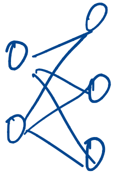


performs action

ML model that decides next action

(history of ~~sta~~ states, rewards)

tells us the effects of this action



static execution plan

RL REQUIREMENTS

Simulation → fine grained computation → flexibility

↳ each simulation could be ~ ms or hours

Training → stateful processing

↳ Model state/ simulator state

stateless processing

↳ data pre-processing

Serving → very low latency

very High throughput

1M tasks / sec

Dynamic execution

↳ future computation depends on output of past compute

RAY API

stateless and side-effect free?] same input → same output

Tasks

Tasks → any function that is run remotely

```
futures = f.remote(args)
```

local variable / future

normal Python function → f(args) local

a handle that you can wait on

Actors

Erlang ~80s/90s

Stateful tasks

Inbox M_0 M_1 M_2 M_3

a1 ← C1 Outbox

class Actor
<state>
def handleMsg(msg):
use <state>
update <state>

```
actor = Class.remote(args)
futures = actor.method.remote(args)
```

method name

msg

= actor.method.remote(args1)

```
objects = ray.get(futures)
ready = ray.wait(futures, k,timeout)
```

args will handled before

args1

Futures can be arguments to tasks

↳ you can spawn (or wait) for tasks within a task

RAY API

Tasks

Nested tasks

```
futures = f.remote(args)
```

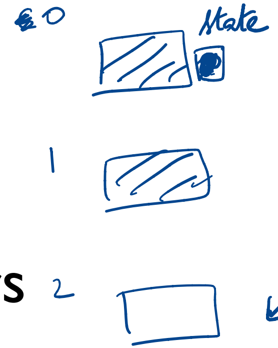
def f (args):

f0 = g.remote ((1,2,3))

o = ray.wait (f0)

(args , o)

Actors



run this anywhere

Task O

<internal>

```
actor = Class.remote(args)
futures = actor.method.remote(args)
```

for i in 1 to 10:

f. remote (i)

wait()

```
objects = ray.get(futures)
ready = ray.wait(futures, k,timeout)
```

trigger to run the computation

Lineage !

COMPUTATION MODEL

create_policy.remote()

Dotted lines

↳ Control edges

spawn a task

spawn an actor

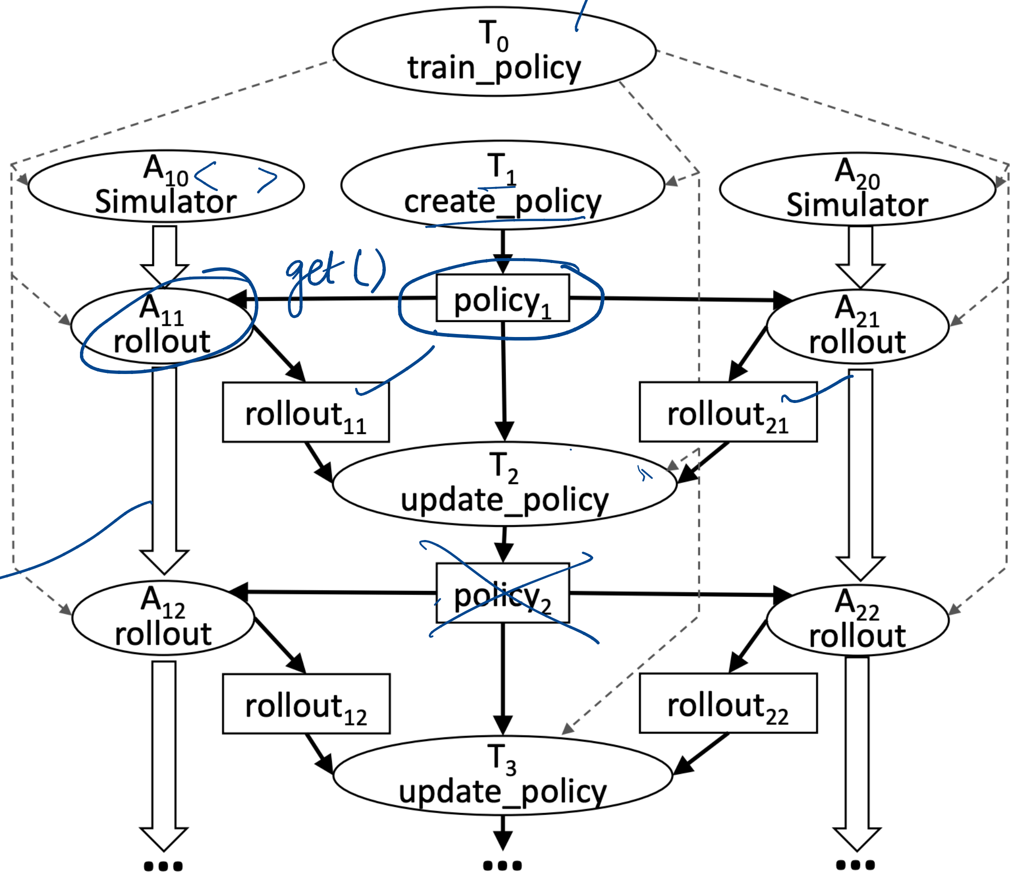
Solid lines

↳ Data edges

⇒ Stateful edges

action on actor

happen sequentially



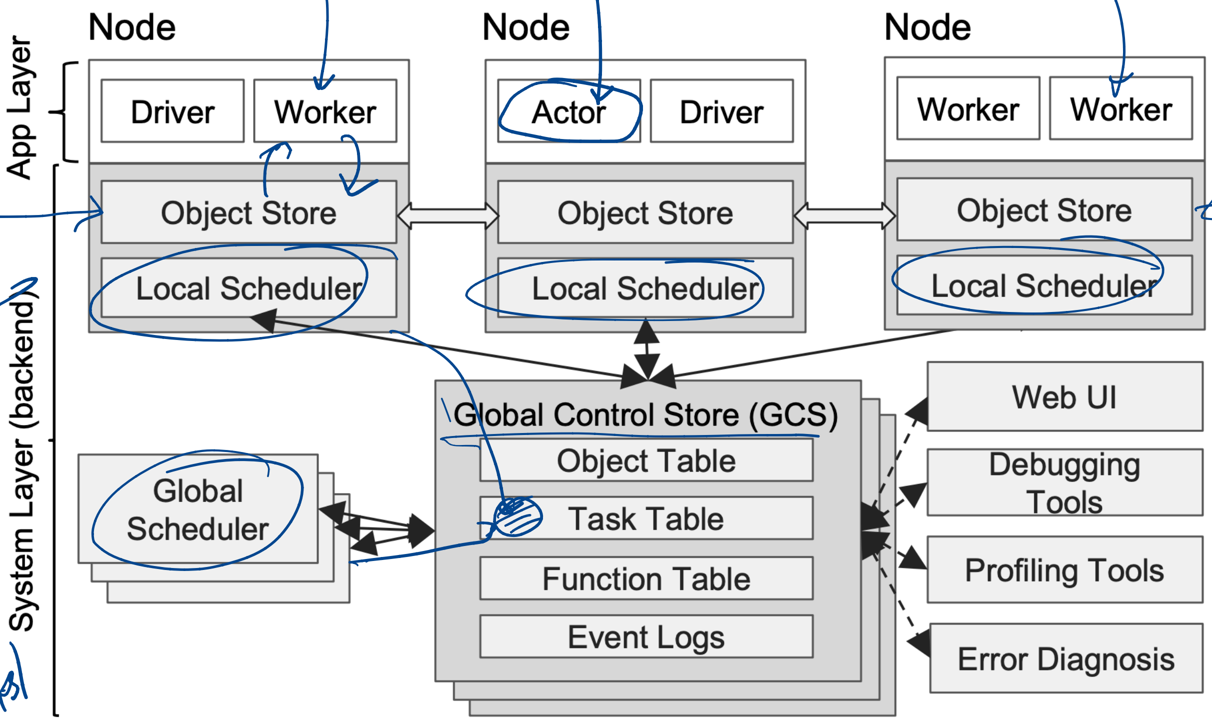
ARCHITECTURE

Deterministic hash

key % hash = machine num

state of actor

Can run tasks



save inputs/outputs from tasks

in-memory data store

Scheduling is challenging given they need 1M tasks/sec.

GLOBAL CONTROL STORE

Sort of a Database

Object table

→ list of all objects and their locations } Namenode metadata

Task table

→ Lineage of tasks

Function table

→ Code blocks corresponding to tasks

Externalizes state!

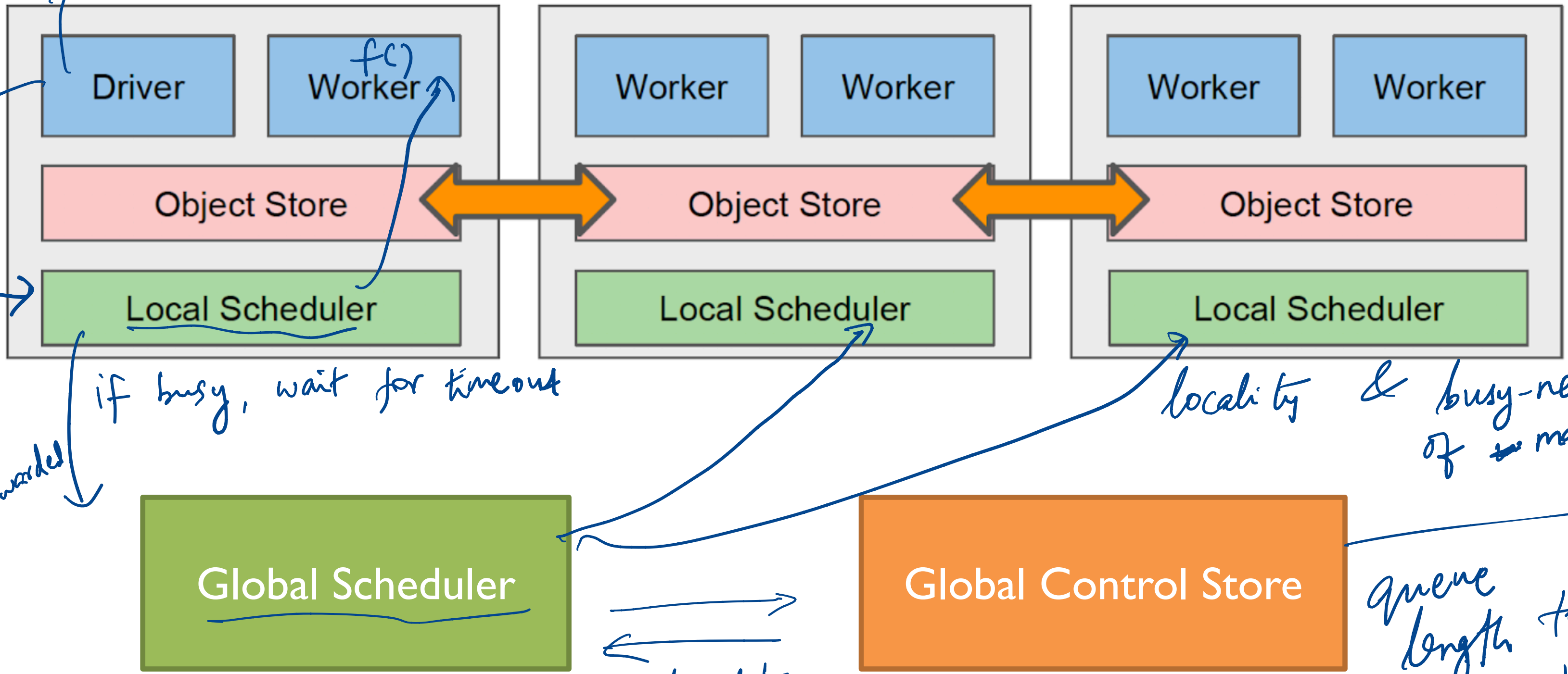
→ Shard
Replicate

Scale more easily, simplify sched design / fault tolerance

RAY SCHEDULER

Can local scheduler take locality?

f.remote(args)



f()

if busy, wait for timeout

forwarded

locality & busy-ness of machine

locality

queue length to determine if node is busy

FAULT TOLERANCE

Tasks → lineage, replay or re-execution of tasks

Actors → periodically checkpoint actors

→ restore ckpt

replay messages

GCS →

replication chain

Scheduler → Stateless!

Nothing?

Re-spawn or launch a new scheduler

SUMMARY

Ray: Unified system for ML training, serving, simulation

Flexible API with support for

- Stateless tasks
- Stateful Actors

Distributed scheduling, Global control store

DISCUSSION

https://forms.gle/PN5FSJB6vVkDjoih8

Consider you are implementing two apps: a deep learning model training and a sorting application. When will use tasks vs actors and why ?

state locality

Actors

stateless, load-balancing

Tasks

Sorting

Does external sorting → State

Deterministic operations

still have dependencies!

divide into smaller parts?

Model Training

weights are state,

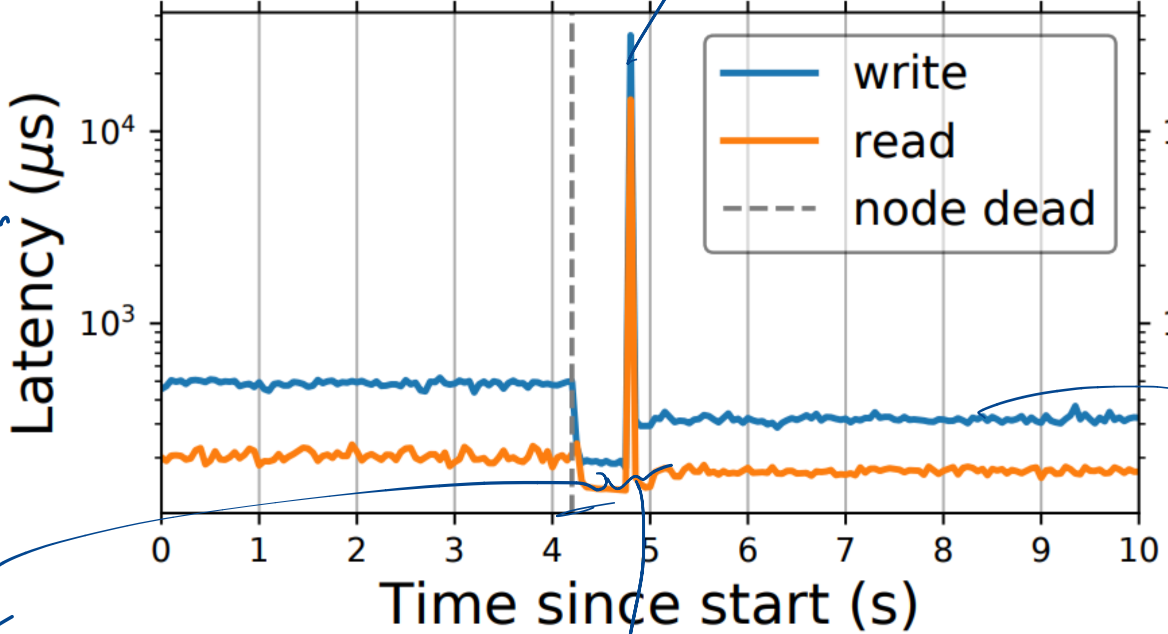
Multiple for data parallel

Can do dependencies between iterations?

fine-grained recovery

warm-up this new replica

one replica no overhead from replication chain



new node has better hardware

better/closer to replica?

goes down after recovery?

during failure goes down here?

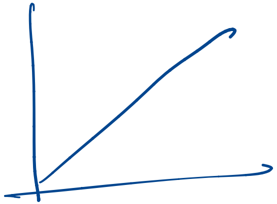
latency spike lasts for 30 ms

NEXT STEPS

Next class: Clipper

Last lecture on ML!

Linear Scalability



Sub linear

Super linear

100 GB

1 disk

12GB/mc

8 mcs

memory