CS 744: RAY

Shivaram Venkataraman
Fall 2020
- 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

PyTorch = Data parallelism

Train

Performance

Portability

TVM

Pipedream = Pipeline parallelism

Reinforcement learning
REINFORCEMENT LEARNING
Reward

Agent controlled by algorithm
RL SETUP

**Agent**

**Training**
Policy improvement (e.g., SGD)

**Serving**
Policy evaluation

**Simulation**
Action \( a_i \)
State \( s_{i+1} \)
Observation
Reward \( r_{i+1} \)

**ML model**

Next action

Anything that performs an action

performs action

Road/traffic

board/game state

Trajectory: \( s_0, (s_1, r_1), \ldots, (s_n, r_n) \)

Tells us the effects of this action

(history of states, rewards)
RL REQUIREMENTS

Simulation
- Static execution plan
- Fine grained computation → flexibility
- Each simulation could be ~ms or hours
- Stateful processing
- Model state/simulator state
- Stateless processing
- By data pre-processing

Training
- Stateful processing
- Model state/simulator state
- Stateless processing
- By data pre-processing

Serving
- Very low latency
- Very high throughput
- 1M tasks/sec
**RAY API**

**Tasks**
- any function that is run remotely

```
futures = f.remote(args)  # local variable / future
```

**Actors**
- stateful tasks

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

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

- futures can be arguments to tasks
  - you can spawn (or wait) for tasks within a task

Futures will handle arguments before `args`.
futures = f.remote(args)

def f(args):
    for i in 1 to 10:
        fo = g.remote((1, 2, 3))
        o = ray.wait(fo)
        futures = futures + o
        objects = ray.get(futures)
        ready = ray.wait(futures, k, timeout)

RAY API

Tasks

Actors

Nested tasks

actor = Class.remote(args)
futures = futures + future

wait()
COMPUTATION MODEL

Dotted lines
Control edges
Spawn a task
Spawn an actor

Solid lines
Data edges

"Stateful edges
Action on actor
Happen sequentially"
Deterministic hash key: machine tasks, can run tasks in-memory data store.

Node
- Driver
- Worker
- Object Store
- Local Scheduler

Node
- Actor
- Driver
- Object Store
- Local Scheduler

Node
- Worker
- Worker
- Object Store
- Local Scheduler

Global Control Store (GCS)
- Object Table
- Task Table
- Function Table
- Event Logs

System Layer (backend)
- Global Scheduler

Application Layer
- Driver
- Worker

Scheduling is challenging given they need tools/IM work/sec.

Fagin's idea: 'idea'

Anger 01 Emir:
GLOBAL CONTROL STORE

Sort of a Database

Object table

→ list of all objects and their locations

Task table

→ lineage of tasks

Function table

→ code blocks corresponding to tasks

Externalizes

⊥

metadata

⊥

state!

⊥

shard

Replicate

scale more

easily, simplify

achieved design

fault tolerance
RAY SCHEDULER

Global Scheduler

Can local scheduler take locality?

Local Scheduler

Object Store

Driver

Worker

Remote lag

Locality & busy-ness of machine

Forwarded

if busy, wait for timeout

Global Control Store

queue length to determine if node is busy

Remote (args)

fung

if busy, wait for timeout

locality

Driver

Worker

Worker

Worker

Local Scheduler

Local Scheduler

Local Scheduler

Object Store

Object Store

Object Store

Global Scheduler
FAULT TOLERANCE

Tasks $\rightarrow$ lineage, replay or re-execution of tasks

Actors $\rightarrow$ periodically checkpoint actors $\rightarrow$ restore checkpoint $\rightarrow$ replay messages

GCS $\rightarrow$ shared | replication chain

Scheduler $\rightarrow$ 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/PN5FSJB6vVkJoih8
Consider you are implementing two apps: a deep learning model training and a sorting application. When will use tasks vs actors and why?

- **Tasks (stateless)**
  - Sorting: Does external sorting → state
  - Model Training: weights are state, multiple for data parallel
  - Load-balancing

- **Actors (state locality)**
  - Deterministic operations still have dependencies! Can divide into smaller parts?
  - Can do dependencies between iterations?
  - Fine-grained recovery
A graph showing latency over time since start. The y-axis is labeled "Latency (μs)" with a logarithmic scale ranging from $10^3$ to $10^4$. The x-axis is labeled "Time since start (s)" ranging from 0 to 10 seconds. The graph has two lines: one for write operations and another for read operations. There is a spike in latency at time 4 seconds, which lasts for 2 seconds. Notes include:

- "new node has better hardware"
- "goes down after recovery?"
- "latency spike lasts for 20 ms"
- "node dead"
- "warm up"
- "replica"
- "replica to replica?"
- "replica goes down here?"
- "recovery?"
- "ageing Minkowski"
- "one replica no overhead from replication"
- "diving failure"
- "goes down here?"

Additional notes:

- "YET"
- "replica"
- "node dead"
- "n goes down"
- "gift"
Next class: Clipper
Last lecture on ML!

- Linear scalability
- Sub linear
- Super linear

Hardware approx.

100 GB
12 GB/sec
1 disk
8 GB memory