SLO Management with Reinforcement Learning on Multi-tenant Serverless FaaS Platforms

Haoran Qiu¹, Weichao Mao¹, Archit Patke¹, Chen Wang², Hubertus Franke²
Zbigniew Kalbarczyk¹, Tamer Basar¹, Ravishankar Iyer¹

¹ UIUC ² IBM Research

The 2nd Workshop on Machine Learning and Systems (EuroMLSys) 2022
Problem Background and Motivation

• SLO (service-level objectives) management is hard in serverless FaaS platforms
  • Fine-grained resource management (concurrency, container resource allocation)
  • Heterogeneous functions (characteristics/SLO), workload dependent, frequent update, …
• Heuristics-based resource management has been proved to be inefficient and untenable
• Reinforcement learning (RL)-based resource management has been proposed in general “serverless” platforms:
  • DeepRM [HotNets ’16], MIRAS [ICDCS ’19], Symphony [ICML ’20], FIRM [OSDI ’20], ADRL [TPDS ’20], AutoPilot [EuroSys ’20], Q-learning-based Autoscaler [CCGrid ’21], …
• Single-agent RL in Single-tenant Environment
Problem Background and Motivation

- Reinforcement learning (RL)-based resource management has been proposed in general "serverless" platforms:
  - DeepRM [HotNets ’16], MIRAS [ICDCS ’19], Symphony [ICML ’20], FIRM [OSDI ’20], ADRL [TPDS ’20], AutoPilot [EuroSys ’20], Q-learning-based Autoscaler [CCGrid ’21], ...
  - Single-agent RL in Single-tenant Environment

- Violation of the standard assumption of environment stationarity
  - RL assumes that the underlying environment is stationary
  - Not true anymore from each RL agent’s perspective when multiple self-interested RL agents are added to the same environment
Violation of Stationarity – A Motivating Example

Environment

Policy $\pi_\theta$

Action $A_t$

CPU.shares $\leftarrow 256$

State $S_t$, Rewards $R_t$

$\%$ CPU.shares $= \frac{256}{1024}$

Environment

Policy $\pi_\theta$

Action $A_{t+1}$

$\%$ CPU.shares $= \frac{512}{1280}$

State $S_{t+1}$, Rewards $R_{t+1}$

Environment

Policy $\pi_\theta$

Action $A_t$

CPU.shares $\leftarrow 512$

State $S_t^1$, Rewards $R_t^1$

$\%$ CPU.shares $= \frac{256}{1024}$

Environment

Policy $\pi_{\theta_1}$

Action $A_t^1$

CPU.shares $\leftarrow 256$

State $S_t^2$, Rewards $R_t^2$

$\%$ CPU.shares $= \frac{256}{1024}$

Environment

Policy $\pi_{\theta_2}$

Action $A_t^2$

$\%$ CPU.shares $= \frac{512}{1280}$

State $S_{t+1}^1$, Rewards $R_{t+1}^1$

Environment

Policy $\pi_{\theta_1}$

Action $A_{t+1}$

$\%$ CPU.shares $= \frac{768}{1792}$

State $S_{t+1}^2$, Rewards $R_{t+1}^2$

$\%$ CPU.shares $= \frac{768}{1536}$

Optimal solution: Agent 1 gets 0.4 (512/1280), Agent 2 gets 0.5 (768/1536)
Goal

• Provide system support that enables multiple RL-based controllers to coexist with each other

• Design principles

1. Performance Isolation
   • During training: converge to a collectively optimal policy
   • During execution: achieve comparable performance to single-agent RL in single-tenant cases

2. Scalability
   • Challenge: In a multi-tenant serverless FaaS platform, new functions from different customers can be increasingly registered

3. Fast Training/Retraining
   • Challenge: Functions from any customer can be registered, removed, or updated at any time, which changes the joint state space
Single-agent RL Pipeline in OpenWhisk

- Client
- API Gateway
- Controller
- Function Requests
- Invoker Node #1
  - Function Pod
  - Function Pod
  - Function Pod
- Invoker Node #2
  - Function Pod
  - Function Pod
  - Function Pod
- Invoker Node #3
  - Function Pod
  - Function Pod
  - Function Pod
- Data Store
- Activation Results
- Source Code Or Images
- Horizontal & Vertical Scaler
- RL Agent
- States Rewards
- Action

One RL-agent per Function

- Docker & Cgroup
- RL Proxy

- Activation Results
- Source Code Or Images
Single-agent RL Design

- **PPO**: a policy gradient method and the default RL algorithm in OpenAI
- **States**: SLO Preservation Ratio ($SP_t$), Resource Utilization ($RU_t(CPU, mem)$), Arrival Rate Changes ($AC_t$), Resource Limits ($RLT_t(CPU, mem)$), Horizontal Concurrency ($NC_t$)
- **Actions**
  - Vertical scaling: +/- step size of resource limits $av_t = \Delta RLT_t(CPU, mem)$
  - Horizontal scaling: +/- step size of number of function containers $ah_t = \Delta NC_t$
- **Reward function**

\[
R_t = \alpha \cdot RU_t + \beta \cdot SP_t + \text{penalty}
\]

- Resource Utilization
- SLO Preservation
- Penalize illegal or undesired actions:
  - Frequent dangling decisions
  - Scale in/up/down when $NC_t = 0$

\[
SP_t = \min\left(\frac{SLO \text{ Latency}}{Actual \text{Latency}}, 1\right)
\]
Reward Function Sensitivity Study (Single-agent RL)

Variant 1: \( R_t = SP_t + \text{penalty} \) (Over-provisioning)

Variant 2: \( R_t = RU_t + \text{penalty} \) (Under-provisioning)
Multi-tenant RL Pipeline in OpenWhisk
Single-agent RL Evaluation

- Comparison baseline: ENSURE [ACSOS ’20]
  - Heuristics-based horizontal & vertical autoscaler
- Single-agent RL achieved similar end-to-end latency with ENSURE in single-tenant cases
- Multi-tenancy results in 64-78% function performance degradation (in terms of end-to-end latency)
Multi-agent PPO (MA-PPO)

- Reward function for each agent: \( R_t = \frac{\sum_i^N R(i)_t}{N} \)
- Extended states for each agent: \( S(i)_t = O(i)_t \cup G_t(i) \), where \( G_t \) is the global system states:
  - Aggregated horizontal actions: \( AH(i)_t = \sum_{j \neq i}^N \frac{ah_t^j}{N-1} \), aggregated vertical actions: \( AV(i)_t = \sum_{j \neq i}^N \frac{av_t^j}{N-1} \)
  - Average SLO preservations: \( ASP(i)_t = \sum_{j \neq i}^N \frac{sp_t^j}{N-1} \), average resource utilizations: \( ARU(i)_t = \sum_{j \neq i}^N \frac{ru_t^j}{N-1} \)

\( A_t \): action
\( O_t \): observation
\( R_t \): reward
\( G_t \): global states
\( \pi_t \): policy (actor)
\( V_t \): value function (critic)
Multi-tenant RL Pipeline in OpenWhisk (with MA-PPO)

States & Rewards

\[ R_t = \sum R(i)_t / N \]
MA-PPO Training

![Graph showing total reward per episode over RL training episodes, with 31.3% drop, 41.5% drop, 39.0% drop, and 11.1% drop.](image)

- Added 5 functions
- Removed 5 functions
- Added 5 functions
- Added 5 functions
MA-PPO Online Performance

• MA-PPO can provide online performance comparable to single-agent RL in single-tenant cases, with the performance degradation ranging from 1.8% (for sentiment-analysis, 1190.2 ms to 1211.5 ms) to 9.9% (for markdown2html, 178.4 ms to 198.1 ms).

• Compared to the single-RL trained in multitenant environments, the MA-PPO achieves 2.5× (for sentiment-analysis, 1211.5 ms to 3047.8 ms) to 4.4× (for image-resize, 154.2 ms to 672.4 ms) improvement.
Conclusion

• Single-agent RL
  • Suffers performance degradation and is unable to train (converge) in multi-tenant cases

• MA-PPO: Customized multi-agent PPO
  • Collect rewards from all agents; Agnostic to agent order and size of the agent group
    • Observations from all other agents (aggregated/averaged values)
  • Able to train (and converge) and achieves comparable performance in multi-tenant cases compared with single RL in single-tenant cases

• Future work
  • Fast retraining: Network parameter sharing, transfer learning
  • Fault tolerance
Thank you!