Metis: Learning to Schedule Long-Running Applications in Shared Container Clusters at Scale

Luping Wang*, Qizhen Weng*, Wei Wang, Chen Chen, Bo Li
Hong Kong University of Science and Technology
Long-Running Applications (LRAs) are critical in modern datacenters

- Also known as latency-critic (LC) services, which are of commercial value, e.g.,
  - Stream processing
  - Interactive data analytics
    - Caching/Storage services ...

- LRAs usually run for hours to months and occupy substantial resources.

Alibaba: 94% LRA-dedicated machine/clusters
Microsoft: 37% LRA-dedicated machine/clusters
Google: High proportion of long production Jobs.

LRAs are non-trivial to schedule

- LRAs have stringent SLO (Service-Level Objective) requirements.
- LRAs have sophisticated performance interactions among themselves.
  - I/O dependencies — *affinity*: better co-located

Image Super Resolution with TensorFlow.

MXNet Model Server, classifying images.
LRAs are non-trivial to schedule

- LRAs have stringent SLO (Service-Level Objective) requirements.
- LRAs have sophisticated performance interactions among themselves.
  - I/O dependencies — *affinity*: better co-located
  - Shared resources contention — *anti-affinity*: better scattered

- CPU last-level cache
- Memory bandwidth
- Network and disk I/O …
• Pursue LRAs’ best end-to-end performance of various metrics.

• Directly learn LRAs’ interactions from traces, instead of prior knowledge.

• Scale to thousands of nodes within moderate latency.
• Introduction

• Prior Arts and Metis’ Approach

• Hierarchical Reinforcement Learning

• Evaluation
Medea scheduler workflow:
(0. Cluster operators identify interactions.)
1. Manually set placement constraints.
2. Solve Integer linear program (ILP).
3. Executes ILP solution.

Prior Arts: Constraint-Based Schedulers

Problems:

1. Expensive to get constraints.
2. Inefficient at scale (>10 hours to place 3,000 containers to 700 machines).
3. Suboptimal. Constraints are qualitative, but not quantitative. Esp. given conflicting constraints.
1. Group container requests.
2. Train an RL agent for each group.
3. Collect placement decisions.
4. Launch containers in cluster.
5. Profile containers’ performance.

Implementation

- Algorithms, baselines, and benchmarks (in Docker) are available at [https://github.com/Metis-RL-based-container-sche/Metis](https://github.com/Metis-RL-based-container-sche/Metis)
Trades computation and latency for better performance. Offline-trained agent performs badly, because the input are highly variant:

- Cluster state changes after each deployment.
- Input container group can have millions of combinations, e.g., Picking 30 containers from 7 apps (with repetition) gives ~2 million outcomes.
• Introduction

• Prior Arts and Metis’ Approach

• Hierarchical Reinforcement Learning

• Evaluation
Hierarchical Reinforcement Learning

- Co-locating containers using Reinforcement Learning
  - End-to-end scheduling process
  - Intelligent interference-capture method: try-trail
  - Generally support various scheduling objectives
  - Scalable
Hierarchical Reinforcement Learning

- Novelty: Hierarchical Reinforcement Learning for Scalability
  - Cluster scale: thousands of machines
  - Large action and state space: poor performance
Hierarchical Reinforcement Learning

- **Novelty: Hierarchical Reinforcement Learning for Scalability**
  - Select a hosting machine: select a sub-cluster first
  - Reduced state and action space
  - Reusable building blocks (**Spatial**)
  - Offline vs Online
Outline

- Introduction
- Prior Arts and Metis’ Approach
- Hierarchical Reinforcement Learning
- Evaluation
Evaluation: Setup

• Prototype deployment on a EC2 clusters
  • Scale: a medium one with 81 nodes and a large one with 729 nodes.
  • Each node: m5.4xlarge instance with 16 vCPUs, 64 GB memory
  • Docker containers: each with 2 vCPUs and 8 GB memory.

• Metrics
  • Container performance: RPS
  • Cluster resource fragmentation: % of empty nodes

• Baselines
  • Medea
  • Paragon
Evaluation: Scheduling Performance

- 81-node cluster
- 200-400 containers
- 25% and 61% higher RPS than Medea and Paragon
- Modest scheduling latency within 10 min

(a) Average RPS with various container group sizes.

(b) Distribution of scheduling latency in all container groups.
Evaluation: Scheduling Scalability

- Large-scale experiments
  - 729-node, 1k-3k containers
  - Comparable performance to that in previous 81-node clusters
  - Timely scheduling within 1 hour

- Sub-scheduler design
  - Accelerates RL convergence by 40×–95×
  - less than 10% loss of RPS
Evaluation: Support of Various Scheduling Objectives

- Maximizing SLO Satisfactions
  - Outperforming Medea and Paragon by $1.6 \times$ and $4.4 \times$ on average

- Minimizing Resource Fragmentation
  - Reward: weighted sum of RPS and vacant machines
  - Smooth trade-off between the two objectives