Mobile-Kube: Mobility-aware and Energy-efficient Service Orchestration on Kubernetes Edge Servers

Saeid Ghafouri, Alireza Karami, Danial Bidekani Bakhtiarvand, Aliakbar Saleh Bigdeli, Sukhpal Singh Gill and Joseph Doyle
Introduction
Energy Proportional Computing


Figure 2. Server power usage and energy efficiency at varying utilization levels, from idle to peak performance. Even an energy-efficient server still consumes about half its full power when doing virtually no work.
Recommendations for improving consolidation

- Recommended utilisation levels of 75% for Hyperscale server users and 45% for smaller users in the US
- In the UK, HMG Sustainable Technology Advice and Reporting (STAR) have identified a “consolidation programmes to maximise use of capacity” as best practice for achieving this goal.
Commitments by Cloud Providers

● Microsoft committed to carbon neutrality by 2030
● Amazon committed to carbon neutrality by 2040
● What about Edge devices?
● In some ways better as they do not need supporting infrastructure for cooling
● However, mostly focused on latency reduction only.
● This can lead to low utilisation of resources
● Mobile-Kube attempts to balance the latency reduction and consolidation objectives for containerised services on the edge
● Also consider user mobility and how this affects the objectives
Container Orchestration Frameworks

- Containerized softwares
- Google Borg
- Docker Swarm
- Kubernetes!

*Figure 1: The high-level architecture of Borg. Only a tiny fraction of the thousands of worker nodes are shown.*
Edge Computing
Problem statement
Reinforcement Learning

New Environment State & Reward

Environment

Action

Agent

$\pi_\theta(a|s)$

$p(s'|s, a)$

$p_\theta(s_1, a_1, \ldots, s_T, a_T) = p(s_1) \prod_{t=1}^{T} \pi_\theta(a_t|s_t)p(s_{t+1}|s_t, a_t)$

$p((s_{t+1}, a_{t+1})|(s_t, a_t)) = p(s_{t+1}|s_t, a_t)\pi_\theta(a_{t+1}|s_{t+1})$

Markov chain on $(s, a)$
Contribution of this work

- A new design for reducing the **latency** and **energy consumption** on Kubernetes-driven edge nodes.
- Use of RL for achieving a trade-off between maintaining reasonable energy consumption and latency. Proposed the use of a distributed RL method named **IMPALA**.
- To test the efficiency of our method we have implemented a simulation framework for training and a real-world emulator on top of **real-world Kubernetes**.
- The RL based method is able to achieve similar energy efficiency of the heuristic methods while reducing the latency by 43%.
Proposed RL Solution
Latency Reduction objective
Proposed RL solution (Overview)

Based on the latency and bin packing objective

The Kubernetes cluster

New service placement

The users and services placement
Proposed RL solution (cont)

- **States**: concatenation of two arrays
  - An array containing the service placements
  - An array containing the users closest station
- **Actions**: next placement of the services in the nodes
- **Rewards**
  \[ R = w_1 R_b + w_2 R_l \]
  - The latency objective is computed from the inverse of the total distances of the users from their service
  - The binpacking objective is simply the number of empty servers
- **Policy network**: A 64*64 fully connected neural network
We used three algorithms for our experiments:

**Vanilla Policy Gradient**: The basis of all policy gradient methods used before in system research for bitrate adaptation

**PPO**: A more advanced version of the policy gradient which tries to minimize the variance by clipping the objective function

**IMPALA**: One of the newest widely used distributed RL algorithm with fast convergence and low variance

Also use heuristic methods (greedy and binpacking) for comparison
System Design
Kubernetes Internal Structure

Kubernetes Resource Model

- Request: reserved amount of resource for a container
- Limit: Maximum amount of resource for a container
- Exceeding limit: OOM error for memory and throttling for CPU
- We used resource request for scheduling

```yaml
spec:
  containers:
    - name: change
      image: m8s.gcr.io/ubuntu-slim:0.1
      resources:
        limits:
          cpu: 351m
          memory: 150Mi
        requests:
          cpu: 200m
          memory: 50Mi
      command: ["/bin/sh"]
      args: ["-c", "while true; do timeout 0.5s yes >/dev/null; sleep 0.5s; done"]
```
Kubernetes Default Scheduler

- Pods the smallest scheduling unit in kubernetes
- Currently the scoring is done based-on the rules defined by Kubernetes user and also heuristic algorithms
- Nodes available resources
- Requested resources
- A two step process
  - **Filtering**: Filtering out suitable nodes
  - **Scoring**: Ranks the nodes based-on a sets of criteria to find the most suitable node
- Assign the pod to the node with the highest rank
Limitations of the default scheduler

- Using the Kubernetes builtin custom scheduler feature was not feasible
- No migration of the pods based on external metrics
- We implemented this feature as deleting on pods in one place and restarting it in the destination node outside the cluster using the Python client API
Our design for changing the Scheduler

- Using the Python client API of Kubernetes
- For the emulation setting we discard the built in scheduler decision and used our own scheduler which resides outside the K8S cluster
- A better design for this should be fully integrated into the K8S
- The user mobility side is simulation
System design
Experimental Setup and Results
System setting and datasets

- **Cabspotting dataset:** The Cabspotting dataset contains GPS traces of taxi cabs in San Francisco (USA), collected in May 2008.
- [http://www.antennasearch.com/](http://www.antennasearch.com/) for the location of cell towers
- Python simulator for user mobility
- 8 kubernetes GKE nodes and 16 stateless services
- Reward scaling
Picture of the network

- Co-located stations and servers
- 5 minute interval mobility
Results - Training

- For training we generated a dataset based on the user locations around the servers
- A simulator that used the real-world K8S for training
Results - Test

- Average over 20 sample episode run
- On the cluster instead of the simulator

Average empty servers for different algorithms

Average latency for different algorithms

# empty servers

Average latency
Results - Example episode

- Single episode run per timestep

# empty servers

Average network latency
Directions for future works

- Checkpointing of stateful services
- Kubernetes full implementation
- Hierarchical and multi-agent RL
Thank you for your attention!

- Source code available at: https://github.com/saeid93/mobile-kube.git
- Currently under review in Transactions of Service Computing
- Email: j.doyle@qmul.ac.uk