Mobility Aware Service Orchestration in Mobile Edge Clouds

Saeid Ghafouri
Academic Advisor: Joseph Doyle

Queen Mary
University of London

✉ s.ghafouri@qmul.ac.uk

Service Orchestration

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

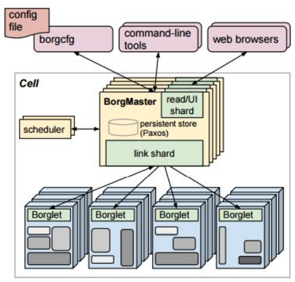


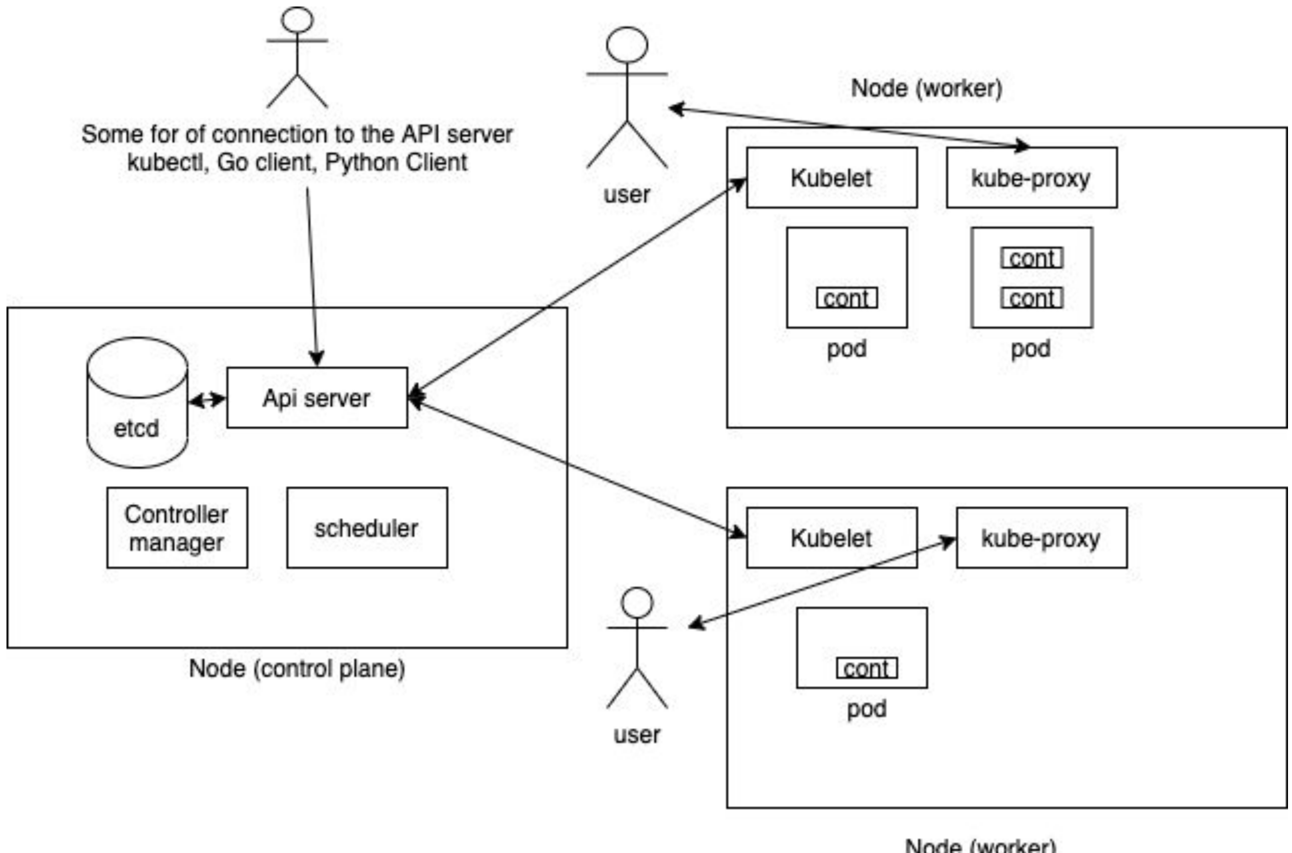
Figure 1: The high-level architecture of Borg. *Only a tiny fraction of the thousands of worker nodes are shown.*

kubernetes

docker

MESOS

Kubernetes Structure - Internals



Current Kubernetes Scheduling Model

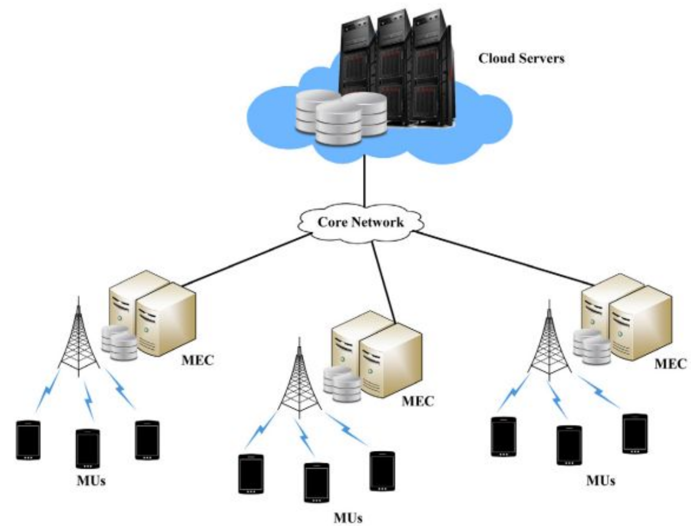
- Pods the smallest scheduling unit in kubernetes
- Currently the scoring is done based-on the rules defined by Kubernetes and Also heuristic algorithms
- Nodes available resources
- Requested resources
- A two step process
  - **Filtering**: Filtering out suitable nodes
  - **Scoring**: Ranks the nodes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

Kubernetes Limitations

- Kubernetes is great for centralised cloud and dominates the market
- Making Kubernets consistent with edge e.g. kubeedge
- **Research Question**: Can we optimise it's scheduling for latency intensive applications based-on **users' mobility** in mobile edge computing while considering **energy consumption**?
- Unrealistic assumptions in previous simulation-basd work

Mobile Edge Computing

- The vision for edge computing is to provide compute and storage resources close to the user in open standards and ubiquitous manner.
- Reducing the latency
- Examples:
  - Video streaming/processing
  - Virtual/augmented reality
  - Gaming-as-a-service



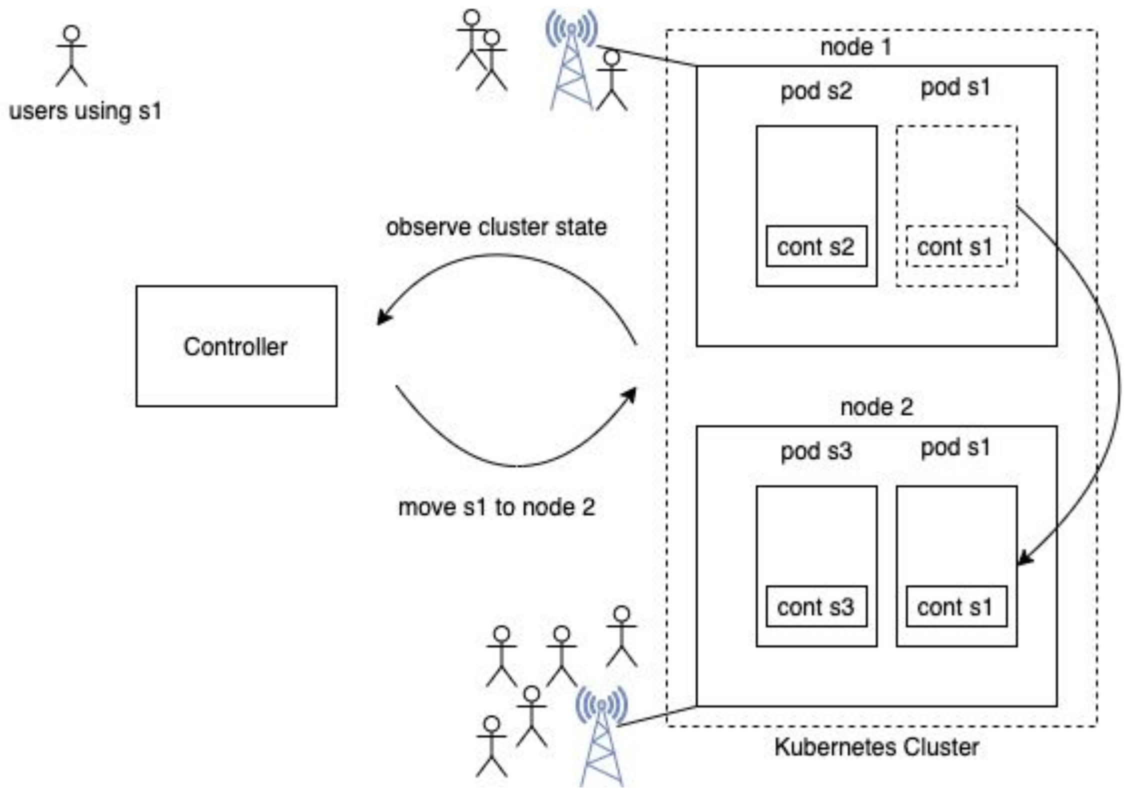
A Simple Scenario

- We have a set of services running on a network of connected servers on the edge of the cloud.
- Users are connected to them via a set of base-stations.
- Each user is only connected to one base station which is the closest base station to that user.
- Users might move around the set of stations.
- The station that the user is connected, could change if its new location is closer to another station.

Implementational Details - Kubernetes

- A Single pod with a single container for each service is considered
- A service is associated with each pod to expose it to the outside world
- A Controller outside the Kubernetes clusters observes the users movements and computes the new placement of pods based-on an reinforcement learning solution
- It then re-organise the services/pods to a new location based-on the current users' locations
- We have use Python client API to access the apiserver

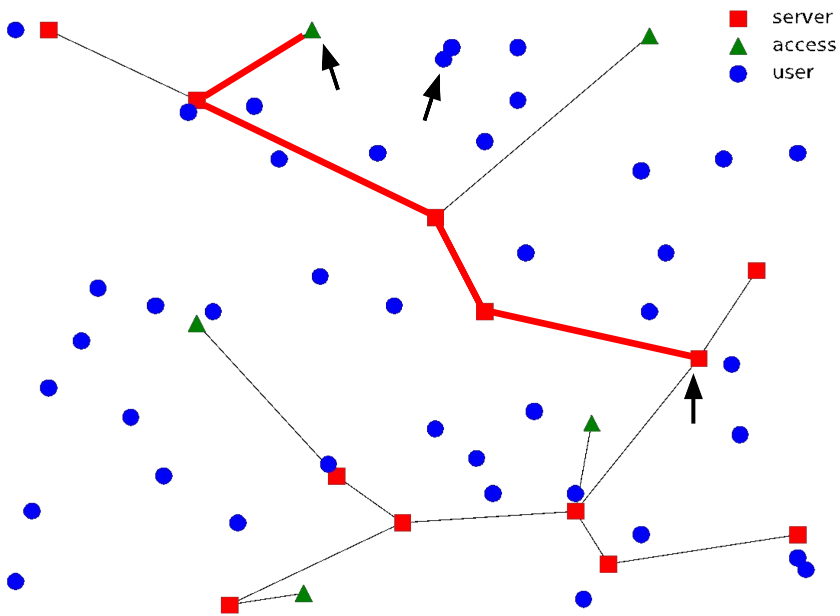
Implementational Details - Kubernetes

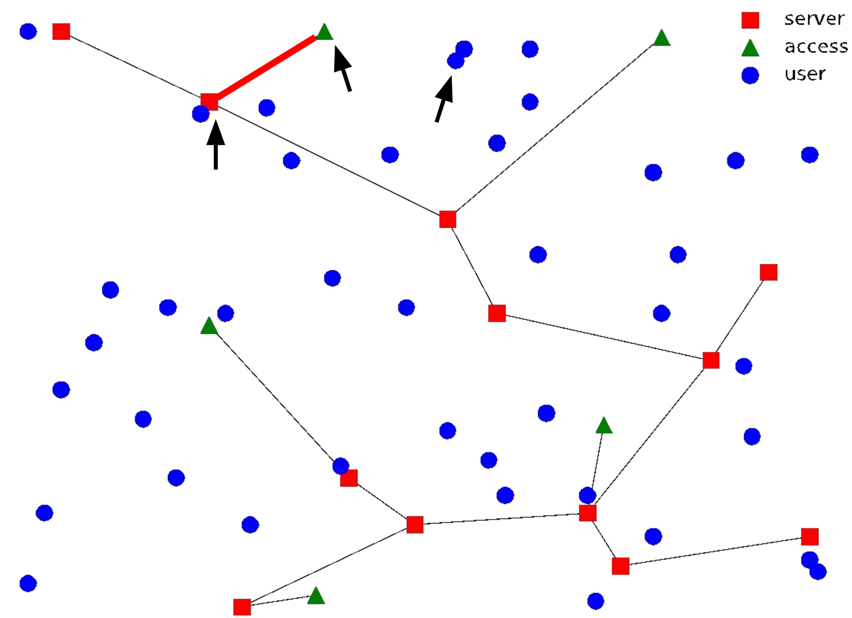


Implementational Details - User movements Simulations

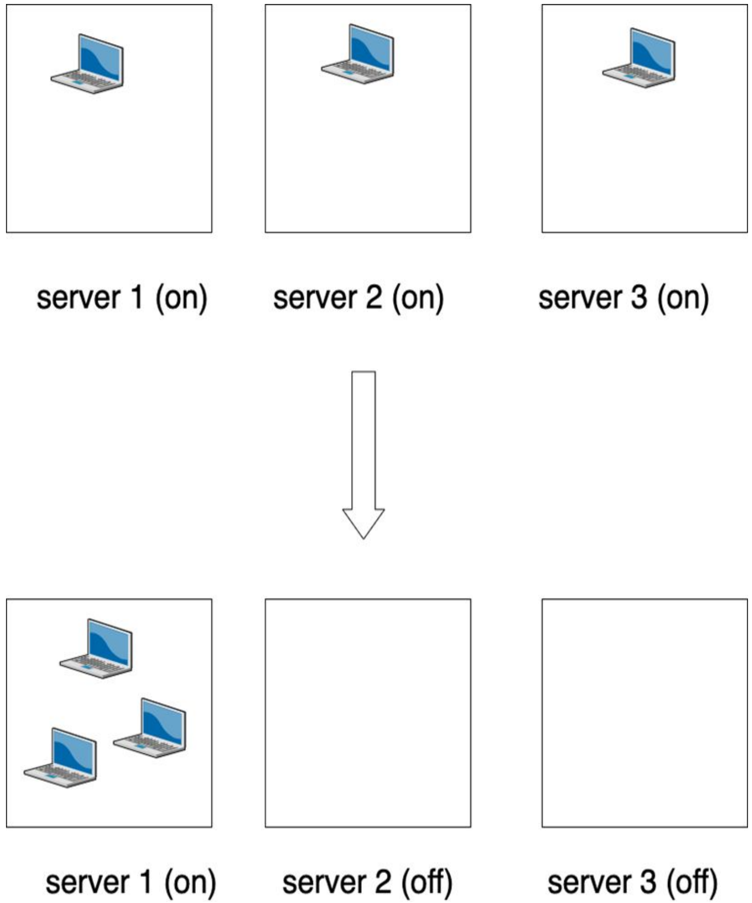
- Cabspotting dataset: The Cabspotting dataset contains GPS traces of taxi cabs in San Francisco (USA), collected in May 2008.
- http://www.antennasearch.com/ for the location of cell towers
- Python simulator for user mobility
- Real world Kubernetes clusters but the user mobility is simulation

First Objectives - Latency Reduction



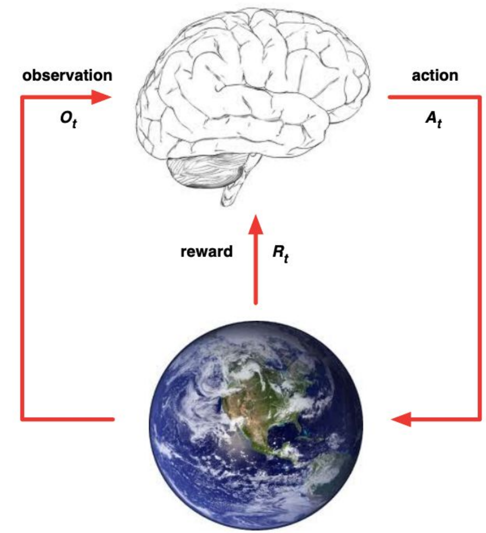


Second Objectives - Bin Packing



Reinforcement Learning as our Optimiser

- At each step t the agent:
  - Executes action At Receives observation Q_t
  - Receives scalar reward R_t
- The environment:
  - Receives action A_t
  - Emits observation Q_{t+1}
  - Emits scalar reward R_{t+1}
- We used an advance RL method called Proximal Policy Optimization (PPO)



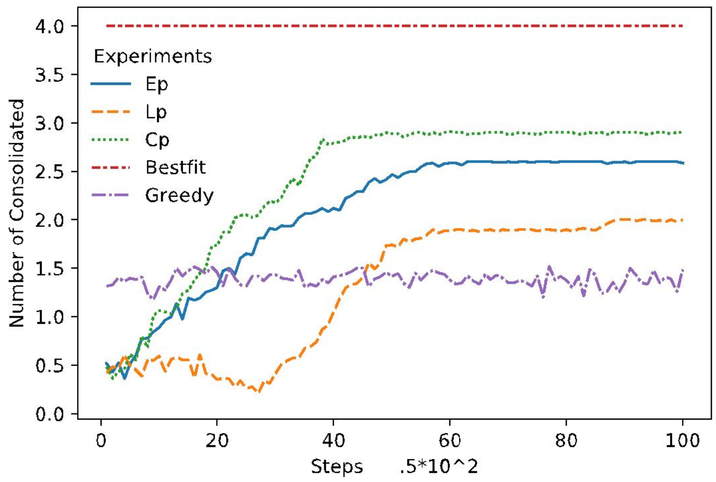
Early Stage Experiments Setting

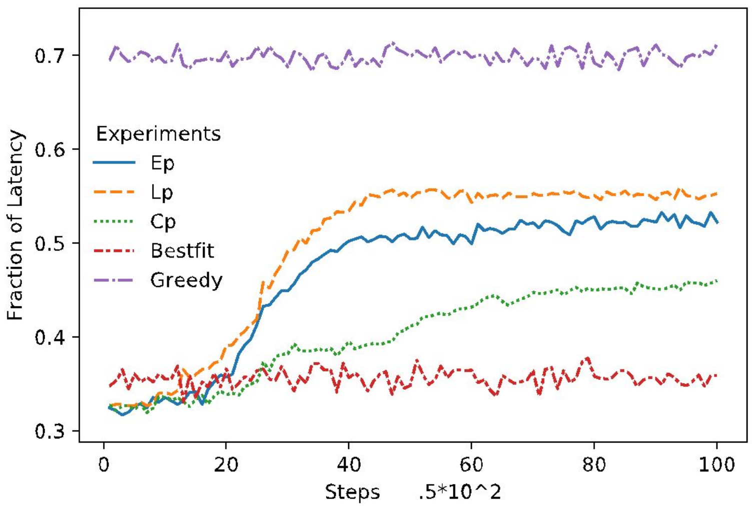
- Real-world experiments (work in progress) on GCP
- Rllib for RL implementation
- Baselines
  - Latency only
    - Greedy algorithm
  - Binpacking only
    - Best-fit bin-packing
    - E.g. Place the service on the server which has the maximum load where it fits
- 12 servers, 5 stations, 40 mobile devices

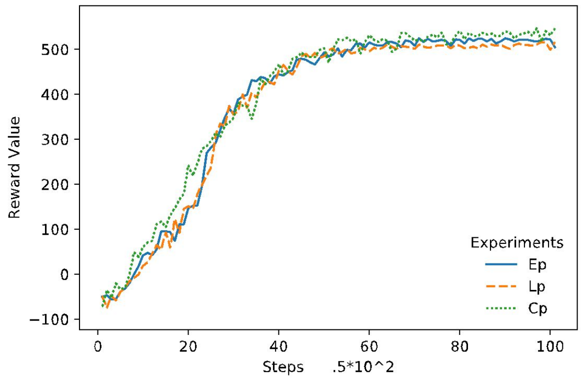
RL Model

- RL reward signal is computed at the end of each episode (certain number of timesteps)
- For Consolidation objective:
  - R_c = (Number of empty servers)/(normalizing_factor) * penalty
- For Latency Objective
  - R_l = (Fraction of users reaching their target latency) * penalty
- Total Reward:
  - $R = w_0 R_c + w_1 R_l$

Early Stage Experiments Results







Limitations and Future works

- Real-world but not Kubernetes-native yet
  - Using Custom resource definitions
  - Using Operators
- Some of container migration and networking challenges are not considered
  - Assuming stateless services with minimal migration cost
  - Consider stateful and stateless services
  - Simulation in the networking side and computing latency purely based-on distance

Thank you for your attention!

Please keep in touch if you are interested:
s.ghafouri@qmul.ac.uk