Distributed Redundant ~~Scheduling~~ Placement for Microservice-based Applications at the Edge

Hailiang ZHAO
Zhejiang Univeristy
http://hliangzhao.me

September 10, 2021

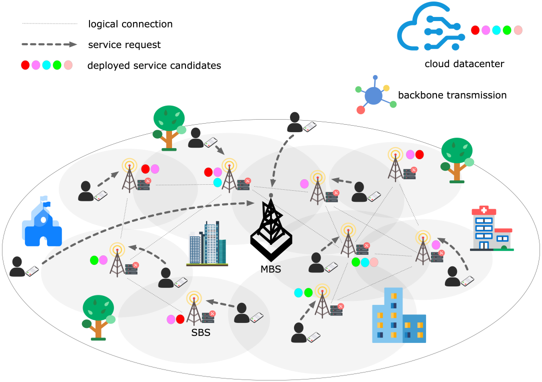
J1C2 Track, IEEE SERVICES 2021

Outline

Outline

The Multi-access HetNet

Multi-access Edge Computing is proposed to *provide services* and to *perform computations* at the network edge without time-consuming backbone transmission.



How to perform service provisioning at the network edge?

Service Placement at the Edge

How to perform service provisioning at the network edge?

- Where (which site) to place the services? $\longrightarrow$ A complicated algorithm, proposed in this paper
- How to deploy their instances? $\longrightarrow$ Docker and K8S

LIMITATIONS of current service placement algorithms:

- The *to-be-deployed* services only be studied in an atomatic way. Time series or composition property of services are not fully taken into consideration. $\longrightarrow$ We study SFC!
- High availability of deployed service is not carefully studied. $\longrightarrow$ Redundancy!

Motivation Scenarios

The Heterogeneous Multi-access Network

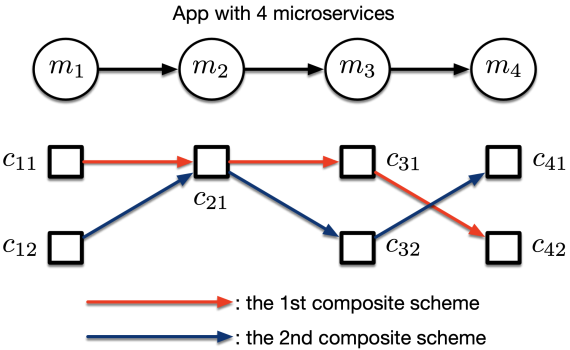
- One Macro Base Station (MBS) $\longrightarrow$ ubiquitous coverage
- Several Small-cell Base Stations (SBSs) $\longrightarrow$ network densification (each is co-located with a small-scale server)
- The SBSs are logically interconnected (Can be viwed as an undirected connected graph)
- The HetNet has a unified mobile service provision platform

Microservices and Candidates

- A SFC (app) consists of several microservices (denoted by m)
- Each microservice has several candidates (denoted by c)
- A candidate can be dispatched to multiple SBSs (edge servers), i.e., the instance of the candidate can be successfully started on the chosen edge servers

Motivation Scenarios

Microservices example:

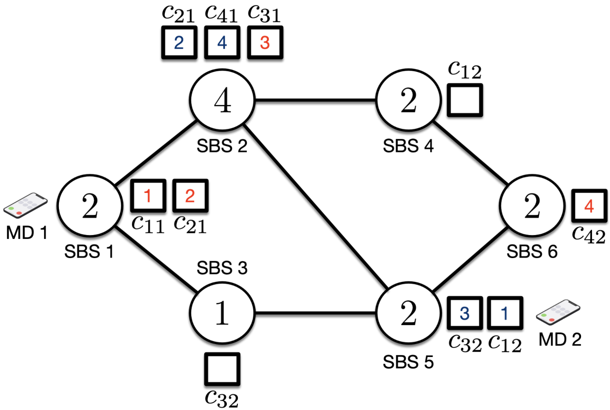


Two service composition schemes for a 4-microservice app.

We actually don't know the end users' service selection results (Thus we have SAA to approximate!).

Motivation Scenarios

A **redundant placement** for the above microservices:



The placement of each candidates on the HetNet.

Our Target

How to judge the goodness of a placement? $\longrightarrow$ **QoS of end users**!

Each mobile device sends its service request to **the nearest** SBS to invoke the first microservice of its SFC.

- If no SBS accessible:

$$\text{mobile device} \Rightarrow \text{MBS} \Rightarrow \text{cloud datacenters}$$

- Otherwise:
  (1) If the request candidate is deployed at the nearest SBS, then process directly;
  (2) If accessible on other SBSs, then processes on these devices with multi-hop transfers;
  (3) Non-accessible on the HetNet, then processed on cloud.

WE SHOULD AVOID BACKBONE TRANSMISSION!

Our Target

How to judge the goodness of a placement? $\longrightarrow$ **QoS of end users**!

For the subsequent microservices, we following one principle:

principle

If the microservice has candidate instance deployed on the HetNet, then process it within the HetNet (multi-hop transfers may required). Otherwise, process it on the cloud by necessity.

WE SHOULD AVOID BACKBONE TRANSMISSION!

We place microservices by minimzing the overall response time of end users.

Our Contributions

- The services we considered are linear (micro)-service function chains (SFCs) with input & output relations
- Each microservice has several candidates, the placement policy we designed also considers "service selection"
- The placement of service instances is based on the "Redundancy" idea: Initialize a microservice instance on multiple geographically distributed edge sites (under their service capacities)
- We take the uncertainty of users' service requests into consideration and simulate it with Sample Average Approximation (SAA)

Outline

Calculating the Response Time

The response time of the i-th mobile device:

$$\tau(E(\boldsymbol{s}(i))) = \sum_{q=1}^{Q} \Big(\tau_{in}(s_q^{c_q}(i)) + \tau_{exe}\big(j_p(s_q^{c_q}(i))\big)\Big) + \tau_{out}(s_Q^{c_Q}(i)).$$

- Q is the number of microservices, $\tau_{in}(\cdot)$, $\tau_{exe}(\cdot)$, $\tau_{out}(\cdot)$ represent the uplink transmission time, execution time, and the downlink transmission time, respectively
- $s_q^{c_q}(i)$ represents the c_q-candidate of the q-th microservice for the i-th mobile device

Calculating the Response Time

How we calculate $\tau_{in}(\cdot)$ and $\tau_{out}(\cdot)$?

It depends on the multi-hop transfers in the connected graph!

Take $\tau_{in}(\cdot)$ as an example —— For the first microservice

$$\tau_{in}(s_1^{c_1}(i)) = \begin{cases} \alpha \cdot d(i,0) + \tau_b, & \mathcal{M}_i = \varnothing; \\ \alpha \cdot d(i,j_i^\star) + \tau_b, & \mathcal{M}_i \neq \varnothing, \mathcal{D}(s_1^{c_1}(i)) = \varnothing; \\ \alpha \cdot d(i,j_i^\star), & \mathcal{M}_i \neq \varnothing, j_i^\star \in \mathcal{D}(s_1^{c_1}(i)); \\ \alpha \cdot d(i,j_i^\star) + \beta \cdot \min_{j^\bullet \in \mathcal{D}(s_1^{c_1}(i))} \zeta(j_i^\star, j^\bullet), & \text{otherwise} \end{cases}$$

For the subsequent microservices:

$$\tau_{in}(s_q^{c_q}(i)) = \begin{cases} 0, & j_p(s_{q-1}^{c_{q-1}}(i)) \in \mathcal{D}(s_q^{c_q}(i)) \text{ or } j_p(s_{q-1}^{c_{q-1}}(i)) = \text{cloud}; \\ \tau_b, & j_p(s_{q-1}^{c_{q-1}}(i)) \neq 0, \mathcal{D}(s_q^{c_q}(i)) = \varnothing; \\ \beta \cdot \min_{j^\bullet \in \mathcal{D}(s_q^{c_q}(i))} \zeta(j_p(s_{q-1}^{c_{q-1}}(i))), j^\bullet), & \text{otherwise} \end{cases}$$

Detailed Formulas...

$$j_p(s_1^{c_1}(i)) = \begin{cases} \text{cloud}, & \mathcal{M}_i = \varnothing \text{ or } \mathcal{D}(s_1^{c_1}(i)) = \varnothing; \\ j_i^\star, & \mathcal{D}(s_1^{c_1}(i)) \neq \varnothing, j_i^\star \in \mathcal{D}(s_1^{c_1}(i)); \\ \text{argmin}_{j^\bullet \in \mathcal{D}(s_1^{c_1}(i))} \zeta(j_i^\star, j^\bullet), & \text{otherwise} \end{cases}$$

$$\tau_{in}(s_1^{c_1}(i)) = \begin{cases} \alpha \cdot d(i,0) + \tau_b, & \mathcal{M}_i = \varnothing; \\ \alpha \cdot d(i, j_i^\star) + \tau_b, & \mathcal{M}_i \neq \varnothing, \mathcal{D}(s_1^{c_1}(i)) = \varnothing; \\ \alpha \cdot d(i, j_i^\star), & \mathcal{M}_i \neq \varnothing, j_i^\star \in \mathcal{D}(s_1^{c_1}(i)); \\ \alpha \cdot d(i, j_i^\star) + \beta \cdot \min_{j^\bullet \in \mathcal{D}(s_1^{c_1}(i))} \zeta(j_i^\star, j^\bullet), & \text{otherwise} \end{cases}$$

$$j_p(s_q^{c_q}(i)) = \begin{cases} \text{cloud}, & \mathcal{M}_i = \varnothing \text{ or } \mathcal{D}(s_q^{c_q}(i)) = \varnothing; \\ j_p(s_{q-1}^{c_{q-1}}(i)), & \mathcal{D}(s_q^{c_q}(i)) \neq \varnothing, j_p(s_{q-1}^{c_{q-1}}(i)) \in \mathcal{D}(s_q^{c_q}(i)); \\ \text{argmin}_{j^\bullet \in \mathcal{D}(s_q^{c_q}(i))} \zeta(j_p(s_{q-1}^{c_{q-1}}(i)), j^\bullet), & \text{otherwise} \end{cases}$$

$$\tau_{in}(s_q^{c_q}(i)) = \begin{cases} 0, & j_p(s_{q-1}^{c_{q-1}}(i)) \in \mathcal{D}(s_q^{c_q}(i)) \text{ or } j_p(s_{q-1}^{c_{q-1}}(i)) = \text{cloud}; \\ \tau_b, & j_p(s_{q-1}^{c_{q-1}}(i)) \neq 0, \mathcal{D}(s_q^{c_q}(i)) = \varnothing; \\ \beta \cdot \min_{j^\bullet \in \mathcal{D}(s_q^{c_q}(i))} \zeta(j_p(s_{q-1}^{c_{q-1}}(i))), j^\bullet), & \text{otherwise} \end{cases}$$

$$\tau_{out}(s_Q^{c_Q}(i)) = \begin{cases} \tau_b + \alpha \cdot d(i,0), & j_p(s_Q^{c_Q}(i)) = \text{cloud}; \\ \beta \cdot \zeta(j_p(s_Q^{c_Q}(i)), j_i^\star) + \alpha \cdot d(i, j_i^\star), & \text{otherwise} \end{cases}$$

Problem Formulation

We want to find an optimal redundant placement policy to minimize the **EXPECTED** overall latency under the limited capability of SBSs:

$$\mathcal{P}_1 : \min_{\mathcal{D}(s_q^{c_q})} \sum_{i=1}^{N} \tau(E(\boldsymbol{s}(i)))$$

Subject to the service capability (maximum deployable instances of each edge servers) limits:

$$\sum_{q \in \mathcal{Q}} \sum_{c \in \mathcal{C}_q} \mathbb{1}\{j \in \mathcal{D}(s_q^{c_q})\} \leq b_j, \forall j \in \mathcal{M},$$

We have expectation operate because we don't know the service selection of end users. Thus we use SAA (a Monte Carlo simulation-based method) to approximate the expectation.

Outline

The SAA-RP Algorithm

With SAA, we use sampling to approximate the expectation (of end users' service selection).

Algorithm 1 SAA-based Redundant Placement (SAA-RP)

1: Choose initial sample size R and R' ($R' \gg R$)
2: Choose the number of replications L (indexed by $\mathcal{L}$)
3: Set up a gap tolerance ϵ
4: **for** l = 1 to L **in parallel do**
5:     Generate R independent samples $\boldsymbol{W}_l^1, ..., \boldsymbol{W}_l^R$
6:     Call GASS to obtain the minimum value of $\hat{g}_R(\boldsymbol{x}_l)$ with the form of $\frac{1}{R}\sum_{r=1}^{R} G(\boldsymbol{x}_l, \boldsymbol{W}_l^r)$
7:     Record the optimal goal $\hat{g}_R(\hat{\boldsymbol{x}}_l^*)$ and the corresponding variable $\hat{\boldsymbol{x}}_l^*$ returned from GASS
8: **end for**
9: $\bar{v}_R^* \leftarrow \frac{1}{L}\sum_{l=1}^{L} \hat{g}_R(\hat{\boldsymbol{x}}_l^*)$
10: **for** l = 1 to L **in parallel do**
11:     Generate R' independent samples $\boldsymbol{W}_l^1, ..., \boldsymbol{W}_l^{R'}$
12:     $v_{R'}^l \leftarrow \frac{1}{R'}\sum_{r'=1}^{R'} G(\hat{\boldsymbol{x}}_l^*, \boldsymbol{W}_l^{r'})$
13: **end for**
14: Get the worst replication $v_{R'}^{\bullet} \leftarrow \max_{l\in\mathcal{L}} v_{R'}^l$
15: **if** the gap $v_{R'}^{\bullet} - \bar{v}_R^* < \epsilon$ **then**
16:     Choose the best solution $\hat{\boldsymbol{x}}_l^*$ among all L replicationss
17: **else**
18:     Increase R (for drill) and R' (for evaluation)
19:     **goto** Step. 4
20: **end if**
21: **return** the best solution $\hat{\boldsymbol{x}}_l^*$

The GASS Subroutine

In the SAA framework, we design a GA-based subroutine to optimize the instance placement decisions.

Algorithm 2 GA-based Server Selection (GASS)

Initialize the population size P, number of iterations $\texttt{it}$, the probability of crossover $\mathbb{P}_c$ and mutation $\mathbb{P}_m$
Randomly generate P chromosomes $\boldsymbol{x}_1, ..., \boldsymbol{x}_P \in \mathcal{X}$
for $t = 1$ to $\texttt{it}$ **do**
  $\forall p \in \{1, ..., P\}$, renew the optimization goal of $\mathcal{P}_2$, i.e. $\hat{g}_R(\boldsymbol{x}_p)$, according to (11)
  for $p = 1$ to P **do**
    if $\texttt{rand}() < \mathbb{P}_c$ **then**
      Choose two chromosomes p_1 and p_2 according to the probability distribution:

$$\mathbb{P}(p \text{ is chosen}) = \frac{1/\hat{g}_R(\boldsymbol{x_p})}{\sum_{p'=1}^{P} 1/\hat{g}_R(\boldsymbol{x_{p'}})}$$

      Randomly choose SBS $j \in \mathcal{M}$
      Crossover the segement of $\boldsymbol{x}_{p_1}$ and $\boldsymbol{x}_{p_2}$ after the partitioning point $\boldsymbol{x}(b_{j-1})$:

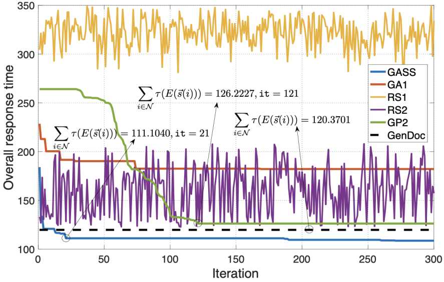
$$[\boldsymbol{x}_{p_1}(b_j), ..., \boldsymbol{x}_{p_1}(b_M)] \leftrightarrow [\boldsymbol{x}_{p_2}(b_j), ..., \boldsymbol{x}_{p_2}(b_M)]$$

    end if
    if $\texttt{rand}() < \mathbb{P}_m$ **then**
      Randomly choose SBS $j \in \mathcal{M}$ and re-generate the segement $\boldsymbol{x}_p(b_j)$
    end if
  end for
end for
return $\mathrm{argmin}_p\, \hat{g}(\boldsymbol{x_p})$ from P chromosomes

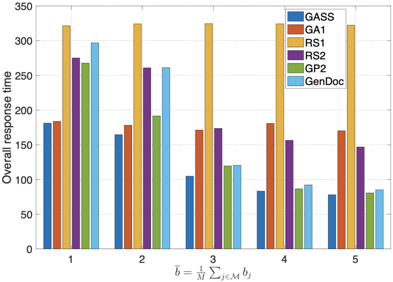
Outline

Experimental Verification

Compared with several benchmark policies, GASS achieves the minimum overall response time.



(a) The convergence rate of all the algorithms with $N = 500$, $M = 40$.



(b) The overall response time vs. $\bar{b}$.