Energy-efficient Task Offloading and Computing Scheme via Hierarchical Reinforcement Learning

Sinwoong Yun

DGIST

lion4656@dgist.ac.kr

10/May/2019

UGRP Meeting

Motivation

- **Energy efficient frequency scheduling**
  - CPU consumes energy with regard to its processing frequency
  - CPU frequency with high level can process task fast
  - However, this consumes more energy than low level frequency
    - Tradeoff!
  - **Find optimal frequency level**
    - Minimize computing energy consumption
    - While satisfying QoS (latency to users)
- **Wired transmission**
  - BSs are connected with wire
    - Distribute a load among BSs to prevent one server from becoming congested
    - 아주 작다 가정하고 생략하였음
    - Wired transmission consumes a few time, energy
  - **Determine wired transmission strategy** via transmission probability

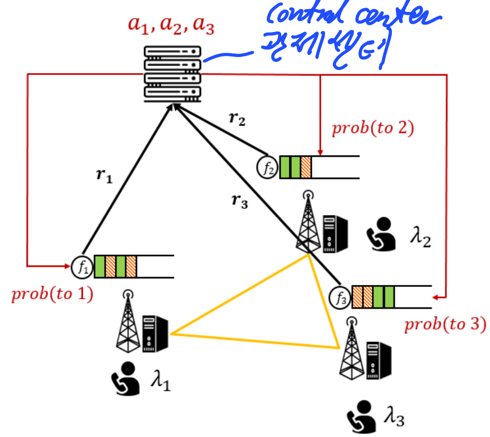
System Model

- **System model**
  - ***K*-users and *N*-servers** are deployed in the area
  - User requests a task at every time slot t
    - Task arrival rate is different for servers
    - Fixed task arrive ▶ need to poisson arrival (plan)

UE→BS

  - Uplink transmission time is random and uniform distribution $U[1,5]$
  - Determine **processor frequency** of each server and **wire transmission**
  - Processing time of the task is invariant when f is same service rate
  - Requested task must be processed before its deadline

control center
관제센터



System Model

- **System model**
  - ***K*-users and *N*-servers** are deployed in the area
  - User requests a task at every time slot t
    - Task arrival rate is different for servers
    - Fixed task arrive ▶ need to poisson arrival (plan)
  - Uplink transmission time is random and uniform distribution $U[1,5]$
  - Determine **processor utilization** of each server and **wire transmission**
  - Processing time of the task is invariant when f is same
  - Requested task must be processed before its deadline

System Model

- **Hierarchical reinforcement learning**
  - Separate the problem into two problems
    - Server side : schedule CPU frequency
    - Center side : determine wire transmission strategy
  - Center controls servers and each server has its own small problem
- **Processor frequency scaling (f_i^t) by DVFS (server)**
  - Each server i processes the task with utilization f_i^t at time step t
  - $0 \le f_i^t \le 1,\ \forall i, t$
  - Power consumption at t : $P_i^t = c_{0,i} + c_{1,i}\left(f_i^t\right)^3$
- **Wire transmission strategy (center)**
  - $\pi^t = \{a_1, a_2, a_3\},\ \forall t$
  - Softmax : Wire transmission probability to BS i is $\mathbb{P}(\textbf{to } \boldsymbol{BS}\ \boldsymbol{i}) = \frac{\boldsymbol{e^{a_i}}}{\sum_{\boldsymbol{j \in BS}} \boldsymbol{e^{a_j}}}$

RL formulation

Transmission

State : $[\#(\overrightarrow{BS_i, type_h}), \overrightarrow{Queue(BS_i)}]$

Action : $[a_1^t, a_2^t, a_3^t] \in [0,1] \ \rightarrow \mathbb{P}(\text{to } BS\ i) = \frac{e^{a_i}}{\sum_{j \in BS} e^{a_j}}$

Reward : $f(r_1^t, r_2^t, r_3^t, E_{wired})\ (e.g., r_1^t + r_2^t + r_3^t + \frac{10}{E_{wired}})$

Frequency of BS 1

State : $[Queue(BS_1)^t]$

Action : $[f_1^t] \in [0,1]$

Reward : $r_1^t = f(E_1^t)\ \left(e.g., \frac{50}{E_1^t}\right)$

$$r_1^{t_{end}} = f(E_1^{t_{end}}) + \sum_{k \in tasks} p_{1k}$$

$$p_{1k} = \begin{cases} -10\ (time\ over) \\ +1\ (otherwise) \end{cases}$$

$\bullet\bullet\bullet$

Frequency of BS N

State : $[Queue(BS_1)^t]$

Action : $[f_1^t] \in [0,1]$

Reward : $r_1^t = f(E_1^t)\ \left(e.g., \frac{50}{E_1^t}\right)$

$$r_1^{t_{end}} = f(E_1^{t_{end}}) + \sum_{k \in tasks} p_{1k}$$

$$p_{1k} = \begin{cases} -10\ (time\ over) \\ +1\ (otherwise) \end{cases}$$

Gym environment structure

Class : Environment

필수구현

__init__	Initialize parameters, action/state space
reset	Reset reward, time slot, server state (Central/Server)
step	$\mathcal{S}, \mathcal{A} \rightarrow \mathcal{S}', \mathcal{R}, done$ 끝내기! (Central/Server)

구현!

usergen	Generate task request for each time slot
wire_transmission	Decide wire transmission for multiple tasks
is_done	Return central reward and done flag

+α.

seed	Setting random seed to reproduce the same result (not use)
render	Display the environment (not use)

Gym environment structure

Class : Server

__init__	Initialize parameters, server queue, uplink tasks
association	BS-user association with random uplink transmission time
arri_pop	Pop one task from arrived tasks (waiting wired transmission)
wait_update	Give time count to tasks
add_to_queue	Add transmitted task to server computing queue
process	Process the tasks with scheduled frequency

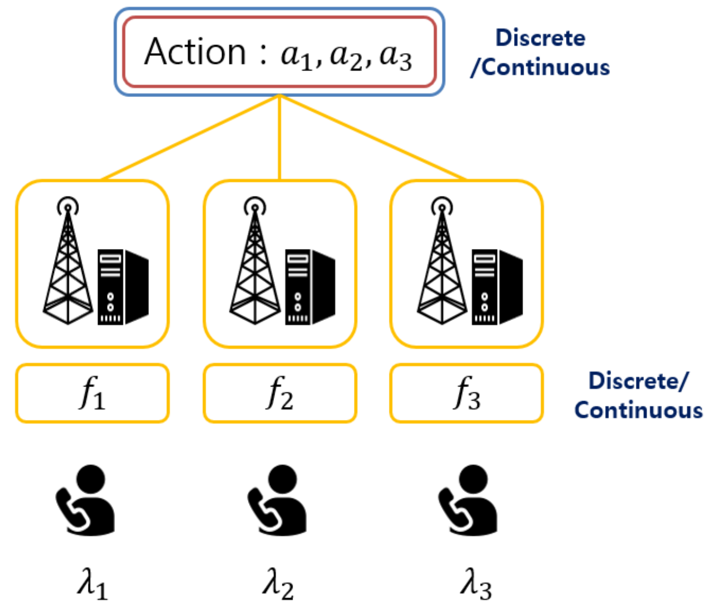
Total structure

Class : Environment

__init__	Initialize parameters, action/state space
reset	Reset reward, time slot, server state (Central/Server)
step	$\mathcal{S}, \mathcal{A} \rightarrow \mathcal{S}', \mathcal{R}, done$ (Central/Server)
usergen	Generate task request for each time slot
wire_transmission	Decide wire transmission for multiple tasks
is_done	Return central reward and done flag
seed	Setting random seed to reproduce the same result (not use)
render	Display the environment (not use)

Class : Server

__init__	Initialize parameters, server queue, uplink tasks
association	BS-user association with random uplink transmission time
arri_pop	Pop one task from arrived tasks (waiting wired transmission)
wait_update	Give time count to tasks
add_to_queue	Add transmitted task to server computing queue
process	Process the tasks with scheduled frequency



Env code - environment

- __init__ (self)

변수 범위 class 內

```
class SWEnv_v3(gym.Env):
    metadata = {'render.modes':['human']}
    def __init__(self):
        self.gen_iter = 40
        self.beta = 0.5
        #Data type은 data size(kb)와 computation cycle(Gb)로 구성되어있음.
        self.type_info  = np.array([[5,15.0],[5,20.0]]) # [maximum uplink transmission time, data size ]
        self.num_type   = 2
        self.num_BS     = 3
        self.max_freq   = 10
        self.token      = 0
        self.has_input  = 0
        self.observation_space_trans = spaces.Box(low=-1, high=500, shape=[6], dtype=np.float32)
        self.action_space_trans = spaces.Box(low=-1, high=1, shape=(3,), dtype=np.float32)
        self.observation_space_freq = spaces.Box(low=-1, high=500, shape=[1], dtype=np.float32)
        self.action_space_freq = spaces.Box(low=-1, high=1, shape=(1,), dtype=np.float32)
        self.tick=100;self.tot_reward=0;
```

변수 범위 def 內

state action reward를 결정하는 변수

Env code - environment

- reset (self, agent = -1)

```
def reset(self, agent=-1):
    if agent == -1:
        self.avg_reward=self.tot_reward/self.tick
        self.tick = 0
        self.state =  np.array([0.,0.,0.,0.,0.,0.])
        self.init_state =  np.array([0.,0.,0.,0.,0.,0.])
        self.init_action = np.array([0.,0.,0.])
        self.BS_set=[server(), server(), server()]
        self.done = False
        self.server_done = [False, False, False]
        self.server_energy_reward = np.array([0,0,0])
        self.total_deadline_reward = np.array([0,0,0])
        self.wire_trans_energy = 0
        state, _, _, _ = self.step(self.init_action,-1)
        return state
```

class에서 server class 3개를 호출.

이를 행렬로 연결되는 부분

center

```
    else:
        state, _, _, _ = self.step(self.BS_set[agent].init_action, agent)
        return state
```

server

Env code - environment

- step (self, action, agent = -1) : center agent (agent == -1)

```
def step(self,action,agent=-1):
    act = np.clip(action+1, 0, 2)/2
    #agent는 -1일 때 central unit이 transmission 결정해주는 것.(transmission action)
    if agent == -1:
        if self.tick < self.gen_iter:
            self.usergen()
        # action은 choice이다.
        # wired transmission 필요한 경우.
        if np.sum(self.state[:self.num_BS] > 0):
            offload_prob = np.exp(act)/np.sum(np.exp(act), axis=0)
            to_BS = np.random.choice(self.num_BS, size=np.sum(self.state[:self.num_BS]), p=offload_prob) #wired_transmission to BS
            wire_task_num = np.bincount(to_BS, minlength=3)
            # array [1,3,4] -> 0번 서버에 1개, 1번 서버에 3개, 2번 서버에 4개 task가 들어갈 예정.
            arri_task_num = np.array(self.state[:self.num_BS])
            # [2,4,2] -> 각 서버에 2개, 4개, 2개의 task가 도착해있음.
            wire_trans_det = arri_task_num - wire_task_num
            # [1,-3,2] ->이 값이 +인 서버에서 -인 애들한테 task를 보내주고, wired transmission energy는 그 개수만큼.
             self.wire_trans_energy = self.wire_transmission(wire_trans_det)
        else:
            self.wire_trans_energy = 0

        for i in range(self.num_BS):
            self.BS_set[i].wait_update()
            self.state[i] = len(self.BS_set[i].arri_task)
            self.state[i + self.num_BS] = np.sum(self.BS_set[i].wait_task[:,0])

        self.tick += 1
        reward, self.done = self.is_done(reward)

        return self.state, reward, self.done, 0
        # return self.state, reward-self.avg_reward, self.done, 0
```

Env code - environment

- step (self, action, agent = -1) : server agent (agent != -1)

```
#agent가 0,1,.. 일 때는 해당 BS가 process하는 것. (frequency action)
else:
    #action은 relative frequency이다.
    energy_reward,deadline_reward = self.BS_set[agent].process(act)
    self.total_deadline_reward[agent] += deadline_reward
    self.server_energy_reward[agent] = reward #reward값을 저장해뒀다가 후에 central reward 계산에 가져다 쓰려고 만듦.
    reward = energy_reward

    self.BS_set[agent].state = np.sum(self.BS_set[agent].wait_task[:,0])
    if len(self.BS_set[agent].sent_task) == 0 and len(self.BS_set[agent].wait_task) == 0 and (self.BS_set[agent].arri_task) == 0 :
        self.server_done[agent] = True
        reward += self.total_deadline_reward[agent]

    return self.BS_set[agent].state, reward, self.server_done[agent], 0
```

- usergen (self)

```
#usergen every step with different rate for server
def usergen(self):
    #first, fixed datas
    usergen_rate = [1,2,3]
    for i in range(self.num_BS):
        type_indicator = np.random.rand(usergen_rate[i])
        user_task_type = type_indicator < 0.5
        self.BS_set[i].association(user_task_type * 1, usergen_rate[i])
```

Env code - environment

- wire_transmission (self, trans_det)

```
def wire_transmission(self, trans_det):
    wire_energy = np.sum(np.abs(trans_det))/2 # wired transmission energy consumption
    # wired transmission 끝날때까지 위 과정 반복해줌.
    while trans_det != [0,0,0]:
        Tx_candidate = np.where(trans_det > 0)[0]
        Rx_candidate = np.where(trans_det < 0)[0]
        wire_Tx = np.random.choice(Tx_candidate)
        wire_Rx = np.random.choice(Rx_candidate)

        wire_trans_task = self.BS_set[wire_Tx].arri_pop()
        self.BS_set[wire_Rx].add_to_queue(wire_trans_task)

        trans_det[wire_Tx] -= 1
        trans_det[wire_Rx] += 1
    return wire_energy
```

Env code - environment

- is_done (self, reward)

```python
#central reward, done
def is_done(self, reward):
    #reward = r1 + r2 + r3 + wired_trans_energy
    if self.tick > 100:
        reward = -100
        print("GAME OVER")
        self.done = True
    #reward = r1 + r2 + r3 + wired_trans_energy
    if False in server_done:
        reward = 10/(self.wired_trans_energy + 1)
        for i in range(self.num_BS):
            reward += self.BS_set[i].energy_reward if (self.server_done[i] == False)
        self.done = False
    else:
        reward = 10/(self.wired_trans_energy + 1)
        reward = reward + np.sum(self.total_deadline_reward)
        self.done = True
    return reward, self.done
```

episode가 너무 길어지게 끊고 일정의 reward를 기록.

Env code - server

- __init__ (self)

```
class server():
    def __init__(self):
        self.type_info  = np.array([[3,20.0,13],[3,20.0,13]]) # [transmission size, computation size, deadline]
        self.sent_task = np.array([]).reshape(-1, 3) # ul 중인거 [data type, elapse time, arrive counter]
        self.wait_task = np.array([]).reshape(-1, 3) # task in server queue [left calc size, data type, elapse time]
        self.arri_task = np.array([]).reshape(-1, 2) # ul 후 wired 기다리는거 [data type, elapse time]
        self.freq_max = 10
        #아래 state와 action은 각각의 server에 대한 frequency 조절에 대한 것.
        self.state =  np.array([0.])
        self.init_state =  np.array([0.])
        self.init_action = np.array([-1.])
        #self.wait_Q = []
        #self.wait_T = np.array([]).reshape(-1, 3)
        self.noise=10**-11
        self.theta=1
        self.UL_const=np.array([3, 5])
        self.deadline = 13
        self.energy_reward = 0
        self.total_time_reward = 0
```

- association (self, user_types, length)

```
def association(self,user_types, length):
    elapse_time = np.random.randint(5,size=length) + 1
    user_waiting=np.array([user_types.reshape(-1), elapse_time, elapse_time]).T
    self.sent_task = np.vstack([self.sent_task, user_waiting]).reshape(-1, 3)
```

- arri_pop (self)

```
def arri_pop(self):
    pop_task = self.arri_task[0]
    self.arri_task=np.delete(self.arri_task, 0, axis=0)
    return pop_task
```

Env code - server

- wait_update (self)

```
#매 time step uplink 끝난 거를 sent에서 arri로 옮기고, wait task들에게 매 step 시간이 흘렀음을 알린다.
def wait_update(self):
    if len(self.sent_task)>0:
        self.sent_task[:, 2]-=1
        arrive = np.array(np.where(0>=self.sent_task[:,2])).reshape(-1)
        self.arri_task = np.vstack([self.arri_task, self.sent_task[arrive, 0:2].reshape(-1, 2)]).reshape(-1, 2)
        self.sent_task = np.delete(self.sent_task, arrive, axis=0)
    if len(self.wait_task)>0:
        self.wait_task[:,2]+=1
```

- add_to_queue (self, input_task)

```
def add_to_queue(self, input_task): # input_task contain [data type, time]
    data_type = self.type_info[int(input_task[0])]
    data_size = data_type[1]
    task = np.append(data_size, input_task) #task는 [left calc size, data type, time]로 구성.
    # task[2]+=1 이거 왜 더하지? 앞에서 안 더하나? 확인 필요.
    self.wait_task = np.vstack([self.wait_task, task.reshape(-1, 3)]).reshape(-1, 3)
```

Env code - server

- process (self, relative_frequency)

```
def process(self, relative_frequency):
    deadline_reward = 0
    left_resource = self.freq_max * relative_frequency
    cpu_energy = 13.5 + 22.7 * (relative_frequency**3)
    self.energy_reward = 50/cpu_energy
    while len(self.wait_task)!=0 and left_resource!=0:
        current_task = self.wait_task[0]
        left_data = self.wait_task[0,0]
        if left_data <= left_resource:
            deadline_reward += 1 if (self.wait_task[0,2] <= self.deadline) else deadline_reward -= 10
            self.wait_task = np.delete(self.wait_task, 0, axis=0)
            left_resource -= left_data
            #print("left resource is {}, data is {}".format( left_resource, left_data))
        else:
            self.wait_task[0, 0]-=left_resource
            left_resource=0

    return  self.energy_reward, deadline_reward
```

Any Questions?

THANK YOU

Email : lion4656@dgist.ac.kr