GEdge(Griffin-Edge) Platform
초저지연 지능형 클라우드 엣지 SW 플랫폼 -

강화학습 기반 멀티엣지 협업 정책 생성 기술

2022.12.20

GS-Link 프레임워크 코어개발자(GS-Linkhq)

윤주상 (joosang.youn@gmail.com)

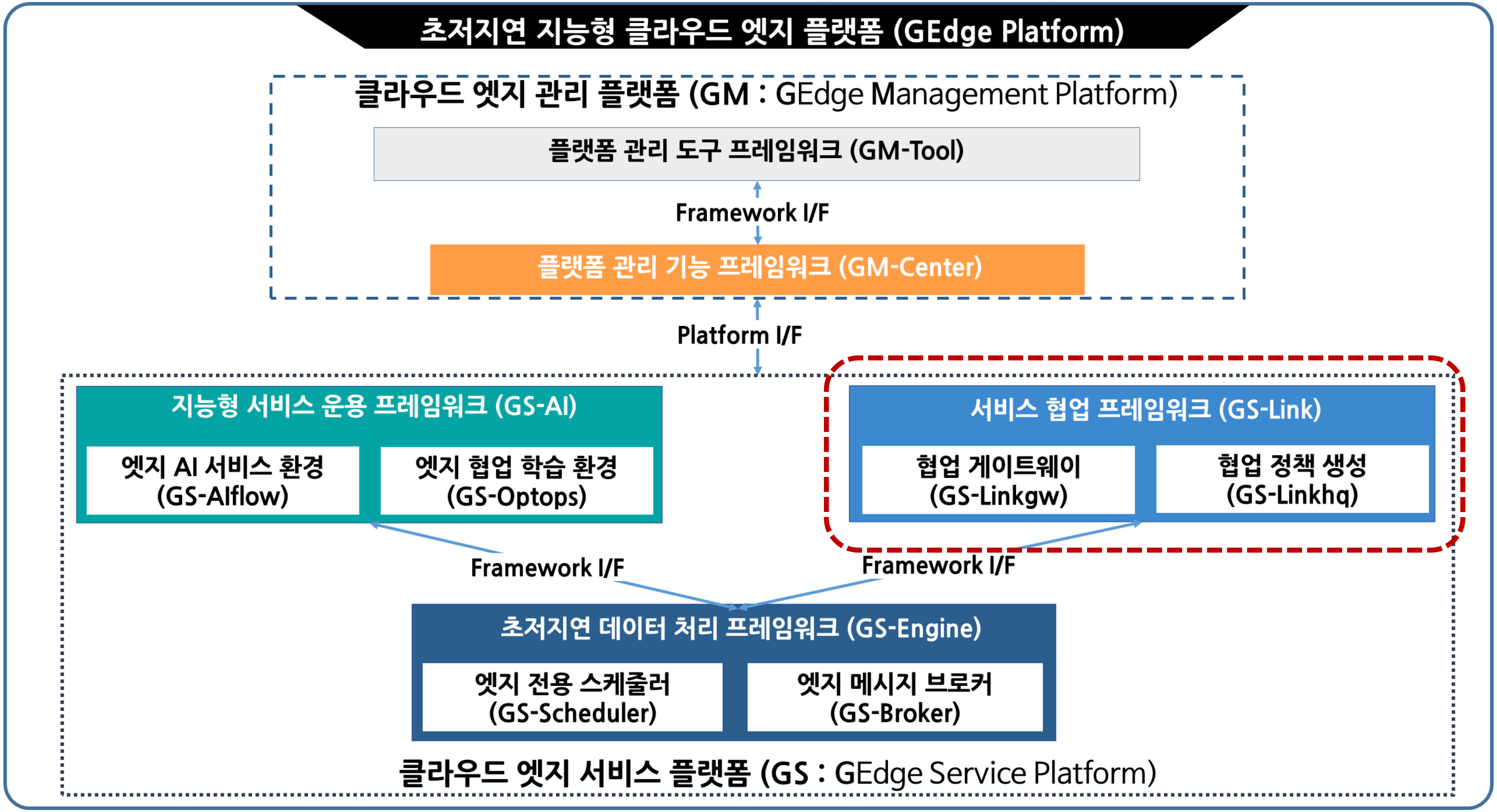
"GEdge Platform" 은 클라우드 중심의 엣지 컴퓨팅 플랫폼을 제공하기 위한 핵심 SW 기술 개발 커뮤니티 및 개발 결과물의 코드명입니다.

- Developer-Driven

GEdge Platform Community 5th Conference (GEdge Platform v3.0 Release) -

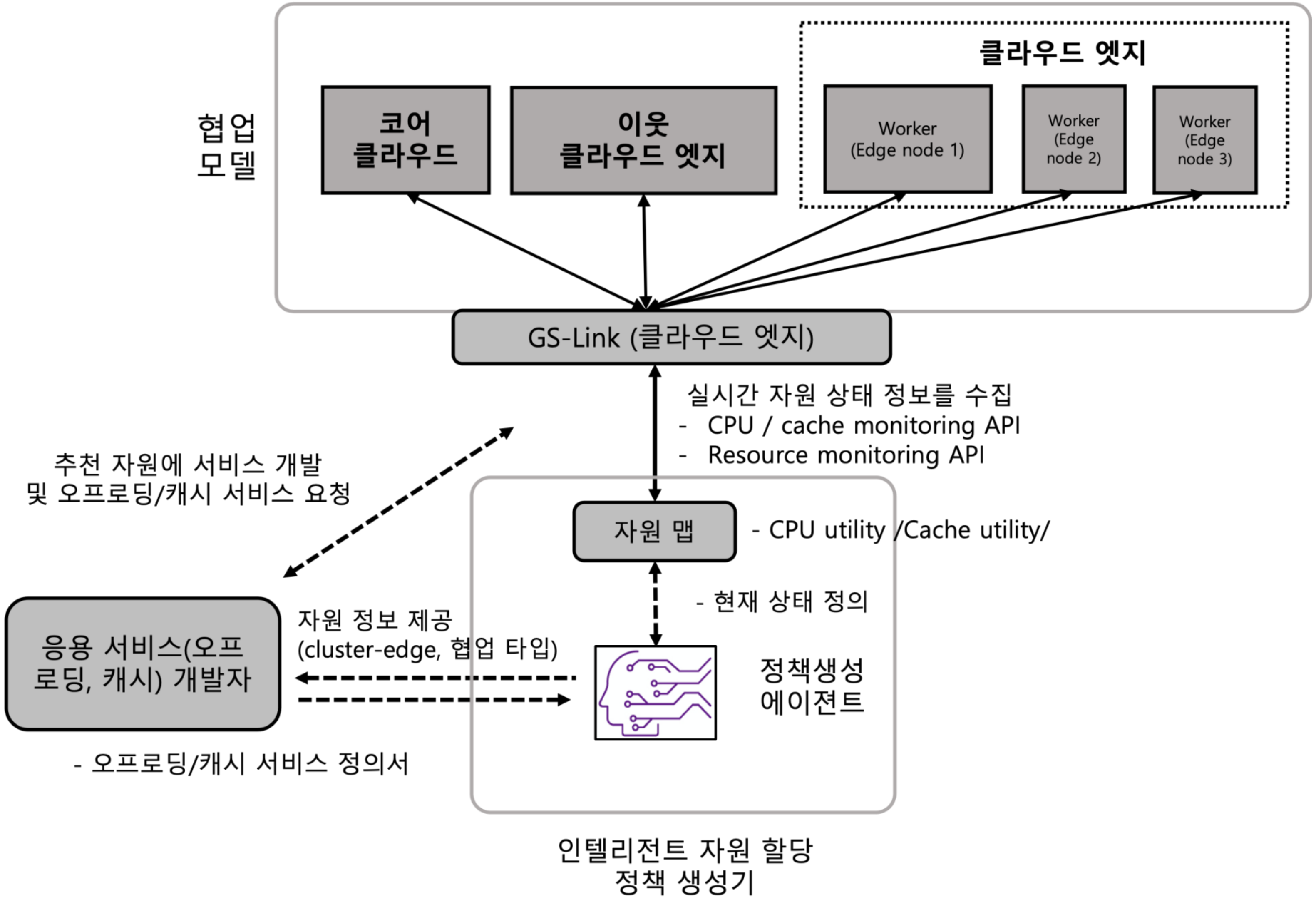
Contents

GEdge 플랫폼 내 Linkhq의 포지셔닝



Ⅰ 강화학습 기반 지능형 오프로딩 정책 생성 기술

1 정책 생성 및 제공 과정

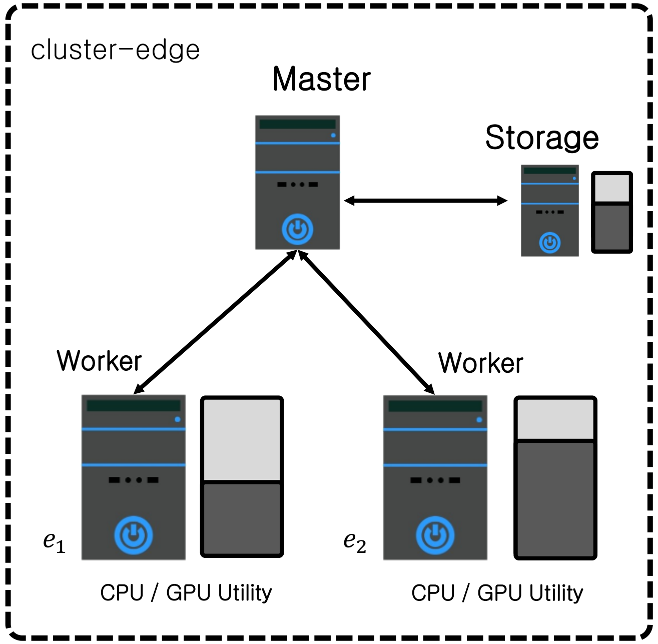

협업
모델
코어
클라우드
이웃
클라우드 엣지
클라우드 엣지
Worker
(Edge node 1)
Worker
(Edge
node 2)
Worker
(Edge
node 3)
GS-Link (클라우드 엣지)
실시간 자원 상태 정보를 수집
- CPU / cache monitoring API
- Resource monitoring API
추천 자원에 서비스 개발
및 오프로딩/캐시 서비스 요청
자원 맵
- CPU utility /Cache utility/
- 현재 상태 정의
자원 정보 제공
(cluster-edge, 협업 타입)
응용 서비스(오프
로딩, 캐시) 개발자
정책생성
에이전트
- 오프로딩/캐시 서비스 정의서
인텔리전트 자원 할당
정책 생성기

2 자원 할당 정책 및 협업 방법

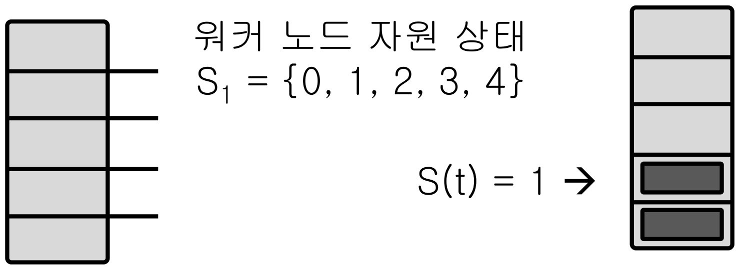
- 자원 할당을 위한 정책
  - Random, Least Load, Round-Robin : 일반적인 스케줄링 기술
  - Rule : 수직(클라우드-엣지 간) 협업 경우 사용률에 따라 클라우드 자원 추천
    - 특정 자원 필요 시 클라우드 자원 선택
  - **DQN : 단일 자원 선택을 위한 심층강화학습 기반 정책 생성**
    - **자원 선택 시 최적의 자원 1개를 선택하여 오프로딩 수행**
  - **DDPQ : 분산 자원 선택을 위한 심층강화학습 기반 정책 생성**
    - **자원 선택 시 여러 자원을 선택하여 분산 오프로딩 수행**

3 클라우드 엣지 내 자원 정의

- 클라우드-엣지 컴퓨팅 자원 모델

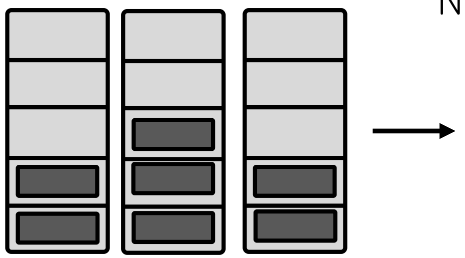


워커 노드 자원
상태 정의 (5−level)



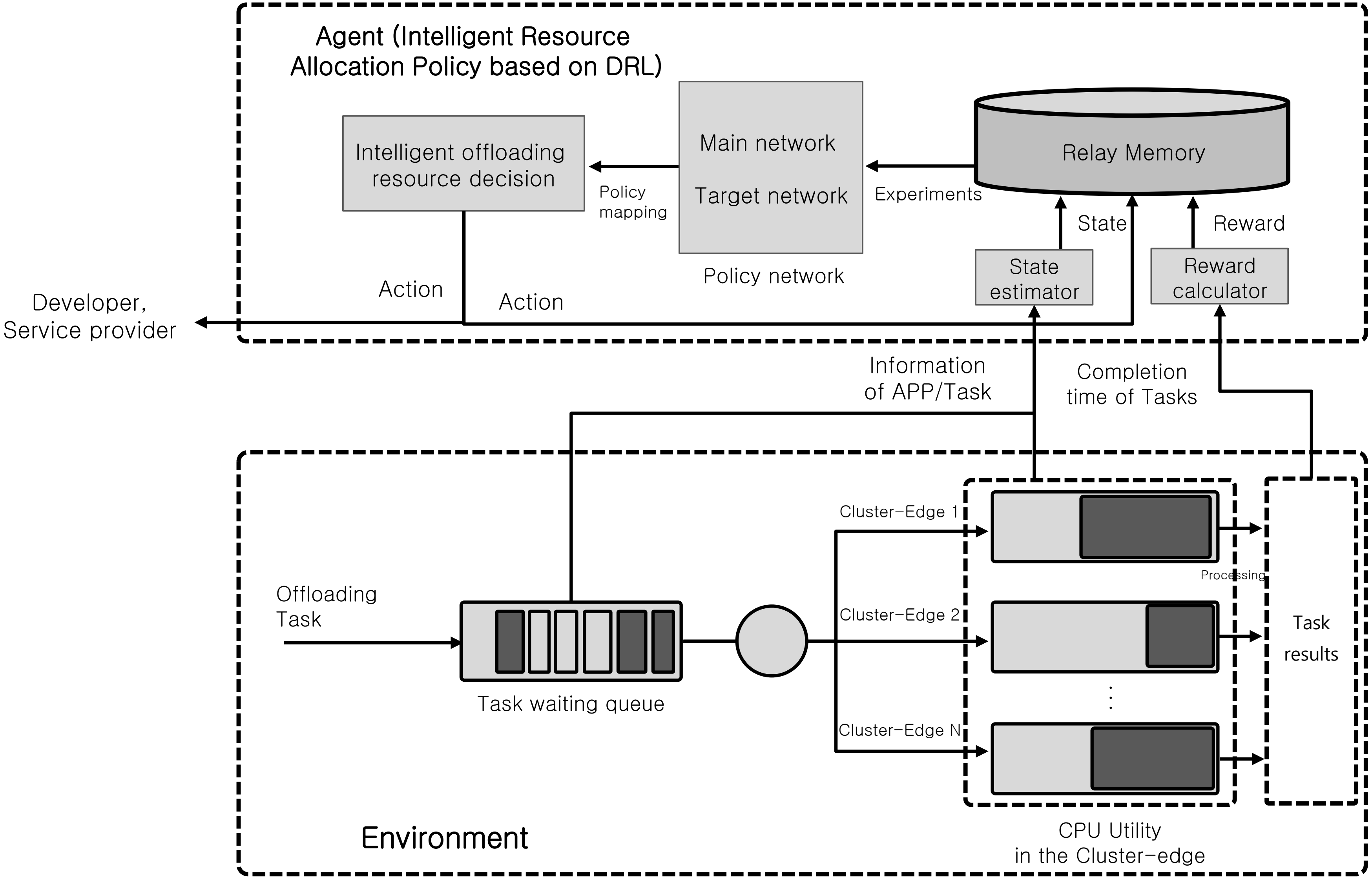
자원 상태 모델
3 workers 경우

$S = \{0, 1, 2, 3, 4, \cdots N\}$
$N = 5*5*5 = 125-1 = 124$



$S(t) = (S_1, S_2, S_3)$
$= (0, 0, 0) = 0$
$= (0, 0, 1) = 1$
$= (0, 0, 2) = 2$
…..

DQN 기반 단일 자원 선택 정책 생성 구조

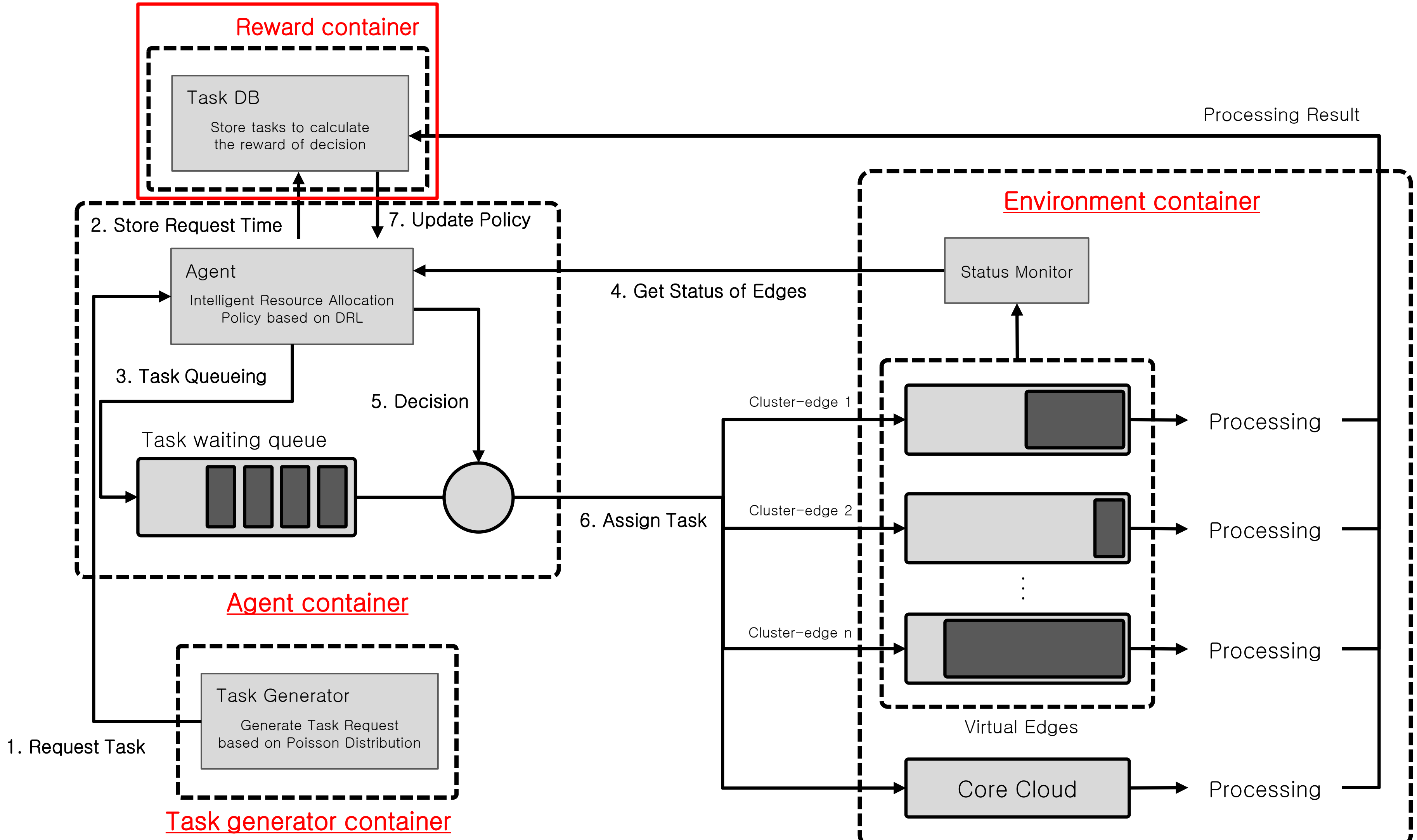


5 Virtual G-Edge Model – 4개 컨테이너로 구성

GEdge Platform

- 정책 학습을 위한 환경 구현
- 구성 요소
  - 정책 에이전트
  - 태스크 발생기
  - 환경
  - 리워드

  - **<u>컨테이너로 구현</u>**



6 Agent – 구현 (DQN 기반)

DQN 신경망

```python
class Brain:
    def __init__(self, num_states, num_actions):
        self.num_actions = num_actions

        self.memory = ReplayMemory(CAPACITY)

        self.model = nn.Sequential()
        self.model.add_module('fc1', nn.Linear(num_states, 32))
        self.model.add_module('relu1', nn.ReLU())
        self.model.add_module('fc2', nn.Linear(32, 32))
        self.model.add_module('relu2', nn.ReLU())
        self.model.add_module('fc3', nn.Linear(32, num_actions))

        self.optimizer = optim.Adam(self.model.parameters(), lr=0.0001)

    def replay(self):
        if len(self.memory) < BATCH_SIZE:
            return

        transitions = self.memory.sample(BATCH_SIZE)

        batch = Transition(*zip(*transitions))

        state_batch = torch.cat(batch.state)
        action_batch = torch.cat(batch.action)
        reward_batch = torch.cat(batch.reward)
        non_final_next_states = torch.cat([s for s in batch.next_state if s is
not None])

        self.model.eval()

        state_action_values = self.model(state_batch).gather(1, action_batch)
```

```python
        non_final_mask = torch.ByteTensor(tuple(map(lambda s: s is not None,
batch.next_state)))

        next_state_values = torch.zeros(BATCH_SIZE)

        next_state_values[non_final_mask] =
self.model(non_final_next_states).max(1)[0].detach()

        expected_state_action_values = reward_batch + GAMMA * next_state_values

        self.model.train()

        loss = F.smooth_l1_loss(state_action_values,
expected_state_action_values.unsqueeze(1))

        self.optimizer.zero_grad()
        loss.backward()
        self.optimizer.step()

    def decide_action(self, state, episode):
        epsilon = 0.5 * (1 / (episode + 1))

        if epsilon <= np.random.uniform(0, 1):
            self.model.eval()
            with torch.no_grad():
                action = self.model(state).max(1)[1].view(1, 1)

        else:
            action = torch.LongTensor(
                [[random.randrange(self.num_actions)]])

        return action
```

7 Agent – 구현 (DQN 기반)

```python
class ReplayMemory:
    def __init__(self, CAPACITY):
        self.capacity = CAPACITY
        self.memory = []
        self.index = 0

    def push(self, state, action, state_next, reward):

        if len(self.memory) < self.capacity:
            self.memory.append(None)

        self.memory[self.index] = Transition(state, action, state_next, reward)

        self.index = (self.index + 1) % self.capacity

    def sample(self, batch_size):
        return random.sample(self.memory, batch_size)

    def __len__(self):
        return len(self.memory)
```

리플레이 메모리

```python
class DQNAgent:
    def __init__(self, num_states, num_actions):
        self.step = 0
        self.episode = 0
        self.brain = Brain(num_states, num_actions)

    def update_q_function(self):
        self.brain.replay()

    def get_action(self, state):
        action = self.brain.decide_action(state, self.episode)
        self.step += 1
        return action

    def memorize(self, state, action, state_next, reward):
        self.brain.memory.push(state, action, state_next, reward)

    def reset(self):
        self.step = 0
        self.episode += 1
```

DQN 학습 Agent

8 Task Generator - 태스크 요청 API 코드

```
class Task(Resource):
    def post(self):
        try:
            data = request.get_json(force=True)

            if type(data) is not dict:
                raise BadRequest
            elif 'task' not in data.keys():
                raise BadRequest
            elif type(data['task']) is not dict:
                raise BadRequest
            elif not {'size', 'cycle'}.issubset(data['task'].keys()):
                raise BadRequest

            task = data['task']
            size = task['size']
            cycle = task['cycle']

            if type(size) is not int or type(cycle) is not int:
                raise BadRequest

            task_id = r.incr('task-index')

            r.hset(task_prefix + str(task_id), 'size', size)
            r.hset(task_prefix + str(task_id), 'cycle', cycle)
            r.hset(task_prefix + str(task_id), 'time', time.time())

            # TODO Allocate task to agent
            res = requests.get(env_url + '/state')
            data = json.loads(res.json())

            state = []
            for edge in data['edges']:
                state.append(len(edge['queue']))

            state = torch.FloatTensor(state)
            state = torch.unsqueeze(state, 0)
```

요청: 태스크 총 사이즈,
CPU 클럭

```
            action = agent.get_action(state)

            pve_logger.critical(action)


            ret = {
                "id": task_id,
                "message": "Successfully created"
            }


            return json.dumps(ret, indent=4), 201

        except BadRequest:
            ret = {
                'errors': [
                    {
                        'message': 'Invalid task format'
                    }
                ]
            }

            return json.dumps(ret, indent=4), 400

        except Exception as e:
            pve_logger.error(e)

            ret = {
                'errors': [
                    {
                        'message': 'Internal Server Error'
                    }
                ]
            }

            return json.dumps(ret, indent=4), 500
```

9 Task Generator - 태스크 완료 API 코드

```python
class DelTask(Resource):
    def delete(self, task_id):
        try:
            if r.delete(task_prefix + task_id):
                ret = {
                    "id": task_id,
                    "message": "Successfully deleted"
                }
                return json.dumps(ret), 200
            else:
                ret = {
                    'errors': [
                        {
                            'id': task_id,
                            'message': 'Task not found'
                        }
                    ]
                }
                return json.dumps(ret), 404

        except Exception as e:
            pve_logger.error(e)
            ret = {
                'errors': [
                    {
                        'message': 'Internal Server Error'
                    }
                ]
            }
            return json.dumps(ret), 500
```

태스크 완료 후 → 리워드 함수로 전달
(액션에 대한 평가)

10 Environment – 상태 수집 API 코드

클라우드 엣지 상태 모니터링 기능

```
class State(Resource):
    def get(self):
        state = env.state()

        ret = {
            'edges': state
        }

        return json.dumps(ret, indent=4), 200
```

11 Environment – 가상 리소스 코드

```python
class Edge:
    def __init__(self, name: str, cpu_frequency: float = 1.0):
        self.name = name
        self.cpu = CPU(cpu_frequency)
        self.task_list = []

    def get_cpu_frequency(self):
        return self.cpu.get_frequency()

    def get_queue(self):
        return self.cpu.get_queue()

    def reset(self):
        self.cpu.reset()

    def state(self):
        ret = {}
        ret['name'] = self.name
        ret['frequency'] = self.get_cpu_frequency()
        ret['queue'] = self.get_queue()
        return ret

    def assign(self, task):
        return self.cpu.assign(task)

    def update(self, dt):
        return self.cpu.update(dt)
```

Edge 정의

```python
class CPU:
    def __init__(self, frequency):
        self.frequency = frequency
        self.reset()

    def get_queue(self):
        return self.q

    def state(self):
        self.q.sort()
        result = []
        result = result + self.q

        return result

    def get_frequency(self):
        return self.frequency

    def assign(self, task):
        self.q.sort()

        if self.q[0] <= 0:
            self.q[0] = task[1] / self.frequency
        else:
            self.q[0] = self.q[0] + (task[1] / self.frequency)

        return self.q[0]

    def reset(self):
        self.q = []

    def update(self, dt):
        for i in range(len(self.q)):
            new_ts = self.q[i] - dt

            if new_ts <= 0:
                new_ts = 0

            self.q[i] = new_ts
```

Edge CPU
→ GPU 동일하게 구현

Environment – 자원 할당 API 코드

태스크 자원 할당 결정 후
→ 환경 내 자원 할당

```
class Task(Resource):
    def post(self, edge_id):
        try:
            data = request.get_json(force=True)

            if type(data) is not dict:
                raise BadRequest
            elif 'task' not in data.keys():
                raise BadRequest
            elif type(data['task']) is not dict:
                raise BadRequest
            elif not {'id', 'size', 'cycle'}.issubset(data['task'].keys()):
                raise BadRequest

            task = data['task']
            task_id = task['id']
            size = task['size']
            cycle = task['cycle']

            if type(task_id) is not int or type(size) is not int or type(cycle)
is not int:
                raise BadRequest

            ret = {
                "id": task_id,
```

성공

```
                "message": "Successfully created"
            }

            return json.dumps(ret, indent=4), 201

        except BadRequest:
            ret = {
                'errors': [
                    {
                        'message': 'Invalid task format'
                    }
                ]
            }

            return json.dumps(ret, indent=4), 400

        except Exception as e:
            pve_logger.error(e)
            ret = {
                'errors': [
                    {
                        'message': 'Internal Server Error'
                    }
                ]
            }

            return json.dumps(ret, indent=4), 500
```

실패

에러

13 Environment – 가상 환경 구현 코드

GEdge Platform

```python
class VGEdge:
    def __init__(self, conf_path="../config.yaml"):
        self.edges = []
        self.space = []
        self.episode = 0
        self.t = 0
        self.reward = 0
        self.rewards = []
        self.conf = get_conf(conf_path)
        for edge in self.conf['edges']:
            self.edges.append(Edge(edge['name'], edge['frequency']))

    def reset(self):
        self.rewards.append(self.reward)
        self.episode += 1
        self.reward = 0
        for edge in self.edges:
            edge.reset()

        self.space = self.state()

        return np.array(self.space).astype(np.float32)

    def act(self, task, action):
        if action not in range(len(self.edges)):
            raise ValueError("Received invalid action={} which is not part of
the action space".format(action))

        self.edges[action].assign(task)

    def step(self, task, action):
        if task:
            self.act(task, action)
```

```python
        for edge in self.edges:
            edge.update(1)

        reward = 0
        done = False

        self.t += 1

        return self.state(), reward, done

    def state(self):
        self.space = []
        for edge in self.edges:
            self.space.append(edge.state())
        return self.space

    def render(self):
        print(self.t)
        for edge in self.edges:
            print(edge.name, edge.state())
        print()

    def get_conf(self):
        return self.conf

    def get_len_state(self):
        len_state = 0
        for edge in self.edges:
            len_state += edge.get_queue()
        return len_state

    def get_num_actions(self):
        return len(self.edges)
```

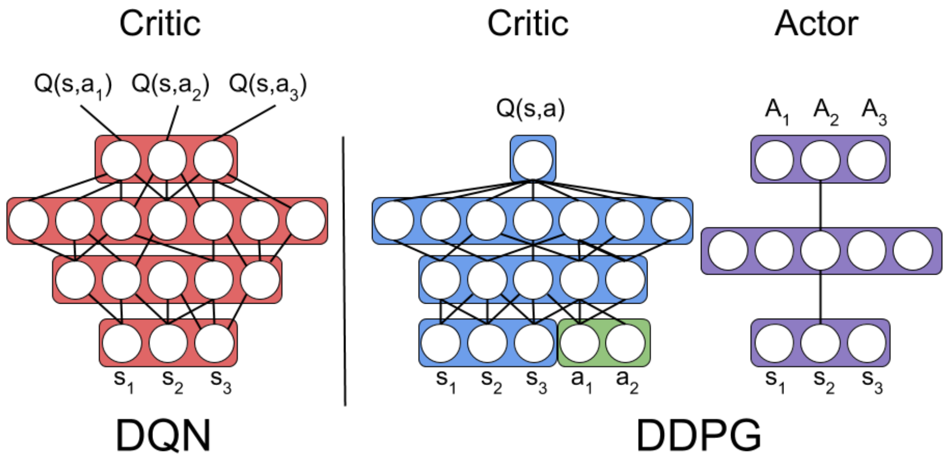
14 액션 생성 결과

- 네트워크 및 엣지 자원 모델
  - 클러스터-엣지 5개 (1개 워커로 구성)
  - CPU 클럭은 1.0Ghz
  - 태스크
    - 10M
    - CPU 요구 : 3GHz
    - 태스크 총 길이는 가변

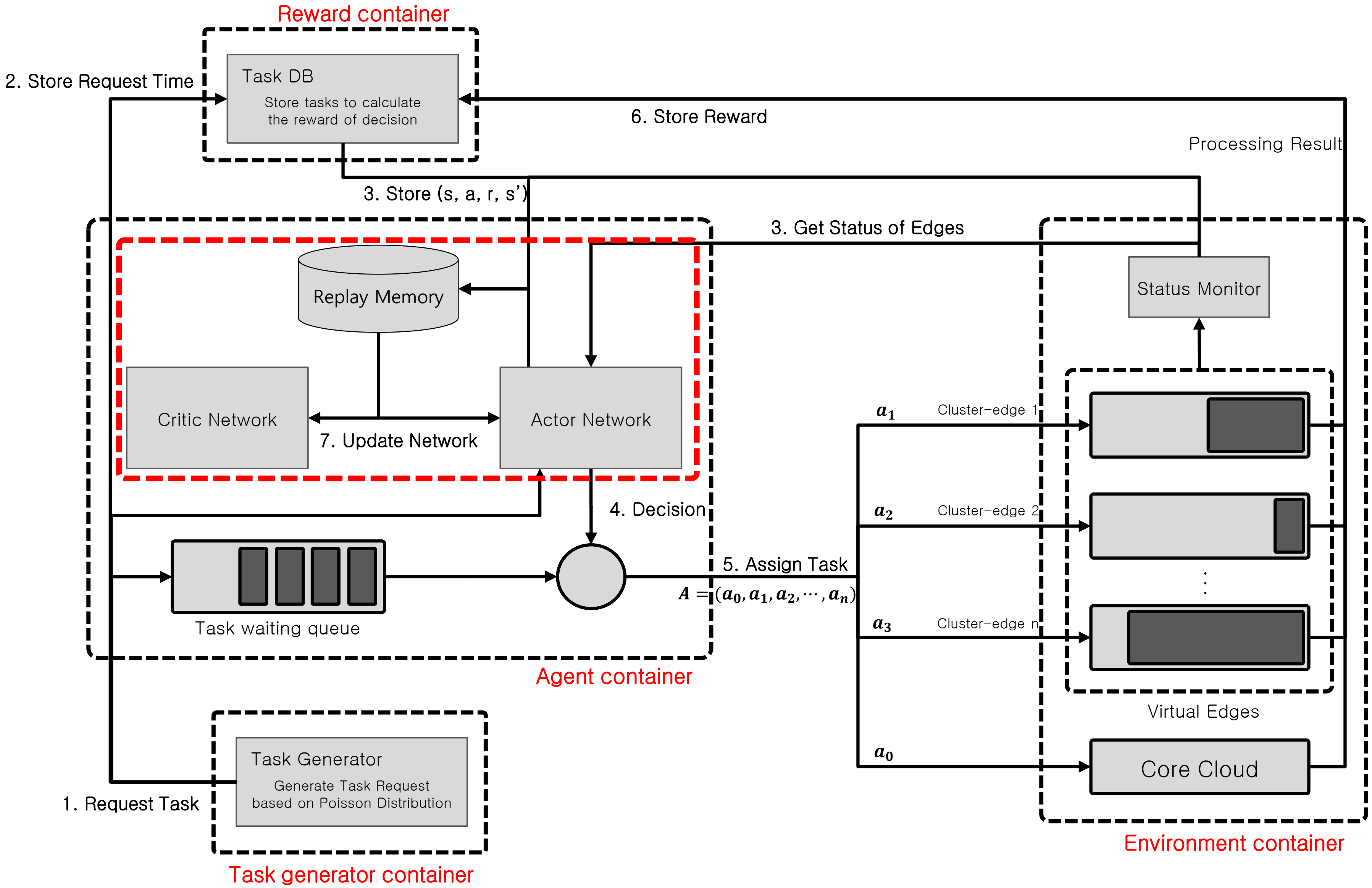
```
generator_1  | 2022-08-24 11:00:55,838 =PVE= [INFO] Generator lambda: 1.000000
agent_1      | 2022-08-24 11:00:55,853 =PVE= [DEBUG] action: 1
generator_1  | 2022-08-24 11:00:55,855 =PVE= [DEBUG] Response: "{\n    \"id\":
1,\n    \"message\": \"Successfully created\"\n}"
generator_1  |
agent_1      | 2022-08-24 11:00:56,846 =PVE= [DEBUG] action: 2
generator_1  | 2022-08-24 11:00:56,847 =PVE= [DEBUG] Response: "{\n    \"id\":
2,\n    \"message\": \"Successfully created\"\n}"
generator_1  |
agent_1      | 2022-08-24 11:00:59,846 =PVE= [DEBUG] action: 0
generator_1  | 2022-08-24 11:00:59,848 =PVE= [DEBUG] Response: "{\n    \"id\":
3,\n    \"message\": \"Successfully created\"\n}"
generator_1  |
agent_1      | 2022-08-24 11:01:00,849 =PVE= [DEBUG] action: 1
generator_1  | 2022-08-24 11:01:00,850 =PVE= [DEBUG] Response: "{\n    \"id\":
4,\n    \"message\": \"Successfully created\"\n}"
generator_1  |
agent_1      | 2022-08-24 11:01:00,850 =PVE= [DEBUG] action: 1
generator_1  | 2022-08-24 11:01:00,852 =PVE= [DEBUG] Response: "{\n    \"id\":
5,\n    \"message\": \"Successfully created\"\n}"
generator_1  |
agent_1      | 2022-08-24 11:01:01,851 =PVE= [DEBUG] action: 1
agent_1      | 2022-08-24 11:01:01,852 =PVE= [DEBUG] action: 0
generator_1  | 2022-08-24 11:01:01,852 =PVE= [DEBUG] Response: "{\n    \"id\":
6,\n    \"message\": \"Successfully created\"\n}"
generator_1  |
generator_1  | 2022-08-24 11:01:01,853 =PVE= [DEBUG] Response: "{\n    \"id\":
7,\n    \"message\": \"Successfully created\"\n}"
generator_1  |
agent_1      | 2022-08-24 11:01:05,851 =PVE= [DEBUG] action: 1
generator_1  | 2022-08-24 11:01:05,852 =PVE= [DEBUG] Response: "{\n    \"id\":
8,\n    \"message\": \"Successfully created\"\n}"
generator_1  |
agent_1      | 2022-08-24 11:01:06,850 =PVE= [DEBUG] action: 3
generator_1  | 2022-08-24 11:01:06,851 =PVE= [DEBUG] Response: "{\n    \"id\":
9,\n    \"message\": \"Successfully created\"\n}"
generator_1  |
agent_1      | 2022-08-24 11:01:09,853 =PVE= [DEBUG] action: 3
```

15 DDPG 알고리즘

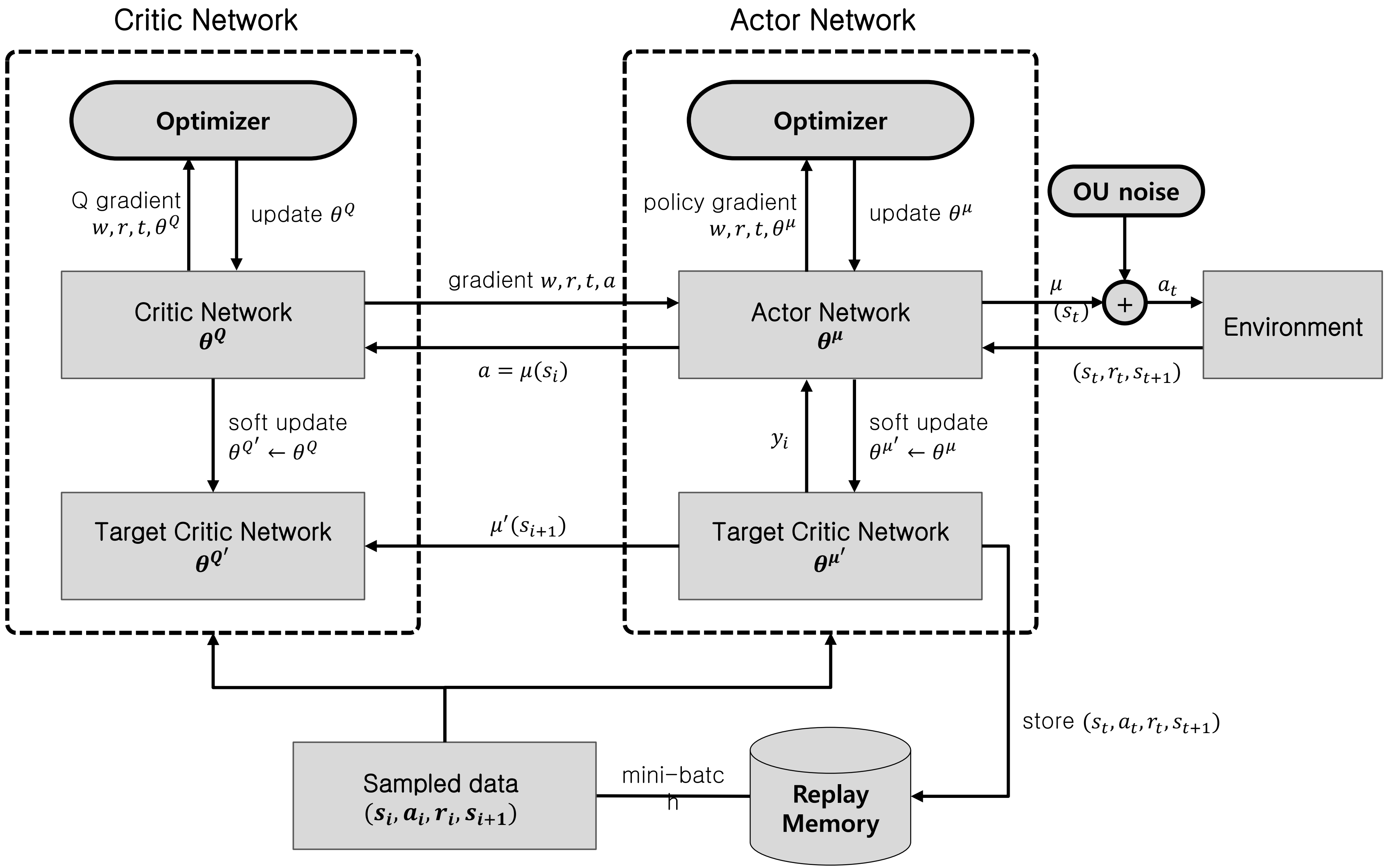
- 분산 자원 할당 정책 생성을 위해 DDPG 알고리즘 사용
  - 기존 DQN에서 continuous action space에 대한 policy를 구하는 것은 매 타임스텝마다 모든 action에 대해 최적화를 진행해야 하기 때문에 적용 불가
  - DDPG에서는 action을 결정하는 actor 네트워크와 action을 평가하는 critic 네트워크를 분리
  - Discrete한 action만 출력할 수 있었던 DQN과 달리 continuous한 action을 출력할 수 있음



16 DDPG 기반 정책 생성 에이전트 구조



제안된 DDPG 구조


Critic Network
Actor Network
Optimizer
Optimizer
Q gradient
w, r, t, θ^Q
update θ^Q
policy gradient
w, r, t, θ^μ
update θ^μ
OU noise
gradient w, r, t, a
Critic Network
$\boldsymbol{\theta^Q}$
Actor Network
$\boldsymbol{\theta^\mu}$
$\mu(s_t)$
a_t
+
Environment
$a = \mu(s_i)$
(s_t, r_t, s_{t+1})
soft update
$\theta^{Q'} \leftarrow \theta^Q$
y_i
soft update
$\theta^{\mu'} \leftarrow \theta^\mu$
$\mu'(s_{i+1})$
Target Critic Network
$\boldsymbol{\theta^{Q'}}$
Target Critic Network
$\boldsymbol{\theta^{\mu'}}$
store (s_t, a_t, r_t, s_{t+1})
Sampled data
$(\boldsymbol{s_i}, \boldsymbol{a_i}, \boldsymbol{r_i}, \boldsymbol{s_{i+1}})$
mini-batc
h
Replay
Memory

Critic-Actor 네트워크 생성 코드

GEdge Platform

```python
def get_actor(num_states, num_actions):
    last_init = tf.random_uniform_initializer(minval=-0.003, maxval=0.003)

    inputs = layers.Input(shape=(num_states,))
    out = layers.Dense(256, activation="relu")(inputs)
    out = layers.Dense(256, activation="relu")(out)
    outputs = layers.Dense(num_actions, activation="sigmoid",
                           kernel_initializer=last_init)(out)

    out = layers.Dense(num_actions, activation="relu",
                       kernel_initializer=last_init)(out)
    outputs = layers.Softmax()(out)

    model = tf.keras.Model(inputs, outputs)

    return model
```

```python
def get_critic(num_states, num_actions):
     State
    state_input = layers.Input(shape=(num_states))
    state_out = layers.Dense(16, activation="relu")(state_input)
    state_out = layers.Dense(32, activation="relu")(state_out)

    # Action
    action_input = layers.Input(shape=(num_actions))
    action_out = layers.Dense(32, activation="relu")(action_input)

    # Concat layers
    concat = layers.Concatenate()([state_out, action_out])

    out = layers.Dense(256, activation="relu")(concat)
    out = layers.Dense(256, activation="relu")(out)
    outputs = layers.Dense(1)(out)

    model = tf.keras.Model([state_input, action_input], outputs)

    return model
```

19 정책 업데이터 코드

GEdge Platform

```python
def update(self, state_batch, action_batch, reward_batch, next_state_batch):
    with tf.GradientTape() as tape:
        target_actions = self.target_actor(next_state_batch, training=True)
        y = reward_batch + self.gamma * self.target_critic(
            [next_state_batch, target_actions], training=True)

        critic_value = self.critic_model([state_batch, action_batch],
                                         training=True
                                         )
        critic_loss = tf.math.reduce_mean(tf.math.square(y - critic_value))

    critic_grad = tape.gradient(critic_loss,
                                self.critic_model.trainable_variables)

    self.critic_optimizer.apply_gradients(
        zip(critic_grad, self.critic_model.trainable_variables)
    )

    with tf.GradientTape() as tape:
        actions = self.actor_model(state_batch, training=True)
        critic_value = self.critic_model([state_batch, actions],
                                         training=True)
        actor_loss = -tf.math.reduce_mean(critic_value)

    actor_grad = tape.gradient(actor_loss,
                               self.actor_model.trainable_variables)

    self.actor_optimizer.apply_gradients(
        zip(actor_grad, self.actor_model.trainable_variables)
    )
```

수학적 모델링

목적함수

$$\pi^*(S) = \arg\min_a \sum_{i=0}^{n} \sum_{j=0}^{m} \frac{d_{i,j}}{c_{i,j}}$$

$$= \arg\min_a \sum_{i=0}^{n} \sum_{j=0}^{m} \frac{\max_k \left\{ \max\left(dt_{i,j}^k, dq_{i,j}^k\right) + dc_{i,j}^k \right\}}{c_{i,j}}$$

$$= \arg\min_a \sum_{i=0}^{n} \sum_{j=0}^{m} \frac{\max_k \left\{ \max\left(\frac{s_{i,j}^o + \alpha_{i,j}^k s_{i,j}^d}{b_{i,k}}, dq_{i,j}^k \right) + \frac{\alpha_{i,j}^k c_{i,j}}{f_i}, \right\}}{c_{i,j}}$$

$$s.t.\ C1 : \sum_{k=0}^{n} \alpha_{i,j,k}^r = 1$$

$$C2 : \forall \alpha_{i,j,k}^r \in [0,1]$$

액션

$$\alpha_{i,j} = \left\{ \alpha_{i,j}^0, \alpha_{i,j}^1, \alpha_{i,j}^2, \cdots, \alpha_{i,j}^n \right\}$$

보상함수

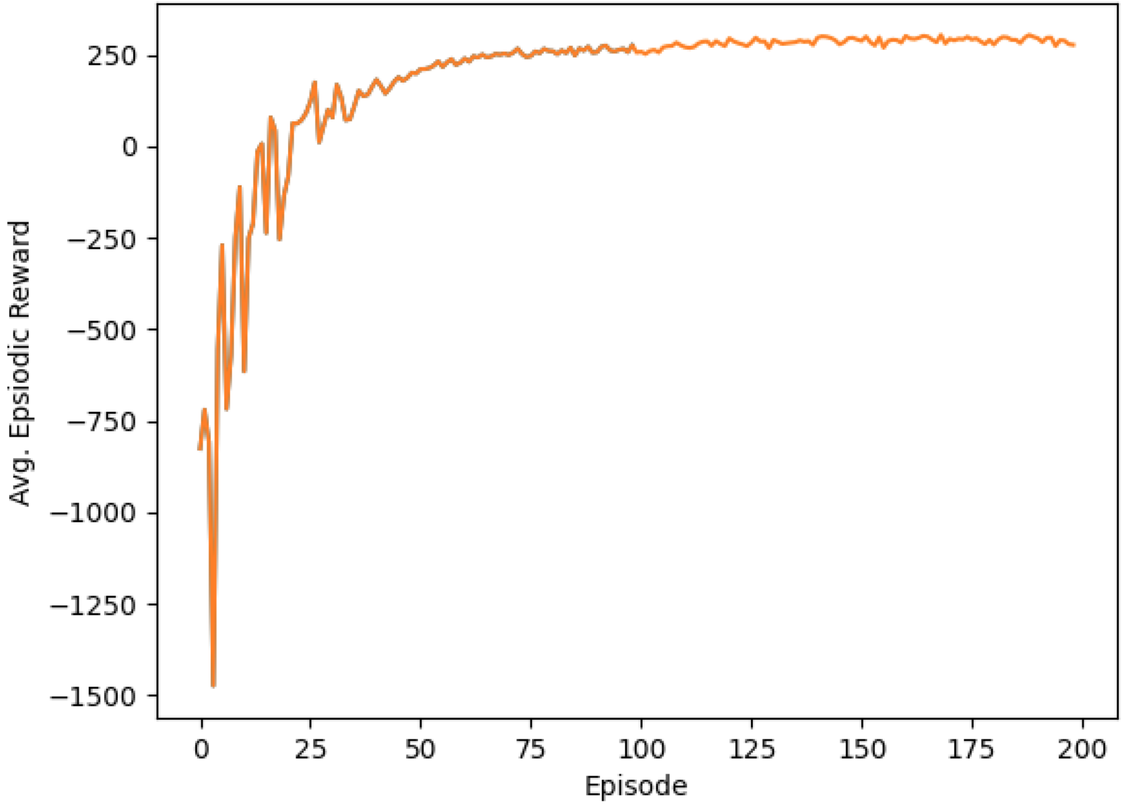
$$R(S,\alpha) = \hat{d}_{i,j} - d_{i,j}$$

$$= \max_k \left\{ \max\left(dt_{i,j}^i, dq_{i,j}^i\right) + dc_{i,j}^i \right\} - \max_k \left\{ \max\left(dt_{i,j}^k, dq_{i,j}^k\right) + dc_{i,j}^k \right\}$$

$$= dq_{i,j}^i + \frac{c_{i,j}}{f_i} - \max_k \left\{ \max\left(\frac{s_{i,j}^o + \alpha_{i,j}^k s_{i,j}^d}{b_{i,k}}, dq_{i,j}^k \right) + \frac{\alpha_{i,j}^k c_{i,j}}{f_i}, \right\}$$

21 분산 오프로딩을 위한 DDPG 기반 정책 생성 성능

Parameter	Value
Number of Edges	3
Overhead size	1,000~10,000 Mb
Data size	1,000~10,000 Mb
Required FLOPS	10,000~100,000 TFLOPS
Computation Power of Cloud	200 TFLOPS/s
Computation Power of Edge	100 TFLOPS/s
Bandwidth between Edge and Server	100 Mb/s
Bandwidth between Edge and neighbor Edge	10,000 Mb/s



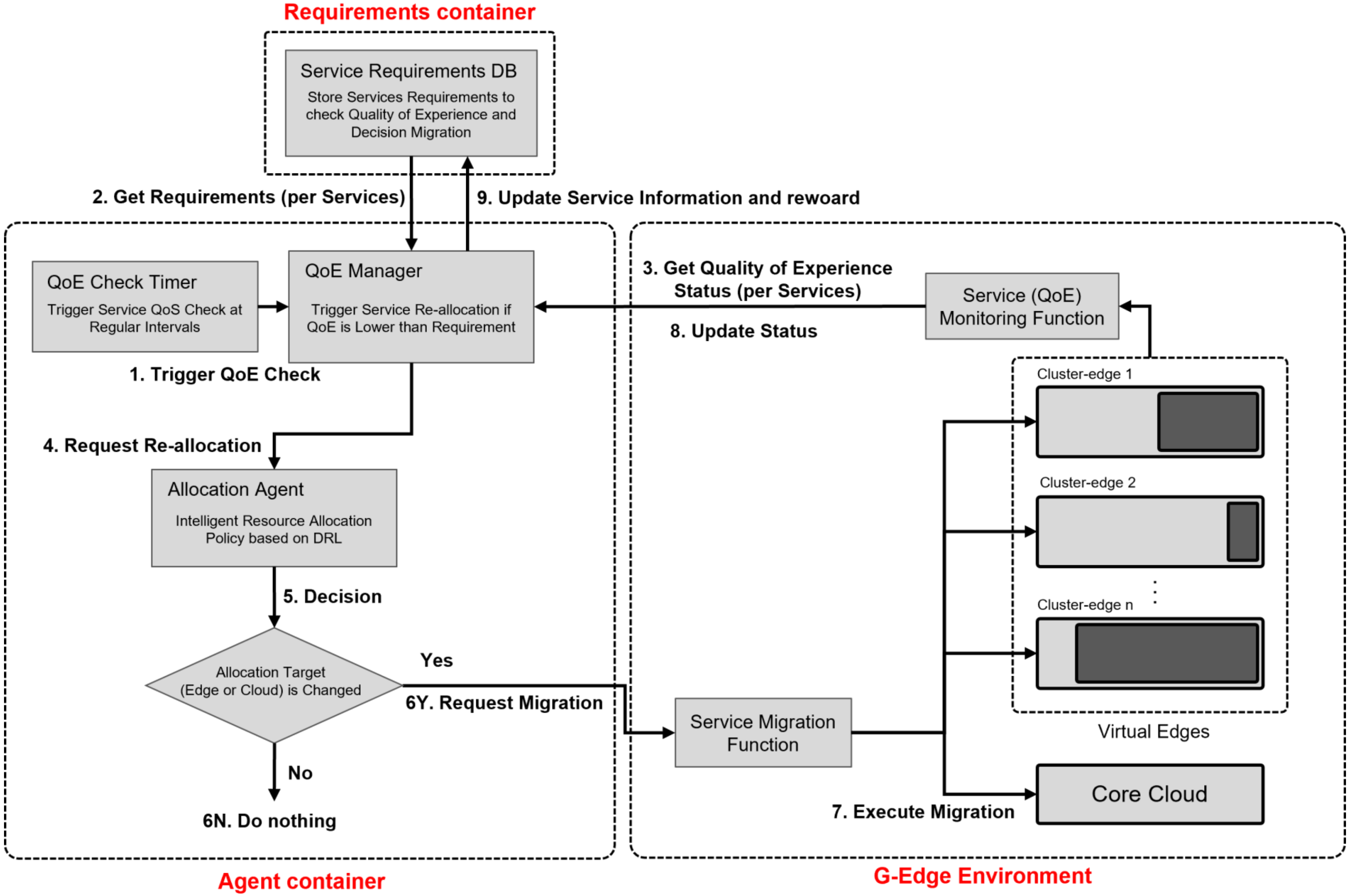
22 DDPG 기반 정책 생성 결과

액션 = [클라우드, 엣지_1, 엣지_2, 엣지_3]

```
ep 180. mean reward: 290.93121937051245,        last action: [0.46488127 0.         0.25084028 0.28427842]
ep 181. mean reward: 278.0553296279726,         last action: [0.5924614  0.25308448 0.15445413 0.        ]
ep 182. mean reward: 291.92556850897734,        last action: [0.34951282 0.08358695 0.21738742 0.34951282]
ep 183. mean reward: 298.8915540030207,         last action: [1. 0. 0. 0.]
ep 184. mean reward: 297.8796925016393,         last action: [1. 0. 0. 0.]
ep 185. mean reward: 291.21058223046043,        last action: [0.44798535 0.         0.10402931 0.44798535]
ep 186. mean reward: 291.11835007988196,        last action: [0.8649427  0.04865093 0.         0.08640638]
ep 187. mean reward: 282.36514903702874,        last action: [0.5553444  0.44465557 0.         0.        ]
ep 188. mean reward: 297.0997645474146,         last action: [0.32301354 0.14320551 0.23540859 0.2983724 ]
ep 189. mean reward: 304.3647023705164,         last action: [0.5765534  0.42344654 0.         0.        ]
ep 190. mean reward: 299.4091970303612,         last action: [0.63897526 0.36102474 0.         0.        ]
ep 191. mean reward: 295.16040387336017,        last action: [1. 0. 0. 0.]
ep 192. mean reward: 284.94558983904284,        last action: [0.5283476  0.         0.47165242 0.        ]
ep 193. mean reward: 296.7982932997275,         last action: [0.87145054 0.12854947 0.         0.        ]
ep 194. mean reward: 297.18235013604215,        last action: [1. 0. 0. 0.]
ep 195. mean reward: 274.2150363662961,         last action: [0.7760477 0.        0.2239523 0.       ]
ep 196. mean reward: 290.1298070822163,         last action: [0.5 0.  0.  0.5]
ep 197. mean reward: 289.05974900475616,        last action: [0.66988057 0.33011946 0.         0.        ]
ep 198. mean reward: 279.21270477786595,        last action: [0.39589486 0.21499671 0.33529088 0.05381754]
ep 199. mean reward: 277.34944085995414,        last action: [0.49236563 0.50763434 0.         0.        ]
```

Ⅱ 지능형 서비스 이동 정책 기술

23 서비스 이동 정책 알고리즘



24 서비스 이동 매니저

```python
...
class QoEManager:
    def __init__(self, env, db, agent):
        self.db = db
        self.env = env
        self.agent = agent

    def qoe_check(self):
        requirements = self.db.get_requirements()
        qoe_status = self.env.get_qoe()

        for service in qoe_status:
            if service['requirement'] > requirements[service['task_id']]:

    def request_allocation(self, service):
        target = self.agent.allocation(service['task_id'])

        if target != service['target']:
            res = self.env.migration(service['task_id'], target)

            if res:
                self.update_db(service, target)
                self.agent.update_reward()

    def update_db(self, service, target):
        self.db.hest(service['task_id'], 'target', target)
...
```

서비스 요구사항 만족여부 확인

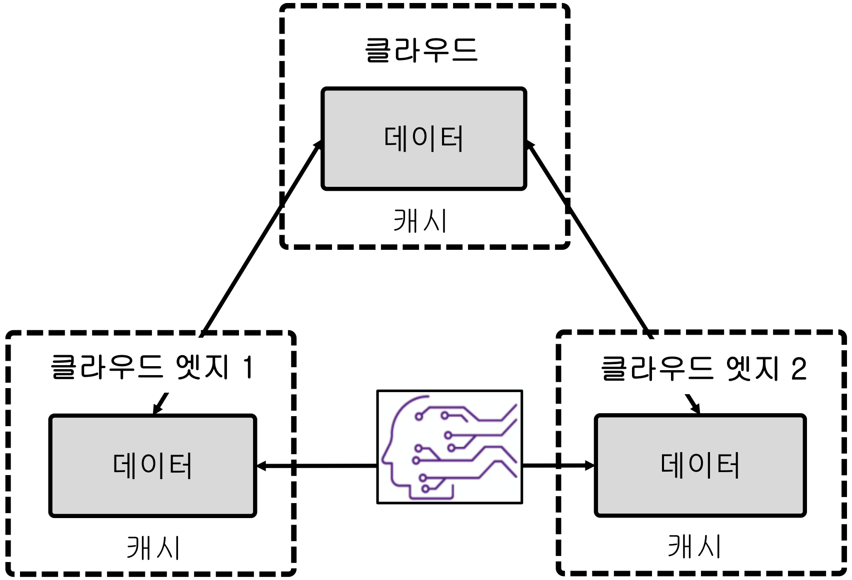
서비스 자원 재할당 요청

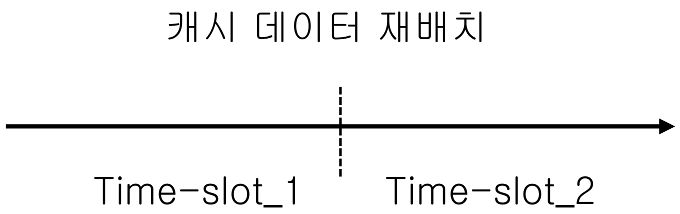
서비스 DB 및 리워드 업데이트

Ⅲ 강화학습 기반 지능형 협업 캐시 정책 생성 기술

데이터 저장 캐시 서비스 시나리오

- 캐시 정책 목적
  - 이동성을 고려한 캐시 콘텐츠 재배치 기술





MN on CE_1 : C(S)_1 (P_10),
MN to CE_2 (>0.5, 이동 확률)
→ time-slot_2에서 C(S)_1 미리 배치 시킴

IV 향후 연구

향후 계획

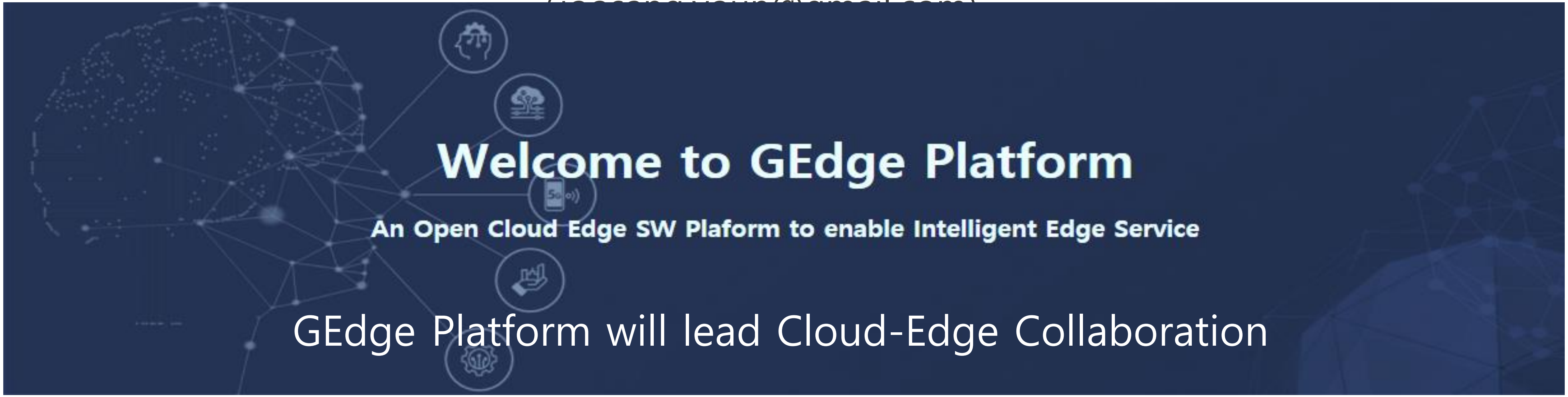
- 강화학습 기반 오프로딩 정책 생성기 학습 모델 개발
  - PPO 기반 분산 오프로딩 정책 생성
- 강화학습 기반 서비스 이동 학습 모델 개발
- 캐싱 정책 알고리즘 개발 & 최적화

- **테스트베드 기반 지능형 정책 생성기 최적화**

감사합니다.

http://gedge-platform.github.io

GS-Link 프레임워크 코어개발자(GS-Linkhq)
윤주상



GEdge Platform will lead Cloud-Edge Collaboration