MEC 기반 스마트 팩토리 환경에서 DRL 를 이용한 태스크 스케줄링

구설원 [1], 임유진 [1]
[1] 숙명여자대학교 IT 공학과
tjrgkr501@sookmyung.ac.kr, yujin91@sookmyung.ac.kr

Task Scheduling Using Deep Reinforcement Learning in Mobile Edge Computing-based Smart Factory Environment

Seolwon Koo[1], Yujin Lim[1]
[1]Dept. of IT Engineering, Sookmyung Women's University

요 약

최근 들어 다양한 제약 조건이 있는 스마트 시티나 스마트 팩토리와 같은 도메인들 내에서 태스크들을 효과적으로 처리하기 위해서 MEC 기술이 많이 사용되고 있다. 그러나 이러한 도메인에서 발생하는 복잡하고 동적인 시나리오는 기존의 휴리스틱이나 메타 휴리스틱 기법을 이용하여 해결하기엔 계산 복잡도가 증가하는 문제점을 가지고 있다. 따라서 최근 들어 이러한 문제점을 해결하기 위한 방법 중 하나로 강화학습과 딥러닝이 결합된 DRL 기법이 주목을 받고 있다. 본 연구는 스마트 팩토리 환경에서 종속성을 가진 태스크들이 실행시간과 태스크가 처리되는 MEC 서버들의 로드 표준편차를 최소화하는 태스크 스케줄링 기법을 제안한다. 모의실험을 통하여 제안 기법은 태스크가 증가하는 동적인 환경에서도 좋은 성능을 보임을 증명하였다.

1. 서론

모바일 엣지 컴퓨팅 (Mobile Edge Computing, MEC)[1] 기술은 말단 디바이스와 지리적으로 거리가 먼 클라우드 서버가 디바이스의 컴퓨팅 작업을 수행함으로써 발생할 수 있는 지연시간 증가 등의 단점을 해결하기 위해 제안된 기술로, 특히 스마트 팩토리나 스마트 시티와 같은 동적이고 복잡한 도메인에서 그 해결책으로써 많은 주목을 받고 있는 기술이다. 이를 통하여 컴퓨팅, 스토리지, 통신 자원의 효과적인 운용뿐만 아니라 동적인 도메인 환경에서 낮은 처리 지연시간이나 네트워크 신뢰성과 같은 요구 사항을 만족시켜 줄 수 있다.

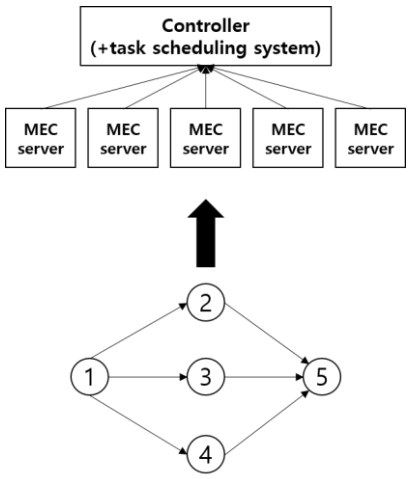
스마트 팩토리 환경은 전통적인 팩토리 환경과 달리 성능의 최적화를 위하여 스스로 학습할 수 있는 능력을 갖고 있으며, MEC 서버를 이용하여 이러한 지능적인 서비스를 효과적으로 제공할 수 있다[2]. 스마트 팩토리 환경에서의 작업은 하나 이상의 종속된 일련의 태스크들로 이루어져 있으며, 또한 작업 단위의 지연시간 제약을 만족시키기 위한 많은 연구들이 진행되고 있다. 작업 단위의 지연시간 제약조건을 만족시키기 위해서는 각 태스크 단위에서의 지연시간을 최소화하는 것이 중요하다. 또한 MEC 서버의 컴퓨팅 자원 또한 제한적이므로 성능 최적화를 위해서는 MEC 서버들 간 로드 밸런싱 문제도 중요하게 여겨진다.

작업 내 태스크들의 종속성을 고려한 기존의 많은 연구들은 휴리스틱[3,4] 또는 메타 휴리스틱[5] 기법을 이용하여 태스크들을 스케줄링 하였다. 요즘 IoT 와 같은 기술 발전으로 인해 높은 계산 성능을 요구하는 디바이스 태스크들이 기하급수적으로 증가하고 있다. 하지만 이러한 휴리스틱과 메타 휴리스틱 기법의 경우 태스크 수가 증가할수록 계산에 대한 복잡도가 급격하게 증가하게 된다. 이러한 문제 해결을 위하여 최근 들어 심층 강화 학습(Deep Reinforcement Learning, DRL) 기법이 고려되고 있다. DRL 은 딥러닝과 강화학습 기법을 이용하여 동적인 시나리오에서의 자원 최적화 문제를 해결할 수 있다.

본 논문은 MEC 와 결합된 스마트 팩토리 환경에서 DRL 기법을 이용하여 작업 실행시간을 줄이면서 MEC 서버들 간의 로드 밸런싱을 위한 태스크 스케줄

링 기법을 제안한다.

2. 시스템 모델



(그림 1) 제안하는 태스크 스케줄링 시스템 아키텍처

본 논문에서 제안하는 전체적인 아키텍처는 (그림 1)를 통해 볼 수 있다. 이는 종속되어 있는 태스크들로 구성된 태스크 집합(T)로 이루어진 하나의 작업(J), 여러 개의 MEC 서버들, 그리고 여러 서버들을 제어하며 모델링 된 DRL 에이전트를 실행시킬 수 있는 컨트롤러가 있다. 하나의 작업(J)에 속한 태스크들 간의 종속성은 DAG(Directed Acyclic Graph)로 모델링 하며 $G = (V, E)$로 정의한다. 노드($v \in V$)는 각 태스크를 의미하며, 방향성 에지($e \in E$)는 태스크 간의 종속관계를 나타낸다. 그리고 컨트롤러는 MEC 서버에 대한 정보와 태스크들에 대한 정보를 다 갖고 있다고 가정한다. 하나의 작업(J)는 종속되어 있는 태스크들의 집합을 말하며 태스크의 집합은 $T = \{t_1, \dots, t_n\}, |T| = N$와 같이 나타낼 수 있다. 시작하는 태스크와 끝나는 태스크는 t_1, t_n로 각각 표현되고, 각 태스크들의 사이즈는 $ts_i, i \in N$ (MI)로 정의했다. MEC 서버의 집합은 $M = \{m_1, \dots, m_k\}, |M| = K$로 나타낸다. 각 서버의 처리 능력은 $mc_t, t \in K$ ($MIPS$)로 정의했다. 각 서버 간의 데이터 통신 오버헤드는 고려하지 않았다. 그 이유는 서버 간 통신을 유선으로 연결되어 있다고 가정하였고, 또한 서버에서 처리된 태스크 실행결과는 입력된 태스크에 비해 그 크기가 매우 작기 때문에 고려하지 않았다. 하나의 작업에 종속된 태스크들이 선택한 MEC 서버에서의 실행이 끝나는 시간은 다음과 같다.

$$FT(t_n) = \max_{t_i \in Pre(t_n)} FT(t_i) + ST(t_n) \quad (1)$$

$$ST(t_n) = ts_n / mc_t \quad (2)$$

식(1)에서 $FT(t_n)$는 t_n의 실행이 끝나는 시간을 의미하며, $ST(t_n)$는 t_n이 할당된 MEC 서버에서의 처리 시간이다. $Pre(t_n)$는 t_n의 바로 이전 단계 태스크들의 집합이란 의미이다. $\max_{t_i \in Pre(t_n)} FT(t_i)$는 $Pre(t_n)$ 집합에 포함된 태스크들의 실행시간이 가장 큰 값을 의미한다. $Pre(t_n)$들 중 실행시간이 가장 큰 시간과 t_n가 할당된 MEC 서버($m_t, t \in K$)에서의 처리 시간 $ST(t_n)$을 합하면 $FT(t_n)$를 구할 수 있다. mc_t는 t_n가 할당된 MEC 서버 m_t의 처리 능력이며, ts_n는 t_n의 태스크 사이즈이다. 만약 시작 태스크 t_1의 실행이 끝나는 시간 $FT(t_1)$을 구하고자 한다면, 시작 태스크의 이전에 연결된 태스크가 없기 때문에 선택된 MEC 서버에서 처리하는 시간만 구하면 된다.

본 연구는 작업(J)의 처리 시간과 MEC 서버들의 로드 표준편차를 최소화하기 위해서 작업(J)에 포함된 태스크들을 DRL 기법을 이용하여 MEC 서버들에 스케줄링하는 기법을 제안한다. 이를 위해 MDP(Markov Decision Process)는 다음과 같이 정의하였다.

상태 (s):

$[MEC_1, \dots, MEC_k,\ d(t_j),\ ts_j,\ \max_{t_i \in Pre(t_j)} FT(t_i), load(t_j)]$,

상태는 서버들의 상태와 태스크의 상태를 결합하였다. $MEC_1, \dots, MEC_k$는 각 서버의 로드를 나타내며, $d(t_j)$는 시작 태스크 t_1부터 태스크 $t_j, j \in N$ 까지 에지 수이다. 그리고 $\max_{t_i \in Pre(t_j)} FT(t_i)$는 t_j와 연결된 직전 태스크들 중 실행이 끝나는 시간이 가장 큰 시간이다. 그리고 $load(t_j)$는 태스크 t_j가 할당된 서버의 로드이다.

행동 (a):

$[1, \dots, m]$,

태스크 t_j가 할당되는 MEC 서버를 나타낸다.

보상함수 (r):

$-(\alpha \cdot (FT(t_j)\_nor) + (1-\alpha) \cdot (std\_server\_nor))$, $FT(t_j)\_nor$ 는 $FT(t_j)$ 를 정규화한 값이고 std_server_nor는 MEC 서버들의 로드 표준편차를 정규화한 값이다.

3. 제안 알고리즘

본 연구에서는 DRL 중에서도 Google DeepMind 에서 제안한 Deep Q-Network(DQN)을 사용한다 DQN 은 Q-learning 이 사용한 테이블 기반 방법론이 아닌 DNN 을 사용한다는 특징이 있다. 그렇기 때문에 상태 집합의 크기가 상당히 큰 환경에서 효과적으로 실행될 수 있다. 본 논문에서는 많은 태스크와 MEC 서버들의 상태들을 고려하기 때문에 DQN 을 이용하였다. 연구에서 제안하는 알고리즘은 <표 1>과 같다.

<표 1> DQN 을 이용한 태스크 스케줄링 기법

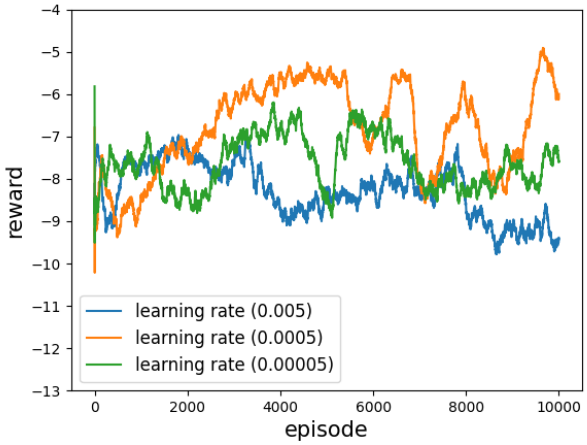
Algorithm 1. Task Scheduling Strategy Using DQN	
Input : the number of episode EP, minibatch B, exploration rate ε, discount factor γ, the number of tasks |T|, update frequency of the target network U	
Output : network parameters θ	
1	initialize replay memory D to capacity M,
2	initialize evaluation action-value function Q with random parameter θ
3	initialize target action-value function $\hat{Q}$ with random

	parameter θ'
4	initialize state s_i
5	**for** episode = 1, EP **do**
6	randomly generate graph of |T| tasks
7	sort task set T by priorities
8	**for** step = 1, |T| **do**
9	with probability ε randomly choose an action a_i
10	other wise a_i=$argmax_a Q(s_i, a;\ \theta)$
11	$s_{i+1},\ r_i \leftarrow$ execute a_i *when* s_i
12	store transition $(s_i, a_i, r_i, s_{i+1}, done_i)$ in D
13	randomly select samples B from D
14	**if** $done_i$ **then**
15	$y_i = r_i$
16	**else**
17	$y_i = r_i + \gamma \max_{a'} \hat{Q}(s_{i+1}, a'; \theta')$
18	**end**
19	Calculate the $loss(y_i,\ Q(s_i, a_i))$
20	Update Q using the Adam algorithm by $Q(s_i, a_i;\ \theta)$
21	Every U steps, copy weights from Q to $\hat{Q}$
22	**end**
23	**end**

<표 1>에서 7 번째 줄은 태스크들이 우선순위에 의해서 정렬되는 것을 의미한다. 태스크들을 정렬하기 위해 필요한 특징은 2 가지이며 정렬하는 과정은 다음과 같다. 먼저 시작 태스크부터 해당 태스크까지의 에지 개수, 즉 거리를 기준으로 오름차순으로 정렬해 주고, 같은 에지 개수를 갖는 태스크들 내에서는 태스크의 데이터 사이즈를 기준으로 오름차순으로 정렬해 준다. 다시 말해서, 시작 태스크에서 가까운 태스크일수록 높은 우선순위를 가지며, 시작 태스크에서 동일한 거리에 있는 태스크가 하나 이상일 경우는 데이터 사이즈를 기준으로 높은 우선순위를 가진다. 19 번째 줄에서 $loss(y_i,\ Q(s_i, a_i))$는 손실 함수를 의미하고 Huber loss 함수를 이용하였다.

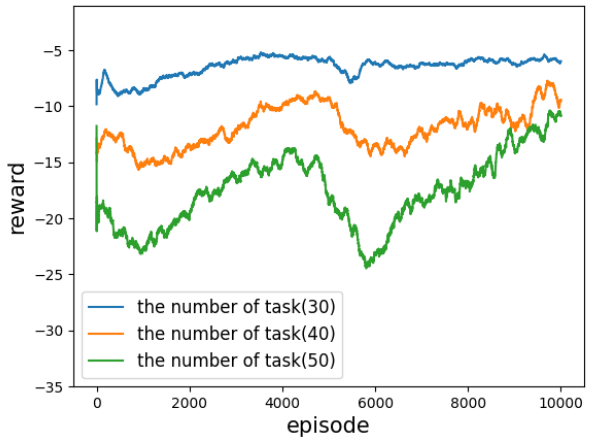
4. 실험 결과

제안하는 연구의 성능을 평가하기 위해 Python 과 Pytorch 라이브러리를 사용하였다. 성능 평가를 위한 태스크들의 종속성은 NetworkX 라이브러리[6]를 이용하여 랜덤으로 생성되도록 하였다. 그리고 작업 내의 시작 태스크와 종료 태스크는 하나씩 구성되도록 하였다. 하나의 작업 내의 태스크들의 데이터 사이즈[7]는 $[5000, 10000]$ MI 로 설정했고 태스크들을 처리할 MEC 서버의 처리 능력은 5000 $MIPS$로 설정했다. 제안한 알고리즘에서 DNN 은 총 2 개의 은닉층으로 구성되어 있다. 탐험과 활용을 위해 ε는 0.08 부터 0.01 까지로 감소시켰다. 그리고 γ은 0.98, α는 0.5 로 설정하였다. α의 경우 2 개의 목적 함수들 사이에서 중요도를 설정하는 값이다. U 는 30 으로 정하였고 에피소드 개수인 EP 는 10,000 으로 설정하였다. 그리고 미니 배치 사이즈는 32, 리플레이 버퍼 크기는 50,000 로 정하였다.



(그림 2) 에피소드에 따른 리워드 값(learning rate 의 변화)

(그림 2)는 DQN 에서 사용할 learning rate 를 정하기 위해 실험을 진행하였다. (그림 2)는 에피소드에 따른 리워드 값을 나타내며 환경은 태스크 30 개, MEC 서버 5 개로 구성하여 진행하였다. Learning rate 의 경우 0.005, 0.0005, 0.00005 로 비교하였다. 0.005 와 0.00005 는 에피소드가 증가될수록 리워드가 감소하는 방향으로 진행되었고, 0.0005 는 에피소드가 증가될수록 리워드가 전반적으로 증가하는 방향을 보였다. 이와 같은 결과를 토대로 실험을 진행할 때 learning rate 를 0.0005 로 설정하였다.



(그림 3) 에피소드에 따른 리워드 값(태스크 개수의 변화)

(그림 3)은 증가하는 태스크 개수에도 MEC 서버의 표준편차와 작업의 실행시간이 최소화되는지 확인하기 위해 실험을 진행하였다. 앞서 (그림 2)와 마찬가지로 에피소드에 따른 리워드 값을 나타내며 태스크 개수의 변화를 30, 40, 50 으로 정하였다. 환경은 MEC 서버가 5 개일 때로 가정하였다. 태스크가 30, 40, 50 일 때 에피소드가 10,000 까지 진행될 때 전부 증가 추이를 보이면서 리워드가 높아지는 방향으로 학습이 잘 되고 있음을 알 수 있다. 리워드 값의 경우 태스크 개수가 30 일 때가 가장 큰 이유는 처리해야

하는 태스크 개수가 상대적으로 적기 때문에 태스크 개수 50 일 때에 비해 MEC 서버들 간의 로드 표준편차와 작업 실행시간에서 차이가 나기 때문이다. 또한 중간에 급격히 리워드가 떨어지는 부분들은 ε 그리디 정책을 사용하여 탐험(exploration)을 할 수 있도록 하였기 때문이다. 평균적으로 리워드가 감소하는 그래프가 아닌 증가하는 그래프를 보이고 있다.

5. 결론

본 연구는 MEC 서버를 스마트 팩토리에 결합한 환경에서 종속성이 있는 태스크들의 스케줄링 알고리즘을 제안하였다. 스마트 팩토리의 경우 지연시간 제약조건이 있어서 작업의 실행시간을 최소화하여야 했고, MEC 서버는 제한적인 능력을 갖고 있기 때문에 로드를 분산시켜야 했다. 그래서 작업의 실행시간을 최소화하면서 MEC 서버 로드의 표준편차를 최소화하기 위해 DQN 기법을 이용하였다. 실험 결과 태스크들이 증가하여도 성능의 저하 없이 잘 작동하는 것을 확인할 수 있었다. 향후, MEC 서버 로드의 표준편차와 지연시간 만족도에 대한 실험을 추가로 진행하고자 한다.

사사 문구

이 성과는 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임 (No. 2021R1F1A1047113).

참고문헌

[1] Y. Mao, C. You, J. Zhang, K. Huang and K. B. Letaief, "A Survey on Mobile Edge Computing: The Communication Perspective," *IEEE communications surveys & tutorials,* vol. 19, no. 4, pp. 2322-2358, 2017.

[2] B. Chen, J. Wan, L. Shu, P. Li, M. Mukherjee and B. Yin, "Smart Factory of Industry 4.0: Key Technologies, Application Case, and Challenges," *IEEE Access*, vol. 6, pp. 6505-6519, 2018.

[3] H. Topcuoglu, S. Hariri and M. Y. Wu, "Performance-effective and Low-complexity Task Scheduling for Heterogeneous Computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 3, pp. 260-274, 2002.

[4] L. F. Bittencourt, R. Sakellariou and E. R. M. Madeira, "DAG Scheduling using a Lookahead Variant of the Heterogeneous Earliest Finish Time Algorithm," *2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing*, Pisa, Italy, pp. 27-34, 2010.

[5] F. Ebadifard and S. M. Babamir, "A PSO-based Task Scheduling Algorithm Improved using a Load-balancing Technique for The Cloud Computing Environment," *Concurrency and Computation: Practice and Experience,* vol. 30, no. 12, pp. 1-16, 2018.

[6] NetworkX (Network Analysis in Python), https://networkx.org/

[7] M. Afrin, J. Jin, A. Rahman, Y. C. Tian and A. Kulkarni, "Multi-objective Resource Allocation for Edge Cloud Based Robotic Workflow in Smart Factory," *Future Generation Computer Systems*, vol. 97, pp. 119-130, 2019.