https://doi.org/10.7840/kics.2024.49.8.1096

엣지 클러스터 구조의 차량 오프로딩 시스템에서 강화학습을 활용한 부하 분산 알고리즘

방 수 정◆, 이 미 정°

RL-Based Load Balancing for Edge Server Based Vehicular Offloading System

Soo-jeong Bang◆, Mee-jeong Lee°

요 약

차량을 위한 지능형 응용 서비스가 활발히 연구되고 있으나, 차량의 제약적인 자원만으로 서비스 요구사항을 만족시키는 것에는 한계가 존재하여 엣지 서버에 작업을 오프로딩하는 방법론이 주목받고 있다. 하지만 엣지 서버는 중앙 집중형 서버에 비해 자원이 제약적이어서 차량이 밀집되거나 고연산 작업이 다량 요청되면 과부하 상태로 이어져 오프로딩 성공률을 낮출 수 있다. 본 논문은 한정적인 엣지 서버의 가용 자원 활용률을 최대화하는 것을 목적으로, 차량 오프로딩 환경에서 엣지 서버 간 부하 불균형 문제를 해결하는 알고리즘을 제안한다. 본 연구는 엣지 서버 간 클러스터를 형성하고 클러스터 내 가용 자원량의 편차를 최소화하는 최적화 문제를 정의하고, 동적인 환경에서도 최적의 결정을 하기 위하여 강화학습 모델 TD3(Twin Delay DDPG)를 활용한 부하 분산 알고리즘을 제안한다. 실험을 통해 제안 알고리즘이 부하 불균형 상황을 효과적으로 완화시킴을 확인하였고 특히 차량 수가 증가하거나 작업의 다양성이 증가하는 환경에서도 높은 오프로딩 성공률을 보이는 것과 부하 분산으로 인한 확장성 향상 효과를 확인하였다.

Key Words : Load balancing, edge computing, deep reinforcement learning, clustering, vehicular network

ABSTRACT

Intelligent vehicular applications are currently under active research. However, there are limitations in meeting the service requirements with the vehicle's limited resources. Therefore, the methodology of offloading tasks to edge servers is gaining attention. However, edge servers are resource-constrained compared to centralized servers, and this can lead to overload and localized offloading failures with high traffic or when a large number of high-computation tasks are requested. In this paper, we propose an algorithm designed to address the problem of load imbalance among edge servers in a vehicular offloading system. The proposed algorithm aims to maximize the utilization of the limited edge servers' resources to increase the offloading success rate. We have formulated an optimization problem to form a cluster among edge servers and minimize the deviation of available resources within the cluster. To facilitate optimal load distribution in dynamic environments, we propose an algorithm that utilizes a reinforcement learning model TD3 (Twin Delay DDPG). Through experiments, we demonstrate that the proposed algorithm effectively alleviates load imbalance,

◆ First Author : Ewha Womans University Department of Computer Science and Engineering, tnwjd7732@ewhain.net, 학생회원
° Corresponding Author : Ewha Womans University Department of Computer Science and Engineering, lmj@ewha.ac.kr, 종신회원
논문번호 : KICS202402-027-C-RN, Received February 7, 2024; Revised March 30, 2024; Accepted April 9, 2024

particularly in environments with a growing number of vehicles or an increasing diversity of tasks. The algorithm achieves a high offloading success rate and confirms the scalability improvement resulting from load balancing.

Ⅰ. 서 론

인공지능과 증강현실 기술의 급속한 발전으로 자율주행 및 ITS (Intelligent Transportation Service) 분야에서 지능형 차량 서비스 연구 또한 빠르게 진행되고 있다[1]. 특히 차량 카메라를 통해 수집된 고해상도 영상 속 실시간 객체 탐지나 3D 지도 기반의 이미지 보조 내비게이션과 같은 기술은 운전자와 탑승객의 경험을 개선하고 교통 안전성을 높이는 데 크게 기여한다[2,3]. 최근 컴퓨터 기술의 급속한 발전으로 인해 기존의 텍스트 위주의 사용자 환경에서 벗어나 이미지, 그래픽, 오디오 및 비디오 데이터 등을 제공하는 멀티미디어 사용자 환경으로 변화하고 있다.

하지만 이러한 지능형 차량 서비스는 연산 집약적이며 대용량의 데이터를 실시간으로 처리해야 한다는 특징을 갖고 있다[4]. 특히 시간 지연은 서비스 만족도를 저하시킬 뿐 아니라 차량 서비스 특성상 안전 문제에 직결될 수 있으므로 신속한 처리가 필수적이다[5]. 이에 따라, 차량에 고사양의 컴퓨팅 및 스토리지 자원이 탑재되고 있지만, 지능형 차량 서비스 요구사항을 독자적으로만 만족시키기는 여전히 한계가 있다[6]. 또한 동시에 여러 작업 요청이 발생하면 차량 내 자원 부족 문제가 더욱 심화되어 대기 및 처리 지연이 증가하고 결과적으로 QoE(Quality of Experience)가 크게 저하될 수 있다.

이러한 문제점을 극복하는 해결책으로 오프로딩 기술이 제안되었다[7]. 오프로딩은 차량, 휴대폰, IoT와 같이 자원이 제한된 기기들이 외부 서버의 도움을 받아 작업을 처리하는 기술이다[7]. 처리해야 할 원시 데이터를 서버로 전송하면 서버가 작업을 대신 수행하고 결과를 반환한다. 이에 따라 자체 자원으로 처리하는 것보다 연산 시간을 크게 단축하여 서비스 요구사항을 만족하는 동시에 에너지 소모와 자원 사용을 최소화할 수 있다[6].

오프로딩 시스템은 서버 유형에 따라 분류하면 클라우드 서버 방식과 엣지 서버 방식으로 나뉜다[8]. 클라우드 서버를 활용하는 경우, 풍부한 자원으로 작업을 처리할 수 있어 연산 시간이 크게 줄어드나 차량과 클라우드 서버 간의 물리적 거리로 인한 데이터 및 결과 전달 과정의 지연이 길어져 실시간 서비스 제공에 한계가 있다[8]. 또한 대용량의 원시 데이터를 네트워크 중앙부로 전송해야 하므로 네트워크 코어 혼잡을 유발할 수 있다[8]. 반면에 엣지 서버 방식은 클라우드 서버의 한계를 극복하는 패러다임으로, 서버가 사용자 가까이에 분산 배치되기 때문에 시간 지연 및 대역폭 부담 문제를 완화할 수 있어 차량 오프로딩 시스템 연구에서 활발히 활용되고 있다[6]. 하지만 분산형 구조로 인해 클라우드 서버와 비교해 상대적으로 제한된 자원을 갖는다는 본질적 한계를 가지고 있는데, 이러한 제약은 특히 차량이 밀집되어 있거나 고연산 작업 요청이 많은 상황에서 성능 저하의 원인이 될 수 있다. 심한 경우, 서버 과부하가 발생하여 작업 처리 대기 시간이 길어질 수 있고, 이는 요구 응답시간을 초과함으로써 오프로딩 실패로 이어질 가능성이 있다[9].

이러한 배경에서, 차량 오프로딩 시스템의 한정적인 엣지 서버 자원을 효과적으로 활용하기 위한 서버 간 부하 분산 연구가 활발히 진행되고 있다. 부하 분산 알고리즘은 차량의 이동성이나 일시적인 작업 요청 과다 상황에서도 작업을 적절히 분산하여 처리하므로 가용 자원을 효과적으로 활용할 수 있어 오프로딩 성공률을 향상시킬 수 있다.

기존 부하 분산 연구는 시스템의 어떤 노드가 부하를 분산하는지에 따라 크게 차량, 클라우드 서버, 그리고 엣지 서버로 구분된다[10,11,12].

[10]은 차량이 인접 서버 중 오프로딩 할 서버를 직접 선택함으로써 서버 부하를 분산하는 알고리즘을 제안했다. 엣지 서버들은 가용 자원 정보를 주기적으로 브로드캐스트하고 차량은 이 정보를 수신하고 저장하다가 오프로딩을 원할 시 이를 참조하여 최적의 서버를 선택한다. 서버 선택 과정에는 외부와의 통신이 없으므로 신속한 의사 결정이 가능하지만, 차량이 선택을 위해 인공지능 모델이나 연산 복잡도가 높은 최적화 알고리즘을 직접 수행해야 하므로 자원 제약이 큰 차량에게는 자원 부족 문제가 더욱 심화될 수 있다. 또한 차량이 주기적으로 외부 정보를 모니터링해야 한다는 부담 또한 존재한다.

[11]은 클라우드 서버와 SDN (Software Define Network) 기술을 활용하여 서버 간 부하 분산 문제를 해결했다. 차량이 오프로딩 요청을 클라우드 서버로 전송하면 서버는 사전에 수집하던 엣지 서버 정보를 활용하여 차량에 대한 최적의 연산 서버를 결정한다. 중앙 SDN 컨트롤러가 결정하기에 전역 정보에 기인한 최적

의 결정이 가능하다는 이점이 있으나, 이러한 결정 프로세스에서 클라우드와 통신 과정에 긴 지연이 발생할 수 있다. 이에 따라 차량이 최적의 서버 정보를 수신할 시점에는 서버들의 가용 자원 상태가 또다시 달라졌을 수 있다.

[12]는 엣지 서버들이 직접 부하를 분산하는 방안을 제안했다. 차량은 가장 가까운 서버에게 작업을 전달하면 엣지 서버는 직접 처리할지 혹은 인접 서버나 클라우드에게 다시 전달할지 결정한다. 인접 서버 중에는 상대적으로 가용 자원이 더 많은 서버를 선택하도록 학습되었는데 이를 위해 서버들은 자신의 자원량에 유의미한 변화가 생길 시 이를 인접 서버들에게 알린다.

이처럼 엣지 서버 단에서 직접 부하를 분산하는 연구는 차량이나 클라우드 서버가 분산하는 방식의 한계점을 모두 극복할 수 있다. 먼저 차량은 높은 복잡도의 알고리즘을 수행하거나 외부 정보를 주기적으로 수신하지 않아도 되며, 클라우드와의 통신이 사라지므로 결정 과정에 시간 지연이 줄어든다. 그러나 네트워크의 모든 엣지 서버가 부하 분산에 참여하여 탐욕적으로 행동하기 때문에, 동시에 여러 서버가 하나의 서버를 선택하여 서버 자원의 균형을 안정적으로 유지하기 어려울 수 있다. 따라서 한정된 서버 자원을 보다 효과적으로 관리하는 부하 분산 연구가 필요하다.

본 논문은 엣지 서버 기반 차량 오프로딩 시스템에서 서버 자원 활용률 최대화를 목표로, 심층 강화학습 기반 서버 부하 분산 알고리즘을 제안한다. 본 연구는 엣지 서버 간 클러스터를 형성하여 요청 작업을 클러스터 내부적으로 처리하는 구조를 제안한다. 클러스터는 한 대의 CH(Cluster head) 서버와 여러 대의 CM(Cluster member) 서버로 구성되며 CH 서버는 클러스터 내 전달된 차량의 작업 분산을 담당한다. 제안 시스템에서 차량은 오프로딩 요청을 가장 인접한 엣지 서버에게 전달한다고 가정한다. 이를 수신한 서버는 현재 가용 자원량이 임계 값보다 크다면 직접 처리하지만, 그렇지 않은 경우엔 부하 분산을 위해 CH 서버에게 전달한다. CH 서버는 클러스터 내 서버 중에서 해당 작업을 수행할 연산 서버와 결과를 차량에게 전달할 서버를 결정한다. 또한 연산 서버의 가용 자원 중 할당할 자원 비율을 결정한다. 이러한 클러스터 구조를 통해 국부적으로 발생한 과부하를 보다 효율적으로 관리함으로써 제약적인 자원으로 인한 과부하 현상과 오프로딩 실패 가능성을 최소화할 수 있다. 또한 중앙 서버의 개입 없이도 네트워크 전체의 부하 분산 문제를 클러스터 단위로 분해하여 해결하기 때문에 확장성이 우수하다.

차량 오프로딩 환경은 차량의 높은 이동성, 다양한 작업 프로파일, 서버 가용 자원량 등의 요인들이 동적으로 예측 불가하게 변화한다. 이러한 환경에서 동적 계획법이나 분기 한정법 (Branch and Bound) 같은 최적화 알고리즘은 많은 반복을 통해 최적해를 탐색하기 때문에 수행 시간이 길어 동적인 환경에 적응하며 최적해를 찾는 데 한계가 있다[13]. 반면, 강화학습 알고리즘은 관측 가능한 상태와 최적의 행동 사이의 비선형적이고 복잡한 관계를 효과적으로 학습하며 빠르게 변화하는 환경에 잘 적응한다는 특징이 있다[14].

따라서 본 논문에서는 강화학습을 활용한 엣지 서버 간 부하 분산 알고리즘을 제안한다. 본 논문은 선행 연구 [15]를 기반으로 한다. 선행 연구에서는 부하 불균형 해소를 위한 최적화 문제를 형성하고 제약 조건으로 지연 시간 요구사항을 포함하였으나, 본 연구에서는 선행 연구에서 고려되지 않은 차량의 이동성을 추가로 고려하여 동적인 차량 환경에서의 오프로딩 성공률을 더욱 향상시킬 수 있도록 한다. 또한 시뮬레이션을 통해 제안하는 알고리즘의 성능 평가를 진행함으로써 제안 알고리즘의 부하 분산 능력과 확장성 측면에서의 우수성을 정량적으로 입증하였다.

본 논문의 기여점을 정리하면 다음과 같다. 먼저 본 논문은 엣지 서버 기반 차량 오프로딩 시스템에서 서버 부하를 분산하기 위한 최적화 문제를 정의한다. 제안 시스템에서는 한정된 자원을 효과적으로 관리하기 위해 엣지 서버 간 클러스터를 형성하였으며 클러스터 내 가용 자원량의 편차를 최소화하는 것을 최적화 문제의 목적으로 설정하였다. 더불어 오프로딩 성능에 영향을 미치는 주요 요인들을 제약 조건으로 포함한다. 첫째로 작업의 지연 시간 요구사항을 고려하여 사용자의 QoE를 만족하는 선에서 서버의 부하 균형을 맞추도록 하였다. 또한, 연산 결과를 다시 차량에게 반환하는 전달 서버를 결정함으로써 이동성으로 인한 오프로딩 실패를 최소화하였다.

이어서, 동적으로 변화하는 차량 네트워크 환경에서 최적화 문제의 해답을 찾기 위해 심층 강화학습 기반 서버 부하 분산 알고리즘을 제안한다. 이를 위하여 앞서 정의한 최적화 문제를 MDP(Markov decision process)로 재정의하였고, 최근 우수한 성능을 보이는 정책 경사 기반 강화학습 알고리즘인 TD3모델을 활용하여 최적의 행동을 찾아낸다.

제안하는 방안의 성능 평가를 위해 알고리즘을 구현하였고 수렴을 확인했다. 또한 4가지의 비교 알고리즘과 성능 비교를 통해 제안하는 방안이 엣지 서버의 가용 자원 불균형 문제를 효과적으로 해결함을 확인했다. 특히 차량 수가 증가하거나 작업의 다양성이 증가하는 상

황에서도 높은 성공률을 보인 것을 통해 우수한 확장성을 확인했다.

본 논문의 구성은 다음과 같다. 1장의 서론에 이어 2장에서는 제안하는 엣지 클러스터 구조의 차량 오프로딩 시스템 및 모델을 설명한다. 3장에서는 서버 부하 분산 최적화 문제를 정의한다. 4장에서는 최적화 문제를 MDP 문제로 재정의한 후 강화학습 기반의 엣지 서버 부하 분산 알고리즘을 설명한다. 5장에서는 실험을 통한 성능 분석을 보이고, 6장에서 결론을 맺는다.

Ⅱ. 시스템 모델

이 장에서는 본 논문에서 제안하는 엣지 서버 기반 차량 오프로딩 시스템을 설명한다. 먼저 시스템 구조를 설명하고 이어서 통신 모델, 연산 모델, 서버 부하 모델, 마지막으로 차량 이동성 모델에 대해 설명한다.

2.1 엣지 클러스터 기반 차량 오프로딩 시스템

그림 1은 제안하는 시스템을 나타낸다. 격자 형태의 도로 환경에 RSU(Road side unit)는 일정한 거리를 두고 그림과 같이 배치되어 있으며 모든 RSU에는 엣지 서버가 탑재되어 있다. 차량들은 격자 도로를 따라 이동하다가 오프로딩을 원할 시 가장 가까운 엣지 서버에게 작업을 전달한다. 차량 i의 작업을 $Q_i = \{D_i, G_i, T_i^{\max}\}$라고 할 때, D_i는 작업 데이터의 크기, G_i는 작업 처리를 위해 필요로 하는 연산 자원량, 그리고 $T_i^{\max}$는 작업의 최소 지연 시간 요구사항을 나타낸다.

제안 시스템에서 엣지 서버들은 사전에 정의된 클러스터 단위로 작업을 처리한다. 클러스터는 한 대의 CH 서버와 여러 대의 CM 서버로 구성된다. 클러스터를 $C = \{0,1,...,K\}$라고 표현할 때, 0은 CH 서버를 나타내며, 1부터 K는 CM 서버이다. 따라서, 클러스터 내에

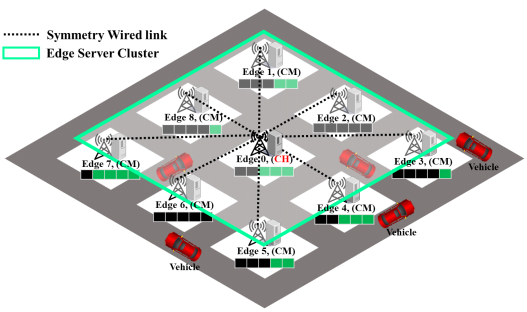


그림 1. 엣지 서버 클러스터 기반 차량 오프로딩 시스템
Fig. 1. Edge server cluster-based vehicle offloading system

는 총 $K+1$개의 서버가 있다. 그림 1에서는 9개의 서버가 하나의 클러스터로 구성된 상태이다.

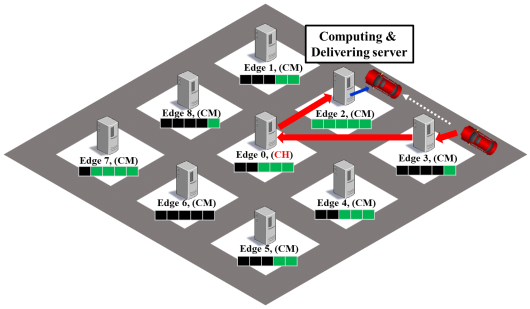
차량으로부터 작업 Q_i를 최초로 수신한 엣지 서버를 벡터 $\mathbf{a_i}$로 표현할 때, 벡터의 크기는 $K+1$이며 각 원소는 $a_{i,m} \in \{0,1\}, \forall m \in C$이다. $a_{i,m}=1$은 클러스터의 m번째 서버가 작업을 최초로 수신함을 나타내며 $a_{i,m}=0$은 그렇지 않음을 나타낸다. 차량에게 작업을 수신한 엣지 서버는 ($a_{i,m}=1$) 먼저 자신의 가용 자원량이 임계값 ψ보다 큰지 확인하고 그렇다면 자체적으로 작업을 처리한다. 하지만 ψ보다 작다면 해당 작업을 CH 서버에게 전달한다. CH 서버는 클러스터 내 작업 부하 분산을 담당하며 작업 Q_i에 대해 아래의 세 가지의 주요한 결정을 내린다.

1) 작업 연산 서버: 클러스터 C의 모든 서버 중 작업을 처리할 엣지 서버이다. 다른 서버보다 가용 자원량에 여유가 있고 작업을 $T_i^{\max}$내에 처리가 가능한 서버 중 선택된다. 작업 Q_i의 연산 서버 결정을 벡터 $\mathbf{d_i}$로 표현할 때, 벡터의 크기는 $K+1$이며 각 원소는 $d_{i,m} \in \{0,1\}, \forall m \in C$이다. 만약 $d_{i,m}=1$이면 클러스터의 m번째 엣지 서버가 작업의 연산을 담당하는 것을 나타내며 $d_{i,m}=0$이면 그렇지 않음을 나타낸다. 결정을 위해 CH 서버는 클러스터 내 서버들의 가용 자원량 정보를 유지해야 하므로 CM 서버들은 자체 가용 자원량에 유의미한 변화가 발생 시 이를 CH 서버에게 알린다.

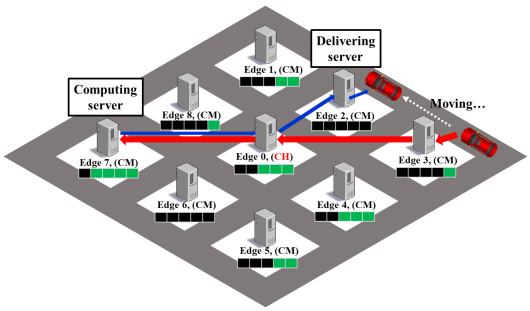
2) 결과 전달 서버: 클러스터 C의 모든 서버 중 작업 Q_i의 결과를 다시 차량에게 반환할 서버이다. 차량은 높은 이동성을 가지고 있으므로 오프로딩을 요청한 시점과 결과를 수신하는 시점에 가장 가까운 엣지 서버가 다를 수 있다. 따라서 이동성을 반영하여 작업 결과를 전달할 서버를 결정한다. 작업 Q_i의 결과 전달 서버 결정을 벡터 $\mathbf{r_i}$로 표현할 때, 벡터의 크기는 $K+1$이며, 각 원소는 $r_{i,m} \in \{0,1\}, \forall m \in C$이다. 만약 $r_{i,m}=1$이면 클러스터의 m번째 엣지 서버가 작업의 결과를 차량에게 최종적으로 전달하는 것을 의미하며 $r_{i,m}=0$이면 그렇지 않음을 나타낸다.

3) 자원 할당 비율: 자원 할당 비율 f_i는 작업 연산 서버의 가용 자원량 중 작업 Q_i에게 할당할 자원 비율이다.

CH 서버는 작업 연산 서버와 결과 전달 서버를 동일한 서버로 결정할 수 있으나, 상황에 따라 서로 다른 두 서버가 각각의 역할을 담당할 수도 있다. 그림 2는



(a) Scenario with the same computing server and delivering server



(b) Scenario with different computing and delivering servers

그림 2. 결정에 따른 오프로딩 데이터 통신 흐름
Fig. 2. Offloading data communication flows based on decision

이러한 두 개의 시나리오를 나타낸다. 차량이 처음 오프로딩 한 시점에는 3번 서버에 연결되어 있으나 연산이 처리되는 동안 이동하여 결과를 전달받을 시점에는 2번 서버와 연결되어 있다.

먼저 그림 2(a)는 작업 연산 서버와 결과 전달 서버가 동일하게 결정된 시나리오이다. 그림의 빨간 선은 오프로딩 데이터 전달 흐름이고 파란 선은 결과 전달 흐름을 나타낸다. CH 서버의 두 가지 서버 결정이 모두 2번 서버이므로 CH 서버는 2번 서버에게 작업 데이터를 전달하고 2번 서버가 작업을 처리한 뒤 결과를 직접 차량에게 전달한다. 그림 2(b)에서는 2번 서버의 가용 자원량이 작업을 처리하기에 충분치 않은 상황이므로 CH 서버는 7번 서버를 작업 연산 서버로 결정한다. 따라서 CH 서버는 오프로딩 데이터를 7번 서버에게 전달하고, 7번 서버는 작업을 처리한 뒤 결과를 CH 서버에게 반환한다. CH 서버가 2번 서버에게 결과를 전달하면 2번 서버가 최종적으로 차량에게 결과를 전달한다.

2.2 통신 모델

제안하는 클러스터 구조의 서버 부하 분산 알고리즘에선 그림 2(b)와 같이 최대 2회의 서버 간 오프로딩 데이터 통신이 발생할 수 있다. 첫 번째는 차량에게 작업을 받은 서버가 부하 분산을 위해 CH 서버에게 전달하는 과정이고, 두 번째는 CH 서버가 작업 연산 서버에게 전달하는 과정이다. 2.2절에서는 위의 두 단계 통신에서 발생하는 시간 지연을 모델링한다.

먼저, 인접한 서버는 유선망으로 연결되어 있으며 각 링크의 가용 대역폭은 대칭성을 띤다고 가정한다. 가용 대역폭은 단위 시간마다 예측 불가하게 변화하나, 하나의 타임 슬롯 내에서는 변동되지 않는다고 가정한다. 또한 연산 결과는 입력 데이터에 비해 매우 작기 때문에 결과 전달 지연은 무시한다.

인접 서버 m과 n 사이의 유선 링크의 대역폭을 이라 $b_{m,n}$ 할 때, 차량에게 최초로 작업을 수신한 서버가 CH 서버에게 전달하는 과정의 통신 지연 T_1^{trans}은 (1)로 계산할 수 있다.

$$T_1^{trans} = \sum_{m=1}^{K} a_{i,m} \frac{D_i}{b_{m,0}} \tag{1}$$

다음으로, CH 서버가 작업 연산 서버에게 전달하는 과정의 통신 지연 T_2^{trans}는 (2)로 계산한다.

$$T_2^{trans} = \sum_{m=1}^{K} d_{i,m} \frac{D_i}{b_{0,m}} \tag{2}$$

(1)을 통해, CH가 최초로 작업을 수신한 경우 T_1^{trans}이 발생하지 않음을 알 수 있으며 (2)를 통해 CH가 작업 연산 서버로 결정되면 T_2^{trans}가 발생하지 않음을 알 수 있다.

결과적으로 서버 간 부하 분산 과정의 총 통신 지연 T_i^{trans}는 (3)과 같다.

$$T_i^{trans} = T_1^{trans} + T_2^{trans} \tag{3}$$

2.3 연산 모델

차량의 높은 이동성과 다양한 작업 프로파일로 인해 엣지 서버별 가용 자원량은 빠르게 변화한다. 서버 m의 가용 자원량을 F_m이라 할 때, Q_i가 할당받은 연산 자원량 z_i는 (4)로 계산한다.

$$z_i = \sum_{m=0}^{K} d_{i,m} F_m f_i \tag{4}$$

이에 따라 Q_i 연산 시간 T_i^{comp}은 (5)와 같다.

$$T_i^{comp} = \frac{G_i}{z_i} \tag{5}$$

앞선 통신 모델과 연산 모델에 따라, 소요되는 총 시간 T_i는 (6)으로 계산할 수 있다.

$$T_i = T_i^{trans} + T_i^{comp} \tag{6}$$

2.4 서버 부하 모델

서버 간 가용 자원량의 균일성을 정량적으로 평가하기 위해 표준편차를 부하 균형의 지표로 활용한다. 표준편차가 낮을수록 서버 간 부하가 균등함을 반영하며 이는 특정 엣지 서버가 과부하 되거나 혹은 매우 낮은 자원 활용률을 갖지 않고 모든 서버가 비슷한 부하를 갖고 있음을 의미한다. 클러스터 C의 평균 가용 자원량 $\overline{F}$는 (7)로 계산한다.

$$\overline{F} = \frac{1}{K+1}\sum_{m=0}^{K} F_m \tag{7}$$

클러스터 C의 가용 자원량 표준편차를 σ라 할 때 (8)에 따라 계산할 수 있다.

$$\sigma = \sqrt{\frac{\sum_{m=0}^{K}(F_m - \overline{F})^2}{K+1}} \tag{8}$$

2.5 차량 이동성 모델

차량 i의 이동성 모델을 $\{x_i, y_i, \theta_i, v_i\}$로 표현할 때, x_i와 y_i는 차량의 오프로딩 요청 시점의 위치이고 θ_i는 이동 방향, 마지막으로 v_i는 이동 속도를 나타낸다. 그림 1과 같은 격자 도로 공간에서 모든 차량은 반드시 하나 이상의 교차로를 향해 이동하고 있음을 알 수 있다. 차량의 이동 방향을 고려하였을 때 가장 가까운 교차로를 $I = \{x_I, y_I\}$라 할 때, 차량 i와 교차로 I 사이의 최단 거리 $d_{residual}$은 (9)로 계산한다.

$$d_{residual} = \sqrt{(x_i - x_I)^2 + (y_i - y_I)^2} \tag{9}$$

따라서 차량 i가 해당 교차로 진입까지 남은 예상 잔여 시간 $t_{residual}$은 (10)와 같다.

$$t_{residual} = \frac{d_{residual}}{v_i} \tag{10}$$

Ⅲ. 서버 부하 분산 최적화 문제 정의

이 장에서는 엣지 서버 기반 차량 오프로딩 환경에서 서버 부하 분산 최적화 문제를 정의한다. 작업별 지연 시간 요구사항과 차량의 이동성 제약 조건 하에서, 서버 자원 부하 불균형을 최소화하는 것을 목표로 작업 연산 서버 $\mathbf{d_i}$, 결과 전달 서버 $\mathbf{r_i}$, 그리고 자원 할당 비율 f_i를 결정한다. 최적화 문제는 (11)과 같다.

$$\begin{aligned}
\mathbf{P1}: & \max_{\mathbf{d_i}, \mathbf{r_i}, \mathrm{f_i}} \log(\sigma_t - \sigma_{t+1}) \\
& s.t.\ C1: 0 \le f_i \le 1 \\
& C2: 0 \le F_m \le F_m^{\max},\ \forall m \in C \\
& C3: T_i \le T_i^{\max} \\
& C4: d_{i,m} \in \{0,1\},\ \forall m \in C \\
& C5: r_{i,m} \in \{0,1\},\ \forall m \in C \\
& C6: \sum_{m=0}^{K} d_{i,m} = 1,\ \forall m \in C \\
& C7: \sum_{m=0}^{K} r_{i,m} = 1,\ \forall m \in C
\end{aligned} \tag{11}$$

$\mathbf{P1}$의 목적 함수에서 σ_t는 시간 스텝 t에서의 가용 자원량 표준편차이고 σ_{t+1}은 부하 분산 수행 이후 가용 자원량의 표준편차를 의미한다. 목적 함수는 시간 t에서의 결정 변수 선택에 따른 클러스터 내 자원 표준편차 감소량을 최대화하는 것을 의미한다. 이때 로그 함수를 적용함으로써 자원 할당 과정에서 목적 함수 최대화를 위해 필요 이상의 자원을 할당하는 것을 제한한다. CH 서버는 주어진 작업에 대하여 제약 조건을 만족하는 선에서 클러스터 내 자원 간 편차를 최대로 줄이는 방법을 결정하게 되고, 궁극적으로는 시스템의 자원 활용률 및 서비스 수용률을 최대화하는 것을 목표한다.

$C1$에서는 자원 할당 비율이 0에서 1 사이의 값을 갖는다. $C2$에서는 서버 m의 가용 자원량이 0보다 크거나 같고 최대 자원량 $F_m^{\max}$보다 작거나 같도록 설정된다. $C3$에서 작업 처리 시간 제약 조건을 설정함으로써, 사용자의 요구사항을 충족하도록 하는데, 각 작업의 최대 처리 시간을 규정하여, 최적화 과정에서 부하 균형을 유지하면서도 시간제한을 초과하지 않도록 한다. $C4$와 $C5$는 작업 연산 및 결과 전달 서버 결정에 대한 이진 변수 제약 조건이고, $C6$과 $C7$은 각각 작업 연산과 결과 전달을 단일 서버에만 할당할 수 있다는 제약이다.

하지만 P1은 $C4$와 $C5$ 제약 조건에 의해 이진 결정 변수들을 포함하므로 혼합 정수 프로그래밍(Mixed Integer Programming, MIP) 문제로 정의되어 비볼록(non-convex) 특성을 가진다. 또한, 엣지 서버 간 통신 링크 대역폭이 예측할 수 없이 변하고, 다양한 종류의 오프로딩 작업을 고려해야 하는 환경적 특성 때문에 문제의 복잡도는 더욱 증가한다. 이러한 환경에서 기존 최적화 기법들은 최적해를 찾는 데 몇 가지 한계를 보일 수 있다. 먼저 MIP 문제는 NP-hard 문제로 알려져 전역 최적해를 찾기 위한 탐색 공간이 지수적으로 증가하여 기존 알고리즘을 활용하면 연산 자원과 시간 측면에서 비효율적일 수 있다[13]. 또한 기존 최적화 알고리즘은 수많은 반복을 통해 최적해를 찾기에 빠르게 변화하는 환경에 즉각 대응하고 최적해를 찾는 것에 한계가 있다[14]. 이에 대한 해결책으로 본 연구는 강화학습 기반 부하 분산 알고리즘을 제안한다.

Ⅳ. 강화학습 기반 엣지 서버 부하 분산 알고리즘

본 논문은 강화학습을 이용해 논문에서 정의한 최적화 문제를 해결한다. 4.1절에서는 강화학습 이론을 설명하고 4.2절에서 엣지 서버 부하 분산 최적화 문제를 MDP 문제로 재정의한다. 마지막으로 4.3절에서 TD3 모델을 활용한 서버 부하 분산 알고리즘을 설명한다.

4.1 강화학습

강화학습은 에이전트가 환경과 직접 상호작용하며 최적의 행동 정책을 학습하는 알고리즘이다[16]. 에이전트는 환경을 관측하여 상태 s를 확인하고 정책 π에 따라 행동 a를 결정한다. 이후 행동의 결과로 보상 r을 얻고 환경은 새로운 상태 s'으로 전이한다. 이러한 일련의 과정을 하나의 학습 경험이라 하며 $\langle s,a,r,s' \rangle$와 같이 표현할 때, 최적의 정책을 얻기 위해서는 적절한 탐험과 탐색을 균형을 맞춰 다양한 학습 경험을 쌓는 것이 중요하다. 궁극적 목표는 최적의 정책 π^*을 찾아 주어진 환경에서 누적 보상을 최대화하는 것이다.

이러한 학습 과정에 기인하여, 특정 문제에 강화학습 알고리즘을 적용하기 위해서는 MDP 모델링이 필수적이다[17]. MDP란 환경을 수학적으로 모델링하는 기법으로 상태(State), 행동(Action), 보상(Reward)으로 정의한다[17]. 상태는 에이전트가 학습하고자 하는 환경에서 관측 가능한 정보이며 행동은 상태를 바탕으로 취할 수 있는 결정이다. 행동에 따라 환경이 새로운 상태로 전이한다. 마지막으로 보상은 행동에 대한 평가를 나타낸다. 에이전트는 보상을 통해 특정 상태 행동 쌍$\langle s,a \rangle$이 가진 가치를 추정하고 업데이트한다.

DRL(Deep Reinforcement Learning)은 기존 강화학습 알고리즘에 DNN(Deep Neural Network)을 적용해 $< s,a >$의 가치를 근사하는 알고리즘이다[18]. 인공신경망의 입력층에 상태와 행동을 넣으면 은닉층을 거쳐 마지막 결과 층에서 예상 가치 값이 도출된다[18]. 따라서 상태 및 행동 공간이 크거나 연속적이어도 가치 예측이 가능하며, 경험한 적 없는 환경을 마주해도 인공신경망이 근사해를 도출한다[18]. 또한 상태와 행동 사이의 복잡한 비선형적 패턴을 효과적으로 학습할 수 있어 다양한 환경에서 높은 성능을 보인다.

다양한 DRL 모델 중, TD3는 DDPG(Deep Deterministic Policy Gradient)의 확장 모델로서 높은 성능과 안정적인 학습을 달성한다[19]. TD3는 정책 경사 기반 모델인 AC(Actor Critic)에서도 과대평가 문제가 발생할 수 있음을 지적하였고 이를 해결하기 위해 서로 다른 가중치를 가지는 두 개의 Critic 네트워크를 사용하는 구조를 제안했다. TD3는 가치함수를 안정적으로 유지하고 타겟 정책 업데이트를 통해 정책 최적화를 수행하며 학습하여 연속적인 행동 공간에서도 효과적으로 작동한다. 따라서 본 연구에서는 TD3 모델을 활용한 엣지 서버 간 부하 분산 알고리즘을 제안한다.

4.2 MDP 문제 정의

제안 시스템에서는 CH 서버가 에이전트로 동작하며 MDP는 아래와 같이 정의된다.

1) 상태 공간: 시간 t에 CH 서버가 관측하는 상태 공간 $x_i(t)$은 (12)로 표현된다.

$$x_i(t)=\begin{bmatrix} D_i, G_i, T_i^{\max}, x_i, y_i, \theta_i, v_i, \\ t_{residual}, F_0, F_1, \cdots, F_m, \cdots, F_K, \\ b_{0,1}, \cdots, b_{0,m}, \cdots, b_{0,K} \end{bmatrix} \tag{12}$$

먼저 작업 Q_i에 대한 정보, 차량 이동성 모델, 차량 이동 방향을 고려한 인접 교차로까지 예상 잔여 시간이 포함된다. 이와 더불어 클러스터 내 엣지 서버의 가용 자원량과 CH와 CM 사이의 유선 링크의 채널 대역폭이 상태 공간으로 표현된다.

2) 행동 공간: 시간 t에 CH 서버가 결정하는 행동 $u_i(t)$은 아래 (13)과 같다.

$$u_i(t)=\begin{bmatrix} d_{i,0}, d_{i,1}, \cdots, d_{i,m}, \cdots, d_{i,K}, \\ r_{i,0}, r_{i,1}, \ldots r_{i,m}, \ldots, r_{i,K}, f_i \end{bmatrix} \tag{13}$$

Q_i에 대한 작업 연산 서버와 결과 전달 서버, 그리고

자원 할당 비율이 행동 공간으로 표현된다.

3) 보상 함수: 보상 함수는 오프로딩 성공 여부에 따라 다르게 설정되며 성공 여부는 두 가지 세부 기준에 따라 판별된다. 첫 번째는 총 처리 시간 T_i가 작업 프로파일에 명시된 지연 시간 요구사항 $T_i^{\max}$을 만족했는지 확인하는 것이며 이는 P1의 과 $C3$와 같다. 두 번째로는 결과 전달 성공 여부이다. 차량이 결과를 수신하는 시점에 가장 가까운 엣지 서버를 $e_i = \{e_{i,m} | e_{i,m} = \{0,1\}, \forall m \in C\}$로 나타낼 때, CH에 의해 결정된 결과 전달 서버 $\mathbf{r_i}$가 이와 일치하는지를 의미하며 이러한 성공 조건을 (14)로 표현 가능하다.

$$\sum_{m=0}^{K} e_{i,m} r_{i,m} = 1 \tag{14}$$

위의 두 가지 성공 조건을 모두 만족시켜 오프로딩을 성공한 경우에는 P1의 목적 함수가 MDP 문제의 보상 함수로 설정된다. 다시 말해, CH 서버의 시간 t에서의 결정이 클러스터 내 전체 가용 자원 표준편차를 감소하는데 기여한 정도에 따라 보상을 받는다. 이에 반하여 오프로딩에 실패하면 보상을 받지 못한다. 이를 나타내는 보상 함수 $R_i(t)$는 (15)로 표현할 수 있다.

$$R_i(t) = \begin{cases} \alpha + \beta log\left(\sigma_t - \sigma_{t+1}\right), success \\ 0, failure \end{cases} \tag{15}$$

α는 오프로딩 성공에 대한 기본 보상 값을 나타내는 상수이며, β는 부하 균형에 기여한 정도에 따라 지급되는 추가 보상에 대한 가중치이다.

4.3 TD3 모델 기반 부하 분산 알고리즘

그림 3은 TD3 알고리즘의 구성 요소와 전체 학습 과정을 나타낸다. 에이전트는 두 개의 critic 네트워크와 한 개의 actor 네트워크를 가지고 있으며 actor와 critic 네트워크들은 각각 target 네트워크를 하나씩 가지고 있다. 네트워크의 역할은 아래와 같다.

1) Critic 네트워크: 상태 행동 쌍에 대해 예상되는 가치 값 Q를 도출한다. 이는 에이전트의 현재 정책 아래에서 특정 행동을 취했을 때 기대할 수 있는 미래 보상의 총합을 나타낸다. DDPG에서는 단일 critic 네트워크를 활용한 반면, TD3에서는 두 개의 독립적인 critic 네트워크를 사용함으로써 가치 추정에서 발생할 수 있는 과대평가 문제를 완화한다. 두 Critic 네트워크의 Q 값 추정치 중 최솟값을 사용하는 접근 방식을 통해

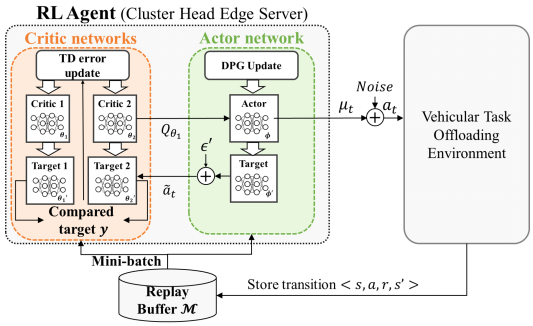


그림 3. TD3 알고리즘 구성 요소 및 학습 과정
Fig. 3. TD3 Algorithm components and learning process

학습이 보다 안정적으로 진행될 수 있다[19].

2) Actor 네트워크: 현재 상태를 입력으로 받아 최적의 행동을 도출하는 네트워크로, 최대 보상을 얻을 수 있는 행동 전략을 학습한다.

3) Target 네트워크: Target Critic과 Target Actor 네트워크는 각각 Critic 및 Actor 네트워크의 목푯값을 도출하며 이 값은 학습 과정에서 손실 값 계산에 활용된다. 이처럼 손실 계산을 위해 별도의 target 트워크를 두는 방식은 급격한 정책 업데이트나 과대평가 문제를 해결한다[20].

알고리즘 1은 TD3의 학습 과정을 나타낸다. 먼저 TD3 에이전트를 구성하는 2개의 critic 네트워크의 가중치 θ_1, θ_2와 actor 네트워크의 가중치 ϕ를 초기화하고 target 네트워크들의 가중치는 각 네트워크와 동일하게 초기화한다. 훈련은 수 차례의 에피소드를 반복하며 진행되어 많은 연산 및 스토리지 자원을 필요로 하므로 클라우드 환경에서 오랜 시간에 걸쳐 훈련된다. 이후 각 CH 서버들은 사전 훈련된 모델을 유지하며 독립적으로 부하 분산 알고리즘을 수행한다.

학습을 위해 에이전트는 환경에서 상태 s를 관측한다. 이후 행동을 결정하는 과정에서 입실론 그리디 알고리즘을 활용함으로써 학습 초반에 에이전트가 다양한 상태 행동 공간을 탐색할 수 있도록 한다. 입실론 그리디 알고리즘에서는 0과 1 사이의 숫자 ρ를 무작위로 추출한 뒤, ϵ과 비교하여 ρ가 더 작다면 랜덤 행동을 취하고 ρ가 크다면 자신의 정책에 따라 최적의 행동 결정을 내린다. 또한 ϵ 값을 에피소드가 증가함에 따라 지수 감쇠함으로써 학습 초반에는 많은 탐색을 진행하고, 후반에는 안정성을 높인다. 이후 결정된 행동에 대한 보상 r을 받고 새로운 상태 s'로 전이하게 되면 하나의 경험 트랜지션$\langle s,a,r,s' \rangle$을 replay buffer에 저장한다. 학습 데이터를 축적하다가 critic 네트워크의 업데이

Algorithm 1: TD3-based in-cluster edge server load balancing algorithm

Initialize two critic networks $Q_{\theta_1}, Q_{\theta_2}$, and actor network π_ϕ with random weights θ_1, θ_2, ϕ
Initialize target networks $\theta_1' \leftarrow \theta_1$, $\theta_2' \leftarrow \theta_2, \phi' \leftarrow \phi$
Initialize replay memory $\mathcal{M}$
for *each episode* $e = 1, 2, \ldots$ **do**
  for *each step* $t = 1, 2, \ldots$ **do**
    Observe state s from the environment
    Generate a random number ρ from [0, 1]
    if $\rho \leq \epsilon$ **then:**
      Select action a with **random policy** and get reward r and transit to new state s'
    else:
      Select action a with **policy** π adding noise and get reward r and transit to new state s'
    Store the transition $< s, a, r, s' >$ in $\mathcal{M}$
    if t mod C **then**
      Sample mini batch of N transitions $< s, a, r, s' >$ from $\mathcal{M}$
      Update critic networks by minimizing the loss
    end if
    if t mod D **then**
      Update ϕ by the deterministic policy gradient
      Update target networks:
    end if
  end for
end for

트 주기가 되면 replay buffer에서 학습에 사용할 N개의 트랜지션을 무작위 추출하여 미니 배치를 구성하고 critic 네트워크의 가중치를 업데이트하며 actor 네트워크 또한 정해진 주기마다 업데이트한다. 마지막으로 target 네트워크들이 업데이트된다.

V. 성능 평가

이 장에서는 제안 알고리즘의 성능을 비교 분석한다. 먼저 5.1절에서는 실험 환경을 설명하고, 5.2절에서 제안 방안의 수렴 가능성을 비롯해 4가지 알고리즘과 성능 비교한 결과를 분석한다.

5.1 실험 환경 설정

실험은 그림 1과 같은 격자 형태의 1차선 양방향 도로 교통 시나리오에서 수행되었다. 9대의 엣지 서버가 일정한 거리를 두고 배치되어 있고 서로 다른 가용 자원량을 가진다. 차량들은 도로를 따라 이동하며 가장 가까운 엣지 서버에게 오프로딩 데이터를 전달한다고 가정한다. 에피소드마다 엣지 서버의 가용 자원량은 평균 50GHz, 표준편차 25GHz를 따르도록 랜덤하게 초기화된다. 보다 정확한 성능 분석을 위해 모든 성능 분석 실험은 100회 반복 실험 후 평균치를 사용하였다. 표 1은 실험에서 사용된 파라미터 값을 정리한 것이다.

또한 제안하는 TD3 기반 부하 분산 알고리즘을 구현하기 위하여 PyTorch로 작성된 강화학습 모델 오픈소스 라이브러리를 활용하였다[21]. TD3를 구성하는 2개의 Critic 네트워크와 1개의 Actor 네트워크 및 3개의 Target 네트워크는 모두 완전 연결 계층으로 이루어진 DNN을 사용했다. Critic 네트워크는 2개의 은닉층을 가지고 있으며, Actor 네트워크는 입력층 다음으로 1개의 완전 연결 계층을 거친다. 이후 에이전트가 결정해야 하는 세 가지 사항에 대해 더욱 깊은 특징을 학습하기 위해, 앞선 공유 계층의 결과를 각 결정을 위한 3개의 독립적인 완전 연결 계층으로 입력한다. 각각의 완전 연결 계층으로부터 도출된 결과가 에이전트의 최종 결정이며 모든 은닉층은 128개의 뉴런을 가지고 있다. 학습 초기 다양한 탐색을 독려하기 위하여 초기 입실론 값을 0.8로 설정하였고 에피소드가 증가함에 따라 입실론에 0.995를 곱하여 점차 탐색의 비율을 감소시켰다. 표 2는 TD3 강화학습 모델과 관련된 파라미터 값을 정리한 것이다.

본 논문에서는 아래 네 가지 엣지 서버 작업 처리 알고리즘과의 비교를 통해 다양한 환경에서의 제안 알고리즘의 성능을 분석한다. 네 가지 비교 알고리즘은 아래와 같다.

Nearest: 차량에게 작업을 처음 수신한 엣지 서버가 항상 연산을 담당하는 알고리즘이다.

Random: 차량에게 작업을 수신한 엣지 서버가 우

표 1. 차량 오프로딩 환경 파라미터
Table 1. Parameters of vehicular task offloading environment

Parameter	Value
Time slot (sec)	0.1
Simulation area	3.0
Number of edge servers	9
RSU coverage (meter)	500
Bandwidth between edge servers (Gbps)	[0.07, 0.1]
Average available computing resources (GHz)	50
Standard deviation of available computing resources (GHz)	25
Data size of task (MB)	[0.01, 1]
Number of required CPU cycles of task (Gigacycle)	[0.0025, 2]
Delay constraint of task (sec)	[0.5, 2]
Number of vehicles	30

표 2. TD3 강화학습 모델 파라미터
Table 2. Parameters of TD3 DRL model

Parameter	Value
Learning rate (critic network)	0.003
Learning rate (actor network)	0.0003
Initial epsilon (ϵ)	0.8
Epsilon decay coefficient	0.995
Minimal epsilon	0.0001
Mini-batch size	256
Maximum replay buffer size	500000
Maximum episodes	5000
Parameter update frequency for target network	3

선적으로 연산을 담당하나, 자체 가용 자원만으로 작업 요구사항을 만족하기 어려운 경우, 1-hop 이웃 서버 중 한 서버를 무작위하게 선택하고 전달하여 처리를 요청하는 알고리즘이다.

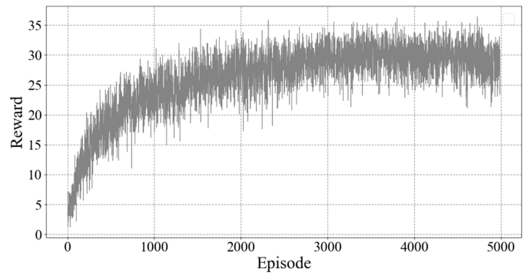
Greedy: 차량에게 작업을 수신한 엣지 서버 기준으로 자신을 포함한 1-hop 이웃 서버 중 가장 가용 자원량이 많은 엣지 서버에게 작업을 전달하여 처리를 요청하는 알고리즘이다.

DDPG: 제안 알고리즘과 유사하게 강화학습을 기반으로 클러스터 내에서 작업 연산 서버, 결과 전달 서버, 할당 자원 비율을 결정하나, DDPG 강화학습 알고리즘을 활용한다. DDPG는 TD3와 달리 1개의 critic 네트워크만을 가진다.

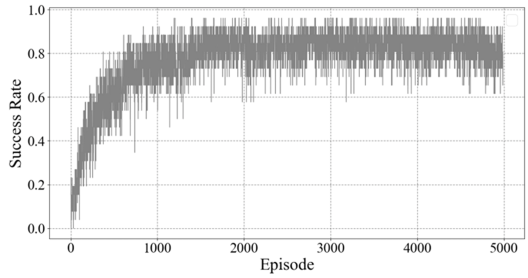
5.2 실험 결과

그림 4는 제안 알고리즘의 훈련 과정 중 주요 성능 지표 변화를 나타낸 것이다. 그림 4(a)는 보상 값을 나타내고 4(b)는 오프로딩 성공률을 보인다. 그림 4(a)와 4(b)에서는 훈련이 거듭됨에 따라 보상 값과 성공률이 점차 증가하다가 에피소드 2000부터 수렴함을 확인할 수 있다. 그림 4에서 보상과 오프로딩 성공률이 모두 안정적으로 수렴한 것을 통해 제안하는 알고리즘의 유한한 횟수의 반복 내에 수렴함을 확인했다.

그림 5는 각 알고리즘을 수행하며 측정한 서버별 평균 가용 자원량을 나타낸다. 모든 알고리즘에 대해 그림 5(a)와 같이 동일한 초기 자원량을 설정한 뒤 차량이 100대인 환경에서 알고리즘을 20초 수행하면서 20초 동안 각 서버별 가용 자원량의 평균을 측정했다. 그림 5(a)를 통해 알 수 있듯이 알고리즘 수행 전에는 서버 간 불균형이 심한 시나리오로, 특히 1번, 5번 서버의 가용 자원량은 매우 작고 0번, 7번, 8번 서버는 상대적



(a) Rewards according to episodes



(b) Success rate according to episodes

그림 4. 제안 알고리즘의 수렴
Fig. 4. Convergence of the proposed algorithm

으로 많은 자원을 가지고 있다. 그림 5(b)의 Nearest는 인접 서버 중 자원이 여유로운 서버가 있더라도 부하 분산을 전혀 하지 않기 때문에 시스템 내 가용 자원 활용률이 낮아 수행 후에도 자원 편차가 여전히 크며 특히 서버 1번과 5번의 자원 부족 현상이 더욱 심화된 것을 볼 수 있다. 그림 5(c), 5(d)의 Random과 Greedy는 작업을 분산하므로 인접 서버 자원을 활용하나, 적절한 자원 할당량을 결정하는 메커니즘이 구현되지 않았기 때문에 시스템 가용 자원을 효율적으로 활용하는 것에 한계가 있어 모든 서버의 가용 자원량이 매우 낮아짐을 확인할 수 있다.

이에 반하여 그림 5(e)와 그림 5(f)처럼 강화학습을 적용한 알고리즘들은 알고리즘 수행 후 서버 간 가용 자원량의 편차가 크게 줄어든 것을 확인할 수 있다. 구체적으로 초기 자원량이 작았던 1번, 5번 서버의 자원량은 유지된 반면, 자원량이 여유로웠던 서버들의 자원이 비교적 크게 줄어든 결과를 통하여, 클러스터 헤드가 클러스터 내 환경을 관측하고 최적의 부하 분산 결정을 통해 효과적으로 부하를 분산함을 알 수 있다. 특히 자원 할당량 결정 알고리즘에 의해 작업의 요구사항을 만족하는 선에서 서버의 가용 자원을 최대한 효율적으로 할당하므로 제안 알고리즘 수행 후의 가용 자원은 Greedy 알고리즘 대비 각 3배 이상 남아있음을 알 수 있다.

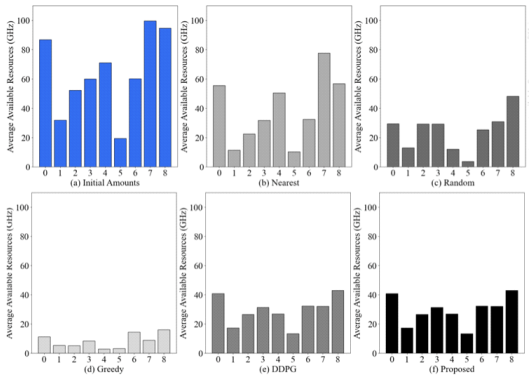


그림 5. 알고리즘 수행 중 서버별 평균 가용 자원량
Fig. 5. Average available resources per edge server during algorithm execution

그림 6은 차량 분포에 따른 알고리즘별 오프로딩 성공률을 나타낸다. 이 실험을 위해서는 오프로딩 요청 차량의 분포를 세 가지로 변화시켜 보았다. 시나리오 (a)는 차량 분포가 한 곳에 밀집된 시나리오로, 차량들이 그림 1의 환경에서 0번 서버 관할 구역 근방에만 위치한 시나리오의 결과이다. 따라서 0번 서버가 대부분의 오프로딩 요청을 수신하게 된다. 시나리오 (b) 역시 차량 분포가 특정 지역에 밀집되긴 하였으나 앞선 시나리오보다는 밀집도가 낮은 상황으로, 그림 1번의 0번, 1번, 3번, 4번 서버 근방에만 차량들이 위치한 시나리오이다. 마지막으로 시나리오 (c)는 차량들이 네트워크 전체에 균일하게 랜덤 분포된 시나리오이다. 먼저 시나리오 (c)에서 제안 방안을 제외한 세 가지 방안은 70%의 유사한 성공률을 보였다. 그러나 Nearest는 서버가 과부하 되어도 인근 서버로의 부하 분산이 불가하다는 특성에 따라 트래픽 불균형이 심해질수록 성능 저하 폭이 컸으며 시나리오 (a) 에서는 가장 낮은 오프로딩 성공률을 보였다. Random과 Greedy는 이웃 서버로의 부하 분산이 이뤄지므로 트래픽 불균형이 심해지더라도 Nearest보다 나은 성능을 보이나, Random은 작업을 자체 처리하기 어려운 상황에서 이웃 서버 중 무작위로 선택하기 때문에 자원이 더욱 고갈된 서버에게 전달될 수 있어 밀집도가 커질수록 성공률이 낮아진다. Greedy는 이웃 서버들과 가용 자원 상태를 공유하므로 자신보다 자원이 부족한 서버에게 전달하는 일은 발생하지 않지만, 여전히 부하 분산의 범위가 1-hop으로 한정되었기 때문에 차량이 밀집될수록 성공률 저하가 불가피하다.

이에 반하여 클러스터 기반의 제안 알고리즘과 DDPG는 엣지 서버 간 클러스터 구조를 형성하고 CH 서버가 클러스터 내 가용 자원 상태를 관리하며 작업을 분산하므로 특정 서버에만 작업이 다량 요청되어도 CH 서버의 부하 분산 결정에 따라 클러스터 내에서 모든 서버의 자원이 고르게 활용된다. 즉, 작업을 수신한 서버와 실질적으로 연산을 처리하는 서버를 분리함으로써 국부적으로 차량이 밀집된 환경에서 부하를 효과적으로 고르게 분산하여 높은 오프로딩 성공률을 보였다. 또한 비교 알고리즘들은 트래픽 불균형이 심해질수록 성공률이 감소되는 것을 볼 수 있으나, 강화학습 기반 알고리즘은 트래픽 불균형 정도와 관계없이 항상 높은 정확도를 보임으로써 높은 이동성을 가지는 차량 환경에서도 안정적으로 좋은 성능을 보일 수 있음을 확인했다. 또한 TD3를 활용한 제안 방안이 DDPG보다 모든 시나리오에서 더 높은 성공률을 기록했다.

그림 7은 작업이 필요로 하는 연산 자원량의 변화에 따른 알고리즘별 성공률을 나타낸다. 필요 연산 자원량이 늘어날수록 작업 하나가 차지하는 서버 자원량이 증

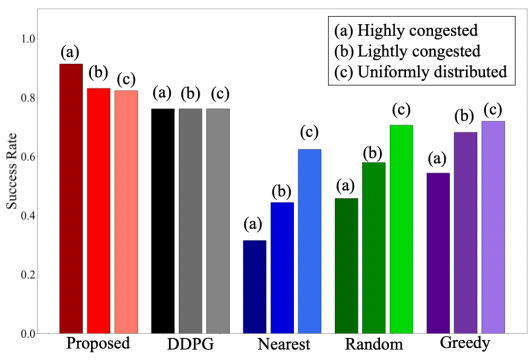


그림 6. 차량 분포 변화에 따른 성공률
Fig. 6. Success rate according to changes in vehicle distribution

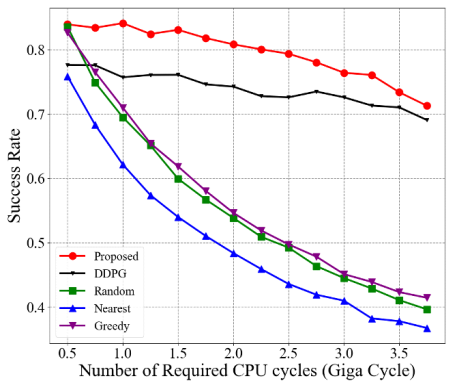


그림 7. 작업의 연산 자원량 변화에 따른 성공률
Fig. 7. Success rate according to variations in the computational resource requirements of tasks

가하기 때문에 모든 알고리즘의 성능에 직접적인 영향을 미친다. 특히, Nearest, Random, Greedy 알고리즘은 요구 연산량 증가에 따라 성공률이 급격하게 하락하는 것을 볼 수 있다. 작업이 요구하는 자원량이 커질수록 시스템 내 가용 자원을 효율적으로 활용해야 하나 부하 분산과 적절한 자원 할당이 이뤄지지 않기 때문이다. 따라서 Nearest가 가장 낮은 성능을 보이며 이웃 서버에게 전달하는 Random과 Greedy가 유사한 결과를 보인다. 반면 제안 알고리즘과 DDPG는 성능 저하 폭이 상대적으로 작은데, 그 이유는 작업의 요구 자원량이 증가하더라도 CH 서버가 클러스터 내 가용 자원이 높은 서버를 중점적으로 선택하며 부하를 분산하고 작업의 지연 시간 요구사항을 반영한 최적의 할당 자원량을 결정하기 때문이다. 특히, 제안 알고리즘은 작업의 요구 연산량이 증가하더라도 모든 알고리즘 중 항상 가장 높은 성공률을 보였다.

그림 8은 오프로딩 작업 데이터의 크기 변화에 따른 알고리즘별 성공률을 나타낸다. Nearest는 서버 간 통신이 발생하지 않으므로 작업의 데이터 크기 증가에 영향을 받지 않아 성능 변화가 없다. 다른 네 가지 알고리즘은 서버 간 데이터 전달이 발생하므로 데이터 크기 증가에 따라 성공률이 저하되는 경향을 보인다. 그 이유는 두 엣지 서버 중 데이터를 수신하여 연산을 담당할 서버는 곧 수신하게 될 작업을 위해 할당량만큼 자원을 예약해두고 다른 작업에 활용하지 못하게 되기 때문이다. 따라서 데이터 크기가 증가할수록 송신 시간이 길어지고, 예약된 자원을 활용하지 못하는 시간이 선형적으로 증가하므로 자원 활용률이 낮아지게 된다.

특히 Random은 자체 자원에 여력이 없는 경우에만 이웃 서버에게 전달하나, Greedy는 자체 처리 가능 여부와 관계없이 항상 가용 자원이 가장 큰 서버에게 전달하므로 통신이 더욱 자주 발생하게 되어 데이터 크기 증가에 따른 성능 저하 폭이 더욱 크다. 강화학습 기반 알고리즘들은 작업의 크기가 클 경우, 통신을 최소화하거나 대역폭이 가장 좋은 서버를 선택하는 등의 효율적인 선택이 가능하므로 성능 저하 폭이 작고 높은 성공률을 보이며, 특히 제안 방안은 실험 과정 동안 항상 DDPG 대비 높은 성공률을 보였다.

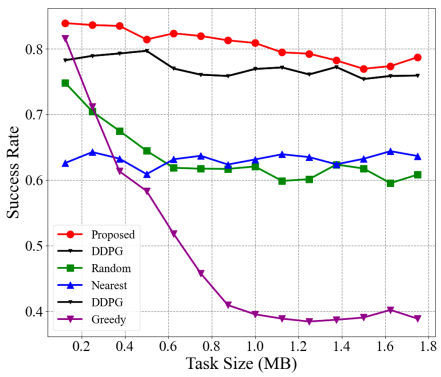


그림 8. 작업의 데이터 크기 변화에 따른 성공률
Fig. 8. Success rate according to changes in the data size of tasks

그림 9는 차량 수 증가에 따른 알고리즘별 성공률을 나타낸다. 차량 수가 약 30대일 때에는 알고리즘 간 성능 차이가 크지 않았으나, 차량 수가 증가함에 따라 비교 알고리즘들의 성공률은 급격히 하락했다. 특히 차량 수가 60대 이상이 되는 시점부터는 서버 간 부하를 분산하는 Greedy와 Random보다도 자체적으로만 작업을 처리하는 Nearest가 우수한 결과를 보임으로써, 동적 환경을 고려하지 않은 부하 분산 메커니즘이 오히려 자원 활용률을 낮추어 오프로딩 성공률을 낮출 수 있음을 확인했다. 강화학습을 적용한 방안들은 CH 서버가 작업의 프로파일, 대역폭, 서버들의 가용 자원량 등의 상태 정보를 종합적으로 고려한 최적의 부하 분산 결정 및 자원 할당 비율을 결정하므로 차량 수가 증가하더라도 가용 자원 활용률을 높여 성능 저하를 최소화하며 높은 확장성을 보였다. 특히 제안 방안은 DDPG 대비 차량 수 증가에도 항상 약 10%의 높은 정확도를 보였다.

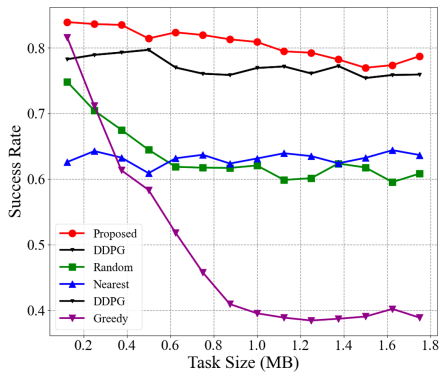


그림 9. 차량 수 증가에 따른 성공률
Fig. 9. Success rate based on the increase in the number of vehicles

Ⅵ. 결 론

본 논문은 차량 오프로딩 환경에서 엣지 서버 자원 활용률 최대화를 목적으로 엣지 서버 간 부하를 분산하는 강화학습 기반 알고리즘을 제안하였다. 엣지 서버 간 클러스터를 형성하여 클러스터 단위로 부하를 분산

하는 시스템 구조를 바탕으로, 클러스터 내 가용 자원의 편차를 최소화하는 서버 부하 분산 최적화 문제를 정의하였다. 동적인 환경에서도 안정적으로 최적해를 찾기 위하여 강화학습 기반 부하 분산 알고리즘을 제안하였는데 실험 결과, 제안 방안은 엣지 서버 간 부하 불균형 문제를 효과적으로 해결하였으며 결과적으로 오프로딩 성공률을 향상시키는 것을 확인하였다. 특히, 차량 수가 증가하거나 작업의 다양성이 큰 환경에서도 비교 알고리즘들보다 높은 보상 및 오프로딩 성공률을 보임으로써 확장성 측면에서 우수함을 확인하였다. 본 연구에서는 사전에 정의된 정적 클러스터 구조를 활용하였으나, 향후 연구에서는 엣지 서버의 가용 자원량을 바탕으로 동적으로 클러스터를 형성하고 부하를 분산하는 방안을 연구하고자 한다. 또한 최대 작업 요구 시간이 다양한 환경에서의 효율적인 자원 관리를 위하여 엣지 서버에 더불어 중앙 집중형 클라우드 서버를 함께 활용하는 오프로딩 시스템을 연구하고자 한다.

References

[1] Y. Ma, et al., "Artificial intelligence applications in the development of autonomous vehicles: A survey," *IEEE/CAA J. Automatica Sinica*, vol. 7, no. 2, pp. 315-329, Mar. 2020. (https://doi.org/10.1109/JAS.2020.1003021)

[2] E. Arnold, et al., "A survey on 3D object detection methods for autonomous driving applications," *IEEE Trans. Intell. Transport. Syst.*, vol. 20, no. 10, pp. 3782-3795, Oct. 2019. (https://doi.org/10.1109/TITS.2019.2892405)

[3] X. Zhao, et al., "Fusion of 3D LIDAR and camera data for object detection in autonomous vehicle applications," *IEEE Sensors J.*, vol. 20, no. 9, pp. 4901-4913, May 2020. (https://doi.org/10.1109/JSEN.2020.2966034)

[4] H. Khayyam, et al., "Artificial intelligence and internet of things for autonomous vehicles," *Nonlinear Approaches in Eng. Appl.: Automotive Appl. Eng. Problems*, pp. 39-68, 2020.

[5] T. L. Willke, P. Tientrakool and N. F. Maxemchuk, "A survey of inter-vehicle communication protocols and their applications," *IEEE Commun. Surv. & Tuts.*, vol. 11, no. 2, pp. 3-20, Second Quarter 2009. (https://doi.org/10.1109/SURV.2009.090202)

[6] A. B. De Souza, et al., "Computation offloading for vehicular environments: A survey," *IEEE Access*, vol. 8, pp. 198214-198243, 2020. (https://doi.org/10.1109/ACCESS.2020.3033828)

[7] C. Jiang, et al., "Toward computation offloading in edge computing: A survey," *IEEE Access*, vol. 7, pp. 131543-131558, 2019. (https://doi.org/10.1109/ACCESS.2019.2938660)

[8] M. Ahmed, et al., "A survey on vehicular task offloading: Classification, issues, and challenges," *J. King Saud University Comput. and Inf. Sci.*, vol. 34, no. 7, pp. 4135-4162, 2022. (https://doi.org/10.1016/j.jksuci.2022.05.016)

[9] Y. Liu, et al., "Deep reinforcement learning for offloading and resource allocation in vehicle edge computing and networks," *IEEE Trans. Veh. Technol.*, vol. 68, no. 11, pp. 11158-11168, Nov. 2019. (https://doi.org/10.1109/TVT.2019.2935450)

[10] X. Zhu, et al., "Multiagent deep reinforcement learning for vehicular computation offloading in IoT," *IEEE Internet of Things J.*, vol. 8, no. 12, pp. 9763-9773, Jun. 2021. (https://doi.org/10.1109/JIOT.2020.3040768)

[11] J. Zhang, et al., "Task offloading in vehicular edge computing networks: A load-balancing solution," *IEEE Trans. Veh. Technol.*, vol. 69, no. 2, pp. 2092-2104, Feb. 2020. (https://doi.org/10.1109/TVT.2019.2959410)

[12] P. Dai, et al., "Multi-armed bandit learning for computation-intensive services in MEC-empowered vehicular networks," *IEEE Trans. Veh. Technol.*, vol. 69, no. 7, pp. 7821-7834, Jul. 2020. (https://doi.org/10.1109/TVT.2020.2991641)

[13] L. Huang, S. Bi, and Y.-J. A. Zhang, "Deep reinforcement learning for online computation offloading in wireless powered mobile-edge

computing networks," *IEEE Trans. Mobile Comput.*, vol. 19, no. 11, pp. 2581-2593, Nov. 2020. (https://doi.org/10.1109/TMC.2019.2928811)

[14] J. Liu, et al., "RL/DRL meets vehicular task offloading using edge and vehicular cloudlet: A survey," *IEEE Internet of Things J.*, vol. 9, no. 11, pp. 8315-8338, Jun. 2022. (https://doi.org/10.1109/JIOT.2022.3155667)

[15] S. Bang and M. Lee, "Distributed server resource optimization in vehicular task offloading environments using deep reinforcement learning-based algorithm," in *KICS Winter Conf. 2024*, Pyeongchang, Korea, Feb. 2024.

[16] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*, MIT press, 2018.

[17] R. S. Sutton, D. Precup, and S. Singh, "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning," *Elsevier Artificial Intell.*, vol. 112, no. 1-2, pp. 181-211, 1999. (https://doi.org/10.1016/S0004-3702(99)00052-1)

[18] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Process. Mag.*, vol. 34, no. 6, pp. 26-38, Nov. 2017. (https://doi.org/10.1109/MSP.2017.2743240)

[19] S. Fujimoto, H. Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," *Int. Conf. Mach. Learn.* PMLR, pp. 1587-1596, 2018.

[20] V. Mnih, et al., "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013. (https://doi.org/10.48550/arXiv.1312.5602)

[21] Z. Ding, *Popular RL Algorithms* (2019), Retrieved Jan. 31, 2024, from https://github.com/quantumiracle/Popular-RL-Algorithms

방 수 정 (Soo-jeong Bang)

2021년 8월 : 이화여자대학교 컴퓨터공학과 졸업

2021년 9월~현재 : 이화여자대학교 컴퓨터공학과 석박통합과정

<관심분야> Deep reinforcement learning, vehicular task offloading, optimization, mobile edge computing

[ORCID:0000-0002-4201-0595]

이 미 정 (Mee-jeong Lee)

1987년 : 이화여자대학교 전자계산학과 졸업

1989년 : University of North Carolina at Chapel Hill 컴퓨터학과 석사

1994년 : University of North Carolina at Chapel Hill 컴퓨터학과 박사

1994년~현재 : 이화여자대학교 컴퓨터공학과 교수

<관심분야> FANET, MANET, VANET, Task Offloading, Wireless Mobile Networks, QoS routing, Internet Traffic Engineering

[ORCID:0000-0001-6968-8817]