https://doi.org/10.7840/kics.2025.50.1.185

클러스터 기반 이종 에지 컴퓨팅 시스템에서 작업 처리 속도 향상을 위한 작업 스케줄링 및 구현

이 상 철◆, 채 승 호°

Task Scheduling for Task Processing Time Improvement and Its Implementation in Cluster-Based Heterogeneous Edge Computing System

Sangcheol Lee◆, Seong Ho Chae°

요 약

본 논문에서는 서로 다른 크기의 자원과 컴퓨팅 능력을 가진 디바이스가 혼재되어 있는 이종 에지 컴퓨팅 시스템 환경에서 작업 처리 속도를 향상시키기 위한 스케줄링 알고리즘을 제안하고 성능을 검증한다. 제안하는 알고리즘은 스케줄러가 네트워크 에지 디바이스들을 활용해 동적으로 컴퓨터 클러스터로 구축함으로써 에지 디바이스의 컴퓨팅 능력을 향상시켜 성능 제약 문제를 해결한다. 또한, 제안된 알고리즘은 요청 작업과 에지 디바이스 상황을 종합적으로 고려하여 작업을 처리하게 함으로써 작업속도를 개선하고 복수의 작업 처리에 대한 시간 효율을 개선한다. 시뮬레이션을 통해 제안하는 알고리즘의 성능 개선을 검증하고, 실제 프로토타입 제작을 통해 알고리즘의 실현 가능성을 검증하였다.

Key Words : Heterogenous edge computing, Internet of things (IoT), Task scheduling, Computer clustering

ABSTRACT

In this paper, we propose the scheduling algorithm to improvement task processing speed in a heterogeneous edge computing system, where the devices have different resource capacities and computing capabilities. The proposed algorithm addresses performance limitations by enhancing the computing capabilities of edge devices, utilizing the scheduler to construct a computing cluster. In addition, the algorithm increases task processing speed by comprehensively considering the conditions of the edge devices and the task requests, which improves the time efficiency in processing multiple tasks. Our simulation results validate the superiority of the proposed algorithm and we validate its practical feasibility via a prototype.

※ 이 논문은 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원-학·석사연계ICT핵심인재양성의 지원(IITP-2025-RS-2022-00156326, 50%)과 정보통신기획평가원-지역지능화혁신인재양성사업의 지원을 받아 수행된 연구임(IITP-2025-RS-2020-II201741, 50%)

◆ First Author : Tech University of Korea Department of IT Semiconductor Convergence Engineering, shng9522@tukorea.ac.kr, 학생회원

° Corresponding Author : Tech University of Korea Department of Electronics Engineering, shchae@tukorea.ac.kr, 종신회원

논문번호 : 202408-188-C-RU, Received August 25, 2024; Revised September 21, 2024; Accepted September 23, 2024

Ⅰ. 서 론

사물인터넷(IoT: Internet of Things)은 4차 산업혁명 실현의 핵심기술로, 최근 빅데이터, 인공지능 등의 최신 기술들과 접목되어 스마트홈, 의료 등과 같은 생활의 모든 측면으로 보급되었다[1,2]. 이에 따라, IoT 디바이스(Device) 수의 급증으로 인해 발생하는 많은 양의 이종 데이터들을 효과적으로 처리하는 것이 매우 중요한 문제로 대두되었다. 기존의 IoT 시스템은 데이터를 클라우드 서버에 전송하여 처리하는 중앙 집중형 방식을 주로 사용하였다[3]. 하지만, 중앙 집중형 네트워크는 다수의 디바이스들이 제한된 대역폭을 공유하여 동작하므로 네트워크 병목 현상을 발생시키며, 이는 심각한 네트워크 지연을 발생시킨다. 뿐만 아니라, 중앙 집중형 IoT 시스템은 데이터를 중앙 서버로 전송함에 따라 개인 정보 보안에 치명적인 문제를 야기시킨다[4].

이러한 문제점을 보완하기 위한 한 가지 대안으로 최근 에지 컴퓨팅(Edge Computing) 기술이 많은 관심을 받고 있다. 에지 컴퓨팅은 네트워크 에지 디바이스들에서 컴퓨팅을 수행하는 분산형 컴퓨팅 기술로, 데이터 소스와 인접한 곳에서 컴퓨팅을 수행하게 함으로써 데이터에 대한 빠른 접근 및 처리를 용이하게 하고 보안 및 네트워크 지연 문제를 해결 가능하게 한다[5]. 에지 컴퓨팅의 에지 디바이스들은 제한된 전력 자원을 가짐으로써 이를 효율적으로 사용하기 위한 연구가 활발히 이루어졌다[6,7]. 에지 컴퓨팅 시스템에서 실시간 이미지 검출 모델 YOLO(You Only Look Once)를 사용하여 시스템 전력 효율성을 평가함으로써 처리량과 전력 효율성 간의 상관관계를 밝혔다[6]. 에지 컴퓨팅 시스템의 비 균일 트래픽 환경에서 전력 소비 최소화를 위한 sleep/active 제어 알고리즘을 제안하고, 임계값 기반 컴퓨팅 능력 제어를 통해 작업에 따른 전력 소비를 줄일 수 있음을 보였다[7].

에지 컴퓨팅은 일반적으로 저사양의 디바이스들로 구성되기 때문에 높은 복잡도를 가진 작업을 처리하기 어렵다[8]. 이를 극복하기 위한 한 가지 대안으로, 컴퓨팅 능력이 있는 다수의 디바이스를 묶어 하나의 고성능 컴퓨팅 디바이스로 동작 가능하게 하는 컴퓨터 클러스터링(Computer Clustering) 기술이 많은 관심을 받고 있다[9]. 최근의 IoT 시스템 및 에지 컴퓨팅 기술의 발전과 더불어, 다양한 성능 및 자원을 가진 이종 디바이스들이 증가하면서 이를 클러스터로 구축하여 관리하는 방법에 대한 중요성이 점차 증가하고 있다[10]. 이종 에지 컴퓨팅 시스템에서 Docker와 Kubernetese를 활용하여 클러스터를 구현하고 YOLO와 Faster R-CNN (Faster Regin-based convolutional Neural Network)와 같은 CNN 기반의 알고리즘 학습 수행이 가능함이 검증되었다[11]. 또한, Raspberry Pi를 활용하여 이종 에지 컴퓨팅 환경 및 클러스터를 구현하고 YOLO를 사용하여 클러스터의 적용 여부에 따른 성능 차이를 확인하는 연구가 수행되었다[12]. 하지만, 해당 연구들은 고정된 클러스터를 기반으로 수행된 연구로, 디바이스의 성능 및 상태에 따라 클러스터를 동적으로 변경하지 못하는 한계를 가지며, 이는 네트워크 상태와 디바이스의 가용성이 실시간으로 변화하는 IoT 환경에서 비효율적 자원의 사용과 작업 지연을 발생시킨다. 뿐만 아니라, 해당 연구들은 작업 부하 처리와 컴퓨팅 및 네트워크 자원 최적화를 위한 작업 처리 스케줄러(Scheduler)를 고려하지 않았다.

최근, 에지 컴퓨팅 환경에서 작업 관리 및 최적화를 위한 스케줄링에 관한 연구가 많은 관심을 받고 있다[13-15]. 클라우드 모바일 에지 컴퓨팅 환경에서 Lyapunov 최적화와 Water-filling을 활용한 작업 스케줄링(Task Scheduling) 알고리즘이 제안되었으며, 해당 스케줄링 알고리즘이 복수개의 작업에 대한 동적 스케줄링을 통해 작업 응답 시간을 단축시킬 수 있음을 보였다[13]. 참고문헌[14]은 IoT 에지 컴퓨팅 환경에서 작업 완료 예상 시간을 기반으로 어떤 디바이스에 작업을 할당할지 결정하는 CTA(Completion Time-based Task Assignment) 알고리즘과 작업별 중요도와 전체 작업의 마감 시간을 고려한 HWA(Hierarchical Weight Allocation) 알고리즘이 제안하고, 제안된 기법들이 요청 작업을 근처 IoT 디바이스들로 효율적으로 할당함으로써 데이터 전송 거리와 네트워크 사용량을 줄이고 작업 처리량을 증가시킬 수 있음을 보였다. 참고문헌[15]은 이종 컴퓨팅 환경에서 작업 완료 시간 최소화를 위한 작업 완료 이후 다음 작업으로 이어지는 시간을 고려하여 우선순위를 정하는 HEFT(Heterogeneous Earliest Finish Time) 알고리즘과 전체 작업 중 가장 긴 처리 시간이 예상되는 작업들을 하나의 프로세서에 집중시키는 방식인 CPOP(Critical Path on a Processor) 알고리즘을 제안하였으며, 해당 알고리즘들이 전체 작업 속도를 향상시킬 수 있음을 보였다. 하지만, 해당 연구들[13-15]은 에지 컴퓨팅 환경에서 스케줄링만을 고려하고 있으며 클러스터링을 통한 성능 향상은 고려하지 않았다.

따라서, 본 논문에서는 이종 에지 컴퓨팅 환경에서 컴퓨팅 능력을 향상시키고 작업 처리 시간을 단축시키기 위한 스케줄링 알고리즘 및 클러스터링을 제안하고 이를 실제 구현 검증한다. 구체적으로, 작업 스케줄링

알고리즘은 사용자의 다중 작업 요청에 대해서 디바이스별 성능, 작업 복잡도, 작업 시간 등을 종합적으로 고려하여 클러스터를 구축하고 작업을 할당하여 신속한 처리를 가능하게 한다. 또한 요청 작업 상황에 맞춰 동적으로 클러스터를 구축 가능하게 함으로써 컴퓨팅 능력을 향상시키고, 전체 작업 처리 시간을 크게 감소시킬 수 있다. 마지막으로 시뮬레이션을 통해 알고리즘의 성능을 검증하고 실제 프로토타입 제작을 통해 실현 가능성을 검증한다.

Ⅱ. 시스템 아키텍처 및 제안 스케줄링 알고리즘

2.1 시스템 아키텍처

그림 1은 본 논문에서 고려하는 시스템 아키텍처를 보여준다. 시스템은 크게 스케줄러와 에지 디바이스 두 부분으로 구분된다. 스케줄러는 에지 디바이스를 제어하여 동적으로 다중 클러스터를 구축하는 작업과 각 클러스터에 할당할 작업을 선택하는 역할을 수행한다.

스케줄러는 라우터 등과 같이 연산 처리 능력이 있는 임의의 디바이스로, 에지 디바이스들의 성능과 상태를 실시간으로 확인하고, 작업 테이블(Task Table)을 생성하여 사용자의 요청 작업을 실시간으로 확인한다. 해당 정보를 바탕으로 제한된 컴퓨팅 자원을 관리하고, 요청받은 작업을 2.2.2절에 후술 된 바와 같이 컴퓨팅 시간과 작업 복잡도를 기준으로 분류한 후 클러스터에 할당하여 작업 시간을 최소화하는 스케줄링 알고리즘을 수행한다. 에지 디바이스들은 서로 다른 컴퓨팅 능력을 가진 이종 디바이스들로 구성되어 있으며 각 디바이스들은 동일한 네트워크에 연결되어 데이터 및 자원을 공유할 수 있다. 에지 디바이스들은 스케줄러의 요청에 따라 동적으로 클러스터를 형성한다. 구축된 환경 조건, 에지 디바이스의 수 등 컴퓨팅 요구 사항에 따라 클러스터의 개수는 여러 개가 될 수 있다. 여기서, 하나의 클러스터는 하나의 마스터 노드(Master Node)와 여러 개의 작업 노드(Worker Node)로 이루어져 있다. 작업 노드의 수는 관리자의 설정과 스케줄링 알고리즘에 따라 결정된다. 마스터 노드는 스케줄링 알고리즘에 따라 클러스터 내의 작업 노드와 컴퓨팅 자원 및 데이터를 공유하여 할당받은 작업을 진행하게 된다.

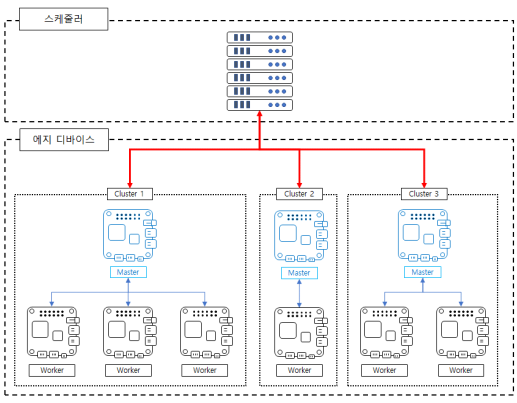

그림 1. 시스템 아키텍처
Fig. 1. System architecture

2.2 스케줄링 프로세스

그림 2는 제안하는 스케줄링의 전반적인 프로세스를 보여준다.

- Step 1: 스케줄러는 에지 디바이스들의 성능과 상태를 실시간으로 전송받아 디바이스 테이블을 구축한다.
- Step 2: 스케줄러는 작업 요청이 들어오면 요청된 작업을 확인하여 작업 테이블을 구축한다.
- Step 3: 스케줄러는 디바이스 테이블 정보와 작업 테이블 정보를 바탕으로 에지 디바이스를 클러스터로 구축한다.
- Step 4: 클러스터 구축이 완료되면, 각 클러스터는 클러스터 번호를 스케줄러에 전송하고 스케줄러는 디바이스 테이블을 업데이트한다.
- Step 5: 스케줄러는 작업의 순위에 따라 클러스터에 작업을 할당하고, 클러스터는 작업을 수행한다.

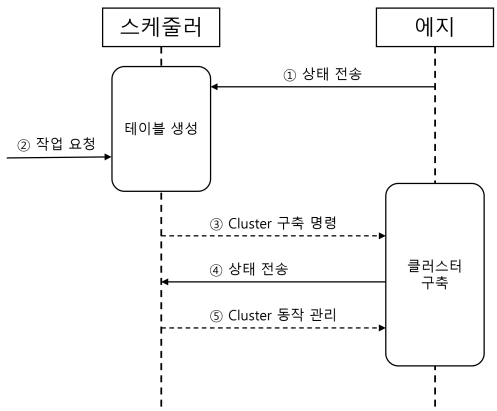

그림 2. 스케줄링 프로세스
Fig. 2. Scheduling process

2.2.1 디바이스 테이블 및 작업 테이블 생성

스케줄러는 에지 디바이스들의 성능과 상태를 전송받아 디바이스 테이블을 생성한다. 디바이스 테이블의 구성 요소는 다음과 같다.

표 1. 디바이스 테이블 구성 요소
Table 1. Device table components

ID	Freq	Class	Act	Time	Iter

여기서, ID는 에지 디바이스의 번호, Freq는 CPU가 사이클당 몇 회의 계산을 수행할 수 있는지 나타내는 클럭 속도, Class는 디바이스 유형을 나타낸다. Class는 Class 1과 Class 2로 나뉘며, Class 1과 2를 가지는 노드 수는 요청된 작업에 따라 스케줄링 알고리즘에 의해 임의로 변동된다. Class 1의 값은 마스터 노드 개수이며, 이에 따라 클러스터 개수가 결정된다. Class 2의 값은 작업 노드의 개수를 의미한다. Act는 디바이스의 작동 여부를 나타내며 해당 디바이스에 진행 중인 작업이 없으면 0, 작업이 있으면 클러스터의 구축 번호를 나타낸다. Time은 진행 중인 작업의 남은 시간, Iter는 남은 반복 횟수를 나타낸다.

스케줄러는 사용자의 작업을 확인하는 작업 테이블 또한 생성하며, 작업 테이블의 구성 요소는 다음과 같이 구성된다.

표 2. 작업 테이블 구성 요소
Table 2. Task table components

ID	Time	Iter	Complex

여기서, ID는 작업의 번호, Time은 작업이 1회 수행될 때 걸리는 시간, Iter은 해당 작업의 총 반복 횟수,

```
+-------+-------+------+-----+---------+-------+
| Field | Type  | Null | Key | Default | Extra |
+-------+-------+------+-----+---------+-------+
| ID    | int   | YES  |     | NULL    |       |
| Freq  | float | YES  |     | NULL    |       |
| Class | int   | YES  |     | NULL    |       |
| Act   | int   | YES  |     | NULL    |       |
| Time  | float | YES  |     | NULL    |       |
| Iter  | int   | YES  |     | NULL    |       |
+-------+-------+------+-----+---------+-------+
6 rows in set (0.00 sec)
```

그림 3. 디바이스 테이블 구축 예시
Fig. 3. Example of device table setup

```
+---------+-------+------+-----+---------+-------+
| Field   | Type  | Null | Key | Default | Extra |
+---------+-------+------+-----+---------+-------+
| ID      | int   | YES  |     | NULL    |       |
| Time    | int   | YES  |     | NULL    |       |
| Iter    | int   | YES  |     | NULL    |       |
| Complex | float | YES  |     | NULL    |       |
+---------+-------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

그림 4. 작업 테이블 구축 예시
Fig. 4. Example of task table setup

Complex는 수행하고자 하는 작업의 복잡도를 의미한다. 그림 3과 그림 4는 디바이스 테이블 및 작업 테이블의 구성 요소를 DBMS(Data Base Management System)를 통해 실제 구현한 예시를 보여준다.

2.2.2 클러스터 구축 및 작업 수행

작업 요청이 들어오면 스케줄러는 앞선 2.2.1절에서 생성한 디바이스 테이블 정보와 스케줄링 알고리즘을 바탕으로 에지 디바이스를 클러스터로 구축하는 과정을 수행한다. 스케줄러는 작업 테이블을 확인하여 요청된 작업을 확인하고, 각 작업의 복잡도는 작업당 필요로 하는 연산 사이클 수로 정의되며, 이를 기준으로 복잡도가 높은 순서에 맞춰 작업 우선순위를 정한다. 스케줄링 알고리즘은 최대 Class 1개의 클러스터를 구축할 수 있으며, 복수의 클러스터에 작업을 독립적으로 할당한다. 이를 통해 여러 작업을 병렬로 처리함으로써 작업 시간을 단축한다. 해당 작업이 완료되면 클러스터는 해체되고 다음 작업을 기다리는 유휴 상태가 된다.

2.3 스케줄링 알고리즘

본 장에서는 에지 디바이스들을 클러스터로 구축하고 사용자의 요청 작업들을 클러스터에 할당하는 스케줄링 알고리즘을 제안한다. 제안하는 스케줄링 알고리즘은 크게 두 부분으로, 마스터 노드 할당 알고리즘과 작업 노드 할당 알고리즘으로 구분된다. 스케줄링 알고리즘 수행 이전에, 관리자는 클러스터에 할당 가능한 작업 노드 수의 집합 $\{w_1, w_2, ..., w_i\}$ $(w_1 < w_2 < ... < w_i)$의 값을 설정한다. 스케줄링 알고리즘에 의해 각 클러스터는 $w_1, w_2, \dots, w_i$ 중 하나의 값을 갖게 되고 해당 값만큼의 작업 노드를 선택하여 클러스터로 구축한다. 여기서 w_i는 시스템의 최대 노드 개수 - 마스터 노드 개수를 초과할 수 없으며, 컴퓨팅 환경을 구성하고 있는 디바이스의 수와 초기 마스터 노드 설정값인 Class 1 값에 따라 $\{w_1, w_2, ..., w_i\}$ 값은 다르게 설정되어야 한다. 이때, 클러스터는 매 작업 처리마다 유휴 작업 노드의 개수와 컴퓨팅 능력, 작업의 복잡도를 기준으로 동적으로 구성된다. 처리하고자 하는 작업의 복잡도가 남아 있는 작업들의 평균 복잡도보다 클 경우 w_i대의 유휴 작업 노드를, 그렇지 않다면 w_{i-1}대의 유휴 작업 노드를 할당한다.

2.3.1 클러스터 구축 및 작업 수행

스케줄러는 사용자가 요청한 작업을 확인하여 작업의 복잡도를 기준으로 작업별 우선순위를 정하고, 유휴

Algorithm 1 마스터 노드 할당 알고리즘

```
Start
while 작업 요청 do
    작업 테이블 확인, 복잡도 순서대로 정렬
    디바이스 테이블 확인, 유휴 마스터 노드 확인
    if 마스터 노드 수 > 남은 요청 작업 then
        남은 요청작업 수와 마스터 노드 수가 같아질 때
        까지 마스터 노드를 작업 노드로 변경
    end if
    할당되지 않은 작업 중 복잡도가 높은 작업을
    유휴 마스터 노드 중 성능이 제일 좋은 노드에 할당
    Algorithm 2 실행
end while
End
```

그림 5. 마스터 노드 할당 알고리즘
Fig. 5. Master node allocation algorithm

마스터 노드를 확인하여 작업을 할당하는 알고리즘을 수행한다. 여기서, 유휴 마스터 노드는 디바이스 테이블의 Act 값이 0인 Class 1의 노드를 뜻한다. 만약 유휴 마스터 노드의 수가 작업 테이블의 남은 작업 수보다 많다면, 스케줄러는 마스터 노드의 Class를 1에서 2로 변경(즉, 유휴 마스터 노드를 작업 노드로 변경)하여 마스터 노드의 수와 남은 작업의 수를 동일하게 만든다. 이후, 마스터 노드의 CPU 성능 순서에 맞게 요청 작업의 복잡도가 높은 순서대로 작업을 할당한다. 또한, 유휴 마스터 노드가 하나만 남아 있다면 해당 마스터 노드에 작업을 할당한다. 모든 마스터 노드에 작업이 할당되면 스케줄러는 작업 노드 할당 알고리즘을 수행하여 클러스터에 구축될 작업 노드의 수를 정하는 과정을 진행한다. 그림 5는 앞서 기술한 마스터 노드 할당 알고리즘의 의사 코드(Pseudo Code)를 보여준다.

2.3.2 작업 노드 할당 알고리즘

스케줄러는 디바이스 테이블의 Class와 Act를 확인하여 유휴 작업 노드의 수를 확인한다. 작업 노드를 할당하는 순서는 다음과 같다.

- Step 1: 유휴 작업 노드의 수가 w_i대 이상 있고, 작업의 복잡도(C_i)보다 작업 테이블에 남아 있는 작업의 평균 복잡도(C_a)가 크다면 w_i개의 작업 노드를, 그렇지 않다면 w_{i-1}개의 작업 노드를 할당하여 클러스터를 구축한다.
- Step 2: 만약 유휴 작업 노드의 수가 w_{i-1}대 이상, w_i대 미만 있을 때, C_i보다 C_a가 더 크다면 w_{i-1}개의 작업 노드를, 그렇지 않다면 w_{i-2}대의 작업 노드를 할당하여 클러스터를 구축한다.
- Step 3: w_i를 w_{i-1}로 감소시킨 후 Step 1, 2의 과정을 반복하여 모든 클러스터가 작업 노드를 할당받거나 유휴 작업 노드가 없을 때까지 반복한다. 만약 유휴 작업 노드가 없다면 해당 클러스터는 작업 노드 없이 마스터 노드만으로 클러스터가 구축된다.

Algorithm 2 작업 노드 할당 알고리즘

```
Start
작업 노드 할당 요청
while 유휴 작업 노드가 있음 do
    if w_i 대 이상의 작업 노드가 있음 then
        현재 작업의 복잡도 C_c와 남은 작업들의 복잡도 평균 C_a를 확인
        if C_a < C_c then
            유휴 작업노드 중 성능이 좋은 w_i 개의 작업 노드 할당
        else
            유휴 작업노드 중 성능이 좋은 w_{i-1} 개의 작업 노드 할당
        end if
    else
        w_i ← w_{i-1}
    end if
    if 모든 클러스터에 작업노드가 할당됨 then
        break
    end if
end while
End
```

그림 6. 작업 노드 할당 알고리즘
Fig. 6. Worker node allocation algorithm

작업 노드 할당이 완료되어 클러스터가 구축되면, 해당 클러스터의 통합 CPU 성능을 기준으로 해당 클러스터에 할당된 작업의 수행 시간을 예측한다. 만약 작업 노드의 수가 적게 할당되더라도 작업을 처리 완료하는 데 소요되는 시간이 길어질 뿐, 어떠한 작업 노드 수가 할당되더라도 해당 작업은 완료할 수 있다. 그림 6은 앞서 기술한 작업 노드 할당 알고리즘의 의사 코드를 보여준다.

Ⅲ. 구현 및 성능 검증

본 장에서는 제안하는 스케줄링 알고리즘(마스터 노드 할당 알고리즘과 작업 노드 할당 알고리즘)에 따른 에지 디바이스의 컴퓨팅 성능 향상을 시뮬레이션으로 검증하고, 참고문헌[13]과 [14]의 스케줄링 기법을 비교군으로 하여 제안하는 스케줄링 알고리즘 성능을 비교하였다. 또한 라즈베리파이(Rapsberry Pi), 네트워크 스위치, 랩톱(Laptop)를 활용하여 실제 프로토타입 제작을 통해 실현 가능성을 검증한다. 구체적으로, 복수의 에지 디바이스와 사용자의 작업 요청이 있는 상황에서 스케줄링 알고리즘의 작업 시간 효율 향상을 확인한다. 또한 라즈베리파이와 랩톱을 동일한 네트워크 단에 연결하여 클러스터 기반 에지 컴퓨팅 프로토타입을 구성하고 스케줄링 알고리즘을 적용해 복수의 작업 요청에 대한 작업 시간 효율 향상과 구현 가능성을 검증한다.

3.1 스케줄링 알고리즘 시뮬레이션 성능 검증

본 장에서 스케줄링 알고리즘 검증을 위한 시뮬레이

션 환경은 다음과 같이 설정하였다. 서로 다른 컴퓨팅 성능을 가지는 에지 디바이스 10대에 대해 무작위 복잡도를 갖는 100개의 작업이 요청된 상황을 고려한다. 각 작업의 복잡도는 작업당 요구하는 CPU 사이클 수를 기준으로 평균 350 Mcycles, 표준편차 210 Mcycles의 정규분포를 따라 임의의 값으로 설정하였다[13]. 또한, 이종 환경을 고려하기 위해 10대의 디바이스 CPU 사이클 수를 500 Mcycles에서 1,500 Mcycles 사이의 임의의 값을 가지도록 하였다. 본 성능 검증에서 사용된 CPU를 기준으로, 1 cycle 당 걸리는 측정 시간은 평균 4.31×10^{-7}ms이다.

스케줄링 알고리즘의 우수성을 검증하고, 스케줄링 알고리즘에서 마스터 노드와 작업 노드의 초기 비율 설정에 따른 성능 차이를 살펴보기 위해 다음 4가지 경우에 대해 시뮬레이션을 진행하였다.

- (Case 1) 스케줄링 알고리즘 미적용, 1개의 마스터 노드와 9개의 작업 노드가 1개의 클러스터를 형성하여 100개의 작업을 순차적으로 처리
- (Case 2) 마스터 노드와 작업 노드 수의 초기 설정 비율을 3:7로 설정하고 스케줄링 알고리즘을 기반으로 작업 처리
- (Case 3) 마스터 노드와 작업 노드 수의 초기 설정 비율을 5:5로 설정하고 스케줄링 알고리즘을 기반으로 작업 처리

모든 실험은 총 10회 반복했으며 결과를 평균화하였다.

그림 7은 위의 3가지 경우에 대한 작업 처리 소요 시간을 비교한 결과를 보여준다. Case 1의 스케줄링 미적용 시 소요되는 작업 처리 시간을 100%로 할 때, 스케줄링을 적용한 Case 2, 3의 경우 각각 약 81%, 78%로 작업 처리 소요 시간을 개선할 수 있음을 보여준다.

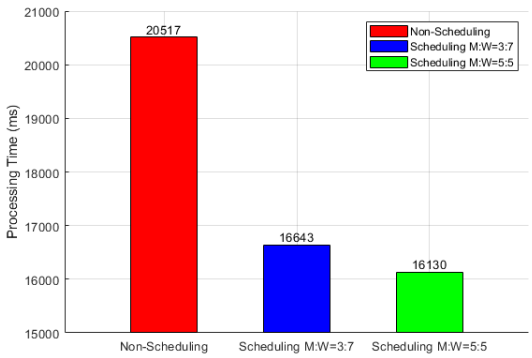

그림 7. 작업 처리 시간 비교 (낮은 복잡도)
Fig. 7. Comparison of processing time (Low Complexity)

또한, 그림 8은 마스터 노드와 작업 노드의 비율의 초기 설정값에 따라 동일한 스케줄링 알고리즘을 적용하더라도 성능 차이가 발생할 수 있음을 보여준다. 마스터 노드 수의 초기값은 클러스터 수의 초기 설정 개수를 의미하므로, 이는 시스템 환경 및 작업의 복잡도 등의 다양한 요소에 따라 마스터 노드와 작업 노드를 최적의 비율로 설정함으로써 보다 더 성능을 개선할 수 있음을 보여준다. 그림 8은 요청 작업들이 높은 복잡도를 가지도록 작업당 요구하는 CPU 평균 사이클 수를 3배로 증가시킨 후 작업 처리 소요 시간을 비교한 결과를 보여준다. Case 1의 작업 처리 시간을 100%로 할 때 Case 2와 Case 3의 경우 각각 83%, 86%로 그림 8과는 다른 결과가 나온 것을 확인할 수 있다. 일반적으로, 높은 복잡도를 갖는 작업이 많은 상황에서는 마스터 노드의 비율을 적게, 낮은 복잡도의 작업이 많은 상황에서는 마스터 노드의 비율을 높게 설정하는 것이 전체 작업 소요 시간을 줄일 수 있는 경향성을 보여준다.

그림 9는 참고문헌 [13]과 [14]의 알고리즘을 비교군으로 하여 스케줄링 알고리즘의 성능을 비교한 그래프

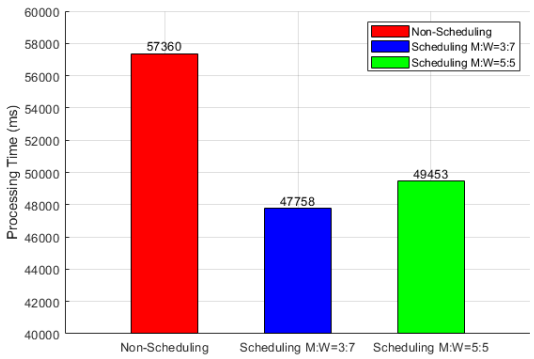

그림 8. 작업 처리 시간 비교 (높은 복잡도)
Fig. 8. Comparison of processing time (High Complexity)

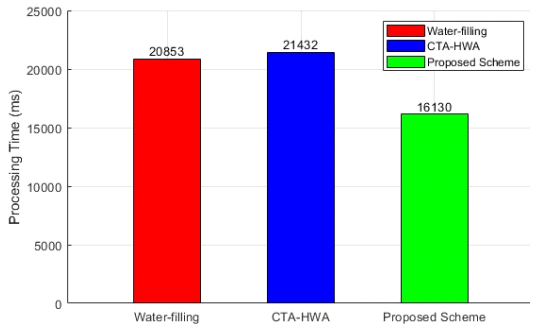

그림 9. 스케줄링 기법별 처리 시간 비교
Fig. 9. Comparison of processing time by scheduling scheme

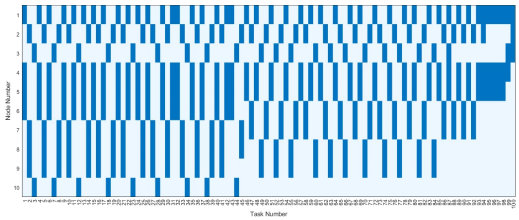

그림 10. 작업별 실행 노드
Fig. 10. Node selection for each task

이다. 참고문헌[13]은 Water-filling 개념을 활용해 에지 디바이스의 용량에 맞게 작업을 분배하고, 남은 작업은 다른 노드 혹은 클라우드 자원을 활용하는 방식으로 작업을 할당하는 방안을 사용하였다. 참고문헌[14]에서 제안하는 알고리즘은 CTA-HWA 알고리즘으로 작업의 예상 완료 시간을 기준으로 우선순위를 정하여 작업을 할당하는 방안을 고려하였다. 3가지 스케줄링 중 작업 처리 시간이 가장 길었던 CTA-HWA을 기준으로, 각각 97%, 75%의 시간을 소요하며 제안하는 알고리즘의 성능이 우수한 것을 확인하였다.

그림 10은 Case 2의 상황에서, 작업별 실행 클러스터와 클러스터 노드들의 히트맵(heatmap)을 그린 그림이며, 그림 11은 시간에 따라 처리 중인 작업들을 표시한 그림이다. 여기서, 총 100개의 작업들에 대해 작업 복잡도가 큰 것부터 작업 번호가 오름차순으로 정렬되어 있다. 즉, 1번이 가장 큰 작업 복잡도를 가지며, 100번이 가장 작은 복잡도를 가진다. 그림 10을 통해 각 작업들이 어떤 노드들에서 처리되고 있는지를 확인할 수 있으며, 그림 11을 통해 복수개의 클러스터가 복수개의 작업들을 연속 및 동시 병렬적으로 처리함으로써 작업 처리 시간 효율 이득을 얻을 수 있음을 보여주고 있다.

3.2 프로토타입 제작 및 성능 검증

그림 12는 다른 사양을 가진 라즈베리파이들과 네트워크 스위치를 사용하여 실제 프로토타입을 구현한 모습을 보여준다. 스케줄러는 Linux OS가 탑재된 Pentium N3710 사양의 랩톱을 사용하였으며, DBMS로 MySQL을 사용하여 디바이스 테이블과 작업 테이블을 생성하였다. 에지 디바이스는 성능이 다른 라즈베리파이 7대에 Linux OS를 탑재하여 사용하였다. 표 3은 각각의 디바이스별 성능을 나타낸다. 네트워크 스위치를 사용하여 스케줄러와 에지 디바이스들을 하나의 네트워크 단에 연결하여 클러스터 구축을 위한 환경을 구성하였다. 작업 생성 및 할당과 클러스터 구축을 위해

표 3. 디바이스 성능 표
Table 3. Device performance table

명칭	CPU	RAM	Storage
Scheduler	4 Core 2.56 GHz	8GB	500GB
Node 1	4 Core 2.00 GHz	8GB	64GB
Node 2	4 Core 1.50 GHz	8GB	64GB
Node 3		8GB	64GB
Node 4		8GB	64GB
Node 5		4GB	32GB
Node 6		2GB	32GB
Node 7	4 Core 1.00 GHz	1GB	16GB

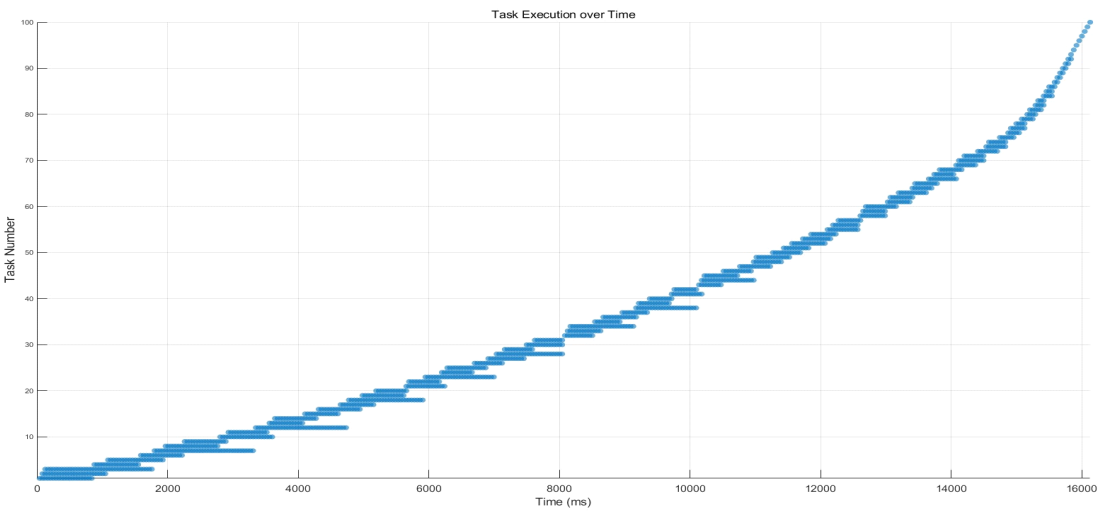

그림 11. 시간당 실행 작업
Fig. 11. Task execution over time

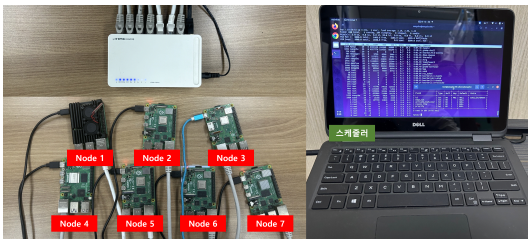

그림 12. 하드웨어 환경 구성
Fig. 12. Hardware Configuration

Docker와 Kubernetes 소프트웨어를 사용하였다. Docker는 응용 프로그램과 환경을 하나의 컨테이너로 포함하여 배포와 실행을 용이하게 해주는 프로그램으로, 이종 컴퓨팅 환경에서도 일관된 작업을 실행할 수 있게 한다. Kubernetes는 Docker에 의해 컨테이너로 포함된 응용 프로그램을 클러스터에 배포, 및 관리를 도와주는 오케스트레이션 목적으로 사용하였다. Kubernetes를 사용하여 네트워크 스위치에 연결된 작업 노드들의 이름과 IP를 마스터 노드의 작업 노드 목록에 추가하였고, Docker를 통해 수행할 작업의 컨테이너를 생성하여 작업을 배포하였다.

구현한 하드웨어 프로토타입을 활용하여 스케줄링 알고리즘의 성능을 검증하였다. 확인을 위해 다음 3가지 경우에 대한 검증을 진행하였다.

- (Case 1) 스케줄링 알고리즘 미적용, 1개의 마스터 노드와 6개의 작업 노드가 1개의 클러스터를 형성
- (Case 2) 마스터 노드와 작업 노드 수의 초기 설정 비율을 2:5로 설정하고 스케줄링 알고리즘을 기반으로 작업 처리
- (Case 3) 마스터 노드와 작업 노드 수의 초기 설정 비율을 3:4로 설정하고 스케줄링 알고리즘을 기반으로 작업 처리

클러스터 할당 가능 작업 수의 초기값은 $\{w_1, w_2\} = \{1,2\}$로, Case 2, 3 모두 동일하게 설정하였다. 작업은 YOLO object detection 모델을 사용하여 이미지 50장에 대한 검출 작업을 진행하였다. YOLO는 이미지를 입력 데이터로 사용하여 학습을 진행하는 CNN 기반 객체 검출 모델로, 이미지의 크기 조정과 같은 전처리 과정과 인공신경망이 하나의 모델로 통합되어 동작하여 작업 시간 대비 정확도가 높은 검출 모델이다. 객체 검출에 사용된 데이터는 COCO(Common Objects in Context) 사진 데이터 세트로 무작위로 선정한 사진 50장을 사용하여 작업을 진행하였다.

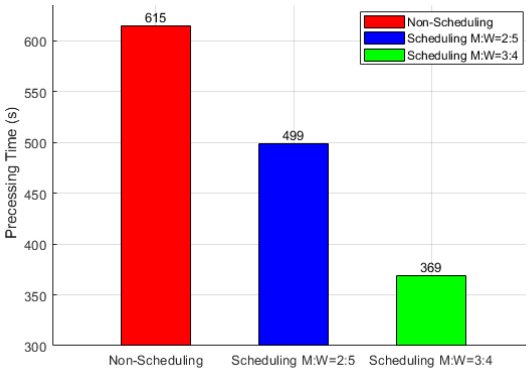

그림 13. 프로토타입을 통한 제안된 알고리즘 성능 검증
Fig. 13. Performance evaluation of proposed algorithm via prototyping

그림 13은 프로토타입 환경에서 스케줄링 알고리즘에 대한 성능을 확인한 그래프로, 위의 3가지 경우에 대한 작업 처리 소요 시간을 비교한 결과를 보여준다. Case 1의 작업 처리 시간을 100%로 할 때, Case 2, 3의 경우 각각 약 81%, 60%로 스케줄링 알고리즘에 의해 작업 시간이 개선된 것을 확인할 수 있었다.

Ⅳ. 결 론

본 논문에서는 클러스터링 기술이 적용된 에지 컴퓨팅 시스템에서 다량의 작업이 요청되었을 때, 작업 처리 속도를 향상시키기 위한 스케줄링 알고리즘을 제안하였다. 제안된 알고리즘은 에지 디바이스를 클러스터로 구축하여 컴퓨팅 능력을 향상 함으로써 성능 제약 문제를 해결하였다. 또한 요청된 작업을 복잡도를 기준으로 순위를 설정하여 클러스터의 성능에 맞춰 할당함으로써 전체 작업 소요 시간 단축을 확인하였다. 제안된 알고리즘은 시뮬레이션을 통해 적용하지 않은 시스템 대비 약 78%의 시간만 사용하여 작업을 완료할 수 있음을 확인하였고, 하드웨어 프로토타입을 구현함으로써 스케줄링 알고리즘의 성능 확인과 하드웨어 구현 가능성을 확인하였다.

References

[1] T. Kim and S. H. Chae, "A novel random access framework for uplink cellular IoT: Non-orthogonal preambles and multi-antennas," *IEEE Commun. Lett.*, vol. 24, no. 4, pp. 748-752, Apr. 2020. (https://doi.org/10.1109/LCOMM.2020.297018

2)

[2] M. Bakator and D. Radosav, “Deep learning and medical diagnosis: A review of literature,” *Multimodal Technol. Interact*, vol. 2, no. 3, 2018. (https://doi.org/10.3390/mti2030047)

[3] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, “Internet of things: A survey on enabling technologies, protocols, and applications,” *IEEE Commun. Surv. & Tuts.*, vol. 17, no. 4, pp. 2347-2376, 2015. (https://doi.org/10.1109/COMST.2015.2444095)

[4] H. Takabi, J. Joshi, and G. Ahn, “Security and privacy challenges in cloud computing environments,” *IEEE Security & Privacy*, vol. 8, no. 6, pp. 24-31, Nov. 2010. (https://doi.org/10.1109/MSP.2010.186)

[5] D. Ko, S. H. Chae, and W. Choi, “MDS coded task offloading in stochastic wireless edge computing networks,” *IEEE Trans. Wireless Commun.*, vol. 21, no. 3, pp. 2107-2121, Mar. 2022. (https://doi.org/10.1109/TWC.2021.3109448)

[6] K. Rungsuptaweekoon, V. Visoottiviseth, and R. Takano, “Evaluating the power efficiency of deep learning inference on embedded GPU systems,” in *Proc. Int. Conf. Inf. Tech.*, pp. 1-5, Nakhonpathom, Thailand, 2017. (https://doi.org/10.1109/INCIT.2017.8257866)

[7] A. A. Amer, I. E. Talkhan, and T. Ismail, “Optimal power consumption on distributed edge services under non-uniform traffic with dual threshold sleep/active control,” in *Proc. Novel Int. and Leading Emerging Sci. Conf.*, pp. 63-66, Giza, Egypt, 2021. (https://doi.org/10.1109/NILES53778.2021.9600496.)

[8] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, “Edge computing: Vision and challenges,” *IEEE Int. of Things J.*, vol. 3, no. 5, pp. 637-646, Oct. 2016. (https://doi.org/10.1109/JIOT.2016.2579198)

[9] M. Baker, R. Buyya, and D. Hyde, “Cluster computing: A high-performance contender,” *IEEE Comput.*, vol. 32, no. 7, Jul. 1999. (https://doi.org/10.48550/arXiv.cs/0009020)

[10] N. Sadashiv and S. M. D. Kumar, “Cluster, grid and cloud computing: A detailed comparison,” in *Proc. Int. Conf. on Computer Sci. & Education*, Singapore, 2011. (https://doi.org/10.1109/ICCSE.2011.6028683)

[11] Y.-W. Chan, H. Fathoni, H.-Y. Yen, and C. -T. Yang, “Implementation of a cluster-based heterogeneous edge computing system for resource monitoring and performance evaluation,” *IEEE Access*, vol. 10, pp. 38458-38471, 2022. (https://doi.org/10.1109/ACCESS.2022.3166154)

[12] S. Lee, C. Lee, and S. H. Chae, “Performance enhancement of heterogeneous edge computing via computer clustering,” in *Proc. Symp. KICS*, pp. 701-702, Jeju Island, Korea, Jun. 2023.

[13] X. Ma, A. Zhou, S. Zhang, Q. Li, A. X. Liu, and S. Wang, “Dynamic task scheduling in cloud-assisted mobile edge computing,” *IEEE Trans. Mobile Computing*, vol. 22, no. 4, pp. 2116-2130, Apr. 2023. (https://doi.org/10.1109/TMC.2021.3115262)

[14] Y. Kim, C. Song, H. Han, H. Jung, and S. Kang, “Collaborative task scheduling for IoT-assisted edge computing,” *IEEE Access*, vol. 8, pp. 216593-216606, 2020. (https://doi.org/10.1109/ACCESS.2020.3041872)

[15] H. Topcuoglu, S. Hariri, and M.-Y. Wu, “Performance-effective and low-complexity task scheduling for heterogeneous computing,” *IEEE Trans. Parallel and Distrib. Syst.*, vol. 13, no. 3, pp. 260-274, Mar. 2022. (https://doi.org/110.1109/71.993206)

이 상 철 (Sangcheol Lee)

2022년 2월 : 한국공학대학교 전자공학부 학사
2024년 2월 : 한국공학대학교 IT반도체융합공학과 석사
2024년 3월~현재 : 한국공학대학교 IT반도체융합공학과 박사과정
<관심분야> 6G 통신 시스템, 에지 컴퓨팅, 인공지능 무선통신 응용, 사물인터넷

채 승 호 (Seong Ho Chae)

2010년 2월 : 성균관대학교 정보통신공학부 학사
2012년 2월 : 한국과학기술원 전기 및 전자공학과 석사
2016년 2월 : 한국과학기술원 전기 및 전자공학부 박사
2016년 3월~2016년 9월 : 한국과학기술원 정보전자연구소 연수연구원
2016년 10월~2018년 2월 : 국방과학연구소 선임연구원
2018년 3월~2023년 8월 : 한국공학대학교 조교수
2023년 9월~현재 : 한국공학대학교 부교수
<관심분야> 6G 통신 시스템, 저궤도 위성통신, 에지 컴퓨팅, 인공지능 무선통신 응용
[ORCID:0000-0001-9092-5039]