논문 2022-17-12

머신러닝 기반 멀티모달 센싱 IoT 플랫폼 리소스 관리 지원

(Machine learning-based Multi-modal Sensing IoT Platform Resource Management)

이 성 찬, 성 낙 명, 이 석 준, 윤 재 석*
(Seongchan Lee, Nakmyoung Sung, Seokjun Lee, Jaeseok Jun)

Abstract : In this paper, we propose a machine learning-based method for supporting resource management of IoT software platforms in a multi-modal sensing scenario. We assume that an IoT device installed with a oneM2M-compatible software platform is connected with various sensors such as PIR, sound, dust, ambient light, ultrasonic, accelerometer, through different embedded system interfaces such as general purpose input output (GPIO), I2C, SPI, USB. Based on a collected dataset including CPU usage and user-defined priority, a machine learning model is trained to estimate the level of nice value required to adjust according to the resource usage patterns. The proposed method is validated by comparing with a rule-based control strategy, showing its practical capability in a multi-modal sensing scenario of IoT devices.

Keywords : Machine learning, Multi-modal sensing, IoT platform, Resource management, oneM2M standard

Ⅰ. 서 론

사물인터넷 (IoT, The Internet of Things)은 1999년 처음 소개된 이후 여러 유무선 통신기술 발달로 인해 사물들이 서로 연결되어 정보를 공유하고 협업하여 사용자에게 더욱 편리한 삶을 제공해주는 제4차 산업혁명의 핵심으로 대두하였다.

사물인터넷 시스템에서 중요한 요소 중 하나는 주변 환경에서 정보를 수집하는 센싱 시나리오를 문제없이 성공적으로 진행하는 것이다. 특히 사물인터넷의 근간을 이루는 임베디드 시스템의 센싱 시나리오 구현을 위해, 다양한 센서들이 서로 다른 형태의 데이터를 수집하는 경우, 이를 공통된 리소스 구조와 표준 인터페이스를 통해 서버에 전송하는 방식으로 상호연동성과 글로벌 접근성을 제공하는 사물인터넷 미들웨어를 사용하는 것이 더욱 중요해졌다.

이를 위해 '세계 어디에서나 접근할 수 있고 서로 협업이 가능한 사물인터넷 서비스 개발을 위한 규약'을 목표로 하여 2012년 oneM2M 표준이 등장하였으며, 2015년의 첫 번째 표준인 릴리즈 1을 선두로 여러 릴리즈와 더불어 많은 연구소와 회사에서 사물인터넷 서비스 플랫폼을 공개하고 서비스 개발에 활용하고 있다 [1].

서비스 계층 표준인 oneM2M을 기반으로 개발된 소프트웨어 플랫폼들은 임베디드 시스템에 설치하여 데이터 수집을 하기에 매우 이상적이다. 하지만 상황에 따라 시스템이 활용할 수 있는 리소스가 제한적인 경우 성공적인 진행을 위해 상황에 맞는 시스템 리소스 관리가 필요하다. 예로서 다수의 멀티모달 센서로부터 센싱이 이뤄지는 시나리오에서는 센싱 데이터의 획득, 처리, 전송 과정에서, 센서마다 특성과 시나리오에 따라 CPU 사용량 등 리소스 사용 패턴이 다르므로, 이를 고려한 리소스 할당 관리가 이루어진다면 센싱 시나리오에서 발생할 수 있는 문제점 (원하는 주기에 측정과 전송이 이루어지지 않는 등)을 방지하여 시스템 안정성을 향상시킬 수 있다.

본 논문에서는 oneM2M 기반 사물인터넷 플랫폼을 설치한 기기를 가정하고, 센서마다 측정 및 전송에 필요한 리소스 사용 패턴을 고려해 필요한 센싱 프로세스의 우선순위를 조절하는 방식으로 시스템 안정성과 센싱 시나리오의 실행 정확성을 높일 수 있는 기법을 제안한다. 이를 위해 멀티모달 센싱 환경에서 CPU, Memory 등 리소스 상태를 수집하여 센서마다 리소스 사용 패턴을 학습하는 머신러닝 모델을 생성하고, 모델의 패턴 분류 결과에 따라 해당 센서의 시나리오 우선순위를 고려한 nice value 조절을 통해 센싱 프로세스의 우선순위를 제어한다. 마지막으로 단순 규칙 기반 프로세스 제어 기법과 비교를 통해 본 논문에서 제안하는 머신러닝 기반 리소스 관리 기법의 동작을 검증한다.

본 논문의 구성은 다음과 같다. 2장에선 기존 자가적응 프레임워크와 우선순위 제어 관련 연구를 기술하고, 3장에서 제안하는 머신러닝 기반 리소스 활용 관리 보조 기법에 관해 설명한다. 4장에서는 실험을 통해 제안한 기법의 실행 가능성과 결과를 검증하고 5장에서는 한계점 및 향후 과제에 관해 기술한 뒤, 마지막으로 6장에서 결론을 제시한다.

*Corresponding Author (yun@sch.ac.kr)
Received: Jan. 31, 2022, Revised: Mar. 7, 2022, Accepted: Mar. 17, 2022.
S.C. Lee: Soonchunhyang University (B.S. Student)
N.M. Sung: KETI (Team Leader, Senior Researcher)
S.J. Lee: KETI (Senior Researcher)
J.S. Yun: Soonchunhyang University (Assistant Professor)
※ 본 논문은 2021년도 정부 (과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원을 받아 수행된 연구임 (No.2018-0-01456, (총괄+세부) 지능 기반 초소형 disposable IoT 동적 자율 구성 및 실행 인프라 기술).
※ 본 논문은 2020년도 정부 (교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임 (No.2020R1I1A3A04037409).

ⓒ IEMEK J. Embed. Sys. Appl. 2022 Apr. 17(2) 93-100
ISSN : 1975-5066
http://dx.doi.org/10.14372/IEMEK.2022.17.2.93

Ⅱ. 관련 연구

1. 리소스 제한 소프트웨어 플랫폼

임베디드 시스템, 사물인터넷 시스템, 그리고 스마트폰과 같이 시스템 리소스가 제한된 환경 (resource-constrained)에서 사용자가 실행하고자 하는 기능을 원활히 수행하기 위한 연구가 활발히 이루어지고 있다. 예를 들어, 제한적인 리소스를 갖는 임베디드 시스템의 경우, Hong은 경량 딥러닝을 위한 합성곱 신경망 기본 연산 인터페이스의 구성과 동작 구조를 제시하며, PC나 Server 플랫폼보다 제한적인 시스템 리소스를 갖는 임베디드 시스템에서 시스템의 특징을 파악함으로써 임베디드 시스템에 최적화된 기술의 필요성을 제시하였다 [2].

2. 자가 적응형 프레임워크

사물인터넷 시스템에서 멀티모달 센서 사용 환경을 구성하는 과정에서 시스템 자체적으로 리소스 활용 구조를 최적화하는 연구도 이루어졌다. Sung과 Yun은 사물인터넷 서비스 계층 표준인 oneM2M를 기반으로 한 소프트웨어 플랫폼들이 리소스와 연결성이 확보된 환경에서는 이상적으로 동작하지만, 제한된 환경에서 안정적인 동작을 위해 자가 적응형 모듈의 필요성을 보였다. 이를 위해 제한된 환경에서 사물인터넷 시스템 간 상호연동성이 보장되고, 연결된 리소스 조건을 고려하여 메모리, 네트워크, 배터리, 데이터를 감지하는 자가적응 모듈을 추가하였다. 결과적으로, 사물인터넷 기기의 보고 주기와 보고 여부를 제어하는 방식으로 스스로 동작을 제어할 수 있는 자가적응 소프트웨어 플랫폼을 제안하였다 [3]. Cheng 등이 제안한 Rainbow는 외부적으로 적응을 조정하는 구조 기반 메커니즘을 가지고, 각기 다른 시스템의 구조와 메커니즘을 다루며 모니터링, 모델링, 문제 감지 메커니즘을 통해 다른 시스템에서 재사용 가능한 모듈의 문제 감지 및 해결로 최선의 적응 방법을 구한다 [4]. Bresciani 등은 자가 적응형 프레임워크인 Tropos에서 'Belief-Desire-Intention' 에이전트 모델에서 에이전트의 실행 계획, 능력, 목표와 그들의 의존성에 대한 명제를 작성, 이를 바탕으로 시스템의 구현을 통해 자가적응 시스템 개발을 위한 프레임워크를 정의하였다 [5]. Morandini 등은 작은 단위의 행동은 표현할 수 있으나, 큰 규모와 다양한 에이전트의 관계에서 설계와 예측이 어려운 Tropos의 문제점에 대해 관계의 도입을 통한 우선순위의 형태로 표현하는 방법을 연구하였다 [6].

3. 태스크 우선순위 제어

Jang 등은 Linux 시스템의 Completely Fair Scheduler (CFS) 애플리케이션의 특징과 태스크 우선순위의 상관관계를 실험하였다 [7], 서로 다른 특징을 가진 애플리케이션들을 동시에 실행하는 환경에서 태스크 우선순위 정밀도를 제어하는 기법을 제안하고, 이를 Linux 환경에서 구현하여 CFS 에서 가중치 값의 범위를 축소 혹은 확장했을 때, 일부 프로세스에서 더 좋은 성능을 보임을 확인하였다. Jain 등은 CFS 에서 머신러닝을 활용하여 시스템에서 프로세스별 Turn-around-time (TaT)을 예측하고 nice value를 이용해 Linux 스케줄러에 적절한 time slice를 할당하여 TaT를 감소시킬 수 있는 프로세스 스케줄링 기술을 제안한다 [8]. Wong 등은 CFS 가 기존의 알고리즘에 비해 그 설계 목적인 CPU 자원 분배에 있어 공평한 분배가 이루어졌는지에 대해 기존의 Linux 스케줄러와의 비교를 통해 분석 후 벤치마크를 통해 CFS 에서 공평한 자원 분배가 이뤄짐을 확인하였다 [9].

4. 머신러닝 IoT 단말 활용

사물인터넷 데이터가 발생하는 기기 위치 또는 기기에서 가까운 전송 현장에서 컴퓨팅을 통해 데이터 처리 대기시간 및 네트워크 대역폭을 줄이는 것을 가능케 하는 '에지 컴퓨팅'이 발달하고 있다 [10]. 그러나 현재 대부분 에지 컴퓨팅에서 머신러닝 등 AI 기술을 다루는 방식은 클라우드에 수집한 데이터를 전달하거나 자체적으로 데이터를 처리/분석하여 그 결과를 제공하는 '사용자 서비스'에 집중되는 경향이 있다. 이러한 사용자 서비스를 위한 머신러닝 활용뿐만 아니라, 표준 사물인터넷 플랫폼 nCube, Linux 파운데이션 오픈소스 플랫폼 EdgeX 등 IoT 에지 컴퓨팅을 위한 개방형 프레임워크에 추가되는 형태로 사물인터넷 기기의 리소스 활용을 돕기 위한 머신러닝 적용 연구가 필요하다.

Ⅲ. 리소스 활용 관리 보조 시스템 구조

사물인터넷 서비스 플랫폼 간 확장성 및 호환성을 위해서 표준 기반 소프트웨어 플랫폼 활용이 필요하다 [3]. 또한 표준 플랫폼 기반 사물인터넷 기기의 리소스가 제한되고 다중 센서 데이터 수집이 필요한 환경에서는 주어진 센싱 수집 시나리오를 만족하기 위해 (예를 들어, 센서마다 센싱 주기와 확보를 보장) 지능화된 리소스 활용 관리 보조 시스템이 필요하다. 따라서 본 논문에서는 리소스 사용 패턴을 학습하여 사물인터넷 기기의 리소스 활용을 보조하는 시스템을 다음과 같이 제안한다.

1. 머신러닝 기반 리소스 활용 관리 보조 시스템의 정의

사물인터넷 기기는 여러 센서가 디바이스 플랫폼에 연결되어 센서 데이터를 수집하고 서버에 송수신이 이뤄진다. 본 논문에서 제안하는 리소스 활용 관리 보조 시스템은 하나의 사물인터넷 기기에 여러 개의 센서가 연결되어 센서 데이터를 수집하는 다중 센싱 시나리오를 가정하였다.

이때 각 센서는 측정하는 물리 정보와 기기와의 통신 방법에 따라 자원 소모량이 다르고, 주어진 센싱 시나리오에 따라 센서마다 필요한 통신주기가 다를 수 있다. 따라서 센서가 획득할 물리 정보, 통신 방법, 통신주기 등 주어진 센싱 시나리오를 준수하며 리소스 할당을 관리 보조하는 방법이 필요하다.

예를 들어 디바이스 플랫폼이 동시에 수행하는 다른 작업 때문에 프로세스들이 경쟁적으로 커널에 CPU 사용을 요구하는 경우, 특정 센싱 프로세스가 정해진 주기에 데이터 획득과 전송이 이루어지지 못할 수 있다. 이 문제 발생 요인을 머신러닝을 통해 학습하여 예측할 수 있다면, 해당 프로세스의 커널 우선순위를 조절하여 (예, nice value를 변경해 CPU 커널의 우선순위를 조절) 센싱 시나리오 수행 실패를 예방할 수 있다.

2. 머신러닝 기반 리소스 활용 관리 보조 시스템 구조

제안하는 시스템 구조는 크게 다음과 같이 두 가지 기능을 가진다.

첫째는 주어진 센싱 시나리오에서 사물인터넷 기기의 리소스 활용 상태를 수집하기 위해, 센서마다 센싱 프로세스의 PID (Process identifier), CPU 사용량 등 정보를 일정 주기로 수집한다. Linux 운영체제 기반 사물인터넷 기기를 가정하였을 때, 내부 명령어 (예, ps, watch, tee)를 활용하면 각 센서 프로세스에 대한 시스템 리소스 수집과 기록을 자동화할 수 있다.

둘째는 수집 정보를 학습하여 기기 프로세스 관리 효율을 높이기 위한 제어부이다. 초기 Linux 시스템 스케줄러에서는 최소한의 설계를 사용하여 다수 프로세서나 하이퍼스레딩이 포함된 대형 아키텍처를 전혀 고려하지 않았으나, CFS 도입 후 nice value 조절로 프로세스의 타임 슬라이스 조정이 가능하게 되었다. 결과적으로 다수의 프로세서가 실행하고 있을 때 nice value 조정이 이뤄진 프로세스의 우선순위를 높여 해당 프로세스의 리소스 점유율을 높일 수 있게 되었다 [11].

CFS 스케줄러는 기본적으로 Linux 시스템의 실행 대기 상태인 프로세스들을 우선순위에 따라 최대한 공평하게 실행하도록 한다. 이때 nice value 값을 조정하면 CFS 스케줄러는 수식 (1)과 수식 (2)에 따라 타임 슬라이스를 변경한다. 수식 (1)에서 태스크 i의 타임 슬라이스는 대기 태스크들이 모두 수행되는 시간 Period와 (태스크 i의 가중치 / 전체 태스크의 가중치 총합)의 곱으로 표현된다. 수식 (2)는 nice value에 따른 태스크의 가중치 변화를 나타낸다. 예를 들어 표 1은 프로세스 P의 nice value 0에서 5로 변경할 때 타임 슬라이스의 변화에 관해 설명하고 있다. 표에서 볼 수 있듯이, nice value가 증가하면 타임 슬라이스 값이 감소하여, 결국 프로세스의 CPU 활용 가능 시간이 감소함을 알 수 있다.

$$TimeSlice = Period \times \frac{Weight(i)}{\sum Weight(i)}, \quad (1)$$

$$Weight \approx \frac{1024}{1.25^{NICE}}. \quad (2)$$

표 1. nice value에 따른 타임 슬라이스 변경 예제
Table 1. Example of modifying time slice according to nice value

Process	Nice	Ratio	Time Slice
P	0	75.3% = 1024 / (330+1024)	15.1ms
	5	24.6% = 335 / (335+1024)	4.9ms

Ⅳ. 개발 및 기능 검증

1. 센싱 시나리오

본 논문에서는 센싱 시나리오를 구현하기 위해 사물인터넷 표준 기반 오픈소스 연합체인 OCEAN (Open allianCE for iot stANdard)에서 공개한 oneM2M 표준 플랫폼을 이용한다 [12].

사물인터넷 기기에 연결된 센서에서 데이터를 수집하기 위해 oneM2M 디바이스 플랫폼인 nCube를 사용한다 [13]. nCube는 수집한 센서 데이터를 oneM2M 리소스로 변환하기 위해 TAS (Thing Adaptation Software)를 활용한다. TAS는 실제 사물 (즉, 센서)을 사물인터넷 기기에 연결하기 위한 일종의 미들웨어로써 리눅스 운영체제에서 하나의 프로세스로 구현될 수 있고, 사물과 nCube 간 연결 통로 역할을 한다. nCube가 수집한 센싱 데이터는 표준 REST API를 통해 oneM2M 서버 플랫폼인 Mobius로 전달된다 [14]. Mobius는 수집된 데이터에 접근하는 글로벌 표준 API를 제공할 뿐만 아니라 제어가 가능한 사물일 경우 (예, 액추에이터) 제어 명령을 전달하는 기능을 제공한다.

제안한 리소스 활용 관리 보조 시스템 구현을 위해, 대상 사물인터넷 기기는 NVIDIA의 Jetson Nano를 이용하였다. 표 2에 기술된 바와 같이 Jetson Nano는 다양한 I/O 인터페이스를 제공하기 때문에 하부에 다양한 통신 방법으로 센서 정보를 수집하는 시나리오를 구현할 수 있다.

센싱 시나리오는 다중 센싱 환경을 고려하여 USB/UART, I2C, SPI, GPIO 통신을 하는 총 10개의 센서가 하나의 사물인터넷 기기에 연결됨을 가정하였다 (미세먼지, 가속도, 조도, 소리, 재실, 초음파 등). 또한 제한된 리소스 환경을 가정하기 위하여 멀티 코어를 지원하는 Jetson Nano의 CPU 사용을 1코어로 제한하였다. 그림 1은 제안하는 센싱 시나리오를 나타내며, 그림 2는 실제로 구현된 사물인터넷 센싱 시스템을 보여준다. 표 3은 센싱 시나리오에 사용되는 센서들의 통신 방식을 기술하였다.

각 센서의 측정 주기와 서버에 전송할 정보를 계산하는 데 필요한 연산은 센서 특성과 활용할 수 있는 시나리오에 따라 다르게 지정하였다. 특히 PIR 센서와 ultrasonic 센서는 다양한 활용 시나리오에 따라 각기 다른 리소스 사용 패턴을 의도적으로 발생시키기 위해 동일한 센서를 여러 개

표 2. 사물인터넷 기기 하드웨어
Table 2. IoT device hardware

Device name	Jetson Nano
GPU	NVIDIA Maxwell architecture with 128 NVIDIA CUDA® cores
Processor	Quad-core, ARM Cortex-A57MPCore
Memory	4 GB 64-bit LPDDR4, 1600MHz 25.6 GB/s
OS	NVIDIA L4T 32.6.1 (Linux Kernel 4.9)
Storage	16 GB eMMC 5.1
I/O	**GPIO, I2C, I2S, SPI, UART**

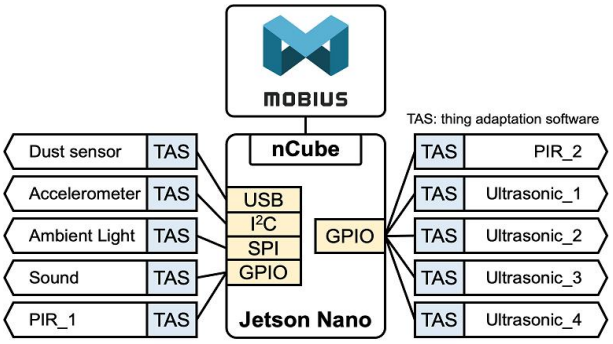

그림 1. 센싱 시나리오
Fig. 1. Sensing scenario

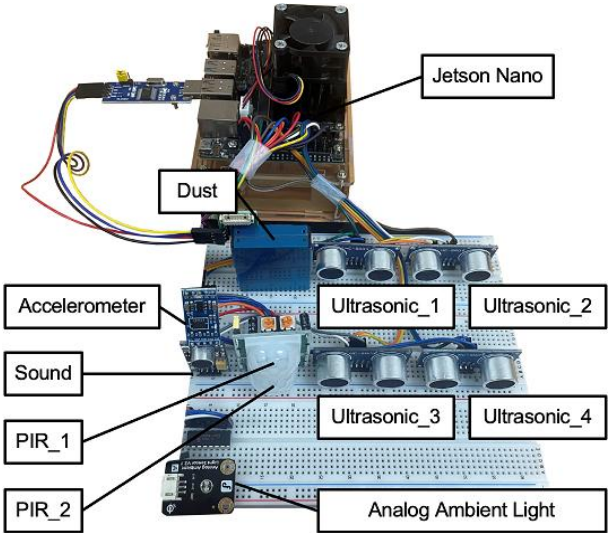

그림 2. 센싱 시나리오 구현
Fig. 2. Sensing scenario implementation

사용하였다 (PIR 센서는 두 개, ultrasonic 센서는 네 개).

표 4에서 볼 수 있듯이, PRI_1 센서의 경우 센서 자체에서 지원하는 최소 센싱 주기인 5초로 센싱 주기를 설정하였고, PIR_2 센서는 최대 센싱 주기인 200초로 센싱 주기를 지정하였다. 두 센서 모두 인터럽트 방식으로 센싱 중 움직임을 감지하고 즉시 서버에 센싱 데이터를 올리게 된다. 4개의 ultrasonic 센서는 각 센싱 주기와 센싱 데이터 전송 조건을 다르게 설정하였으며, Ultrasonic_1과 Ultrasonic_2는 3cm의 고정된 거리를 측정하고 Ultrasonic_3과 Ultrasonic_4는 가변 거리를 측정할 수 있도록 실험환경을 구성하였다.

2. 센서 측정 우선순위

다중 센싱 환경에서 주어진 시나리오에 따라 센서의 측정 우선순위가 달라질 수 있다. 본 논문에서는 센서의 우선순위를 센서마다 예상되는 요구 리소스 양에 따라 내림차순으로 설정하였다. 또한 동일한 센서라 할지라도 앞 장에서 설명한 센서의 활용 시나리오에 따라 다른 우선순위를 가진다. 예를 들어, 표 5에서 보이듯이, 동일한 ultrasonic 센서라도 전송 주기는 모두 5초로 같으나, 가장 많은 연산량이 예상되는 Ultrasonic_4가 나머지 ultrasonic 센서보다 높은 우선순위를 가진다.

센서의 측정 우선순위 범위는 각 프로세스 우선순위 조절을 위한 nice value 범위가 -19부터 20까지 총 40단계임을 고려하여 0부터 4까지 총 5단계로 이루어져 있으며, 각각 0은 Very High, 1은 High, 2는 Normal, 3은 Low, 4는 Very Low를 의미하는 우선순위로 설정하였다.

표 3. 센서 통신 방식
Table 3. Sensor communication protocols

Sensor	Protocol
Dust (PMSA003)	USB (UART coversion)
Accelerometer (MPU-6050)	I2C
Analog Ambient Light (DFR0026)	SPI
Sound (LM386)	GPIO (interrupt)
PIR_1 (HC-SR501)	GPIO (interrupt)
PIR_2 (HC-SR501)	GPIO (interrupt)
Ultrasonic_1 (HC-SR04)	GPIO
Ultrasonic_2 (HC-SR04)	GPIO
Ultrasonic_3 (HC-SR04)	GPIO
Ultrasonic_4 (HC-SR04)	GPIO

표 4. 동일 센서의 차이점
Table 4. Differences between the same sensors

Sensor	Sensing and upload method
PIR_1 (HC-SR501)	Sensing every 5 seconds and uploading to the server upon detection
PIR_2 (HC-SR501)	Sensing every 200 seconds and uploading to the server upon detection
Ultrasonic_1 (HC-SR04)	Sensing and sending to server every 5 seconds
Ultrasonic_2 (HC-SR04)	Sensing every 5 seconds and sending 5 measurement results to the server when accumulated
Ultrasonic_3 (HC-SR04)	Upload it to the server every 5 seconds If the measured value differs by 2 cm or more, the data is updated.
Ultrasonic_4 (HC-SR04)	Upload it to the server every 5 seconds If the measured value differs by 5 cm or more, the data is updated.

표 5. 센서 측정 우선순위
Table 5. Priority of measuring sensors

Sensor	Scenario Priority
Dust (PMSA003)	3, Low
Accelerometer (MPU-6050)	2, Normal
Analog Ambient Light (DFR0026)	3, Low
Sound (LM386)	0, Very High
PIR_1 (HC-SR501)	4, Very Low
PIR_2 (HC-SR501)	4, Very Low
Ultrasonic_1 (HC-SR04)	2, Normal
Ultrasonic_2 (HC-SR04)	1, High
Ultrasonic_3 (HC-SR04)	1, High
Ultrasonic_4 (HC-SR04)	0, Very High

3. 우선순위 조절을 위한 nice value 제어

다중 센서를 가진 사물인터넷 플랫폼의 시스템 리소스 관리는 수동 조정 또는 많은 규칙 기반 제어가 필요하며 이는 전통적 프로그래밍 방식으로 해결하기 어려운 복잡한 문제이다. 예를 들어, 새로운 센서가 추가되거나 센서의 동작 시나리오가 변경되는 경우 관련된 모든 규칙 수정이 필요하다.

이를 보완할 방법으로 수집되는 데이터를 기반으로 규칙을 반영한 모델을 생성할 수 있는 학습기법이 최근 많이 활용되고 있다. 특히 머신러닝 기법을 리소스 관리를 위해 도입하면 도메인 전문가에 의한 특성 추출과 다양한 알고리즘을 적용할 수 있다는 장점과 더불어 추가적인 데이터 수집 기반 온라인 학습으로 '자동으로 변화에 적응'하는 장점이 있기에, 이는 수동으로 규칙 업데이트가 이뤄지는 방식에 비해 효율적이라 볼 수 있다.

따라서 본 논문에서 제안하는 nice value 제어부는 실시간으로 시스템 리소스 상태 변화를 학습하여 주어진 센싱 시나리오와 측정 우선순위를 보장하기 위해 각 센싱 프로세스의 nice value를 자동으로 조절한다. 전통적 규칙 기반 프로그래밍 방법은 센싱 시나리오에 변화가 동적으로 이루어지는 경우 (예, 센서 추가 또는 주기 변화) 규칙 수정이 필요하고 리소스 상태 변화에 실시간으로 대응하기 어렵다.

nice value 제어를 위해, 먼저 주어진 센싱 시나리오, 센서 측정 우선순위, 현재 시스템 리소스 상태를 머신러닝 모델의 특성으로 설정한다. 다음으로 현재 nice value 대비 얼마만큼의 조절 (증가 또는 감소)이 필요한지 머신러닝 모델로 예측하여, 그 결괏값을 Linux 시스템의 renice 명령어에 적용하는 방식으로 해당 센서가 갖는 프로세스의 nice value를 조절한다.

4. 데이터 수집

일반적인 Linux 시스템에서 리소스 활용 상태를 일반적인 Linux 시스템에서 리소스 활용 상태를 파악하기 위해 수집 가능한 정보는 다양하나, 본 논문에서 제안하는 머신러닝 기반 리소스 활용 패턴 파악을 위해 표 6에 보이는 바와 같이 총 6개 시스템 정보를 수집하였다.

실제 다중 센싱 시나리오에서 사용되는 센서 수와 종류는 시나리오에 따라 다르다. 또한 센서마다 측정을 위한 계산량과 전송을 위한 시스템 리소스 사용 패턴이 다를 수 있다. 따라서 본 논문에서 제안하는 데이터 수집 방법은 다음과 같다.

먼저 센싱 시나리오 변화에 따라 새로운 센서 하나가 추가됨을 가정한다. 추가된 센서 측정과 데이터 전송을 위한 시스템 프로세스를 모방하기 위해, CPU에 무작위 연산을 통한 부하를 주는 가상 TAS 프로그램을 개발하였다. 대상 사물인터넷 기기가 싱글 코어 환경임을 가정했기 때문에 가상 TAS 프로그램은 0~70 사이의 랜덤한 PCPU 값을 갖게 하였고, 따라서 전체 PCPU 항목을 100 이하로 제한할 수 있다. 최종적으로 리소스 데이터 수집을 위해, TAS 프로그램을 1분마다 정지와 구동을 반복하여 데이터 수집에 있어 다양성을 부가하였다.

표 6. 시스템 리소스 수집 항목
Table 6. System resource collection category

Title	Descriptions
COMM	Command name
TIME	Process resource check time (YY.MM.DD-T-HH.MM.SS)
PID	Process ID
PCPU	Cpu utilization [%]
PRI	Priority of the process
NI	Nice value

표 7. 추출된 특성 설명
Table 7. Descriptions of extracted features

Feature	Descriptions
PCPU	Cpu utilization [%]
ALL_PCPU	Sum of PCPU of all sensor processes
PRI	Priority of the process
S_PRI	Scenario Priority of the process

TAS 프로세스의 nice value 조절 값 예측을 위한 머신러닝 모델의 특성은 PCPU, 전체 센서 프로세스의 PCPU의 합을 나타내는 ALL_PCPU, PRI, 그리고 센서의 시나리오 우선순위인 S_PRI (Scenario Priority)으로 구성된다. 표 7은 추출한 특성에 대한 설명을 기술하고 있다. 본 논문에서 활용한 센서는 모두 전체 메모리 대비 1% 이하 메모리 사용량을 보였기에, 메모리 관련 정보는 머신러닝 모델의 특성에서 제외하였다.

데이터 수집 과정은 각 센서 샘플링 주기와 시스템 리소스 정보 수집 프로그램의 처리 속도를 고려하여, 실험실 환경에서 하루 동안 10초 주기로 총 8640번 데이터를 수집하고 머신러닝 모델을 학습시켜 실험을 진행하였다.

머신러닝 모델 학습을 위한 타깃값 (nice value 조절 값, $NICE_{control}$) 정의를 위해 수식 (3)으로 표현된 알고리즘을 사용하였다. 이 알고리즘은 대상 TAS 프로세스의 CPU 사용량 ($PCPU_{TAS}$), 전체 프로세스 CPU 사용량 ($PCPU_{ALL}$), 총 5단계인 시나리오 우선순위를 고려한다. nice value 조절 값은 조절 시나리오에 따라 스케일값을 변경할 수 있다 (수식에서는 5 사용).

$$NICE_{control} = \frac{PCPU_{TAS}}{PCPU_{ALL}} \times (5 - Priority_{Scenario}) \times 5. \quad (3)$$

수집한 데이터셋에 위 알고리즘을 적용하여 타깃값들을 계산한 결과, Dust, Analog ambient light, PIR_1, PIR_2, Ultrasonic_1, Ultrasonic_2 센서들은 모두 $NICE_{control}$이 0으로 동일한 값을 가졌다. 이는 위 센서들은 측정 및 전송 시 리소스 사용이 적어 가상 TAS 프로세스와 관계없이 잘 동작함을 의미하고, 따라서 추가적인 머신러닝 모델 생성 및 실험 검증과정에서 제외하였다.

표 8. 머신러닝 알고리즘 간의 성능 비교
Table 8. Performance comparison among machine learning algorithms

Method	Precision	Recall	F1-score
ExtraTreesClassifier	0.998	0.999	0.998
SVM (SVC)	0.991	0.981	0.986
RandomForestClassifier	0.949	0.949	0.949
KNeighborsClassifier	0.897	0.895	0.896

표 9. 10-폴드 교차 검증 실험의 혼동 행렬 결과
Table 9. Confusion matrix results of 10-fold cross-validation

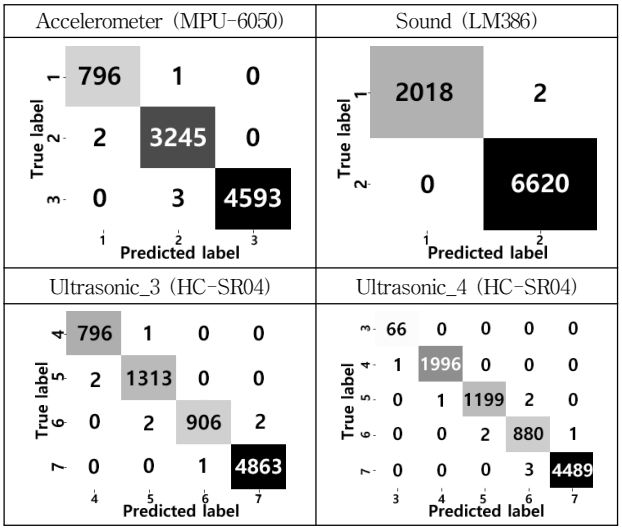

표 10. 비교 실험 결과
Table 10. Comparative experiment results

Sensor	Nice value control method	
	Machine learning	If then rule
	Number of failure	
Accelerometer	3	7
Sound	1	2
Ultrasonic_3	27	45
Ultrasonic_4	21	39
Sum	**52**	**93**

5. 머신러닝

학습 모델의 생성과 검증을 위해 파이썬 기반 머신러닝 라이브러리인 Scikit-Learn을 활용하였다. 학습 모델의 성능 측정 지표로는 타깃값에 따라 불균형 데이터셋이 수집될 수 있으므로 정확도 (accuracy)는 제외하고, 정밀도 (precision), 재현율 (recall), F1-score를 채택하였다. 연결한 10개 센서마다 머신러닝 모델을 생성하여 성능을 평가하고, 총 10개 성능 지표들을 평균 내어 최종 성능 지표로 산출하였다.

머신러닝 알고리즘으로는 인스턴스 기반 학습 알고리즘인 k 최근접 이웃 (k-nearest neighbor), 클래스 간 마진을 보장하는 서포트 벡터 머신 (SVM, support vector machine), 결정 트리 앙상블인 랜덤 포레스트 (random forest), 임의 특성 선택을 반영한 엑스트라 트리 (extremely randomized tree)를 선택하였다. 다음 표 8에는 선택한 네 개의 머신러닝 알고리즘에 대하여 10-폴드 교차 검증 후 각 최종 성능 측정 지표의 결괏값을 보여준다. 네 개의 머신러닝 알고리즘 중 특성값을 활용하여 가장 높은 성능 지수로 nice value 조절 값을 예측한 엑스트라 트리 알고리즘을 향후 기능 검증을 위해 채택하였다 (F1-score 기준 0.998).

표 9는 위에서 설명한 엑스트라 트리의 10-폴드 교차 검증 실험의 혼동 행렬 결과를 보여준다. 혼동 행렬 분석 결과, Accelerometer, Sound, Ultrasonic_3, Ultrasonic_4 센서에 대해 학습된 머신러닝 모델이 각 프로세스 리소스 활용 상태에 따라 요구되는 nice value 조절 값을 높은 성능으로 분류할 수 있음을 보인다. 따라서, 학습된 머신러닝 모델과 리소스 상태에 따라 예측된 nice value 조절 값을 다음 장에 기술할 기능 검증에 활용한다.

6. 기능 검증

최종적으로 대상 사물인터넷 기기인 Jetson Nano에 연결된 10개 센서에 대해 측정 및 데이터 전송 과정에서 수집한 데이터셋을 기반으로 학습한 엑스트라 트리 모델을 사용해 각 센싱 프로세스의 우선순위 제어를 위한 nice value 조절 값 예측이 가능하다.

제안한 기법의 기능 검증을 위해 다음과 같은 테스트 시나리오를 구현하였다. 먼저 데이터 수집과정과 유사하게 무작위 연산을 통해 CPU에 부하를 주는 가상 TAS 프로그램을 구동하여 센싱 시나리오가 변하는 환경을 조성하였다. 이 과정에서 센서마다 주어진 측정 및 전송 조건을 위배하였을 경우 (예, 전송 주기인 5초 이후 측정 데이터 전송이 이루어졌을 때), 이를 해당 프로세스의 '실패 (failure)'로 정의하였다.

센싱 프로세스의 실패를 줄이기 위해 리소스 활용 관리 기법을 적용해 볼 수 있다. 제안한 머신러닝 기반 기법과 성능 비교를 위해 단순 규칙 기반 (if then rule) nice value 조절 방법도 다음과 같이 검증을 위해 구현하였다.

단순 규칙 기반 nice value 조절은 센서의 우선순위를 배제하고 해당 TAS 프로세스의 PCPU만을 고려하여 요구되는 nice value를 직접 renice 명령어를 통해 부여한다. 수식 (4)는 이 규칙을 적용하기 위해 0부터 100까지의 PCPU와 -19부터 20의 nice value의 선형 변환 과정을 수식화하였다.

$$NICE = 20 - PCPU_{TAS} \times 0.4. \quad (4)$$

표 10은 두 방법의 성능 비교를 위해 표 9에서 선택된 네 개 센서를 대상으로, 두 방법을 테스트 시나리오에 적용하고 검증 시간 동안 실패 빈도를 비교하였다.

비교 실험의 결과를 토대로, Ultrasonic_3, 4의 경우 센서 특성상 거리 측정을 위한 기본적으로 요구되는 연산량이 있고 측정 대상까지 거리가 변화할 수 있기에 연산 시간이 달라질 수 있다. 그러나 센싱 정보 전송이 주기적으로 정확하게 이루어져야 하므로 가상 TAS 프로세스 PCPU가 상대적

으로 커졌을 경우 다른 센서들에 비해 실패 빈도가 높음을 확인할 수 있다.

Accelerometer와 Sound의 경우 통신 인터럽트와 GPIO 인터럽트를 사용하는 방식이기 때문에 가상 TAS 프로세스와 관계없이 ultrasonic 센서에 비해 안정적으로 센싱 데이터를 서버에 전송함을 확인할 수 있다.

머신러닝 기반 기법과 단순 규칙 기반 기법을 비교했을 때, 머신러닝 기법은 총 52번의 실패 빈도가 발생한 것과 달리 단순 규칙 기법은 총 93번의 실패 빈도가 발생하였다. 결과적으로 제안한 머신러닝 기반 리소스 활용 관리 보조 기법이 동적 센싱 시나리오 환경에서 센서 동작 실패를 줄이고 효용성을 확인할 수 있다.

V. 한계점과 향후 과제

본 논문에서 데이터 수집을 위해 구현한 센싱 시나리오 환경은 연결한 10개 센서의 TAS 프로세스가 차지하는 PCPU의 총합이 최대 52%이었으며, Linux 시스템에서 메모리 사용량을 나타내는 PMEM의 합의 최대는 5.4%이었다. 싱글 코어 사물인터넷 기기를 가정하였기에 최대 PCPU 합이 100 이하가 되도록 가상 TAS 프로세스를 모방하였으나, 멀티 코어 기기를 위한 실험이 추가로 필요할 것이다. 또한 많은 메모리가 필요하거나, 센서가 유선이 아닌 무선 네트워크 연결 방식일 경우 기기의 전력 사용량이 달라지므로 [15], 이를 고려하여 센싱 시나리오 동작의 안정성을 높이기 위한 연구도 필요하다.

현재는 각 센서 프로세스의 실패를 판단하기 위해 전송 주기를 정확하게 지켰는지만 확인하고 있다. 그러나 정확한 전송 주기뿐만 아니라 수집한 센싱 정보의 정확성에 대한 검증이 추가로 필요하다. 즉, 가상 TAS 프로세스와 관계없이 센싱 정보 전송이 주기에 따라 정확하게 이루어졌을지라도 그 정보가 정확한 측정 정보인지 확인하여 센서 프로세스 실패를 추가로 판단하는 연구가 필요하다.

마지막으로, 본 논문에서 사용한 Jetson Nano는 전통적인 머신러닝뿐만 아니라 최근 많은 연구에서 활용하는 딥러닝 학습도 가능하다. 딥러닝을 활용할 때 센서마다 학습 모델을 생성하는 방식 외에도, 전체 센싱 프로세스를 특성으로 활용하여 센서마다 적합한 nice value를 예측할 수 있는 통합 신경망 기반 학습 모델을 활용할 수 있을 것이다.

VI. 결 론

국제 사물인터넷 표준인 oneM2M과 같이 서비스 계층 표준을 기반으로 개발된 소프트웨어 플랫폼은 센서 등 사물과의 연결성을 제공하기에는 이상적이나, 플랫폼이 설치되는 사물인터넷 기기의 리소스 상태와 활용 방식은 고려하고 있지 않다. 특히 리소스가 제한된 환경이라면 스스로 자신의 리소스 사용 상태를 파악하고 이에 따라 한정된 리소스의 적절한 분배가 가능한 리소스 활용 방법 개발이 필요하다.

본 논문에서는 프로세서의 최대 연산량이 제한된 사물인터넷 센싱 시나리오를 가정하고 다양한 통신 방법으로 사물인터넷 기기와 센서를 oneM2M 표준 플랫폼인 nCube와 미들웨어인 TAS를 이용해 연결하였다. 이 환경에서 프로세스의 리소스 사용 상태를 머신러닝을 위한 특성벡터로 정의하고, 가상 TAS 프로세스가 임의 연산량을 가지고 발생하는 경우 안정적인 동작을 위해 요구되는 nice value 조절 값을 학습을 통해 예측하였다. 기능 검증을 위해 테스트 시나리오를 구현하고 단순 규칙 기반 관리 기법과 비교하여 제안한 기법의 효용성을 검증하였다.

References

[1] J. Kim, S. C. Choi, J. Yun, J. W. Lee, "Towards the OneM2M Standards for Building IoT Ecosystem: Analysis, Implementation and Lessons," Peer-to-Peer Networking and Applications. Appl. Vol. 11, pp. 139-151, 2018.

[2] TTA. TTAR-10.0137. Technical Specification, 2020 (in Korean).

[3] N. M. Sung, J. Yun, "Self-adaptive IoT Software Platform for Interoperable Standard-based IoT Systems," IEMEK J. Embed. Sys. Appl. Vol. 12, No. 6, pp. 369-375, 2017 (in Korean).

[4] S. W. Cheng, A. C. Huang, B. Schmerl, "Rainbow: Architecture-Based Self-Adaptation with Reusable Infrastructure," IEEE on Computer, Vol. 37, No. 10, pp. 46-54, 2004.

[5] P. Bresciani, P. Giorgini, F. Giunchiglia, J. Mylopoulos, A. Perini, "Tropos: An Agent-Oriented Software Development Methodology. Autonomous Agents and Multi-Agent Systems," Autonomous Agents and Multi-Agent Systems, Vol. 8, No. 3, pp. 203-236, 2004.

[6] M. Morandini, L. Penserini, A. Perini, "Towards Goal-Oriented Development of Self-Adaptive Systems," SEAMS '08 Proceedings of the 2008 International Workshop on Software Engineering for Adaptive and Self-managing Systems, pp. 9-16, 2008.

[7] J. Jang, Y. Lee, J. Hong, "Task Priority Control Method Based on the Characteristics of Applications in CFS," The Journal of the Korea Contents Association, Vol. 21, No. 6, pp. 12-18, 2021 (in Korean).

[8] N. N. Jain, S. K. R, S. Akram, "Improvising Process Scheduling Using Machine Learning," 3rd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology, pp. 1379-1382, 2018.

[9] C. S. Wong, I. K. T. Tan, R. D. Kumari, J. W. Lam, W. Fun, "Fairness and Interactive Performance of O(1) and CFS Linux Kernel Schedulers," 2008 International

Symposium on Information Technology, Vol. 4, pp. 1-8, 2008.
[10] IITP, Weekly ICT Trends, No. 2024, pp.6, 2021.
[11] S. M. Mostafa, S. Kusakabe, "Achieving Better Fairness for Multithreaded Programs in Linux using Group Threads Scheduler," 2013 International Workshop on ICT at Beppu, 2013.
[12] http://www.iotocean.org.
[13] http://developers.iotocean.org/archives/module/ncube-thyme-nodejs.
[14] http://developers.iotocean.org/archives/module/mobius.
[15] H. S. Chung, H. B. Lee, T. Y Chung, "Development and Performance Analysis of an Effective Smart Plug System based on K10026 Regulation," IEMEK J. Embed. Sys. Appl. Vol. 11, No. 5, pp. 287-298, 2016 (in Korean).

Seongchan Lee (이 성 찬)

2017~Department of Internet of Things from Soonchunhyang University, Republic of Korea (B.S.)

Field of Interests: HCI, IoT, AI-enabled applications
Email: seongchanno1@gmail.com

Nakmyoung Sung (성 낙 명)

2010 Digital Information from Hankuk University of Foreign Studies (B.S.)
2015 Department of Electronics Engineering from Hankuk University of Foreign Studies (M.S.)

Career:
2011~ Autonomous IoT Research Center from KETI (Team Leader / Senior Researcher)
Field of Interests: Autonomous IoT, Smart City
Email: nmsung@keti.re.kr

Seokjun Lee (이 석 준)

2011 Computer Science from Yonsei University (B.S.)
2018 Computer Science from Yonsei University (Ph.D.)

Career:
2018~2019 NAND UFS Firmware Development Team from SK Hynix (TL)
2019~ Autonomous IoT Research Center from KETI (Senior Researcher)
Field of Interests: Autonomous IoT, Smart City
Email: sjlee88@keti.re.kr

Jaeseok Yun (윤 재 석)

1997 Electronical Engineering from Chonnam National University (B.S.)
1999 Mechatronics Engineering from GIST (M.S.)
2006 Mechatronics Engineering from GIST (Ph.D.)

Career:
2006~2009 PostDoc Research Scientist, Georgia Tech
2009~2016 Senior Researcher, KETI
2016~ Dept of IoT, Soonchunhyang University (Assist. Prof.)
Field of Interests: Ubicomp, IoT, AI-enabled applications
Email: yun@sch.ac.kr