논문 2023-18-36

KubEVC-Agent : 머신러닝 추론 엣지 컴퓨팅 클러스터 관리 자동화 시스템

(KubEVC-Agent : Kubernetes Edge Vision Cluster Agent for Optimal DNN Inference and Operation)

송 무 현, 김 규 민, 문 지 훈, 김 유 림, 남 채 원, 박 종 빈, 이 경 용*
(Moohyun Song, Kyumin Kim, Jihun Moon, Yurim Kim, Chaewon Nam, Jongbin Park, Kyungyong Lee)

Abstract : With the advancement of artificial intelligence and its various use cases, accessing it through edge computing environments is gaining traction. However, due to the nature of edge computing environments, efficient management and optimization of clusters distributed in different geographical locations is considered a major challenge. To address these issues, this paper proposes a centralization and automation tool called KubEVC-Agent based on Kubernetes. KubEVC-Agent centralizes the deployment, operation, and management of edge clusters and presents a use case of the data transformation for optimizing intra-cluster communication. This paper describes the components of KubEVC-Agent, its working principle, and experimental results to verify its effectiveness.

Keywords : Edge Cluster, Container Ochestration, Machine Learning Inference, Infrastructure as Code, Data Transform

Ⅰ. 서 론

인공지능 서비스 개발에 필수적인 머신러닝 알고리즘과 개발 플랫폼 및 다양한 하드웨어 가속기의 발전은 많은 분야에서 새로운 응용 사례를 제시해 주며 활용 범위를 넓혀 가고 있다. 인공지능 학습 및 추론 서비스의 경우 머신러닝 개발 초기에는 고성능 장비를 장착한 데이터 센터에서 주된 작업이 이루어졌다. 그러나, 원격의 데이터 센터와의 통신을 위한 네트워크 지연시간 발생, 데이터 보안 위협 등의 이유로 최근에는 엣지 컴퓨팅 환경에서 다양한 시나리오의 머신러닝 응용 및 알고리즘이 개발되고 있다 [1]. 머신러닝 추론은 사용자의 입력이 주어지면 머신러닝 모델에서 정의한 결괏값을 반환해 준다. 따라서, 사용자의 위치와 가까운 곳에서 서비스를 제공해 주는 것이 네트워크 지연시간이 적어 성능 차원에서 이득이 있을 수 있기에 엣지 컴퓨팅 자원을 활용하는 것이 효과적일 수 있다 [2].

엣지 컴퓨팅 환경에서 작동하는 머신러닝 모델 및 이미지 분류, 자연어 처리와 같이 인공지능 서비스의 종류도 다양해지고 있다. 그러나, 엣지 컴퓨팅 환경에서 워크로드 별로 적합한 여러 기기에서 추론 및 학습 작업을 관리하는 것은 어려워지고 있다 [3]. 또한, 엣지 컴퓨팅의 경우, 여러 지리적 위치에 분포하여 클러스터 단위로 배치된다. 분산 배치된 클러스터를 관리할 때 별도의 에이전트나 쿠버네티스 [4]와 같은 관리 플랫폼을 사용하지 않으면 사용자가 각 클러스터 장치에 직접 또는 원격으로 장치에 접근하여 소프트웨어 업데이트 또는 보안 패치와 같은 유지보수를 개별적으로 수행하게 된다. 이는 사용자에게 불편을 초래할 뿐만 아니라 엣지 클러스터의 확장성 측면에서도 제약이 생길 수 있다. 엣지 컴퓨팅은 클라우드 컴퓨팅에 비해 자원도 제한적이므로 효율적인 소프트웨어 선택과 최적의 아키텍처 설계 방안이 중요하다. 더 나아가 엣지 컴퓨팅 클러스터에서 머신러닝 추론을 진행하기 위해서는 각 장치에 모델을 배포하는 방안이나, 클러스터 내부 통신 프로토콜 등을 고려해야 한다. 쿠버네티스가 발전함에 따라 다양한 기능이 추가되고 있지만, 인공지능 서비스 및 알고리즘 개발자에게 클러스터 관리를 위한 개발 환경 구축 문서 작성, 모니터링 구현, 서비스의 확장성 구성은 여전히 매우 어려운 일이다.

본 논문에서는 엣지 클러스터의 배포, 운영, 관리의 중앙 집중화 및 자동화를 가능하게 하고, 클러스터 내부 통신의 모범 사례를 쿠버네티스 엣지 비전 클러스터 에이전트, KubEVC-Agent를 통해 제시한다. KubEVC-Agent는 앤서블 (Ansible) [5] 과 같은 코드 형 인프라 (Infrastructure as Code, IaC) 도구 [6] 및 컨테이너 오케스트레이션 도구인 쿠버네티스를 사용하여 자동화 및 중앙 집중화를 달성하였다. 또한, 클러스터 내부 통신에 Protobuf 데이터 형식을 사용해 클러스터 내부 통신을 최적화하여 머신러닝 엣지 클러스터 환경의 모범 사례를 제시하였다. 쿠버네티스는 또한 임베디드, IoT 기기 등 저사양 장치에서 클러스터를 효율적으로 구성할 수 있는 K3S [7]와 같은 경량 쿠버네티스 (Lightweight Kubernetes) 배포판도 있어 이에 대한 옵션도

*Corresponding Author (leeky@kookmin.ac.kr)
Received: Sep. 27, 2023, Revised: Oct. 31, 2023, Accepted: Nov. 6, 2023.
M. Song, K. Kim, J. Moon, Y. Kim, C. Nam: Kookmin University (B.S. Student)
J. Park: KETI (Principal Researcher)
K. Lee: Kookmin University (Assoc. Prof.)
※ 이 논문은 2023년도 정부 (과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원을 받아 수행된 연구임 (No. 2021-0-00907, 능동적 즉시 대응 및 빠른 학습이 가능한 적응형 경량 엣지 연동분석 기술개발).

ISSN : 1975-5066
http://dx.doi.org/10.14372/IEMEK.2023.18.6.293

제공하며 일반 쿠버네티스와의 비교도 진행하였다.

본 논문은 다음과 같이 구성되어 있다. 2장에서는 KubEVC-Agent에 사용된 관련 기술에 대해 다룬다. 2장 후반에서는 KubEVC-Agent의 구성요소와 동작 원리를 설명한다. 5장 후반에서는 KubEVC-Agent의 실행 데모와 이를 사용함으로써 얻을 수 있는 이점에 대한 실험 결과를 보여준다.

Ⅱ. 관련 기술

1. 컨테이너 오케스트레이션

한정된 양의 적은 컨테이너는 관리자가 직접 제어하고 관리할 수 있다. 그러나 분산된 시스템 아키텍처 전반에서 컨테이너를 이용할 땐 관리와 운영이 복잡해진다. 컨테이너 오케스트레이션은 분산된 시스템에서 컨테이너화된 애플리케이션에 대한 스케일링, 클러스터링, 로깅 및 모니터링에 대한 관리 및 제어를 자동화해 준다. 대표적인 도구로는 쿠버네티스 (Kubernetes, K8S)가 있다.

1.1 쿠버네티스

쿠버네티스는 컨테이너의 집합인 파드 (Pod)를 기본 단위로 사용하여 호스트에 파드를 배치하거나 관리한다. 클러스터는 크게 두 가지 노드 형태인 마스터 노드와 워커 노드로 구성된다. 마스터 노드는 클러스터 전체를 관리하는 컨트롤 플레인 (Control Plane) 역할을 수행한다. 워커 노드는 컨트롤 플레인으로 부터 명령을 전달받아 실제 컨테이너 애플리케이션을 구동하는 역할을 한다.

쿠버네티스 클러스터 내의 기본적인 구성단위를 오브젝트라 표기하며, 컨테이너화 되어 배포되는 애플리케이션의 구성요소로는 파드, 볼륨, 서비스, 네임스페이스가 있다. 파드는 하나 또는 여러 개의 컨테이너의 묶음이다. 쿠버네티스는 단일 컨테이너가 아닌 파드 단위로 하여 하나 이상의 컨테이너를 배포한다. 이때, 디플로이먼트 (Deployment)를 사용하여 하나 이상의 파드의 배포 및 관리를 담당하도록 구성할 수 있다. 파드 내의 컨테이너는 네트워킹과 스토리지, 볼륨을 공유하며, 동일한 노드에서 실행된다. 서비스는 클러스터 내에서 사용할 수 있는 파드의 집합에 대한 네트워크 엔드포인트를 제공하고 로드밸런싱을 수행한다. 네임스페이스는 클러스터 내의 오브젝트들을 논리적으로 분리할 수 있게 해준다. 클러스터 외부 트래픽을 내부로 라우팅하기 위해 인그레스 (Ingress)를 사용할 수 있다.

이런 특징과 장점에도 불구하고, 쿠버네티스에 입문하여 실제 사용하기까지의 진입 장벽이 높아 초기 설정에서 많은 시간이 소요되며, 높은 이해도가 필요하다.

1.2 경량 쿠버네티스

컴퓨팅 자원이 적은 시스템에서는 쿠버네티스를 원활하게 사용하기가 어려울 수 있다. 쿠버네티스 배포판 중 사물인터넷, 엣지, ARM 시스템 등 저사양 환경에서 사용하기 위해 만들어진 것을 경량 쿠버네티스라 일컬으며, 대표적으로 K3S가 있다.

K3S는 간편한 설치와 K8S에 비하여 더 낮은 메모리를 이용하고, 모든 걸 100MB 미만의 바이너리로 제공한다. 이러한 이유로 쿠버네티스, K8S의 절반이라는 의미의 K3S라는 이름이 표기되었다. 일반적으로 K3S는 쿠버네티스 (K8S)와 동일한 동작 매커니즘을 따른다.

단일 바이너리로 구성된 만큼 설치 과정이 간단하다. K8S의 데이터 저장소인 Etcd 대신에 SQLite로 대체하고, 내장된 Flannel CNI를 사용하여 CNI 설정이 단순화되고 경량화되었다.

2. 코드 형 인프라

코드 형 인프라는 컴퓨팅 환경의 서버, 네트워크, 스토리지와 같은 자원을 코드를 통해 관리하고 생성하는 관리 프로세스를 의미한다. 코드로 작성하므로 소프트웨어 개발에서 사용되는 버전 관리와 같은 기법을 사용할 수 있고 배포를 자동화하여 인프라를 효율적으로 관리할 수 있다.

코드 형 인프라 구축이 가능하게 만드는 대표적인 도구로는 테라폼 (Terraform)과 앤서블, 쿠버네티스가 있다. 테라폼은 인프라를 코드로 정의하고 관리하는 인프라 생성 도구이다. 앤서블은 구성 관리 및 자동화 도구로, 서버 내부 설정 자동화에 사용된다. 쿠버네티스는 컨테이너 애플리케이션 배포 및 스케일링의 코드 형 인프라를 제공한다.

3. 엣지 클러스터 통신 프로토콜

클러스터 내에서 통신할 때는 범용적으로 사용되는 REST와 특정 프로그램에 맞도록 설계된 RPC를 사용할 수 있다. REST는 주로 GET, POST, PUT, DELETE와 같은 HTTP 메서드를 사용하는 직관적인 API를 제공하는 반면에 RPC는 원격 서버나 다른 프로세스에서 함수 또는 메서드를 호출하기 위한 프로토콜과 매커니즘을 제공한다. 또한, REST는 주로 JSON과 같은 텍스트 기반의 데이터 형식이 사용되지만, RPC는 이미지 형식 지원이 가능하며 이진 데이터 포맷을 사용할 수 있다. 마지막으로 REST는 자원과 상태를 중심으로 설계되어 리소스와 URI를 사용해 자원을 식별하는 한편, RPC는 메서드 호출을 중심으로 설계되어 클라이언트가 원격 메소드를 직접 호출한다.

gRPC [8]는 데이터 교환 형식으로 Protobuf를 사용하는 것을 통해 데이터를 직렬화하여 데이터 크기를 축소하기 때문에 네트워크 지연시간에서 이점을 가져갈 수 있다 [9]. Protobuf에서 부동소수점을 직렬화하는 방법은 다음과 같다. 첫째, IEEE 754 기반으로 이진화된 소수를 8비트 단위의 블록으로 나눈다. 둘째, 각 블록의 MSB에 0을 추가하여 9비트 블록을 만든 뒤, 3자리 단위로 나눈다. 셋째, 나눠진 블록을 8진수로 변환한다. 넷째, 모든 블록의 변환이 완료되면, 리틀-엔디안 순서로 재배치하여 직렬화한다. 데이터를 텍스트로 변환하여 전송하는 JSON 방식과 다르게 전송하고자 하는 행렬의 이진 데이터를 추가적인 자료형 변환 없이 저장하여 데이터의 크기가 작아진다. 그러므로 대역폭이 제한된 엣지 환경에서 효율적인 통신을 가능하게 한다. 또한 직접적인 함수 호출을 통한 효율적인 통신을 하여 네트워크 지연시간이 줄어들 수 있다.

Ⅲ. KubEVC-Agent

1. KubEVC-Agent 개요

머신러닝 엣지 클러스터 구성 자동화 및 관리 중앙 집중화, 내부 통신의 모범 사례를 제시하는 KubEVC-Agent는 엣지 클러스터를 구성하기 위한 솔루션으로 쿠버네티스를 사용하고 코드 형 인프라 도구, 프로메테우스 (Prometheus) [10], 그라파나 (Grafana) [11]와 같은 오픈소스 모니터링 도구를 사용하여 아래와 같은 내용을 수행하는 모듈로 이루어져 있다.

- 새 노드를 위한 필요 소프트웨어 설치 및 시스템 구성
- 새로운 클러스터 생성
- 기존 클러스터에 새 노드 가입
- 클러스터에 머신러닝 추론 서비스 배포
- 유지보수를 위한 중앙 집중 모니터링 솔루션 제공

이를 통해 쿠버네티스를 이용하여 엣지 클러스터를 배포할 때도 관리 복잡성과 같은 사용자의 진입 장벽을 낮추고, 모범 사례 및 참조 아키텍처의 기반이 될 수 있는 솔루션을 구현하였다. 또한 오픈소스로 공개하여 누구나 이용, 개선 방향 제시, 기여를 할 수 있도록 하였다 [12].

2. 환경 구성 모듈

새로운 장치가 클러스터에 가입하기 위한 소프트웨어를 설치 및 구성하는 단계이다. 새로운 장치가 KubEVC-Agent의 통제를 받을 수 있고, 클러스터에서 정상적으로 작동할 수 있도록 소프트웨어 세트를 설치 및 구성하고 커널 관련 설정을 수행하게 된다. KubEVC-Agent의 경우, 코드 형 인프라 구성 관리 도구인 앤서블을 사용하므로 관리 노드에 시큐어-셸 (SSH) 구성도 사전에 필수적이다. 그림 1은 에이전트를 통해 새 클러스터 노드가 가입하는 과정을 나타낸다. 이처럼 각 모듈은 독립적인 역할을 가지면서도 서로 유기적으로 연계되어 작동하도록 구현하였다.

2.1 필요 소프트웨어 설치

첫 번째로, 필요 시스템 소프트웨어 설치가 이루어진다. 엣지 컴퓨팅은 AMD64나 ARM32/64 등 다양한 아키텍처의 호스트들로 클러스터가 이루어질 수 있으므로 필요에 따라 소프트웨어를 자신의 플랫폼에 맞게 인식하여 설치하게 된다. 또한, socat과 conntrack같은 쿠버네티스 클러스터 네트워킹에 필요한 의존성 패키지도 설치하게 된다. 그리고 해당 클러스터 내에서 파일 동기화를 위하여 NFS (Network File System) 클라이언트나 iSCSI 초기자와 같은 패키지를 설치하게 된다. 컨테이너를 실행하기 위한 런타임으로는 쿠버네티스에서 권장하는 OCI (Open Container Initiative)인 containerd를 설치하며, 실제 컨테이너 런타임 (Container Runtime)으로 사용되는 runc도 설치한다. 쿠버네티스는 컨테이너 런타임 인터페이스 (CRI, Container Runtime Interface)를 가지고 있으며, containerd는 CRI의 요청을 받아 이미지를 가져오고 최종적으로 runc를 통해 실제 파드에 사용될 컨테이너를 구동하게 된다. 마지막으로 사용자가 사

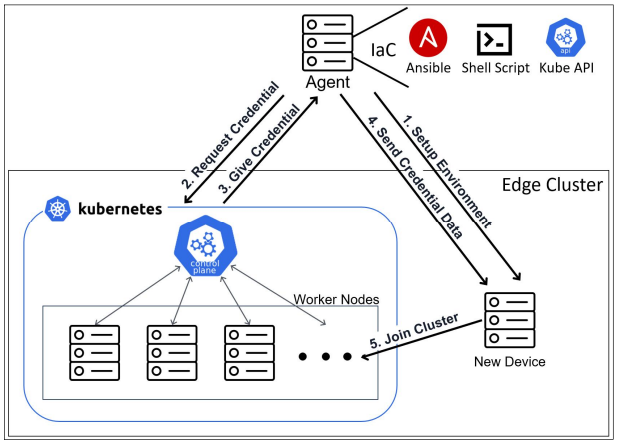

그림 1. 에이전트를 통한 환경 배포 구성도
Fig. 1. Diagram of environment deployment through agent

용하고자 하는 쿠버네티스 배포판을 선택하여 자원이 제한되는 환경에서는 K3S와 같은 경량 쿠버네티스를 통해 클러스터를 구성하는 옵션도 함께 제공하도록 하였다.

2.2 리눅스 시스템 커널 설정

쿠버네티스와 같은 컨테이너 오케스트레이션의 클러스터 노드로서 동작하기 위해서는 다음과 같은 많은 시스템 커널 설정이 요구된다.

첫 번째로 패킷 포워딩을 활성화해 주어야 한다. 쿠버네티스는 추상화된 별도의 클러스터 네트워크를 사용해서 클러스터 내 통신을 하기에 각 노드간 패킷 포워딩이 활성화되어 있지 않다면 정상적인 동작을 기대할 수 없다. 따라서 리눅스 시스템 커널의 속성을 관리하는 도구인 sysctl을 통해 IPv4 및 IPv6 패킷 포워딩을 활성화하는 매개변수를 활성화해 주어야 한다.

두 번째로 브리지 네트워크 인터페이스에서 패킷 필터링을 활성화하기 위해 br_netfilter 모듈을 활성화해 주어야 한다. 쿠버네티스에서 사용하는 가상의 네트워크 어댑터는 실제 호스트의 네트워크 어댑터에 브리지 된 형태로 사용되게 된다. 따라서, 쿠버네티스의 각 노드가 시스템에 설치된 iptables와 같은 방화벽 도구와 상호작용하기 위해서 br_netfilter 모듈의 활성화가 필요하다.

세 번째로 메모리 스왑을 비활성화해 주어야 한다. 일반적인 서버가 아닌 클라이언트로 사용될 수 있는 라즈베리파이와 같은 장비는 기본적으로 OS 차원에서 스왑이 활성화되어 있을 수 있다. 만약 스왑이 활성화되어 있다면 저장장치 일부를 메모리로 사용하게 되면서 컨테이너가 스왑 메모리상에서 동작하게 되어 성능 저하를 유발할 수 있다. 또한, 스왑은 실제 시스템의 메모리 사용량을 초과하여 실행시킬 수 있도록 지원하므로 해당 노드에서 사용할 수 있는 자원 크기의 예측이 어려운 점도 있으며, 스왑을 사용하게 되면 메모리 자원이 파일 시스템을 통해 공유되어 보안 취약점을 유발할 수도 있다.

네 번째로 cgroup (Control group)이 CPU 및 메모리 자원을 관리, 할당할 수 있도록 설정해 주어야 한다. cgroup은

프로세스의 CPU, 메모리, 디스크 입출력 등과 같은 시스템 자원을 제한하고 격리할 수 있는 리눅스 커널의 기능이다. 컨테이너 환경은 격리된 환경의 프로세스를 기반으로 동작하므로 예측할 수 있는 자원을 할당하기 위해서는 프로세스의 자원 사용을 제한할 수 있는 cgroup을 통해 컨테이너에 CPU 코어나 메모리 자원을 할당할 수 있도록 커널에서 활성화를 해주어야 한다. 마찬가지로 컨테이너 런타임인 containerd에서도 cgroup을 통해 자원을 할당할 수 있도록 설정을 해주어야 한다. 이를 통해 예측할 수 있는 컨테이너 성능을 달성할 수 있다.

3. 에이전트를 통한 클러스터 관리

에이전트는 초기 환경 구성이 되어 있는 장치를 대상으로 클러스터를 구성하거나, 기존에 구성된 클러스터에 새로운 장치를 가입시키는 기능을 수행할 수 있다. 또한 여러 지역별로 분산된 클러스터를 관리하기 위해 클러스터 설정 파일을 선택할 수 있는 기능도 제공할 필요가 있다.

3.1 클러스터 구성 모듈

배포하고자 하는 지리적 위치에 엣지 클러스터가 구성되어 있지 않다면, 에이전트를 통해 쿠버네티스 마스터 노드를 지정하여 엣지 클러스터를 구성할 수 있다. 사용자는 클러스터 이름 및 추상화된 클러스터 내부 네트워크에서 사용할 CIDR 대역을 지정해야 한다. 클러스터 구성이 완료되면 컨트롤 플레인의 API 서버는 일반적으로 내부 사설 네트워크에 배치되기 때문에 외부에서 에이전트가 API 서버에 접근하기 위해서는 VPN이나 포트 포워딩과 같은 요소를 고려해야 한다.

3.2 클러스터 노드 가입 모듈

배포된 기존 엣지 클러스터에 추가적인 노드를 클러스터에 가입시키는 경우도 고려해야 한다. 이때 고려할 점은 보안적 요소이다. 사용자가 직접 새로운 노드를 클러스터에 가입시키기 위해서는 컨트롤 플레인 서버에서 가입하기 위한 토큰을 생성하여 수기로 적어서 가입하거나, 네트워크를 통해 토큰을 전송해서 가입하는 방식을 취할 수 있다. 그러나 앞서 설명한 방법은 노드가 많아질수록 관리가 불편해지며 새로운 노드가 클러스터에 가입하기도 전에 컨트롤 플레인 API에 직접 접근하여 토큰값을 가져오는 것은 보안 측면에서 바람직하지 않다. 따라서 새로 가입하고자 하는 노드가 에이전트에 클러스터 가입 요청을 보내고 에이전트에서 중재하여 클러스터 가입이 이루어지도록 하는 것이 바람직하다.

추가 클러스터 노드가 가입할 때는, 그림 1에 나타난 그것 과같이 에이전트가 컨트롤 플레인 API에 가입을 위한 토큰을 요청한다. 토큰 발급 시, 일회성 사용이 목적이므로 TTL (Time-To-Live)을 1분 미만으로 설정하고 발급한다. 컨트롤 플레인 API로부터 발급받은 토큰을 에이전트를 통해 새로 가입하고자 하는 장치에 전달한다. 그러면 새로운 장치는 토큰을 통해 클러스터에 가입할 수 있게 되며 가입이 완료되면 TTL로 인해 토큰이 삭제되거나, 에이전트에서 토큰 삭제를 요청할 수 있다.

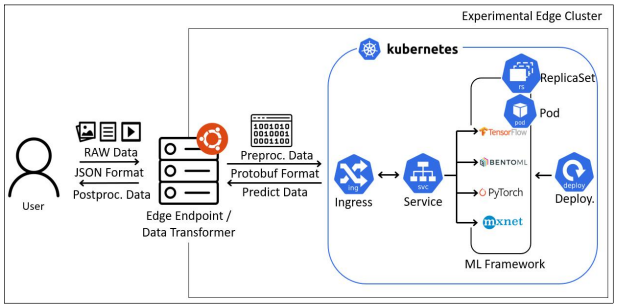

그림 2. 프로토콜 및 데이터 형식 최적화 모듈 다이어그램
Fig. 2. Protocol and data format optimization module diagram

4. 추론 서비스 배포

4.1 추론 서비스 배포 모듈

에이전트를 통해 쿠버네티스 클러스터 API에 요청을 보내는 것으로 클러스터에 추론 서비스를 배포할 수 있다. 추론 서비스 애플리케이션은 텐서플로우 서빙 (Tensorflow Serving)이나 TorchServe, BentoML, MXNet 등 원하는 머신러닝 프레임워크 또는 모델의 종류를 선택하여 배포할 수 있다. 그러나 사용할 애플리케이션의 종류뿐만 아니라 클러스터 내부에서 통신할 때 사용할 프로토콜이나 데이터 형식 또한 고려해야 한다.

범용적으로 사용되는 REST의 경우 사용자가 데이터의 내용을 파악하기 간단하거나 애플리케이션의 호환성 제약 등의 문제 없이 간단하게 사용할 수 있다는 장점이 있지만, 사용자가 데이터의 내용을 파악하기 간단하다는 것은 곧 데이터가 직렬화나 압축 등이 이루어지지 않았음을 뜻한다. 이는 클러스터 내부 통신에 큰 비용과 부하가 발생한다는 것을 의미한다. 따라서 클러스터 내부 통신에서 Protobuf와 같은 애플리케이션마다 특화되는 데이터 형식을 사용함으로써 네트워크 지연시간 감소를 달성할 수 있다. 이러한 네트워크 지연시간 감소는 엣지 컴퓨팅 구성요소에서 중요한 부분이다. 우리가 제안하는 KubEVC-Agent가 구성하는 엣지 추론 클러스터에서는 그림 2와 같이 추론 서비스 구성 시 통신 데이터 형식에 Protobuf를 사용하도록 구현하여 지연시간을 최소화하였다.

그러나, 사용자가 추론 서비스를 이용하기 위해 직접 각 애플리케이션에 최적화된 Protobuf 형식으로 통신을 요청하기에는 접근성이 많이 떨어질 수 있다. 따라서 엣지 클러스터 엔드포인트에서 프로토콜 및 데이터 형식 변환을 수행할 수 있는 일종의 프록시 (Proxy) 모듈을 추가로 구현하였다. 사용자가 텍스트, 이미지, 비디오와 같은 일반적인 데이터를 보내게 되면, 클러스터 엔드포인트에서는 클러스터 내부의 추론 서비스의 모델이 해석할 수 있는 형식으로 데이터 전처리를 수행하고, 추론 서비스에 Protobuf 데이터 형태로 변환하여 전송하면 Protobuf로 응답한 결과를 사용자에게 다시 JSON 데이터 형식으로 변환 후 전송하도록 구현하여 더 낮은 네트워크 지연시간 환경을 구현하였다.

4.2 모델 배포 및 업데이트 구성

엣지 클러스터와 같은 분산 시스템 환경에서는 사용자에게 동일한 서비스를 제공하기 위해 노드별 파일 시스템의 일관성을 유지하는 것이 중요하다. 특히 추론 서비스를 제공할 때는 업데이트된 모델이 얼마나 신속하게 실제 클러스터에 배포되는지가 중요하다. 또한, 모델 크기가 클 때 저장장치의 자원이 한정적인 엣지 컴퓨팅 환경에서 모든 각 분산 노드에 모델을 배포하는 것은 큰 부담이 될 수 있다. 따라서 KubEVC-Agent에서는 별도의 파일 서버가 있다고 가정하고 NFS (Network File System)이나 iSCSI와 같은 네트워크 저장장치 솔루션을 이용할 수 있도록 하였으며, 쿠버네티스의 PV (Persistent Volume), PVC (Persistent Volume Claim)기능을 활용하여 쿠버네티스에서 네트워크 저장장치 솔루션에 대한 접근 정보를 관리하도록 하였다. 따라서 추론 서비스를 제공하는 파드 생성 시, 모델 파일이 위치한 네트워크 저장장치를 가리키는 PV를 통해 접근을 요청하는 PVC 자원을 생성하여 파드가 네트워크 저장장치에 직접 접근할 수 있다. 이를 통해 모델이 업데이트될 경우, 파드에서 네트워크로 직접 접근하여 다시 모델을 불러오도록 구성함으로써 모든 노드에 모델을 분산 업데이트할 필요가 없기에 더 효율적이다.

5. 중앙 집중형 모니터링

엣지 컴퓨팅 클러스터와 같은 분산 시스템에서 모니터링을 통해 장치들을 유지보수 및 감독하는 것도 중요한 문제이다. KubEVC-Agent에서는 프로메테우스와 같은 오픈소스 모니터링 솔루션을 배포하여 엣지 클러스터 장치들을 중앙에서 모니터링 지표를 수집할 수 있도록 구성하였다. 이를 통해 쿠버네티스의 서비스 모니터 (Service Monitor)기능을 활용하여 분산 배치된 파드의 정보 및 해당 파드에 배포된 추론 애플리케이션에 특화된 상태 정보를 확인할 수 있도록 구현하였다. 수집한 모니터링 메트릭을 바탕으로 오픈소스 모니터링 시각화 도구인 그라파나를 사용하여 시각화하였다. 이는 분산 배치된 환경의 자원들을 한눈에 확인할 수 있게 도와준다.

Ⅳ. 실험 및 구현 데모

1. 실험 환경

실험 환경은 지역별로 분산된 엣지 클러스터 중 하나를 구성하는 것으로 가정하고, 성능 및 구현 기능을 평가하는 데에 중점을 두었다. 라우터 (Router) 및 엣지 엔드포인트 기능을 수행하는 프록시, 네트워크 저장장치 솔루션, 에이전트 역할을 수행하는 장치는 표 1에 나타나 있는 장치대로 일반적인 x86 시스템의 우분투 (Ubuntu) 운영체제를 실행하도록 구성하였다. 또한, 실제로 워크로드를 처리할 엣지 클러스터 장치들은 임베디드, IoT 등의 시스템에서 널리 사용되는 라즈베리파이 (Raspberry Pi) 4B 8GB 모델을 사용하였다. 네트워크 속도는 기가비트 네트워크 환경에서 실험을

표 1. 실험에 사용된 장치 정보
Table 1. Device Information used in the experiment

Component	Spec
Agent / Edge Endpoint	Processor: 11th Gen Intel (R) Core (RM) i5-1135G7 Memory: LPDDR4 16GB Storage: SATA3 SSD, RW Speed: 560, 530MB/s OS: Ubuntu 20.04LTS (non-graphic)
Cluster Node	Model : Raspberry Pi 4 Model B Processor: Quad-core Cortex-A72 (ARM v8) 64-bit Memory: LPDDR4 8GB Storage: microSDXC (UHS-I), RW Speed: 190, 130MB/s OS: Raspberry Pi OS Lite (64-bit)
Ethernet Speed	Gigabit Ethernet (1Gbps)
Container Runtime	containerd: v1.7.2 runc: v1.1.9
General Kubernetes	bootstrap tools - kubeadm: v1.27.3 node agent - kubelet: v0.15.1
Lightweight Kubernetes	k3s: v1.27.3+k3s

표 2. 실험에 사용된 모델 정보
Table 2. Model Information used in the experiment

	MobilenetV1	InceptionV3	YOLOV5	BERT
Input Size	224x224	299x299	640x640	500
Model Size	18MB	97MB	28MB	428MB
Dataset	ImageNet	ImageNet	COCO	IMDB
GFLOPS	1.15	11.5	16.5	13.39

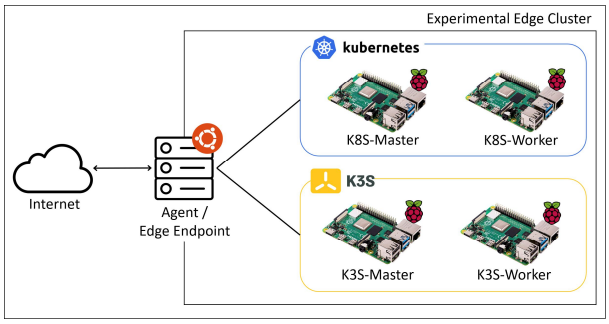

그림 3. 실험 환경 다이어그램
Fig. 3. Experiment environment diagram

진행하였으며, 추론 애플리케이션은 텐서플로우 (TensorFlow) 기반의 텐서플로우 서빙을 통해 구성하였다 [13]. 또한 엣지 컴퓨팅 환경에서 일반적인 쿠버네티스 클러스터와 경량 쿠버네티스 클러스터 두 가지 경우를 모두 구성하여 쿠버네티스 배포판별 기본 자원 사용량에 따른 영향을 분석하였다. 추론 서비스에 배포할 모델로는 표 2에 나타나 있는 것처럼 MobilenetV1 [14], InceptionV3 [15], YoloV5 [16], BERT [17]를 사용하였으며 네트워크 저장장치 솔루션은 NFS를 통해 구성하였다. 실험 환경에 대한 다이어그램은 그림 3에서 확인할 수 있다.

표 3. 자동화 배포 모듈별 소요 시간
Table 3. Time per automated deployment module

	General Kubernetes	Lightweight Kubernetes
Env Setup	4m 12sec	1m 56sec
Init. Cluster	5m 23sec	42sec
Join Cluster	16sec	27sec
Deploy Endpoint	28sec	1m 8sec
Deploy ML Framework	22sec	5sec
Deploy Monitoring	1m 6sec	41sec
Total	11m 47sec	4m 59sec

2. 코드 형 인프라 도구를 통한 자동화 구성 소요 시간

표 3는 에이전트를 통해 아무것도 구성되지 않은 장치들을 이용하여 엣지 컴퓨팅 환경에서 머신러닝 추론 서비스를 배포할 때 기능별로 실행이 완료되는데 소요 시간을 표현하였으며, 실제로 쿠버네티스 클러스터 환경에 배포 요청 후 배포가 완료된 시간까지 표현하였다. 실험 결과를 보았을 때 코드 형 인프라 도구를 활용하여 환경을 구축하는 것은 기존 사용자가 수작업으로 노드를 직접 설정하는 것에 비해 매우 효율적인 것을 확인할 수 있다. 구성 내용이 코드로 모두 작성되어 있기에 추가적인 환경 배포 시 일관성을 보장하고 유지보수를 더욱 간편하게 해주는 것도 확인할 수 있었다. 경량 쿠버네티스를 구성했을 때 소요 시간 또한 측정하였다. 경량 쿠버네티스는 일반적인 쿠버네티스에 비해 모든 바이너리가 하나의 바이너리 파일로 통합되고 바이너리 용량이 100MB 밑으로 축소되었으며 기본적으로 사용되는 쿠버네티스의 기능이 적은 만큼 총소요 시간이 더 적게 소요되는 것을 확인할 수 있었다. 그러나 워커 노드 클러스터 가입과 로드 밸런서와 인그레스 컨트롤러와 같은 엔드포인트의 구성요소를 배포할 때는 더 많은 시간이 소요된 것을 확인할 수 있다. K3S의 경우에는 바이너리 설치와 동시에 클러스터 구성 혹은 클러스터 가입을 진행하기에 환경 구성에서 모든 바이너리가 설치되는 일반 쿠버네티스에 비해 시간이 추가로 소요되었으며, 엔드포인트 배포 시 K3S는 기본적으로 kube-proxy와 같은 프록시 서비스가 탑재되어 있지 않아 추가적인 시간이 소요되었다.

그림 4는 실제 배포 작업 진행 중 나타나는 코드 실행 모습을 나타낸다. 앤서블을 이용하여 구성하였기에 각 노드에서 무슨 작업이 진행되고 있는지 직관적으로 확인할 수 있으며, 실패 시 로그를 상세히 확인할 수 있다.

그림 5는 배포 완료 후 쿠버네티스 클러스터 노드 가입 정보 및 머신러닝 추론 애플리케이션인 텐서플로우 서빙이 정상적으로 배포되어 작동하는 모습을 보여준다.

3. 쿠버네티스 배포판에 따른 자원 사용량 차이

경량화된 쿠버네티스는 기본적으로 생성되는 파드의 개수도 적은 것을 확인할 수 있었다. 또한, 배포판별 쿠버네티스 기본 구성요소의 CPU와 메모리 사용량에 대해 측정하였을

```
kubevc@kubevc-agent:~$ ./2.2_k3s_worker_node_join.sh

PLAY [Create token for worker node] ********************************

TASK [Get k3s master token] ****************************************
changed: [master]

TASK [Set Global Variable] *****************************************
ok: [master]

TASK [Get private IP address of Master node] ***********************
changed: [master]

TASK [ansible.builtin.set_fact] ************************************
ok: [master]

PLAY [Join the worker] *********************************************

TASK [Download k3s worker cluster script] **************************
ok: [worker1]

TASK [Install k3s worker cluster] **********************************
changed: [worker1]

PLAY RECAP *********************************************************
master                     : ok=4    changed=2    unreachable=0
worker1                    : ok=2    changed=1    unreachable=0
```

그림 4. 워커 노드 가입 자동화 모듈 실행 모습
Fig. 4. Executing the worker node join module

```
kubevc@kubevc-agent:~$ kubectl get nodes --kubeconfig=$HOME/.kube/k8s
NAME         STATUS   ROLES           AGE     VERSION
k8s-master   Ready    control-plane   7d20h   v1.27.3
k8s-worker   Ready    <none>          7d19h   v1.27.3
kubevc@kubevc-agent:~$ kubectl get deployment -A --kubeconfig=$HOME/.kube/k8s
NAMESPACE        NAME                              READY   UP-TO-DATE   AVAILABLE   AGE
bert-imdb        tensorflow-serving-deployment     1/1     1            1           72m
ingress-nginx    ingress-nginx-controller          1/1     1            1           171m
kube-system      calico-kube-controllers           1/1     1            1           7d20h
kube-system      coredns                           2/2     2            2           7d20h
metallb-system   controller                        1/1     1            1           171m
```

그림 5. 배포 완료 후 노드 및 애플리케이션 구동 확인
Fig. 5. Verifying node and application post-deployment

표 4. 쿠버네티스 플랫폼별 리소스 사용량
Table 4. Resource usage by kubernetes platform

	General Kubernetes		Lightweight Kubernetes	
	CPU (%)	RAM (MB)	CPU (%)	RAM (MB)
Master Node	6.8	1054.6	2.2	777.3
Worker Node	1.6	106.8	0.9	155.6

때, 표 4와 같이 나타났다. 일반 쿠버네티스의 마스터 노드에서는 API 서버, 스케줄러, Etcd, 컨트롤러 매니저, kube-proxy, kubelet를 확인하였고, 일반 쿠버네티스의 워커 노드에서는 kubelet, kube-proxy의 사용량을 확인하였다. 경량 쿠버네티스의 경우 하나의 바이너리로만 작동하기에 K3S 프로세스에 대한 사용률을 확인하였다. 마스터 노드에서는 경량 쿠버네티스가 CPU와 메모리에 대해서 CPU는 약 50%, 메모리는 약 30% 정도 자원이 덜 소모되는 것을 확인

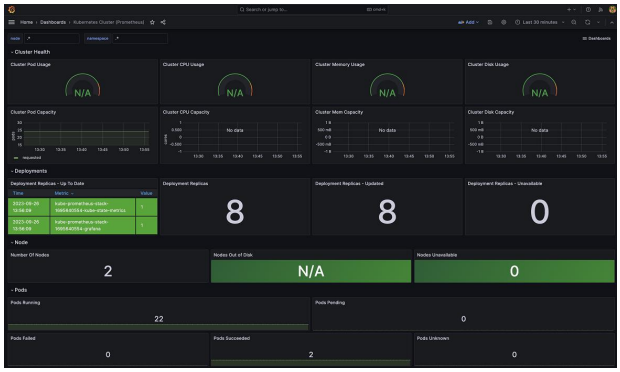

그림 6. 그라파나를 통한 클러스터 지표 시각화 예시
Fig. 6. Example of cluster metric visualization through Grafana

할 수 있었다. 워커 노드에서는 일반 쿠버네티스와 경량 쿠버네티스 간 큰 차이를 보이지 않았다. 그러나 경량 쿠버네티스는 단순화된 시스템 구조를 하고 있어 새로운 애플리케이션을 배포할 때도 최소로 필요한 것만 사용되기 때문에 경량 쿠버네티스를 사용하는 것이 유리할 수 있다. 이처럼 자원이 제한되는 상황에서는 경량 쿠버네티스 클러스터를 구성하는 것이 더 좋은 선택일 수도 있다는 것을 보여준다.

4. 중앙 집중화된 모니터링 구성

엣지 컴퓨팅과 같은 분산 시스템 환경은 자원 사용률 확인이나 추후 유지보수를 위한 데이터 수집을 개별적으로 수행할 때 불편할 수 있다. KubEVC-Agent는 오픈소스 모니터링 솔루션인 그라파나와 프로메테우스를 사용하여 분산된 노드의 각 시스템 정보나 특정 애플리케이션에 특화된 메트릭을 수집할 수 있도록 구현하였다. 그림 6은 프로메테우스를 통해 수집된 지표가 그라파나를 통해 시각화되어 중앙에서 클러스터 자원 정보가 한눈에 보여준다. 사용자는 이를 이용하여 효율적으로 모니터링을 수행할 수 있을 것이다.

5. 프로토콜 및 데이터 형식 최적화 모듈 개발에 따른 성능 향상

엣지 컴퓨팅은 네트워크 지연시간에 치명적인 만큼 클러스터 내부 통신에서 지연시간을 줄이는 것도 중요하다는 것을 이전에 언급하였다. 따라서 범용적인 프로토콜인 REST를 그대로 사용하는 것 대신에 본 논문에서 제안하는 프로토콜 및 데이터 형식 최적화 모듈을 사용하였을 때 지연시간 관점에서 얻을 수 있는 이점에 대해 실험을 진행하였다. 실험은 4개의 모델에 대해서 기존에 사용되던 일반적인 방법과 본 논문에서 제안하는 방법으로 각 20회 요청을 보내었다. 이 과정에서 추론 시간이 포함된 네트워크 지연시간을 기록하여 그림 7과 같이 박스 플롯 그래프로 표현하였다. 그림 7에서 보다시피, 본 논문에서 제안한 방법을 사용하여 구성하였을 때 추론 시간, 네트워크 지연시간 등을 포함한 결과에서 많은 성능 향상 폭을 보이는 것을 확인할 수 있다. JSON의 경우 모든 데이터를 문자열로 변환하여 저장하므로 원본 데이터의 타입과 상관없이 표시되는 길이에 비례하여 데이터의 크기가 결정된다. 유리수를 이진수로 근사하여 저장하는 부동소수점 데이터 특성상, 실제 유리수 값을 정확히 저장할 수 있는 경우가 드물어 저장된 데이터에 따라 표시되는 길이가 많이 늘어날 수 있다. 이러한 특성은 부동소수점 데이터들로 이루어진 행렬을 JSON으로 전송하려는 상황에서 불리하게 작용할 수 있다. 본 논문에서 제안하는 모듈의 데이터 변환 형식인 Protobuf의 경우 전송하고자 하는 데이터의 자료형과 상관없이 이진 데이터를 직렬화하여 전송하므로 기존 데이터의 대비 크기의 변화가 거의 발생하지 않는다. 이는 YoloV5와 같은 전처리 데이터 및 예측 데이터의 용량이 큰 모델에 더욱더 효과적이다. 본 실험에서는 텐서플로우 서빙이 쿠버네티스를 통해 여러 파드로 배치되어 기술적인 이유로 추론 시간과 네트워크 지연시간을 분리하여 평가할 수 없었다. 그러나 데이터 형식 최적화에 따라서 순수 추론 시간이 차이가 나지 않는 것을 데이터 크기가 미미한 BERT 모델의 실험 결과에서 확인할 수 있었다. 이를 통해 각 모델에서 기존 데이터 형식을 사용했을 때와 비교하여 본 논문에서 제안하는 데이터 형식 최적화 모듈을 토해 추론 시 클러스터 내부 통신 시간이 단축됨을 확인할 수 있었다. 송수신 데이터 크기가 큰 YoloV5 모델에서 본 논문에서 제안하는 모듈을 사용할 때 송수신 데이터 크기가 기존 방식에 비해 85.4% 감소하였으며 전송시간에서는 83.6% 감소하였다. BERT 모델에서는 외려 송수신 데이터 크기가 기존 방식에 비해 48.7% 증가하였다. 그러나 BERT와 같은 자연어 처리 (NLP) 모델에서는 상대적으로 데이터 크기가 작은 텍스트를 입력으로 사용하기 때문에 실제 송수신 데이

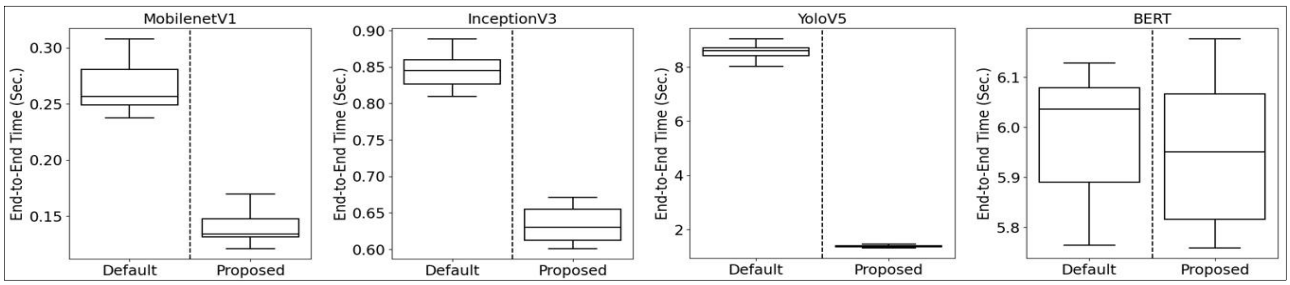

그림 7. 각 모델에서 통신 프로토콜 별 지연시간
Fig. 7. End-to-End Latency by protocol in each model

표 5. 데이터 형식에 따른 모델별 입출력 크기 (MB)
Table 5. Input/Output size (MB) for each model according to data format

	MobilenetV1		InceptionV3		YOLOV5		BERT	
Format	JSON	PB	JSON	PB	JSON	PB	JSON	PB
Input	3.014	0.574	5.524	1.023	24.547	4.688	0.004	0.006
Output	0.026	0.004	0.026	0.004	63.593	8.171	0.0001	0.0001

터 크기의 차이는 0.002MB 정도로 미미한 것을 표 5에서 확인할 수 있다. 전송시간에서는 0.007% 감소하여 오차범위 내 동일하다고 평가할 수 있는 수치를 보였다. 이러한 결과는 실제 추론 시간에 비해 추론 데이터 전송 시간이 실제 사용자가 체감하는 추론 시간에 많은 영향을 끼치는 것을 보여주었으며, 본 논문에서 제안한 방법으로 이를 크게 단축할 수 있음을 보여준다.

V. 결 론

엣지 컴퓨팅 환경에서 사용할 수 있는 머신러닝 가속기의 다양성 및 엣지 장치의 성능이 나날이 증가하면서 사용자에게 엣지 클러스터를 통해 초저지연 머신러닝 추론 서비스를 제공하는 것이 점점 주목받고 있다. 엣지 컴퓨팅 환경은 클라우드 컴퓨팅 환경에 비해 자원에 제약이 있기에 효율적인 클러스터를 구성하는 방법이 중요하다. 엣지 컴퓨팅이나 클라우드 컴퓨팅 환경처럼 분산 배치된 클러스터와 같은 분산 시스템을 오케스트레이션 하기 위한 도구인 쿠버네티스의 중요성 또한 높아지고 있다. 본 논문에서 제안하는 KubEVC-Agent를 통해 머신러닝 추론 엣지 클러스터 구성을 자동화하여 기술 사용에 대한 진입 장벽 없이 손쉽게 엣지 클러스터를 배포할 수 있도록 하였다. 본 논문에서 제안하는 프로토콜 및 데이터 형식 최적화 모듈을 사용하는 것으로 클러스터 내 통신을 최적화하여 모범 사례를 제시하였다. 이를 통해 머신러닝 추론을 진행할 때, YOLOV5와 같이 데이터의 크기가 큰 모델의 경우 약 75%의 성능 향상이 있는 것을 확인할 수 있었다.

향후 연구에서는 GPU, TPU와 같은 다양한 가속기가 장착된 장치들로 이루어진 엣지 컴퓨팅 클러스터를 효율적으로 구성할 수 있도록 하고, 사용하고자 하는 모델에 따라서 통신 프로토콜을 최적화하도록 할 필요가 있다. 또한, 구성된 엣지 클러스터 내에서 사용자가 요청하는 모델에 따라 적합한 가속기가 있는 장치로 스케줄링하는 기법을 연구할 예정이다.

References

[1] H. Hua, Y. Li, T. Wang, N. Dong, W. Li, J. Cao, “Edge Computing with Artificial Intelligence: A Machine Learning Perspective,” ACM Comput. Surv. Vol. 55, No. 9, pp. 1-35 2023.
[2] J. Chen, X. Ran, “Deep Learning With Edge Computing: A Review,” IEEE, Vol. 107, No. 8, pp. 1655-1674, 2019.
[3] A. Marchisio, M. A. Hanif, F. Khalid, G. Plastiras, C. Kyrkou, T. Theocharides, M. Shafique, "Deep Learning for Edge Computing: Current Trends, Cross-Layer Optimizations, and Open Research Challenges," IEEE, pp. 553-559, 2019.
[4] https://kubernetes.io/
[5] https://www.ansible.com/
[6] M. Artac, T. Borovssak, E. D. Nitto, M. Guerriero, D. A. Tamburri, "DevOps: Introducing Infrastructure-as-Code," IEEE, pp. 497-498, 2017.
[7] https://k3s.io/
[8] https://grpc.io/
[9] S. Popić, D. Pezer, B. Mrazovac, N. Teslić, “Performance Evaluation of Using Protocol Buffers in the Internet of Things Communication,” IEEE, pp. 261-265, 2016.
[10] https://prometheus.io/
[11] https://grafana.com/
[12] http://kubevc.ddps.cloud/
[13] https://www.tensorflow.org/
[14] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, H. Adam, "Mobilenets: Efficient Convolutional Neural Networks for Mobile Vision Applications," arXiv Preprint arXiv:1704.04861, pp. 1-9, 2017.
[15] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, Z. Wojna, "Rethinking the Inception Architecture for Computer Vision," In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2818-2826, 2016.
[16] J. Redmon, S. Divvala, R. Girshick, A. Farhadi. "You Only Look Once: Unified, Real-time Object Detection," In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 779-788, 2016.
[17] J. Devlin, M. W. Chang, K. Lee, K. Toutanova, "Bert: Pre-training of Deep Bidirectional Transformers for Language Understanding" arXiv Preprint arXiv:1810.04805, pp. 1-16, 2018.

Moohyun Song (송 무 현)

2020~Computer Science from Kookmin University (B.S.)
2023~Undergraduate Student Researcher at Distributed Data Processing System Lab. from Kookmin University

Field of Interests: Cloud Computing, Edge Computing
Email: mhsong@kookmin.ac.kr

Kyumin Kim (김 규 민)

2019~Computer Science from Kookmin University (B.S.)
2023~Undergraduate Student Researcher at Distributed Data Processing System Lab. from Kookmin University

Field of Interests: Bigdata Processing, Distributed Computing, Cloud Computing, Container Ochestration
Email: okkimok123@kookmin.ac.kr

Jihun Moon (문 지 훈)

2021~Computer Science from Kookmin University (B.S.)
2023~Undergraduate Student Researcher at Distributed Data Processing System Lab. from Kookmin University

Field of Interests: Edge Computing, AI Platform, Container Ochestration
Email: answlgns2056@kookmin.ac.kr

Yurim Kim (김 유 림)

2020~Computer Science from Kookmin University (B.S.)
2023~Undergraduate Student Researcher at Distributed Data Processing System Lab. from Kookmin University

Field of Interests: Container Technology, AI Application
Email: belbet01@kookmin.ac.kr

Chaewon Nam (남 채 원)

2020~Computer Science from Kookmin University (B.S.)
2023~Undergraduate Student Researcher at Distributed Data Processing System Lab. from Kookmin University

Field of Interests: Cloud Computing, Cost Optimization
Email: rabbit7653@kookmin.ac.kr

Jongbin Park (박 종 빈)

2011 Electronics Engineering from Sungkyunkwan University (Ph.D.)
2012~Principal Researcher, Information Media Research Center, Communications & Media R&D Division, Korea Electronics Technology Institute (KETI)

Career:
2008 Invited Researcher. Canada Research Center Canada (CRC)
2023 Professorial Lecturer. School of Global Studies. Hallym University

Field of Interests: Multimodal Data/Signal Embedding, Computational Imaging, Distributed/Quantum/Embedded Computing
Email: jpark@keti.re.kr

Kyungyong Lee (이 경 용)

2004 Electrical and Computer Engineering from Sungkyunkwan University (B.S.)
2014 Electrical and Computer Engineering from University of Florida (Ph.D.)
2016~Computer Science from Kookmin University (Assoc. Prof.)

Career:
2004~2008 Samsung Electronics. Suwon. Korea
2012~2014 HP Labs. Palo Alto. USA
2014~2016 Amazon Web Services. Seattle. USA

Field of Interests: Cloud Computing, Distributed Computing, Middleware Software
Email: leeky@kookmin.ac.kr