JKSCI

한국컴퓨터정보학회논문지
Journal of The Korea Society of Computer and Information
Vol. 29 No. 3, pp. 107-116, March 2024
https://doi.org/10.9708/jksci.2024.29.03.107

Performance Analysis to Evaluate the Suitability of MicroVM with AI Applications for Edge Computing

Yunha Choi*, Byungchul Tak**

*Student, School of Computer Science and Engineering, Kyungpook National University, Daegu, Korea
**Professor, Dept. of Computer Science and Engineering, Kyungpook National University, Daegu, Korea

[Abstract]

In this paper, we analyze the performance of MicroVM when running AI applications on an edge computing environment and whether it can replace current container technology and traditional virtual machines. To achieve this, we set up Docker container, Firecracker MicroVM and KVM virtual machine environments on a Raspberry Pi 4 and executed representative AI applications in each environment. We analyze the inference time, total CPU usage and trends over time and file I/O performance on each environment. The results show that there is no significant performance difference between MicroVM and container when running AI applications. Moreover, on average, a stable inference time over multiple trials was observed on MicroVM. Therefore, we can confirm that executing AI applications using MicroVM instead of container or heavy-weight virtual machine is suitable for an edge computing.

▸ **Key words**: IoT, MicroVM, Firecracker, Edge Computing, AI

[요 약]

본 논문에서는 엣지 컴퓨팅 환경에서 MicroVM의 AI 애플리케이션 수행 시 성능을 분석하고, 이것이 현재 사용되고 있는 컨테이너 기술과 전통적인 가상머신을 대체할 수 있는지 알아본다. 이를 위해 라즈베리파이 4에서 Docker 컨테이너, Firecracker MicroVM, KVM 가상머신 환경을 각각 구축하고 대표적인 AI 애플리케이션들을 실행하였다. 그리고 실험 환경별로 추론 시간, 총 CPU 사용량 및 추세, 파일 I/O 성능을 분석하였다. 실험 결과, MicroVM에서 AI 애플리케이션을 수행하였을 때 컨테이너와의 큰 성능 차이는 없었으며, 오히려 반복적인 애플리케이션 수행에서 평균적으로 안정적인 추론 시간을 확인할 수 있었다. 따라서, 본 연구를 통해 엣지 컴퓨팅 환경에서 컨테이너와 가상머신을 대체하여 MicroVM을 사용한 AI 애플리케이션 운용이 적합할 수 있다는 것을 확인하였다.

▸ **주제어**: 사물인터넷, MicroVM, Firecracker, 엣지 컴퓨팅, 인공지능

• First Author: Yunha Choi, Corresponding Author: Byungchul Tak
*Yunha Choi (choiyhking@knu.ac.kr), School of Computer Science and Engineering, Kyungpook National University
**Byungchul Tak (bctak@knu.ac.kr), Dept. of Computer Science and Engineering, Kyungpook National University
• Received: 2024. 01. 05, Revised: 2024. 02. 28, Accepted: 2024. 02. 28.

http://www.ksci.re.kr pISSN:1598-849X | eISSN:2383-9945

I. Introduction

Internet of Things(IoT)는 최근 IT 산업에서 가장 주목받고 있는 키워드 중 하나이다. IoT는 우리 주변의 모든 사물이 인터넷에 연결되어, 표준화된 프로토콜과 통신 기능이 내장된 센서를 통해 서로 정보를 교환하고 통신할 수 있는 네트워크를 의미한다 [1]. IoT 디바이스의 예시로는 스마트 전구, 핸드폰, 스마트 워치뿐만 아니라 스마트 시티 등 다양한 형태와 크기로 우리의 일상생활 곳곳에 존재한다. 2023년을 기준으로 약 150억 개가 넘는 IoT 디바이스들이 전 세계에 설치되어 있다 [2]. 최근 COVID-19 팬데믹을 겪으면서 스마트 홈 디바이스와 개인용 헬스케어 장비가 늘어나고 VR/AR/MR 디바이스, 자율주행, 스마트 농업/스마트 팩토리 등 사회 전반의 모든 산업에서 IoT 디바이스의 수요가 계속 늘어나고 있기에 그 수는 기하급수적으로 계속 늘어날 전망이다 [3]. 또한, AI 기술과 5G, LPWAN(Low Power Wide Area Network) 등 다양한 네트워크 기술, IoT 디바이스의 하드웨어 사양이 급격히 발전함에 따라 AI와 IoT의 결합이 주목받고 있다. 많은 기업에서 개발하고 있는 AI 솔루션에 가장 중요한 것은 좋은 품질의 데이터인데, IoT 디바이스의 센서를 통해 얻을 수 있는 대용량의 데이터가 AI 모델을 학습, 테스트하는 것에 큰 역할을 할 수 있다. 하지만 다양한 센서에서 발생하는 엄청난 양의 데이터를 멀리 떨어져 있는 중앙 클라우드 서버에서 모두 처리하게 되면 시스템의 응답 시간이 느려지는 등의 문제가 발생한다. 그러므로 Edge/Fog 컴퓨팅을 통해, 사용자 또는 데이터 발생지와 가까운 곳에서 데이터를 처리하여 부하를 분산시키고 레이턴시를 낮추는 방법이 고려된다.

IoT 디바이스들은 일반적인 컴퓨팅 환경과 달리, 한정된 자원(CPU, 메모리, 디스크, 전력 등)을 가진다. 그러므로 IoT 디바이스에서 애플리케이션을 배포, 운영할 때는 고려해야 할 사항이 많다. 지금까지의 Edge/Fog 컴퓨팅에서는 컨테이너 가상화 기술에 의존해 왔다. 하지만 컨테이너 관리 플랫폼으로 인한 복잡도 증가, 호스트 커널을 공유하기 때문에 발생할 수 있는 보안 취약점 등의 단점이 존재한다. 그리하여 이러한 문제점들을 해결하기 위해 최근 AWS Firecrakcer[4], Kata Containers[5]와 같은 microVM(Light-weight Virtual Machine) 기술이 개발되었다. 따라서 본 연구에서는 microVM이 컨테이너와 전통적인 가상머신 기술을 대체하여 AI 애플리케이션을 동작시키기에 적절한 가상화 환경인지를 알아보고자 한다. 이를 위해 가장 많이 사용되는 엣지 디바이스 중 하나인 라즈베리파이 4를 사용하여 microVM에서 AI 애플리케이션을 수행하였을 때, 다른 환경들과의 성능을 비교하고 어떤 차이를 보이는지 다양한 측면에서 분석하였다. 라즈베리파이는 엔비디아의 GPU가 장착되어 있지 않아 딥러닝 애플리케이션을 수행할 때 병렬처리 능력, CUDA Toolkit 그리고 cuDNN 등의 고성능 라이브러리를 사용하지 못하기 때문에 딥러닝 모델 학습 및 추론 시에 성능 제약이 발생한다. 하지만 라즈베리파이의 저렴한 가격과 높은 범용성 그리고 로보틱스, 헬스케어, 이미지/비디오 영상 처리 등의 AI 분야에서 충분히 활용될 수 있으므로 라즈베리파이를 활용한 다양한 환경에서 성능 분석이 필요하다 [6].

본 논문은 2장에서 실험에 사용된 여러 가상화 환경의 이론적 개념, 벤치마크 그리고 선행 연구를 소개한다. 3장에서는 실험 환경의 자세한 세부 사항 및 방법에 대한 설명을, 4장에서는 실험 결과 제시 및 분석을 진행한다. 마지막으로 5장에서는 본 논문의 결론과 한계, 추후 연구 방향을 제시한다.

II. Preliminaries

1. Background

1.1 KVM

KVM(Kernel-based Virtual Machine)은 리눅스 커널에 포함된 모듈로서, CPU가 Intel VT 또는 AMD-V와 같은 가상 CPU 기능을 지원할 때 해당 리눅스 운영체제를 하이퍼바이저(Hypervisor) 역할을 하도록 바꿔준다. 하이퍼바이저 역할을 맡게 된 호스트 시스템은 여러 개의 게스트 가상머신을 생성할 수 있고, 각각의 게스트 머신들에게 독립된 가상 환경을 제공한다. 하드웨어 리소스 에뮬레이터인 QEMU(Quick Emulator)는 주로 KVM과 함께 사용되어 디스크, 네트워크, USB 등의 하드웨어 에뮬레이션을 담당한다.

1.2 Docker

컨테이너 기술은 전통적인 가상머신과 비교했을 때 가볍다는 특징이 있다. 가상머신 환경에서는 호스트 시스템 상에서 동작하는 다수의 게스트 머신이 각각 별도의 운영체제를 가져야 하고, 이들에게 실제 하드웨어가 장착된 것처럼 만들어 주기 위해 하드웨어 리소스 에뮬레이션 과정이 필요하다. 하지만 컨테이너는 호스트 시스템의 커널을 공유한다. 그러므로 독립된 프로세스를 가질 수 있게 해주는 리눅스 namespaces와 CPU, 메모리, 네트워크 등의

자원 제어를 가능하게 해주는 cgroups 기능을 사용하여 완전한 가상화를 하지 않기 때문에 전통적인 가상머신에 비해 비교적 가볍다. 또한, 컨테이너는 배포 단계에서 많은 장점이 있다. 배포하고자 하는 애플리케이션을 동작시키는 데 필요한 다양한 소프트웨어 라이브러리, 런타임, 환경변수 설정 그리고 실행 명령 등을 포함하여 배포할 수 있으므로, 효율적인 배포 및 관리를 할 수 있다. Docker는 이러한 컨테이너 환경을 조금 더 편하게 사용할 수 있도록 다양한 기능을 제공하는 플랫폼이다. 본 연구에서 지칭하는 Docker 컨테이너는 Docker에서 runc 컨테이너 런타임을 사용하여 생성하는 컨테이너를 의미한다.

1.3 Firecracker

기존의 컨테이너 가상화는 1.2절에서 언급한 장점들 때문에 많이 사용되었다. 하지만 컨테이너는 호스트 시스템의 커널을 공유한다는 점에서 잠정적 보안 위험이 존재한다. 그리하여 컨테이너의 가볍다는 장점과 기존 가상머신의 안정성, 보안성을 함께 가진 microVM 기술이 등장하였다. AWS Firecracker는 다양한 microVM 프로젝트 중 하나이다. Firecracker는 전통적인 가상머신에서 사용되던 KVM/QEMU 구조에서 KVM은 그대로 유지하고, QEMU를 microVM의 설정 및 관리를 위해 새롭게 제작된 VMM(Virtual Machine Monitor), 디바이스 모델, API로 대체하였다. Firecracker는 호스트마다 초당 150개의 microVM을 생성할 수 있고, 부팅 시간은 125ms 이하로 매우 빠르다. 보안성이 좋은 Rust 프로그래밍 언어를 사용하였으며, 게스트 머신의 공격을 최소화하기 위해 단순한 디바이스 모델을 제공한다. Firecracker는 서버리스 워크로드를 처리하기 위해 개발되었으며, AWS Lambda와 Fargate 및 다양한 서비스에서 활용되고 있다 [4].

1.4 Edge AIBench

Edge AIBench는 IoT 엣지-클라우드 시스템을 테스트하기 위해 개발된 벤치마크이다. 특히, Edge AIBench 2.0은 자율주행 시나리오를 기반으로 제작되었다 [7]. 실제 자율주행의 전체 시나리오는 매우 복잡하지만, 수많은 모듈 중에서 중요한 것은 Object Detection과 Image Classification이다. 또한, 원래 클라우드 서버에서 이루어지는 딥러닝 모델 학습과 데이터 전처리처럼 짧은 시간 내에 완료되는 모듈들은 벤치마크에서 제외하였다. 결과적으로 Edge AIBench 2.0에서는 미리 학습된 딥러닝 모델들을 사용하고 Object Detection, Traffic Light Classification, Road Sign Classification, 그리고 Lane Keeping 이 4개의 대표적인 벤치마크 애플리케이션만 제공한다.

2. Related Works

1장에서 설명한 것처럼 Edge/Fog 컴퓨팅 환경에서 AI와의 결합을 통한 다양한 연구가 수행되고 있다.

Hua et al. [8]에서는 AI와 엣지 컴퓨팅의 결합이 활발하게 이루어지는 이유 두 가지를 제시하였다. 첫째, AI 기반 솔루션을 활용하여 엣지 컴퓨팅에서 발생하는 작업 스케줄링, 자원 할당, 보안 관련 문제 등을 해결하려고 한다. 둘째, 스마트 시티, 스마트 제조업, IoV(Internet of Vehicle) 등에서 사용되는 AI 애플리케이션을 엣지 노드에서 수행하여, 네트워크 지연과 데이터 프라이버시 문제를 해결하려고 한다. 이처럼, 엣지 컴퓨팅 구조에서 딥러닝 모델의 추론은 엣지 노드에서 진행되는 경우가 많다. 그러므로 엣지 디바이스에서의 딥러닝 모델 추론 성능은 매우 중요한 지표이며 심층적인 연구가 필요하다.

주로 사용하는 엣지 디바이스에는 라즈베리파이, 엔비디아 Jetson Nano 그리고 구글 Coral Dev Board 등이 있다. 1장에서 언급한 바와 같이 라즈베리파이는 몇 가지 단점에도 불구하고, 저렴한 가격과 높은 범용성을 가지며 다양한 AI 분야에서 충분히 활용될 수 있다. Villanueva et al. [9]에서는 운전자 졸음 탐지를 위해 라즈베리파이를 사용하였다. 카메라를 통해 촬영된 이미지와 SqueezeNet을 사용하여 얼굴 특징의 패턴을 파악하고, 운전자에게 알람을 주는 시스템을 제안하였다. Lavanya et al. [10]에서는 자체 센서를 활용하여, 측정된 토양 데이터를 라즈베리파이 상의 AI 애플리케이션에서 처리한다. 토양의 부족한 영양 정보를 파악한 후, 농부들에게 정보를 제공하여 곡물 수확량을 높이는 시스템을 제안하였다. 이러한 사례들을 통해, 라즈베리파이가 엣지 컴퓨팅 환경에서 가장 우수한 성능을 보이는 것은 아니지만 다양한 방면으로 활용되는 것을 확인할 수 있다.

또한, 컨테이너와 가상머신이 이미 많이 사용되고 있음에도 불구하고 각 환경의 단점들 때문에 microVM 기술이 등장하였다. 이에 microVM이 기존의 가상화 환경과 비교하여 현업 비즈니스에 도입될 만큼의 성능을 낼 수 있는지에 대한 다양한 연구가 이루어졌다.

Anjali et al. [11]에서는 일반적인 고성능 컴퓨팅 환경에서 native, LXC(Linux Containers), Firecracker, gVisor 4개의 환경에서 사용 가능한 시스템 콜과 실행되는 커널 코드의 양을 분석하였다. 또한, 네트워크, 메모리 관리 그리고 파일 접근 측면에서 종합적으로 성능을 비교하였다. 그 결과, 파일 접근 측면에서 Firecracker가 다른

플랫폼들보다 훨씬 앞섬을 보였다.

Wang [12]에서는 microVM이 FaaS(Function as a Service)에 적합한 컴퓨팅 환경인지를 확인하기 위하여 가상머신, 컨테이너 그리고 microVM의 startup time, 디스크 I/O, 네트워크, 메모리 성능을 비교하였다. Startup time은 가상머신이 가장 느리고, 컨테이너가 가장 빨랐다. 디스크 I/O 측정에는 iozone을 사용하였다. Firecracker는 레코드의 크기가 작은 경우 컨테이너보다 성능이 상대적으로 떨어지지만, 레코드 크기가 증가함에 따라 컨테이너와 비슷한 성능을 보이거나 앞선다. 네트워크 성능 측정에는 iperf3를 사용하였다. 컨테이너는 native 수준으로 빠른 속도를 보였다. 반면에, virtio를 사용하는 Firecracker는 전 가상화(Full Virtualization)가 필요한 가상머신보다는 빠름을 보였다.

Goethals et al. [13]에서는 엣지 디바이스에서 light-weight 가상화 환경의 성능을 분석하였다. 실험은 x86-64 시스템과 라즈베리파이에서 수행되었다. 연구진들은 벤치마크 수행을 위해 간단한 REST 서비스를 구현하고 GET, POST 메서드를 정의하였다. 실험 결과, 싱글코어 실험에서의 request 처리 성능은 OSv가 가장 빨랐다. 또한, microVM의 성능은 커널 디자인과 가상 드라이버의 종류에 따라 크게 다를 수 있음을 확인하였다. 그러나 4개의 코어를 사용하는 실험에서 컨테이너와 gVisor는 사용 가능한 코어 수에 따라 원활하게 확장되었지만, 가상머신 기반 플랫폼들의 확장성은 상대적으로 떨어졌다. 그리고 POST request에 대한 응답 시간 비교에서, 컨테이너는 평균적으로 상당히 낮은 레이턴시를 기록하였다. 가상머신 기반 플랫폼들의 평균 레이턴시는 높았지만, 상당히 안정적인 최대 레이턴시를 보였다.

Lee et al. [14]에서는 라즈베리파이 4 모델 B와 엔비디아의 Jetson Nano 2GB, 4GB에서 native, 컨테이너, 가상머신, Firecracker, Kata container의 성능을 다양한 측면에서 종합적으로 분석하였다. 결과 중에서 네트워크 I/O에 주목할 만하다. 해당 연구에서는 netperf benchmark를 사용하여 TCP 패킷을 서버로 보내어 네트워크 transmission 성능을 측정하였다. 그 결과 Firecracker와 Kata container는 거의 비슷한 성능을 보였으며, microVM이 기존 컨테이너보다 평균적으로 3배 좋은 성능을 보였다. 또한, microVM이 컨테이너보다 더 높은 전력 소비량을 보이지만 네트워크 성능을 함께 고려하였을 때 microVM이 12% 더 높은 효율성을 보였다.

하지만 선행 연구들의 분석 결과만으로는, 실제 AI 애플리케이션을 microVM에서 실행했을 때 기존 플랫폼들과 비교하여 어떤 성능을 보일지 정확히 예측하기 어렵다. 이에 따라, 본 연구에서는 이러한 부분에 집중하여 실험과 분석을 수행하였다.

III. Experiment

1. Hardware Specification

본 실험에는 라즈베리파이 4 Model B를 사용하였다. 이 장치는 Quad core Cortex-A72 1.50 GHz CPU와 8GB LPDDR4-3200 SDRAM을 탑재하고 있다. 실험 환경의 분리를 위해, 32GB micro-SD 카드 3개를 활용하여 각각의 가상화 환경을 구축하였다.

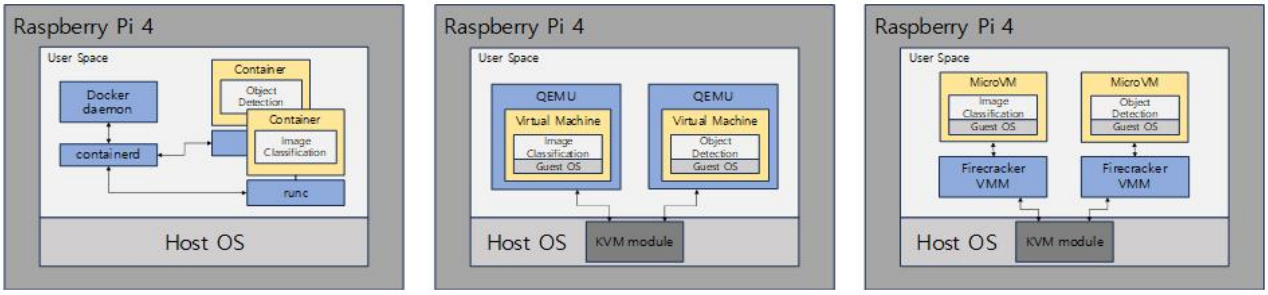
2. Method

Fig. 1은 실험에서 사용한 각 환경의 내부 구조를 나타낸 것이다. 총 4개의 AI 애플리케이션을 사용하였지만 Fig. 1에서는 공간의 제약으로 인해 2개의 애플리케이션만 표현하였다. 라즈베리파이의 호스트 운영체제는 Ubuntu Desktop 22.04.3 LTS를 사용하였다.

Docker 컨테이너는 AI 벤치마크 애플리케이션별로 각각의 Dockerfile을 작성하여, 총 4개의 Docker 이미지를 생성하였다. Dockerfile에서는 베이스 이미지로 Ubuntu 22.04를 선택하였으며, RUN 명령어를 이용하여 필요한 패키지 및 라이브러리 그리고 AI 애플리케이션의 깃허브 레파지토리를 다운로드 하였다. 마지막으로, 컨테이너 실행 시 애플리케이션 수행 스크립트(e.g. run.sh, test.py)가 작동하도록 ENTRYPOINT를 정의하였다.

Firecracker의 경우 공식 깃허브에서 제공하는 매뉴얼[15]을 참고하여 Ubuntu 22.04를 운영체제 이미지로 사용하는 microVM을 생성하였다. 4개의 애플리케이션 각각에 대해 rootfs(Root File System) 파일을 별도로 만들어 환경을 분리하였고, microVM 부팅 시에는 --config-file 옵션을 추가하여 microVM의 자원 설정과 관련된 json 파일을 지정하였다. 해당 json 파일은 microVM이 사용할 게스트 커널과 rootfs 파일 설정이 필수적으로 포함되어야 하며, 추가적인 옵션으로 4개의 vCPU, 4GB의 메모리를 microVM에 할당하도록 설정하였다. 이와 같은 설정을 바탕으로 벤치마크 애플리케이션 당 microVM 1개씩 총 4개의 microVM을 생성하였다.

KVM 가상머신도 Firecracker와 유사하게 Ubuntu 22.04를 운영체제로 선택하였고, virt-install 명령어의 옵션으로 4개의 vCPU, 4GB 메모리 그리고 8GB 디스크 파



(a) Container (b) Virtual Machine (c) MicroVM

Fig. 1. Diagram of experimental environment

일을 설정해 주었다.

본 연구에서는 Edge AIBench 벤치마크 중에서 가장 핵심적인 두 가지 애플리케이션을 선정하여 실험을 수행하였다. 첫 번째로, Object Detection에서는 사전 학습된 YOLOv5 딥러닝 모델을 사용하였고, 추론 과정에는 MS COCO(Microsoft Common Objects in Context) 데이터 셋을 활용하였다. 두 번째로, Traffic Light Classification에서는 학습된 CNN(Convolutional Neural Network) 모델을 사용하였으며, 추론에는 GTSR(German Traffic Sign Recognition) 데이터 셋을 활용하였다.

또한, Edge AIBench 외에 실제 IoT 제품에서 널리 활용되는 두 가지 AI 애플리케이션을 추가로 선정하였다. 첫 번째로, Face Recognition에서는 DSFD(Dual Shot Face Detector)[16]와 RetinaFace[17]를 기반으로 추론 성능이 향상된 파이토치 구현 모델[18]을 사용하였으며, 추론에는 FDDB(Face Detection Data Set and Benchmark) 데이터 셋[19]을 활용하였다. 두 번째로, Speech-To-Text에서는 PyTorch Hub에서 불러올 수 있는 사전 학습된 Silero 모델[20]을 사용하였으며, 추론에는 LJ Speech 데이터 셋[21]을 활용하였다.

본 연구에서 활용한 실험 방법은 다음과 같다. 첫 번째로, 각 애플리케이션에서 사전 학습된 모델을 사용하여 추론 과정을 수행하고, 이에 소요되는 시간(Inference time)을 측정하였다. 두 번째로, 벤치마크 애플리케이션을 실행하는 동안 각 환경의 CPU 사용량을 측정하였다. Docker 컨테이너의 경우에는 docker stats 명령어를 사용하였고, Firecracker와 KVM은 리눅스 ps 명령어로 게스트 머신의 프로세스 아이디를 추적한 뒤, 리눅스 top 명령어로 해당 프로세스의 CPU 사용량을 측정하였다. 세 번째로, 엣지 컴퓨팅 구조에서는 엣지 레벨에서 데이터 전처리를 수행한 후 상위 레벨로 데이터를 전송하는 경우가 많다. 이를 반영하여, AI 애플리케이션의 수행 결과(e.g. 음성 파일에서 번역된 텍스트, 바운딩 박스가 적용된 얼굴 사진)를 라즈베리파이에서 원격 서버로 전송하는 상황을 가정하고 데이터 전송 성능을 측정하였다. 리눅스 scp 명령어를 사용하여 MS COCO 데이터 셋 중에서 한 장의 파일 크기가 48KB인 컬러 사진 300장을 원격 IP 주소로 전송하고, v 옵션을 추가하여 전송된 데이터의 양, 걸린 시간, 전송 속도를 확인하였다. 실험에 사용된 라즈베리파이와 서버 PC는 하나의 사설 네트워크에 연결되어 있다. 마지막으로, AI 모델 학습이나 추론 과정에서는 대용량의 데이터가 사용되므로, 각 환경에서 데이터 읽기/쓰기 성능을 측정하였다. 이를 위해, 리눅스 시스템에서 널리 사용되는 벤치마크 도구인 sysbench를 활용하였다. sysbench에서 fileio 옵션을 사용하여 각 환경에서의 파일 I/O 성능을 측정하였고, O_DIRECT 옵션을 추가하였다. O_DIRECT 옵션을 사용하게 되면 I/O를 진행할 때 리눅스 커널 버퍼에 데이터를 캐싱하지 않고 직접적으로 디스크에 접근하기 때문에, 각 실험 환경의 순수 디스크 성능을 측정하기 위해 해당 옵션을 설정해 주었다. 또한, 2개의 쓰레드를 사용하여 총 5GB 파일에 대하여 파일 블록의 크기를 변경해 가며 실험을 진행하였다.

IV. Result

1. Performance Analysis on AI Applications

1.1 Inference Time

Fig. 2는 세 가지 실험 환경에서 4개의 벤치마크 애플리케이션을 각각 50회씩 수행한 후, 추론 시간의 평균값을 나타낸 것이다. 마지막 페이지에 제시된 Appendix Table 1은 Image Classification 애플리케이션의 수행을 10회씩 추가할 때마다 추론 시간의 표준편차 변화를 나타낸 것이다. 이를 통해, 50회의 실험만으로도 애플리케이션 추론 시간의 변화가 적어져 안정된 표준편차를 보이는 것을 확인할 수 있다. 실험 결과를 통해 알 수 있는 사실들은 다음

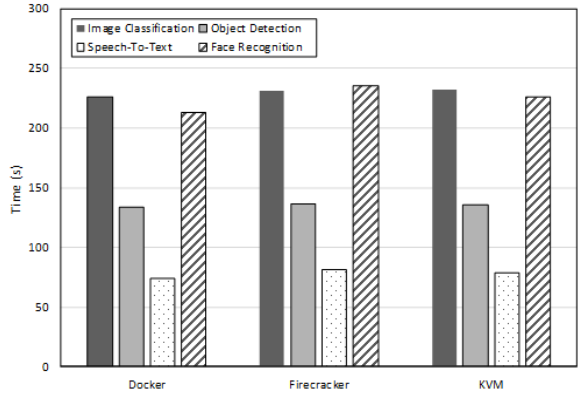


Fig. 2. Comparison chart of inference time

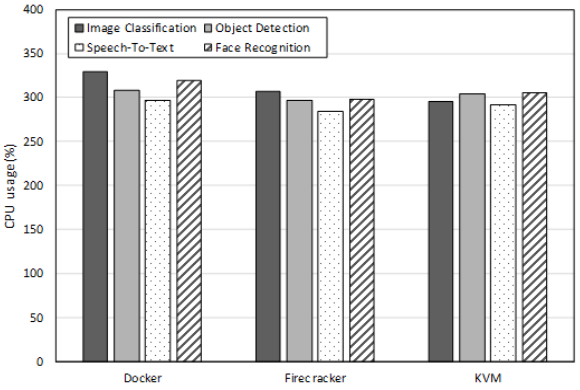


Fig. 4. Comparison chart of total CPU usage

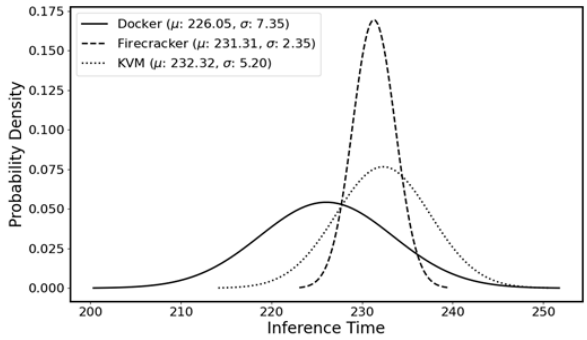


Fig. 3. Normalized distribution of Image Classification inference times in different environments

과 같다. 첫째, 네 가지 벤치마크 애플리케이션 모두에서 Docker 컨테이너가 평균적으로 가장 빠른 추론 시간을 보였다. 둘째, 대부분 애플리케이션에서 컨테이너와 microVM 그리고 가상머신의 평균 추론 시간 차이는 모두 10초 이내였다. 셋째, Face Recognition에서 실험 환경들이 가장 큰 차이를 보였다. Docker 컨테이너와 비교하여, Firecracker microVM은 약 22.6초 느리고 KVM 가상머신은 약 12.8초 느린 추론 시간을 보였다.

Fig. 3은 각 환경에서 Image Classification 추론 시간의 정규분포 그래프이다. 평균 추론 시간은 Docker 컨테이너가 가장 빠르지만, Firecracker microVM이 가장 작은 표준편차를 보였고, KVM 가상머신 또한 컨테이너에 비해 작은 표준편차를 보인다. 다른 애플리케이션의 실행 결과에서도 비슷하게 microVM과 가상머신의 표준편차가 가장 작다. 이는 다른 실험 환경에 비하여 microVM에서 딥러닝 모델의 추론 과정이 매번 안정적으로 이루어짐을 의미한다고 볼 수 있다.

가상머신은 각각의 전용 운영체제를 가지고 있어 프로세스의 자원 할당이 안정적이며, 이로 인해 추론 과정 시 성능 변동이 적다. 반면, 컨테이너는 호스트 시스템의 운영체제와 자원을 공유하기 때문에 성능 변동이 발생할 수 있다. 실험 결과를 통해, microVM에서 AI 애플리케이션 추론 시간의 편차가 적으므로 비즈니스에 적용할 때 고려해야 할 변수를 줄일 수 있을 것으로 예상된다.

1.2 CPU Usage

Fig. 4는 1.1과 동일 실험을 진행하는 동안 측정한 각 환경의 CPU 사용량을 나타낸다. 보통 CPU 사용량의 최대치는 100%로 표기한다. 하지만 본 실험에서는 자세한 CPU 사용량을 분석하기 위해, 4개의 CPU 코어의 사용량 각각을 최대 100%로 하여 총 400%를 기준으로 측정하였다. 벤치마크 애플리케이션이 동작할 때는 프로세스의 CPU 사용량이 최소 100%를 넘어갔기 때문에, 측정값 중 100% 이상인 값만을 대상으로 평균값을 구하였다.

실험 결과를 통해 알 수 있는 사실은 다음과 같다. 첫째, 동일 환경에서 실행된 4개의 벤치마크 애플리케이션 간에 CPU 사용량의 차이는 크지 않다. 일반적으로, 사용된 딥러닝 모델과 데이터의 크기가 클수록 더 많은 CPU를 사용하였지만, 그 차이는 적었다. 둘째, Docker 컨테이너의 CPU 사용량이 가장 높았으며, microVM과 가상머신은 비슷하였다.

Fig. 5, 6, 7은 각 실험 환경에서 Image Classification 벤치마크 애플리케이션을 동작시킨 후, 시간에 따른 CPU 사용량 변화를 나타낸 것이다. Image Classification은 컬러 신호등 이미지에 대하여 빨간불, 파란불, 노란불을 판단하는 애플리케이션이다. 다른 애플리케이션 및 반복적인 수행에서도 모두 비슷한 추이를 보인다. 위 그래프를 통해 실험 환경에 따라 평균 CPU 사용량이 다른 이유를 확인할 수 있다. Docker 컨테이너의 경우, 벤치마크 애플리케이션의 추론 과정이 시작되면 프로세스의 CPU 사용량의 변화가 완만하다. 하지만 Firecracker와 KVM의 경우, 변화가 반복적으로 빠르게 발생한다. 결과적으로, 꾸준히 높은 CPU 사용량을 유지하는 Docker 컨테이너가 추론 과정을

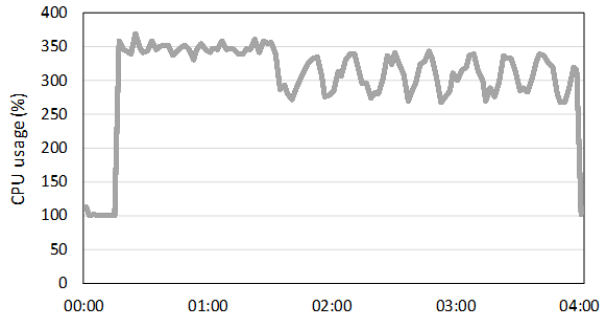


Fig. 5. Docker container's CPU usage trend and x-axis is time (mm:ss)

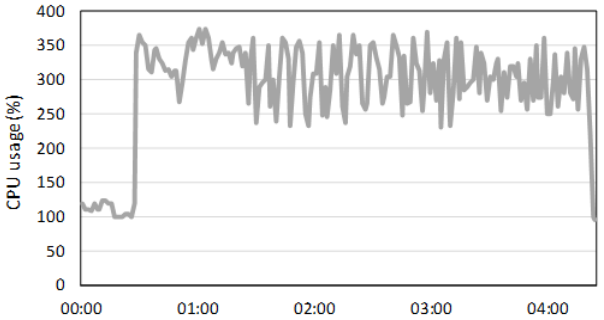


Fig. 6. Firecracker microVM's CPU usage trend and x-axis is time (mm:ss)

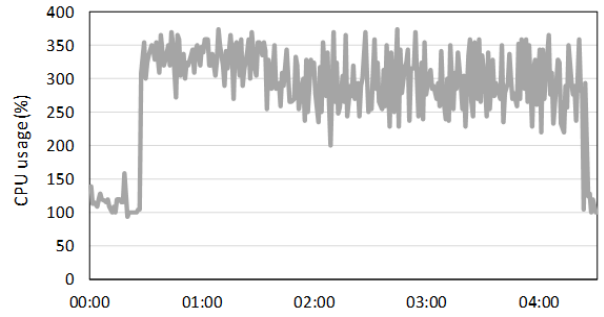


Fig. 7. KVM virtual machine's CPU usage trend and x-axis is time (mm:ss)

가장 빨리 마치게 되며, Firecracker와 KVM이 그 뒤를 따른다.

마지막으로, 벤치마크 애플리케이션에서 추론 과정이 본격적으로 진행될 때는 프로세스의 CPU 사용량이 최소 200% 이상을 넘어갔다. CPU 사용량이 최초로 200%를 달성하기까지 Docker 컨테이너는 약 15초, Firecracker와 KVM은 약 26초가 걸리는 것으로 파악되었다. 이러한 것은 컨테이너와 가상머신의 가상화 방식과 구조적인 차이 때문으로 보인다.

Table 1. Comparison of data transfer performance

	Docker	Firecracker	KVM
Time (s)	4.2	6.7	13.5
Transfer Speed (KB/sec)	3480.47	2184.02	1085.47

Table 2. Comparison of CPU performance on sysbench

	Docker	Firecracker	KVM
Total execution time (s)	10.0022	10.00205	10.00202

2. Other Performance Analysis

2.1 Data transfer performance using scp

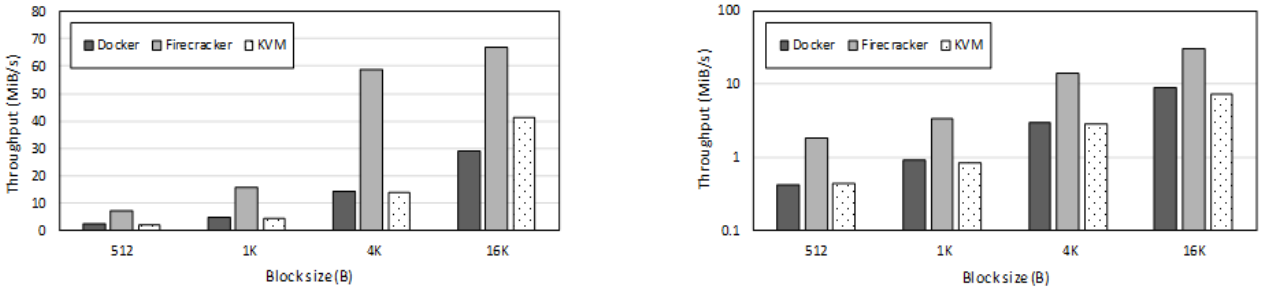
Table 1은 각 실험 환경에서 원격 PC로 데이터를 전송하였을 때의 성능을 비교한 결과이다. 실험 결과, 동일 크기의 데이터를 Docker 컨테이너가 가장 빠르게 전송하였다. Firecracker microVM은 KVM 가상머신에 비해 약 2배 빠르게 데이터를 전송하였다. 이는 microVM의 간소화된 구조와 디바이스 모델 때문이다.

2.2 CPU Benchmark using sysbench

Table 2는 sysbench 벤치마크를 활용하여 3가지 실험 환경에서 CPU benchmark를 수행한 결과이다. 명령어 실행 시, 옵션으로 쓰레드의 개수를 4개로 설정하여 라즈베리파이의 4개 core를 모두 사용하도록 설정하였다. 실험 결과, 각 환경에서 CPU 성능의 차이는 거의 없음을 확인할 수 있다.

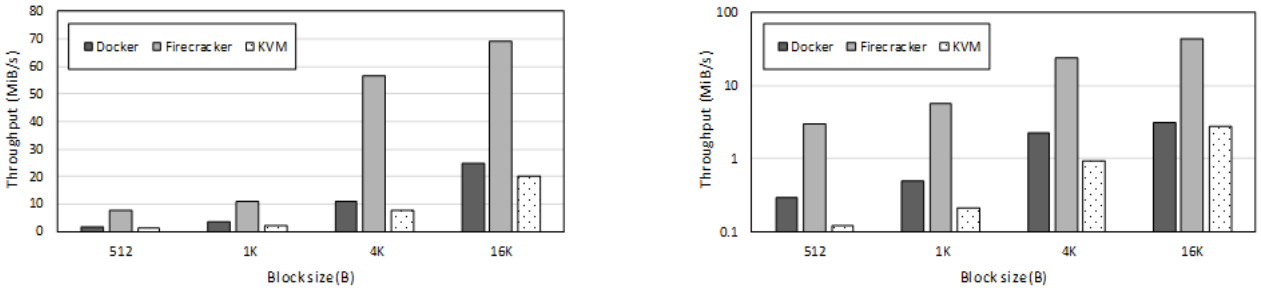
2.3 File I/O Benchmark using sysbench

Fig. 8(a), 8(b)은 각 실험 환경에서 파일 블록의 크기를 바꿔가며 sequential read와 sequential write를 할 때 성능을 측정한 결과이다. 3가지 실험 환경 중에서 Firecracker가 read와 write 둘 다 가장 좋은 성능을 보였다. 라즈베리파이의 실제 디스크와 상호작용하는 Docker 컨테이너와 달리, Firecracker microVM은 물리 디스크로 직접 데이터를 쓰지 않고 가상머신의 디스크로 할당된 메모리영역과 상호작용하기 때문에 높은 성능을 보인다. 또한, microVM은 반 가상화된 virtio-blk를 block I/O에 사용하기 때문에, 조금 더 효율적으로 동작한다. KVM도 기본적으로 virtio를 사용하지만, Firecracker는 2장 1.3절에서 설명한 것처럼 KVM보다 코드 수가 적고 간소화되었기 때문에 오버헤드가 줄어 약간 더 높은 성능을 보인다.



(a) sequential read (b) sequential write

Fig. 8. Result of sysbench sequential workload



(a) random read (b) random write

Fig. 9. Result of sysbench random workload

Fig. 9(a), 9(b)를 통해 볼 수 있듯이, random read와 random write에서도 sequential 워크로드와 마찬가지로 Firecracker가 다른 플랫폼들에 비해서 가장 좋은 성능을 보인다. 다만, Docker와 Firecracker 간의 파일 I/O 성능의 차이가 큼에도 불구하고 벤치마크 애플리케이션의 수행 시간의 차이가 크지 않은 것을 보아, 해당 벤치마크 애플리케이션에서는 파일 I/O의 성능이 큰 영향을 주지 않는 것으로 보인다. 추후 연구에서 다른 유형의 벤치마크를 통해 연관성을 정확히 파악할 필요가 있다.

V. Conclusions

본 연구에서는 엣지 컴퓨팅 환경에서 AI 애플리케이션을 실행 및 배포할 때, 현재 사용되고 있는 컨테이너와 가상머신을 대체하여 microVM이 사용될 수 있는지를 판단하고자 실험을 진행하였다. 라즈베리파이에서 Docker 컨테이너, Firecracker microVM, KVM 환경을 구축한 후 여러 벤치마크 애플리케이션을 수행하고 성능을 비교하였다. 실험 결과, 컨테이너에서의 벤치마크 애플리케이션 추론 시간이 가장 짧았지만, microVM과 KVM도 비슷한 성능을 보였다. 또한, 벤치마크 애플리케이션을 수행하는 동안 CPU 사용량은 추론 시간이 빠를수록 많았다. 자세한 분석을 통해, 반복적으로 벤치마크 애플리케이션을 수행할 때, microVM이 컨테이너에 비해 일정한 추론 시간이 소요됨을 확인하였다. 마지막으로, 원격 PC로의 데이터 전송 실험에서 microVM은 컨테이너보다 느리고 KVM보다 빠른 성능을 보였으며, 파일 I/O 성능 분석에서는 다른 실험 환경과 비교하여 월등히 좋은 성능을 보였다.

결론적으로, 엣지 디바이스의 microVM에서 AI 애플리케이션을 수행하였을 때, 기존의 컨테이너와 성능 면에서 크게 차이가 나지 않으며 오히려 높은 안정성, 파일 I/O 성능을 제공하는 것을 확인하였다. 호스트 시스템과 커널을 공유하지 않아 높은 보안성을 가진다는 microVM의 장점도 함께 고려한다면, 엣지 컴퓨팅 환경에서 microVM의 컨테이너 대체에 대한 적합성 일부 측면을 확인하였다고 할 수 있다. 이를 통해 자율주행, 스마트 시티 등과 같은 실시간 처리 성능과 데이터 보안이 중요한 실제 시나리오에서 IoT, 엣지 컴퓨팅 및 AI 애플리케이션 구동을 위한 환경 구축 시 microVM이 충분히 활용될 수 있음을 알 수 있다. 하지만 라즈베리파이가 엣지 컴퓨팅 환경을 대표하지는 않으므로, 본 연구의 실험 결과가 일반적으로 적용될지에 대해서는 면밀한 검토가 필요하다.

추후 연구에서는 엔비디아 Jetson Nano, 구글 Coral Dev Board 등의 다양한 엣지 디바이스를 활용하여 종합적인 엣지 컴퓨팅 시나리오를 분석할 것이다. 또한, 엣지

디바이스에서 메모리는 중요한 자원 중 하나이므로, 다양한 가상화 환경에서 AI 애플리케이션이 수행될 때 메모리 사용에 관한 자세한 분석을 진행할 것이다.

ACKNOWLEDGEMENT

This study was supported by the BK21 FOUR project (AI-driven Convergence Software Education Research Program) funded by the Ministry of Education, School of Computer Science and Engineering, Kyungpook National University, Korea (4199990214394). This work was also supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (No. NRF-2021R1A 5A1021944).

REFERENCES

[1] J.A. Stankovic, "Research Directions for the Internet of Things," IEEE Internet of Things Journal, Vol. 1, No. 1, pp. 3-9, Feb 2014. DOI: 10.1109/JIOT.2014.2312291

[2] Statista, Number of Internet of Things (IoT) connected devices worldwide from 2019 to 2023, with forecasts from 2022 to 2030 (in billions), *https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/*

[3] S. Al-Sarawi, M. Anbar, R. Abdullah, and A.B. Al Hawari, "Internet of Things Market Analysis Forecasts, 2020-2030," Fourth World Conference on Smart Trends in Systems, Security and Sustainability (WorldS4), pp. 449-453, July 2020. DOI: 10.1109/WorldS450073.2020.9210375

[4] A. Agache, M. Brooker, A. Florescu, A. Iordache, A. Liguori, R. Neugebauer, P. Piwonka, and D.M. Popa, "Firecracker: Lightweight virtualization for serverless applications," 17th USENIX NSDI, pp. 419-434, Feb 2020.

[5] Kata Containers, *https://katacontainers.io/*

[6] M. Fezari, and A.A Dahoud, "Raspberry Pi 5 : The new Raspberry Pi family with more computation power and AI integration," Nov 2023. DOI: 10.13140/RG.2.2.13547.52009

[7] T. Hao, W. Gao, C. Lan, F. Tang, Z. Jiang, and J. Zhan, "Edge AIBench 2.0: A scalable autonomous vehicle benchmark for IoT-Edge-Cloud systems," BenchCouncil Transactions on Benchmarks, Standards and Evaluations, Vol. 2, No. 4, Oct 2022. DOI: 10.1016/j.tbench.2023.100086

[8] H. Hua, Y. Li, T. Wang, N. Dong, W. Li, and J. Cao, "Edge computing with artificial intelligence: A machine learning perspective," ACM Computing Surveys, Vol. 55, No. 9, pp. 1-35, Jan 2023. DOI: 10.1145/3555802

[9] A. Villanueva, R. L. L. Benemerito, M. J. M. Cabug-Os, R. B. Chua, C. K. D. Rebeca, and M. Miranda, "Somnolence detection system utilizing deep neural network," International Conference on Information and Communications Technology (ICOIACT) IEEE, pp. 602-607, 2019. DOI: 10.1109/ICOIACT46704.2019.8938460

[10] G. Lavanya, C. Rani, and P. Ganeshkumar, "An automated low cost IoT based fertilizer intimation system for smart agriculture," Sustainable Computing: Informatics and Systems, Vol. 28, Dec 2020. DOI: 10.1016/j.suscom.2019.01.002

[11] Anjali, T. Caraza-Harter, and M.M. Swift, "Blending containers and virtual machines: a study of firecracker and gVisor", Proceedings of the 16th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, pp. 101-113, March 2020. DOI: 10.1145/3381052.3381315

[12] Z. Wang, "Can "micro VM" become the next generation computing platform?: Performance comparison between light weight virtual machine, container, and traditional virtual machine," IEEE International Conference on Computer Science, Artificial Intelligence and Electronic Engineering (CSAIEE), pp. 29-34, Oct 2021. DOI: 10.1109/CSAIEE54046.2021.9543457

[13] T. Goethals, M. Sebrechts, M. Al-Naday, B. Volckaert, and F. De Turck, "A functional and performance benchmark of lightweight virtualization platforms for edge computing," IEEE EDGE, pp. 60-68, Aug 2022. DOI: 10.1109/EDGE55608.2022.00020.

[14] K. Lee, and B. Tak, "MicroVM on Edge: Is It Ready for Prime Time?," International Symposium on the Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS), pp. 1-8, Oct 2023. DOI: 10.1109/MASCOTS59514.2023.10387638

[15] Firecracker, *http://acoms.atit.co.kr:9090/tkioa/index.jsp?publisher_cd=tkioa*

[16] J. Li, Y. Wang, C. Wang, Y. Tai, J. Qian, J. Yang, C. Wang, J. Li, and F. Huang, "DSFD: dual shot face detector," Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 5060-5069, June 2019. DOI: 10.1109/CVPR.2019.00520

[17] J. Deng, J. Guo, Y. Zhou, I. Kotsia, and S. Zafeiriou, "Retinaface: Single-stage dense face localisation in the wild," arXiv preprint, 2019. DOI: 10.48550/arXiv.1905.00641

[18] DSFD-Pytorch-Inference, *https://github.com/hukkelas/DSFD-Pytorch-Inference*

[19] V. Jain, and E. Learned-Miller, "Fddb: A benchmark for face detection in unconstrained settings," UMass Amherst technical report, Vol. 2, No. 6, pp. 1-11, 2010.

[20] Silero Team, Silero Models: pre-trained enterprise-grade STT / TTS models and benchmarks, *https://github.com/snakers4/silero-models*

[21] K. Ito, and L. Johnson, The LJ Speech Dataset, *https://keithito.com/LJ-Speech-Dataset/*

Appendix

Table 1. Standard deviation variation of Image Classification inference time by the number of trials

Trial	Docker	Firecracker	KVM
1st ~ 10th	9.15	5.28	8.25
1st ~ 20th	8.61	4.76	6.98
1st ~ 30th	8.0	3.29	6.31
1st ~ 40th	7.52	2.83	5.74
1st ~ 50th	**7.35**	**2.35**	**5.20**

Authors

Yunha Choi received the B.S. degree in Computer Science and Engineering from Kyungpook National University, Daegu, Korea in 2024. He is interested in IoT, cloud/edge computing and AI.

Byungchul Tak received his B.S. degree from Yonsei University in 2000, M.S. from KAIST in 2003 and Ph.D. degrees in Computer Science and Engineering from the Pennsylvania State University at University. Park in 2012. Dr. Tak joined the faculty of the School of Computer Science and Engineering at Kyungpook National University, Dague, Korea, in 2017. He is currently an Associate Professor. His research interests are in cloud computing, distributed systems, operating system and big data analytics.