논문 22-47-04-15 The Journal of Korean Institute of Communications and Information Sciences '22-04 Vol.47 No.04
https://doi.org/10.7840/kics.2022.47.4.690

컨테이너 기반의 모바일 엣지 컴퓨팅을 위한 무중단 서비스 이관 시스템 구현

김 태 운◆°

Implementation of the Zero-Downtime Service Migration System for Mobile Edge Computing Based on Containerization

Taewoon Kim◆°

요 약

엣지 컴퓨팅은 연산 및 스토리지 자원을 네트워크 가장자리에 배치함으로써 연산 오프로딩을 비롯한 클라우드 컴퓨팅의 장점뿐 아니라 낮은 지연시간, 네트워크 대역폭 절약, 보안 위협 감소 등의 장점을 제공한다. 모바일 엣지 컴퓨팅 환경에서 사용자 단말의 이동으로 인해 엣지 서버와 물리적 거리가 멀어지는 경우 지연시간 감소를 위해 서비스 이관이 필요하며, 특히 이관이 진행되는 동안 지속해서 서비스를 제공하는 무중단 서비스 이관기법이 필수적이다. 본 논문에서는 도커 컨테이너 가상화 플랫폼을 사용하는 모바일 엣지 컴퓨팅 환경에서 상태 기반 서비스 이관기법 및 무중단 서비스 이관기법의 구현을 소개한다. 본 논문에서 다루는 기법은 도커 컨테이너의 계층적 레이어 구조 및 체크포인트 기능을 활용하여 파일 시스템의 변경사항, 프로세스 실행상태, 메모리 상태 등을 보존하는 컨테이너 이관을 지원한다. 또한, 이관이 진행되는 도중에 발생하는 상태 변경을 추적하기 위해 리플레이 버퍼를 사용한다. 실행 중인 컨테이너 전체를 이관하는 기법과 컨테이너의 변경사항만 이관하는 두 가지 기법을 구현하고 실험한 결과, 두 기법 모두 컨테이너 이관이 진행되는 중에 서비스 중단이 발생하지 않는 것을 확인했다. 게다가, 컨테이너의 변경사항만 이관하는 기법은 컨테이너 전체를 이관하는 기법에 비해 이관시간 및 오프로딩 서비스 응답시간이 감소하는 것을 확인했다.

키워드 : 엣지 및 포그 컴퓨팅, 모바일 컴퓨팅, 서비스 이관, 네트워크 응용 및 서비스, 모바일 클라우드 컴퓨팅 응응

Key Words : Edge and fog computing, Mobile computing, Service migration, Network applications and services, Application for mobile cloud computing

ABSTRACT

By placing computing and storage resources at the network edge, edge computing inherits the advantages of cloud computing, such as providing offloading services, in addition to the additional benefits, including the reduced network delay, bandwidth use and security threat. On edge computing, the user mobility may increase the physical distance to edge server, resulting in the increased network delay. To keep the delay minimized, service migration is required, and it is important to guarantee the zero-downtime during migration. This paper introduces the implementation of a stateful, live containerized service migration for edge computing on the

※ 이 논문은 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(No. 2021R1F1A1059109)
◆° First and Corresponding Author : Hallym University, School of Software, taewoon@hallym.ac.kr, 정회원
논문번호 : 202111-319-D-RN, Received November 17, 2021; Revised January 3, 2022; Accepted January 3, 2022

Docker container platform. The considered migration methods can migrate the container state by using both the layered architecture of Docker containers and the checkpoint function. Also, to trace the state changes that may happen during migration, a replay buffer is used. This paper introduces the implementation of two migration techniques that one migrates the entire running container, while the other migrates only the state changes. By experiments, it is validated that the considered migration techniques do not incur any service downtime even during when migration is in progress. In addition, it is found that the method that migrates only the state changes outperformed the other.

Ⅰ. 서 론

자원 집약적 데이터센터를 기반으로 하는 클라우드 컴퓨팅 서비스[1]는 가상화 기술을 통해 자원 효율성을 극대화한다. 또한, 소비자의 필요에 따른 신속한 주문형(on-demand) 자원 할당, 사용량 당 지불 방식(pay-as- you-go)을 통한 합리적 요금 정책, 인프라 구축·운영비용 절감 등의 장점에 힘입어 클라우드 컴퓨팅은 전통적인 시스템 운영 환경(on-premise)의 효과적인 대안으로 자리 잡고 있다. 연산성능 및 배터리 제약이 큰 이동형 단말을 사용하는 소비자는 클라우드 컴퓨팅 자원으로의 오프로딩 기술을 활용하여 복잡한 연산의 처리시간 감소 및 배터리 절약의 효과를 얻을 수 있다[2].

하지만, 제한된 수의 데이터 센터는 사용자 단말로부터 거리가 멀고, 이로 인해 네트워크 지연시간 및 오프로딩 서비스 응답시간이 길어지는 단점이 있다[3]. 따라서, 지연시간에 민감한 일부 응용 서비스(예: 감시/모니터링, 증강현실, 헬스케어, 자율주행[4])는 클라우드로의 오프로딩 서비스를 사용하기 어려운데, 이를 극복하기 위해 엣지 컴퓨팅(Edge Computing, EC)[5-7] 기술이 등장했다. EC은 분산형 클라우드 컴퓨팅의 일종으로 볼 수 있는데, 네트워크 가장자리에 컴퓨팅 및 스토리지 자원을 배치하는 형태이기 때문이다. 사용자 단말과 가까워진 물리적 거리로 인해 통신 지연시간 및 오프로딩 서비스 응답시간이 감소하는 것은 물론, 사용자 데이터의 송수신이 사용자 근처에서만 발생하므로 대역폭 절약, 보안 위협 감소 등의 추가적인 장점이 있다.

모바일 엣지 컴퓨팅(Mobile Edge Computing, MEC) 환경에서는 사용자 단말이 이동성을 가지며, 현재 서비스를 제공하는 엣지 서버(Edge Server, ES)와 이동형 사용자 단말(Mobile User Device, MUD) 간 물리적 거리가 멀어지는 경우가 발생할 수 있다. 이는 통신 지연 및 서비스 응답시간 증가로 이어져 서비스의 실시간성을 보장하기 어려워진다. MUD가 이동하는 경우라 할지라도 지속해서 지연시간을 최소화하기 위해 지리적으로 가까운 ES에 서비스를 이관(Migration)하는 기법에 관한 연구가 활발히 진행되고 있다.

서비스 이관기법은 다양하게 분류할 수 있는데, 상태보존 여부에 따라 State-less(상태를 보존하지 않음)와 Stateful(상태를 보존함) 기법으로 구분하고, 이관 중 서비스 단절 발생 여부에 따라 Cold Migration(서비스 단절이 발생함)과 Live(Hot 또는 Zero -Downtime, 서비스 단절이 발생하지 않거나, 그 영향을 최소화함) Migration으로 구분한다[8]. 본 논문에서는 Stateful Migration과 Live Migration을 동시에 지원하는 이관기법을 구현하며, 이를 통해 이관 전·후에 동일한 서비스 상태를 유지하고, 이관이 진행되는 도중에도 지속적으로 서비스를 제공할 수 있다. 이관 전·후에 서비스가 동일한 상태를 가지기 위해서는 “상태”를 전송할 필요가 있는데, 이때 상태는 파일 시스템의 변경사항, 프로세스 실행상태, 메모리 상태(즉, 메모리에 저장된 값) 등을 포함한다.

클라우드 및 엣지 컴퓨팅에서 사용하는 다양한 가상화 기법 중 컨테이너 가상화 기술은[9] 호스트 운영체제를 공유함으로써 경량의 가상환경을 구축할 수 있고, 호스트 가상화 기법에 비해 신속한 이관을 가능케 한다. 도커[10]는 가장 널리 사용되는 컨테이너 가상화 플랫폼 중 하나이며, 네이티브에 가까운 성능을 보이는 것으로 알려져 있다[11,12]. 본 논문에서는 도커 기반의 컨테이너 가상화 환경을 사용하는 EC 환경에서 상태를 보존하고 서비스 중단이 없는 컨테이너 이관 기술을 구현한다.

본 연구는 C-Migr(Complete-Migration) 과 P-Migr (Partial-Migration)의 두 가지 컨테이너형 서비스 이관기법을 구현한다. C-Migr은 현재 구동 중인 컨테이너 전체를 전송하여 파일 시스템 변경사항을 보존하고, 도커 체크포인트 기능을 사용하여 프로세스 실행상태와 메모리 상태를 보존 및 전송한다. 반면, P-Migr은 도커 컨테이너의 계층적 레이어 구성을 활

용하여 파일 시스템의 변경사항만 전송하고 도커 체크포인트를 통해 프로세스 및 메모리 상태를 전송한다. 본 논문은 기성 데스크탑 PC 및 노트북을 사용하여 두 가지 이관기법을 구현하는 방법을 소개한다. 또한, 자동적 이관기법을 구현하여 관리자의 개입 없이 컨테이너형 서비스의 자동 이관을 지원하는 시스템, 및 그 구현을 소개한다.

본 논문은 다음과 같이 구성되어있다. 2장에서는 컨테너 기반의 서비스 이관에 관한 기존 기술을 검토한다. 3장에서는 두 가지 이관기술의 구현을 소개하고, 4장에서는 실험 결과를 분석한다. 마지막으로 5장에서는 결론 및 향후 연구 방향을 논의한다.

Ⅱ. 관련 연구

컨테이너형 서비스 이관 또는 컨테이너 이관기법은 "상태 전송" 및 "상태 재구성"의 두 가지로 분류할 수 있다. 원활한 설명을 위해 다음의 시나리오를 가정한다: MUD가 ES-x에서 동작하는 컨테이너 CT-x를 통해 오프로딩 서비스를 받고 있으며, 네트워크 지연시간 감소를 위해 ES-y로 MUD의 컨테이너를 이관하려 한다. ES-y로 이관된 컨테이너를 CT-y라 하자. 상태 전송 이관기법은 ES-x에서 서비스 중인 CT-x의 상태를 ES-y로 전송하고, 이를 통해 ES-y에서 새로이 구동하는 컨테이너 CT-y가 기존의 컨테이너 CT-x와 동일한 상태를 갖게 한다. 반면, 상태 재구성 기법은 ES-y로 상태 자체를 전송하지 않고, ES-x의 컨테이너 CT-x과 동일한 상태를 만드는 방법을 전송한다. 예를 들어, CT-x가 실행한 명령어(Command Log 또는 Command Trace)를 시간순으로 정렬한 로그가 대표적인 예이다.

Puliafito[8]는 컨테이너를 이관하는 세 가지 상태 전송 기법인 Pre-Copy, Post-Copy, Hybrid-Copy를 구현하고 이관시간, 서비스 중단 시간, 송수신 트래픽 총량을 기준으로 성능 비교를 수행했다. Pre-Copy 기법은 CT-x의 현재 상태를 최대한 ES-y로 전송하고 CT-x를 중단한 다음, ES-y에서 CT-y를 시작하는 방식이다. Post- Copy는 CT-x을 중단한 다음 CT-x의 실행상태만 ES-y로 전송하여 ES-y에서 CT-y를 실행하고, CT-y에서 Page Fault가 발생하면 CT-x으로부터 메모리 상태를 전송받는 Lazy Migration 기법이다. 마지막으로 Hybrid-Copy는 Pre-Copy와 Post-Copy 기법을 결합한 형태이다. Puliafito의 논문에서 다루는 이관기법은 파일 시스템의 상태 변화를 전송하지 않는다는 점 이외에도, Pre-Copy에서 전송되지 않는 상태가 발생할 수 있다는 점, 그리고 Post-Copy에서 잦은 Page Fault로 인한 잦은 상태전송 및 서비스 응답시간 증가가 발생할 수 있는 등의 제약이 있다.

Liu[13]는 호스트 가상화 기반의 가상 머신을 이관하기 위한 무중단 이관기법인 Trace/Replay 기법을 제안했고, Yu[14]는 컨테이너 가상화 환경에서 유사한 기법을 적용하는 연구를 수행했다. 해당 기법은 상태 전송과 상태 재구성의 혼합된 형태로 볼 수 있다. Liu 및 Yu가 제안한 기법의 동작 방식은 다음과 같다. 먼저, CT-x의 상태 및 파일 시스템 변경사항을 파일로 기록하고 ES-y로 전송하며, CT-x를 중단하지 않는다. ES-y는 수신한 파일로부터 CT-y를 실행한다. 파일 전송 이후에도 CT-x는 당분간 서비스를 지속하고 수신한 이벤트를 로그로 기록(trace)하여 CT-y로 전송한다. CT-y는 수신한 로그를 재현(replay)하고, 이 과정이 반복된다. CT-x가 전송하는 로그의 크기가 일정 수준 이하로 작아지면 로그 전송 후 CT-x를 정지하고, CT-y는 수신한 로그를 재현한 뒤 앞으로의 서비스는 CT-y가 처리한다. 단, 응용 분야 또는 운영 환경에 따라 시스템 이벤트를 로그로 기록하는 것이 어려울 수 있으며, 이를 위해 ReVirt와 같은 별도의 프로그램을 사용해야 할 수 있다. 참고로, Liu는 ReVirt를 사용한 이벤트 로깅 시 약 8%의 성능 오버헤드가 발생했다고 보고했다.

위에서 살펴본 기법 모두 이관 과정에서 일시적인 서비스 중단이 발생하며, 이로 인해 MUD의 오프로딩 서비스 요청이 분실되거나 응답시간이 길어지는 등 서비스 품질을 저하할 수 있다는 제약이 있다. 게다가, 위에서 언급한 기법뿐 아니라 일반적인 무중단 이관기법[15,16]의 공통적인 특징은, CT-x를 중단한 후 CT-y가 정해진 준비과정을 거치고 나서 서비스를 이어가는 방식으로 동작한다는 것이다. 즉, CT-y가 정상적으로 서비스를 제공할 준비가 되었는지를 확인하기 전에 CT-x가 중단된다. 따라서, CT-x를 종료한 이후 ES-y가 CT-y를 구동 또는 준비하는 과정에서 장애가 발생하면 해당 사용자의 오프로딩 서비스가 중단될 수 있고, 기존 연구는 이에 대한 고려가 부족하다.

본 논문에서 구현하는 이관기법은 Liu[13]의 무중단 이관기법을 기반으로 하며, 위에서 언급한 한계점을 개선하기 위해 CT-x의 중단을 지연하는 방식을 사용한다. 먼저, CT-x의 프로세스 실행상태, 메모리 상태 및 파일 시스템 변경상태를 전송한다. CT-y가 준비되기 전까지 CT-x가 사용자의 오프로딩 서비스 요청을 처리할 수 있도록 요청 메시지를 CT-x로 포워딩하고, CT-y가 CT-x와 완벽하게 동기화된 것을 확인한 후에

야 CT-x를 중단하는 지연 방식을 구현한다. CT-x는 이관이 완료되는 마지막 순간까지 중단되지 않기 때문에 이관 중 ES-y 또는 CT-y에 장애가 발생할 경우, CT-x를 통해 서비스를 지속할 수 있다.

Ⅲ. 무중단 서비스 이관 시스템 구현

본 논문에서 가정하는 통신 환경 및 네트워크 구성은 [그림 1]과 같다.

MUD-m은 IEEE 802.11 Access Point (AP)를 통해 네트워크에 연결되며, 각 AP는 하나의 ES와 유선으로 연결되어있다고 가정한다[4]. 또한, 사용자 MUD-x는 AP-x를 통해 처음으로 네트워크에 연결되는 상황을 가정한다. AP-x에 접속한 MUD-m는 지연시간 최소화를 위해 ES-x에 독립적으로 사용할 컨테이너 CT-x(m)을 생성한 후 이를 통해 오프로딩 서비스를 받는다. MUD-m이 AP-x의 서비스 지역에서 벗어나고 AP-y의 서비스 지역에 가까워짐에 따라 AP-y로의 Handoff가 발생한다[17].

IEEE 802.11 handoff 절차는 크게 두 가지로 나뉘는데, 후보 AP를 찾는 Discovery 단계와 선택된 AP로 접속을 요청하는 Reassociation 단계이다. MUD-m은 새로운 AP에게 접속을 요청하기 위해 IEEE 802.11에서 정의한 REASSOCIATION REQUEST 메시지를 AP-y로 전달한다. 해당 메시지를 수신한 AP-y는 MUD-m의 상태정보(예: 자격증명, 계정정보 등)를 AP-x로부터 전달받기 위해 AP-x에게 IAPP(Inter Access Point Protocol) 메시지를 보낸다. AP-x는 IAPP 응답 메시지를 전송할 때, MUD-m이 오프로딩 서비스 사용자라는 정보를 포함시킬 수 있다. 따라서, REASSOCIATION REQUEST 메시지 수신을 통해 AP-y는 MUD-m의 연결요청을 인지하고, IAPP 메시지 교환을 통해 AP-x(ES-x)는 MUD-m의 연결해제를 인지하며 IAPP 메시지 교환을 통해 AP-y(ES-y)는 MUD-m이 오프로딩 서비스 사용자라는 것을 인지한다.

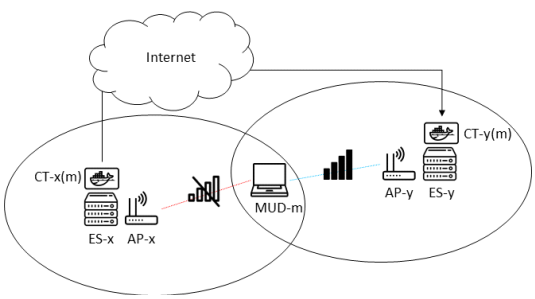

그림 1. 통신 환경 및 네트워크 구성도
Fig. 1. Communication and network diagram

오프로딩 서비스의 지연시간을 줄이기 위해, AP-y로 Handoff가 발생하면 ES-x에서 구동 중인 CT-x(m)을 반드시 ES-y로 이관한다고 가정한다. 컨테이너 이관으로 인해 발생하는 트래픽은 인터넷 또는 Wide Area Network(WAN)를 통해 전송된다. [그림 1]에 도식된 환경은 두 개의 AP로 구성된 단순한 네트워크이나, 본 논문에서 구현하는 이관 시스템은 세 개 또는 그 이상의 AP로 구성된 네트워크에도 동일하게 적용할 수 있다.

본 논문은 도커 컨테이너 가상화 플랫폼에서 동작하는 EC을 위한 컨테이너 이관기법을 구현하며, 구현에 사용한 도커 버전은 17.03.2-ce이다. 본 논문에서 구현하는 이관기법은 총 두 가지(C-Migr 및 P-Migr)이며 두 기법 모두 상태를 보존하는 Stateful 이관기법이다. 즉, 새로이 이관된 CT-y(m) 컨테이너는 원래의 CT-x(m) 컨테이너와 완벽히 동일한 파일 시스템, 프로세스 실행상태 및 메모리 상태를 가진다. C-Migr 및 P-Migr의 동작 순서를 [표 1]에 요약했다.

T1은 도커의 체크포인트로 구현했다. 체크포인트 기능은 아직 실험적(Experimental) 기능으로 분류되며, 리눅스 CRIU(Checkpoint/Restore In Userspace) 도구를 백엔드(backend)로 사용한다. CRIU는 현재 구동 중인 컨테이너를 정지하고 실행상태를 이진파일 형태로 파일 시스템에 저장한다. 참고로, 본 연구에서 구현에 사용한 CRIU 버전은 v3.15이다. T2 및 T4에서 파일을 전송하기 위해 리눅스 scp (Secure Copy)

표 1. C-Migr 및 P-Migr 이관 기법의 동작 순서
Table 1. Sequence of operations of C-Migr and P-Migr migrations

T1) CT-x(m)의 프로세스 실행상태 및 메모리 상태를 파일로 저장
T2) T1에서 저장한 파일을 ES-y로 전송
T3) CT-x(m)의 파일 시스템 변경 내용 기록
T4) 파일 시스템 변경 내용을 ES-y로 전송
T5) ES-y는 CT-y(m) 컨테이너를 생성하고, T2 및 T4에서 수신한 파일을 사용하여 컨테이너 복원 시작
T6) AP-y는 수신한 사용자의 오프로딩 서비스 요청을 CT-x(m)으로 포워딩하고 사본 저장
T7) T5 과정이 완료되어 CT-y(m)이 준비상태로 전환되면 AP-y가 저장한 서비스 요청 사본을 순서대로 CT-y(m)에 전달하여 실행
T8) AP-y가 저장한 서비스 요청 사본을 모두 CT-y(m)에서 실행한 것을 확인 후, 앞으로의 서비스 요청을 CT-y(m)으로 전달하고 CT-x(m)으로 컨테이너 중단 요청 전송

도구를 사용했다. SCP는 SSH 프로토콜을 사용하여 보안 위협에 안전한 파일 전송을 지원한다.

T4는 CT-x(m)이 파일 시스템에 가한 변경사항을 추적하고 변경된 파일을 추출하는데, 이는 두 가지 방식으로 구현할 수 있다. 첫 번째 방식은 C-Migr 기법에서 사용하는 방식으로, 구동 중인 컨테이너 전체를 이미지로 변환하는 것이다. 도커 컨테이너는 하나 또는 그 이상의 Read-Only Layer로 구성된 이미지를 통해 생성되고, 컨테이너 실행 중에 발생하는 파일 시스템의 변경 내용은 최상위 계층인 Container Layer에 기록된다. 따라서 실행 중인 컨테이너 전체를 이미지로 변환할 경우 Container Layer뿐 아니라 Read-Only Layer가 모두 포함된 새로운 이미지를 생성할 수 있다. 두 번째 방식은 Container Layer에 기록된 내용만 추출하는 것으로, 이는 P-Migr 기법이 사용하는 방법이다. ES-x에서 CT-x(m) 컨테이너를 생성할 때 사용한 것과 동일한 이미지가 ES-y에 이미 저장되어 있거나, 해당 이미지를 신속하게 다운(Pull)받을 수 있는 Private Repository가 있는 경우 Read-Only Layer를 ES-x로부터 수신할 필요가 없다. 따라서, 이 경우 Container Layer만 ES-y로 전송하면 이관 시간(Migration Time)을 크게 절약할 수 있다. Container Layer에 기록된 파일 시스템 변경 내용을 확인하는 방법은 두 가지가 있는데, 첫 번째 방법은 docker diff 명령으로 변경된 파일을 추적하고 docker cp 명령으로 해당 파일을 추출하는 것이고, 두 번째 방법은 Container Layer에 저장된 파일 시스템에 직접 접근하는 것이다. 본 연구에서는 신속한 Container Layer 추출을 위해 Overlay2 스토리지 드라이버의 UpperDir이 가리키는 경로에 저장된 Container Layer 파일을 직접 접근하는 두 번째 방법을 사용한다.

T5는 ES-y에서 새로운 컨테이너 CT-y(m)을 생성하는데, C-Migr과 P-Migr의 동작 방식이 서로 다르다. C-Migr은 Container Layer가 저장된 전체(Complete) 이미지를 수신하므로, 수신한 이미지로 컨테이너를 생성하고, 수신한 체크포인트로 컨테이너를 구동하면 CT-x(m)과 동일한 상태를 가지는 컨테이너 CT-y(m)을 복원할 수 있다. 반면, P-Migr은 Container Layer만 포함하는 부분적(Partial) 이미지를 전송받는다. 따라서, CT-x(m) 컨테이너를 생성할 때 사용한 원본 이미지와 동일한 이미지로 컨테이너를 생성하고, 생성한 컨테이너의 Container Layer에 수신받은 Container Layer 내용을 저장한다. 마지막으로, 수신한 체크포인트로 컨테이너를 구동하여 CT-x(m)과 동일한 상태를 가지는 컨테이너 CT-y(m)를 복원한다.

본 논문에서 구현하는 두 가지 이관기법은 이관이 진행되는 중에 서비스 중단이 발생하지 않는 Live 이관기법이다. 이를 구현하기 위해 컨테이너 무중단 및 패킷 포워딩 기법을 사용한다. MUD-m이 AP-y로 Handoff 되면 컨테이너 이관이 시작되고, 오프로딩 요청을 비롯해 MUD-m이 보내는 모든 패킷은 AP-y가 수신한다. 하지만 컨테이너 이관에는 일정 시간이 소요되며, MUD-m이 AP-y로 Handoff 된 직후에는 CT-y(m)이 준비되지 않은 상태이다. 본 논문에서 구현하는 이관기법은 이관이 진행 중인 상황에서도 끊김없는 오프로딩 서비스를 제공하기 위해, CT-y(m)이 완벽히 준비되기 전까지 CT-x(m)를 중단하지 않고 AP-y가 수신한 오프로딩 요청 메시지를 AP-x로 포워딩하여, 아직 구동 중인 CT-x(m)을 통해 오프로딩 서비스를 지속한다(T6). 도커 체크포인트 기능을 사용하면 체크포인트 생성과 동시에 컨테이너를 중단하게 되어 있는데, "--leave-running=true" 옵션을 사용하면 컨테이너 중단을 방지할 수 있다. 또한, C-Migr 기법은 컨테이너 전체를 새로운 이미지로 생성하기 위해 "commit" 및 "save" 명령을 사용하고, 이때, 컨테이너가 중단되는데, "--pause=false" 옵션을 사용해 컨테이너 중단을 방지할 수 있다. 이관이 시작된 후, AP-y는 MUD-m의 오프로딩 요청을 AP-x로 포워딩하여 CT-x(m)로부터 오프로딩 서비스를 받게 하고, ES-y에서 CT-y(m)이 구동된 후 서비스 준비가 완료되면, CT-y(m)은 자신의 상태를 AP-y로 전송한다. AP-y는 해당 메시지를 AP-x를 통해 ES-x로 전송하고, 이때 ES-x는 CT-x(m)을 종료하여 CT-x(m)가 점유하던 시스템 자원을 반납한다.

CT-x(m)가 체크포인트를 생성하여 전송하고, 파일 시스템 전체(C-Migr) 또는 일부분(P-Migr)을 전송한 뒤, 일정 시간 후에 CT-y(m)가 서비스 준비를 완료하는데, 그동안 CT-x(m)에 상태 변화(즉, 파일 시스템의 변경, 프로세스 실행 상태 변화, 메모리 상태 변화)가 발생할 수 있다. 이러한 상태 변화를 추적하고 CT-y(m)에 반영하기 위해 Liu가 제안한 Trace/Replay 기법을 사용한다. 즉, CT-x(m)이 이관을 위해 상태정보를 ES-y로 전송한 이후로부터 CT-y(m)이 완벽히 준비되기 직전까지 AP-y는 MUD-m으로부터 수신한 모든 오프로딩 요청을 AP-x로 포워딩하는 동시에 선입선출 방식으로 동작하는 로컬 큐 자료구조에 요청 메시지 복사본을 저장한다. CT-y(m) 생성 후 체크포인트로 상태를 복원한 뒤 AP-y로 시그널을 전송하고, 이를 수신한 AP-y는 큐

에 저장된 명령을 CT-y(m)으로 전송하여 컨테이너가 해당 요청을 실행하게 한다(T7). AP-y의 큐에 저장된 모든 요청이 CT-y(m)에 의해 처리되면 AP-y는 AP-x에게 이 사실을 알려서 CT-x(m)이 중지되도록 하고, 앞으로 수신하는 모든 MUD-m의 오프로딩 요청을 CT-y(m)으로 전달한다(T8).

[그림 2]는 본 논문에서 구현하는 각 모듈의 기능 블록 다이어그램이다.

이관기법의 검증 및 성능 분석을 위해 MUD, AP, ES에서 발생하는 각종 이벤트를 전송하도록 구현했고, 이를 저장하기 위해 Event Log Repository(LogRepo)를 개발했다. MUD, AP, ES는 주요 이벤트 발생 시 현재 시각과 이벤트 내용을 기록한 로그를 생성하고, 이를 LogRepo로 전송한다. 시스템별 기준 시간 차이로 인한 오류를 없애기 위해 MUD, AP, ES의 시간을 동기화했고, ES에서 컨테이너 생성 시 TZ 옵션으로 Time Zone을 동일하게 설정했다. Storage Management 기능 블록은 수신한 로그를 저장하는 방식을 결정하며, 기본값으로 파일에 기록한다. File Writer 기능 블록은 수신한 로그값을 Storage Management 블록이 설정한 저장소에 저장한다. AP의 MUD Management는 MUD가 AP로 Association 또는 Reassociation 하는 과정을 처리한다. Replay Buffer는 이관 중 발생하는 오프로딩 서비스 요청을 보관하는 기능을 수행하며, Packet Forwarder는 이관이 완료되기까지 수신한 오프로딩 서비스 요청을 다른 AP로 포워딩하는 기능을 수행한다. Log Reporter 기능 블록은 AP, ES, MUD에 모두 존재하며, 이벤트 발생 시 해당 로그를 LogRepo로 전달하는 기능을 수행한다. ES는 컨테이너의 생성/관리/삭제를 위해 Container Management 기능 블록을 사용하고, 컨테이너 이관은 Migration Controller 기능 블록이 처리한다. MUD는 사용자의 이동성을 에뮬레이트하는 Mobility Management 기능 블록, 오프로딩 서비스 요청을 생성하는 Offloading Request Generator 기능 블록 등으로 구성된다.

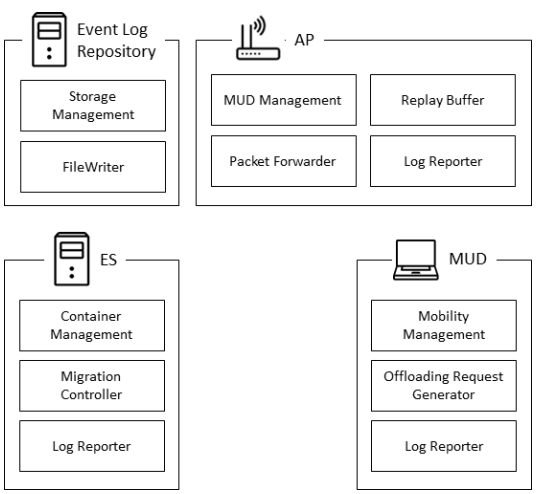

그림 2. 본 논문에서 구현하는 각 모듈의 기능 블록 다이어그램
Fig. 2. Function block diagrams of the considered migration methods

Ⅳ. 실험 및 결과

[그림 1]에 도식화된 네트워크 환경 및 [그림 2]의 기능 블록을 기성의 하드웨어에 구성했고, 구현 및 실험에 사용한 네트워크 구성은 [그림 3]과 같다.

IEEE 802.3 LAN 및 IEEE 802.11ac WLAN을 지원하는 ipTIME A8004NS-M 공유기를 통해 네트워크를 구성하고, 동일한 사양의 PC 3대 (Intel CoreTM i5-8500 CPU, 16GB RAM, 128GB SSD and 1000Mbps Ethernet network interface card) 및 노트북 1대 (Intel CoreTM i5-1035G4 CPU, 8GB RAM, 256GB SSD, and IEEE 802.11ac- compatible WiFi network interface card)를 사용했다.

PC#1과 PC#2는 각각 AP-x, ES-x와 AP-y, ES-y를 구현하기 위해 사용했다. 원하는 시점에 AP간 Handoff 이벤트를 발생시키기 위해 AP를 에뮬레이트하는 프로그램을 PC#1과 PC#2에 구현했다. MUD-m의 기능은 Laptop#1에 구현했고, 연결된(associated) AP로 초당 1회의 속도로 오프로딩 연산 요청을 송신하도록 설정했다. 실험이 시작되면 AP-x에 MUD-m이 접속하고 ES-x에 컨테이너를 생성하여 오프로딩 서비스를 시작한다. 실험 시작 후, 50초가 지나면(즉,

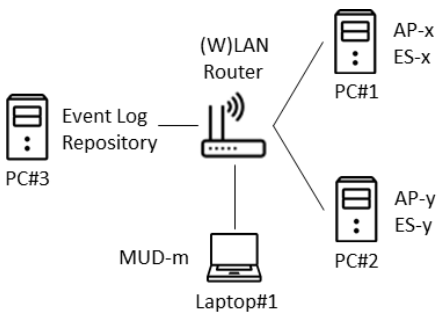

그림 3. 구현 및 실험을 위한 네트워크 환경 구성도
Fig. 3. Network configuration for implementation and experiment

50번째 오프로딩 요청을 송신할 때) AP-y로의 Handoff가 발생하고 동시에 ES-y로의 컨테이너 이관이 시작된다. Event Log Repository(LogRepo)는 PC#3에 구현했다.

Python:3.8 공식 이미지를 베이스 이미지로 하여 연산 오프로딩을 구현한 Python 응용 프로그램을 포함하는 커스텀 이미지를 생성했고, 이를 이용하여 ES에서 컨테이너를 생성했다(참고: [그림 4]).

MUD-m은 1초에 한 번씩 오프로딩 요청을 보내는데, 송신 패킷에는 0에서 시작해서 1씩 증가하는 순서번호(Sequence Number)가 포함되어 있다. 컨테이너는 오프로딩 요청을 받으면 해당하는 연산을 수행하고, 이에 대한 어플리케이션 계층 ACK (Acknowledgement)를 회신한다. 이때, ACK에는 해당하는 오프로딩 요청 패킷에 기록된 순서번호와 동일한 순서번호를 포함하도록 했다. 이를 통해 MUD가 송신한 각각의 요청이 처리되었는지 여부를 확인할 수 있다. 또한, MUD가 오프로딩 요청을 송신한 시간으로부터 해당 ACK를 수신한 시간 간격을 측정하여 서비스 응답시간을 측정했다. MUD-m이 AP-y로 Handoff 되면 AP-y는 AP-x를 통해 컨테이너 이관요청을 보내고, 이를 수신한 ES-x는 이관작업을 시작하는데, 이때 ES-x는 LogRepo로 이관 시작을 알린다. 또한, ES-y에서 CT-y(m)이 완전히 준비되면 ES-y는 LogRepo로 이관 완료를 알린다. 두 이벤트의 시간 간격을 계산하여 이관시간을 측정할 수 있다.

[그림 3]에 도식화된 실험 환경은 [그림 1] 네트워크 환경을 구현한 것이다. [그림 1]에서는 AP-x와 AP-y가 인터넷 또는 WAN을 통해 연결되고, 이로 인해 응답시간이 길거나 전송속도(대역폭)가 느릴 수 있다. 하지만, 실 구현환경 [그림 3]에서 AP 기능을 하는 PC#1과 PC#2는 (W)LAN에 연결되어있어 항상 고속통신이 가능하다. PC#1과 PC#2 사이에서 측정한 Round-Trip Time (RTT)는 수십 ms 수준이고 전송대역폭은 약 1Gbps였다. 인터넷을 통해 데이터가 송수신되는 상황을 재현하기 위해 리눅스 tc(traffic control) 도구를 사용하여 AP간 데이터 송수신 시 RTT를 200ms 수준으로 높였고, AP간 통신 대역폭을 10MBps로 설정했다. 실험은 200초간 5회 실시했고, Outlier의 영향을 최소화하기 위해 평균값을 사용해 이관시간 및 서비스 응답시간을 측정했다.

[그림 5]는 LogRepo에 저장된 로그 중 일부를 캡처한 화면이다. 로그에서 MUD는 USER로, AP-x와 ES-x는 각각 AP-1, EdgeServer-1로, AP-y와 ES-y는 각각 AP-2, EdgeServer-2로 표시된다. 로그를 확인한 결과 MUD가 보낸 모든 요청이 ES에서 처리되었고, 그 응답을 MUD가 수신한 것을 확인하였다(즉, 서비스 단절 없음). 또한, 이관이 진행되는 중에는 응답시간이 크게 증가한 것을 볼 수 있는데, 이것은 AP-y가 수신한 오프로딩 요청을 AP-x로 포워딩하고, 리눅스 tc 도구를 활용하여 AP간 RTT를 높게 설정했기 때문이다.

[그림 5(a)]는 실험 시작 직후에 수집된 로그이다. USER로 표기된 MUD는 순서번호를 포함하여 오프로딩 서비스 요청(SVCQ, Service Request)을 연결된 AP로 보내고, 해당 요청이 처리된 경우 동일한 순서번호를 포함한 응답(SCVR, Service Response)을 AP를 통해 ES로부터 수신하는 것을 확인할 수 있다.

[그림 5(b)]는 컨테이너 이관이 시작된 시점에 수집된 로그이다. MUD(USER)는 AP-y(AP-2)로 Handoff 하기 위해 REASSOCIATION REQUEST를 AP-y로 전송하고, 이때 AP-x(AP-1)에서 컨테이너 이관이 시작되었음을 알린다. 컨테이너 이관 전에는 응

```
FROM python:3.8

ENV PYTHONUNBUFFERED=1

COPY offloading-service.py /tmp/service.py
COPY common.py /tmp/common.py
COPY application.py /tmp/application.py

ENTRYPOINT ["/usr/local/bin/python3", "/tmp/service.py"]
```

그림 4. 커스텀 이미지를 생성하기 위한 Dockerfile 구성
Fig. 4. Dockerfile for generating a custom image used in this study

```
2021-09-02-11-44-48-794958 USER AP-1 USER ASSOCIATION REQUEST (sent)
2021-09-02-11-44-49-327071 AP-1 192.168.0.106 USER ASSOCIATION
REQUEST (recvd)
2021-09-02-11-44-48-810684 USER 192.168.0.105 AP-1 ES1-READY (recvd)
2021-09-02-11-44-53-839044 USER AP-1 USER SVCQ 0 (sent)
2021-09-02-11-44-54-042137 USER 192.168.0.105 AP-1 SVCR 0 (recvd)
2021-09-02-11-44-54-854571 USER AP-1 USER SVCQ 1 (sent)
2021-09-02-11-44-54-870203 USER 192.168.0.105 AP-1 SVCR 1 (recvd)
```

그림 5. (a) 실험 시작 시 수집된 로그
Fig. 5. (a) Logs collected at the beginning of an experiment

```
2021-09-02-11-45-41-411788 USER AP-1 USER SVCQ 47 (sent)
2021-09-02-11-45-41-427113 USER 192.168.0.105 AP-1 SVCR 47 (recvd)
2021-09-02-11-45-42-427316 USER AP-1 USER SVCQ 48 (sent)
2021-09-02-11-45-42-442947 USER 192.168.0.105 AP-1 SVCR 48 (recvd)
2021-09-02-11-45-43-442839 USER AP-1 USER SVCQ 49 (sent)
2021-09-02-11-45-43-458164 USER 192.168.0.105 AP-1 SVCR 49 (recvd)
2021-09-02-11-45-43-458164 USER AP-2 USER REASSOCIATION REQUEST (sent)
2021-09-02-11-45-43-487237 AP-2 192.168.0.106 USER REASSOCIATION
REQUEST (recvd)
2021-09-02-11-45-43-990344 AP-1 192.168.0.106 USER BYEE (recvd)
2021-09-02-11-45-44-133129 AP-1 ES-1 migr begins
2021-09-02-11-45-44-449376 USER AP-2 USER SVCQ 50 (sent)
2021-09-02-11-45-44-683718 USER 192.168.0.101 AP-2 SVCR 50 (recvd)
2021-09-02-11-45-45-464866 USER AP-2 USER SVCQ 51 (sent)
2021-09-02-11-45-45-714876 USER 192.168.0.101 AP-2 SVCR 51 (recvd)
```

그림 5. (b) 컨테이너 이관이 시작된 시점에 수집된 로그
Fig. 5. (b) Logs collected when the migration has just started

```
2021-09-02-11-47-49-757618 USER AP-2 USER SVCQ 174 (sent)
2021-09-02-11-47-49-993802 USER 192.168.0.101 AP-2 SVCR 174 (recvd)
2021-09-02-11-47-50-759310 USER AP-2 USER SVCQ 175 (sent)
2021-09-02-11-47-50-986183 USER 192.168.0.101 AP-2 SVCR 175 (recvd)
2021-09-02-11-47-51-769312 USER AP-2 USER SVCQ 176 (sent)
2021-09-02-11-47-51-892493 AP-2 AP-2 migr finished
2021-09-02-11-47-52-115187 USER 192.168.0.101 AP-2 SVCR 176 (recvd)
2021-09-02-11-47-52-784449 USER AP-2 USER SVCQ 177 (sent)
2021-09-02-11-47-52-800088 USER 192.168.0.101 AP-2 SVCR 177 (recvd)
2021-09-02-11-47-53-792356 USER AP-2 USER SVCQ 178 (sent)
2021-09-02-11-47-53-807970 USER 192.168.0.101 AP-2 SVCR 178 (recvd)
```

그림 5. (c) 컨테이너 이관이 완료된 시점에 수집된 로그
Fig. 5. (c) Logs collected when the migration has just finished

답시간이 수십 ms인 반면, 이관이 시작된 이후에는 수백 ms로 증가하였고, 이는 AP-y(AP-2)가 수신한 오프로딩 서비스 요청을 AP-x(AP-1)로 포워딩하기 때문이다.

[그림 5(c)]는 컨테이너 이관이 완료된 시점에 수집된 로그이다. ES-y는 CT-y(m)가 완전히 준비되면 AP-y(AP-2)에게 시그널을 보낸다. 컨테이너 이관이 완료되기 전에는 응답시간이 수백 ms인 반면, 이관이 완료된 후에는 응답시간이 수십 ms로 감소한 것을 볼 수 있다. 이는 더이상 오프로딩 서비스 요청을 AP-x(AP-1)로 포워딩하지 않기 때문이다.

실험을 위해 제작한 이미지는 Python:3.8을 베이스 이미지로 하고, 베이스 이미지의 크기는 약 909MB이다. 컨테이너 이관이 시작되는 시점을 기준으로, 컨테이너의 메모리 사용량은 약 32.7MB이고 Container Layer에 기록된 수정 파일의 크기는 총 20MB이다. 체크포인트를 사용해 프로세스 및 메모리 상태를 저장한 결과 그 크기는 약 29MB로, 실제 사용 중인 메모리 크기보다 약간 작았다. C-Migr의 경우 컨테이너 전체 및 체크포인트를 전송하므로, 총전송량은 909MB(베이스 이미지), 20MB(Container Layer), 29MB(체크포인트)를 합한 958MB이다. 반면, P-Migr은 원본 이미지를 전송하지 않으므로, 총전송량은 약 49MB이다. AP간 대역폭이 10MBps인 것을 고려할 때, C-Migr 기법의 이관시간이 P-Migr에 비해 클 것이며, 이는 대부분의 이관시간을 베이스 이미지 전송에 사용하기 때문이다.

실험 결과, C-Migr 기법의 이관시간은 평균 124.90초이고, P-Migr 기법의 이관시간은 평균 6.91초였다. 또한, 각 오프로딩 서비스 요청에 대한 응답을 받기까지 C-Migr은 평균 0.2029초, P-Migr은 평균 0.0289초가 걸렸다. 즉, P-Migr 기법은 C-Migr에 비해 이관시간을 94.47% 개선했고, 평균 응답시간을 85.47% 개선했다. 위에서 살펴본 바와 같이, C-Migr은 Read-Only Layers를 전송하며 그 크기가 909MB이므로 컨테이너 이관에 상당한 시간을 소모했다. 반면, P-Migr은 수십 MB의 데이터만 전송하므로 이관시간이 상대적으로 짧았다. 참고로, 두 기법 모두 이관 중에 서비스 단절이 발생하지 않았다.

[그림 6]과 [그림 7]은 C-Migr과 P-Migr을 사용하여 컨테이너를 이관할 때, 각 오프로딩 서비스 요청에 대한 응답시간을 측정한 것이다. 총 5회 실험을 반복했고, 이때 측정된 최솟값(파란 실선), 최댓값(검은 실선), 평균값(붉은 실선)을 표시했다. 실험이 시작된 후, MUD가 50번째 오프로딩 요청을 송신하는 시점에 AP-y로 Handoff가 발생하고 컨테이너 이관이 시작된다. 하지만, AP-y의 ES-y로 컨테이너가 완벽히 이관되기까지 상당한 시간이 소요되며, 그동안은 AP-x의 ES-x에서 동작 중인 CT-x(m)이 오프로딩 서비스를 제공한다. 이때, AP간 패킷 포워딩으로 인해 오프로딩 서비스의 응답시간이 급격히 증가하며, 이 현상은 ES-y로 컨테이너 이관이 완료되는 시점까지 지속된다. [그림 6]과 [그림 7]에서 max에 해당하는 결과를 보면 tc로 설정한 RTT를 크게 상회 하는 서비스 지연

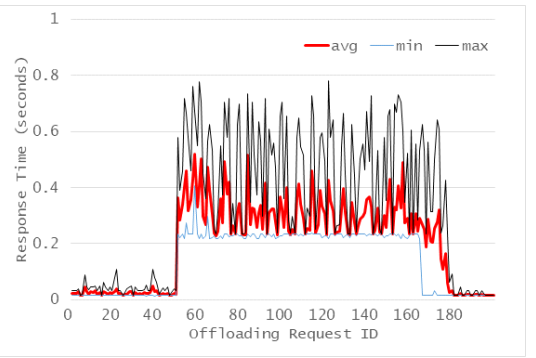

그림 6. C-Migr 기법을 사용한 컨테이너 이관 시 측정한 각 오프로딩 요청의 응답시간
Fig. 6. Per-request response time with C-Migr migration

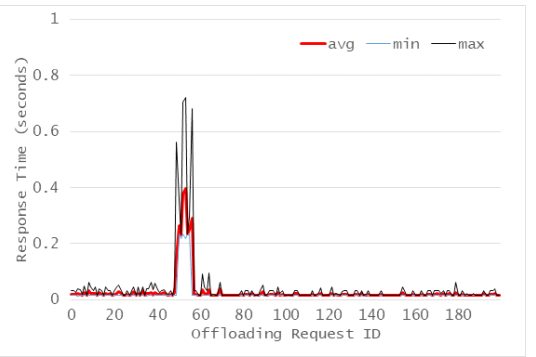

그림 7. P-Migr 기법을 사용한 컨테이너 이관 시 측정한 각 오프로딩 요청의 응답시간
Fig. 7. Per-request response time with P-Migr migration

시간이 발생한 것을 볼 수 있다. 이는 실험 당시에 발생한 백그라운드 트래픽 및 PC#1과 PC#2에서 발생한 백그라운드 프로세스의 영향 때문이다.

실험 결과를 토대로 P-Migr이 C-Migr에 비해 우월한 성능을 보임을 알 수 있다. 하지만, P-Migr이 항상 C-Migr에 비해 선호되는 것은 아니다. 위에서 언급한 바와 같이, C-Migr이 Container Layer만 전송할 수 있었던 것은 ES-x가 CT-x(m)을 구동하기 위해 사용한 베이스 이미지가 ES-y에 이미 저장되어 있거나, 신속하게 다운받을 수 있다는 가정이 있었기 때문이다. 하지만, 예를 들어 CT-x(m)을 구동하기 위해 사용한 이미지가 ES-x에만 존재하는 커스텀(Custom) 이미지이고, Public 또는 Private Repository에서 구할 수 없는 이미지인 경우, P-Migr 기법은 사용할 수 없다. 또한, Repository에서 구할 수 있는 이미지라 할지라도, 해당 이미지를 다운로드 하는데 걸리는 시간이 C-Migr을 통한 이관시간보다 큰 경우, 오히려 C-Migr이 우수한 성능을 보일 것이다.

Ⅴ. 결론 및 향후 연구 방향

본 논문에서는 상태를 보존하며 서비스 중단이 발생하지 않는 컨테이너 이관기법의 구현, 실험 및 검증을 진행했다. 실행 중인 컨테이너 전체를 이관하는 C-Migr과 Container Layer만을 이관하는 P-Migr 기법은 모두 상태를 전송하는 방식의 이관기법이나, Read-Only Layers를 전송하는지 여부에 따라 구분된다. P-Migr 기법은 C-Migr에 비해 이관시간을 94.47% 개선했고, 평균 응답시간을 85.47% 개선했으나, 경우에 따라 C-Migr만 사용 가능하거나, C-Migr 기법이 P-Migr 기법에 비해 우수한 성능을 보일 수 있다. 본 논문에서 구현한 이관기법은 파일 시스템의 변경 내용뿐 아니라 프로세스 상태 및 메모리 상태를 보존하며, 따라서 상태 기반의 서비스에 활용할 수 있다. 또한, 이관이 진행되는 중에 서비스 중단이 발생하지 않으므로 서비스 중단에 민감한 분야에 활용할 수 있다.

향후 연구 방향은 다음과 같다. 상태를 전송하는 이관기법이 아닌 상태 재구성에 기반한 이관기법을 구현하고, 실험을 통해 검증할 계획이다. 특히, 상태 재구성에 필수적인 Command Trace/Log를 효과적으로 추출하는 기법에 관한 연구를 수행할 계획이다. 또한, 본 논문에서는 AP가 밀집하지 않은 네트워크 상황을 가정했으나, 현실에서는 다수의 AP가 밀집하게 설치되어 다수 AP의 서비스 영역(Coverage)이 중첩되는 경우가 상당수 존재한다. 이때, 최적의 AP를 선택하여 Handoff 함으로써 이관시간 및 이관 중에 발생하는 서비스 지연시간을 최소화하는 방법에 관한 연구를 수행할 계획이다.

References

[1] R. L. Grossman, "The case for cloud computing," *IT Prof.*, vol. 11, no. 2, pp. 23-27, 2009.

[2] K., Karthik, and Y.-H. Lu. "Cloud computing for mobile users: Can offloading computation save energy?." *IEEE Comput.*, vol. 43, no. 4 pp.. 51-56, Apr. 2010.

[3] W. Shi and S. Dustdar, "The promise of edge computing," *IEEE Comput.*, vol. 49, no. 5, pp. 78-81, May 2016.

[4] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, "Mobile edge computing: A survey," *IEEE Internet Things J.*, vol. 5, no. 1, pp. 450-465, Feb. 2018.

[5] M. Satyanarayanan, "The emergence of edge computing," *IEEE Comput.*, vol. 50, no. 1, pp. 30-39, Jan. 2017.

[6] X. Sun and N. Ansari, "EdgeIoT: Mobile edge computing for the Internet of Things," *IEEE Commun. Mag.*, vol. 54, no. 12, pp. 22-29, Dec. 2016.

[7] J. S. Ah, "Edge computing and vehicles: Opportunities and challenges for the future," *J. KICS*, vol. 46, no. 5, pp. 834-847, 2021.

[8] C. Puliafito, C. Vallati, E. Mingozzi, G. Merlino, F. Longo, and A. Puliafito, "Container migration in the fog: A performance evaluation," *Sensors*, vol. 19, no. 7, pp. 1-22, Mar. 2019.

[9] D. Bernstein, "Containers and cloud: From LXC to docker to kubernetes," *IEEE Cloud Comput.*, vol. 1, no. 3, pp. 81-84, Sep. 2014.

[10] *Docker*, https://www.docker.com/.

[11] K. Govindaraj and A. Artemenko, "Container live migration for latency critical industrial applications on edge computing," in *Proc. IEEE ETFA 2018*, pp. 83-90, Turin, Italy, 2018.

[12] P. Saha and A. Beltre, “Evaluation of docker containers for scientific workloads in the cloud,” in *Proc. ACM PEARC 2018*, pp. 1-8, Pittsburgh, PA, USA, 2018.

[13] H. Liu, H. Jin, X. Liao, L. Hu, and C. Yu, “Live migration of virtual machine based on full system trace and replay,” in *Proc. ACM HPDC*, pp. 101-110, New York, NY, USA, 2009.

[14] C. Yu and F. Huan, “Live migration of Docker containers through logging and replay,” in *Proc. ICMII 2015*, pp. 623-626, Zhuhai, China, 2015.

[15] L. Ma, S. Yi, N. Carter, and Q. Li, “Efficient live migration of edge services leveraging container layered storage,” *IEEE Trans. Mob. Comput.*, vol. 18, no. 9, pp. 2020-2033, 2019.

[16] A. Machen, S. Wang, K. K. Leung, B. J. Ko, and T. Salonidis, “Live service migration in mobile edge clouds,” *IEEE Wireless Commun.*, vol. 25, no. 1, pp. 140-147, 2018.

[17] A. Mishra, M. Shin, and W. Arbaugh, “An empirical analysis of the IEEE 802.11 MAC layer handoff process,” *ACM SIGCOMM Comput. Commun. Rev.*, vol. 33, no. 2, pp. 93-102, 2003.

김 태 운 (Taewoon Kim)

2008년 : 부산대학교 정보컴퓨터공학과 학사

2010년 : 광주과학기술원 정보통신공학과 석사

2018년 : 아이오와주립대학 컴퓨터공학과 박사

2018년~현재 : 한림대학교 소프트웨어융합대학 조교수

<관심분야> 무선 통신 및 네트워크, 클라우드/엣지 컴퓨팅, 수치 최적화, 강화학습 등

[ORCID:0000-0002-7811-5022]