논문 21-46-02-12 The Journal of Korean Institute of Communications and Information Sciences '21-02 Vol.46 No.02
https://doi.org/10.7840/kics.2021.46.2.280

SDN 기반 분산 클라우드 오케스트레이션 시스템 구현

김 용 환◆, 김 동 균°

Implementation of an Orchestration System for Distributed Cloud Environments Based on Software Defined Networking

Yong-hwan Kim◆, Dongkyun Kim°

요 약

최근 클라우드 사용자 및 서비스의 확산과 다변화에 따라 기존의 클라우드 자원 운영 및 서비스 제공 방식에서 벗어나 보다 세분화된 형태의 가상화(Virtualization) 기술을 바탕으로 한 소프트웨어 기반의 자원 관리 및 통합 제어 시스템에 대한 수요가 증대되고 있다. 하지만 사용자가 요구하는 다양한 서비스를 제공하기 위하여 컴퓨팅, 네트워킹을 포함한 다양한 클라우드 자원의 관리 및 제어를 통합적으로 고려하기 어려우며, 더욱이 이를 기존의 시스템이 온디맨드로 신속하게 관리 및 제어하기에는 한계가 있다. 이에 따라, 본 논문에서는 분산 클라우드 환경에서 컨테이너 형태의 가상화된 서비스 자원과 소프트웨어 정의 네트워크 환경에서의 가상화된 네트워크 자원을 통합하여 효율적으로 제공할 수 있는 방안으로써 분산된 클라우드 자원들의 위치 및 부하 등을 고려한 통합 오케스트레이션 시스템을 제안한다. 또한 실제 구축되어 운용중인 SDN 광역망 환경에 제안하는 오케스트레이션 시스템을 구현하여 적용함으로써 제안 시스템을 실증하는 한편 사용자 요구에 따라 동적으로 생성된 컨테이너 간 데이터 전송 성능 측정하여 이의 결과를 보인다.

Key Words : SDN, Virtualization, Orchestration, Cloud, Resource Allocation

ABSTRACT

Recently, according to the proliferation and diversification of cloud users and services, the demand for a software-based resource management and integrated control system based on a more granular virtualization technology is increasing away from the existing cloud resource operation and service provision method. However, it is difficult to comprehensively consider the management and control of various cloud resources, including computing and networking, in order to provide various services required by users. Futhermore, existing systems are limited to quickly manage and control on demand. Accordingly, we propose an integrated orchestration system that considers the location and load of the distributed cloud resources. The proposed system can efficiently provide by integrating container-type virtualized service resources in a distributed cloud environment and virtualized network resources in a software-defined network environment. In addition, the proposed system is demonstrated by implementing and applying the proposed orchestration system to the SDN wide area network environment that is actually built and operated. We also show the results measuring the data transmission performance between containers that are dynamically created according to the user requirement.

※ 본 논문은 2020년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원을 받아 수행된 연구임 (2018-0-00749, 인공지능 기반 가상 네트워크 관리기술 개발).

◆ First Author : Korea Institute of Science and Technology Information, yh.kim086@kisti.re.kr, 정회원
° Corresponding Author : Korea Institute of Science and Technology Information, mirr@kisti.re.kr, 정회원

논문번호 : 202011-268-B-RU, Received November 4, 2020; Revised November 23, 2020; Accepted November 23, 2020

Ⅰ. 서 론

TCP/IP를 기반으로 한 인터넷이 등장한 이후 지난 30여 년간 크게 변하지 않던 네트워킹 분야에서 다양한 사용자 계층 및 기기들의 출현에 따라 네트워크의 복잡성이 증대되고 유연한 네트워크, 관리 편의성, 투자비 절감에 대한 요구가 증가하는 등 새로운 환경 및 요구사항에 적합한 네트워크 및 서비스의 대한 수요가 급증하고 있다[1-4]. 이러한 네트워크 패러다임의 변화에 대응하기 위하여 소프트웨어 정의 네트워킹(SDN, Software Defined Networking)기술은 데이터 영역과 제어 영역을 분리하고 중앙의 제어평면을 통해 네트워크를 중앙 집중 방식으로 제어함으로써 다양한 분야의 서비스 및 플랫폼을 지원할 수 있는 기술로 활용이 가속화 되고 있다[5,6]. 이의 대표 기술로써 네트워크 가상화 기술이 있으며 이는 기존에 하드웨어로 제공되던 네트워크 리소스를 소프트웨어로 추상화함으로써 기존 네트워크를 논리적으로 분할하여 운용할 수 있도록 지원한다.

한편, 최근 수많은 기업들이 자사의 시스템을 클라우드로 전환하는 등 본격적으로 클라우드 시장이 활성화됨에 따라 서비스형 소프트웨어(SaaS, Software as a Service)가 거의 모든 부문의 소프트웨어 성장을 주도하고 있다[7]. 이에 따라 클라우드 분야에서 다양한 사용자 및 서비스 요구사항의 수용, 물리적인 인프라 비용 및 서버 운용 관리 비용의 최소화, 방대한 데이터 처리, 유연한 자원 관리 등이 주요 이슈로 부각되고 있다. 이러한 이슈들을 해결하기 위하여 다양한 네트워크 오퍼레이터 및 서비스 사업자들은 컨테이너 기반의 서버 가상화 기술 및 컨테이너 통합 관리 기술에 주목하고 있다[8-12].

여기서 서버 가상화 기술은 물리 서버 자원의 낭비를 줄이기 위해 서버의 리소스들을 논리적으로 분할하여 다양한 서비스를 지원하는 기술을 말한다. 본 논문에서는 서버 가상화 기술로써 각광받고 있는 컨테이너 기반 서버 가상화 기술을 고려하고 있으며, 컨테이너 통합 관리 기술로써 오픈소스 기반의 소프트웨어 플랫폼인 쿠버네티스(Kubernetes)[13]를 대상으로 위의 이슈를 다루고자 한다. 쿠버네티스는 서버의 자원 조건과 제약사항 등을 고려한 컨테이너 자동 배치(Auto-Placement), 컨테이너 문제 발생 시 자동 재시작(Auto-Restart), 컨테이너 자원 요구에 따른 자원 확장 및 축소의 자동화(Auto-Scaling), 서비스 무중단 상태에서의 배포 및 복구(Replication, Rolling Update, Rollback), 다양한 클라우드 저장소 및 파일 시스템 연동 등의 많은 장점을 가지고 있다[14].

또한 클라우드 환경이 점차 분산화되면서 보다 작은 단위의 컨테이너 중심으로 재편됨에 따라 컴퓨팅 및 스토리지 자원과 이를 연결하는 네트워킹 자원의 통합적인 고려가 요구되고 있다. 하지만 사용자가 요구하는 서비스를 제공하기 위하여 네트워킹을 포함한 클라우드의 관리 및 제어를 통합적으로 고려하기 어려우며, 더욱이 이를 기존의 시스템이 온디맨드로 신속하게 관리 및 제어하기에 한계가 있다. 이에 따라, SDN과 컨테이너 기반의 분산 클라우드 환경을 통합적으로 고려한 새로운 시스템 및 방안에 대한 요구가 증대되는 실정이다. 이는 SDN과 컨테이너 기술 기반의 서버 가상화와 네트워크 가상화 기술을 활용하여 자원을 보다 효율적으로 사용할 수 있으며, 오케스트레이션 시스템을 통하여 사용자 혹은 관리자의 요청에 따라 동적으로 유연한 자원 관리 및 제어가 가능하기 때문이다.

본 논문에서는 새로운 네트워킹 서비스 요구사항들에 대응하기 위하여 SDN 기반의 네트워크 가상화 기술과 컨테이너 중심의 서버 가상화 기술을 통합 연계함으로써 다양한 분산 컴퓨팅 자원을 온디맨드로 신속하고 편리하게 제공할 수 있는 오케스트레이션 방안에 대하여 제안하고 이의 구현 결과를 보인다. 제안하는 오케스트레이션 방안에서 중점적으로 고려하는 사항은 다음과 같다.

- 지역적으로 분산된 다중 클라우드 환경에서 서비스 자원의 효율적 관리 및 활용을 위한 위치 및 부하 기반의 오케스트레이션 방법
- 사용자 요구에 따라 분산된 형태의 다양한 물리·가상자원(컴퓨팅, 스토리지, 호스트 등)을 네트워킹을 포함하여 온디맨드로 신속하고 편리하게 통합 제공

본 논문의 구성은 다음과 같다. 2장에서 SDN 및 컨테이너 관리 기술 기반 분산 서비스 자원 연계 구조를 제시하고, 3장에서는 위치 및 부하 기반의 컨테이너 자원 선정 및 네트워킹 연동 방안을 제안하는 한편 제안하는 SDN 기반 분산 클라우드 오케스트레이션 시스템의 구현 결과를 보이고, 4장에서 실제 구축되어 운용중인 SDN 환경에 제안방안을 적용하여 동적으로 생성된 컨테이너 간 데이터 전송 성능을 측정한 결과를 제시하고 5장에서 본 논문의 결론을 맺는다.

Ⅱ. 시스템 구조 및 기본 절차

제안하는 SDN 기반의 분산 서비스 자원 오케스트

레이션 기술은 네트워크 자원의 관리 및 제어를 위한 SDN 환경과 서비스 자원(컴퓨팅/스토리지 등)의 관리 및 제어를 위한 컨테이너 기반 분산 시스템 환경을 개방형 응용 프로그램 프로그래밍 인터페이스(API, Application Programming Interface)를 통하여 통합적으로 연계 관리하는 것을 목적으로 한다. 본 장에서는 이러한 제안 시스템의 구조 및 기본 절차에 대하여 설명하며, 3장에서 제시하는 제안 시스템의 구현 및 운용을 위하여 SDN 광역 통신망은 KREONET-S[15] 네트워크 인프라, 컨테이너 관리자는 쿠버네티스, SDN 컨트롤러는 ONOS(Open Network Operating System)[16], 가상 네트워크 관리자로는 한국과학기술정보연구원의 가상 네트워킹 시스템(VDN, Virtual Dedicated Networking)[17-18]을 활용하였다. 그리고 이때, 분산 클라우드 환경을 구성하기 위하여 다수의 쿠버네티스 환경을 지리적으로 분산된 위치에 독립적으로 배치하였다.

그림 1은 제안 시스템의 전체적인 구조 및 기본 절차를 보여주며, 각 구성요소의 역할과 기능은 다음과 같다.

- 오케스트레이터: 지능형 제어가 가능한 외부 서버 및 직접적인 사용자의 서비스 요청(컴퓨팅, 스토리지, 네트워크 등)에 따라 2-3절에서 제시되는 위치/부하 기반의 컨테이너 기반 클라우드 리소스 동적 생성 및 가상 네트워크 슬라이스 온디맨드 연동/활용을 총괄
- 컨테이너 관리자: 오케스트레이터로부터 선정된 서비스 클라우드 위치, 컴퓨팅, 스토리지 등의 서비스 자원의 요청을 받아 해당 위치의 클라우드에 컨테이너 기반의 서비스 자원을 할당하고 이의 수행 결과를 응답
- 가상 네트워크 관리자: 오케스트레이터로부터 사용자(종단 호스트 등), 할당된 컨테이너 기반 서비스 자원, 요구 대역폭 정보 등을 포함한 가상 네트워크 슬라이스 관리 요청을 받아 이를 처리하고 SDN 컨트롤러를 통하여 서비스 자원과 연계된 물리 네트워크 자원의 관리를 수행하고 이의 결과를 응답
- SDN 컨트롤러: 가상 네트워크 관리자로부터 요청받은 물리 네트워크 자원의 실질적 제어 역할을 수행(플로우 규칙, 미터 관리 등)
- SDN 광역 통신망(Networking)/분산 자원(Computing/Storage): 물리적인 서비스 및 네트워크 자원의 주체

사용자 혹은 다른 시스템에서 서비스 자원 및 이에 따른 네트워킹을 오케스트레이터에 요청하는 경우, 오케스트레이터는 먼저 지리적으로 분산 배치된 다수의 쿠버네티스 환경들 중에 적절한 서비스 환경 및 자원을 선택하여 컨테이너 형태로 사용자에게 생성 및 제공한다. 이 때, 쿠버네티스 환경 선택은 서비스 자원

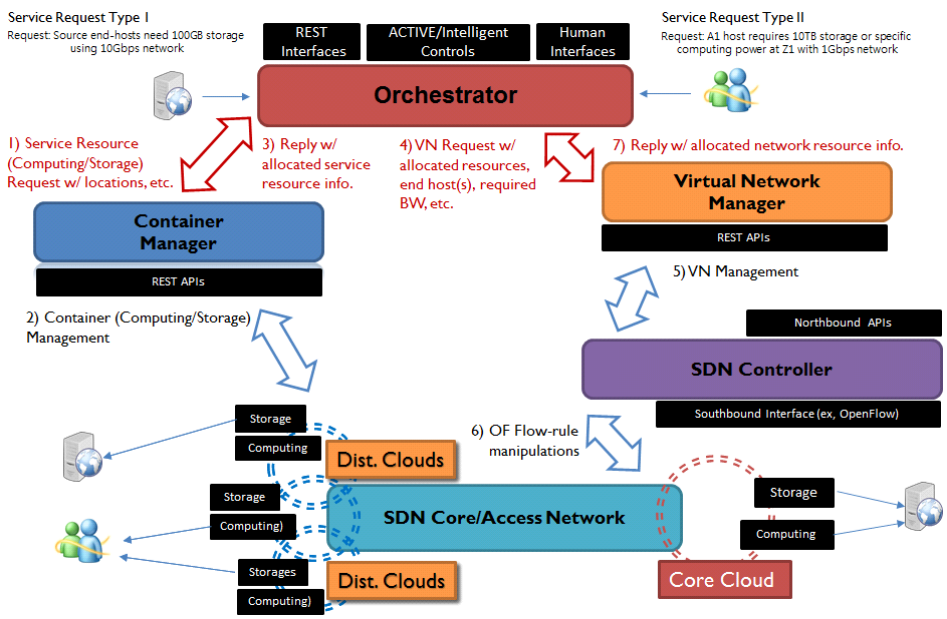


그림 1. SDN 기반 분산 자원 오케스트레이션 구조 및 절차
Fig. 1. SDN-based distributed resource orchestration structure and procedure

생성 시에 사용자 혹은 다른 시스템이 지정할 수도 있고, 사용자 호스트(들)의 위치 및 쿠버네티스 워커노드의 부하 등을 고려하여 시스템이 적절한 서비스 환경을 자동으로 선택할 수도 있다. 이러한 위치 및 부하 기반의 컨테이너 자원 선정 및 네트워킹 연동 방법은 3장에서 상세히 다룬다. 선택된 쿠버네티스로부터 요구한 서비스 자원이 컨테이너 형태로 생성되면 해당 컨테이너의 관리는 해당 쿠버네티스가 담당하며, 컨테이너와 해당 지역 SDN 스위치간의 연결 및 호스트 인식·설정 절차 등 네트워킹 및 관련 서비스에 대한 제어 및 관리는 SDN 컨트롤러가 담당한다.

Ⅲ. SDN 기반 분산 클라우드 오케스트레이션 시스템 개발

3.1 시스템 기능

제안하는 SDN 기반 분산 클라우드 오케스트레이션 시스템은 ONOS 컨트롤러 상에서 동작하는 애플리케이션 형태로 설계 및 개발되었으며, 해당 시스템이 동작하기 위하여 시스템 설정 및 연계 시스템 연동 기능, 서비스 자원 관리(생성/삭제/조회 등) 기능, 서비스 자원 간 네트워킹 연동 기능이 요구된다.

오케스트레이션 시스템 설정 및 연계 시스템 연동 기능은 API 레벨에서 쿠버네티스, ONOS, VDN 시스템과의 연계를 수행하기 위하여 요구되는 설정 및 랩퍼 클래스(Wrapper Class), 시스템 별 접근 권한허가, REST 등 API 호출을 수행하기 위한 인증, 통합 오케스트레이션 관련 설정 등이 포함된다. 이를 위하여 해당 관련 정보를 ONOS 환경 설정 디렉토리(ex, tools/package/config)에 별도의 JSON 파일을 두고 저장하여 관리한다. 또한, 이의 정보에는 다음절에서 소개하는 위치 및 부하 기반 컨테이너 자원 선정 및 네트워킹 연동 기능을 수행하기 위한 다양한 매개변수들이 포함된다. 표 1은 이러한 오케스트레이션 시스템 설정 및 연계 시스템 연동을 위하여 요구되는 주요 매개변수들을 제시한다. 여기서, “config”는 각각의 쿠버네티스와의 Open API 통신을 위한 인증 정보로 해당 쿠버네티스 서버의 /etc/kubernetes/admin.conf 파일을 복사하여 오케스트레이션 서버에 저장하고 이의 경로를 지정한다.

서비스 자원 생성 기능은 사용자가 입력한 설정 값을 바탕으로 컨테이너 오브젝트(Pod)를 생성하여 컨테이너 호스트 서비스를 제공한다. 이때, 사용자는 호스트 이미지(docker image), CPU, 메모리, 스토리지 등 자원에 대한 요구사항을 입력할 수 있으며, 이를 통해 위치 및 부하 기반으로 컨테이너가 생성될 서비스 위치가 자동으로 지정될 수 있다. 위치를 기반으로 컨테이너 서비스 리소스를 자동 생성할 경우에는 사용자가 선택한 호스트 목록이 참조된다. 또한 컨테이너 호스트의 IP 설정은 사용자가 입력한 동적 호스트 구성 프로토콜(DHCP, Dynamic Host Configuration Protocol)설정에 따라 할당하는데 이의 내용은 3.3절에서 다룬다.

서비스 자원 간 네트워킹 연동 기능은 위에서 생성된 컨테이너 오브젝트를 사용자가 입력한 DHCP 설정과 호스트 목록을 포함하여 가상전용망인 VDN으로 연계할 수 있다. 또한 서비스 컨테이너 오브젝트 삭제 시, 사용자가 오브젝트 별로 선택하여 삭제하도록 하며, 이 때 참조하는 모든 서비스 컨테이너 엔트리가 제거된 경우 이를 위하여 존재하는 VDN 또한 자동으로 삭제하도록 한다. 서비스 컨테이너 리스트를 통하여 사용자가 초기에 저장한 서비스 컨테이너 오브젝트 설정 값과 전체 또는 특정 서비스 컨테이너 오브젝트의 사용자 입력 값 및 현재 상태 등의 상세 정보를 볼 수 있다.

표 1. 오케스트레이션 시스템 설정 주요 매개 변수
Table 1. Main parameters for orchestration system configuration

Name (Type, Required/Optional)	Description
id (string, required):	unique region id
name (string, required):	region name
mode (string, required):	service mode
config (string, required):	kubeconfig file path
namespace (string, required):	namespace of kubernetes
macAddress (string, optional):	mac address of general mode service interface
city (string, optional):	city as location
weights (object, required):	weight for location select, specify city and value

3.2 위치 및 부하 기반 컨테이너 자원 선정

본 절에서는 오케스트레이션 시스템을 통하여 서비스 컨테이너 오브젝트 생성 시 다수의 쿠버네티스 환경 중 현재 상태에서 적절한 쿠버네티스를 선정하는 방안을 서술한다. SDN 광역망인 KREONET-S 인프라에 쿠버네티스 환경은 국내 3개 지역(대전, 서울, 부산), 해외 1개 지역(시카고)에 구축되어 있으며 이는 각 지역의 SDN 엑세스 스위치와 물리적인 회선을 통하여 연결·구성된다. 표 2는 쿠버네티스 환경 선정을 위한 변수 정보를 보인다.

KREONET-S의 VDN 시스템과 다수의 쿠버네티스 시스템 사이의 연동 및 쿠버네티스 선택은 모든 쿠버네티스 환경에 대하여 해당 서비스 요청에 대한 서비스 발생 위치, 요구 자원 등을 종합적으로 고려하여 서비스를 제공할 적절한 쿠버네티스 지점을 선택함으로써 이뤄진다. 사용자로부터 제안하는 오케스트레이션 시스템으로 서비스 요청이 발생하면 먼저 위치 및 자원 활용 속성을 기반으로 해당 서비스를 담당할 컨테이너 관리자를 적절하게 선택한다. 이 때, 사용자 서비스를 위해 지리적으로 사용자와 가까운 컨테이너 관리자를 선택하면 서비스 자원에 대한 사용자 트래픽의 전송 구간이 줄어든다. 그리고 이에 따라 전반적인 네트워크 인프라 사용을 줄이는 한편 네트워크 트래픽 혼잡을 감소시킬 수 있다. 이에 대한 제안하는 오케스트레이션 시스템에서의 위치 및 부하 기반 쿠버네티스 선정 함수는 다음과 같다.

$$C_{selected} = MAX_{C_i \in C}(w_{location} \cdot C_{location} + w_{resource} \cdot C_{resource})$$
$$where\ 1)\ w_{location} \geq 0, w_{resource} \geq 0,\ 2)\ w_{location} + w_{resource} = 1. \quad (1)$$

위치 기반 쿠버네티스 환경 가중치($C_{location}$)는 1)서비스 위치와 쿠버네티스 위치 사이의 링크 대역폭 활용률과 2)각 쿠버네티스 위치의 중심도를 기반으로 위치 관점에서의 쿠버네티스 가중치를 계산할 수 있다. 위의 두 가지 요소를 고려한 함수는 다음과 같이 정의된다.

$$C_{locataion} = \alpha \cdot l_{C_i, L_j} + \beta \cdot cv_{C_i} \quad (2)$$
$$where\ 1)\ l_{C_i, L_j} = \Pi_{l_k \in P_{(C_i, L_j)}},$$
$$2)\ cv_i = dv(C_i) / \sum_{C_j \in C} dv(C_j),\ 3)\ \alpha \geq 0, \beta \geq 0,\ 4)\ \alpha + \beta = 1.$$

정규화 된 네트워크 활용률($l_{i,j}$)은 C_i와 L_j 사이의 데이터 전송 경로에서 서로 다른 두 위치를 연결하는 외부 링크 활용률의 곱으로 계산할 수 있다. 광역 네트워크에서 서로 다른 두 위치 사이의 외부 링크 네트워크 트래픽은 백홀 혼잡의 주요 원인이기 때문에 균형 있는 링크 대역폭 활용이 중요하다. 또한, 외부 링크와 같은 핵심 네트워크 인프라의 사용을 줄이는 것이 전반적인 네트워크 활용도를 높이기위한 중요한 문제이기 때문에 C_i와 L_j간의 경로상의 외부 링크의 수가 증가함에 따라 $C_{location}$은 증가해야 한다.

한편, 노드 중심성(Node Centrality)은 전체 쿠버네티스 환경들 중에 각각의 사용자 서비스 제공 측면에서의 쿠버네티스 중요도를 식별하는 순위로써 정의된

표 2. 쿠버네티스 환경 선정을 위한 변수 정의
Table 2. Notations for Kubernetes environment selection

Notation	Description
L	Set of distributed service location
C	Set of locations deployed Kubernetes infrastructure
$l_{i,j}$	Normalized link utilization between location i and j, $0 \leq l_{i,j} \leq 1$
cv_i	Normalized centrality value of location i, $0 \leq cv_i \leq 1$
$dv(c_i)$	Data volume related with requested user service in C_i
$p_{(C_i, L_j)}$	Set of external links on the path between C_i and L_J
d(c)	Initial service CPU resources demand
d(m)	Initial service memory resources demand
d(s)	Initial service storage resources demand
r(c, C_i)	Availalble CPU resources in Kubernetes C_i
r(m, C_i)	Availalble memory resources in Kubernetes C_i
r(s, C_i)	Availalble storage resources in Kubernetes C_i

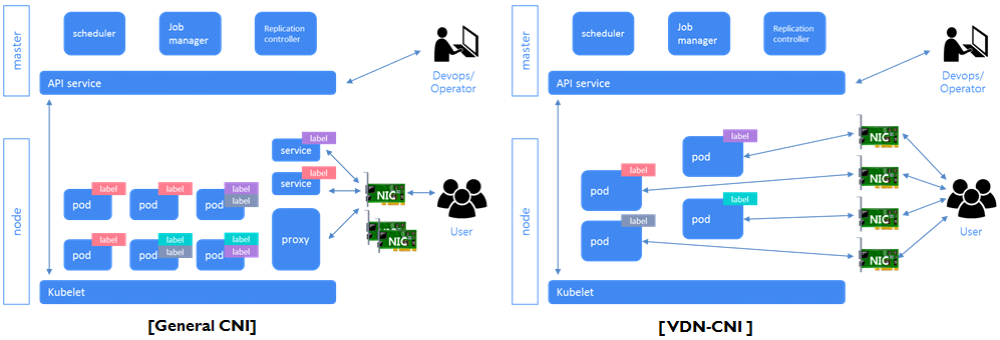


그림 2. 일반적인 CNI와 VDN-CNI 구조 비교
Fig. 2. Comparison of general CNI and VDN-CNI structures

다. 이는 중심성을 정의하는 방법에 따라 다양한 의미를 가질 수 있는데, 대표적인 중심도로는 정도 중심성(Degree Centrality), 근접 중심성(Closeness Centrality), 매개 중심성(Betweenness Centrality), 고유벡터 중심성(Eigenvector Centrality) 등이 있다[19]. 본 논문에서는 각각의 쿠버네티스 환경이 지닌 임의의 사용자 서비스와 관련된 데이터양의 비율을 기반으로 노드 중심성을 정의하였다. 여기서 중심성이 높다는 의미는 다른 쿠버네티스 환경과 데이터를 교환하지 않고도 서비스를 담당하는 쿠버네티스 환경에서 사용자가 필요한 데이터를 쉽게 가져올 수 있음을 의미한다.

부하 기반 쿠버네티스 환경 가중치($C_{resource}$)는 서비스 자원 요구를 만족하는 쿠버네티스 환경의 1) 가용 CPU, 2) 가용 메모리, 3) 가용 스토리지의 값을 전체 쿠버네티스 환경 중 최대 가용 자원을 값을 기반으로 정규화하여 다음과 같이 쿠버네티스 부하 기반 가중치를 계산한다. 이 때, 각 쿠버네티스의 현재 가용 자원 정보는 오케스트레이션 시스템에서 쿠버네티스 오픈 API 호출을 통하여 획득 가능하다.

$$C_{resource} = a_1 \cdot r_{nor}(c, C_i) + a_2 \cdot r_{nor}(m, C_i) + a_3 \cdot r_{nor}(s, C_i) \quad (3)$$
$$where\ 1)\ r(c, C_i) > d(c), r(m, C_i) > d(m), r(s, C_i) > d(s),$$
$$2)\ r_{nor}(c, C_i), r_{nor}(m, C_i), r_{nor}(s, C_i)$$
$$: normalized\ factor\ (0 \le r_{nor} \le 1),$$
$$3)\ a_n \ge 0, a_1 + a_2 + a_3 = 1.$$

3.3 SDN 네트워크 연계 컨테이너 네트워크 인터페이스

쿠버네티스 기반 컨테이너 관리자로부터 생성된 컨테이너 오브젝트의 네트워크 설정을 수행하고, 이를 SDN 광역망 인프라에서 해당하는 컨테이너 관리자 지역의 SDN 엑세스 스위치에 자동으로 연결하여 기본적인 네트워킹을 포함한 다양한 SDN 서비스를 제공하기 위해서는 별도의 컨테이너 네트워크 인터페이스(CNI, Container Network Interface)가 요구된다.

제안 방안에서는 이를 위하여, 1) 고성능 데이터 전송 서비스 용도의 CNI와 2) 시스템 운영 관리 용도의 CNI를 별도로 개발하여 목적에 부합하는 컨테이너에 할당할 수 있도록 다중 CNI를 통합된 형태로 제공하는 VDN-CNI를 설계 및 개발하였다. VDN-CNI의 기본 요구 사항은 생성된 컨테이너의 데이터 전송 성능을 보장하고 SDN 컨트롤러가 해당 컨테이너를 하나의 호스트로써 인식하도록 개발하는 것이다.

고성능 데이터 전송 서비스 제공을 위하여 연구 개발된 VDN-CNI의 기본 구조는 그림 2에서 볼 수 있다. 크게 두 가지의 CNI가 개발되었고, 이를 VDN-CNI:Host-device와 VDN-CNI:Macvlan라고 명명하였다. 전자의 경우 사전에 SDN 스위치에 연결된 물리 서버의 개별 네트워크 인터페이스 카드(NIC, Network Interface Card)를 서비스를 위하여 생성되는 컨테이너에 전용으로 할당하고 이를 SDN 스위치와 연결함으로써 SDN 기반의 고성능 네트워킹을 제공하는 CNI이다. 한편, 쿠버네티스 시스템의 운영 관리 및 모니터링 용도의 컨테이너 생성 시, 각 컨테이너별로 네트워크 인터페이스 카드를 할당하는 것은 물리적인 자원 낭비이기 때문에 지정한 하나의 네트워크 인터페이스 카드를 공유하여 논리적인 형태로 네트워킹을 제공하는 컨테이너 네트워크 인터페이스인 VDN-CNI:Macvlan를 보조 형태로 추가 개발하였다.

그림 3은 Host-device 모드의 VDN-CNI로 서비스 컨테이너 오브젝트 생성 시, 실제 물리적인 서버의 NIC을 직접 할당함을 보여주며, SDN 광역망의 가상망 관리 시스템 상에서 호스트 형태로 인지되는 모습

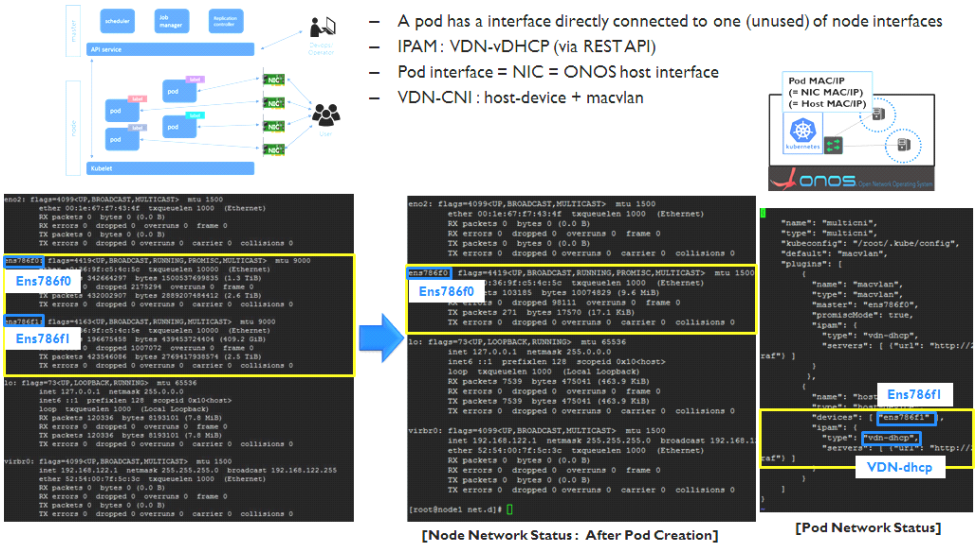


그림 3. VDN-CNI:Host-device 기반 컨테이너(Pod) 생성 및 시스템 인지 사례
Fig. 3. Example of VDN-CNI:Host-device based container (Pod) creation and system recognition

을 나타낸다. 이때, 서로 다른 IP 대역을 활용하는 다양한 사용자 그룹에게 서비스를 제공할 수 있어야 한다. 즉, 컨테이너의 IP 주소, 서브넷 마스크, 게이트웨이 등의 네트워크 설정 및 관리를 일반적인 DHCP를 이용하여 수행하기 어려운데, 이는 한정적인 IP 대역만으로 사용자가 요구하는 네트워크를 설정할 수 없기 때문이다.

따라서, 제안 시스템에서는 KREONET-S가 제공하는 VDN 별로 서로 다른 IP 대역을 활용하여 호스트의 네트워크 설정을 수행할 수 있는 별도의 DHCP 애플리케이션인 vDHCP를 개발하여 활용하였다. vDHCP는 사용자가 요구하는 IP 대역을 지정하면 해당 IP 대역으로 컨테이너의 네트워크 설정을 수행한다. 한편, KREONET-S와 독립적으로 운영되는 쿠버네티스에서 vDHCP를 활용하기 위하여 vDHCP 관리 및 제공 기능을 지원하는 표준화/정형화된 형태의 외부 인터페이스가 개발되었다. 이를 통하여 쿠버네티스에서 컨테이너 네트워크 설정을 위하여 KREONET-S의 vDHCP를 활용할 수 있다. 표 3은 이러한 컨테이너 생성 설정 파일의 형식을 보여주며, 해당 설정 파일을 쿠버네티스의 /etc/cni/net.d 디렉토리에 생성함으로써 쿠버네티스 환경에 적용할 수 있다.

3.4 제안 시스템 구현

본 절에서는 SDN 기반의 분산 서비스 자원 오케스트레이션 시스템의 구현 결과에 대하여 보인다. 해당 시스템은 ONOS 상에 "VDNO" 라는 이름의 애플리케이션으로 개발되었으며, 애플리케이션을 활성화하면 항목 선택 메뉴에서 "VDN Orchestration" 항목을 확인 할 수 있다. 해당 항목으로 들어가면 VDNO 리스트를 볼 수 있고, 리스트의 개별 기능을 확인하여 사용자가 요구하는 서비스 자원을 컨테이너 형태로 생성 가능하다. 그리고 사용자 및 호스트를 지정하여 이들 사이의 요구 대역폭을 보장하는 가상전용망인 VDN을 통합하여 생성 및 제공할 수 있다.

그림 4는 그래픽 사용자 인터페이스(GUI, Graphical User Interface)를 통한 VDNO 설정 입력 및 컨테이너 오브젝트 생성 예를 보여준다. 서비스 컨테이너 생성은 VDNO 리스트 상단의 생성 버튼을 클릭하고 사용자 호스트를 지정한 뒤 CPU, 메모리, 스토리지, 위치 등의 설정 값을 입력 후 컨테이너 추가 버튼으로 요구하는 서비스 자원을 생성할 수 있다. 이

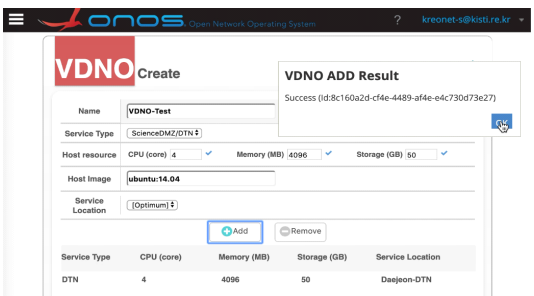


그림 4. VDNO 설정 화면에서의 컨테이너 생성 부분
Fig. 4. GUI for VDNO configuration including container creation

표 3. 쿠버네티스에서의 VDN-CNI 환경 설정
Table 3. VDN-CNI configuration in Kubernetes environment

Kubernetes CNI Configuration

```
{
    "name": "multicni",
    "type": "multicni",
    "kubeconfig": "/home/kube/.kube/config",
    "default": "macvlan",
    "plugins": [
        {
          "name": "macvlan",
          "type": "macvlan",
          "master": "NIC",
          "promiscMode": true,
          "ipam": {
            "type": "vdhcp",
            "servers": [ {"url": "http://IP Address:Port/dhcp API path/", "user": "user_name", "password":
"password_text"} ]
          }
        },
        {
          "name": "host-device",
          "type": "host-device",
          "devices": [ "NIC#1", "NIC#2",  "NIC#3", ...],
          "ipam": {
            "type": "vdhcp",
            "servers": [ {"url": "http://IP Address:Port/dhcp API path/", "user": "user_name", "password":
"password_text"} ]
          }
        }
    ]
}
```

때, 생성되는 서비스 자원을 담당하는 쿠버네티스 환경은 사용자가 지정하지 않은 경우 3.2절에서 제안한 방법을 통하여 시스템이 자동으로 선정한다. 또한 생성된 컨테이너 오브젝트와 KREONET-S 인프라 사이의 네트워킹 연결은 3.3절의 CNI를 통하여 자동 인지 및 설정된다.

한편 사용자 호스트 지정 및 가상망 대역폭, vDHCP 등 VDN 설정은 그림 5와 같이 할 수 있다. 이를 통하여 생성된 컨테이너 호스트와 사용자 호스트 사이의 VDN을 생성할 수 있고, 컨테이너 생성 시 해당 vDHCP 설정에 따라 컨테이너의 IP 설정을 수행한다. 그림 6과 그림 7은 각각 ONOS 상에서 확인한 생성된 컨테이너 호스트의 정보와 컨테이너 호스트와 사용자 호스트를 포함하여 생성된 VDN 토폴로지를 보인다.

생성되어 관리중인 컨테이너 호스트 등은 VDNO 리스트 GUI에서 확인 할 수 있다. 또한 해당 컨테이

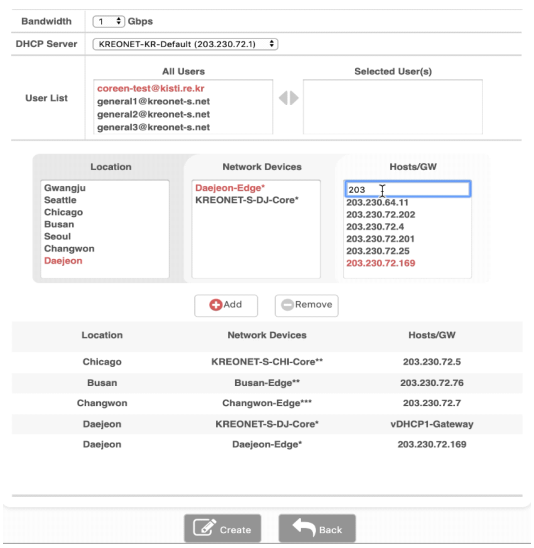


그림 5. VDNO 설정 화면에서의 VDN 생성 부분
Fig. 5. GUI for VDNO configuration including VDN creation

```
onos> vdno-list
ID { 2b60bfe5-7960-4039-b36c-2fd56d0e336b }:
   name [VDNO-Test], mode [DTN], region [Seoul-DTN]
   dhcpId[0], state [COMPLETE]
   image [ubuntu:14.04]
   cpu [5]core, memory [4096]MB, storage [60]GB
   ipAddress [203.230.72.3], hostId [A0:36:9F:7F:89:93/None], vdnId [9a93c0
b3269]
ID { 8c160a2d-cf4e-4489-af4e-e4c730d73e27 }:
   name [VDNO-Test], mode [DTN], region [Daejeon-DTN]
   dhcpId[0], state [COMPLETE]
   image [ubuntu:14.04]
   cpu [4]core, memory [4096]MB, storage [50]GB
   ipAddress [203.230.72.2], hostId [A0:36:9F:C5:4C:5E/None], vdnId [9a93c0
b3269]
```

그림 6. ONOS CLI를 통한 생성 컨테이너 호스트 정보 확인
Fig. 6. Information of created container host list through ONOS CLI

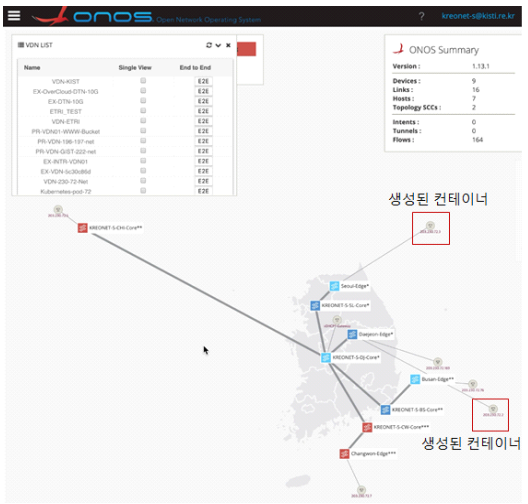


그림 7. VDNO를 통하여 생성된 VDN 토폴로지
Fig. 7. VDN topology created through VDNO

너 삭제는 VDNO 리스트를 보여주는 GUI에서 대상 항목을 선택한 후 상단의 삭제 버튼을 클릭하여 삭제를 수행할 수 있다. 삭제를 시도하면 진행 여부를 확인한 후 최종적으로 해당 오퍼레이션이 수행되며, 참조하는 모든 컨테이너 호스트가 제거된 경우 관련된 VDN 또한 자동으로 삭제되도록 구현하였다. 그림 8은 VDNO 리스트 GUI 및 선택된 컨테이너 제거 예를 보여준다.

Ⅳ. 성능 분석

본 장에서는 실제 구축되어 운용중인 SDN 광역망 환경인 KREONET-S에 제안하는 오케스트레이션 시스템을 적용하고 사용자 요구에 따라 동적으로 생성된 컨테이너 오브젝트 간 데이터 전송 성능 측정 결과를 제시한다. SDN 광역망인 KREONET-S는 7개 핵심 지역망 센터(서울, 대전, 부산, 광주, 창원, 시카고, 시애틀)를 중심으로 각각 코어 및 에지/액세스 네트워크 디바이스가 구축되어 있으며, ONOS 분산제어플랫폼을 통해 캐리어급 SDN 광역망의 운영 및 토폴로지 정보 관리를 수행 중에 있다.

본 장의 성능 분석 과정에서 컨테이너 생성 및 관련 설정을 수행을 위하여 3.4절에서 구현한 VDNO 시스템을 활용하였으며 SDN 네트워크 연계 컨테이너 네트워크 인터페이스는 3.3절에서 제안한 VDN-CNI를 채택하였다. 한편, 네트워크 성능 측정은 Iperf3을 활용하였다.

그림 9는 KREONET-S 상에 제안 시스템을 통하여 생성된 컨테이너에 접속하여 최대 전송 단위(MTU, Maximum Transmission Unit), CPU 속도, 송신 큐 사이즈 조정 등의 네트워크 성능 튜닝 작업을 수행한 이후의 성능 결과를 보인다. 1G VDN와 10G VDN으로 연결된 컨테이너 간의 성능이 각각 1Gbits/sec, 9.8Gbits/sec로 물리적인 NIC을 성능을 온전히 보임을 확인할 수 있다. 이는 VDN-CNI:Host-device가 서버의 NIC을 컨테이너에 직접적으로 할당하는 패스스루(Pass-through) 방식으

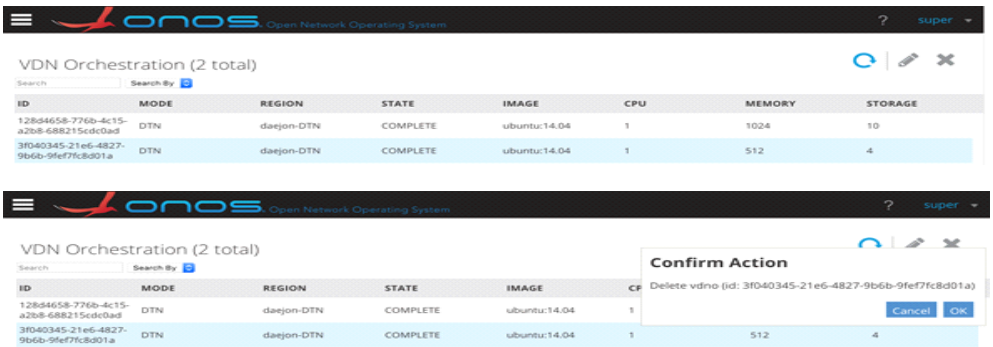
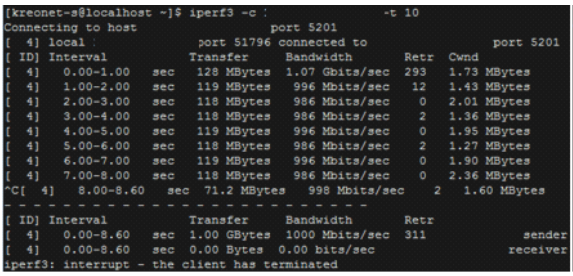


그림 8. VDNO 리스트 GUI 및 선택된 컨테이너 삭제 예
Fig. 8. Example of GUI for VDNO list and deletion of selected container



(a) 1G VDN 네트워크 성능 테스트 (a) 10G VDN 네트워크 성능 테스트

그림 9. 네트워크 성능 튜닝 이후 VDNO 컨테이너 간 네트워크 성능 확인
Fig. 9. Network performance between VDNO containers after network performance tuning

로 구현되어 있기에 가능하다.

그림 10는 KREONET-S 상에 적용된 제안 시스템을 통하여 생성된 컨테이너 간의 네트워크 전송 성능 결과를 보인다. 데이터 전송 실험은 10G VDN 환경하에 이뤄졌으며 컨테이너의 네트워크 성능이 튜닝 상태에서 성능을 측정하였다. VDN-CNI:Host-device를 통하여 네트워킹 설정을 수행한 컨테이너 사이의 데이터 전송은 평균 9.88Gbits/sec의 성능을 보였으며, VDN-CNI:Macvlan의 경우 평균 9.7Gbits/sec의 성능을 보였다. 이를 통하여 VDN-CNI로 생성된 컨테이너 간 네트워크 통신은 NIC 성능을 온전히 보임을 확인 할 수 있다. 한편, 시스템 운영 관리 용도의 Macvlan CNI는 고성능 데이터 전송 서비스 용도의 Host-device CNI에 비하여 불안정한 네트워크 전송 성능을 보임을 볼 수 있었다. 이러한 차이는 하나의 NIC을 소프트웨어적으로 공유하여 활용하는 Macvlan CNI와는 달리 Host-device CNI는 컨테이너에 하나의 물리적인 NIC을 온전히 할당받아 사용하는데 기인한다.

그림 11은 제안 시스템을 통하여 생성된 두 종류(VDN-CNI:Host-device, VDN-CNI:Macvlan)의 컨테이너와 일반적으로 잘 알려진 CNI인 Calico, Flannel, Weave Net CNI로 생성된 컨테이너, 물리적인 서버의 네트워크 전송 성능 비교 결과를 보인다. 이 때, 네트워크 성능 튜닝에 대한 영향을 제거하기 위하여 성능 튜닝은 수행하지 않았다. 이를 통하여 제안 시스템을 통하여 생성된 서비스 용도의 VDN-CNI:Host-device 컨테이너는 물리적인 서버 NIC과 동일한 결과를 보이며, 운영/관리 용도의 VDN-CNI:Macvlan의 경우에도 기존의 CNI에 비하여 높은 성능을 보임을 확인 할 수 있었다.

앞선 성능 분석 결과를 통하여 확인한 바와 같이 VDNO 시스템을 통하여 생성된 컨테이너들의 종단간 네트워크 성능은 물리적인 NIC의 성능에 거의 근접하게 달성됨을 확인할 수 있었다. 하지만 종단간의 네트워크 성능이 실질적인 파일 전송이 이뤄지는 디스크간(D2D, Disk to Disk) 데이터 전송 성능을 보장하지는 못한다. 이에 따라, 본 절에서는 D2D 전송 성능을 측정하기 위하여 Globus Online 시스템[20]의 대용량 파일 전송 기능을 활용하여 제안 시스템에서 생성된 컨테이너간 데이터 전송 성능 시험을 수행하였다. 그림 10의 상단은 Globus Online에서 컨테이너 형태로 생성되어 등록된 서울 지역의 쿠버네티스 컨테이너 호스트와 (k8s-sl-dtn.kreonet-s.net)이 부산 지역의 쿠

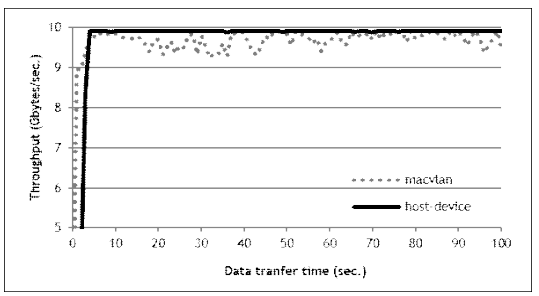


그림 10. 제안 CNI를 사용하는 VDNO 컨테이너 간 네트워크 성능 비교
Fig. 10. Comparison of network performance between VDNO containers using proposed CNI

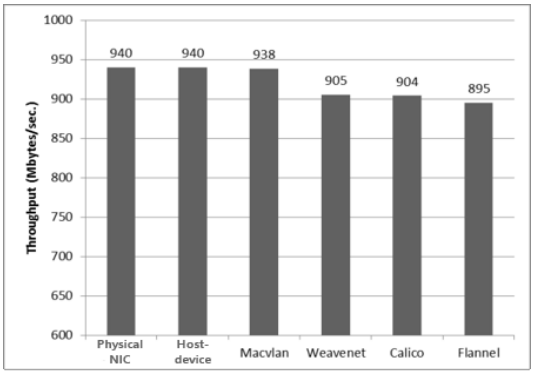


그림 11. CNI 네트워크 성능 비교
Fig. 11. Comparison of CNI network performance

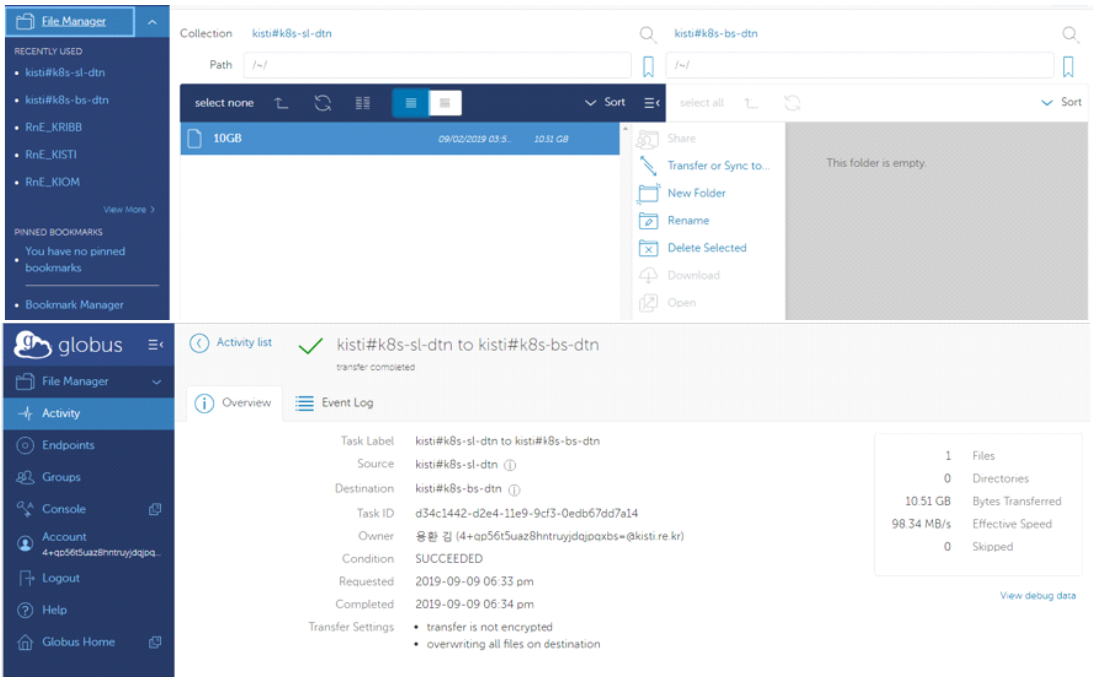


그림 12. VDNO 컨테이너 간 디스크간 데이터 전송 성능 확인
Fig. 12. disk to disk performance between VDNO containers

버네티스 컨테이너 호스트(k8s-bs-dtn.kreonet-s.net)에게 10GB 파일을 전송하는 화면이며, 하단은 해당 파일 전송이 완료된 이후 Globus Online 시스템에서 측정한 컨테이너간의 D2D 전송 성능을 보인다. 해당 성능은 98.34Mb/s(약 790 Mbps)로 물리적인 서버에 튜닝이 완료되고 별도의 독립적인 물리 회선으로 연결된 호스트간의 성능과 거의 유사한 성능을 보임을 확인할 수 있다.

V. 결 론

본 논문에서는 변화하는 클라우드 환경 및 서비스 요구사항들에 대응하기 위하여 SDN 기반의 네트워크 가상화 기술과 컨테이너 기반의 서버 가상화 기술을 통합 연계함으로써 다양한 분산 물리 및 가상자원을 온디맨드로 신속하고 편리하게 제공할 수 있는 오케스트레이션 기술의 연구결과 및 개발현황에 대하여 소개하였다. 그리고 이를 실제 SDN 광역망 인프라에 구현 및 적용하고 서비스 가능 수준의 데이터 전송 성능을 보임으로써 제안 시스템을 검증하고 유용성을 입증하였다. 이는 자체 네트워크와 분산 클라우드 환경을 구축하고 있는 5G 사업자 등의 네트워크 오퍼레이터 및 서비스 프로바이더 측면에서 다양한 자원들을 통합하여 효율적으로 운영 관리하며, 사용자에게 다양한 클라우드 기반의 서비스를 제공하기 위한 하나의 프레임워크로써 의미를 지닌다. 또한 향후 오케스트레이터를 기반으로 다양한 SDN 및 클라우드 기술 등을 비롯한 4차 산업 혁명의 핵심 분야인 IoT, 빅데이터, AI 등의 연계 분야의 응용 기술 개발 확대에 기여하리라 기대한다.

한편, 가상화 기술 기반의 클라우드 환경 구축 과정에서의 운영·관리 및 서비스 제공 측면에서 자동화와 지능화에 대한 필요성이 점차 커지는 추세이기 때문에 향후 인공지능 및 기계학습 기반의 분산 클라우드 오케스트레이션 시스템에 대한 연구개발을 지속적으로 수행할 예정이다. 특히, 향후에 본 논문에서 제안하는 위치 및 부하 기반 컨테이너 자원 선정을 위한 파라미터 값 조정을 통한 전체 네트워크 사용률 및 쿠버네티스 환경의 물리 자원 최적화 문제를 인공지능 및 기계학습 기반으로 해결해보고 한다.

References

[1] Cisco Visual Networking Index, "*Forecast and Methodology*," Cisco White Paper 2014-2019, 2015.

[2] M. K. Weldon, "*The future X network: A bell labs perspective*," Taylor & Francis Group, LLC, 2016.

[3] A. Manzalini, et al., "Software-defined networks for future networks and services," *White Paper based on the IEEE Workshop SDN4FNS*, 2014.

[4] P. Porambage, et al., "Survey on multi-access edge computing for internet of things realization," *IEEE Commun. Surv. & Tuts.*, vol. 20, no. 4, pp. 2961-2991, 2018.

[5] N. McKeown, et al., "OpenFlow: Enabling innovation in campus networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69-74, Apr. 2008.

[6] T. D. Nadeau and K. Gray, "*SDN: Software Defined Networks: An authoritative review of network programmability technologies*," 1st Ed., O'Reilly Media, Inc., 2013.

[7] STAMFORD, Conn., "*Gartner Forecasts Worldwide Public Cloud Revenue to Grow 17.3 Percent in 2019*," Apr. 2019.

[8] T. Taleb, et al., "On multi-access edge computing: A survey of the emerging 5G network edge cloud architecture and orchestration," *IEEE Commun. Surv. & Tuts.*, vol. 19, no. 3, pp. 1657-1681, 2017.

[9] X. Ma, et al., "Cost-efficient resource provisioning for dynamic requests in cloud assisted mobile edge computing," *IEEE Trans. Cloud Comput.*, 2019.

[10] Y. Kim, J.-M. Gil, and D. Kim, "A location aware network virtualization and reconfiguration for 5G core network based on SDN and NFV," *Int. J. Commun. Syst.*, 2020. e4160.

[11] D. Kim and Y.-H. Kim, "Dynamic virtual network slicing and orchestration for selective MEC services over Wide-Area SDN," *Algorithms*, vol. 13, no. 10, p. 245, 2020.

[12] J. Yi, "Zero touch service & network automation and network function virtualization standard technology," *KICS Inf. and Commun. Mag.*, vol. 36, no. 9, pp. 49-54, 2019.

[13] *Kubernetes web site*, Retrieved Nov. 3, 2020, from https://kubernetes.io/

[14] D. Bernstein, "Containers and cloud: From lxc to docker to kubernetes," *IEEE Cloud Computing*, vol. 1, no. 3, pp. 81-84, 2014.

[15] *KREONET-S web site*, Retrieved Nov. 3, 2020, from http://www.kreonet-s.net/

[16] *Open Network Operating System (ONOS) web site*, Retrieved Nov. 3, 2020, from https://opennetworking.org/onos/

[17] Y. H. Kim, K.-H. Kim, and D.-K. Kim, "Design and implementation of virtually dedicated network service in sd-wan based advanced research & educational (R&E) network," *J. KICS*, vol. 42, no. 10, pp. 2050-2064, 2017.

[18] D. Kim, et al., "Cloud-centric and logically isolated virtual network environment based on software-defined wide area network," *Sustainability*, vol. 9, no. 12, p. 2382, 2017.

[19] L. C. Freeman, "Centrality in social networks conceptual clarification," *Social Networks*, vol. 1, no. 3, pp. 215-239, 1978-1979.

[20] *Globus Online web site*, Retrieved Nov. 3, 2020, from https://www.globus.org/tags/globus - online

김 용 환 (Yong-hwan Kim)

2010년 8월 : 한국기술교육대학교 정보미디어공학과 석사

2015년 8월 : 한국기술교육대학교 컴퓨터공학과 박사

2016년 2월~현재 : 한국과학기술정보연구원(KISTI) 연구원

<관심분야> SDN/NFV, Network Virtualization, Mobility Management, Social Networks

[ORCID:0000-0003-3323-0323]

김 동 균 (Dongkyun Kim)

1999년 2월 : 충남대학교 컴퓨터과학과 석사

2005년 2월 : 충남대학교 컴퓨터과학과 박사

2006년 4월~2007년 3월 : 미국 테네시대학(UT)/오크리지국립연구소(ORNL) Research Associate III/방문연구원

2000년 6월~현재 : 한국과학기술정보연구원(KISTI) 책임연구원

<관심분야> SDN/NFV, SD-WAN, Research Networking, Network Virtualization

[ORCID:0000-0001-9484-3399]