Journal of the Korea Industrial Information Systems Research Vol.26 No.1, Feb. 2021 :11-19
http://dx.doi.org/10.9723/jksiis.2021.26.1.011 ISSN:1229-3741

MEC를 위한 세션 테스트 도구 개발+.

(Implementation of Session Test Tool for MEC)

김 태 영[1], 김 태 현[2], 진 성 근[3]*
Tae-Young Kim, Tae-Hyun Kim, and Sunggeun Jin)

요 약 5G 네트워크의 등장으로 초저지연 서비스에 대한 요구가 제기되었다. 그러나, 사용자로부터 지리적으로 멀리 위치한 클라우드 센터의 컴퓨팅 서비스로는 이러한 요구를 만족할 수 없다. 이러한 요구에 따라 클라우드 컴퓨팅 서비스를 사용자 근처에 위치한 기지국 혹은 교환국에 전진 배치하여 저지연 서비스를 제공하는 Multi-access Edge Computing (MEC) 기술이 주목받고 있다. 우리는 구글의 Kubernetes를 기반으로 MEC를 위한 클라우드 컴퓨팅 환경을 구축하였다. 이때, 안정적인 동작 확인을 위해 많은 수의 컨테이너가 발생시키는 로드에 강건하게 견딜 수 있는지 실험적으로 확인할 필요가 있다. 이를 위하여 우리는 Kubernetes 환경에서 다양한 컨테이너를 생성하여 네트워크 자원과 컴퓨팅 자원의 안정도를 측정할 수 있는 도구를 개발하였다.

핵심주제어: MEC, 쿠버네티스, 클라우드 컴퓨팅, 컨테이너, 파드 배치, 모니터링

Abstract The emerging Fifth Generation (5G) network technology brings us a new demand for low latency services. However, it may not be possible for long-distanced cloud computing servers to support users with satisfactory low latency services. For this reason, Multi-access Edge Computing (MEC) technology are gaining attraction since it is designed to provide low latency services to users by placing cloud computing resources to base-stations or mobile switching centers nearby users. Accordingly, it is necessary to verify the deployed containers on the MECs are reliable enough to provide low latency services empirically. For the purpose, we develop a testing tool to verify the reliability as well as network resources status of running MECs by deploying containers on the MECs in a Kubernetes environment.

Keywords: MEC, Kubernetes, Cloud computing, Container, Pod placement, Monitoring

1. 서 론

클라우드 컴퓨팅 기술의 발전으로 컴퓨팅 자원을 직접 구매하지 않고 대여하여 활용할 수 있게 되었다. 그러므로 사용자가 강력한 컴퓨팅 자원을 위해 큰 비용을 투자하지 않고도 사용할 수 있게 됨으로써 우리의 일상에서 클라우드 컴퓨팅이 제공하는 다양한 서비스를 활용할 수 있게 되었다. 5G 네트워크 기술이 접목된 휴대용 단말기 보급이 이루어짐에 따라 대용량 네트워

* Corresponding Author: sgjin@daegu.ac.kr
+ 이 연구는 2017학년도 대구대학교 학술연구비 지원으로 수행되었음.
Manuscript received November 06, 2020 / revised December 11, 2020 / accepted December 21, 2020
1) 대구대학교 컴퓨터 정보공학과, 제1저자
2) 대구대학교 컴퓨터 정보공학과, 제2저자
3) 대구대학교 컴퓨터소프트웨어학과, 교신저자

크 대역과 저지연 서비스를 제공할 때 중앙집중식으로 처리하는 클라우드 컴퓨터팅 만으로는 한계가 생긴다. 대용량 데이터 처리의 부하분산과 저지연을 달성하기 위해, 마치 사용자 바로 옆에 존재하여 데이터를 처리하는 MEC(Mobile Edge Computing)기술이 등장하였다. 무선통신 서비스를 제공받는 사용자와 가까운 기지국이나 교환국에 클라우드 컴퓨팅 자원의 일부를 전진 배치함으로써 네트워크 지연시간을 줄이고 사용자 요청에 대한 요구에 빠르게 응답할 수 있도록 지원한다. 이를 통해, 초저지연, 대용량 대역폭 제공, 실시간 네트워크 정보 접근을 가능하게 해준다. MEC 기반의 데이터를 처리하기 위한 시스템으로 우리는 Kubernetes를 기반으로 구축하였다. 시스템 구축에 이어 실제 서비스를 하기 위해서는 서비스를 제공하기 전에 원하는 서비스 품질을 만족하는지 확인할 필요가 있다. MEC 환경에서 클라우드 환경을 구축하고 구축한 클라우드 환경에서 지연율을 측정하고, VM과의 상관관계를 증명하는 등 다양한 연구가 있으나, 네트워크에 따른 지연율을 측정하고 해당 지연율에 따른 안정적인 환경을 위해 최대 세션수를 파악하고 적용하는 방법에 대해선 연구가 많이 진행되지 않았다 (Yala, Louiza 2018). 본 연구에서는 서비스 품질 확인을 위해 다양한 컨테이너를 생성하고 서버 컴퓨터에 부하와 대량의 트래픽을 발생시켜 MEC 서버의 한계점을 파악한다.

우리는 MEC 환경에서 컨테이너를 생성하고 관리함으로써 MEC 서비스 품질이 어떻게 유지되는지 손쉽게 확인하기 위한 도구를 개발하였다. 우리가 개발한 도구를 통하여 시험 시나리오를 지정하고 이에 따라 변화하는 네트워크 자원량과 컴퓨팅과 관련된 자원량을 시각적으로 확인할 수 있다. 또한, Kubernetes 환경에서 필요한 복잡한 명령어를 사용하지 않고 GUI 기반의 사용자 인터페이스를 제공함으로써 쉽고 간단하게 테스트를 진행 할 수 있다.

본 고는 2장에서는 기술적 배경에 관해서 설명하고 3장에서 도구의 구현을 설명한다. 4장에서 실험의 환경과 결과를 설명하고 5장에서 결론을 맺는다.

2. 배 경

현재 5세대 이동통신망인 5G 기술의 발전에 따라 NFV/SDN과 함께 사용되는 MEC는 중앙집중형 클라우드 센터의 컴퓨팅기술만으로는

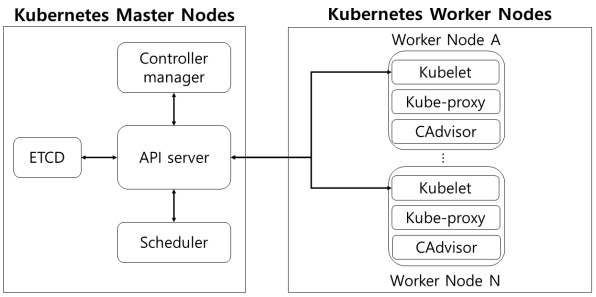

Fig. 1 Kubernetes Node Architecture

처리한계가 있어 이를 보완하기 위해 핵심기술로 사용되고 있다. 초저지연 서비스를 위해 데이터를 사용하는 사용자 근처에서 데이터 처리를 하고 통신 서비스를 제공하는 기지국에서 클라우드 분산 기술을 이용해 다양한 서비스를 단말기 사용자 가까이 배치하여 코어 네트워크가 최적화되어 1ms 이하의 초저지연 서비스를 제공한다 (Patel et al., 2014; Park et al., 2018).

초저지연 MEC 서비스 제공을 위한 클라우드 컴퓨터 관리 시스템으로써 무중단 서비스 제공을 위한 다양한 오케스트레이션 도구 중 성능이 뛰어나고 MEC 환경에서 서비스 배포 및 관리가 용이한 google의 Kubernetes를 사용한다 (Aderaldo et al., 2017).

Kubernetes는 일종의 MSA(micro service architecture)인 docker 컨테이너를 자동화 관리 및 배포를 도와주는 오케스트레이션 도구이다. Docker는 하나의 서비스만을 제공하는 가장 작은 프로세스인 컨테이너로 구성된다. Kubernetes는 여럿의 docker 컨테이너를 하나의 Pod라는, Kubernetes에서 가장 작은 개념으로 구성된다. Pod는 Kubernetes의 모든 시스템에 적용이 된다. Pod는 사용자에게 서비스를 제공하는 어플리케이션이 될 수 있으며, Kubernetes를 제어하는 프로세스가 될 수 있다.

Fig. 1은 Kubernetes의 구조도이다. Kubernetes는 마스터 노드와 워커 노드로 구성되어 있다. 마스터 노드는 워커 노드의 제어 및 관리, 워커 노드는 하나 이상의 컨테이너 그룹인 Pod를 배치하여 서비스 제공을 담당한다. 마스터 노드는 크게 ETCD, scheduler, API server, controller manager로 이루어져 있다. ETCD는 key, value 저장소이며, Kubernetes cluster의 각종 메타 데이터를 저장한다. Kubernetes의 s-cheduler는 각종 서비스, 즉 Pod를 배포하기 위해 Kubernetes cluster에 존재하는 최적의 워커 노드를 계산하여 Pod를 배치한다. API server는 일종의 HTTP 서버로 Kubernetes의 내부 또는 외부의 각종 API를 제공한다. Controller manager는 Kubernetes cluster의 상태를 주기적으로 측정하여 사용자가 원하는 상태를 유지한다.

특정 노드에 부하를 받아 서비스 제공이 어려워지면 Fig. 1의 Kubelet을 통해 노드의 상태를 마스터 노드의 API server에 전달한다. 상태를 전달받은 API server는 scheduler를 통해 Pod를 최적의 노드에 배치하여 업스케일링을 시도한다. Kubernetes는 스케줄링의 요소로 CPU, memory, I/O 사용만을 고려한다. 현재 Kubernetes 스케줄링의 한계로 네트워크 환경이 중요한 서비스, 즉 저지연을 만족하는 서비스나 대량의 트래픽을 감당해야 하는 경우일때 scheduler는 CPU, memory, I/O 자원만을 이용하기 때문에 현재 네트워크 상태를 전혀 고려하지 않고 적절하지 않은 노드에 Pod를 배치한다. 적절하지 않은 노드에 Pod가 배치되기 때문에 초저지연 서비스 제공이 불가능하다. 따라서 제한적인 자원만을 이용하는 Kubernetes scheduler의 네트워크 고려와 MEC 서비스를 제공하기 전 최적의 서비스 제공 환경을 확인하기 위해 세션 테스트 도구를 제시한다.

3. 세션 테스트 도구 구현

3.1 스케줄러 확장

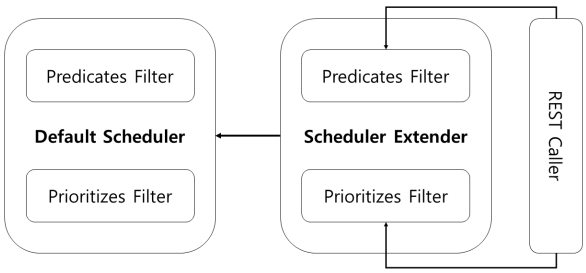

Fig. 2 Architecture of Scheduler Extender

Kubernetes는 기본적으로 CPU, memory, I/O의 자원 사용량에 근거하여 로드밸런싱을 한다. 그러나 이러한 한정된 자원으로 선별하는 경우 다양한 환경에서의, 즉 단순히 Kubernetes에서 기본적으로 사용하는 자원이 아닌 네트워크, GPU 등을 고려하는 서비스는 최적의 로드밸런싱이 불가능하다. 네트워크 자원을 스케줄링에 고려하는 테스트 도구를 위해서 사용한 확장 스케줄러를 소개한다.

Fig. 2는 우리가 세션 테스트 구현을 위해 활용한 확장 스케줄러 구조도이다 (Kubernetes, 2015). 확장 스케줄러는 기본 스케줄러와 같이 하나의 Pod로 제공되며, 기본 스케줄러와 동등하게 API server로부터 배치 명령이 떨어지면 스케줄러에서 각 노드의 상황을 판단하고 kubelet을 통해 배치한다. 스케줄러는 Predicates 필터기, priorities 필터기로 나누어진다. Predicate 필터기는 predicate 알고리즘으로 구성되어 있고, 알고리즘은 Kubernetes에서 label, policy 등에 의해 스케줄링에 제외되는 경우를 계산한다. Priorities 필터기는 priorities 알고리즘으로 구성되 있으며, 알고리즘은 Kubernetes cluster에 소속된 노드의 가용가능한 자원량을 점수로 환산하여 계산 후 노드 선택 알고리즘이다.

확장 스케줄러는 기본 스케줄러와 마찬가지로 Predicate, priorities 필터기로 구성되며, 기본 스케줄러의 Predicate 알고리즘에 의해 노드 필터링 이후 확장 스케줄러의 predicate 알고리즘을 모두 수행하여 노드들을 필터링한다. 수행이 완료되면 기본 스케줄러와 확장 스케줄러의 Priorities 알고리즘을 모두 수행하여 노드를 선정한다(T.Y. Kim et al.(2020). Priorities 알고리즘을 수행할 때는 Kubernetes의 모니터링 도구

인 Prometheus와 Kubernetes의 API server의 정보를 호출하여 사용하고 이때 restcaller 모듈을 이용해 해당 정보를 각 필터기에 반환한다.

3.2 모니터링 시스템

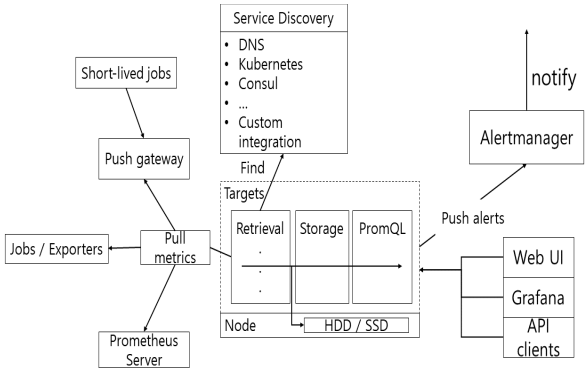

Fig. 3 Architecture of Prometheus

Fig. 3은 우리가 모니터링 시각화를 위해 사용한 도구인 Prometheus의 구조도이다 (Prometheus, 2014). Prometheus는 다른 모니터링 도구의 경우 각 서버에서 클라이언트를 설치하고 해당 클라이언트가 metric 데이터를 수집하여 서버로 전송하여 모니터링하는 push 방식인데 비해 Promtheus는 반대로 각 노드에서 노드 익스포터를 설치하여 클라이언트들이 모니터링 데이터를 수집하고 서버가 필요에 의해 사용하는 pull 방식을 취하고 있다. 이러한 방식으로 수집된 모니터링 데이터들은 REST API를 통해 다른 모듈에 인터페이스를 제공하며 이를 확장 스케줄러와 Prometheus adapter에서 활용한다. 앞서 설명한 확장 스케줄러와 Prometheus를 이용하여 MEC 서버 동시 세션 테스트 도구를 개발한 예시를 소개한다. 동시 세션 테스트 도구는 사용자 편의성을 위해 CLI(Command Line Interface)가 아닌 GUI(Graphical User Interface)를 활용한다. 모니터링 도구는 Prometheus를 이용하여 metric 데이터를 수집하였고 Node..js를 이용하여 웹 어플리케이션을 제작하였다.

3.3 제안된 스케줄러 블록

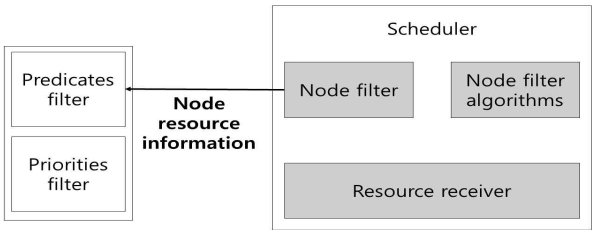

Fig 4. Scheduler Block Architecture

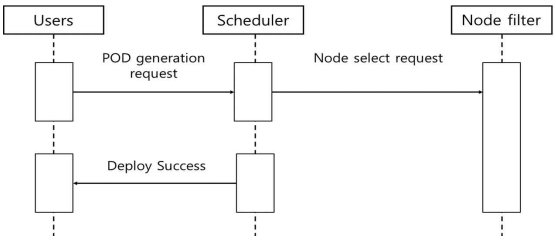

Fig. 5 Node Filter Unit

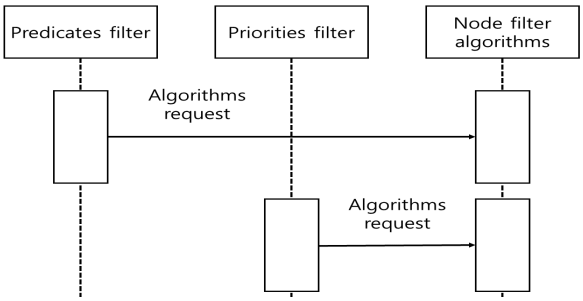

Fig. 6 Node Filter Algorithms

우리가 세션 테스트를 위해 만든 스케줄러 블록은 Fig. 4와 같다. 테스트 도구는 2개의 블록으로 이루어져 있으며, 각각 스케줄러, 모니터링으로 나누어진다. 스케줄러 블록은 노드 필터기, 노드 필터 알고리즘, 자원 리시버, Predicates 필터기, priorities 필터기로 이루어져 있다.

스케줄러는 자원 리시버를 이용하여 연결된 모든 노드로부터 자원 현황 정보를 받는다. 정보를 기준으로 노드 필터는 Predicates 필터와 priorities 필터의 알고리즘 계산을 거쳐 Pod를 할당받을 노드를 선별한다. 다음은 스케줄러

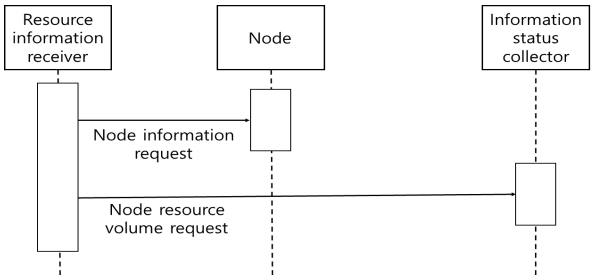

Fig. 7 Resource Information Receiver

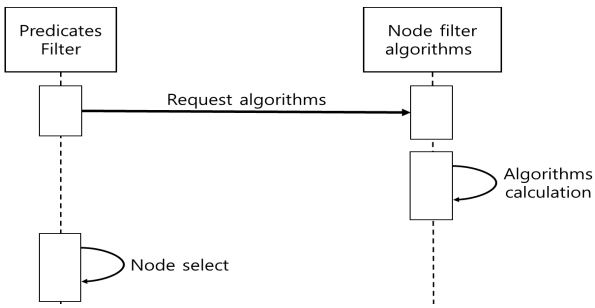

Fig. 8 Predicates Filter Unit

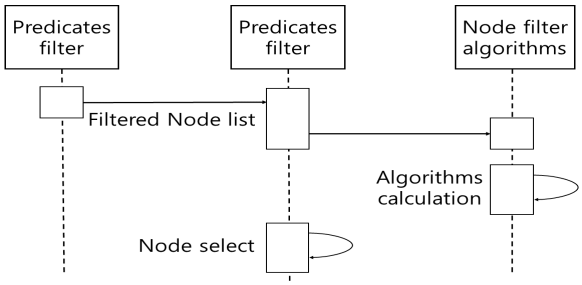

Fig. 9 Priorities Filter Unit

블록의 세부 기능이다. Fig. 5는 노드를 선택하는 노드 필터의 큰 구성이며, 사용자가 Pod 생성 요청을 하게 되면 스케줄러가 노드 필터기에게 노드 선택 요청을 한다. 노드 필터기에서 노드를 선택하는 알고리즘들의 계산을 통해 최종적으로 사용자에게 서비스를 제공한다. Fig. 6은 Fig. 5 노드 필터기로부터 Pod 할당을 위해 노드 필터링 요청을 받으면 predicates 필터기를 이용하여 일차적으로 노드 필터를 한 후 필터된 노드를 priorities 필터기를 이용해 Pod를 할당할 노드를 선택한다. Fig. 7은 스케줄러를 통해 노드를 선별하기 위해 자원 정보 리시버는 연결된 모든 노드의 정보와 자원 현황 정보를 요청한다. Fig. 8은 Predicates 필터링에서 노드 필터 알고리즘에 알고리즘을 요청하여 계산한 뒤 점수에 따라 일차적으로 노드를 선별한다. Fig. 9에서 Fig. 7의 1차로 필터 된 노드를 받은 후 노드 필터 알고리즘에 계산을 요청하여 가중치에 따라 최종 노드를 선택한다.

3.4 제안된 모니터링 블록

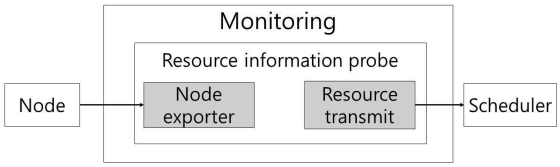

Fig. 10 Monitoring Block Architecture

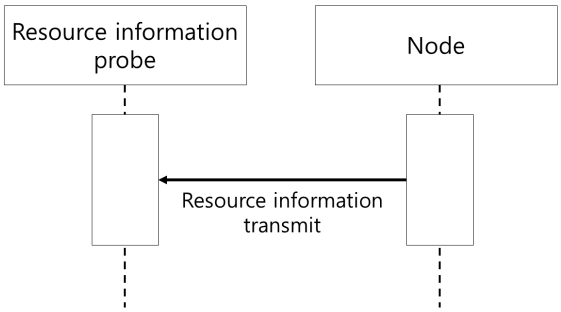

Fig. 11 Resource information Probe Unit

세션 테스트를 위해 우리가 제안한 Prometheus 기반의 모니터링 블록을 소개하겠다. Fig. 10은 모니터링 블록의 전체구조도이며, 모니터링에 필요한 자료를 노드 익스포터로부터 가져와 스케줄러에 필요한 데이터를 전송한다. 스케줄러에 필요한 데이터는 CPU, memory, 네트워크, I/O metric으로 하였다. 해당 데이터는 기존 스케줄러에서 고려하는 CPU, memory, I/O metric에서 네트워크 자원도 고려하기 위해 스케줄링 요소로써 추가하였다. 블록 구조로는 노드가 수집해 놓은 데이터를 가져오기 위한 자원 현황 수집기인 노드 익스포터와 스케줄러에 필요한 정보를 보내주기 위한 자원 전송자로 구성되어 있다. Fig. 11은 자원 현황 수집기이며, 각 노드에 존재하는 자원 현황 수집기가 데이터를 모아

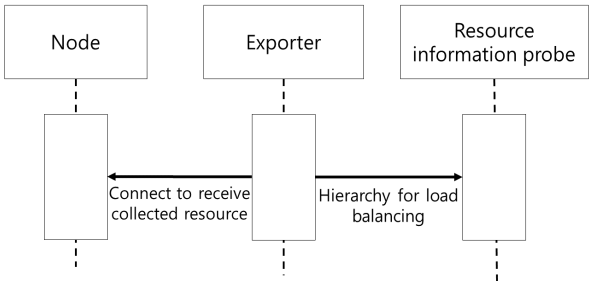

Fig. 12 Node Exporter

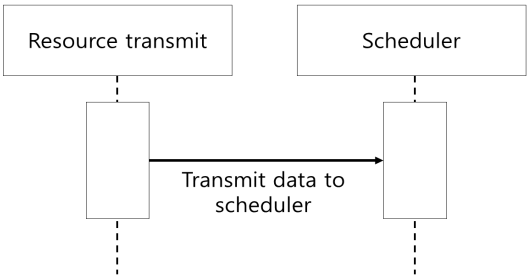

Fig. 13 Resource Transmit Unit

저장한다. 저장한 데이터는 컨테이너, Pod와 같은 분류 기준에 따라 분류되며, 한 곳에 모아 저장하거나 로그 파일 형태로 저장된다. 자원 현황 수집기는 해당 데이터를 가져온 뒤 스케줄러의 요청에 따라 필요한 데이터를 전송한다. Fig. 12는 자원 현황 수집기가 노드 또는 다른 자원 현황 수집기에 연결하기 위한 노드 익스포터다. 자원 현황 수집기가 노드 익스포터를 통해 노드에 연결하고 수집한 데이터를 가져오거나, 다른 자원 현황 수집기에 연결하여 부하 분산을 위한 계층 구조를 가진다. Fig. 13은 자원 현황 수집기가 수집한 데이터들을 모아 스케줄러에 필요한 데이터를 전송하는 유닛이다. 자원 현황 수집기가 수집한 데이터를 스케줄러의 요구에 맞추어서 필요한 데이터를 전송한다. 스케줄러는 자원 현황 수집기를 통해 자원 현황 정보를 전달받아 필터링이나 스케줄링 계산을 위해 사용한다.

3.5 세션 테스트 도구 구현

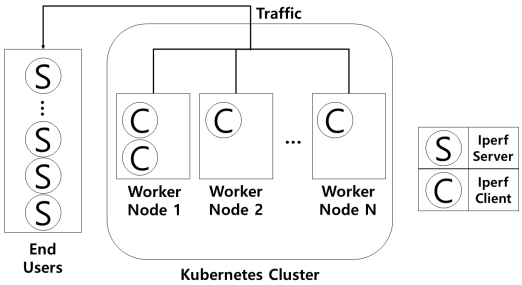

Fig. 14 Session Test Tool Architecture

Fig. 15 Node-Oriented Monitoring

Fig. 16 Resource-Oriented Monitoring

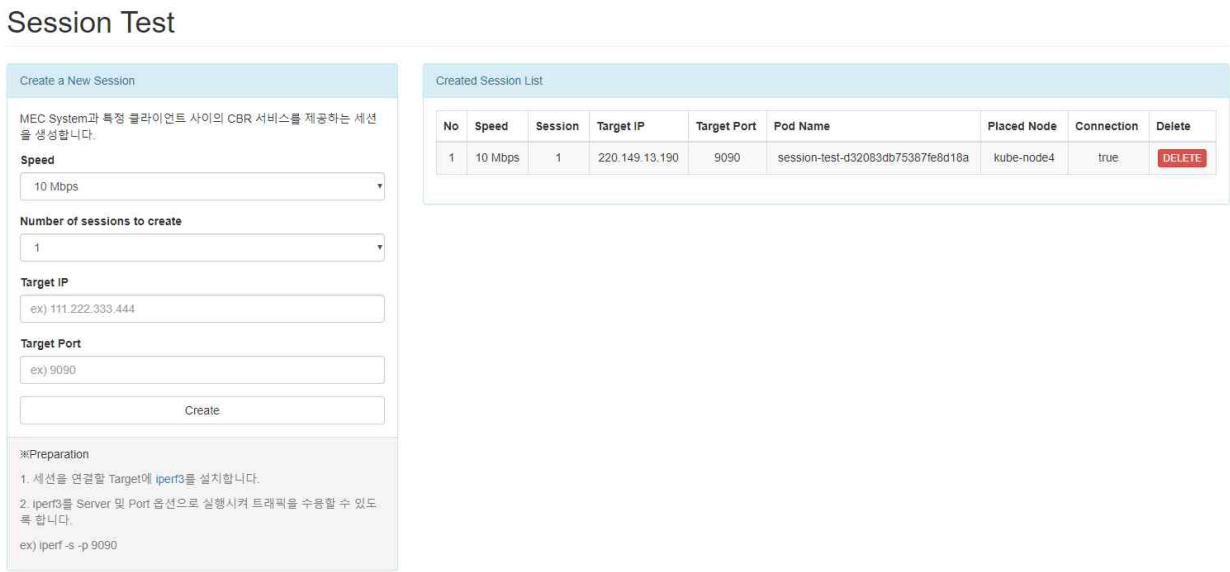

Fig. 17 Example of Session Test

일반적인 클라우드 서버의 취지는 물리적인 서버를 중앙집중식으로 구성하여 사용자와 거리가 있다. 그에 반해 MEC는 중앙집중식으로 구성된 클라우드 서버는 저지연을 만족하기 힘들기 때문에 마치 사용자 바로 옆에 서버가 존재하는 듯하여 저지연을 만족한다. 해당 시험은 저지연을 만족할수 있는 세션의 개수를 고려하여 진행하였다. 스케줄러 블록과 모니터링 블록을 토대로 테스트 도구를 개발하였다. Fig. 14는 세션 테스트 도구의 세션 테스트 기능 구조도이다. 네트워크 부하를 위해서 Packet generator tool인 iperf3를 사용하였다. Iperf client는 MEC에서 사용자에게 서비스를 제공한다고 가정을 할 때 발생하는 트래픽을 위한 packet generator이며, iperf server는 서비스를 받는 사용자들로 가정한다.

하나의 Kubernetes cluster 내부에 N개의 노드가 사용자에게 서비스를 제공한다고 가정할 때 하나의 세션당 하나의 iperf client를 사용한다. 테스트 도구의 웹 형태로 소개할 Fig. 15, Fig. 16과 Fig. 17을 개발하였다. 현재 구현된 웹 서비스는 모니터링과 세션 테스트 2가지 기능으로 구성되어 있다. Fig. 15와 Fig. 16은 하나로 구성된 모니터링 페이지이며, Kubernetes cluster 내부 노드의 자원 현황을 시각화하였다. 모니터링 페이지는 노드 관점, 자원관점으로 두 분류로 나누어지며, 측정되는 자원은 CPU 사용률, MEM 사용률, 네트워크 사용률로 구분된다. 노드 관점은 Fig. 15와 같다. Fig. 15는 Kubernetes cluster에 포함된 모든 노드를 마스터, 워커노드 순으로 정렬하고 노드별 사용 중인 자원량을 백분율로 표기하였다. Fig 16은 자원관점으로 시각화되며 측정에 사용되는 자원의 노드별 사용률을 그래프로 만들어 시각화한다. 모니터링 페이지는 Prometheus에서 측정된 결괏값을 REST API를 이용해 수집하여 수집된 값을 바탕으로 시각화한다. Fig. 17은 세션 테스트 페이지이며, MEC 세션 생성을 담당한다. 사용자는 CLI로 작성할 필요 없이 제공된 템플릿에서 세션을 생성할 목적지, throughput, 생성할 세션의 개수를 선택할 수 있다. 생성 요청을 하면 REST API를 이용하여 MEC cluster의 마스터 노드에 전송이 되어 session Pod가 생성이 되고 사용자가 지정한 양만큼의 트래픽이 발생하게 된다. 생성 완료가 되면 Fig. 17의 Created Session List 항목에 현재 생성된 Pod와 Pod의 생성 정보들이 제공된다. 테스트 종료를 위한 Pod 회수 명령이 제공된다.

4. 세션 테스트 도구 시험

4.1 시험 환경

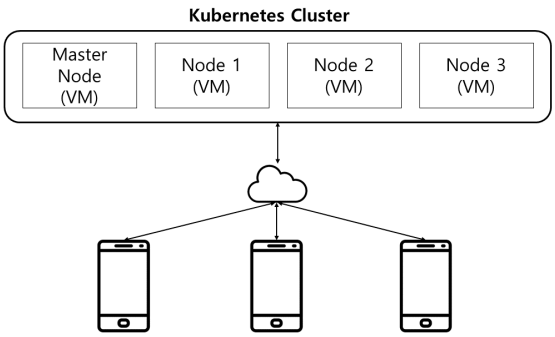

Fig. 18 Experimental Environment

제작한 세션 테스트 도구를 실험하기에 앞서 다음과 같이 실험환경을 설정하였다. 실험환경은 Fig. 18과 같이 100Mbps 망 및 하나의 물리 컴퓨터에서 4개의 가상머신을 구축하여 하나의 마스터 노드, 3개의 워커노드로 Kubernetes cluster를 구축하였다. 세션 수 증가를 위해 휴대용 기기인 안드로이드 기기 3개를 이용하였다. 휴대용 기기는 별도로 분리된 네트워크 환경에서 Iperf3를 이용하여 트래픽을 발생시켜 세션 수를 점진적으로 늘려가 시험 서버의 한계수용지점까지 시험하였다. 세션은 개당 7Mbps의 트래픽을 유발하며, 세션당 1분의 측정 시간을 가졌으며 지연율은 휴대용 기기와 서버 사이의 ICMP 프로토콜을 이용하여 평균값을 측정하였다.

4.2 시험 결과

100Mbps의 환경에서 7Mbps의 부하를 제공하는 세션은 총 13개의 세션이 유지가 되었다. 세션은 10개일 때, 즉 70Mbps의 트래픽이 시험 환경에서 MEC가 만족해야 하는 최대의 저지연율인 1ms 미만을 안정적으로 유지한 것을 Fig. 19의 그래프를 통해 확인하였다. 세션이 10개가 초과된 이후 MEC의 저지연율을 만족하지 못하고 지연시간이 지수적으로 증가함을 확인하였다.

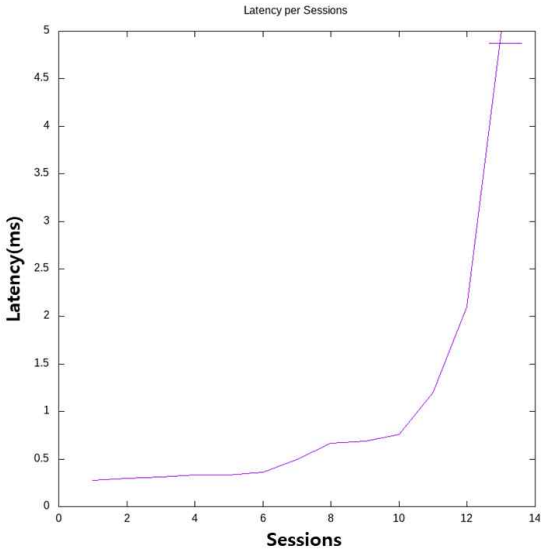

Fig. 19 Latency per Sessions

5. 결 론

5G 네트워크의 상용화에 따라 기하급수적으로 증가하는 데이터 처리가 요구되는 현재, 중앙집중 처리 클라우드의 한계가 생겨 MEC의 개념이 대두되었다. MEC는 사용자와 가까운 기지국에서 데이터를 처리하기 때문에 해당 기지국의 수용 가능한 세션의 한계점을 파악하여야 한다. 본 고에서 MEC 환경에서 사용하는 컨테이너 오케스트레이션 툴인 Kubernetes를 이용한 세션 테스트 도구를 제작하였다. 세션 테스트 도구는 기존의 Kubernetes가 스케줄링에 관여하지 않는 네트워크를 이용 하였으며, 네트워크 메트릭을 이용하여 적절한 Kuberentes 워커 노드에 배치되게끔 설정하였다.

시험 환경의 세션을 점진적으로 늘려나가 전체 대역폭의 약 70% 구간, 즉 10개이하의 세션을 유지할 때 MEC 목표 저지연율인 1ms가 만족되는 것을 확인하였다. 세션의 개수가 10개를 초과한 경우 Kubernetes cluster가 더 이상 저지연을 만족하지 못하고 기하급수적으로 대기시간이 길어지는 것을 확인하였다. 현재 시험 환경은 하나의 물리 컴퓨터에서 가상머신 4개를 만들고 측정된 결과값이기 때문에 실제 물리 서버를 사용하는 기지국과는 환경이 다르다는 한계점이 존재하여 개

선해나가야 할 것이다.

References

C. M. Aderaldo, N. C. Mendonc¸a, C. Pahl, and P. Jamshidi (2017). Benchmark requirements for microservices architecture research, *1st International Workshop on Establishing the Community-Wide Infrastructure for Architecture-Based Software Engineering,* 17(4), 8-13

J.M. Park, H.G. Jeon, I.G. Jeon (2018). Trends in MEC Technology for 5G Network Interworking, *The Journal of The Korean Institute of Communication Sciences,* 35(11), 3-10

Kubernetes (2015). *Configure Multiple Schedulers*, https://kubernetes.io/docs/tasks/extend-kubernetes/configure-multiple-schedulers/ (Accessed on Oct. 6th, 2020)

Kubernetes (2015). *Scheduling Policies,* https://kubernetes.io/docs/reference/scheduling/policies (Accessed on Oct. 6th, 2020)

Patel, M., Naughton, B., Chan, C., Sprecher, N., Abeta, S., & Neal, A. (2014). *Mobile-edge computing introductory technical white paper.* White paper, mobile-edge computing (MEC) industry initiative

Prometheus (2014). *What is Prometheus?, http://prometheus.io/docs/introduction/overview* (Accessed on Oct. 6th, 2020)

T.Y. Kim et al. (2020). Kubernetes Scheduler Framework Implementation with Realtime Resource Monitoring, *IEMEK J. Embed. Sys.,* 15(3), 129-137

Yala, Louiza, Pantelis A. Frangoudis, and Adlen Ksentini(2018). Latency and availability driven VNF placement in a MEC-NFV environment. *IEEE Global Communications Conference (GLOBECOM).,* Dec. 9-13, Abu Dhabi, United Arab Emirates, pp. 1-7

김 태 영 (Tae-Young Kim)
- (현재) 대구대학교 컴퓨터 정보공학부 재학 중
- 관심분야: 클라우드 컴퓨팅, 컨테이너 가상화

김 태 현 (Tae-Hyun Kim)
- (현재) 대구대학교 컴퓨터 정보공학부 재학 중
- 관심분야: 클라우드 컴퓨팅, 컴퓨터 보안

진 성 근 (Sunggeun Jin)
- 정회원
- 경북대학교 전자공학 공학사
- 경북대학교 전자공학 공학석사
- 서울대학교 무선통신 공학박사
- 한국전자통신연구원 선임연구원
- (현재) 대구대학교 컴퓨터 소프트웨어전공 부교수
- 관심분야: 클라우드 컴퓨팅, 5G/6G 무선 네트워크