논문 21-46-03-07 The Journal of Korean Institute of Communications and Information Sciences '21-03 Vol.46 No.03
https://doi.org/10.7840/kics.2021.46.3.466

마이크로서비스의 특징을 반영한 컨테이너 오케스트레이션 프레임워크 배치 기법

정 수 민◆, 김 정 구°, 염 근 혁*, 박 준 석**

Container Orchestration Framework Deployment Technique Based on the Feature of Microservice

Sumin Jeong◆, Jeong Goo Kim°, Keunhyuk Yeom*, Joonseok Park**

요 약

본 논문에서 마이크로서비스(microservice) 구조의 어플리케이션을 구성할 때, 마이크로서비스의 약결합(loosely-coupling) 특성을 반영한 배치 명세를 제안한다. 또한, 제안된 배치 명세 기반의 마이크로서비스 어플리케이션을 컨테이너 오케스트레이션 프레임워크(container orchestration framework)에 배치하는 방안을 제안한다. 제안된 기법을 Kubernetes의 기본 템플릿을 활용한 배치와의 차이점을 비교하고, 그 성능을 측정하기 위하여 시뮬레이션을 통하여 마이크로서비스 응답 지연시간을 측정하였다. 그 결과, Kubernetes의 기본 템플릿을 활용한 경우는 마이크로서비스 구조를 반영하지 않아 무작위로 배치되는 경향을 보였으나 제안된 기법의 경우 응답 지연시간을 줄이는 적절한 위치에 마이크로서비스가 배치됨을 확인하였다. 그리고 기존의 방법에 비해 평균 응답 지연시간이 약 1.95ms 감소하였다.

키워드 : 클라우드 컴퓨팅, 마이크로서비스, 마이크로서비스 배치, 컨테이너, 컨테이너 오케스트레이션
Key Words : Cloud Computing, Microservice, Microservice deployment, Container, Container Orchestration

ABSTRACT

To construct applications of microservice, a deployment specification reflected on the loosely-coupling characteristics is proposed in this paper. And a technique to place applications of the proposed deployment specification-based microservice architecture in the container orchestration framework is also proposed. The differences between the deployment using Kubernetes' basic templates and the proposed techniques are compared, and to measure its performance, a microservice response latency is measured. As a result, Kubernetes' basic templates tended to be randomly deployed because they did not reflect the microservice structure, in the case of the proposed technique, however microservice is placed in an appropriate location to reduce response latency. In addition, the average response latency is reduced by approximately 1.95 ms compared to traditional methods.

※ 이 논문은 부산대학교 기본연구지원사업(2년)에 의하여 연구되었음
◆ First Author : Pusan National University Department of Information Convergence Engineering, sumin2708@gmail.com, 학생회원
° Corresponding Author : Pusan National University School of Computer Science and Engineering, kimjg@pusan.ac.kr, 종신회원
* Pusan National University School of Computer Science and Engineering, yeom@pusan.ac.kr, 정회원
** Pusan National University Research Institute of intelligent Logistics Big data, pjs50@pusan.ac.kr, 정회원
논문번호 : 202012-310-B-RN, Received December 8, 2020; Revised December 30, 2020; Accepted January 4, 2021

Ⅰ. 서 론

IT 서비스를 제공하는 기업들이 최근 클라우드 컴퓨팅 트렌드인 클라우드-네이티브(cloud-native) 관점에서 컨테이너 기반의 마이크로서비스 개발에 노력을 기울이고 있다[1]. 컨테이너[2] 환경은 종래에 활용되던 하이퍼바이저(hypervisor) 기반의 가상화 기술보다 경량화된 것으로, 마이크로서비스[3]를 동작시키기 쉬운 가상화 환경이다.

마이크로서비스는 기존의 PaaS(Platform-as-a-Service)와 SaaS(Service-as-a-Service)[4]에서 주로 사용되는 모놀리식(monolithic)[5] 서비스와 대비되는 구조로, 독립적으로 동작 가능한 작은 규모의 서비스를 뜻한다. 이러한 마이크로서비스들을 묶어 어플리케이션의 형태로 구현한 마이크로서비스 어플리케이션은 약 결합(loosely coupling)과 높은 재사용성(high reusability)을 갖는 새로운 클라우드 컴퓨팅 서비스의 패러다임이다.

대형 클라우드 벤더들은 클라우드 컴퓨팅 기술의 트렌드에 맞춰 컨테이너 기반 마이크로서비스를 지원하기 위한 컨테이너 오케스트레이션 프레임워크와 독자적인 마이크로서비스 배포 플랫폼을 제시하고 있다. 다양한 컨테이너 오케스트레이션 플랫폼 중 현재 주로 활용되는 플랫폼으로는 Google의 Kubernetes와 오픈소스인 Docker Swarm이 있으며, Google의 Kubernetes 플랫폼의 구조와 기능들은 컨테이너 오케스트레이션 플랫폼의 사실상 표준(de-facto standard)으로 인정받고 있다[6].

컨테이너를 활용한 마이크로서비스 어플리케이션은 마이크로서비스가 탑재된 컨테이너의 상태, 배치 공간의 특성, 결합 마이크로서비스 간의 배치 위치 등의 다양한 특성에 따라 어플리케이션의 성능 차이가 발생한다. 그러나, Kubernetes를 포함한 대부분의 컨테이너 오케스트레이션 프레임워크들은 마이크로서비스 어플리케이션을 배치할 때, 마이크로서비스의 동작 특성인 약결합을 고려하지 않고, 컨테이너 요구 리소스 기반의 일률적인 배치를 진행한다[7]. 즉, 마이크로서비스 간의 결합을 고려하지 않는 배치를 진행하므로 마이크로서비스 어플리케이션 성능에 영향을 주는 호출 지연시간이 발생할 수 있다.

따라서, 본 논문에서는 컨테이너 기반 마이크로서비스 어플리케이션을 구성할 때, 마이크로서비스의 약결합 특성을 반영한 배치 명세를 제안한다. 그리고 제안된 배치 명세 기반의 마이크로서비스 어플리케이션을 컨테이너 오케스트레이션 프레임워크에 배치하는 방안을 제안한다.

Ⅱ. 컨테이너 오케스트레이션 프레임워크

2.1 마이크로서비스

마이크로서비스는 팀 단위의 개발이 가능한 작은 규모의 서비스이며, 추상화된 인터페이스를 통해 약결합 구조를 형성한다[3]. Kalske 등[8]은 마이크로서비스 구조의 발전 방향과 문제점을 기술적인 관점과 구조적인 관점에서 분류하고 정리하였다. 기술적 관점의 경우 테스팅 시스템의 부재, 서로 다른 마이크로서비스 통합의 어려움, 마이크로서비스 구조에서 불안정한 마이크로서비스를 특정하는 방법 등이며, 구조적 관점에서는 마이크로서비스의 효율성을 고려한 구성, 서비스들의 논리적인 구분 방법 등이 해결해야 할 문제라고 주장했다.

김대호 등[9]은 UML 설계자료를 토대로 모놀리식 어플리케이션의 구성요소들을 사용자 인터페이스를 제공하는 Boundary, 비즈니스 로직을 제공하는 Control, 및 데이터 저장 및 처리를 담당하는 Entity의 세 가지 타입의 마이크로서비스로 분류하여 마이크로서비스로 추출하는 기법을 제안하였다.

그림 1에 모놀리식 구조와 마이크로서비스 구조의 차이점을 나타내었다[9]. 그림과 같이 모놀리식 구조는 기능 단위의 개별적 서비스 제공이 가능한 구조가 아닌 어플리케이션이 하나의 단위로 구성되어 제공된다. 이와 달리 마이크로서비스는 각각의 기능이 별개의 독립적인 서비스로 동작하고, 이들을 약결합하는 방식으로 연결되며, 마이크로서비스별로 개별적인 DB가 작성되어 데이터의 충돌을 방지하는 형태로 구성된다.

Mayer 등[10]은 마이크로서비스에서 모니터링이 필요한 정보는 Static, Dynamic, Infrastructure에 대한 정보들로 구분하였다. 여기서 Static 정보란 서비스를 활용하는 데 있어서 변화하지 않는 기본 정보를 말하

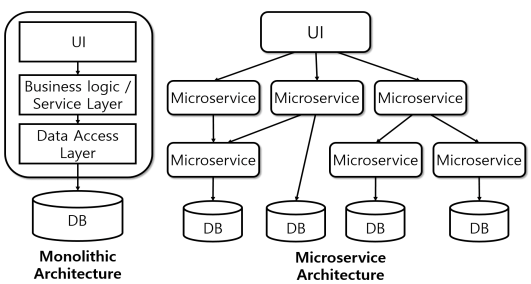


그림 1. 모놀리식 구조와 마이크로서비스 구조
Fig. 1. Monolithic architecture and microservice architecture.

며, API 인터페이스, 개발자 및 마이크로서비스 배포 정보 등의 서비스 기반을 뜻한다. Dynamic 정보는 서비스를 활용하면서 지속해서 변화하는 정보를 말하며, 서비스 이용 인원, 서비스 이용 횟수 등의 사용자 요청 정보의 대부분이 이에 속한다. 또한, Infrastructure 정보는 해당 서비스가 동작하고 있는 하드웨어 자원에 대한 정보 대부분이 이에 속한다.

2.2 Kubernetes 프레임워크의 분석

본 절에서는 마이크로서비스의 약결합 특성을 반영하여 컨테이너에 배치하는 명세를 개발하기 위해 컨테이너 오케스트레이션 프레임워크를 분석한다.

표 1은 Kubernetes에서 마이크로서비스의 운영을 위해 필요한 컴포넌트를 분석한 것이다. 표에서 Pod는 마이크로서비스의 동작을 수행하며, Service는 Pod와 클라이언트의 연결을 중재하는 인터페이스의 역할을 한다. Controller에 해당하는 컴포넌트들은 Pod의 기능을 보조하는 역할이며, 고가용성, 상시가용, 주기적 실행 등 Pod의 기능에서 필요한 상황에 따라 활용한다. 따라서, 마이크로서비스의 동작을 클라이언트에서 수행하기 위해서는 Service와 Pod가 연결된 형태로 제공되어야 한다. 이에 따라, Pod와 Service는 마이크로서비스를 클라이언트에 제공할 때, 필수적인 요소라 할 수 있다.

그림 2에 Pod와 클라이언트의 통신 구조를 나타내었다. Pod와 클라이언트는 직접 통신하지 않고

표 1. 마이크로서비스 동작에 필요한 구성요소
Table 1. Componets required for microservice operation.

Module Name	Name	Essential	Description
Interface	Service	O	Network communication interface component
Function	Pod	O	Basic microservice component
Controller	Deployment	X	High availability microservice component
	DeamonSet	X	Always-on microservice component
	CronJob	X	Periodic microservice component

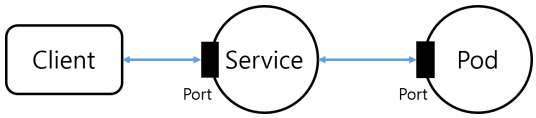


그림 2. Pod와 클라이언트의 통신 구조
Fig. 2. Communication structure between Pod and Client.

Service 인터페이스 컴포넌트를 매개체로 통신한다. 그림 2의 두 컴포넌트를 만드는 Kubernetes 템플릿을 그림 3에 나타내었다. 그림에서 Pod와 Service는 라벨과 컨테이너 포트를 통해 1:1 매칭되는 것을 알 수 있다. 따라서 마이크로서비스를 Kubernetes에 배치할 때, 하나의 마이크로서비스에 Service와 Pod 컴포넌트가 각각 하나씩 배치된다. 또한, 그림 3에서 마이크로서비스를 제공하는 템플릿에는 Kubernetes에서 컨테이너 배치를 위한 요소들이 나열되어 있다. 그림과 같은 컨테이너 배치 요소는 컨테이너를 배치하고 관리하는데 중요한 명세 항목들이지만, 이들이 마이크로서비스 어플리케이션의 약결합을 통한 동작을 명세하지는 못한다.

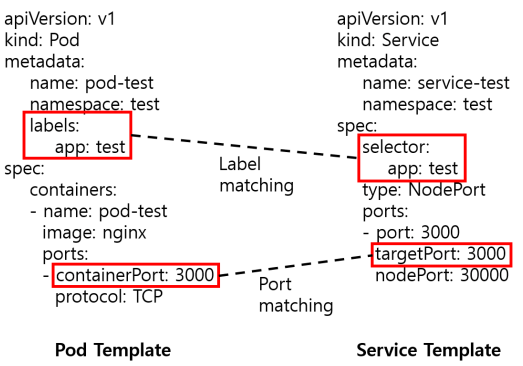


그림 3. Pod와 Service를 만드는 kubernetes 템플릿
Fig. 3. Kubernetes template for creating Pod and Service.

Ⅲ. 마이크로서비스 배치기법

3.1 마이크로서비스 특성 정의

마이크로서비스가 Kubernetes에 배치되기 위해서는 마이크로서비스의 특성을 반영하는 새로운 템플릿 정의가 필요하다. 따라서 마이크로서비스의 특성을 정의하고, Kubernetes의 필수요소들을 나열하여 표 2와 같은 컨테이너 오케스트레이션 프레임워크의 템플릿으로 변환 가능한 명세 항목을 정의하였다.

표 2에서 특성명의 MS-Number는 마이크로서비스의 특성을 정의한 항목이다. CO-Number는 컨테이너

표 2. 마이크로서비스의 특성과 컨테이너 오케스트레이션 필수요소 분류
Table 2. Container orchestration framework requirement and microservices featrue specification.

Feature Name	Description
MS-01	Specifying the BCE(Boundary-Control-Entity) type of microservice
MS-02	Specifying the function feature of microservice
MS-03	Specifying microservice connected to input/output
CO-01	Specifying unique name on container orchestration framework
CO-02	Specifying containerization microservice
CO-03	Specifying interface port on container orchestration framework

오케스트레이션 프레임워크인 Kubernetes 템플릿의 작성에 필요한 항목이다.

MS-01은 마이크로서비스 구조가 BCE(Boundary -Control-Entity) 패턴을 갖도록 하는 특성이고, MS-02는 마이크로서비스들 사이의 약결합 특성으로 해당 마이크로서비스가 갖는 특징적인 기능을 나타낸다. 그리고 MS-03은 마이크로서비스의 입/출력과 연결되는 마이크로서비스를 나타낸다.

CO-01은 Kubernetes 내에서 개별 마이크로서비스를 특정하고, CO-02는 개별 마이크로서비스가 배치될 때 구동되는 기능을 특정한다. 그리고 CO-03은 마이크로서비스의 포트와 Kubernetes의 포트의 연결을 특정한다.

그림 4는 마이크로서비스의 특성 요소를 JSON (JavaScript Object Notation) 형식의 템플릿으로 작성한 것이다.

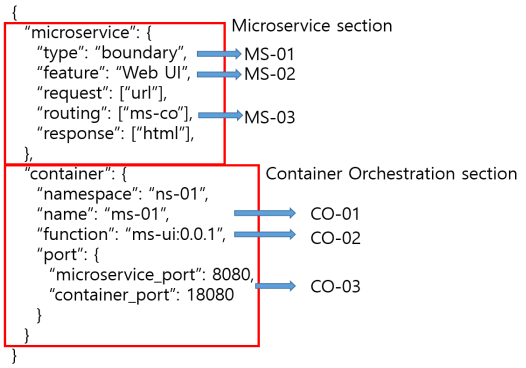


그림 4. 마이크로서비스의 특성을 반영한 명세 템플릿
Fig. 4. Specification template based on the feature of microservice.

3.2 마이크로서비스 특성을 반영한 배치기법

본 절에서는 1절에서 제안한 마이크로서비스의 특성과 모니터링으로 획득 가능한 Kubernetes 배치 공간의 인프라 특성을 반영한 마이크로서비스 배치기법을 제안한다.

배치기법에 대한 개략적인 구조를 그림 5에 나타내었다. 그림에서 마이크로서비스의 배치는 적합성 판별기를 통해 산출된 레이팅 데이터를 기반으로 진행된다.

적합성 판별기는 마이크로서비스 배치 지점의 물리적 여유 공간을 반영하는 1차 적합성 판별과 마이크로서비스의 배치 지점에 따른 특성을 반영하는 2차 적합성 판별로 구분한다.

배치 지점의 물리적 리소스(CPU, Memory, Storage)의 적합성을 수치화하는 수식을 수식 (1)에서 (4)에 나타내었다.

$$(\text{standard level}) = \frac{(\text{CPU maximum request})}{(\text{CPU minimum request})} \quad (1)$$

식 (1)에서 CPU maximum request는 각 배치 지점에서 하나의 컨테이너에 할당 가능한 최대 수치의 CPU를 말하고, CPU minimum request는 하나의 컨테이너에 할당 가능한 최소 수치의 CPU를 말한다. Kubernetes는 CPU minimum request로 0.05 core를 제공한다.

$$(\text{Memory minimum request}) = \frac{(\text{Memory maximum request})}{(\text{standard level})} \quad (2)$$

식 (2)는 정의한 standard level을 활용하여 하나의 컨테이너에 할당할 최소 수치의 Memory를 계산하기 위한 식이다. Memory maximum request는 각 배치 지점에서 하나의 컨테이너에 할당 가능한 최대 수치의 Memory를 말한다.

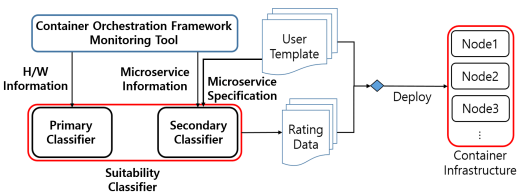


그림 5. 제안된 배치기법의 개략도
Fig. 5. Proposed deployment technique schematic diagram.

$$(\text{Storage minimum request}) = \frac{(\text{Storage maximum request})}{(\text{standard level})} \quad (3)$$

식 (3)은 정의한 standard level을 활용하여 하나의 컨테이너에 할당할 최소 수치의 Storage를 계산하기 위한 식이다. Storage maximum request는 각 배치 지점에서 하나의 컨테이너에 할당 가능한 최대 수치의 Storage를 말한다.

$$(\text{Primary Suitability}) = \frac{(\text{available CPU})}{(\text{CPU minimum request})} + \frac{(\text{available Memory})}{(\text{Memory minimum request})} + \frac{(\text{available Storage})}{(\text{Storage minimum request})} \quad (4)$$

식 (4)는 1차 적합성을 수치화하는 방법을 보인다. 1차 적합성은 각 노드의 여유 리소스를 앞서 계산한 최소 수치의 CPU, Memory, Storage를 나누는 것으로 계산한다.

2차 적합성 판별을 위한 기준을 표 3에 나타내었다. 표에 제시된 기준은 마이크로서비스의 특성을 고려하기 위한 항목들로 유사성(Similarity), 인프라 환경(Infrastructure environment), 연결성(Connectivity), 성능(Performance)의 네 가지 지표를 활용한다.

유사성은 비슷한 마이크로서비스가 배치된 배치 지점을 선택하기 위한 항목이다.

인프라 환경은 배치 지점 중 마이크로서비스의 밀집도 및 컨테이너 수용량과 같은 배치 환경적 이점을 갖는 배치 지점을 선택하기 위한 항목이다.

연결성은 Boundary-Control-Entity 패턴에 따른 연결 관계가 존재하는 마이크로서비스의 연결 지연 최소화를 위한 배치 지점을 선택하기 위한 항목이다.

성능의 저 지연성(Law Latency)은 배치 지점별 마이크로서비스 응답 지연 성능을 고려한 선택을 위한 항목이다. 응답 지연 시간은 마이크로서비스의 성능을 비교하는 주요한 항목이다. 예를 들어, 마이크로서비스를 배치하는 상용서비스인 AWS Lambda에서는 모니터링 도구로 X-Ray를 활용하는데, 성능 측정을 응답 지연 시간을 활용하고 있다.

성능의 고가용성(High Availability) 부분은 마이크로서비스의 컨테이너 재시작 횟수가 가장 적은 배치 지점을 선택하기 위한 항목이다.

표 3. 2차 적합성 판별표
Table 3. Secondary suitability distinction table.

Category	Category detail	Description
Similarity		Node with same BCE (Boundary-Control-Entity) type in microservice specification
		Node with same feature in microservice specification
Infrastructure environment		Node with minimum number of microservices
		Node with maximum primary classifier score
Connectivity		Node with microservice in Boundary-Control- Entity connection relationship
Performance	Low Latency	Node with minimum response latency
	High Availability	Node with minimum container restart time

Ⅳ. 모의실험 및 고찰

본 장에서는 제안하는 마이크로서비스 특성을 반영한 컨테이너 오케스트레이션 프레임워크 배치기법의 성능을 검증하기 위한 모의실험을 진행하였다.

4.1 모의실험 환경 및 설계

모의실험에 사용된 환경은 표 4와 같다.

모의실험은 두 가지 경우에 대해서 진행한다. 첫째, 기존의 Kubernetes 프레임워크의 기본 템플릿을 활용한 배치와 제안하는 기법을 활용한 배치의 분포를 비교한다. 이는 Kubernetes에서 제공하는 컨테이너 배치 방식과 제안하는 기법이 마이크로서비스의 약결합 기반 동작을 반영하는지를 확인한다.

표 4. 모의실험 환경
Table 4. Simulation environment.

Category	Name	Role	Spec
H/W	kube1	Master Node	CPU: 2 core RAM: 4096 MB Disk: 50 GB
	kube2	Worker Node	CPU: 1 core RAM: 4096 MB Disk: 50 GB
	kube3	Worker Node	CPU: 1 core RAM: 4096 MB Disk: 50 GB
S/W	Kubernetes	Framework	v 1.19.4
	prometheus	Monitoring Agent	v 2.22.2
	grafana	Monitoring Agent	v 7.3.4

둘째, Kubernetes 프레임워크에서 제공하는 기본 템플릿을 활용한 배치와 제안하는 기법을 활용한 배치에서 타입별 마이크로서비스의 응답 지연시간을 측정한다. 마이크로서비스의 응답 지연시간은 성능을 나타내는 주요한 지표로써 응답 지연시간 측정을 활용하여 제안하는 기법의 성능 개선 정도를 확인한다.

모의실험은 Boundary-Control-Entity로 연결되는 마이크로서비스 어플리케이션을 배포하고, 타입별 마이크로서비스가 배포될 때의 상황을 확인한다.

4.2 모의실험 결과

모의실험을 진행하기 전 상태의 Kubernetes 프레임워크의 여유 하드웨어 리소스를 표 5에 나타내었다. 표를 통해 컨테이너 기반 마이크로서비스가 배치될 수 있는 충분한 공간이 확보되어 있음을 알 수 있다.

배치 순서에 따른 1차 적합성을 표 6에 나타내었다. 표에서 Boundary 마이크로서비스를 배치하는 상황인 Initial 상태에서 1차 적합성은 CPU의 여유 공간이 가장 많은 kube1이 가장 높았다. 추후 진행되는 Control, Entity 마이크로서비스를 배치하는 실험에서도 마찬가지로 CPU 여유 공간에 영향을 많이 받는 경향을 보인다. 이는 가용량이 적은 CPU의 최소치를 기준으로 구간을 나눈 것이 비교적 가용량이 많은 하드웨어 리소스들의 반영률은 낮춘 것으로 판단된다.

Boundary 마이크로서비스를 배치하는 경우에 대한 2차 적합성을 표 7에 나타내었다. 모의실험에서 Boundary 마이크로서비스를 배치하는 경우는 모든 배치 지점에 다른 마이크로서비스가 배치되지 않은 상태이므로, 유사성과 연결성에 대해서는 고려하지 않는다. 따라서, 최초로 배치되는 마이크로서비스는 인프라 환경과 성능 요소를 활용한 배치가 진행되고, 표

표 5. 모의실험 진행 전 여유 하드웨어 리소스여유 공간
Table 5. Free hardware space at before simulation.

Node Name	CPU (core)	Memory (GB)	Storage (GB)
kube1	0.95	3.4712	23.917
kube2	0.8	3.7057	23.987
kube3	0.8	3.7057	23.987

표 6. 배치 순서별 1차 적합성
Table 6. Primary suitability by deployment order.

state	kube1	kube2	kube3
Initial	55.051	47.389	47.389
After Boundary	50.153	47.389	47.389
After Control	47.153	47.389	47.389

표 7. 초기 상태에서 2차 적합성
Table 7. Secondary suitability at initial state.

Category	kube1	kube2	kube3
Similarity #1	0	0	0
Similarity #2	0	0	0
Infrastructure environment #1	0	1	1
Infrastructure environment #2	1	0	0
Connectivity	0	0	0
Performance #1	1	0	0
Performance #2	1	0	0
Total	3	1	1

에서 인프라 환경과 성능적 요소가 가장 높은 kube1이 최고의 2차 적합성을 갖게 된다.

Control 마이크로서비스를 배치하는 경우에 대한 2차 적합성은 표 8에 나타내었다. Boundary 마이크로서비스가 kube1에 배치되면서 마이크로서비스 응답 지연시간이 증가한다. 이를 반영하면 kube2가 배치 지점의 응답 지연시간 성능적 측면은 kube1 보다 낮은 지연을 보였으나, kube1의 인프라 환경의 우수성과 Boundary 마이크로서비스와의 연결성이 생기면서 kube1이 가장 높은 적합성을 갖게 된다.

Entity 마이크로서비스를 배치하는 경우에 대한 2차 적합성은 표 9에 나타내었다. Boundary와 Control 마이크로서비스가 kube1에 배치되면서 하드웨어 리소스 사용량이 매우 증가하여 kube2와 kube3가 인프라 환경 측면에서 좋은 성능을 보인다. kube1은 기 배치된 마이크로서비스를 통해 마이크로서비스 간의 연결성을 가지고 있지만, kube2와의 응답 지연시간 성능과 인프라 환경의 차이로 인해 2차 적합성은 kube2가

표 8. Boundary 마이크로서비스 배치 후 2차 적합성
Table 8. Secondary suitability at After Boundary state.

Category	kube1	kube2	kube3
Similarity #1	0	0	0
Similarity #2	0	0	0
Infrastructure environment #1	0	1	1
Infrastructure environment #2	1	0	0
Connectivity	1	0	0
Performance #1	0	1	0
Performance #2	1	0	0
Total	3	2	1

표 9. Control 마이크로서비스 배치 후 2차 적합성
Table 9. Secondary suitability at After Control state.

Category	kube1	kube2	kube3
Similarity #1	0	0	0
Similarity #2	0	0	0
Infrastructure environment #1	0	1	1
Infrastructure environment #2	0	1	1
Connectivity	1	0	0
Performance #1	0	1	0
Performance #2	1	0	0
Total	2	3	2

가장 높음을 알 수 있다.

적합성 판별을 통한 배치 지점 선택은 표 10 에 나타내었다. 표에서 1차와 2차 적합성을 토대로 배치 지점별 순위를 책정하고, 순위 기반의 가중치를 활용하여 배치 지점을 선택하였다. 이는, 1차와 2차 적합성의 기준과 범위가 다르고, 하나의 판별 단계에 치중된 결과를 갖지 않도록 순위별 가중치를 매기는 것으로 최종 선택을 진행하였다.

Kubernetes 프레임워크의 기본 배치와 제안하는 기법을 활용한 배치 분포 차이를 표 11에 나타내었다. Boundary, Control 마이크로서비스의 경우 두 배치 방식 모두 하나의 노드에 집중되어 배치되는 것을 확인할 수 있었으나, Entity의 경우 기존의 배치방법은 kube1과 kube2의 양쪽에 분산 배치됨을 알 수 있다. Kubernetes 프레임워크가 마이크로서비스 구조를 반영하여 마이크로서비스의 실행성능을 보장한다면, 제안된 기법의 배치 결과와 같이 배치 지점의 변동은 발생하지 않아야 한다. 따라서, Kubernetes 프레임워크는 마이크로서비스 구조를 반영한 배치를 지원하지 않으며, 프레임워크 상태에 따른 배치를 하고 있음을

표 10. 최종 순위표 및 선택된 노드
Table 10. Final suitability result and selected node.

	kube1	kube2	kube3	Select Node
Initial	1st(3)	2nd(2)	2nd(2)	kube1
	1st(3)	2nd(2)	2nd(2)	
After Boundary	1st(3)	2nd(2)	2nd(2)	kube1
	1st(3)	2nd(2)	3rd(1)	
After Control	3rd(1)	1st(3)	1st(3)	kube2
	2nd(2)	1st(3)	2nd(2)	

표 11. Kubernetes 기본 템플릿 배치와 적합성 판별에서 선택된 배치 분포
Table 11. Difference of deployment distribution, kubernetes basic template and determining suitability.

Service Type	Node name	Kubernetes (time)	Suitability (time)
Boundary	kube1	0	10
	kube2	0	0
	kube3	10	0
Control	kube1	0	10
	kube2	10	0
	kube3	0	0
Entity	kube1	5	0
	kube2	5	10
	kube3	0	0

알 수 있다.

마이크로서비스의 응답 지연시간 측정은 총 100회를 실시하여 최솟값부터 50회를 추출하였다. 이는, 순간적인 네트워크 환경에 따른 변수를 최소화하여 실험을 진행하기 위함이다.

Kubernetes 프레임워크의 기본 템플릿을 활용한 배치와 제안한 기법을 활용한 배치의 평균 응답 지연시간 차이를 그림 6에 나타내었다. 그림에서 기존의 Kubernetes 배치는 평균 응답 지연이 10.15ms 발생하였고, 제안하는 기법의 지연은 8.2ms로 평균 응답 지연이 1.95ms 개선되었다.

응답 지연시간 측정 결과에서 Boundary 마이크로서비스는 평균 응답 지연시간이 가장 높으며, 기존 기법과 제안하는 기법의 차이가 미미하다. 이는 Boundary 마이크로서비스가 갖는 특징과 연관이 있다. Boundary 마이크로서비스는 마이크로서비스 어플리케이션에서 클라이언트 인터페이스의 역할을 한다.

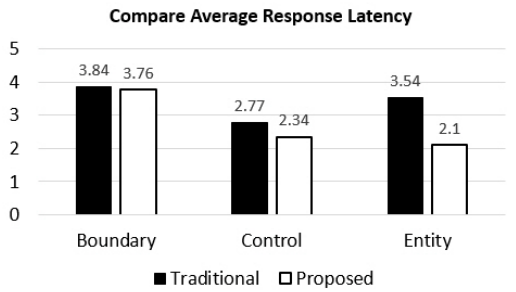


그림 6. 기존 방법과 제안하는 방법의 응답 지연시간 차이
Fig. 6. Compare response latency between traditional method and proposed method.

따라서, 클라이언트와 Boundary 마이크로서비스 사이의 지연시간을 측정하는 것이 되므로, 내부적인 흐름보다는 네트워크 상황에 따른 지연이 발생하게 된다. 모의실험은 안정적인 네트워크 환경하에서 진행하였기 때문에, 두 기법의 차이가 미미하게 나타난 것으로 판단된다.

Control 마이크로서비스 경우 응답 지연은 기존의 기법에 비해 약 0.43ms 단축되었다. 이는 Control 마이크로서비스를 배치할 때 고려한 2차 적합성에서의 연결성을 고려한 배치 지점 선택이 해당 실험에서 나타난 것으로 판단된다.

Entity 마이크로서비스의 경우는 약 1.44ms 단축되었다. 이 경우는 Kubernetes에서 마이크로서비스 구조를 배치할 때, 동작 성능을 고려하지 않음을 보여준다. 기존의 기법을 활용하는 Entity 배치 지점에서의 평균 응답 지연은 3.54ms로 Control 마이크로서비스의 2.77ms와 0.77ms의 차이를 보인다. 이는 외부 네트워크 응답 지연을 고려하는 Boundary 마이크로서비스와 비슷한 수준이다. 따라서, 이 결과는 단순 인프라 환경에 따른 배치와 약결합을 고려한 배치의 차이를 보여주는 결과라고 판단된다.

V. 결 론

본 논문에서는 컨테이너 기반 마이크로서비스를 컨테이너 오케스트레이션 프레임워크에 배치할 때, 마이크로서비스의 모니터링을 통해 획득한 정보와 명세 정보를 활용하여 배치하는 기법을 제안하였다.

기존의 컨테이너 오케스트레이션 프레임워크에서 제공하는 템플릿을 활용한 자동 배치는 마이크로서비스의 특성을 반영하는 명세를 작성하지 못하며, 배치되는 마이크로서비스들은 프레임워크의 상황만을 반영하여 배치한다. 이는 본 논문의 모의실험 환경하에서 기본 템플릿을 활용한 배치 분포 분석을 통해 알 수 있다. 마이크로서비스의 약결합을 반영한 배치를 시행한다면, 같은 상황에서 같은 배치가 이루어져야 한다. 하지만 기존 기법인 템플릿을 활용한 배치는 같은 상황에서도 같은 배치가 이루어지지 않았으며, 이는 마이크로서비스의 동작을 고려한 배치가 아님을 말한다. 반면, 동일한 모의실험에서 제안하는 배치기법은 일정한 배치 분포를 보였고, 이는 현재 배치하려는 프레임워크의 상황뿐 아니라, 마이크로서비스의 동작 특성에 따른 특징이 반영된 결과이다.

또한, 마이크로서비스 어플리케이션을 배치하는 경우의 성능 개선도를 확인하기 위해 기존 기법과 제안하는 기법의 응답 지연시간 측정 및 비교 평가를 실시하였다. 모의실험은 Boundary-Control-Entity로 연결되는 마이크로서비스 어플리케이션을 배치하는 실험을 기존 기법과 제안하는 기법을 활용하여 각각 진행하였다. 결과로 Boundary 마이크로서비스의 응답 지연은 약 0.08ms가 단축되었고, Control 마이크로서비스는 약 0.43ms, Entity 마이크로서비스는 1.44ms 단축되었다. 따라서, 평균 응답 지연이 약 1.95ms 정도 단축되는 결과를 보였다. 이를 통해 마이크로서비스의 약결합적 특성을 반영하여 어플리케이션을 배치함으로써, 서비스 응답 지연시간을 개선할 수 있음을 확인하였다.

본 논문에서 제안한 2차 적합성 판별기의 경우 마이크로서비스가 컨테이너 오케스트레이션 프레임워크에서 제공하는 메트릭을 기반으로 한 적합성 판별을 실시하므로 마이크로서비스가 가질 수 있는 특징 전체를 반영할 수 없다. 따라서 마이크로서비스의 기능들을 분류하는 방법에 관한 연구와 분류된 기능에 따른 특징 명세를 추출하는 방법에 관한 연구가 요구된다.

References

[1] D. K. Yoon, *Cloud-native technology trends* (2019), 2019. 12. 07, https://www.ceart.kr/web/board/BD_board.view.do?domainCd=2&bbsCd=1033&bbscttSeq=20190516161927708&monarea=00008

[2] S. Soltesz, H. Pötzl, M. E. Fiuczynski, A. Bavier, and L. Peterson, “Container-based operating system virtualization: A scalable, high-performance alternative to hypervisors,” in *Proc. 2nd ACM SIGOPS / EuroSys Eur. Conf. Comput. Syst. 2007*, pp. 275-287, Lisbon, Portugal, Mar. 2007.

[3] F. Rademacher, S. Sachweh, and A. Zundorf, “Differences between model-driven development of service-oriented and microservice architecture,” *2017 IEEE Int. Conf. Software Architecture Wkshps.*, pp. 38-45, Gothenburg, Sweden, Apr. 2017.

[4] D. Rani and R. K. Ranjan, “A comparative study of SaaS, PaaS, and IaaS in cloud computing,” *IJARCSSE*, vol. 9, no. 6, pp. 458-461, Jun. 2014.

[5] A. Bucchiarone, N. Dragoni, S. Dustdar, S. T.

Larsen, and M. Mazzara, "From monolithic to microservices: An experience report from the banking domain," *IEEE Softw.*, vol. 35, no. 3, pp. 50-55, May 2018.

[6] J. Shah and D. Dubaria, "Building modern clouds: Using docker, kubernetes & google cloud platform," *2019 IEEE 9th Annu. CCWC*, pp. 184-189, Las Vegas, USA, Jan. 2019.

[7] G. Turin, A. Borgarelli, S. Donetti, E. B. Johnsen, S. L. T. Tarifam, and F. Damiani, "A formal model of the kubernetes container framework," *Int. Symp. Leveraging Appl. of Formal Methods,* Springer, pp. 558-577, Rhodes, Greece, Oct. 2020.

[8] M. Kalske, N. Mäkitalo, and T. Mikkonen, "Challenges when moving from monolith to microservice architecture," *ICWE 2017 Current Trends in Web Eng.*, pp. 32-47, Rome, Italy, May 2017.

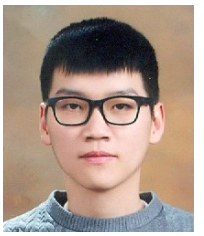
[9] D. Kim, J. Park, and K. Yeom, "Microservice construction method based on UML design assets of monolithic applications," *The J. KINGComputing*, vol. 14, no. 5, pp. 7-18, Oct. 2018.

[10] B. Mayer and R. Weinerich, "A dashboard for microservice monitoring and management," *2017 IEEE ICSAW,* pp. 66-69, Gothenburg, Sweden, Apr. 2017.

[11] E. Truyen, D. Van Landuyt, D. Preuveneers, B. Lagaisse, and W. Joosen, "A comprehensive feature comparison study of open-source container orchestration frameworks," *Applied Sci.*, vol. 931, pp. 53-76, Mar. 2019.

[12] L. A. Vayghan, M. A. Saied, M. Toeroe, and F. Khendek, "Microservice based architecture: Towards high-availability for stateful applications with kubernetes," *2019 IEEE 19th Int. Conf. Softw. Quality, Reliability and Secur. (QRS)*, pp. 176-185, Sofia, Bulgaria, Jul. 2019.

정 수 민 (Sumin Jeong)



2019년 8월 : 부산대학교 전기컴퓨터공학부 졸업

2019년 9월~현재 : 부산대학교 정보융합공학과 석사과정

<관심분야> 컴퓨터공학, 서버리스 컴퓨팅, 클라우드 컴퓨팅

[ORCID:0000-0002-2763-5570]

김 정 구 (Jeong Goo Kim)

1988년 2월 : 경북대학교 전자공학과 공학사

1991년 2월 : 경북대학교 대학원 전자공학과 공학석사

1995년 8월 : 경북대학교 대학원 전자공학과 공학박사

1995년 3월~현재 : 부산대학교 정보컴퓨터공학부 교수

1999년 3월~2001년 2월 : 미국 UCSD 방문교수

2010년 3월~2011년 2월 : 호주 UniSA 방문교수

<관심분야> 정보 및 부호이론, 디지털통신 시스템, IoT 시스템, 클라우드 컴퓨팅

염 근 혁 (Keunhyuk Yeom)

1985년 2월 : 서울대학교 계산통계학과 학사

1992년 2월 : Univ. of Florida 컴퓨터공학과 석사

1995년 2월 : Univ. of Florida 컴퓨터공학과 박사

1996년~현재 : 부산대학교 정보컴퓨터공학부 교수

<관심분야> 소프트웨어 재사용, 프로덕트 라인 공학, 소프트웨어 아키텍처, 컴포넌트 기반 소프트웨어 개발, 적응형 소프트웨어 개발, 클라우드 컴퓨팅, 빅데이터

박 준 석 (Joonseok Park)

2002년 2월 : 부산대학교 컴퓨터공학과 석사
2012년 2월 : 부산대학교 컴퓨터공학과 박사
2012~2014년 : 부산대학교 박사후 연구원
2014~현재 : 부산대학교 지능물류빅데이터연구소 연구교수

<관심분야> 클라우드 컴퓨팅, 서버리스 컴퓨팅, 프로덕트 라인 공학, 소프트웨어 정의 네트워크, 서비스 지향 아키텍처, 소프트웨어 아키텍처