https://doi.org/10.7840/kics.2025.50.9.1433

다중 클라우드의 효과적인 스케줄링을 위한 Migration 정책 기반 워크로드 재분배 설계 및 구현

윤 지 혜◆, 김 영 한°

A Design and Implementation of Migration Policy Based Workload Redistribution for Effective Scheduling of Multi-Cloud System

Jihye Yun◆, Younghan Kim°

요 약

멀티 클라우드 환경에서 비용 절감 등을 위해 퍼블릭 클라우드 사용보다 프라이빗 클라우드를 우선적으로 사용하도록 하는 등 특정 클라우드를 우선적으로 사용하게 하는 스케줄링이 요구된다. 그러나 프라이빗 클라우드 등 해당 클라우드의 사용 가능한 자원 부족 시 스케줄링이 불가능하여 운용자에 의한 별도의 워크로드 재배치 등이 진행되어야 한다. 본 논문에서는 다중 클라우드 환경에서 선호된 클라우드로의 스케줄링이 가능하도록 사전에 정의된 워크로드 이전 정책(Migration Policy)에 따라 워크로드 재분배를 수행하는 워크로드 자동 이전 제어기(Migration Controller)를 새롭게 제안하고 구현하였다. 제안된 기능을 통해 원하는 스케줄링의 성공율을 높일 수 있으며 정책에 따라 이전 재배치를 조정할 수 있다. 검증을 위해 실제 다중 클라우드 환경을 구축하고 적용 실험을 통해 기존 대비 프라이빗 클라우드 등 선호 클라우드의 활용률을 높이고 스케줄링 거절률(Rejection Rate)을 감소시킬 수 있음을 확인하였다.

키워드 : 멀티 클라우드, 비용 절감, 워크로드 재분배, 무중단
Key Words : Multi Cloud, Cost Reduction, Workload Redistribution, Zero Downtime

ABSTRACT

In order to reduce costs in a multi-cloud environment, policy-based scheduling is required to give priority to specific clouds, such as private clouds over public clouds. However, scheduling is not possible when the available resources of the cloud, such as the private cloud, are insufficient, and separate workload reallocation, etc. must be performed by the operator. In this paper, we propose and implement a workload automatic migration controller that performs workload reallocation according to a predefined workload migration policy to enable scheduling to a preferred cloud in a multi-cloud environment. We verified that the utilization rate of preferred clouds such as private clouds is increased, and the scheduling rejection rate can be reduced compared to existing ones through application experiments by building an actual multi-cloud environment with the proposed function.

※ 본 논문은 2025년도 과학기술정보통신부의 재원으로 정보통신기획평가원의 지원을 받아 수행된 연구임(No.RS-2024-00398379, (총괄4-세부2) 텔코용 고성능/고가용성 6G 크로스-클라우드 인프라 기술개발)
◆ First Author : Soongsil University Department of Electronic Engineering, jhyun@dcn.ssu.ac.kr, 학생회원
° Corresponding Author : Soongsil University Department of Electronic Engineering, younghak@ssu.ac.kr, 종신회원
논문번호 : 202411-286-C-RN, Received November 15, 2024; Revised January 9, 2025; Accepted March 14, 2025

Ⅰ. 서 론

멀티 클라우드 환경은 프라이빗 클라우드와 퍼블릭 클라우드를 비롯한 여러 클라우드를 함께 사용하는 환경을 의미한다. 멀티 클라우드 환경에서 민감성이 높은 데이터 및 워크로드는 프라이빗 클라우드에서 운용하고, 일반적인 워크로드는 퍼블릭 클라우드에 배치시킴으로써 보안과 유용성을 함께 얻을 수 있다[1,2]. 또한 비용 절감 등을 위해 퍼블릭 클라우드 사용보다 프라이빗 클라우드를 우선적으로 사용하도록 하는 등 특정 클라우드를 우선적으로 사용하게 하는 정책 기반 스케줄링이 요구된다[3,4]. 그러나 프라이빗 클라우드 등 해당 클라우드의 사용 가능한 자원 부족 시 스케줄링이 불가능하여 운용자에 의한 별도의 워크로드 재배치 등이 진행되어야 하고 이전되는 워크로드에 의해 제공되던 서비스의 연속을 위한 방법도 추가 관리해야 한다. 예를 들어 쿠버네티스의 Preemption과 같은 기능을 활용하여 기존에 실행 중이던 워크로드를 삭제하고 새로 요구된 워크로드를 배포할 수 있다. 그러나 이때 삭제된 워크로드의 서비스는 중단되게 되어 해당 서비스의 가용성을 떨어뜨린다는 문제도 생기게 된다[5]. 결과적으로 원하는 스케줄링을 위해 임의의 정책기반 하에 일부 워크로드를 재배치하는 방법이 필요하나 기존의 방법들은 이러한 과정이 관리자에 의해 이루어지고 있어 이를 자동화한 기법이 요구되고 있다.

본 논문에서는 다중 클라우드 환경에서 선호된 클라우드로의 스케줄링이 가능하도록 사전에 정의된 워크로드 이전 정책(Migration Policy)에 따라 워크로드 재분배를 수행하는 워크로드 자동 이전 제어기(Migration Controller)를 설계하고 구현하였다. 또한 선택한 워크로드를 다른 클라우드로 이전할 때 해당 워크로드에서의 서비스가 지속될 수 있도록 하는 방법을 같이 설계하였다. 그림 1에서와 같이 워크로드 재분배 기능을 타겟 클라우드로의 배치를 희망하는 워크로드를 더 배치할 수 있게 하여 종합적인 워크로드 배치 만족도를 높일 수 있게 할 것이다.

제안된 기능을 검증하기 위해 베어 메탈 서버에 오픈스택을 설치하여 구축한 프라이빗 클라우드와 퍼블릭 클라우드를 함께 활용하여 하이브리드 클라우드를 구성하고 Linux Foundation의 EMCO(Edge Multi-Cluster Orchestrator) 프로젝트를 활용하여 종합관리 플랫폼을 구축하고 추가적으로 Migration Controller와 해당 동작의 설정 정보를 위한 리소스를 추가 개발하여 구현하였다[6]. 이어 적용 실험을 통해 기존 대비 프라이빗 클라우드 등 선호 클라우드의 활용률을 높이고 스케줄링 거절률(Rejection Rate)를 감소시킬 수 있음을 확인하였다.

본 논문의 구조는 다음과 같다. 제 Ⅱ장에서는 관련 연구로서 기존 멀티 클라우드 환경에서의 워크로드 스케줄링 방법과 이전 방법에 대해 고찰하고, 제 Ⅲ장에서는 제안하는 워크로드 재분배 기능을 위한 제안 구조와 동작 절차를 설명한다. 제 Ⅳ장에서는 제안한 시스템의 구현 결과와 성능을 분석하고, 제 Ⅴ장에서 결론을 맺는다.

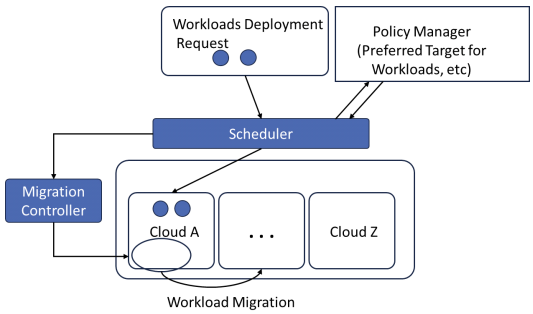

그림 1. 다중 클라우드 시스템에서 스케줄링을 위한 정책기반 워크로드 재배치
Fig. 1. Policy based Workload Redistribution for Scheduling over Multi-Cloud System

Ⅱ. 관련 연구

2.1 멀티 클라우드 환경에서의 워크로드 스케줄링 방법

멀티 클라우드 환경에서 스케줄링을 위해서는 기본적으로 워크로드의 특성에 따라 요구사항을 만족하는 클라우드를 선정하고 그중 자원 상태를 포함한 여러 메트릭을 활용하여 최적 클라우드로 선정하게 된다.

이와 관련하여 쿠버네티스 기반 클라우드와 KubeEdge와 같은 엣지 클라우드가 결합된 환경에서의 data-intensive 한 애플리케이션을 대상으로 하는 스케줄링 프레임워크가 제안되었다[7]. 제안된 스케줄러는 AI 모델 학습 등의 관련 작업을 대상으로 엣지와 클라우드 중 어디서 처리할지에 대한 스케줄링을 데이터의 특성에 따라 엣지 클라우드를 우선적으로 선택하도록 하여 엣지 리소스의 활용율을 높이고 클라우드 사용 비용을 줄이는 것을 목표로 하였다. 다만 본 제안은 엣지 클라우드의 자원이 부족한 경우는 워크로드 재배치 없이 스케줄링이 안 되도록 하여 본 논문 에서와 같은 워크로드 재배치에 의한 최적화는 다루고 있지 않다.

기존 클라우드 내에 존재하는 레이블 체계를 확장하

여 컴퓨팅 노드의 속성과 애플리케이션의 요구사항에 대해 더욱 명확하게 표현함으로써 스케줄링에 사용자의 요구사항을 더욱 반영할 수 있게 한 방법도 제안되었다[8]. 요구사항의 예로는 최소 지연시간, 재생 가능한 에너지를 사용하는 클라우드 사용, 지리적 제약 사항, 배포 비용 등이 포함될 수 있게 하였다. 구체적인 구현 구조에서 기존 쿠버네티스의 레이블링 체제를 확장하고 기존 kube-scheduler에서 요구사항의 미세 조정이 가능하도록 했다. 그러나 이러한 개별 클라우드 내의 스케줄러에 대한 확장 방법은 다중 클라우드 환경에 단순히 적용할 수 없는 한계가 있고 나아가 워크로드 재배치와 같은 스케줄링이 불가능할 경우의 해결 방법 등은 같이 고려되고 있지 않다.

Linux Foundation의 EMCO 프로젝트는 멀티 클러스터 간 애플리케이션과 서비스의 배포를 자동화하기 위한 프로젝트이다[6]. EMCO에는 클러스터와 애플리케이션의 중앙 오케스트레이션 역할을 수행하는 Management 클러스터와 실제 애플리케이션이 배포되는 워크로드 클러스터가 존재한다. Management 클러스터에서는 멀티 클라우드상에 존재하는 일부 멀티 클러스터를 논리적 집합으로 묶어 Logical Cloud를 구성하고, 그 내에서 애플리케이션 스케줄링을 수행한다. 이때 애플리케이션의 리소스 요구사항을 반영하여 해당 요구사항을 충족하는 클러스터로 스케줄링을 수행할 수 있다. EMCO에는 이러한 애플리케이션의 스케줄링을 위한 다양한 리소스가 존재하는데, 리소스의 종류에 따라 이를 관리하는 개별적인 컨트롤러가 있으며, 각 컨트롤러가 독립적으로 동작하여 필요에 따라 새로운 컨트롤러의 확장이 용이하도록 설계되었다. 그러나 현재까지 스케줄링에 정책적 요구사항을 반영하거나 스케줄링 실패 시 워크로드 재배치를 통해 특정 클러스터에 배포를 요구하는 워크로드 수용 방법 등은 제시되지 못하고 있다.

이 외에도 다양한 환경에서의 스케줄링 요구사항을 고려하여 진행된 많은 연구가 있지만, 대부분의 연구들은 워크로드 재분배 기능과 통합된 방법은 제안되고 있지 않다[9-11].

2.2 멀티 클라우드 환경에서의 워크로드 이전 방법

특정 클라우드 또는 클러스터에 문제 발생 시 실행 중인 워크로드를 이전하기 위해서는 서비스의 중단을 최소화하기 위한 전략이 필요하며, 워크로드의 특성에 따라 추가적인 백업 및 복원 과정이 요구될 수 있다.

머신러닝 워크로드를 대상으로 온프레미스 환경의 리소스가 모두 소진되었을 때 다른 퍼블릭 클라우드로 스케줄링하는 방안이 최근 제시되었다[12]. 제시된 HCS(Hybrid Cloud Scheduler)에서는 초기 워크로드의 스케줄링은 온프레미스 환경의 쿠버네티스 클러스터에서 수행되며, 해당 클러스터에서 pending 상태의 워크로드가 감지된 경우 해당 워크로드에서 필요로 하는 모든 리소스의 합을 구하여 최소 크기의 VM Spec을 계산하여 퍼블릭 클라우드에 클러스터를 동적으로 생성하고 pending 된 머신러닝 워크로드를 이전하여 작업이 수행될 수 있도록 한다. 본 연구에서는 pending 되는 워크로드의 종류를 머신러닝 워크로드로 특정하여 해결 방법을 제시하고 있으나 퍼블릭 클라우드의 인스턴스를 생성하고 머신러닝 워크로드에 특정하고 있어서 본 논문에서와 같은 모든 워크로드를 대상으로 하는 것과 워크로드 재배치와 같은 방법은 고려되지 않고 있다.

한편, 서비스의 QoS를 고려한 워크로드 이전에 관한 연구로서 워크로드의 초기 배치 이후 지속적으로 SLA(Service Level Agreement)를 모니터링한 후 SLA를 만족시키지 못할 경우 해당 워크로드를 다른 곳으로 이전하는 방법을 제안하고 있다[13]. 그러나 본 연구는 처음 스케줄링이 요구된 워크로드를 선호된 클라우드에 스케줄링이 안 될 경우 기존 워크로드를 이전하도록 하는 스케줄링과 이전 기능을 결합한 본 제안모델과는 달리 SLA에 따른 독립된 이전 기능만을 제안하고 있으며 서비스의 지속성을 위한 방법도 제시되지 못하고 있다.

이 외에도 딥러닝 모델을 사용하여 CPU 사용량, 메모리 사용량, 네트워크 트래픽 등의 메트릭 지표를 활용해 임곗값을 넘을 것으로 예측되는 Pod를 이전하게 한 연구도 있었으나 이 역시 워크로드 이전만을 다루고 있고 스케줄링과 연동된 방법으로 제안되고 있지 않다[14]. 또한 다양한 이전을 위한 알고리즘은 제안되었으나 서비스의 중단이나 스케줄링과의 연동 등은 고려되고 있지 않다[15,16].

이상과 같이 대부분의 기존 연구들은 스케줄링과 워크로드 이전이 독립된 영역으로 연구되었고 본 제안과 같이 상호 연동된 방법은 제안되고 있지 않다.

본 논문에서는 스케줄링 과정에서 문제 발생 시 사전 정의된 여러 기준을 활용하여 이전하기에 적합한 워크로드를 선택하고 서비스의 중단 없이 이를 이전하는 시스템을 제안한다. 한편 세부적인 워크로드의 이전 기술과 관련하여 Stateless 워크로드는 단순하지만 Stateful 워크로드의 경우 기존 사용되던 최신 정보의 백업과 복원 기술이 요구되나 해당 요소 기술들은 기존 연구 결과들을 적용하여 해결하였다[17-22]. 이상의 제안된 상세 구조 및 기능을 다음 장에서 고찰한다.

Ⅲ. 멀티 클라우드 환경에서의 워크로드 재분배 기능 설계

3.1 제안 구조

본 논문에서 제안하는 멀티 클라우드 환경 및 워크로드 재분배 기능 적용 구조는 그림 2와 같다. 기본적으로 쿠버네티스 기반 클라우드 환경을 가정하여 쿠버네티스의 기본 단위인 클러스터를 대상으로 워크로드 클러스터와 배포할 워크로드에 대한 중앙 관리를 수행하는 Management 클러스터가 존재하며, 워크로드 클러스터에 실제 워크로드들이 배포된다. Management 클러스터의 Cluster Registration Controller는 Cloud Provider와 클러스터를 등록하는 역할을 수행하는데, 이때 퍼블릭 클라우드보다 프라이빗 클라우드를 먼저 사용할 수 있도록 하기 위해 클라우드별 선호도를 함께 저장한다. Scheduler에서는 배포할 워크로드를 등록하고 클라우드별 선호도를 고려한 스케줄링을 수행한다. 본 연구에서 제안한 Migration Controller는 스케줄링 과정에서 배포를 수행해야 하는 클러스터에 리소스가 부족하여 워크로드 재분배가 필요하다고 판단된 경우 호출되며, 사전에 정의된 Migration Policy와 워크로드 클러스터의 Metrics-server로부터 수집한 상황정보를 활용하여 이전할 워크로드를 선택하고 선택된 워크로드의 이전을 수행한다. External Storage는 Stateful 워크로드를 위한 데이터가 저장되는 곳으로, Stateful 워크로드가 다른 클러스터로 이전되더라도 기존 상태 정보가 저장된 본 Storage를 통해 지속적인 서비스가 가능하도록 하였다.

본 논문에서는 워크로드 재분배를 수행할 때 다양한

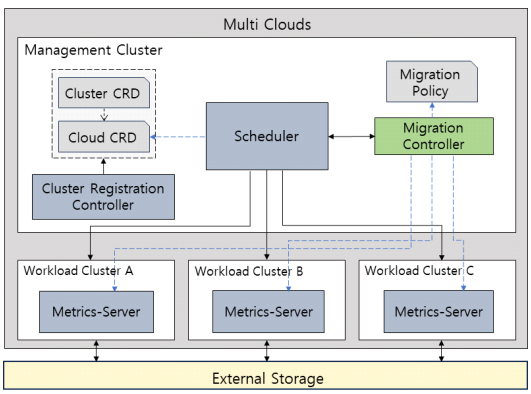

그림 2. 멀티 클라우드 환경에서의 워크로드 재분배를 위한 시스템 구조
Fig. 2. System architecture for workload redistribution in multi-cloud environment

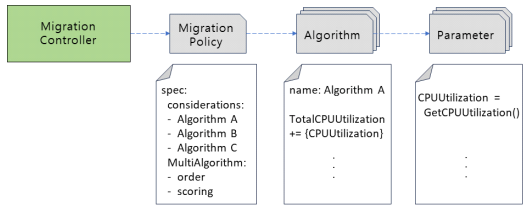

그림 3. Migration Controller 내부 리소스
Fig. 3. Internal resources of migration controller

요구사항을 반영할 수 있도록 Migration Controller 내의 세부 자원을 그림 3과 같이 설계하였다. Migration Policy에서는 워크로드의 리소스 사용량 기반 워크로드 선택 또는 마이크로 서비스 간 연결성 기반 워크로드 선택과 같이 사용자의 요구사항에 따라 다양한 알고리즘이 선택될 수 있으며, 이러한 알고리즘은 계산식에 따라 다양한 입력 데이터를 필요로 한다. 이러한 요구사항은 필요에 따라 새로운 policy로 시스템에 적용할 수 있는 구조를 설계하였고 또한 이에 연계된 알고리즘을 변경 또는 신규 적용 가능할 수 있게 Algorithm 리소스를 추가하였다. 또한 Algorithm 리소스에서 사용하는 입력 데이터를 생성하는 Parameter 리소스를 분리하여 설계하였다. 그리고 여러 알고리즘에서 동일한 입력 데이터를 사용하는 경우 Parameter 리소스를 재활용하여 계산에 활용할 수 있도록 하였다. 이를 통해 Algorithm과 Parameter를 사전에 정의해 놓기만 하면 Migration Policy에서는 적용할 알고리즘의 선택 등을 정책 데이터로 입력을 통해 상황에 따라 알고리즘을 변경하여 적용할 수 있도록 하였다. 이때 여러 알고리즘을 동시에 적용하는 경우에는 우선순위에 따라 순차적으로 알고리즘을 반영하는 order 방식과 알고리즘별 점수를 계산하여 최종적으로 가장 높은 점수를 받은 워크로드가 선택되도록 하는 scoring 방식 중 선택이 가능하도록 하였다.

3.2 서비스 중단 없는 워크로드 이전 구조

일반 워크로드는 이전되더라도 문제없으나 Stateful 워크로드의 경우 별도로 연결되었던 volume의 관리가 같이 요구된다. 기본적으로 쿠버네티스에서는 Stateful 워크로드의 이름과 생성 순서를 보장하기 위해 StatefulSet을 활용할 수 있으며, 이전한 후에도 스토리지와의 연결성을 유지하기 위해 External Storage를 사용할 수 있다. 그리고 그림 4와 같이 StatefulSet의 volumeClaimTemplates를 사용하여 각 Pod마다 PVC를 생성하도록 하고 PV를 해당 PVC와 연결하면, Pod의 생성 순서에 따라 원하는 PV에 연결되도록 할 수

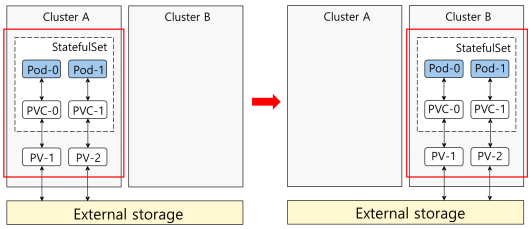

그림 4. Stateful 워크로드의 데이터 연결성을 유지하여 이전하는 방법
Fig. 4. Migration strategies while maintaining data connectivity for stateful workloads

있다. 이를 통해 워크로드를 이전한 경우에도 기존 데이터에 동일하게 접근할 수 있게 된다.

이 외에 선택한 워크로드를 서비스 중단 없이 이전하기 위한 방법으로는 LoadBalancer를 활용하였다. 그림 5에서와 같이 LoadBalancer의 IP 주소를 사용하여 서비스를 등록하였다. 외부에서 해당 서비스를 액세스할 경우 LoadBalancer로 접근하게 되고 기존 워크로드가 운용되던 곳에서 새로 이전된 곳으로 LoadBalancer에 의해 자동 연결되게 하여 서비스 연속성을 해결하게 된다. 즉 워크로드의 선택이 완료되면 해당 워크로드를 다른 클러스터로 이전하여 준비될 때까지 기다린 후 LoadBalancer에서 해당 멤버로 추가한다. 이후 기존 워크로드를 삭제하고 나면 신규 이전된 워크로드로 서비스 중단 없는 연결이 제공되게 된다.

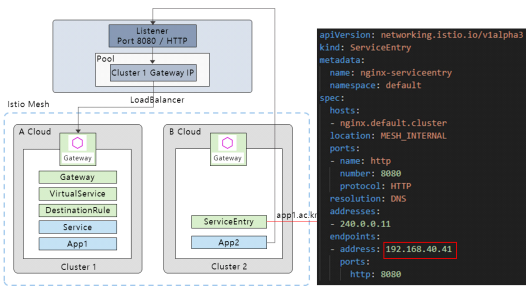

그림 5. LoadBalancer를 활용한 서비스 등록
Fig. 5. Service registration using LoadBalancer

Ⅳ. 구현 및 성능 분석

4.1 EMCO 내의 구현 구조

본 연구에서는 제안한 시스템의 구현을 위해 Linux Foundation의 EMCO 프로젝트를 활용하였다. 그림 6은 멀티 클러스터 오케스트레이션을 위한 EMCO 프로젝트 내에 구현한 구조로서 기존 대표적인 컴포넌트들과 각 컴포넌트가 관리하는 리소스 및 추가된 기능 등을

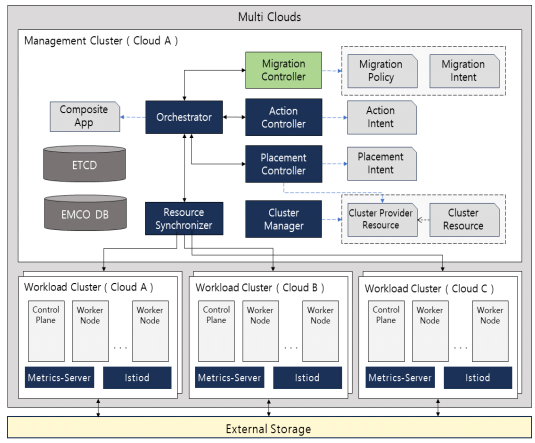

그림 6. EMCO내 구현 구조
Fig. 6. Implementation Architecture on EMCO

표현한 것이다. Cluster Manager가 관리하는 리소스에는 Cluster Provider와 Cluster가 있으며, 쿠버네티스 클러스터의 kubeconfig 파일을 이용하여 워크로드 클러스터를 등록한다. Cluster Provider는 클러스터가 어느 벤더로부터 제공되었는지 논리적으로 구분하는 역할을 한다. Orchestrator는 배포할 워크로드를 애플리케이션 단위의 패키지 형태로 등록하고, 스케줄링 과정에서 필요에 따라 다른 컨트롤러를 호출하며 전체 스케줄링 프로세스를 관리한다. Placement Controller는 Placement Intent를 통해 각 워크로드의 요구사항을 표현하며, 이를 만족하는 클러스터로 스케줄링을 수행한다. 그리고 Action Controller는 Action Intent를 활용하여 스케줄링이 완료된 이후 함께 배포할 새로운 리소스를 추가하거나 워크로드에 customization을 수행할 수 있게 한다. 이 모든 과정이 완료되면 Resource Synchronizer를 통해 워크로드 클러스터로 실제 배포가 이루어진다.

이외 Migration Controller를 추가로 구현하고, Migration Intent를 활용하여 각 워크로드의 이전 가능 여부와 우선순위를 표현하고, Migration Policy를 통해 다양한 알고리즘을 활용하여 워크로드를 선택하도록 하였다. 그리고 스케줄링 시 프라이빗 클라우드를 우선

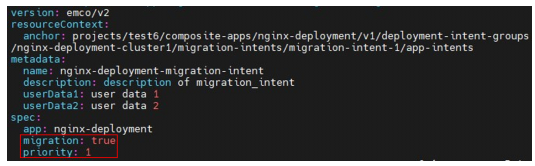

그림 7. 클라우드 별 선호도 표현을 위해 확장한 Cluster Provider 리소스 예시
Fig. 7. Example of an extended Cluster Provider resource for expressing cloud-specific preferences

적으로 사용할 수 있도록 하기 위해 그림 7과 같이 기존 Cluster Provider 리소스의 .spec.priority 필드를 확장하여 클라우드별 선호도를 표현하고, Placement Controller에서 이를 고려하여 스케줄링을 수행하도록 하였다.

4.2 Migration Controller 동작 예시

본 논문에서는 Migration Policy가 따로 적용되지 않은 경우 그림 7의 선호도 정보를 활용하여 이전할 워크로드를 선택하도록 구현하였다. Migration Intent의 spec.migration 필드가 .spec.app 필드에 해당하는 워크로드의 이전 가능 여부를 의미하며, true일 경우 .spec.priority 필드를 통해 우선순위를 표현한다. 그리고 Migration Policy에서도 여러 알고리즘을 사용하고자 할 때에는 그림 8과 같이 우선순위를 표현하여 우선순위에 따라 알고리즘을 순차적으로 적용하여 선택하도록 하였다.

이를 실험하기 위해 두 가지 알고리즘을 사용하였다. 첫 번째 로 서비스의 부하가 덜한 워크로드를 선택하기 위해 CPU Utilization을 계산하여 값이 낮은 워크로드를 선택하는 CPUUtilization이라는 이름의 알고리즘을 사용하였고, 두 번째로 워크로드를 이전한 후 기존 대비 마이크로 서비스 간 통신 지연 시간 증가를 최소화하기

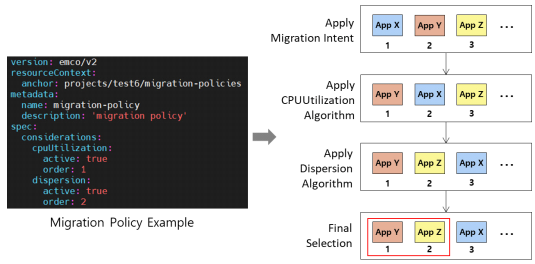

그림 8. Migration Policy 예시
Fig. 8. Example of Migration Policy

위해 하나의 애플리케이션을 이루는 마이크로 서비스의 분산 정도를 계산하여 여러 클러스터에 분산된 형태로 배포되어 있는 워크로드를 선택하기 위한 Dispersion 알고리즘을 사용하였다.

Migration Controller를 통한 워크로드 이전 상세 절차는 그림 9와 같다.

그림 9에서 Application 1을 Cluster X에 배포해야 하는 상황에서 Cluster X에 리소스가 부족한 경우에 이루어지는 절차에 대해 표현하였으며, Migration Controller를 호출하기 이전에 Application 1을 Cluster X에 스케줄링하기까지의 과정은 생략하였다.

Application 1의 배포 요청은 Resource Synchronizer로 보내지며, Application 1은 Cluster X

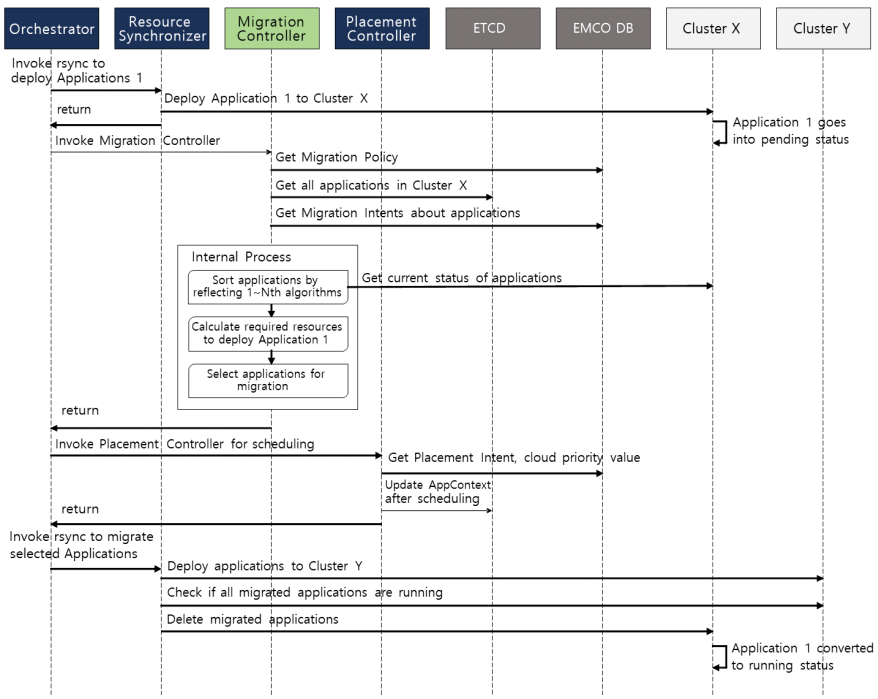

그림 9. 워크로드 이전 절차
Fig. 9. Workload migration procedure

에서 pending 상태로 기다리게 된다. 그리고 Orchestrator에 의해 Migration Controller가 호출된다. Migration Controller는 Migration Policy 정보를 확인하고, Cluster X에서 실행 중인 모든 애플리케이션 데이터를 불러온다. 그리고 해당 애플리케이션들의 Migration Intent를 불러와 .spec.migration이 true인 것들만 선별하고, 이를 .spec.priority 값에 따라 정렬한다. 이후 Migration Policy를 통해 적용된 알고리즘을 순차적으로 확인하여 정렬을 수행하며, 앞 단계의 정렬 결과 동일한 값을 가지는 애플리케이션들이 다음 알고리즘을 통해 다시 정렬된다. 모든 알고리즘에 대한 정렬이 완료되면, Application 1을 배포하기 위해 필요한 리소스를 계산하여 해당 리소스를 만들어줄 수 있을 만큼 이전할 워크로드를 선택한다. Migration Controller의 작업이 완료되면 이전되는 애플리케이션의 스케줄링을 위해 Placement Controller가 호출되고, 클라우드의 선호도를 고려하여 Cluster X를 제외한 임의의 Cluster Y로 스케줄링이 수행된다. 이전하는 과정은 Resource Synchronizer에 의해 이루어지며, Resource Synchronizer는 애플리케이션 의 모든 Pod가 running 상태가 될 때까지 기다렸다가 기존 워크로드를 삭제한다. 이를 통해 Cluster X에는 Application 1이 실행되기 위해 필요한 충분한 리소스가 확보되며, pending 상태에 있던 Application 1이 running 상태로 전환된다.

4.3 실험 및 성능 분석

실험을 위해 온프레미스 클라우드와 AWS 간 site to site VPN을 통해 그림 10과 같이 실험 환경을 구축하였다.

온프레미스 클라우드에 Management 클러스터를 구축하였으며, 워크로드 클러스터는 마스터 노드 1개와 워커 노드 1개로 구축하여 온프레미스 클라우드에서 2개, AWS 클라우드상에서 1개를 각각 등록하였다.

그리고 스케줄링에 대한 워크로드의 요구사항을 직

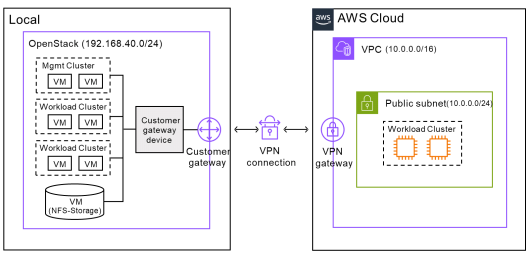

그림 10. 온프레미스 클라우드와 AWS를 활용한 실험 환경
Fig. 10. Experiment environment using on-premise cloud and AWS

표 1. 워크로드 클러스터 사양 및 레이블
Table 1. Workload cluster specifications and labels

Cluster Location	Cluster Name	Cluster Specification	Applied Label
On-premise	Onprem- A	Master Node 1 (2 vCPUs, 4GiB Memory) Worker Node 2 (8 vCPUs, 6GiB Memory)	ABC, AC, AB, A
	Onprem-B	Master Node 1 (2 vCPUs, 4GiB Memory) Worker Node 2 (8 vCPUs, 6GiB Memory)	ABC, AB, BC, B
AWS	AWS-C	Master Node 1 (2 vCPUs, 4GiB Memory) Worker Node 2 (8 vCPUs, 6GiB Memory)	ABC, AC, BC, C

관적으로 표현하기 위해 레이블을 활용하였다. 모든 경우의 수를 수용할 수 있도록 하기 위해 표 1과 같이 각 클러스터에 자기 자신이 포함된 모든 레이블을 부여하였다. 여기서 레이블의 각 문자는 Cluster Name의 마지막 문자를 의미한다. 예를 들어 ABC의 레이블을 가지는 클러스터에 배포될 수 있는 애플리케이션은 모든 클러스터에 배포될 수 있음을 의미한다. Stateful 워크로드의 데이터 저장을 위한 External Storage는 온프레미스 클라우드의 VM 1개에 NFS 서버를 통해 구성하였다.

첫 번째 실험에서는 StatefulSet과 PV를 통해 워크로드를 이전한 후에도 Pod의 순서와 스토리지 간 연결성이 보장될 수 있음을 보였다. replicas를 2로 설정하여 각각의 Pod가 고유의 PV에 바인딩 되도록 설정하였고, NFS 서버의 공유 디렉토리를 마운트하여 pod-0과 pod-1이 각각의 저장소와 일관된 연결을 유지하는지 확인하였다. 이를 위해 두 개의 PV(pv1, pv2)를 NFS 서버의 각기 다른 공유 디렉터리(/mnt/shared1, /mnt/shared2)와 연결하였다. 공유 디렉터리에는 웹 페이지 파일인 index.html을 위치시키고, shared1과 shared2를 구분할 수 있도록 index.html을 수정하였다. PVC는 StatefulSet의 volumeClaimTemplates를 통해 생성하고, 생성된 PVC가 사전에 정의된 PV와 바인딩 되도록 하였다. 그리고 Statefulset과 PV를 다른 클러스터로 이전하여 Pod와 스토리지의 연결성을 확인하였다.

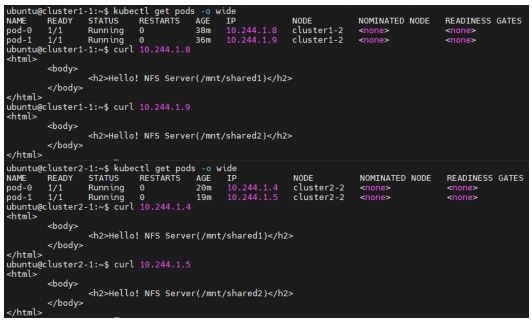

그림 11. Stateful 워크로드를 이전하기 전과 후의 Pod와 스토리지 연결성
Fig. 11. Connectivity between Pods and storage before and after migrating stateful workloads

실험 결과 그림 11에서 워크로드를 이전하기 전과 후에 모두 pod-0은 pv1과 마운트 되었으며, pod-1은 pv2와 마운트 되었다. 이를 통해 Stateful 워크로드를 이전한 후에도 Pod의 순서 및 스토리지와의 연결성이 변함없이 유지됨을 확인할 수 있다.

이후 실험에서는 본 논문에서 제안한 Migration 기능과 Cloud Preference를 적용한 스케줄링 방법의 성능을 분석하기 위한 실험을 진행하였다. 실험을 위해 표 2와 같이 1~3개의 마이크로 서비스로 이루어진 15개의 애플리케이션을 사용하였으며, 각 마이크로 서비스에 랜덤하게 CPU와 메모리 Request를 부여하였다.

본 실험에서는 Migration과 Cloud Preference를 적용한 경우와 그렇지 않은 경우의 워크로드 배포 수에 따른 프라이빗 클러스터와 퍼블릭 클러스터의 리소스 점유율을 비교하였다. 배포하는 워크로드의 수를 5개씩 증가시키면서 클러스터 별 리소스 점유율을 측정하였으며, 워크로드의 스케줄링 요구사항을 임의로 변경하면서 워크로드의 배포를 5회 반복한 후 리소스 점유율의 평균값을 계산하였다.

실험 결과 그림 12, 13과 같이 Migration과 Cloud Preference를 적용하지 않은 경우 5개, 10개, 15개를 배포한 모든 경우에 프라이빗과 퍼블릭에서 비슷한 리

표 2. 배포할 애플리케이션 종류
Table 2. Type of application to be distributed

Number of microservices	Count	Resource Requirements	Scheduling Requirements
1	10	(1 to 5) cores, (2000 ~ 7000) MiB memory request per microservice	Scheduling on clusters with specific labels
2	3		
3	2		

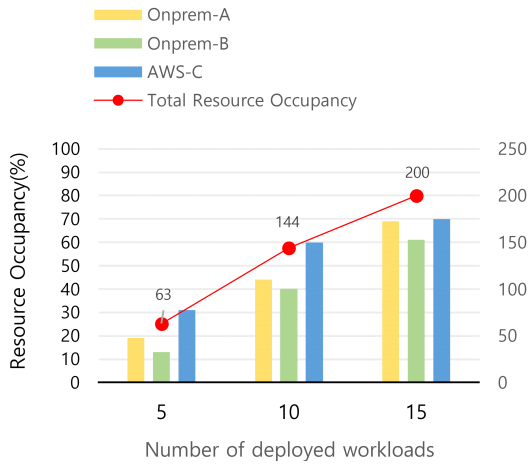

그림 12. Migration과 Cloud Preference를 적용하지 않은 경우
Fig. 12. In case Migration and Cloud Preference are not applied

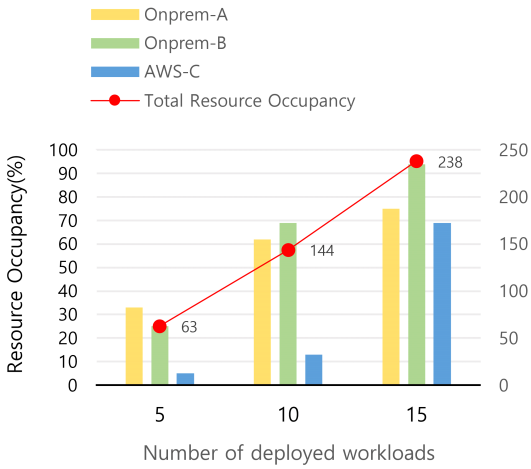

그림 13. Migration과 Cloud Preference를 적용한 경우
Fig. 13. In case Migration and Cloud Preference are applied

소스 점유율을 보였지만, Migration과 Cloud Preference를 적용한 경우 5개, 10개의 워크로드를 배포하는 동안은 프라이빗 클러스터의 리소스 점유율이 퍼블릭 클러스터의 리소스 점유율보다 월등히 높음을 알 수 있다. 이후 15개를 배포했을 때 퍼블릭 클러스터의 리소스 점유율이 급격히 올라간 것은 프라이빗 클러스터의 리소스가 소진되었음을 의미한다. 이를 통해 Migration과 Cloud Preference를 적용한 경우 프라이빗 클라우드의 활용률을 높여 불필요한 퍼블릭 클라우드의 사용을 줄일 수 있음을 확인할 수 있다. 또한, 종합 리소스 점유량을 비교했을 때 5개와 10개를 배포한 상

황에서는 두 가지 경우에 모두 동일한 종합 리소스 점유량을 보이지만, 15개를 배포했을 때는 Migration과 Cloud Preference를 적용한 경우에 리소스 점유량이 더 높음을 확인하였다. 이는 Migration을 수행하지 않은 경우에 클러스터의 리소스 부족으로 인해 배포되지 못한 워크로드가 있음을 의미하며, Migration을 수행한 경우에는 워크로드 재분배를 통해 더 많은 워크로드가 배포된 것이다.

그림 14에서는 Migration을 수행했을 때 어느 정도로 Rejection Rate가 감소하는지를 알아보기 위해 Cloud Preference를 적용한 상태에서 Migration의 수행 여부에 따른 배포 요청의 Rejection Rate를 비교하였다. 이를 위해 워크로드의 스케줄링 요구사항을 임의로 변경하면서 전체 워크로드의 배포를 5회 반복한 후 Rejection Rate의 평균값을 측정하였다. 실험 결과 Migration을 수행한 경우 기존 대비 약 50%가량

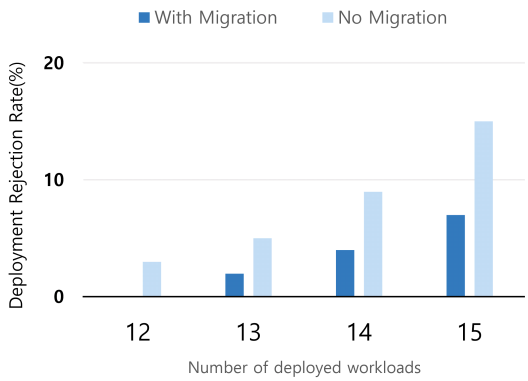

그림 14. Migration을 수행한 경우와 수행하지 않은 경우의 Deployment Rejection Rate
Fig. 14. Deployment Rejection Rate with and without Migration

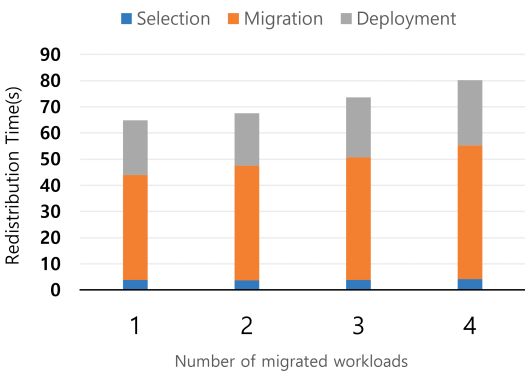

그림 15. 이전되는 워크로드 수에 따른 재분배 수행 시간
Fig. 15. Redistribution execution time based on the number of migrated workloads

Rejection Rate를 줄일 수 있음을 확인하였다.

마지막 실험에서는 이전되는 워크로드 수에 따른 재분배 수행 시간을 분석하였다. 이전되는 워크로드를 실행하기 위해 필요한 컨테이너 이미지는 모두 다운로드 되어 있지 않은 상태에서 수행하였으며, 배포하는 워크로드의 리소스 요구사항을 증가시키면서 이전되는 워크로드의 수가 증가하도록 실험하여 이전되는 워크로드 수에 따른 재분배 및 각 단계의 소요 시간을 측정하였다.

실험 결과 그림 15에서와 같이 워크로드를 이전하는 과정에서 이미지 다운로드로 인해 이전하는 워크로드 수가 증가할수록 시간이 조금씩 늘어남을 확인하였다.

V. 결 론

본 논문에서는 멀티 클라우드 환경에서 비용 절감 등을 위해 퍼블릭 클라우드 사용보다 프라이빗 클라우드를 우선적으로 사용하도록 하는 등 특정 클라우드를 우선적으로 사용하게 하는 정책 기반 스케줄링과 해당 클라우드의 사용 가능한 자원 부족 시 자동 워크로드 재배치를 통해 요구된 워크로드를 배포할 수 있는 기능을 통합한 오케스트레이션 기능을 설계하고 구현하였다.

설계된 시스템은 베어 메탈 서버에 오픈스택을 설치하여 구축한 프라이빗 클라우드와 퍼블릭 클라우드를 함께 활용하여 하이브리드 클라우드를 구성하고 Linux Foundation의 EMCO(Edge Multi-Cluster Orchestrator) 기반 위에 Migration Controller를 추가하고 기존 컴포넌트 및 리소스 일부를 확장하여 구현하였다. 특히 이전되는 워크로드의 서비스 유지를 위한 구조를 같이 구현하였고 운용실험을 통해 스케줄링 rejection rate를 줄이면서 원하는 클라우드의 활용성을 높이게 함을 비교 검증하였다. 이러한 다중 클라우드환경에서 워크로드 재배치 기능은 최종적인 워크로드 배치 성공율을 높일 수 있어 기존 시스템보다 나은 스케줄링 성공율을 얻게 하였다. 또한 제안된 시스템은 실제 다중 클라우드 환경에서 사용자가 원하는 다양한 정책적 클라우드 선택 요구사항을 간단히 추가 배포할 수 있게 설계되어 다양한 워크로드 배포 정책을 실현할 수 있게 할 수 있을 것이다.

References

[1] J. Lei, Q. Wu, and J. Xu, “Privacy and security-aware workflow scheduling in a

hybrid cloud," *Future Generation Computer Syst.*, vol. 131, pp. 269-278, Jun. 2022. (https://doi.org/10.1016/j.future.2022.01.018)

[2] Z. Sun, H. Huang, Z. Li, C. Gu, R. Xie, and B. Qian, "Efficient, economical and energy-saving multi-workflow scheduling in hybrid cloud," *Expert Syst. with Appl.*, vol. 228, no. 120401, Oct. 2023. (https://doi.org/10.1016/j.eswa.2023.120401)

[3] B. Wang, C. Wang, Y. Song, et al., "A survey and taxonomy on workload scheduling and resource provisioning in hybrid clouds," *Cluster Comput*, vol. 23, pp. 2809-2834, Feb. 2020. (https://doi.org/10.1007/s10586-020-03048-8)

[4] L. Cheng, A. Kalapgar, A. Jain, et al., "Cost-aware real-time job scheduling for hybrid cloud using deep reinforcement learning," *Neural Comput. and Appl.*, vol. 34, pp. 18579-18593, Jun. 2022. (https://doi.org/10.1007/s00521-022-07477-x)

[5] Kubernetes-main document, *Pod priority preemption*, 2024, from https://kubernetes.io/docs/concepts/scheduling-eviction/pod-priority-preemption/

[6] EMCO-main document, 2024, from https://project-emco.io/

[7] J. Luo, B. Tang, and J. Zhang, "Container scheduling in hybrid cloud-edge collaborative system," *GLOBECOM 2022*, pp. 5662-5667, Rio de Janeiro, Brazil, Dec. 2022. (https://doi.org/10.1109/GLOBECOM48099.2022.10001714)

[8] L. F. Altran, G. Galante, and M. S. Oyamada, "Label-affinity-scheduler: Considering business requirements in container scheduling for multi-cloud and multi-tenant environments," *2022 XII Brazilian Symp. Comput. Syst. Eng. (SBESC)*, pp. 1-8, Fortaleza/CE, Brazil, Nov. 2022. (https://doi.org/10.1109/SBESC56799.2022.9964784)

[9] J. P. M. Vilaça, "Orchestration and distribution of services in hybrid cloud/edge environments," M.S. Thesis, Universidade do Minho, 2022.

[10] R. Bi, T. Peng, J. Ren, X. Fang, and G. Tan, "Joint service placement and computation scheduling in edge clouds," *2022 IEEE ICWS*, pp. 47-56, Barcelona, Spain, Jul. 2022. (https://doi.org/10.1109/ICWS55610.2022.00022)

[11] S. Long, W. Wen, Z. Li, K. Li, R. Yu, and J. Zhu, "A global cost-aware container scheduling strategy in cloud data centers," in *IEEE Trans. Parall. and Distrib. Syst.*, vol. 33, no. 11, pp. 2752-2766, Nov. 2022. (https://doi.org/10.1109/TPDS.2021.3133868)

[12] J. Kieley, "A hybrid cloud kubernetes scheduler for machine learning workloads," M.S. Thesis, Arizona State University, ProQuest Dissertations Publishing, Aug. 2021.

[13] K. Kaur, F. Guillemin, and F. Sailhan, "Live migration of containerized microservices between remote Kubernetes clusters," *IEEE INFOCOM 2023*, pp. 1-6, Hoboken, NJ, USA, May 2023. (https://doi.org/10.1109/INFOCOMWKSHPS57453.2023.10225858)

[14] T. Heo, J.-H. An, and Y. Kim, "Design and implementation of migration manager between cloud edge platforms," in *Proc. Int. Conf. RACS '20*, pp. 142-145, New York, NY, USA, Oct. 2020. (https://doi.org/10.1145/3400286.3418279)

[15] S. A. Khan, M. Abdullah, W. Iqbal, M. A. Butt, F. Bukhari, and S.-U. Hassan, "Automatic migration-enabled dynamic resource management for containerized workload," in *IEEE Syst. J.*, vol. 17, no. 2, pp. 2378-2389, Jun. 2023. (https://doi.org/10.1109/JSYST.2022.3204748)

[16] K. Kaur, F. Guillemin, and F. Sailhan, "A microservice migration approach to controlling latency in 5G/6G networks," *IEEE ICC 2023*, pp. 4912-4917, Rome, Italy, May-Jun. 2023. (https://doi.org/10.1109/ICC45041.2023.10279178)

[17] J. Kim and E. Jeong, "Restore schedule operator for automating cluster to cluster

recovery in Kubernetes," *Summer Annual Conf. IEIE,* pp. 2329-2332, Jeju Island, Korea, Jun. 2023.

[18] Velero-main document, 2024, from https://velero.io/

[19] Kubernetes-main document, *Volume-snapshots*, 2024, from https://kubernetes.io/docs/concepts/storage/volume-snapshots/

[20] M. Gundall, J. Stegmann, M. Reichardt, and H. D. Schotten, "Downtime optimized live migration of industrial real-time control services," *2022 IEEE 31st ISIE*, pp. 253-260, Anchorage, AK, USA, Jun. 2022. (https://doi.org/10.1109/ISIE51582.2022.9831601)

[21] K. Govindaraj and A. Artemenko, "Container live migration for latency critical industrial applications on edge computing," *2018 IEEE 23rd Int. Conf. ETFA*, pp. 83-90, Turin, Italy, Sep. 2018. (https://doi.org/10.1109/ETFA.2018.8502659)

[22] P. Souza Jr., D. Miorandi, and G. Pierre, "Good shepherds care for their cattle: Seamless pod migration in geo-distributed Kubernetes," *2022 IEEE 6th ICFEC*, pp. 26-33, Messina, Italy, May 2022. (https://doi.org/10.1109/ICFEC54809.2022.00011)

윤 지 혜 (Jihye Yun)

2022년 2월 : 안양대학교 정보전기전자공학과 학사

2024년 8월 : 숭실대학교 정보통신공학과 석사

<관심분야> 멀티 클라우드 오케스트레이션, 스케줄링, 쿠버네티스, CI/CD

김 영 한 (Younghan Kim)

49권 9호, pp. 1306-1314, 9월 2024 참조