KIPS Trans. Comp. and Comm. Sys.
Vol.11, No.3 pp.73~82
ISSN: 2287-5891 (Print), ISSN 2734-049X (Online)

https://doi.org/10.3745/KTCCS.2022.11.3.73

An Improvement of Kubernetes Auto-Scaling Based on Multivariate Time Series Analysis

Yong Hae Kim† · Young Han Kim††

ABSTRACT

Auto-scaling is one of the most important functions for cloud computing technology. Even if the number of users or service requests is explosively increased or decreased, system resources and service instances can be appropriately expanded or reduced to provide services suitable for the situation and it can improves stability and cost-effectiveness. However, since the policy is performed based on a single metric data at the time of monitoring a specific system resource, there is a problem that the service is already affected or the service instance that is actually needed cannot be managed in detail. To solve this problem, in this paper, we propose a method to predict system resource and service response time using a multivariate time series analysis model and establish an auto-scaling policy based on this. To verify this, implement it as a custom scheduler in the Kubernetes environment and compare it with the Kubernetes default auto-scaling method through experiments. The proposed method utilizes predictive data based on the impact between system resources and response time to preemptively execute auto-scaling for expected situations, thereby securing system stability and providing as much as necessary within the scope of not degrading service quality. It shows results that allow you to manage instances in detail.

Keywords : Multivariate Time Series, VAR, Kubernetes, Auto-Scaling

다변량 시계열 분석에 기반한 쿠버네티스 오토-스케일링 개선

김 용 회† · 김 영 한††

요 약

오토-스케일링은 클라우드 컴퓨팅 기술이 ICT 핵심 기반 기술로 자리 잡을 수 있는 가장 중요한 기능 중 하나로써 사용자나 서비스 요청의 폭발적인 증가 또는 감소에도 시스템 자원과 서비스 인스턴스를 적절하게 확장 또는 축소하여 상황에 맞는 서비스의 안정성과 비용 대비 효과를 향상하는 기술이다. 하지만 특정 시스템 자원에 대한 모니터링 시점의 단일 메트릭 데이터를 기반으로 정책이 수립·실행되다 보니 이미 서비스에 영향이 있거나 실제 필요한 서비스 인스턴스를 세밀하게 관리하지 못하는 문제점이 있다. 이러한 문제점을 해결하기 위해서 본 논문에서는 시스템 자원과 서비스 응답시간을 다변량 시계열 분석 모델을 사용하여 분석·예측하고 이를 기반으로 오토-스케일링 정책을 수립하는 방안을 제안한다. 이를 검증하기 위해 쿠버네티스 환경에서 커스텀 스케쥴러를 구현하고, 실험을 통해 쿠버네티스 기본 오토-스케일링 방식과 비교 분석한다. 제안하는 기법은 시스템 자원과 응답시간 사이의 영향에 기반한 예측 데이터를 활용하여 예상되는 상황에 대한 오토-스케일링을 선제적으로 실행함으로써 시스템의 안정성을 확보하고 서비스 품질이 저하되지 않는 범위내에서 필요한 만큼의 인스턴스를 세밀하게 관리할 수 있는 결과를 보인다.

키워드 : 다변량 시계열 분석, VAR, 쿠버네티스, 오토-스케일링

1. 서 론

클라우드 컴퓨팅 기술은 초연결, 초지능화로 대표되는 4차 산업혁명을 뒷받침하는 ICT 서비스 인프라의 핵심 기반 기술로 자리잡고 있으며, 최근에는 시스템 환경에 대한 의존성이 없고, 경량화를 통한 속도 및 이식성에 강점이 있는 컨테이너 기반의 클라우드 컴퓨팅 기술이 고수준의 프로세스 분리 기능 등에서 상대적인 단점을 가짐에도 불구하고 하이퍼바이저 기반의 서버 가상화 기술을 빠르게 대체하고 있다[1].

ICT 기술의 발전과 비즈니스 모델 및 서비스 모델의 변화에 따라 서비스 규모와 범위가 커지고 있으며 글로벌화 되는 추세를 보인다. 클라우드 컴퓨팅 기술은 이러한 서비스 유형과 규모, 적용 범위에 따라 탄력적으로 서비스 구성하고 운영할 수 있는 기술적 특징을 가지고 있으며 이를 위한 가장 핵심적인 기술 중 하나가 오토-스케일링 기술이다[2]. 오토-스케일링은 사용자나 서비스 요청의 폭발적인 증가와 감소에도 시스템 자원과 서비스 인스턴스를 적절하게 확장하거나 축소할 수 있는 기능으로 시스템을 현재 서비스 상황에 맞게 자동으로 구성함으로써 서비스의 안정성을 확보하고 비용 대비

† 정 회 원 : 숭실대학교 IT융합학과 박사과정
†† 종신회원 : 숭실대학교 전자정보공학부 교수
Manuscript Received : July 28, 2021
First Revision : September 24, 2021
Accepted : October 20, 2021
* Corresponding Author : Young Han Kim(younghak@ssu.ac.kr)

효과를 극대화할 수 있다[3].

이와 관련한 기존 연구들은 특정 시스템 자원의 현재 상태를 기반으로 스케일링하는 모델을 제안하거나[4], 사용자 정의 메트릭을 수집하여 컨테이너 서비스 특성에 맞는 스케일링을 수행하는 방법[5], 자원 사용량이나 서비스 요청 횟수를 시계열 분석 모델 또는 머신러닝 기술에 기반한 예측 데이터를 활용하여 스케일링을 효과적으로 수행할 수 있도록 하는 다양한 연구들이 진행되었다[6-8]. 하지만 기존 연구들은 특정 자원의 현재 상태나 제한적인 예측 정보를 이용한 오토-스케일링 방안에 초점에 맞춰져 있다 보니 다양한 요소를 고려해야 하는 현업상황에 적용하기 어려운 부분이 있었다. 또한 현재 시스템의 상태와 제한적 예측 데이터가 실제 서비스의 품질에 어떠한 영향을 미치는지에 대한 분석과 판단은 포함되어 있지 않기 세부적인 스케일링 정책을 적용하는 것에 한계가 있다.

본 논문에서는 모니터링되는 시스템 자원 정보와 서비스 지표를 결합한 다변량 시계열 분석을 통한 예측을 통해 가까운 미래시점에 예상되는 시스템의 상태와 서비스 상황에 맞춰 사전 예방적인 선제적 스케일링을 수행하여 시스템을 안정적으로 유지함과 동시에 서비스 품질에 대한 영향을 최소화하는 방안을 제안하고 제안하는 방안이 효과적인 오토-스케일링을 수행하는지 확인하고자 쿠버네티스(Kubernetes) 환경에서 구현하고 임의의 워크로드를 발생시켜 오토-스케일링 알고리즘의 성능을 실험하고 분석한다.

본 논문의 구성은 2장에서 컨테이너 오케스트레이션과 오토-스케일링 그리고 다변량 시계열 분석에 대한 연구를 살펴보고 3장에서는 다변량 시계열 분석에 기반한 오토-스케일링 방안을 제안하고 구현 방법을 설명한다. 4장에서는 제안하는 방안의 성능을 평가하기 위한 테스트를 진행하여 효과를 검증하고 5장에서 결론과 향후 연구 방향으로 마무리한다.

2. 관련 연구

2.1 Container Orchestration

모놀리식으로 배포, 운영하던 것을 마이크로서비스아키텍처(Microservice Architecture, MSA)기반의 컨테이너 서비스화하여 구축하고 서비스하기 위해서는 효과적이고 효율적인 컨테이너의 배포, 관리를 위한 방안이 필요하다. 컨테이너 자체가 프로세스나 응용프로그램을 시스템으로부터 격리하기 위한 기술이기 때문에 개별 컨테이너를 생성하고 배치하는 것 자체는 쉬운 일이나 개별 컨테이너를 연결하고 관리하면서 필요에 따라 서비스 규모를 확장 또는 축소하면서 전체를 하나로 운영될 수 있도록 하기 위해서는 많은 주의와 노력이 요구된다. 서비스에 필요한 여러 대의 서버와 이를 활용한 컨테이너 실행환경을 효과적으로 구축하고 컨테이너의 배포, 관리, 확장을 지원하는 기술이 컨테이너 오케스트레이션(Container Orchestration)으로[9,10] 이는 서버 관리자의 역할을 대신하는 자동화된 관리, 운영환경을 구성할 수 있도록 지원하는 도구를 말하며 응용프로그램이나 서비스가 복잡하거나 두 개 이상의 서비스 컨테이너가 매쉬업 하여 서비스되는 상황이라면 컨테이너 오케스트레이션을 활용하여 관리하는 것이 좋다.

대표적인 컨테이너 오케스트레이션 도구로 쿠버네티스, Docker Swarm, Apache Mesos[11-13] 등이 있으며 그중에서도 쿠버네티스가 현재 가장 많이 사용되는 대표적인 컨테이너 오케스트레이션 도구이다. 쿠버네티스는 구글에서 내부적으로 사용하던 오케스트레이션 도구를 2014년 오픈소스로 공개하고 리눅스 재단에 기증한 것으로 현재는 리눅스 재단 산하의 클라우드 컴퓨팅 생태계 확산을 담당하는 CNCF(Cloud Native Computing Foundation)에서 오픈소스 프로젝트로 관리하고 있다[14].

CNCF에서는 매년 클라우드 네이티브 기술의 시장 정보를 조사해서 발표하는데 최신 보고서(Cloud Native Survey 2020)에 따르면 응답 기업의 91%가 오케스트레이션 도구로 쿠버네티스를 사용하고 있고, 그중 83%가 운영환경에도 적용중으로 사실상 표준으로 사용되고 있다[15].

2.2 오토-스케일링

클라우드 컴퓨팅은 필요에 따라 서비스를 확장하거나 축소함으로써 상황에 맞게 유연하게 처리할 수 있는 특징을 가지고 있으며 이는 온프레미스 방식의 시스템 구성과는 차별화된 장점 중 하나로 볼 수 있다. 오토-스케일링은 클라우드 컴퓨팅의 가장 기본적인 요소 기술 중 하나로 시스템의 상황이나 서비스의 상태에 따라 서버 사이징을 조절하여 갑작스런 트래픽의 집중처럼 예상치 못한 서비스 부하에 대해 효과적으로 대응하여 안정적인 서비스를 유지할 수 있도록 해주는 기능이다. 이를 위해 CPU, Memory, Disk I/O, Network I/O 와 같은 클라우드 리소스 사용량에 대한 매트릭 값을 주기적으로 수집하여 모니터링하고 설정한 스케일링 정책에 따라 적절한 서버 사이즈를 자동으로 조정하여 대응한다.

오토-스케일링은 할당된 자원을 변경하는 방식과 서비스 인스턴스를 개수를 변경하여 대응하는 방식이 있다. Fig. 1과 같이 전자는 스케일-업/다운(Scale-Up/Down)으로 서비스 노드상에 특정 리소스의 할당을 늘리거나 줄여서 시스템 상

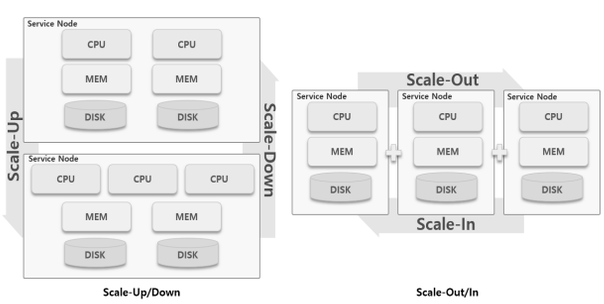


Fig. 1. Auto-scaling Methods

태나 서비스 상황에 대응하는 방법이고, 후자는 스케일-인/아웃(Scale-In/Out)으로 서비스 노드를 추가 실행하거나 종료하여 서비스를 상황에 맞게 안정적으로 유지한다.

오토-스케일링 기능을 활용하면 서비스 유휴 상태일 때는 서비스 노드를 최소로 유지를 하고 트래픽이 증가하거나 부하가 발생할 때, 이를 모니터링하고 있다가 적정한 서비스 자원이나 서비스 노드 개수를 늘리는 자동화된 유연한 대응으로 서비스 안정성을 높일 수 있다. 오토-스케일링 기능은 주기적으로 특정 시간대나 기간에 서비스 요청이나 트래픽이 집중되거나 급증 또는 그 반대의 경우나 배치 작업이나 주기적인 분석/데이터 적재 등 특정 시간대에 일시적인 컴퓨팅 자원이 필요한 경우와 같이 시시각각 변하는 워크로드에 맞게 시스템 리소스를 즉시 제어하거나 서비스 패턴에 맞게 트래픽을 배분하여 서비스의 성능과 안정성을 향상시키고 필요한 시점에 필요한 만큼만 사용함으로써 비용 절감 효과도 얻을 수가 있다.

2.3 다변량 시계열 분석

시계열 분석은 과거 시계열 데이터의 패턴이 미래에도 지속적으로 유지된다는 전제를 가지고 현재까지 수집된 데이터들을 분석하여 미래에 대한 예측을 수행하는 것이다. 각 시점에서 분석대상이 되는 변수의 개수로 일변량 시계열 분석(univariate time series analysis)과 다변량 시계열 분석(multivariate time series analysis)로 구분할 수 있는데 이는 분석대상 변수가 하나이냐, 두 개 이상이냐로 구분하는 것이다. 다변량 시계열 데이터를 분석하는 가장 큰 목적은 좀 더 신뢰할 수 있는 예측 시스템을 얻기 위함이다.

1) 시계열 분석

시계열 데이터(Time Serise Data)란 시간을 기준으로 측정된 데이터를 말하며 연도별, 분기별, 월별, 일별 또는 시간별 등 시간의 경과에 따라 순서대로 관측되는 자료다. 시계열 데이터는 이산적 시점에서 측정되는 이산시계열(discrete time seires)과 연속적으로 측정되는 연속시계열(continuous time serise)로 구분할 수 있다. 여기서 연속시계열 데이터는 시간의 모든 시점에서 측정된 자료를 의미하며 이산시계열 데이터는 특정한 시점에 측정한 관측값을 의미한다. 일반적으로 관측값 사이의 구간 간격을 일정하게 하여 측정된 데이터를 말한다. 연속시계열의 경우 모든 시점에서 측정되었기 때문에 분석하기 어려운 데이터 상태로 보통의 경우 이산시계열 데이터를 분석에 사용한다. 시계열 데이터는 다음과 같은 규칙적인 패턴과 불규칙적인 패턴의 특성을 가진다.

a) 규칙적인 패턴
- 자기 상관성(Autocorrelativeness)
  : 시계열 그 자체로 이전과 이후 결과 사이에서 발생하는 현상
- 이동평균(Moving Average)
  : 이전에 생긴 불규칙한 사건이 이후의 결과에 편향성을 초래하는 현상

b) 불규칙적인 패턴(White Noise)
- 일반적으로 이야기하는 White Noise를 말하며 평균이 0이며 일정한 분산을 지닌 정규분포에서 추출된 임의의 수
- 회귀식에서 설명되지 않는 Error의 의미와 유사

시계열 분석은 시계열 데이터에서 추세변동, 계절변동, 순환변동, 불규칙변동 등을 파악하여 미래를 예측하는 분석 방법으로 독립변수를 이용하여 종속변수를 예측하는 전통적인 기계학습 방법론에 대해서 시간을 독립변수로 사용한다는 특징을 가지는데 이는 독립변수와 종속변수 간 관계를 분석하고 설명할 때 가장 많이 사용하는 일반적인 회기분석과는 차이가 있다. 회기분석은 데이터의 분포나 두 데이터의 결과 간의 상관성을 주로 다루지만, 시계열 분석은 앞서 설명한 바와 같이 시간을 고려한다는 점에서 가장 큰 차이가 있다[16].

시계열 데이터가 지고 있는 대표적인 특징인 규칙성과 불규칙성을 가지는 패턴에 기반해서 시계열 분석 모델이 개발되어 왔고, 가장 대표적인 모형으로는 Box and Jenkins(1976)의 ARIMA(Autoregression Integrated Moving Average)모형이 사용된다. ARIMA모형은 현재의 관측치가 과거의 어떠한 규칙성에 의해서 재현되며, 이러한 규칙성은 미래에도 유지된다고 전제하고 미래를 예측하고자 했다. ARIMA는 자기회귀와 이동평균을 둘 다 고려하는 모델로서 시계열의 비정상성을 설명하기 위해 관측치 간의 차분을 사용하는데 이는 현실에 존재하는 대부분의 시계열 데이터는 불안정성을 내포하고 있기 때문에 모형 그 자체에 비정상성을 제거하는 과정을 포함하여 모델의 성능을 높이고자 했다. ARIMA는 모형 설정이 용이하다는 장점을 가진 반면 변수들 사이의 상호작용을 무시하고 있어 일변량 분석이라는 한계를 가지고 있다[17].

2) 벡터자기회기

일변량 시계열 분석은 종속변수가 독립변수들에만 영향을 받는다는 가정이 존재하는데 현실 세계에서는 종속변수와 독립변수는 상호 영향을 주고받는 경우가 대부분이다. 이런 경우 회기모형이나 일변량 시계열 분석 방법으로는 한계가 있는데 이러한 한계를 보완한 모형이 크리스토퍼 앨버트 심스 교수가 1980년에 발표한 경제학 통계분석 모형인 벡터자기회기(Vector Autoregression, VAR) 모형이다[17].

VAR은 회귀분석과 시계열 분석의 특징을 결합하여 변수 간에 나타날 수 있는 상관관계와 인과관계를 추정할 수 있는 다변량 시계열 분석 모형으로 일변량 자기회기 모형을 다변량 자기회기 모형으로 확장하여 경제학 등에서 내생변수의 변화에 따른 효과분석과 예측을 수행할 때 자주 활용되는 대

표적인 다변량 시계열 분석 방법이다. VAR 모형은 연립방정식 체계와 매우 유사하며 모형의 오차항을 구조적으로 해석하고 식별제약의 일부가 오차항의 공분산행렬에 가해진다는 특징을 지닌다[17]. 또한 특정 이론에 기초한 가설을 전제하지 않고 실제로 관찰되는 시계열 데이터들이 주는 정보만을 최대한 이용하여 보이는 현상 그대로를 분석한다. 즉, VAR 모형은 모든 변수에 대한 시차변수을 동시에 종속변수로 이용하여 결과를 분석하는 것과 같다. N개의 다변량 정상시계열로 구성된 $X_t = (X_{1t}, X2_t, \cdots, X_{Nt})$가 p시차인 자기회기과정으로 구성된 벡터자기회기모형 VAR(p)라하며 다음 Equation (1)과 같이 정의한다. 여기서 C는 $(Nx1)$상수벡터, Θ_i는 현시점의 변수와 시차변수들간 시차회귀 계수인 (NxN) 행렬, ε_t는 $(Nx1)$의 벡터백색잡음이다[17].

$$X_t = C + \theta_1 X_{t-1} + \cdots + \theta_p X_{t-p} + \varepsilon_t \\ = C + \sum_{i=1}^{p} \theta_i X_{t-i} + \varepsilon_t \quad (1)$$

Equation (2)은 두 개의 종속변수에 대한 t-1 시간까지만을 고려하는 VAR 알고리즘 예시이다[18]. 백터자기회기 방정식은 종속변수의 개수만큼의 개별 회기 방정식으로 풀 수 있다.

$$\begin{aligned} &\text{Equation of VAR(1)} && Y_{[t]} = AY_{[t-1]} + e_{[t]} \\ &\text{where} && Y_{[t]} = \begin{bmatrix} Y_{1t} \\ Y_{2t} \end{bmatrix}, A = \begin{bmatrix} \phi_{11}\phi_{12} \\ \phi_{21}\phi_{22} \end{bmatrix}, Y_{[t-1]} = \begin{bmatrix} Y_{1t-1} \\ Y_{2t-1} \end{bmatrix}, \\ & && e_{[t]} = \begin{bmatrix} e_{1t} \\ e_{2t} \end{bmatrix} \sim N(0, \sum_{e[t]}) \\ &\text{Each Equation of VAR(1)} && Y_{1t} = \phi_{11} Y_{1t-1} + \phi_{12} Y_{2t-1} + e_{1t} \\ & && Y_{2t} = \phi_{21} Y_{1t-1} + \phi_{22} Y_{2t-1} + e_{2t} \end{aligned} \quad (2)$$

3. 설계 및 구현

본 장에서는 다변량 시계열 분석을 활용한 클라우드 오토-스케일링 방안을 제안한다. 쿠버네티스 클러스터로 서비스 모듈이 구동되는 일반 노드와 이를 관리하는 마스터 노드를 구성하고 컨테이너가 실행되는 Pod 상태와 서비스 상태를 모니터링하여 예측에 기반한 오토-스케일링 알고리즘을 실행한다.

3.1 구조 및 구성요소

본 논문에서는 커스텀 스케쥴러를 통해 Pod 오토-스케일링을 수행하는 쿠버네티스 클러스터 구성을 제안한다. 쿠버네티스는 마스터 노드와 일반 노드들로 구성되며 마스터 노드는 노드에서 발생하는 다양한 이벤트를 감지하고 응답, 관리하는 역할을 수행하는데[19] 본 논문에서 제안하는 다변량 시계열 분석을 통한 오토-스케일링을 수행하는 커스텀 스케쥴러는 마스터 노드에서 실행된다.

Fig. 2는 제안하는 커스텀 스케쥴러가 포함된 아키텍처이다. 마스터 노드에는 scheduler, API server, controler, etcd 등 쿠버네티스 플레인 컴포넌트들뿐만 아니라 일반 노드에서 실행되는 Pod들에 대한 오토-스케일링을 전담 수행하

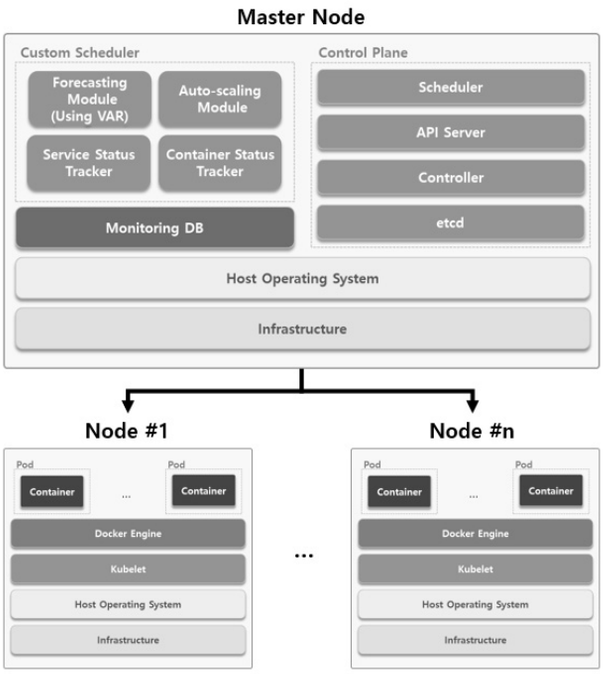


Fig. 2. System Configuration of Custom Scheduler

Table 1. Functions of Custom Scheduler Components

Node	Component	Function
Master Node	Service Status Tracker	Collect service status information running on service containers
	Container Status Tracker	Collect status information for containers running on each Worker Node
	Forecasting Module	Multivariate analysis and prediction data generation (resource and service performance at next observation point)
	Auto-Scaling Module	Auto-scaling based on forecasting data from the Forecasting Module (Scale-In/Out)

는 커스텀 스케쥴러가 구성되어 실행된다. 커스텀 스케쥴러의 4가지 구성요소의 역할과 기능은 Table 1에 정리되어 있다.

3.2 수행 절차

다변량 시계열 데이터 분석을 통한 오토-스케일링을 수행하기 위해 분석과 예측에 필요한 메트릭 데이터를 주기적으로 수집한다. 수집하는 메트릭 데이터에는 실행 중인 각 Pod의 CPU 사용량뿐만 아니라 외부에서 호출된 이벤트에 대한 서비스 응답시간까지 수집하여 분석과 예측에 사용한다. MSA 사상에 기반한 작고 경량화된 단일 서비스 모듈 하나 정도만 컨테이너에 올리는 경향을 감안하면 컨테이너의 자원 사용률이 높아진다는 것은 그만큼 외부의 서비스 호출이 발생한다는 것이고 서비스 처리 건수가 늘어남에 따라 자원 사용률과 서비스 응답시간에 영향을 줄 수 있다는 점에서 수집

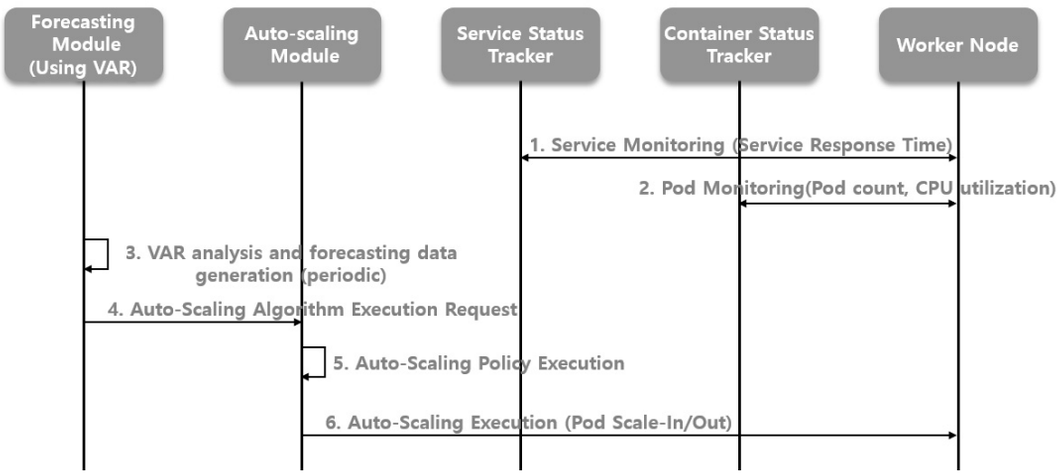


Fig. 3. Sequence Diagram

하여 분석하는 두 가지 메트릭 사이에는 상관성이 있다.

Fig. 3은 다변량 시계열 데이터 분석을 통한 오토-스케일링 수행 과정을 나타낸다.

Service Status Tracker와 Container Status Tracker는 각각 Pod에서 실행되는 서비스 응답시간과 실행중인 Pod의 개수와 CPU 사용률을 일정 간격으로 모니터링을 하여 이들 데이터를 DB에 기록한다. Forecasting Module은 주기적으로 VAR 모형을 이용한 분석과 예측을 수행하고, 이 예측 데이터를 기반으로 Auto-scaling Module이 설정된 정책에 따라 스케일-인 또는 스케일-아웃을 수행하여 Pod 상태와 서비스 상태에 따라 실행되는 Pod의 수를 적절하게 조절한다.

3.3 다변량 시계열 분석 알고리즘

VAR 모형을 사용한 다변량 시계열 데이터 분석 알고리즘은 최근 5분간 서비스 상태 메트릭 데이터(서비스 응답시간)와 컨테이너의 자원 사용량(CPU 사용량)을 지정된 시간 간격으로 집계하고 구간 평균값을 사용하여 모델을 생성하여 예측을 수행한다. Equation (3)은 두 개의 종속변수인 서비스 응답시간(Y_r)과 CPU 사용량(Y_c)에 대한 p차수인 VAR 방정식이다.

$$\text{Equation of VAR(p)}\quad Y_{[t+1]} = AY_{[t]} + \varepsilon_{[t+1]}$$

$$\text{where}\quad Y_{[t+1]} = \begin{bmatrix} Y_{ct+1} \\ Y_{rt+1} \end{bmatrix},\ A = \begin{bmatrix} \emptyset_{11} & \dots & \emptyset_{1p} \\ \emptyset_{21} & \dots & \emptyset_{2p} \end{bmatrix},\ Y_{[t]} = \begin{bmatrix} Y_{ct} \\ Y_{rt} \end{bmatrix} \tag{3}$$

$$\text{Each Equation of VAR(p)}\quad Y_{ct+1} = \emptyset_{11}Y_{ct} + \emptyset_{12}Y_{ct-1} + \dots + \emptyset_{1p}Y_{ct-p-1} + e_{1t+1}$$

$$Y_{rt+1} = \emptyset_{11}Y_{rt} + \emptyset_{12}Y_{rt-1} + \dots + \emptyset_{1p}Y_{rt-p-1} + e_{2t+1}$$

쿠버네티스의 기본 오토-스케일링 기능은 지정된 시스템 자원의 현재(t시점) 사용률을 이용하여 오토-스케일링 동작여부를 결정하지만 제안하는 방식은 Fig. 4와 같이 현재까지의 수집된 데이터를 기반으로 Equation (3)과 같이 VAR 모형을 활용하여 다음 구간(t+1 시점)에 대한 서비스 응답시간과 CPU

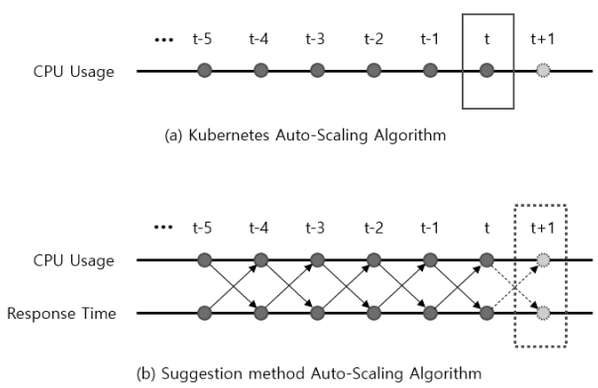


Fig. 4. Suggestion Method Auto-Scaling Algorithm

사용률을 예측하여 설정된 정책에 기반하여 Pod의 실행개수를 늘리거나 줄이는 선제적인 오토-스케일링을 수행한다.

Fig. 5는 통계분석 패키지를 이용한 다변량 시계열 분석 및 예측을 수행하는 알고리즘의 Pseudo Code이다.

VAR 모형을 이용한 다변량 시계열 데이터 분석을 통해 다음 시차의 예측 결과를 가지고 오토-스케일링을 수행하게 된다 오토-스케일링 알고리즘은 VAR 모델을 이용하여 예측한

Pseudo Code for Multivariate Time Series Analysis Algorithm

```
# Load matrix data
metricDatasOfWorker = loadMetricDatasOfWorkerNodeForVAR(nTerm)

# Create VAR model instance
forecastModel = VAR(metricDatasOfWorker)

# Select Model select indicator(aic)
k_ar = forecastModel.select_order().selected_orders['aic']

# Model Fitting
forecastModelResults = forecastModel.fit(maxlags=k_ar, ic='aic')

# Generate forecasting data
forecastDataOrg = forecastModelResults.forecast()
```

Fig. 5. Pseudo Code for Analysis

CPU 사용량이 모니터링 임계값을 넘었을 때 예측한 서비스 응답시간을 현재 상태값과 비교하여 증감 여부에 따라 스케일링을 결정한다.

CPU 사용량이 모니터링 관리 포인트를 넘어선 상태에서 서비스 응답시간이 느려질(증가) 것으로 예측될 경우에는 Pod를 추가 실행하는 Scale-Out을 수행해서 실행 중인 서비스 노드를 증가시켜 대응한다. 이는 서비스 호출에 따른 CPU 자원을 임계값 이상으로 많이 사용하는 상태에서 응답시간이 느려지는 상황을 의미하는 것으로 서비스 노드를 늘려서 로드밸런싱을 통해 늘어나는 서비스 호출을 분산처리하기 위함이다. CPU 사용량이 임계값 이상이나 서비스 응답시간은 빨라질 것으로 예상되거나 CPU 사용량은 감소하는데 서비스 응답시간은 유지되거나 증가하는 경우에는 현재 실행된 Pod 상태를 유지한다. 두 측정 지표 중 하나만 관리 영역 상태로 감지되는 경우에는 쿠버네티스가 클러스터 내부에서 컴퓨팅 자원 활용을 자동으로 조절하여 대응하는 기능을 사용하여 대응한다. 이를 통해 자원 사용이 증가하는데 서비스에는 영향이 없거나 자원 사용은 관리범위 내인데 서비스 응답시간에 영향이 있을 것으로 예측될 경우에는 워커노드 자체가 가지고 있는 리소스 범위내에서 유휴 자원을 설정된 범위 내에서 사용함으로써 처리하여 대응할 수 있다. Table 2는 항목별 예측값의 증감 여부에 따른 스케일링 정책을 정리하였다.

CPU 사용량이 임계값 이하이고 응답시간이 빨라질 경우에는 점진적으로 실행된 Pod 개수를 줄여나간다.

제안하는 오토-스케일링 알고리즘을 실행하는데 필요한 파라미터들은 Table 3과 같다.

Fig. 6는 VAR 모델을 이용한 예측 데이터를 이용한 오토-스케일링 알고리즘의 Pseudo Code이다.

Table 2. Auto-Scaling Policy

	Increase CPU Usage	Reduce CPU Usage
Increase Response Time	Scale-Out	Keep the status quo
Reduce Response Time	Keep the status quo	Scale-In

Table 3. Parameters for Auto-Scaling

Parameter	Description
SPmin	Minimum number of pods to be executed
SPcurrent	Number of Pods currently running
SPmax	Maximum number of Pods you can run
CPUUCL	Scale-Out reference value as upper CPU utilization control limit for performing auto-scaling algorithms
CPULCL	Scale-In reference value as lower CPU utilization control limit for performing auto-scaling algorithms

Pseudo Code for Auto-Scaling Algorithem

```
for pod to pods
    # VAR forecasting
    doVARForecast()
    # Predicting situations where CPU predictions are above threshold
    # and service response times are slow
    if pod.cpu_forecast => CPUCL and pod.responseTime_forecast => pod.responseTime_current
        if SPcurrent < SPmax
            increase Pod
        end if
    # Predicting situations where CPU predictions are below threshold
    # and response times are accelerated
    else if pod.cpu_forecast < CPUCL and pod.responseTime_forecast < pod.responseTime_current
        # Overall Pod CPU Utilization Predicted Layer Average Below LCL
        # Decreased by Minimum Number of Running Pods
        if pod.cpuavg_forecast < CPULCL
            decrease Pod to SPmin
        else
            decrease Pod
        end if
    end if
end for
```

Fig. 6. Pseudo Code for Auto-Scaling Algorithem

Source code for VAR Analysis

```
# Import packages for VAR Analysis
from statsmodels.tsa.api import VAR
from statsmodels.tsa.stattools import adfuller

# Create VAR Model instance
forecastModel = VAR(workerMetricData_real)

#  Model delect indicator by AIC(Akaike's Information Criterion)
lagorderresults = forecastModel.select_order()
k_ar = lagorderresults.selected_orders['aic']

# Model Fitting
forecastModelResults = forecastModel.fit(maxlags=k_ar, ic='aic')

# Forecasting
forecastDataOrg = \
forecastModelResults.forecast(workerMetricData_real.values[-k_ar:], 1).reshape(3,)
```

Fig. 7. Source Code For VAR Analysis

3.4 커스텀 스케쥴러 구현

커스텀 스케쥴러는 Pod들의 상태와 서비스 상태를 VAR 모형으로 분석하고 예측된 데이터 이용하여 오토-스케일링을 수행한다. 커스텀 스케쥴러 개발은 VAR 모형을 이용한 분석과 예측이 가능하고, 그 예측 결과를 바탕으로 컨테이너 오토-스케일링을 수행하는데 용이해야 한다. Python에는 R에서 가능했던 다양한 회귀분석과 시계열 분석 방법론을 지원하는 statsmodels 패키지가 있다[20]. statsmodels은 통계적 추정 및 검정과 회귀분석 그리고 시계열 분석에 필요한 기능을 가지고 있다. 본 논문에서는 이 패키지에서 제공하는 VAR 모형을 이용하여 Forecasting Module를 구현한다. Fig. 7은 VAR 모형을 이용한 예측 데이터 생성 코드를 나타낸다.

VAR 모델에서 사용되는 AIC 값은 데이터셋에 대한 통계 모델의 상대적 품질을 평가하는 것으로 최소 정보 손실을 갖는 모델이 선택될 수 있도록 한다. 낮은 AIC 값이 모형의 적합도가 높은 것을 의미한다.

Fig. 8는 오토-스케일링 코드를 나타낸다. 예측 데이터를 최근 관측된 데이터와 비교하여 상황에 맞게 스게일-인/아웃을 수행하며 전체 Pod CPU 사용률 평균값이 CPU 사용률 관리 하한값보다 낮은 사용률로 측정되는 경우 Pod 실행개수를 최저 개수로 조정한다.

Source Code for Auto-Scaling

```
CPUUCL = 90
CPULCL = 40
SPMax = 4
SPMIN = 2

# Scale-In/Out Function
def podScaling(deploymentName, replicasCount):

    output = os.popen('kubectl scale deploy {0}
--replicas={1}'.format(deploymentName, replicasCount)).read()

    if 'scaled' in output.strip():
        return True
    else:
        return False

for pod in pods:
    if autoScaleCheckFlag == False:
        #forecast는 responsetime, cpu 순
        podMetricData_current = podMetricData_real.tail(1)
        # Auto-Scaling : out
        if forecastDataCorrect[1] > CPUUCL and forecastDataCorrect[0] >  \
          podMetricData_current[0]
            if autoScaleInOutFlag == False and runningPodCount < SPMax:
                print('scale out : {0}-{1}'.format(testStatus, 'each'))
                result = podScaling(deploymentName, runningPodCount + 1)
                autoScaleCheckFlag = result
                if result:
                    break
        # Auto-Scaling : in
        else if forecastDataCorrect[1] < podMetricData_current[1] \
                and  forecastDataCorrect[0] < podMetricData_current[0]:
            if forecastDataPodMeanCorrect < CPULCL:
                print('scale in default : {0}-{1}'.format(testStatus, 'each'))
                result = podScaling(deploymentName, SPmin)
                if result:
                    break
            else:
                if runningPodCount >SCmin
                result = podScaling(deploymentName, runningPodCount - 1)
                autoScaleCheckFlag = result
```

Fig. 8. Source Code for Auto-Scaling

4. 실험 및 분석

본 장에서는 제안하는 다변량 시계열 분석을 활용한 컨테이너 오토-스케일링 방안의 성능을 평가하기 위한 테스트를 진행하고 결과를 확인하여 성능을 평가한다. 성능 평가는 시스템 부하에 따른 컨테이너의 자원 사용률과 서비스 응답시간을 비교하여 수행한다. 오토-스케일링이 적용되지 않은 시스템과 운영환경에서 일반적으로 사용되는 자원 사용률 기반의 오토-스케일링이 적용된 시스템 그리고 본 논문에서 제안하는 다변량 시계열 분석에 기반한 오토-스케일링 기법이 적용된 시스템의 결과를 비교하여 어떤 시스템이 효과적인지 확인한다. 실험에서는 JMeter[21]를 사용하여 시스템에 대한 부하를 증가시켜 증가한 부하에 따른 자원 사용률과 서비스 응답시간을 비교한다. 오토-스케일링이 적용되지 않은 시스템 대비 오토-스케일링을 적용했을 때의 자원 사용률과 서비스 응답시간의 변화를 비교하여 오토-스케일링 서비스 안정성에 미치는 영향을 확인하고, 이후 오토-스케일링 방식에 따른 자원 사용률과 응답시간을 확인한다.

4.1 실험 환경

본 논문에서의 실험 환경은 퍼블릭 클라우드 컴퓨팅 서비스를 활용하여 4개의 클라우드 서버 인스턴스를 생성하여 쿠버네티스 클러스터를 구축한다. 4개의 서버 인스턴스 중 1개는 쿠버네티스 마스터 노드로 사용하고 2개는 일반 노드로 사용하고 나머지 1개는 쿠버네티스 클러스터 구성 및 일반 노드에서 사용되는 Docker image를 위한 Docker Private Registry 구축에 사용된다.

Table 4. Server Instance

Sortation	Master Node	Worker Node	Etc.
Purpose	Kubernetes Master Node	Kubernetes Worker Node	Docker Private Register
Instance Type	t3.small	t3.medium	t3.small
CPU	vCPU 2EA	vCPU 2EA	vCPU 2EA
RAM	2GB	4GB	2GB
OS	Ubuntu 20.04	Ubuntu 20.04	Ubuntu 20.04

실험에 사용된 클라우드 서버 인스턴스와 시스템 구성도는 Table 4와 Fig. 9와 같다.

실험에는 쿠버네티스 클러스터뿐만 아니라 로깅에 필요한 DBMS와 로깅처리에 따른 처리 지연을 막고자 Message Queue, 그리고 쿠버네티스 Pod에 사용할 Docker Image를 위한 Private Resistry도 구성하며 실험에 필요한 서비스 부하를 발생시키기 위해 JMeter도 포함한다.

4.2 실험 방법 및 시나리오

본 논문의 실험은 제안하는 오토-스케일링 기법이 정상적으로 동작하는지, 쿠버네티스 기본 스케쥴링 방식 대비 다변량 분석을 통한 예측 정보를 활용한 스케일링의 효과를 확인하기 위해 부하 테스트를 진행한다. 쿠버네티스 클러스터에 RestfulAPI 형태의 테스트용 서비스를 기동하고 일정 시간 동안 워크로드를 발생시켜 테스트 중에 실행 중인 Pod 개수, 응답시간, CPU 사용률을 비교한다. 실험에 사용한 테스트용 RestfulAPI는 서비스가 호출되면 다층 신경망을 이용하여 컴퓨팅파워를 사용하도록 구성했다. Fig. 10은 JMeter 테스트 설정한 것으로 Max Thread에 도달할 때까지 점진적으로 Ramp-up하여 최대 부하를 발생시키도록 한다. 설정한 Max Thread 개수는 100개로 Ramp-up 구간에서는 4초마다 5개씩 쓰레드를 증가시켜 1분 20초에 Max Thread에 도달하고 이를 120초간 유지하여 실행되는 Pod 개수, 서비스 응답시간 그리고 각 Pod별 CPU 사용률 정보를 로깅한다.

4.3 실험 결과

본 논문의 실험 결과를 평균응답시간, 평균 CPU 사용률, 평균 실행 Pod 개수로 비교를 해보면 Fig. 11의 (a), (b), (c)와 같다. 본 논문에서 제안한 오토-스케일링 방식과 쿠버네티스의 기본 스케쥴링 방식을 측정 항목별 평균값으로 비교하면 응답시간 평균의 경우 약 1.5% 가 빨라지고, CPU 사용률 평균의 경우 약 0.3% 정도 적게 자원을 사용하고 Pod 실행 갯수 평균의 경우 4.5% 적게 실행되는 것으로 나타난다. 수치상으로 보여지는 개선 효과는 크지 않으나 오토-스케일링이 시스템의 상황이나 서비스의 상태에 따라 서비스 노드를 확장하여 갑작스러운 트래픽의 집중과 같은 예상치 못한 서비스 부하에 대해 효과적으로 대응하여 안정적인 서비스를

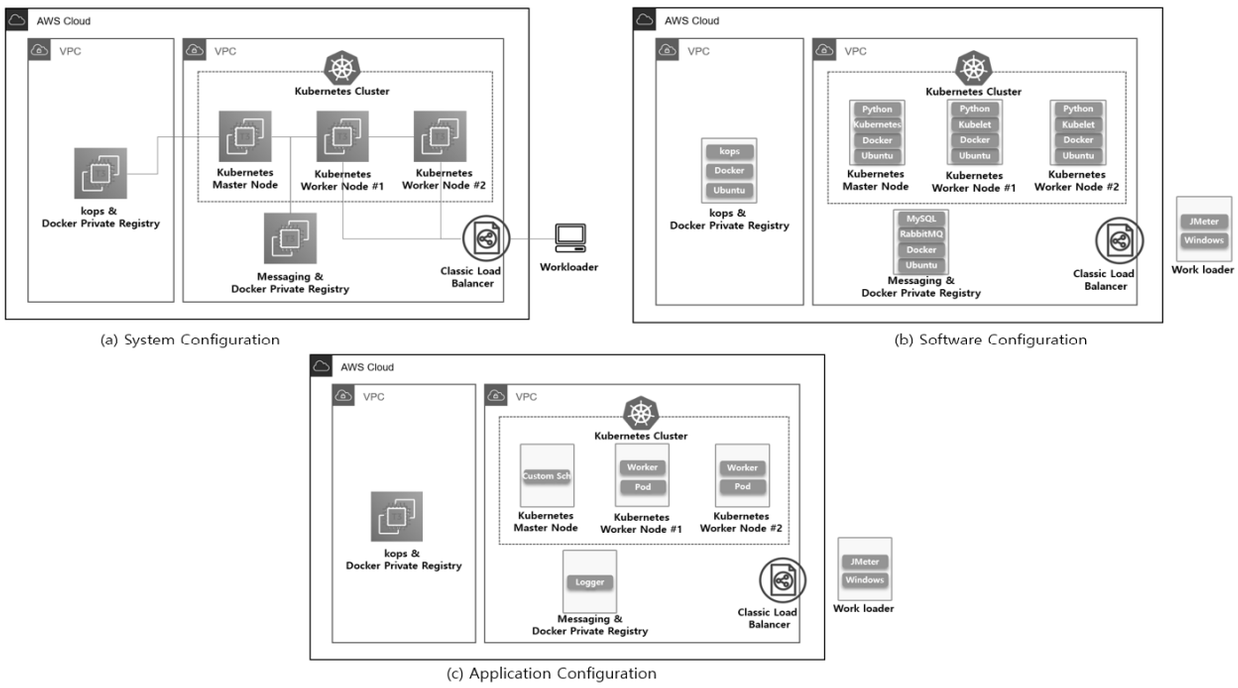


Fig. 9. Configuring the Experimental Environment with AWS

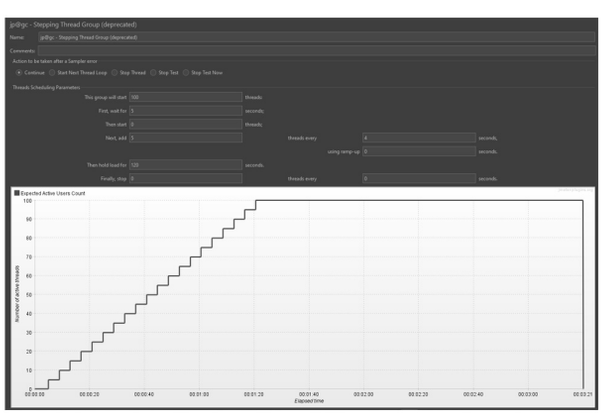


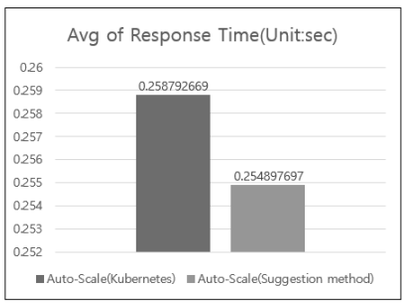
Fig. 10. JMeter Test Setting

유지할 수 있도록 하거나, 필요 이상의 서비스 노드가 실행되는 것을 막아 시스템 자원을 효율적으로 사용할 수 있도록 하는 기능이라는 점에서 실험의 세부적인 상황에 맞는 적절한 스케일링이 수행되고 이를 통해 해당 시점의 시스템 상황과 서비스 상황을 안정적으로 관리했는지 살펴볼 필요가 있다.

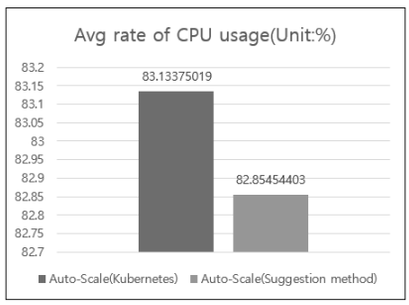
Fig. 11의 (d), (e)는 두 가지 오토-스케일링 방식의 실행 중인 Pod 개수와 서비스 응답시간, Pod 개수와 CPU 사용률을 비교하는 차트이다. 제안한 방식이 기존의 쿠버네티스 오토-스케링일과 의미있는 차이를 보이고 있는 구간이 표시되어 있다.

Fig. 11의 ① 부분은 실험을 위해 서비스 호출이 점진적으로 증가하면서 발생하는 로드 증가 및 응답시간 지연에 따라 커스텀 스케쥴러에서 선제적으로 Pod를 추가 실행함으로써 CPU 사용률을 약 13% 낮게 안정적으로 대응하는 것을 보여준다. Fig. 11의 ② 지점은 테스트 부하가 Max Tread에 다다르는 시점으로 커스텀 스케쥴러가 스케일-아웃을 실시하여 Pod 개수를 3개에서 4개로 변화하여 자원 사용 및 응답시간이 늦어지는 것을 사전 예방하고자 선 대응하는 것을 볼 수 있다. 앞서 살펴본 Pod의 개수가 2에서 3으로 늘어나는 시점과 비교하여 볼 때 상대적으로 CPU 사용률이 적게 떨어지는 것은 설정된 Max Thread 부하가 실험에서 설정한 Pod 최대 실행 개수에서 부하가 상당한 설정으로 Pod 실행 개수의 제약이 없었다면 이전 상황에서 살펴봤듯이 선제적으로 Pod를 관리하였을 것으로 예상된다. Fig. 11의 ③ 지점은 Max Thread 상의 부하가 유지되고 있지만 CPU 사용률과 응답시간이 감소되는 트렌드를 감안한 예측에 따라 Pod를 줄였으나 서비스는 안정적으로 유지되고 있음을 보여주고 있다. 이는 적절한 오토-스케일링을 통해 일정한 성능을 유지하면서도 서비스 노느나 Pod를 적게 사용할 수 있도록 관리 함으로써 퍼블릭 클라우드 서비스 환경에서는 클라우드 서비스 비용을 줄일 수도 있음을 시사한다. Fig. 11의 ④ 부분은 일시적으로 자원 사용이 증가하는 현상에 제안하는 방식의 스케쥴러가 기민하게 대응했다 원래 상태로 복귀할 수 있음을 보여준다.

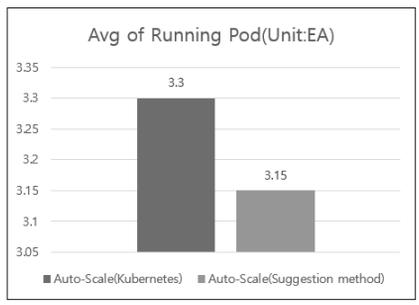
실험 결과를 종합해보면 본 논문에서 제안한 다변량 시계열 분석에 기반한 커스텀 스케쥴러는 쿠버네티스의 기본 스케쥴러와 비교하여 비록 수치상의 비교 결과는 다소 미미할지라도 예측 정보에 기반한 선제 대응을 통해 자원 사용과 서비스 응답시간의 변화를 안정적으로 유지·관리하거나 서비스 품질의 저하가 없거나 최소화되면서도 적절한 서비스 인스턴스를 실행하여 운영비용의 절감할 수 있을 것으로 기대된다.



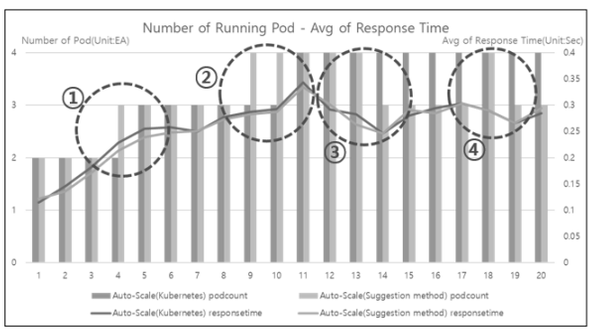
(a) Avg of Response Time



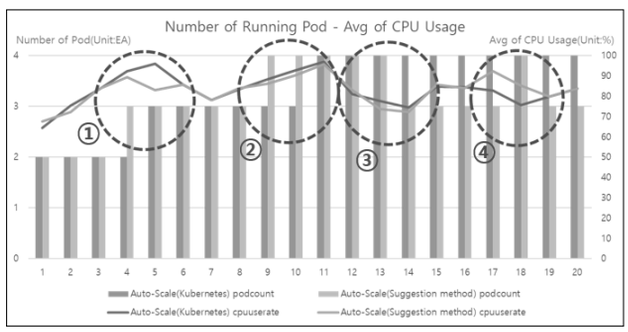
(b) Avg rate of CPU usage



(c) Avg of Running Pod



(d) Number of Running Pod - Avg of Response Time



(e) Number of Running Pod - Avg of CPU Usage

	①							②					③					④		
t	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Running Pod	2	2	2	3	3	3	3	3	4	4	4	4	4	3	3	3	3	4	3	3
Gap of Running Pod Count	0	0	0	1	0	0	0	0	1	1	0	0	0	-1	-1	-1	-1	0	-1	-1
Gap of Response Time	0.00710	-0.01000	-0.01000	-0.01529	-0.01670	-0.01000	0.00050	-0.00498	-0.00484	-0.00520	-0.00899	0.00991	-0.01999	0.00000	0.01029	-0.00970	0.00000	0.00000	0.00000	0.01000
Gap of CPU Usage	3.00	-3.00	0.21	-3.00	-13.00	0.00	0.00	0.60	-2.02	-2.45	-1.45	2.00	-4.00	-2.04	1.00	-0.42	9.58	9.41	0.00	0.00

(f) difference in data values between the suggestion method and Kubernetes

Fig. 11. Comparison of Results for Each Auto-scaling Algorithm

또한 급격한 상태 변화에도 빠른 스케일 대응으로 서비스 변화에 대한 대응 민감도가 기존 방식 대비 높고, 효과적임을 확인할 수 있다.

5. 결 론

클라우드 컴퓨팅은 기존 온프레미스 기반의 시스템 구축 및 운영과 비교하여 많은 장점이 있고 그 확산이 가속화되고 있다. 특히 오토-스케일링 기능은 클라우드 컴퓨팅 환경에서 제공하는 가장 기본적인 요소 기술 중 하나로 서비스 환경의 변화와 서비스 상태에 따라 빠르고 유연하게 서비스 확장하거나 축소하여 적절한 대응을 할 수 있는 효과적인 기술이다.

이에 본 논문에서는 클라우드 환경에서 다변량 시계열 데이터 분석에 기반한 오토-스케일링 방안을 제안하였다. 제안한 오토-스케일링 알고리즘은 서비스 모듈에 대해서 단일 메트릭으로 수집되어 개별적으로 모니터링하고 관리하던 방식에서 메트릭간 영향도를 복합적으로 분석하고 예측하여 시스템의 안정성은 높이고 서비스의 품질은 유지 또는 향상될 수 있도록 선제적이고 공격적으로 오토-스케일링 정책을 수행하는 방식이다.

VAR 모형은 어떠한 이론적 배경과 가설이 없이도 실제 관찰되는 시계열 데이터들이 주는 정보만을 최대한 이용하여 시계열들간의 상태를 분석할 수 있게 한다. 이러한 모형의 특징을 활용하여 서비스 컨테이너로 유입되는 서비스 호출에 따라 Pod의 CPU 사용량 그리고 응답시간을 분석하여 다음 시차의 각 항목에 대한 예측값을 생성하고 예측되는 상황에 따라 새로운 Pod를 실행하거나 종료하는 스케일링을 수행함으로써 서비스 부하를 분산하고 응답시간 보장할 수 있으며, 상황에 맞게 실행되는 Pod 개수를 적절하게 관리할 수 있음을 실제 서비스 컨테이너가 구동되는 노드와 마스터 노드를 구성하여 성능 평가를 통해 결과를 확인하였다.

본 논문에서 제안한 오토-스케일링 방안은 컨테이너 가상화 기술을 사용한 클라우드 환경에 적용할 수 있다. 시스템의 워크로드에 맞게 시스템 리소스를 즉시 제어하거나 서비스 패턴에 맞게 트래픽을 배분하여 서비스의 성능과 안정성을 향상시키고 필요한 시점에 필요한 만큼만 사용함으로써 비용 절감 효과도 얻을 수 있을 것으로 기대한다.

향후에는 좀 더 다양한 메트릭 정보를 수집하고 현업환경

의 복합적인 시스템 상황에 대한 효용성을 높일 수 있는 고도화 연구를 진행하고자 한다.

References

[1] S. W. Ahn, "Changes in cloud virtualization technology - container-based cloud virtualization and DevOps," *Issue Report of Software Policy & Research Institute*, No.2018-008, pp.1-36, 2018.

[2] A. Fazli, A. Sayedi, and J. D. Shulman, "The effects of autoscaling in cloud computing," *Journal of Management Science*, Vol. 64, No.11, pp.5149-5163, 2018.

[3] M. A. Netto, C. Cardonha, R. L. Cunha, and M. D. Assuncao, "Evaluating auto-scaling strategies for cloud computing environments," *2014 IEEE 22nd International Symposium on Modelling, Analysis & Simulation of Computer and Telecommunication Systems*, IEEE, 2014.

[4] S. H. Eum and J. S. Kim, "Method for efficient storage resource management in docker container-based cloud environments," in *Proceeding The Korean Institute of Information Scientists and Engineers*, pp.1247-1249, Jul. 2020.

[5] M. S. Yhu, C. G. Song, H. C. Yu, and E. Y. Lee, "Task type based autoscaling criteria selection and performance analysis in Kubernetes cluster," in *Proceeding the Korean Institute of Information Scientists and Engineers*, pp.1382-1834, Jun. 2021.

[6] S. Y. Choi, "Cloud infrastructure abnormal monitoring using machine learning," *Journal of the Korea Society of Computer and Information*, Vol.25, No.4, pp.105-112, Apr. 2020.

[7] Y. H. Jang, J. W. Jeong, J. H. Lee, H. C. Yu, and E. Y. Lee, "Modelling the kubernetes horizontal pod autoscaler based on number of requests for comparing the autoscaling algorithm," in *Proceeding the Korean Institute of Information Scientists and Engineers*, pp.1385-1837, Jun. 2021.

[8] L. Toka, G. Dobreff, B. Fodor, and B. Sonkoly, "Machine learning-based scaling management for kubernetes edge clusters," *IEEE Transactions on Network and Service Management*, Vol.18, No.1, pp.958-972, 2021.

[9] W. S. Choi, H. J. Chung, and Y. M. Nah, "Orchestration method for efficient resource management in container-based cloud environments," *Journal of Korean Institute of Next Generation Computing*, Vol.15, No.2, pp.71-78, Apr. 2019.

[10] S. M. Jeong, J. G. Kim, K. H. Yeom, and J. S. Park, "Container orchestration framework deployment technique based on the feature of microservice," *Journal of the Korean Institute of Communication Sciences*, Vol.46, No.3, pp.466-475, Mar. 2021.

[11] Kubernetes [Internet], https://kubernetes.io

[12] Docker Swarm [Internet], https://docs.docker.com/engine/swarm

[13] Apache Mesos [Internet], http://mesos.apache.org

[14] CNCF Kubernetes Projects [Internet], https://www.cncf.io/projects/kubernetes

[15] CNCF [Internet], https://www.cncf.io/reports/#cloud-native-surveys

[16] A. Nielsen, "Chapter.6 Satistical Models for Time Series," in Practical Time Series Analysis, Hanbit Media, Inc., pp.217-264, 2021.

[17] K. S. Moon, "A understanding of vector autoregressive model," *Journal of the Korean Official Statistics*, Vol.2 No.1, pp.23-56, Apr. 1997.

[18] Time Series Algorithm (Multivariate linear probability) [Internet], https://github.com/cheonbi/OnlineTSA/blob/master/Lecture4_Algorithms_TimeSeries_Linear_Multivariate_KK.ipynb

[19] K. I. Kim, et al., "Kubernetes Architecture for Cloud Service," *Information and Communications Magazine*, Vol.35, No.11, pp.11-19, Oct. 2018.

[20] statsmodels [Internet], https://www.statsmodels.org/stable/index.html

[21] JMeter [Internet], http://jmeter.apache.org/

김 용 회

https://orcid.org/0000-0002-9621-9757
e-mail : yonghaefromjinhae@gmail.com
2002년 대전대학교 컴퓨터공학과 (공학사)
2021년 숭실대학교 IT융합학과(공학석사)
2021년 ~ 현 재 숭실대학교 IT융합학과 박사과정
관심분야 : 클라우드 컴퓨팅, 시계열 데이터 분석, 자연어 처리, 최적화

김 영 한

https://orcid.org/0000-0002-1066-4818
e-mail : younghak@ssu.ac.kr
1984년 서울대학교 전자공학과(공학사)
1986년 한국과학기술원 전기 및 전자공학 (공학석사)
1990년 한국과학기술원 전기 및 전자공학 (공학박사)
1994년 ~ 현 재 숭실대학교 전자정보공학부 교수
관심분야 : 모바일 네트워킹, 엣지 클라우드 시스템, ICN 및 센서 네트워킹