Docker 환경에서 효율적인 자원 관리를 위한 Serverless Framework

한상욱*, 채민수*, 박희우**, 이화민***
*순천향대학교 컴퓨터학과
**순천향대학교 ICT융합재활공학과
***순천향대학교 컴퓨터소프트웨어공학과
e-mail:(sanguk, cms, leehm)@sch.ac.kr

Serverless Framework for Efficient Resource Management in Docker Environment[1)]

Sang-Wook Han*, Min-Su Chae*, Hee-Woo Park**, Hwa-Min Lee***
*Dept of Computer Science, Soonchunhyang University
**Dept of ICT Convergence Rehabilitation Engineering, Soonchunhyang University
***Dept of Computer Software Engineering, Soonchunhyang University

요 약

Docker에서 제공하는 API들은 컨테이너 엔진을 통하여 실행되어지게 된다. 따라서 컨테이너의 생성과 삭제 또는 정보에 대한 요청에 대해서 속도가 매우 느린 것이 현실이다. 많은 컨테이너에서 API 요청을 보내는 경우에는 부하가 더욱 심각하다. 따라서 본 연구에서는 Serverless 환경에서 컨테이너에서 발생하는 API 요청에 대한 부하를 줄이기 위해서 컨테이너 내부에 자원관리에 관련된 모듈을 적용 시킬 수 있는 방법과, Serverless 환경에서 여러 개의 컨테이너 또는 여러 대의 Docker 서버의 자원을 관리 할 수 있는 프레임워크를 제시한다.

1. 서론

최근 데이터센터는 클라우드의 가상화기법을 중심으로 서버뿐만 아니라 스토리지 및 네트워크 인프라까지 가상화를 하고 있다. 따라서 Xen, KVM 등과 같은 하이퍼바이저 기반의 가상머신 기술이 클라우드 인프라 구축에 주로 사용되어지고 있다. 하지만 Xen이나 KVM과 같은 하이퍼바이저 기반의 가상머신 생성 기술은 가상머신을 생성할 때 매우 높은 자원 사용량이 요구되며 할당된 자원을 사용하지 않을 경우에는 자원의 낭비를 가져오게 된다[1].

Docker를 이용한 Serverless 방식은 웹 서버와 같이 일시적인 서비스가 필요한 경우 매우 유용하며, 프로그램 개발자의 관점에서는 API등과 같은 개발 환경을 신경 쓰지 않고 로직에만 집중할 수 있다는 장점이 있다. 따라서 앞으로의 클라우드 환경은 Docker와 같이 자원 관리의 편의성을 제공하는 컨테이너 방식을 사용하여 사용자의 편의성을 더욱 높일 수 있을 것으로 기대된다.

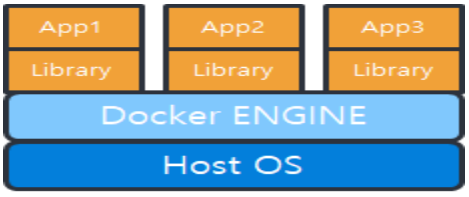
많은 사용자가 서비스를 요청 했을 경우, 컨테이너를 사용하는 방식과 기존의 하이퍼바이저를 사용하는 방식을 비교하였을 때 컨테이너 방식이 작업을 처리하는 효율성이 높다[2]. 따라서 클라우드 사용자의 작업이 빠르게 완료됨에 따라 효율적인 스케줄링을 지원할 수 있는 프레임워크가 매우 중요하다[3].

1) 이 논문은 2017년도 정부(과학기술정보통신부)의 재원으로 정보통신기술진흥센터의 대학ICT연구센터육성사업과 한국연구재단의 기초연구사업으로 수행되었음(IITP-2018-2014-1-00720, NRF-2017R1A2B4010570).

2. 관련 연구

2.1. Docker

Docker는 컨테이너 기반의 어플리케이션 자동화 오픈소스 프로젝트이다. Docker는 리눅스 커널에서 제공하는 가상화 기능을 사용하며 코드, 런타임, 시스템 라이브러리 등과 같이 실행에 필요한 파일들을 이미지로 구축하여 컨테이너를 생성한다. Docker는 가상 머신과 달리 별도의 운영체제가 요구 되지 않는다. Docker는 컨테이너의 커널을 공유하지만 Host 운영체제에서 유저스페이스로 격리되어 실행된다. 기존에는 리눅스에서만 사용이 가능했으나 윈도우10부터 Hyper-V 기능을 통해 Docker를 사용할 수 있다[4]. (그림 1)은 Docker의 구조이다.



(그림 1) Docker의 구조도

2.2. Serverless

기존에는 이벤트 중심 처리 기반 이였으나 최근 컨테이너 관리 및 소프트웨어 개발 전략으로 활용되고 있다. McGrath[5]가 구현한 Serverless Computing의 경우에는 요구되는 메모리량이 아닌 최대 메모리 크기로 제한하고 있으며, Docker API를 이용하여 처리 하고 있다. 자원 모니터링을 하기위해 API를 사용하는 경우, 하나의 컨테이너 정보를 얻기 위해 Docker 엔진에 API 요청을 한다. 이때, Docker 엔진이 구동중인 컨테이너 정보를 가져오기 때문에 성능 저하가 발생한다. 따라서 자원 효율이 낮아지는 문제가 있다.

일반 사용자에게 보다 편리한 사용자 인터페이스 환경을 제공하기 위해서는 현재 Windows 기반 사용자 인터페이스의 차원을 넘어서 사용자의 작업을 대행해 줄 수 있는 에이전트 시스템이 제공되어야 한다. 또한 에이전트 시스템 서비스의 확장과 사용보급화를 위한 응용이 가능한 미들웨어 플랫폼에 대한 연구개발이 이루어져야 한다.

3. Serverless Framework 설계

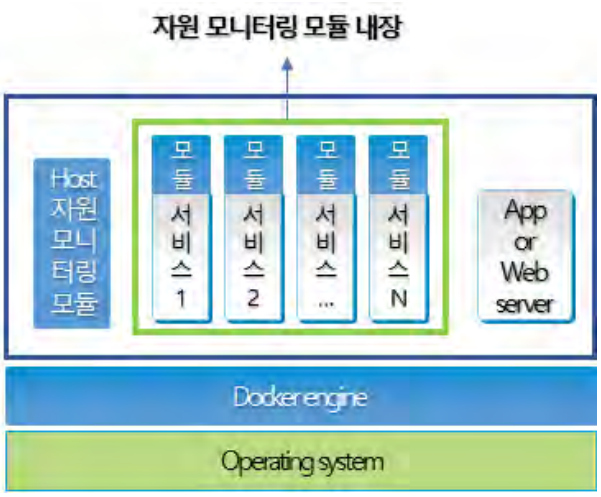
3.1. 제안하는 Docker 기반 Serverless Framework

현재 Docker 기반의 Serverless환경은 (그림 2)와 같이 구성되어 있다.

(그림 2) Docker 기반 Serverless 구조

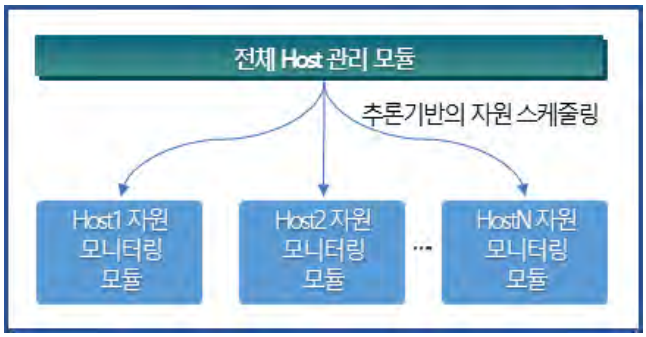
(그림 2)에서처럼 각각의 서비스를 제공하는 컨테이너들과 어플리케이션이나 웹을 지원하는 서버들로 구성되어 있다. 관리모듈 컨테이너에서 특정한 작업 수행을 위하여 서비스 컨테이너의 생성, 삭제, 자원정보 등의 정보를 요청하는 경우 Container engine에 API를 요청하게 되는데 이 때 부하가 발생하게 된다. 따라서 본 논문에서는 이를 해결 할 수 있는 프레임워크를 제안한다. (그림 3)은 제안하는 Docker 기반 Serverless 구조를 나타낸다.

각각의 서비스 컨테이너에 자원을 모니터링하는 모듈을 내재시키고 Host 자원관리 모니터링 모듈에서 Container engine에 API를 요청할 필요 없이 서비스 컨테이너에서 직접 컨테이너 정보를 받아오는 방식이다. 이러한 방식으로 구조를 변경하게 되면 여러 대의 Docker 서버를 관리하는 것에도 매우 효율적이다.



(그림 3) 제안하는 Docker 기반 Serverless 구조

제안하는 구조를 이용하여 여러 대의 Docker 서버들과 사용자 요구에 따른 자원 분배 스케줄링도 효율적으로 관리할 수 있다. (그림 4)는 제안하는 Serverless framwork를 나타낸다.



(그림 4) 제안하는 Serverless framwork

(그림 4)와 같이 제안하는 구조를 가진 여러 대의 Docker 서버를 관리하는 측면에서 기존의 성능 저하를 없애고 빠른 자원 추론과 분배가 가능한 전체 Host 모듈을 설계 할 수 있다. 각각의 모듈에 대한 자세한 설명은 3.2절부터 기술하였다.

3.2. Docker 자원 모니터링 모듈

Docker API를 이용하여 처리할 경우 성능 저하의 문제점이 존재하기 때문에 CPU 사용률과 메모리 사용량을 측정하는 모듈을 개발하여 성능 저하 문제를 해결해야 한다. 가장 먼저 컨테이너와 서버의 자원 모니터링 모듈이 필수적이다. 네트워크 문제로 발생하는 성능 저하를 줄이기 위하여 컨테이너에서 수집되는 정보를 호스트 서버의 모니터링 모듈에 전달한다. 서버의 모니터링 모듈은 서버의 자원 모니터링 정보와 서버 내 수행되는 컨테이너들의 자원 모니터링 정보를 전체 호스트 관리 모듈에 전달한다.

3.3. 리소스 추론 모듈

효율적인 자원 관리를 위해서 실행중인 서비스의 유형과 기존에 실행되었던 이미지를 바탕으로 리소스를 추론한다. 이를 위해 각 서비스 유형에 따른 리소스 사용량을 분석하고, 기존에 실행된 이미지를 바탕으로 성능을 분석하기 위해 컨테이너 자원 모니터링 모듈에 의해 수집된 정보를 사용한다.

<표 1> 리소스 추론 모듈 알고리즘

Designed Resource inference module algorithm

```
1.  function getResourceInference(type, image)
2.  {
3.     cpu = getEstimateCPUusageByServiceType(type)
4.     memory = getEstimateMemoryuageByServiceType(type)
5.     cpu2 = getEstimateCPUusageByMonitoring(image)
6.     memory2 = getEstimateMemoryusageByMonitoring(image)
7.     needCPU = (cpu + cpu2) / 2
8.     needMemory = (memory + memory2) / 2
9.     return needCPU, needMemory
10. }
```

<표 1>은 리소스 추론 모듈의 알고리즘이다. 리소스 추론 시 특정 값에 수렴하지 않도록 평균을 내어 사용한다.

3.4. 전체 호스트 관리 모듈

전체 호스트 관리 모듈은 자원 모니터링 모듈을 통해 얻어진 정보를 바탕으로 각 호스트 서버에 사용자가 요청한 자원을 할당한다. 특히, 리소스 추론 모듈을 통해 효율적으로 자원을 할당한다.

<표 2> 컨테이너 할당 알고리즘

Designed Container allocation algorithm

```
1.  function containerAllocation(hosts, type, image)
2.  {
3.     cpu, memory = getResourceInference(type, image)
4.     select = getUselessHost(hosts)
5.     for(host : hosts)
6.     {
7.        if(host.isAllocation(cpu, memory))
8.        {
9.           if(select.addUsage(cpu, memory) >
    host.addUsage(cpu, memory)
10.             select = host
11.       }
12.    }
13.    container = select.createContainer(image)
14.    return container
15. }
```

<표 2>는 컨테이너 할당 알고리즘이다. 리소스 추론 모듈을 통해 사용할 리소스를 예측한다. 추론된 결과를 바탕으로 할당 가능한 서버를 조회한다. 리소스를 할당할 때 자원 사용률이 가장 적은 서버를 선택하여 컨테이너를 생성한다.

4. 결론

본 논문에서는 Docker 환경에서 효율적인 자원 관리를 위한 Serverless framework를 설계하였다. 설계한 프레임워크는 다음과 같은 장점을 갖는다. 첫째, 컨테이너를 이용하기 때문에 개발자 관점에서는 로직에만 집중해서 개발할 수 있다. 둘째, 자원 모니터링의 경우 본 논문에서 제안하는 모듈을 사용할 경우에 성능 저하가 발생하지 않는다. 셋째, 리소스 추론을 통하여 효율적인 스케줄링과 자원 관리가 가능하다. 향후 설계한 프레임워크를 구현할 계획이다. 현재는 자원 사용률만을 고려한 리소스 할당 알고리즘만을 구현하였으나, 추후 개선된 리소스 할당 알고리즘을 설계할 계획이다.

참고문헌

[1] 배유미, 정성재, 소우영. (2014). 웹 서버 구성을 통한 가상머신과 컨테이너 방식 비교 분석. 한국정보통신학회논문지, 18(11), 2670-2677.

[2] Chae, M., Lee, H., & Lee, K. (2017). A performance comparison of linux containers and virtual machines using Docker and KVM. Cluster Computing, 1-11.

[3] Moon, Y., Yu, H., Gil, J. M., & Lim, J. (2017). A slave ants based ant colony optimization algorithm for task scheduling in cloud computing environments. Human-centric Computing and Information Sciences, 7(1), 28.

[4] Merkel, Dirk. "Docker: lightweight linux containers for consistent development and deployment." Linux Journal 2014.239 (2014): 2.

[5] McGrath, Garrett, and Paul R. Brenner. "Serverless computing: Design, implementation, and performance." Distributed Computing Systems Workshops (ICDCSW), 2017 IEEE 37th International Conference on. IEEE, 2017.