분산 컨테이너 클러스터 상의 JMS 메시지 처리 연구

조정근*, 최은미*,**[1]
국민대학교 *소프트웨어학부 / **금융정보보안학과
e-mail: {barunpuri, emchoi}@kookmin.ac.kr

JMS Message Processing on a Distributed Container Clustering

Jungkeun Cho*, Eunmi Choi*,**[1]
*College of Computer Science / **Dept. of Financial Information Security,
Kookmin University, South Korea

요 약

클라우드 가상화에서 최근 가장 많이 사용되는 컨테이너(container) 기술은 성능 향상과 이식성 및 확장성의 특징을 가지고 있다. 본 논문에서는 컨테이너 특성을 살펴보고, Docker 를 포함한 여러 컨테이너들을 비교해 본다. 본 연구의 시스템 아키텍처로, JMS(Java Message Service)기반의 분산 컨테이너 클러스터를 구성하였다. 분산 컨테이너 클러스터 상에서 메시지 처리 속도를 지표로 삼아 컨테이너 수와 Docker engine 의 메모리의 증가에 따른 성능을 비교 분석하였다.

1. 서론

클라우드 컴퓨팅은 자원을 공유하고 효율적으로 이용할 수 있는 장점을 가지고 있으므로, 자원 활용의 서비스와 Utility 모델의 효과로 보편화되고 있다. 클라우드 컴퓨팅의 자원의 효율적인 운영을 위한 핵심기술이 가상화(virtualization)부분이라고 할 수 있다. 최근 가상화 기술중에서도 가장 많이 사용되는 컨테이너(container) 기술은 성능 향상과 이식성 및 확장성의 특징을 가지고 있다.

본 논문에서는, 컨테이너의 특징과 다양한 컨테이너들의 종류를 살펴본다. 그리고 분산된 컨테이너의 클러스터 상에서 메시지 서버를 기반하여 서비스 클러스터링을 하는 구조를 소개한다. 이를 위하여 비동기식 메시징의 처리를 위한 JMS 를 활용하여 이식성이 있는 분산 컨테이너 클러스터를 구성하였다. 본 연구에서는, JMS 기반 컨테이너들 상에서 처리하는 성능을 비교하며, 컨테이너의 개수 증가에 따른 업무 처리 속도로 성능 지표로 세워서 실험 결과를 도출하였다. 이에 따라서, 메시지 서버를 미들웨어로 구성하는 컨테이너 기반 분산 시스템 상에서의 메시지 처리에 대하여 분석하였다. 컨테이너의 수에 따라서 성능 지표의 변화가 일어나는 것을 비교함으로써 분산 컨테이너 클러스터링의 업무 처리에 대한 특성을 살펴보았다.

2. 다양한 컨테이너들의 특성 비교

가상화 기술이 이식성(Portability)의 확대와 서비스의 확장 및 성능의 향상을 위하여 최근 다양한 컨테이너들이 활용이 되고 있다. 이 장에서는 기존의 가상머신과 차별된 컨테이너의 특성에 대하여 살펴보며, 다양한 컨테이너들을 조사 및 비교했다.

2.1 컨테이너의 특성

기존의 가상화 기술은 하드웨어와 운영체제 상위에서 계층화를 하여 실행 환경에 대한 가상화를 제공하는 것이다. 이러한 Virtual Machine (VM)은 물리적인 호스트 시스템 상에 하이퍼바이저를 설치한 후 그 위에 가상의 운영체제와 어플리케이션을 패키징 한 가상머신을 만들어 실행하는 방식이었다. 하드웨어 레벨 가상화는 VM 을 실행할 수 있는 VM 운영체제가 필요하여 통상 수 GB 단위의 크기를 가지고 있다. 이로 인하여, 여러 개의 VM 들을 한 호스트 상에서 동시에 구동하게 되면 발생하는 심각한 성능 저하 등의 문제를 가지고 있다.

(그림 1) 가상머신과 컨테이너의 구조[1]

이러한 기존의 가상화 기술의 단점을 대체하기 위

[1] 교신저자. 이 논문은 2019년도 BK21플러스(국민대학교) 사업에 의하여 지원되었음.

하여 컨테이너 기술이 사용이 되고 있다. 컨테이너 기술은 하드웨어 레벨이 아니라 운영체제 레벨 가상화로 운영체제를 제외하고 어플리케이션 실행에 필요한 파일들을 패키징 한다. 따라서 컨테이너에는 운영체제가 포함되지 않아 크기가 수십 MB 로 매우 작다. 또한 각 어플리케이션은 운영체제의 커널을 직접 공유하여 하드웨어 레벨 가상화보다 메모리를 훨씬 적게 차지하고 빠르게 작동한다. 다양한 접근법을 가지고, 클라우드 상의 가상화 운영[3]과 컨테이너 사용에 대한 성능 평가에 대한 선행 연구[2]가 진행되고 있다.

2.2 다양한 컨테이너들의 비교

이 절에서는 Linux 시스템에서 컨테이너 솔루션으로 사용된 LXC, 최근 가장 많이 사용되고 있는 도커, 그리고 어플리케이션 컨테이너에 최적화된 rkt 에 대하여 비교해 보았다.

LXC(Linux Containers)[4]는 단일 호스트에서 여러 개의 고립된 Linux 시스템을 실행하기위한 가상화 방법이다. LXC 는 가상 환경이 커널에서 공유되어 새로운 커널을 시작할 필요 없이 하나의 커널에서 다른 컨테이너를 관리할 수 있다. 가상화를 위한 하드웨어 에뮬레이트 작업 없이 분리된 공간을 만들기 때문에 오버 헤드가 줄어드는 장점을 가진다.

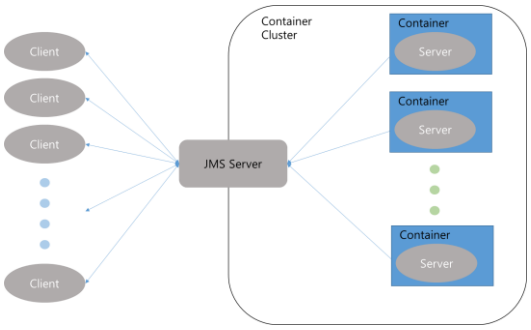
컨테이너 기술의 사실상 표준으로 알려진 도커(docker)[5]역시 초반에는 LXC 를 기반으로 하였다가 자체적인 libcontainer 기술을 사용하였고, 추후에는 runC 기술을 사용하였다. 다른 컨테이너 시스템과는 다르게 도커는 docker engine 을 사용하여 host OS 의 영향을 받지 않는다. 또한 도커는 컨테이너를 실행하기 위한 정보를 이미지 형식으로 저장하는데 이 이미지로 컨테이너를 생성할 수 있고 컨테이너의 상태가 바뀌거나 삭제되더라도 이미지는 변하지 않고 남아 있는다. 이미지는 Docker Hub 와 Docker Registry 로 간단하게 공유될 수 있으며, 도커는 클라우드 이미지, 클러스터링용 서버 등 다양한 기능을 제공하고 있다.

rkt[6]는 컨테이너 작업에 특화된 운영체제인 CoreOS 에 의해 제작된 앱 컨테이너를 구동하는데 최적화된 컨테이너 엔진이다. 다른 컨테이너 엔진에 비해 보안에 집중한 컨테이너 솔루션으로 rkt 로 수행한 컨테이너는 관리자 권한을 얻을 수 없다. 도커처럼 특정 데몬을 두지 않고 명령창에서 바로 init 시스템을 호출하여 컨테이너를 실행한다.

앞서서 살펴보았던 컨테이너들 중에, 도커는 도커 엔진을 사용하여 host OS 에 영향을 받지 않는 장점을 가지고 있다. 실험 환경 시 컨테이너를 실행하는데 있어서 운영체제의 영향을 받지 않는 도커를 본 연구에서는 사용하였다.

3. JMS 기반한 분산 컨테이너 상의 서버 클러스터

본 논문에서는 JMS(Java Message Service)를 기반으로 하는 분산된 컨테이너들을 연동하는 서버 클러스터를 구성하였다. 그림 2 와 같이 컨테이너 클러스터는 JMS 의 메시지 서버를 중심으로 게시/가입(Publisher/Subscriber)의 아키텍처 구조로 이루어져 있다. 일반적인 서버 클러스터링은 정적인 구성으로 Dispatcher 역할을 하는 Front-end Master 로 클러스터를 구성을 하지만, 컨테이너 클러스터는 이식성과 변경가능성 및 확장성을 최대화할 수 있도록 메시지 서버의 미들웨어(Message-Oriented Middleware)[7][8]로 구성을 하였다.



(그림 2) JMS 기반한 컨테이너 클러스터 구조

3.1 JMS 활용

JMS(Java Message Service)[9]는 안정적인 비동기식 메시징에 적용되는 일련의 규칙과 의미의 집합을 규정하며 메시지 구조, 프로그래밍 모델 및 API 를 정의한다. 여기서 비동기식 메시징은 메시지 사용자가 작업중이거나 오프라인인 경우에도, 시스템은 사용자가 온라인이 될 때 메시지 수신이 가능하도록 한다. 따라서 구성 요소의 상호 작용에 상당한 유연성의 제공되고 한 구성 요소가 실패하더라도 전체 구성 요소의 실패로 연결되지 않으므로 견고성이 더해진다.

JMS 는 두 가지 메시지 전달 모델인, 지점간(Point-to-Point) 모델을 구성할 수 있는 Queue 와 게시/가입(Publisher-Subscriber)모델을 구성할 수 있는 Topic 타입을 지원한다. 지점간 모델의 메시지는 Queue 로 전달된 다음, Queue 에 등록된 사용자 중 하나에게 한번에 하나씩 전달된다. 각 메시지는 반드시 한 사용자에게만 전달되며 사용자가 없는 경우에는 Queue 에 메시지를 보관했다가 사용자가 등록되면 메시지를 전달한다. 이런 모델은 Request 에 대한 서비스를 진행하는 서버 모델 상황에서 주로 사용이 된다.

한편, 게시/가입 모델은 사용자 수의 제한 없이 메시지를 전달한다. 보내고자 하는 Topic 으로 메시지가 전달되어 Topic 에 가입한 모든 서버들에게 전달된다. 이러한 모델은 하나의 일어난 사건에 대하여 가입된 다른 서버들이 처리하거나 통보를 받도록 하는 상황에서 주로 사용이 된다.

게시/가입 모델을 사용하여 실험을 진행하게 되면 클라이언트의 요청을 서버에서 적절한 시간 내에 받지 못할 경우나 하나의 요청에 대하여 중복된 응답이 발생할 수도 있다. 하지만 지점간 모델을 사용하게 되면 요청 메시지가 반드시 하나의 사용자에게

전달이 되기 때문에 모든 메시지에 대하여 응답을 확인할 수 있고, 하나의 응답만이 도착하기 때문에 성능을 측정하기 적합하여 지점간 모델을 사용하여 실험을 진행하였다.

4. 분산 컨테이너 클러스터 성능 실험 결과

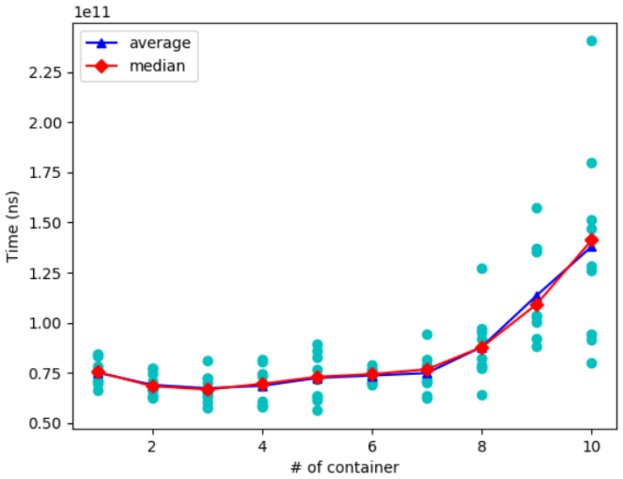
4.1 실험 환경

본 실험은 Windows10 Enterprise(16 비트) 운영체제에 16GB DDR4 메모리와 Intel i7-8700(12 CPUs) 프로세서, 그리고 GeForce gtx 1050 GPU 환경에서 진행되었다.

클라이언트에서 생성하는 부하(load)는 텍스트 메시지 형식으로 보내도록 구성이 되었다. 서버에서는 메시지 요청의 타입에 따라서 클라이언트에게 텍스트 메시지로 응답을 주거나, 클라이언트에서 요청한 파일을 읽어서 전송해 준다. 이에 따라서 생성되는 부하는, 텍스트 메시지: 10KB 파일 요청: 1M 파일 요청 = 2:1:1 정도의 비율을 가지고 구성을 하였다.

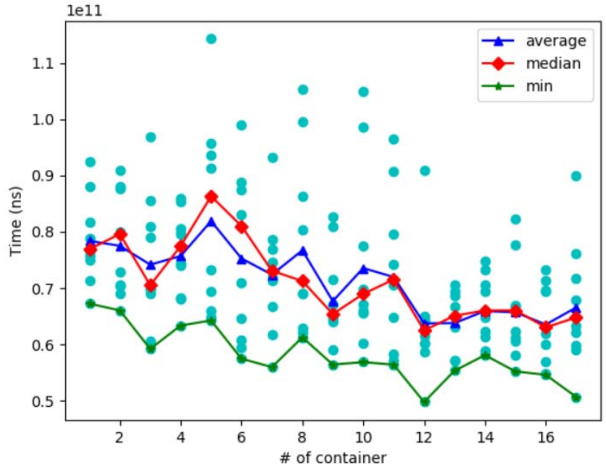
4.2 컨테이너 증가에 따른 메시지 처리 속도 비교

본 논문에서는 메시지 처리 속도의 지표를 클라이언트의 요청에 대한 서버의 응답 시간인 RTT(Round Trip Time), 즉 Turnaround Time 으로 세워서 측정을 하였다. 컨테이너의 수 증가에 따른 메시지 처리 속도를 RTT 로 비교하였다. 클라이언트에서 서버로 보내는 메시지는 간단한 요청의 텍스트 형식이며, 서버는 요청에 따른 character 형식과 byte 형식의 message 를 보낸다. 요청을 보내고 응답을 받을 때까지를 시간을 측정하여 nano second 단위에서 측정을 하였다. 테스트 셋은 클라이언트에서 100 개의 request 메시지를 보내도록 구성하였으며, 각 경우의 수마다 10 회를 수행하여 결과값을 비교 분석하였다.



(그림 3) Docker Engine 의 메모리가 2048MB 일때 컨테이너의 수에 따른 수행 시간 (RTT)

그림 3 은 도커 엔진의 메모리가 2GB 일 때 컨테이너 수가 증가함에 따른 RTT 수행 시간의 변화를 나타내고 있다. 실험에서의 평균값과 중간 값으로 컨테이너 개수에 따른 처리 속도의 추이를 보여준다. 그림 3 의 그래프에서 3-7 개정도의 컨테이너까지는 도커 엔진에서 처리하는데 문제가 없지만, 8 개이상의 컨테이너로 늘어나게 되면 도커 엔진에서 컨테이너를 관리하는데 소모되는 자원이 많아진다. 이에 따라서 RTT 속도가 점차적으로 늘어나는 것을 볼 수 있다.



(그림 4) Docker Engine 의 메모리가 8156MB 일때 컨테이너의 수에 따른 수행 시간 (RTT)

그림 4 는 도커 엔진의 메모리가 8GB 일 때 컨테이너 수에 따른 수행 시간의 변화를 나타내고 있다. 실험치들의 평균값, 중간 값, 최소값을 컨테이너의 증가에 따른 추이로 보여준다. CPU 의 수 보다 더 많은 컨테이너를 동시에 구동하여도, 도커 엔진에서 컨테이너들을 할당된 메모리 환경에서 효율적으로 관리를 할 수 있게 된다는 것을 볼 수 있다. 그러므로, 성능의 손실 없이 작업을 수행하는 것을 관찰하였다.

5. 결론

이 논문에서는 클라우드 가상화 기술인 컨테이너 시스템을 살펴보고 몇몇 컨테이너 시스템들이 어떠한 특성을 가지는지 살펴보았다. JMS 를 미들웨어로 하는 분산 컨테이너 클러스터를 구성하여 컨테이너를 실행하는 도커 엔진의 메모리와 컨테이너의 수를 늘려가며 성능을 비교해 보았다. 그 결과 도커 엔진의 메모리가 충분하지 않을 때에는 컨테이너가 늘어날수록 성능의 증가가 확연하지 않고 엔진의 한계에 다다르면 성능이 매우 떨어졌다. 반면에 도커 엔진의 메모리가 충분히 크면 성능의 손실 없이 컨테이너가 증가할수록 시간이 줄어드는 것을 확인할 수 있다.

본 실험에서는 클라이언트의 request 들이 클러스터로 이루어진 컨테이너들의 처리 속도에 비하여 상대적으로 적으므로, speedup 의 추이가 크게 나타나지 않는 것을 볼 수 있다. 이에 향후 연구는, 분산 시스템 환경에서 클러스터링된 컨테이너들 상에서의 성능과 다양한 클라이언트의 요청의 벤치마크에 대한 처리 결과 분석으로 볼 수 있다.

이 논문은 2019년도 BK21플러스(국민대학교) 사업에 의하여 지원되었음.

참고문헌

[1] Google Cloud, 컨테이너의 개념과 이점, https://cloud.google.com/containers/?hl=ko
[2] 황정연, 류호용. “KVM(Kernal Virtual Machine)과 Docker 간의 성능비교 및 예측 분석”, in Journal of KIIT. Vol.13, No.11, pp.127-136, Nov 2015
[3] Mohan Krushna Varma Nandimandalam, Eunmi Choi, "Comparative Study of Various Platform as a Service Frameworks", The International Journal on Cloud Computing: Services and Architecture (IJCCSA), Vol. 6. No. 1, pp23-34, February 2016.
[4] Linux Containers, Introduction of LXC, https://linuxcontainers.org/ko/lxc/documentation/
[5] Docker, Docker Docs, https://docs.docker.com/
[6] CoreOS, rkt, https://coreos.com/rkt/
[7] Sun Microsystems,Inc, Sun Java System Message Queue3 Technical Overview, https://docs.oracle.com/cd/E19435-01/819-2222/concepts.html#wp221545
[8] Guangxxuan Chen, Yanhui Du, Panke Qin and Lei Zhang. “Research of JMS Based Message Oriented Middleware for Cluster”, in 2013 Fifth International Conference on Computational and Information Sciences, Jun 2013
[9] Michael Pang and Piyush Maheshwari. “Benchmarking Message-Oriented Middleware – TIB/RV vs. SonicMQ”, in Concurrency: Practice and Experience, Vol.17 Issue: 12 p1507-1526, 20, August 2015