Docker Swarm 의 메모리 과점유로 인한 성능 하락의 상관 관계 분석†,‡

장준범, 박봉우, 최희석, 유헌창
고려대학교 컴퓨터전파통신공학과
e-mail : {jb0726, bongwoo, hsrangken, yuhc}@korea.ac.kr

Correlation Analysis of Performance Degradation Due to Heavy Occupancy of Memory in Docker Swarm

JuneBeom Jang, Bongwoo Bak, HeeSeok Choi, Heonchang Yu
Dept. of Radio Communication, Korea University

요 약

소프트웨어 애플리케이션의 가상화를 지원하는 컨테이너(container)는 일반적인 가상머신(VM; virtual machine)과 같은 운영체제의 격리된 인스턴스 형태이지만 VM 과는 달리 호스트 OS 자원을 공유하여 자원을 효율적으로 사용할 수 있고 이식성이 좋으며 배포를 쉽게 할 수 있는 등 장점이 있다. 컨테이너의 중요성과 활용도가 높아지면서 그것을 관리하고 통제하는 오케스트레이션 솔루션도 각광을 받고 있다. 본 논문에서는 Docker 에 내장된 오케스트레이션 기능 중 하나인 Docker Swarm 이 과도하게 메모리를 사용하는 문제점을 해결하고자 한다. 먼저, Docker Swarm 의 구조에서 Manager 노드와 Worker 노드의 서비스를 증가시켜 실행시킨 후 성능을 평가한 후 과점유의 원인을 파악한다. 실험 결과 메모리 과점유의 원인은 컨테이너가 작동을 멈춘 후에도 여전히 메모리를 점유하고 있어 컨테이너를 증가시킬수록 메모리 이용률이 줄어들지 않는 것이 증명되었다.

1. 서론

최근 컴퓨팅 환경은 가상화를 기반으로 한 IaaS 중심의 클라우드 서비스 이외에, 유저들이 애플리케이션을 개발, 실행, 관리할 수 있게 하는 플랫폼 제공을 목적으로 하는 PaaS 서비스도 활성화되어 있다. 하지만 VM 은 소프트웨어로 구현된 하드웨어 위에 OS 를 설치하고 그 위에 소프트웨어를 설치함으로써 시스템이 무거워지고 느려진다는 단점이 있다. 이러한 단점을 극복하기 위해 반가상화 기술인 Xen 이 등장하였지만 성능 문제는 해결되지 않았다. 문제 해결을 위해 프로세스를 단순히 단독으로 격리시키는 컨테이너 기술의 등장으로 일부 성능 문제를 해소하게 되었다 [1].

이동이나 확장이 용이하여 유연한 컨테이너 기술의 장점을 극대화 시키기 위해 Fleet, Mesos, Kubernetes, EC2 컨테이너 Service, Docker Swarm 등 여러가지 컨테이너 †오케스트레이션 솔루션이 개발되었다. 이러한 솔루션은 스케줄링, 클러스터링, 서비스 디스커버리, 로깅, 모니터링 등의 작업을 제공한다. 본 논문에서 Docker Swarm 은 [2]의 클러스터 환경에서 클러스터를 이루는 노드 중 일부에 문제가 발생했을 때 서비스에 지장을 주지 않으며 클러스터가 제어 가능한 상태에 있어야 하는 가용성 클러스터를 고려한다. Swarm 은 관리 기능인 Manager 노드와 컨테이너를 실행하는 Worker 노드의 구조로 이루어져 있다. 하지만 Swarm 클러스터는 필요 이상의 메모리를 사용하는 문제점이 있다. 따라서 본 연구에서는 다양한 실험을 통해 Swarm 클러스터의 CPU, 메모리 사용량을 확인하고 과도한 메모리 사용의 원인을 찾아 오케스트레이션 서비스의 최적화를 목적으로 한다.

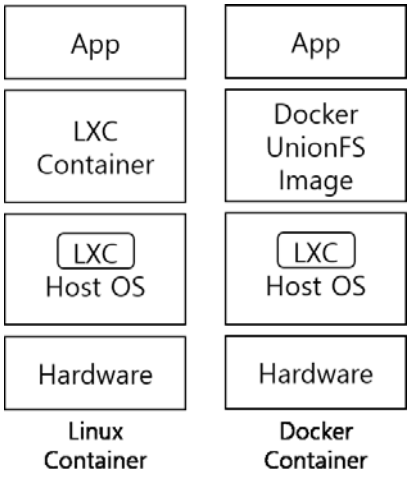
2. 관련 연구

Docker 는 자체 컨테이너 이미지를 관리할 수 있는 기능을 제공하는 데몬이다. (그림 1)은 Linux 컨테이너와 Docker 컨테이너 구조의 차이를 보여준다. Docker 컨테이너는 구현을 위해 기존 Linux 컨테이너(LXC)를 사용하고 이미지관리 및 유니온 파일 시스템 기능을 추가했다.

Docker Swarm 의 클러스터는 Raft Consensus Algorithm 을 사용한다. Raft Consensus Algorithm 은 여러 노드들 중 리더를 선출한 후, 리더에게 복제된 로그에 대한 관리 책임을 완전하게 부여함으로써 합의를 구현한다. 또한 리더는 클라이언트의 로그를 받아 클러스터를 통해 복제하여 서로의 로그가 일치되도록 해야 한다. 만약 리더 노드에 문제가 생겨 더

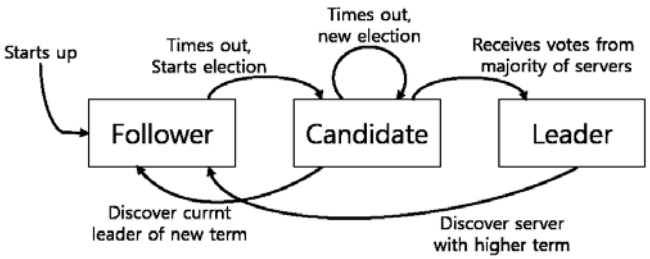
†본 연구는 과학기술정보통신부 및 정보통신기술진흥센터의 대학 ICT 연구센터 지원사업의 연구결과로 수행되었음 (IITP-2018-0-001405)
‡이 논문은 2018 년도 정부(과학기술정보통신부)의 재원으로 정보통신기술진흥센터의 지원을 받아 수행된 연구임 (No.2018-0-00480)

이상 제역할을 수행하지 못하면 다른 노드가 새로운 리더가 된다. 그로 인하여 전체 Manager 노드의 과반 이상이 유효할 경우에 서비스를 중단하지 않고 가동시킬 수 있는 결함 포용이 가능하다.



(그림 1) Linux 컨테이너와 Docker 컨테이너

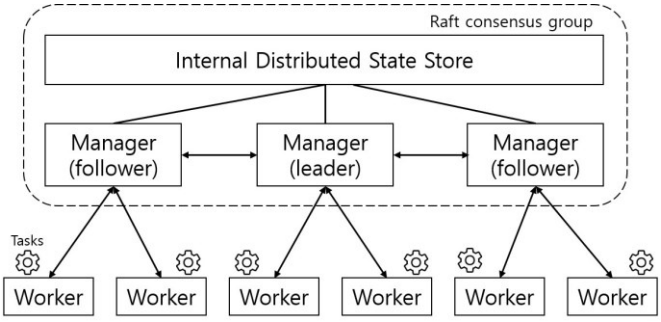
(그림 2)는 새로운 리더를 선출하는 과정이다. 팔로워(follower)가 리더에게 연락을 받지 못하면 후보자가 되어 선거를 시작한다. 전체 클러스터에서 제일 많은 투표를 받은 후보가 새로운 리더가 된다. 또한 Worker 노드를 Manager 노드로 승격시켜 사용할 수도 있다. 이처럼 Manager 노드에 문제가 발생해도 다른 노드를 승격시키거나 새로운 리더를 선출하여 서비스의 고가용성을 유지할 수 있다[3].



(그림 2) 리더 선출 과정

3. 시스템 환경

본 연구에서 기반하는 Docker Swam 시스템 환경은 (그림 3)과 같다. Docker Swarm 은 Worker 관점에서 하나의 서비스를 다수의 작업으로 분해하여 작업의 양을 조절해 주는 부하 분산의 역할을 한다. 하지만 부하 분산 기능을 안정적으로 실행하기 위해 클러스터 구성이 필요하다. 이를 위해 Docker Swarm 은 Manager Pool 을 운용한다. Manager Pool 은 여러 개의 Manager 를 클러스터 안에 둘 수 있다. Manager 로 지정된 노드들은 서로의 상태 정보를 실시간으로 동기화하며 컨테이너 스케줄링을 담당한다. 이것을 위한 다른 저장공간은 필요하지 않아서 클러스터의 구성이 단순해진다. 하지만 Manager 의 수가 많아지면 상태정보 동기화에 오버헤드가 생겨 실험결과에 왜곡이 발생할 수 있으므로 2 개의 Manager 노드와 5 개의 Worker 노드를 사용한다.

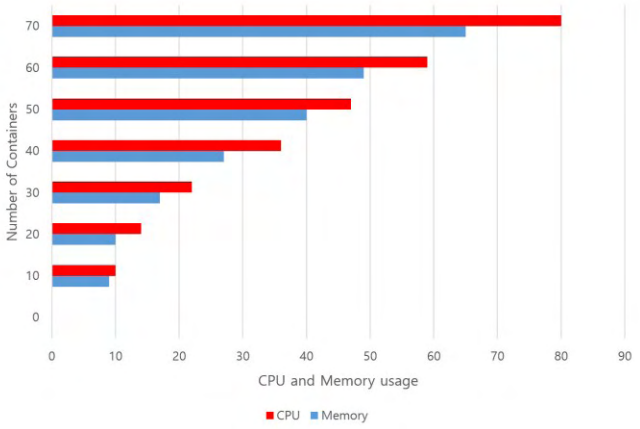


(그림 3) Docker Swarm 구조

4. 실험 및 분석

본 연구에서는 Worker 노드 5 개에 서비스를 배포하고 다양한 조건에서의 CPU, 메모리 사용률을 지속적으로 모니터링한다. 모니터링한 결과를 토대로 임의적으로 설정한 조건과 CPU, 메모리 이용률의 상관 관계를 파악하여 메모리 과점유의 원인을 찾는다.

(그림 4)는 컨테이너를 70 개 배포했을 때 CPU, 메모리 이용률을 모니터링한 것이다. 컨테이너를 증가시킬수록 CPU 와 메모리 이용률도 함께 증가한다. 이는 컨테이너가 증가할수록 서비스에서 파생되는 작업이 많아져 CPU 이용률이 증가하며 컨테이너마다 메모리가 할당되기 때문에 동시에 이용률이 증가한다. 이것의 데이터를 기준으로 조건을 변경하며 실험을 진행한다.

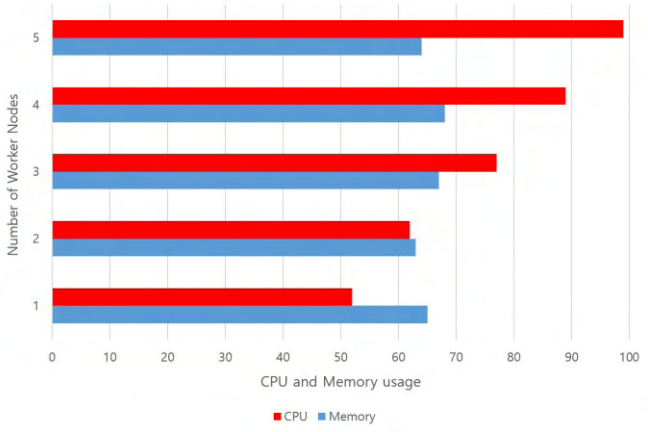


(그림 4) 컨테이너 개수에 대한 CPU, 메모리 이용률

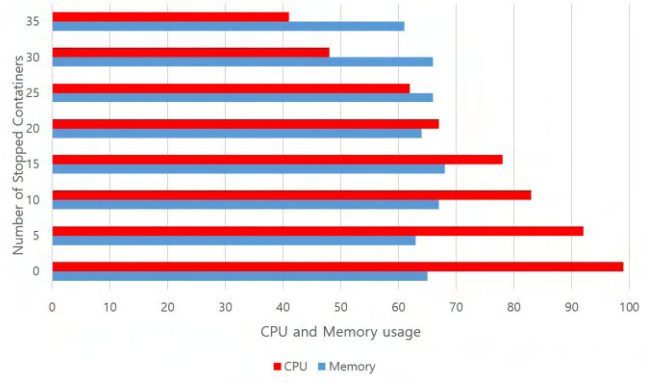
(그림 5)는 Swarm 전체 노드 개수에 대한 CPU, 메모리 이용률을 모니터링한 것이다. 처음에는 Manager 노드 2 개, Worker 노드 1 개로 시작하여 Worker 노드만 하나씩 확장시켜 총 5 개까지 확장시킨다. 또한 컨테이너는 처음부터 70 개를 배포한다. 노드가 증가할수록 CPU 이용률은 100 퍼센트에 가까워지는 것을 볼 수 있다. 하지만 메모리 이용률엔 거의 변화가 없으므로 노드의 개수와 메모리 이용률엔 상관 관계가 없는 것으로 파악된다.

(그림 6)은 작동하지 않는 컨테이너에 대한 CPU, 메모리 이용률을 모니터링한 것이다. 이것 또한 작동중인 컨테이너 35 개, 작동하지 않는 컨테이너 35 개의 개수를 합한 70 개로 실험을 진행한다. 실험 결과, 작동하지 않는 컨테이너의 수가 증가할수록 CPU 이용률은 감소하지만 메모리 이용률은 거의 변동이 없는 것으로 확인되었다. (그림 7)은 작동하지 않는 컨테이너를 삭제한 후의 CPU, 메모리 이용률이다. 지우는 삭제한 늘어날수록 메모리 이용률이 줄어드는 것을 볼 수 있다.

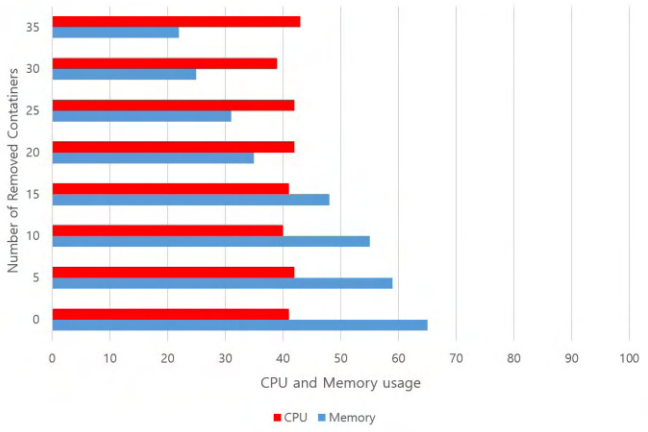
(그림 6)과 (그림 7)의 결과를 토대로 컨테이너의 상태보다 컨테이너의 개수가 메모리와 상관관계가 있는 것을 발견했다. Docker Swarm 은 작업이 끝난 컨테이너의 메모리를 반환하지 않아 컨테이너를 증가시킬수록 이전의 컨테이너 메모리 이용률에 현재의 컨테이너 메모리 이용률이 더해져서 메모리를 효율적으로 사용하지 못하게 된다.



(그림 5) 노드 개수에 대한 CPU, 메모리 이용률



(그림 6) 작동하지 않는 컨테이너에 대한 CPU, 메모리 이용률



(그림 7) 컨테이너 삭제 후 CPU, 메모리 이용률

5. 결론

Docker 의 효율적인 컨테이너의 관리와 통제를 위한 오케스트레이션 기능인 Docker swarm 은 메모리 점유에 관한 문제가 있으며 본 논문은 그것을 실험에 의하여 원인을 증명하였다. 원인은 컨테이너가 실행 또는 실행 중지 상태에 상관없이 컨테이너의 메모리 이용률에는 영향을 미치지 않는 것이다. 오로지 메모리 이용률에 영향을 주는 요소는 컨테이너의 개수라는 것이 밝혀졌다. Docker swarm 은 자동으로 컨테이너를 삭제하는 기능이 존재하지 않으므로 현재로서는 사용하지 않는 컨테이너를 수동으로 삭제하여 메모리 확보를 하거나 물리적으로 메모리 용량을 늘리는 것으로 메모리 과점유를 막을 수 있다. 소규모의 컨테이너를 사용하는 경우 수동으로 유저가 컨테이너를 삭제하는 것이 가능하지만 만약 대규모의 컨테이너를 사용하면 유저가 사용이 끝난 컨테이너를 지우는 것은 매우 비효율적일 것이다.

참고문헌

[1] DUA, Rajdeep; RAJA, A. Reddy; KAKADIA, Dharmesh. Virtualization vs 컨테이너 ization to support paas. In: *Cloud Engineering (IC2E), 2014 IEEE International Conference on*. IEEE, 2014. p. 610-614.

[2] NAIK, Nitin. Building a virtual system of systems using Docker Swarm in multiple clouds. In: *Systems Engineering (ISSE), 2016 IEEE International Symposium on*. IEEE, 2016. p. 1-3.

[3] ONGARO, Diego; OUSTERHOUT, John K. In search of an understandable consensus algorithm. In: *USENIX Annual Technical Conference*. 2014. p. 305-319.