Docker Swarm 기반 리소스 부하 영역별 컨테이너 처리 성능 평가

강태신, 이재학, 김형준, 유헌창
고려대학교 대학원 컴퓨터학과
{teri98, smreodmlvl, ledzep0830, yuhc}@korea.ac.kr

Evaluation of Container Handling Performance by Docker Swarm-based Resource Stress Area

Taeshin Kang, Jaehak Lee, Hyungjun Kim and Heonchang Yu
Dept. of Computer Science and Engineering, Korea University

요 약

클라우드 수요의 지속적인 증가로 인해 효율적인 자원 관리를 수행하는 분산 클라우드 서비스가 주목받고 있으며, 컨테이너 가상화에서 여러 노드에 컨테이너 배치 및 관리를 수행하는 오케스트레이터의 컨테이너 스케줄링 기법에 대한 연구가 진행되고 있다. 하지만 클러스터 집합의 컴퓨팅 자원이 부족할 시 프로세스의 저지연 처리를 보장하기 위한 연구는 활발히 수행되고 있지 않다. 본 연구에서는 Docker Swarm 기반 제한된 컴퓨팅 자원을 가진 클러스터 환경에서 다양한 부하가 발생했을 때 일반 프로세스의 평균 수행 시간 및 평균 지연 처리 정도를 측정 및 분석한다. 이를 통해 오케스트레이터 스케줄링 최적화를 위한 연구 방향성을 제시하며, 향후 관련 연구의 활성화에 기여할 것으로 기대된다.

1. 서론

클라우드 컴퓨팅은 통신망, 서버, 저장장치 등의 컴퓨팅 자원에 언제 어디서나 네트워크를 통해 접근하는 기능을 제공하는 모델이다[1]. 클라우드는 높은 탄력성과 고가용성으로 인해 많은 분야에서 활용되고 있는데, 특히 인공지능 기술 등 첨단 IT 기술 기반의 자율주행시대가 도래하면서 카-클라우드(Car-to-Cloud) 서비스가 초점화되고 있다[2-3]. 그러나 차량들로부터 발생되는 대용량 데이터(교통정보, 신호 데이터, 사고정보 등) 처리 시 CPU, 메모리 등 하드웨어 과부하로 인해 자원이 부족하게 되면 카-클라우드 서비스의 Quality of Service (QoS)가 저하될 수 있다. 특히 실시간성 및 가용성을 보장해야 하는 카-클라우드 서비스는 차량 컴퓨팅 장치의 자원이 비교적 열악하기 때문에 이러한 문제점이 더욱 부각될 수 있다.

이러한 문제를 해결하기 위해 연구가 활발하게 이루어지고 있다. Marco Malinverno 는 모바일 에지 컴퓨팅(MEC)을 활용하여 통신 프로세스에 의해 발생하는 지연 시간을 억제했다[4]. 하지만 컨테이너 환경에서 자원 결핍이 발생했을 때 프로세스의 처리속도 지연을 단축시키기 위한 많은 과제가 아직 남아 있다.

본 연구는 Docker Swarm 기반 클러스터 환경에서의 워크로드 유형별 프로세스 실행시간 비교를 실험을 통해 분석한다. CPU, 메모리, 스토리지 부하가 없는 환경에서 프로세스 집합에 대한 수행시간과 워크로드 유형별 부하 프로세스가 포함된 프로세스 집합을 수행했을 때의 실행시간을 비교함으로써 유형별 프로세스 처리 속도 저하 정도를 비교한다. 그리고 이를 통해 오케스트레이터 스케줄링 최적화를 수행할 시 어떤 하드웨어 영역의 자원 경합을 중점적으로 개선해야 하는지 제시한다.

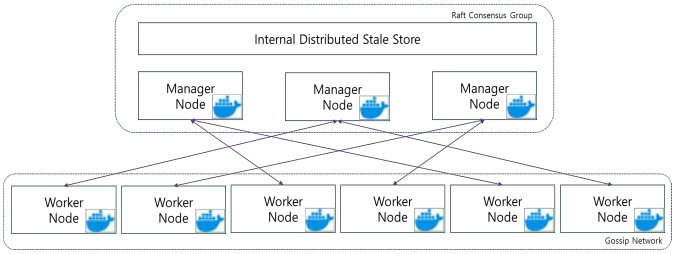
본 논문은 먼저 2 장에서 컨테이너 오케스트레이션 개념에 대해 정의하고 3 장에서 실험 수행 환경 및 테스트 수행 방법에 관해 기술한다. 4 장에서는 Docker Swarm 에서 일반적인 상황과 각 워크로드 유형에서의 일반 프로세스에 대한 실행 성능을 평가한다. 마지막 5 장에서 해당 실험에 대한 의의와 향후 연구 방향을 기술한다.

2. 컨테이너 오케스트레이션

오케스트레이션(orchestration)은 다중 컴퓨터 시스템을 소프트웨어를 통해 자동화하여 구성, 조정 및 관리를 수행하는 것을 말한다. 클라우드 또는 응용서비스 공급자는 컨테이너 오케스트레이션을 통해 다중 컨테이너 패키징 된 애플리케이션의 구성 선택, 배포 및 제어 방법 등을 정의할 수 있다[5]. 이러한 오케스트레이션을 수행하는 소프트웨어가 오케스트레이터이다. 즉, 컨테이너 오케스트레이터는 컨테이너의 리소스 제한, 로드 밸런싱, 오토 스케일링 등의 기능을 제공하는 소프트웨어 도구이다. 컨테이너 오케스트레이

터에는 가장 범용적으로 사용되고 있는 Kubernetes 와 본 논문의 실험에서 사용한 Docker Swarm 등이 있다.

Docker Swarm 은 Docker 엔진에 내재된 기본 클러스터 관리 도구이다. Docker 는 컨테이너 기반 가상화 플랫폼으로 프로세스를 격리함으로써 컨테이너를 실행하는 로직을 수행하는 소프트웨어이다. Swarm 은 Docker 에 포함되어 있기 때문에 Docker CLI 를 사용하여 Swarm 을 생성하고, Swarm 에 애플리케이션 서비스 배포 및 동작 관리를 수행할 수 있다[6].



(그림 1) Docker Swarm 서비스 구성요소

그림 1 은 Docker Swarm 의 서비스 구성 요소를 나타낸 것이다. 노드는 관리자 노드(Manager Node)와 작업자 노드(Worker Node) 2 가지가 존재한다. 관리자 노드는 작업자 노드 집합을 관리하는 역할을 수행한다. 여러 개의 관리자 노드가 존재할 수 있으며, 합의 알고리즘을 통해 서비스 상태를 일관되게 유지한다. 또한 내장형 내부 분산 상태 저장소를 사용하여 외부 저장소를 요구하지 않는다. 작업자 노드는 Docker 엔진의 인스턴스로서 컨테이너의 실행만을 수행한다.

3. 실험환경 및 수행방법

본 실험에서는 1 개의 관리자 노드와 4 개의 작업자 노드로 클러스터 환경을 구성한다. 두 노드 유형 모두 Virtual Box 기반 가상머신으로 구성하였다. 표 1 과 표 2 는 실험 환경을 나타낸다.

<표 1> 실험 시스템 하드웨어 환경

CPU	Intel i7-9700 8-Core Processor 3.00GHz
RAM	DDR4-3200 16GB x 1, DDR4-2400 8GB x 1
STORAGE	Samsung SSD 970 EVO Plus 500GB
HOST OS	Microsoft Windows 11 Education

<표 2> Docker Swarm Cluster Node 환경

Type of Node	Manager Node	Worker Node
Number of Nodes	1	4
Number of CPU Cores	4	2
RAM Capacity	2048 MB	1536 MB
Storage Capacity	50GB	30GB
Guest OS	Ubuntu 18.04.4 LTS	

본 실험에는 2 가지의 벤치마크 프로그램을 통해 성능 평가를 수행한다. 첫 번째 프로그램은 자체 개발한 벤치마크 프로그램이며, 본 논문에서 일반 프로그램으로 정의한다. 이 프로그램은 -1,000,000 이상 1,000,000 이하의 값을 임의로 선정하고, 이들을 num 변수에 더하는 과정을 1,000,000 번 반복 수행하여 최종 결과 값을 출력한다. 두 번째는 Linux 의 stress 를 이용하여 하드웨어 유형별로 집약적인 워크로드를 발생시키는 프로그램이다[7].

본 실험에서는 크게 2 가지의 테스트 케이스로 나눌 수 있다. 첫번째는 워크로드 집약이 없는 상태로 일반 프로그램만을 수행하는 테스트 케이스이다. 두 번째는 자원 집약적인 워크로드가 발생하는 프로그램과 일반 프로그램이 동시에 실행되는 테스트 케이스이다. 표 3 은 테스트 케이스 별 두 컨테이너 프로그램의 요구 개수를 나타낸다.

<표 3> 테스트 케이스 별 컨테이너 요구 개수

Type of Testcase	General	Resource Stress
General Program	50	40
Hard-resource Program	0	10

실험 방법에 대하여, 먼저 하드웨어 워크로드가 없는 상태에서의 컨테이너 처리 속도를 측정한다. 관리자 노드는 자신과 4 개의 작업자 노드에게 컨테이너 50 개를 60 초 동안 배포한다. 각 노드는 평균적으로 10 개의 컨테이너를 처리하게 될 것이다. 실험 결과 분석을 위해 각 노드의 컨테이너 처리 속도의 평균을 구한다.

다음은 노드에 하드웨어 집약적인 워크로드가 존재할 때 일반 컨테이너의 처리 속도를 측정한다. 테스트는 CPU, 메모리, 스토리지 세 영역에서 각각 적은 부하, 중간 부하, 많은 부하 유형으로 3 번 측정한다. 표 4 는 하드웨어 부하가 존재할 때 영역별 stress 에 사용된 파라미터를 나타낸 것이다. 각 프로세스의 Timeout 은 10 초로 동일하게 설정한다.

<표 4> 테스트 영역별 stress 파라미터

	Number of Process	Number of Bytes
CPU	1, 5, 10	-
Memory	1	256, 512, 1024MB
Storage	1	512, 1024, 2048MB

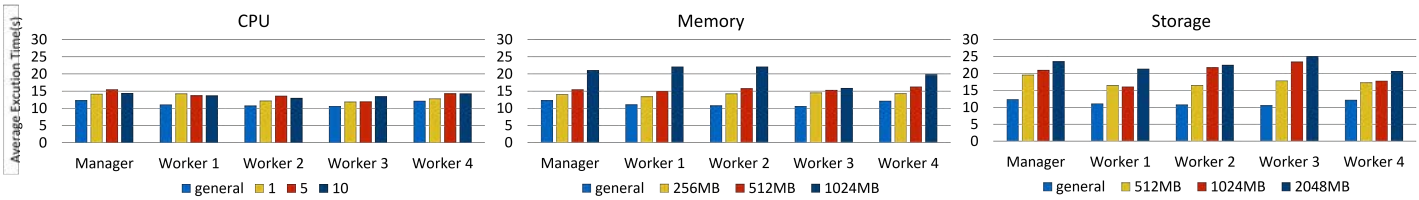
관리자 노드는 일반 프로그램 컨테이너 40 개와 중량 리소스 프로그램 컨테이너 10 개를 자신과 4 개의 작업자 노드에게 60 초 동안 배포한다. 각 노드의 컨테이너 처리 속도 측정은 부하가 없는 상태에서의 처리 속도 측정 방법과 동일한 방식을 사용한다.

4. 실험 결과 분석 평가

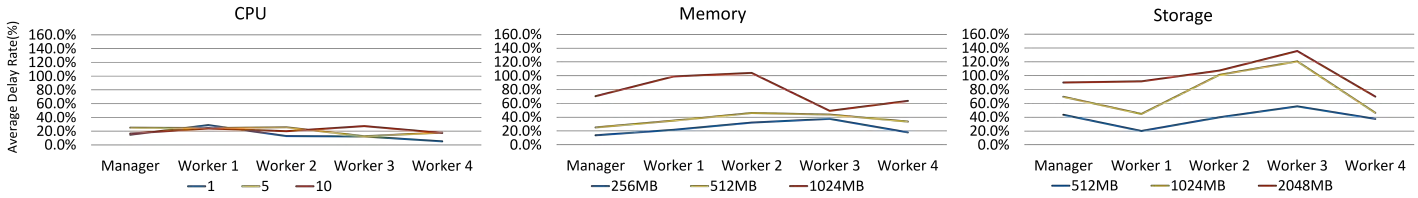
그림 2 는 Docker Swarm 에서 일반 및 리소스 부하 환경에서 각 노드의 일반 컨테이너 평균 처리 시간을 보여준다. 그림 3 은 일반 환경에서의 평균 처리 시간을 기준으로 각테스트 케이스의 노드별 일반 프로세스 평균 실행 지연율을 나타낸 것이다.

CPU 부분에서 단위 부하 프로세스 당 1 개의 sqrt() 연산 프로세스가 생성되었을 때 평균 14.8%의 성능 저하가 발생했으며, 5 개와 10 개가 생성되었을 때에는 약 21%의 프로세스 처리 지연이 발생했다. 5 개와 10 개가 생성되었을 때에는 약 21%의 프로세스 처리 지연이 발생했다. 5 개와 10 개의 성능차이가 거의 나타나지 않았는데, 이는 10 초의 timeout 내에 처리할 수 있는 프로세스의 개수를 초과한 것으로 추정된다.

메모리 부분에서 단위 부하 프로세스에 256MB 가 할당되는 경우 평균 약 25% 프로세스 처리 속도가 감소하였다. 512MB 와 1024MB 에서는 평균적으로 약



(그림 2) 각 부하 환경에서의 노드별 일반 프로세스 평균 처리 시간



(그림 3) 부하 환경 유형 및 케이스별 각 노드의 프로세스 평균 지연율

36.7%, 77.3%만큼의 지연이 발생했다. 특히 1024MB 에서 급격히 증가한 모습을 볼 수 있는데, 이는 프로세스 처리를 위한 처리를 위한 메모리가 부족하여 Swap 메모리 사용 빈도가 급격히 늘어났기 때문이다.

Swap 메모리는 디스크 공간을 활용하여 가상의 메모리를 할당하는 것을 말한다. Swap 메모리 사용 빈도가 늘어나면 페이지 스와핑으로 인한 인터럽트 발생 빈도와 I/O 가 증가하기 때문에 프로세스 처리성능 저하로 직결된다. 이러한 성능 저하를 막기 위해 Swap 메모리를 비활성화할 수 있다. 그러나 Swap 메모리 비활성화 시 메모리 부족으로 인해 프로세스 실패가 증가할 수 있다. 최악의 경우 시스템이 실패할 수 있어 Swap 메모리 비활성화는 본 실험에서 처리 성능 저하를 막기 위한 대안이 될 수 없다.

스토리지 부하 부분에서 단위 부하 프로세스에 512MB 할당 시 평균적으로 약 39%의 프로세스 처리 지연이 발생했다. 1024MB 와 2048MB 에서는 약 76% 와 99%의 처리 지연이 발생했다. 이는 디스크 I/O 가 처리성능에 크게 영향을 미친다는 것을 나타낸다.

메모리와 스토리지 부분에서 각 노드별 프로세스 평균 처리 시간에 있어서 차이가 발생하는 것을 볼 수 있다. 이는 크게 두 가지의 요인이 복합적으로 작용한 것으로 추정된다. 첫째, 각 노드의 가상화 프로세스가 호스트 운영체제의 스케줄링 정책에 영향을 받는다. 둘째, 프로세스 컨테이너가 Swarm 의 스케줄링 정책에 의해 동등하게 분배되지 않고, 특정 노드에 편중 할당되었다. 그러나 전체 노드 집합의 지연율에 대한 경향성은 깨지지 않았음을 확인할 수 있다.

5. 결론

본 논문은 Docker Swarm 에서 일반 환경과 비교 환경에서 각각의 프로세스의 수행시간을 측정하고 이를 비교함으로써 워크로드 유형별 성능 저하 정도에 대한 실험 및 분석을 수행하였다. 특히 메모리와 스토리지 집약적인 워크로드 발생 시 성능 저하가 크게 나타났으며, 이는 스토리지 액세스 작업이 프로세스 성능 저하에 영향을 크게 미친다는 것을 의미한다.

이를 통해 본 연구는 자율주행과 같은 실시간 응답성과 가용성이 중요한 시스템에서 메모리, 스토리지 집약적인 워크로드가 발생했을 때 리소스의 사용량 및 가용량을 고려한 컨테이너 스케줄링 알고리즘의 최적화가 필요하다는 것을 보여준다. 또한 프로세스 처리 지연 감소를 위해 Swap 메모리를 비활성화하더라도 메모리 부하 시 시스템 가용성을 보장하기 위한 연구의 필요성을 제시한다. 향후 연구는 컨테이너 기반 메모리 및 스토리지 부하 환경에서 성능 저하를 최소화할 수 있는 컨테이너 스케줄링 알고리즘을 설계하고 제안한 알고리즘에 입각한 컨테이너 스케줄러를 구현하려 한다.

본 연구는 과학기술정보통신부 및 정보통신기획평가원의 대학 ICT 연구센터지원사업의 연구결과로 수행되었음 (IITP-2018-0-01405)
본 연구는 과학기술정보통신부 및 정보통신기획평가원의 ICT 혁신인재 4.0 사업의 연구결과로 수행되었음(IITP-2022-RS-2022-00156439)

참고문헌

[1] Erl, Thomas, Ricardo Puttini, and Zaigham Mahmood, “Cloud computing: concepts, technology, & architecture”, Pearson Education, 2013.
[2] S. Herrnleben et al., “Towards Adaptive Car-to-Cloud Communication”, 2019 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops), 2019, pp. 119-124.
[3] J. Güldenring and C. Wietfeld, "Scalability Analysis of Context-Aware Multi-RAT Car-to-Cloud Communication," 2020 IEEE 92nd Vehicular Technology Conference (VTC2020-Fall), 2020, pp. 1-6.
[4] Malinverno, Marco, et al., “An Edge-Based Framework for Enhanced Road Safety of Connected Cars”, IEEE Access, vol. 8, pp. 58018-58031, 2020.
[5] Casalicchio, Emiliano, “Container orchestration: a survey”, Systems Modeling: Methodologies and Tools (2019): 221-235.
[6] “Swarm Mode Overview”, https://docs.docker.com/engine/swarm.
[7] “stress”, https://linux.die.net/man/1/stress.