확장가능한 시스템 설계를 위한 쿠버네티스 컨테이너 런타임별 웹 애플리케이션 성능 분석

강민재, 김형준, 유헌창
고려대학교 컴퓨터학과

{austin9228, ledzep0830, yuhc}@korea.ac.kr

A Performance Analysis of Web Applications on Kubernetes with Container Runtimes for Scalable System Design

Minjae Kang, HyungJun Kim, Heonchang Yu
Dept. of Computer Science and Engineering, Korea University

요 약

쿠버네티스는 컨테이너 오케스트레이션 플랫폼으로, 마이크로서비스 운영에 널리 사용된다. 이는 쿠버네티스가 오토스케일링을 지원하여 서비스 제공자가 부하의 변동에 효과적으로 대응할 수 있도록 지원하기 때문이다. 본 논문은 쿠버네티스 환경에서 runc 와 runsc 두 런타임을 대상으로 CPU 및 I/O 집약적 부하를 주는 웹 애플리케이션을 실행하여 처리 성능을 분석했다. 분석 결과 초당 요청 수가 일정 수준을 넘자 성능이 급락하였으며, 이를 바탕으로 워크로드를 고려한 오토스케일링의 필요성을 제고한다.

1. 서론

컨테이너 오케스트레이션 도구인 쿠버네티스의 발전과 더불어 이를 이용한 마이크로서비스 기반 애플리케이션이 급격하게 증가하고 있다. 쿠버네티스는 다양한 컨테이너 런타임을 지원하고, 런타임마다 특징이 상이하여 성능 또한 차이가 발생한다. 그러나 기존 연구에서는 런타임 유형에 따라 실제 웹 애플리케이션을 실행할 때 발생하는 성능 차이에 대한 연구가 미흡한 실정이다. 본 논문은 가장 널리 사용되는 runc 및 runsc 를 대상으로 CPU 및 I/O 집약적인 부하를 주는 웹 애플리케이션을 실행하여 성능 차이를 비교하였다. 실험 결과 워크로드 유형에 따라 성능 하락 폭 및 시점이 다르게 나타났고, 이를 바탕으로 워크로드를 고려한 오토스케일링의 필요성을 제고한다.

2. OCI 컨테이너 런타임

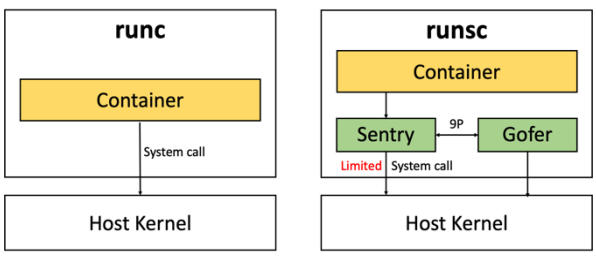
OCI(Open Container Initiative)[1]는 실제 컨테이너를 생성하고 실행하는 런타임에 대한 표준을 정하는 단체로, 해당 단체의 표준을 준수하는 런타임을 OCI 런타임이라고 한다. 도커와 쿠버네티스에서 기본 런타임으로 채용중인 runc 와 구글에서 개발한 runsc (gVisor)가 대표적인 예시이다.

2.1. runc

runc[2]는 가장 널리 사용되는 OCI 런타임으로, libcontainer 의 CLI(Command Line Interface) wrapper 도구이다. runc 는 libcontainer 를 통해 namespaces 와 cgroups 을 사용하여 컨테이너를 호스트 커널과 분리된 환경에서 실행한다.

2.2. runsc

runsc 는 gVisor[3]에서 사용하는 OCI 런타임으로, Sentry 와 Gofer 를 사용하여 호스트 커널 접근을 제한하는 방식으로 격리 수준을 높였다. Sentry 는 실제 시스템 콜을 필터링하는 유저 영역의 커널으로, x86 64 아키텍처 기준 350 개 중 270 개의 시스템 콜을 지원한다. Gofer 는 9P(Plan 9) 프로토콜을 사용해 Sentry 와 통신하며 파일시스템에 접근하는 역할을 수행한다.



(그림 1) runc 와 runsc 구조

3. 어플리케이션 유형별 런타임 성능 분석

3.1. 시스템 환경 설정

<표 1> 시스템 및 환경 설정

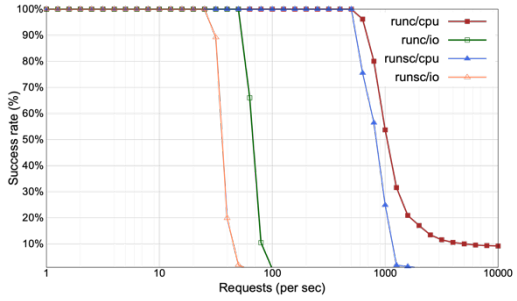
VM specification	Instance Type	Tencent Cloud SA2
	CPU	4 core 2.6GHz AMD EPYC™
	Memory	DDR4 16GB
Software version	Kubernetes	1.24
	containerd	1.6.21
	runc	1.17
	runsc	1.10

시스템 및 환경 설정은 표 1 과 같다. VM 3 대를 활용하여 컨트롤 플레인 1 대, 워커 노드 2 대로 클러스터를 구축했고, 웹 서버는 Apache 2.4.57 과 PHP 8.2.9 를 사용했다.

3.2. 실험 설계 및 비교 분석

부하 실험을 위해 GET 요청을 받으면 100×100 행렬 곱셈을 수행하는 CPU 집약적인 API 와 텍스트 파일을 열고 내용을 새 파일에 저장 후 삭제하는 작업을 5000 회 수행하는 I/O 집약적인 API 를 구현했다.

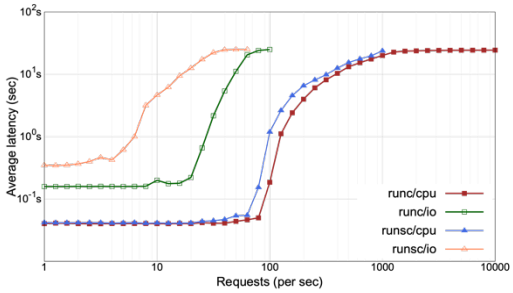
실험은 HTTP 부하 실험 도구인 vegeta[4]를 이용하여 웹 애플리케이션에 GET 요청을 보내는 방식으로 수행했다. CPU 부하 실험은 초당 1 회부터 10,000 회까지, I/O 부하 실험은 초당 1 회부터 1,000 회까지 로그 스케일로 횟수를 증가시키며 시행했다. 응답 측정에는 30 초 타임아웃을 두었고, 측정 지표는 요청 횟수별 성공률과 응답 시간이다. 위 방식으로 5 회 수행했으며, 수행 결과의 평균을 구하였다.



(그림 2) 런타임별 요청 횟수에 따른 성공률

그림 2 를 통해 runc 가 runsc 에 비해 성공률이 높음을 확인할 수 있다. 성공률 하락 시점을 비교했을 때, I/O 집약적인 부하의 경우 runc 는 초당 63 회, runsc 는 초당 31 회로 2 배 가량 차이가 발생했다. 이는 runsc 가 시스템 콜 필터링을 위해 추가 레이어를 거치기 때문이다. 반면 CPU 집약적 부하에서는 성공률 하락 시점이 동일하게 나타났다.

CPU 집약적 부하 실험에서 runsc 의 성공률이 1,000 회 이후 1% 미만으로 나타나는 것은 전체 시스템콜 수가 증가하며 전반적인 처리 속도가 하락하여 타임아웃 내 요청을 처리하지 못하기 때문이다.



(그림 3) 런타임별 요청 횟수에 따른 평균 응답 시간

그림 3 에서도 runsc 가 I/O 집약적인 부하를 수행할 때 runc 대비 최대 20.15 초의 응답 시간 차이를 보이며 CPU 집약적 부하에 비해 성능 차이가 뚜렷하게 나타났다. CPU 집약적인 부하의 경우 성능 차이가 최대 3.71 초로 크지 않은데, 이는 CPU 집약적인 작업은 시스템 콜 발생이 빈번하지 않기 때문이다.

지면상의 이유로 응답 시간 분포는 논문에 표현하지 못했으나, 응답 시간이 일정한 초반에 runc 가 runsc 대비 분산이 작게 나타났다. 이는 runc 를 사용할 때 안정적인 서비스를 제공할 수 있음을 나타낸다.

4. 결론 및 향후 과제

runc 가 runsc 에 비하여 성공률과 응답 시간 모두 앞섰다. 성능 격차는 I/O 집약적 부하에서 더 크게 나타났는데, 응답 시간은 runsc 가 최대 20.15 초 느렸고, 성공률 하락 시점은 32 회 빨랐다. CPU 집약적 부하의 경우, runsc 의 응답 시간이 최대 3.71 초 느린 것으로 나타났고, 성공률 하락 시점은 630 회로 동일했다. I/O 집약적 부하에서 성능 하락 폭이 큰 것은 runsc 의 시스템 콜 필터링 오버헤드 때문이다.

두 런타임 모두 특정 요청 수를 넘으며 성능이 급락했다. 워크로드 유형에 따라 저하 시점은 달랐으나, 동일 유형 내에서는 성능 저하 시점이 유사했다. 이에 착안하여, 성능 저하를 최소화할 수 있도록 워크로드를 고려한 오토스케일링 기법을 연구할 계획이다.

Acknowledgment

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-2023-RS-2022-00156439)

참고문헌

[1] Open Containers Initiative, About the OCI [internet], https://opencontainers.org/about/overview/

[2] Open Containers Initiative, runc [internet], https://github.com/opencontainers/runc

[3] Google, gVisor [internet], https://gvisor.dev/

[4] tsenart, vegeta [internet], http://github.com/tsenart/vegeta