CRIU를 이용한 Docker Container 체크포인트 성능 평가

한예진*, 최종무
단국대학교
{hyj0225, choijm}@dankook.ac.kr

Performance Evaluation of Docker Container using CRIU

Yejin Han*, Jongmoo Choi
Dankook University

요 약

CRIU는 애플리케이션의 상태를 저장하고 복구하기 위한 Checkpoint/Restore (C/R) 기능을 제공하는 리눅스 소프트웨어이다. 리눅스에서 실행중인 모든 프로세스들은 CRIU를 이용하여 체크포인트 시점의 상태를 디스크에 파일로 저장하고, 이후 저장된 파일을 읽어 중지 시점으로부터 프로세스를 다시 시작할 수 있다. CRIU는 클라우드 환경에서 여러 대의 컨테이너 자원들 간의 부하 균등을 위해 사용될 수 있으며, 이 과정에서 컨테이너 간의 간섭을 최소화하여 소요되는 체크포인트 시간을 줄이는 것이 요구된다. 본 논문은 CRIU의 동작 방식을 분석하고 여러 Docker Container가 실행 중인 환경에서 체크포인트 시간을 측정하였다. 실험 결과 여러 Docker Container가 수행되는 상황에서 동시에 체크포인트를 진행할 경우 ubuntu 애플리케이션의 수행시간은 0.36초에서 2.287초로 크게 증가하였다.

1. 서 론

현대 클라우드 환경에서 대규모 데이터센터 서버들은 Docker[1]와 같은 컨테이너 기술을 채택하여 사용하고 있다. Docker 컨테이너를 배포하는 형식인 Docker 이미지는 애플리케이션 구동에 필요한 시스템 자원만을 가상화하여 만들어진다. 따라서 게스트 OS를 포함하는 가상머신에 비해 적은 오버헤드로 애플리케이션을 빠르게 배포할 수 있어서 널리 사용되고 있다.

수많은 컨테이너가 동작하고 있는 가상 환경에서는 컨테이너의 라이프사이클을 관리하기 위해 실행 중인 상태를 저장하고 체크포인트 시점으로부터 컨테이너를 복원하는 Checkpoint/Restore (C/R) 기능이 필수적이다. Docker는 컨테이너의 C/R을 위해 리눅스 소프트웨어인 CRIU(Checkpoint/Restore In Userspace)[2]를 내부에서 통합하여 사용하고 있다. C/R는 예기치 못한 시스템 결함에서의 복구, 컨테이너 자원을 균등하게 분배하거나 시스템 업그레이드를 위한 실시간 live migration[3] 등을 가능하게 한다.

이러한 C/R 과정에서 소요되는 시간을 최소화하여 컨테이너가 빠르게 재시작할 수 있어야 한다. 하지만 체크포인트 작업이 이루어지는 과정에서 컨테이너 간의 간섭이 발생하면 이로 인한 성능저하가 생겨날 수 있다. 이는 QoS (Quality of Service)를 제공해야 하는 클라우드 환경에서 서비스 지연을 가져온다는 문제점이 있다.

본 논문은 Docker C/R 과정에서 사용하는 CRIU의 동작 과정을 설명하고 다양한 애플리케이션이 구축된 Docker 컨테이너들에 대해 체크포인트 실험을 진행하여 성능을 분석하였다.

* 이 논문은 2021년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원을 받아 수행된 연구임 (No.2021-0-01475, (SW 스타랩) 비정형 빅데이터를 위한 새로운 키-밸류 DB 개발)와 2019년도 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임(No. 2019R1F1A1062284)

2. Container checkpoint and restore

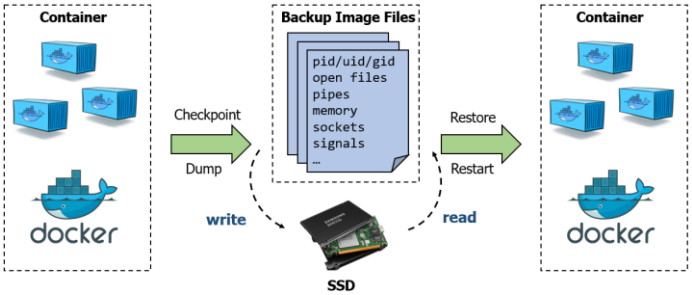

그림 1 Docker container checkpoint/restore 개요

그림 1은 Docker 컨테이너의 C/R이 수행되는 전반적인 과정을 보여준다. 여러 대의 Docker 컨테이너가 수행하고 있는 환경에서 컨테이너를 중단하고 체크포인트 명령을 내리면 현재 시점의 상태를 dump 한다. 체크포인트 수행 시 pid/uid/gid, 프로세스가 오픈한 파일들, 파이프, 메모리 페이지, 네트워크 소켓, signal 등을 포함한 컨테이너의 상태를 SSD에 이미지 파일로 쓰게 된다. 이후 컨테이너 재시작 명령을 보내면 백업한 이미지 파일을 읽어 중단한

시점으로부터 복원하여 컨테이너를 다시 시작하는 구조이다. 이러한 Docker의 컨테이너 Checkpoint는 Docker의 실험 기능(experimental feature)으로 내부적으로 CRIU를 통해서 이루어진다.

2.1 CRIU: Checkpoint/Restore In Userspace

CRIU는 실행 중인 프로세스를 중단시키고 현재 프로세스의 상태를 디스크에 파일로 저장한 뒤 이후 중단된 지점부터 프로세스를 복원하는 리눅스 소프트웨어로 kernel이 아닌 user space에서 작업이 이루어진다. CRIU에 대한 연구는 컨테이너 환경[4] 이외에도 RDMA 애플리케이션의 live migration[5], OS의 빠른 업데이트[6] 등 여러 시스템 분야에서 계속해서 진행 중이다. 다음은 CRIU의 핵심 기능인 Checkpoint와 Restore 과정이다.

2.2 Checkpoint 과정

CRIU는 ptrace[7]를 사용하여 process의 중단과 parasite code를 넣어 dump를 진행하며 프로세스의 현재 상태를 저장하기 위해 PID와 proc 파일 시스템을 필요로 한다. 프로세스의 실행을 중지한 후 CRIU는 process의 pipe, memory map, snapshot을 /proc/$PID/maps, /proc/$PID/map_files, proc/$PID/pagemap에서 가져와서 디스크에 이미지 파일로 저장한다.

2.3 Restore 과정

프로세스를 복구하기 위해서는 저장된 이미지 파일을 읽는다. 이후 CRIU는 checkpoint 할 당시와 동일한 PID로 프로세스를 fork하여 복구하게 된다. 이때 다른 프로세스가 해당 PID를 사용 중이면 복구할 수 없으며 /proc/sys/kernel/ns_last_pid를 읽어서 확인한다. 이는 부모-자식 프로세스 트리를 정확하게 복원하게 위함이다. CRIU는 프로세스를 fork 한 뒤 파일을 오픈하고 네임스페이스를 준비하고 메모리 영역을 매핑한다. 기본적인 task 자원들을 복구하고 난 후 CRIU 프로세스는 target 프로세스로 변형되어 전체 복구과정을 끝내게 된다.

3. Docker container 체크포인트 실험

표 1 실험 환경

CPU	Intel(R) Core(TM) i5-4440, 3.10GHz
Memory	32GB
Storage	Samsung 850 PRO 256GB SSD (SATA Interface)
Kernel	Linux 5.7.7
Tool	CRIU v3.15

본 논문은 표 1의 실험 환경을 기반으로 여러 Docker container의 체크포인트 성능을 평가하였다. 체크포인트 시 백업 이미지 파일들이 저장되는 storage는 SATA 인터페이스를 사용하는 Samsung 850 PRO 256G를 사용하였다.

그림 2는 서로 다른 크기의 ubuntu, redis, nginx, mongo, mysql 애플리케이션을 Docker의 컨테이너로 수행시키고 각각 체크포인트를 수행하여 성능을 평가한 결과이다. x축은 63.1MB에서 447MB에 이르는 Docker 이미지 크기이며 y축은 체크포인트 수행 시간(초)을 나타낸다. Checkpoint – alone은 수행 중인 컨테이너에 대해 개별적으로 체크포인트를 진행한 결과이며 Checkpoint – with others는 5개의 Docker 컨테이너가 백그라운드에서 수행 중일 때 동시에 체크포인트를 진행하였다.

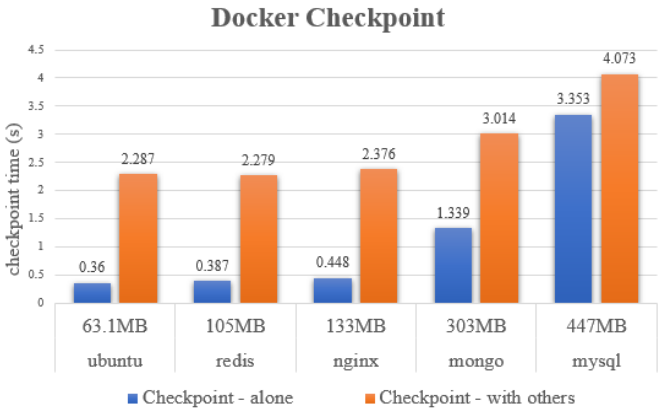

그림 2 Docker container 체크포인트 성능 평가

독립적으로 체크포인트를 수행하였을 때 애플리케이션 이미지 크기가 커짐에 따라 체크포인트 시간이 0.36초에서 3.353초까지 증가한다. 그런데 백그라운드에서 5개의 Docker의 컨테이너에 대해 동시에 체크포인트를 수행하면 ubuntu는 0.36초에서 2.287초로, mysql은 3.353초에서 4.073초로 크게 증가하였다. 이는 크기가 큰 이미지가 대한 체크포인트 작업이 늦게 끝나고 이 작업이 상대적으로 빨리 끝나는 이미지 체크포인트 작업에 영향을 미치기 때문이다.

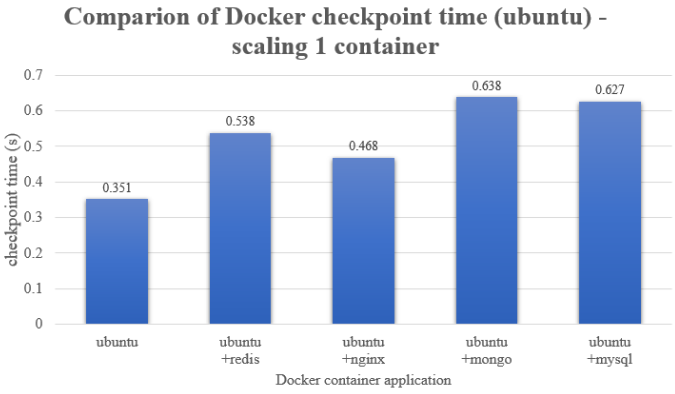

그림 3 Ubuntu container checkpoint job 간섭 평가

이에 추가로 컨테이너 작업 간 간섭을 확인하기 위해 ubuntu 애플리케이션과 추가로 1개의 컨테이너에 대해 동시에 체크포인트를 수행시키는 실험을 같이 진행하였다. 그림 3에서 ubuntu 컨테이너 체크포인트 수행을 단독으

로 진행했을 때 0.351초에서 redis와 같이 수행했을 때 0.538초로, nginx와는 0.468초, mongo와는 0.638초, mysql 애플리케이션과 체크포인트를 같이 수행하면 0.627로 소폭 증가하였음을 알 수 있다. 결과적으로 2개의 컨테이너가 동작하는 상황에 비해 다수의 컨테이너가 동시에 돌고 있는 환경에서 컨테이너 간 성능 간섭 문제가 크게 발생할 수 있음을 확인하였다.

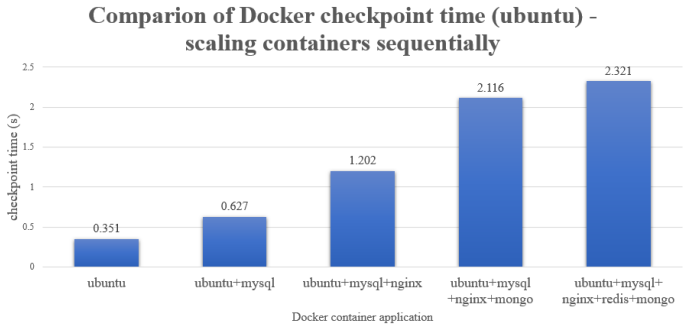

그림 4 Ubuntu container와 순차적으로 증가하는 container checkpoint job 간섭 평가

그림 4는 컨테이너 체크포인트 간섭을 비교하기 위해 1개가 아닌 추가적으로 컨테이너를 증가시키면서 실험을 진행하였다. 컨테이너를 하나씩 증가시키면서 체크포인트를 동시에 수행할수록 수행시간이 선형적으로 증가하였다. 결국 여러 컨테이너가 함께 수행되는 환경에서 간섭을 줄이기 위한 추가적인 기법이 필요하다.

4. 결 론

본 논문은 Docker 컨테이너가 수행되는 환경에서 애플리케이션 이미지 크기에 따라 체크포인트 수행시간을 측정하여 실험을 진행하였다. Docker의 컨테이너는 개별적으로 체크포인트가 수행될 경우 애플리케이션 이미지 크기에 따라 증가하는 시간을 보여주었다. 한편, 여러 대의 컨테이너들이 수행 중인 상황에서 동시에 체크포인트를 진행하면 작업 간 간섭이 발생하여 체크포인트 소요 시간이 크게 증가하는 것을 확인하였다. 향후 컨테이너 체크포인트 작업 간의 간섭을 최소화하기 위해 SSD의 주소공간을 독립된 영역(Zone)으로 나눠서 관리하는 ZNS SSD를 활용하여 컨테이너 C/R의 성능 개선을 이루도록 연구를 진행할 예정이다.

참고 문헌

[1] C. Anderson, “Docker.” IEEE Software, vol. 32, no. 3, 2015.
[2] CRIU. Checkpoint/Restore In Userspace, [Online]. Available: https://criu.org/Main_Page.
[3] Bo Xu, et al. “Sledge: Towards Efficient Live Migration of Docker Containers”, IEEE CLOUD, 321-328, 2020.
[4] Pekka Karhula, et al. “Checkpointing and Migration of IoT Edge Functions”, EdgeSys, 60-65, 2019
[5] Maksym Planeta, et al. “MigrOS: Transparent Live-Migration Support for Containerised RDMA Applications”, USENIX ATC, 47-63, 2021.
[6] Sanidhya Kashyap, et al. “Instant OS Updates via Userspace Checkpoint-and-Restart”, USENIX ATC, 605-619, 2016.
[7] ptrace. ptrace linux man page [Online]. Available: https://man7.org/linux/man-pages/man2/ptrace.2.html.