서비스 맞춤형 컨테이너를 위한 컨테이너의 레이어 파일 시스템 연구

용찬호, 허의남
경희대학교 컴퓨터공학과
e-mail : {alice_k106, johnhuh}@khu.ac.kr

Study on Layered File System for Service-customized Container

Chanho Yong, Eui-Nam Huh
Department of Computer Science and Engineering, Kyung Hee University

요 약

OS-level 가상화 기술은 애플리케이션을 배포하기 위한 새로운 패러다임으로서, 가상 머신을 대체할 수 있는 기술이다. 특히 컨테이너는 기존의 리눅스 컨테이너에 유니온 마운트 포인트(Union Mount Point) 와 레이어 구조의 이미지를 적용함으로써 보다 빠르고 효율적인 애플리케이션의 배포가 가능하다. 이러한 컨테이너의 특징들은 RoW(Redirect-on-Write), CoW(Copy-on-Write) 등의 스냅숏 기능을 제공하는 특정 파일 시스템에서만 사용될 수 있으며, 어플리케이션의 특징에 따라 적절한 파일 시스템을 사용해야한다. 따라서 본 논문에서는 컨테이너 이미지의 레이어 구조를 사용할 수 있는 파일 시스템들의 특징을 설명하고 이에 따른 쓰기 작업의 성능 평가를 진행한다.

1. 서 론

컨테이너 기술은 애플리케이션을 개발하고 배포하기 위한 새로운 관점을 제시하는 리눅스 컨테이너 기반의 가상화 기술로서, 기존의 하이퍼바이저 레벨 가상화 기술인 가상 머신에 비해 성능의 오버헤드가 거의 존재하지 않아 이를 대체할 수 있는 가상화 기술로서 주목받고 있다[2]. 컨테이너를 배포하기 위한 포맷인 이미지는 운영체제를 구성하기 위한 커널을 포함하고 있지 않아 가상 머신에 비해 이미지의 크기가 매우 작으며, 이미지를 레이어 구조로 배포함으로써 보다 효율적인 애플리케이션의 배포가 가능하다는 장점이 있다.

이미지의 레이어 구조를 구성하기 위해 사용되는 핵심 기술인 스냅숏(Snapshot)은 RoW, CoW 를 지원하는 파일 시스템을 구성해야만 사용할 수 있으며, 컨테이너가 지원하는 대표적인 파일 시스템은 AUFS(Advanced multi layered Unification Filesystem), Devicemapper 등이 있다. 이러한 파일 시스템은 운영체제, 커널버전, 컨테이너의 읽기 및 쓰기 빈도, 이미지 레이어 개수 등에 따라 성능이 상이하기 때문에 애플리케이션의 개발 및 배포 시나리오, 서비스 환경에 적합한 파일 시스템을 선택하는 것이 요구된다. 그러나, 컨테이너의 최적화된 성능을 보장하기 위한 구체적인 파일 시스템의 선택 기준 및 성능 평가에 대한 연구는 미비한 상황이다. 따라서 본 논문은 도커에서 사용되는 대표적인 파일 시스템들의 특징을 조사하고 성능 평가를 진행한다.

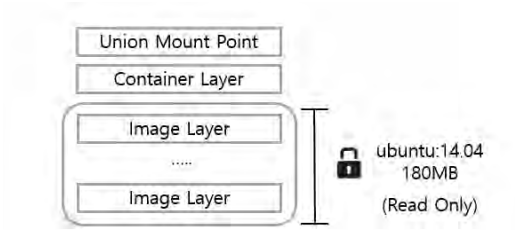
본 논문의 2 장에서는 컨테이너 이미지 구조에 대한 간략한 설명과 유니온 마운트 포인트를 사용하기 위한 스냅숏 기술들, 컨테이너에서 사용할 수 있는 대표적인 상용화 파일 시스템의 종류와 특징을 설명한다. 3 장에서 각 파일 시스템의 컨테이너 실행 시간 및 쓰기 작업의 성능 평가를 진행한 뒤 4 장에서 결론 및 향후 계획으로 본 논문을 마친다.

2. 관련 연구

2.1 도커 컨테이너와 이미지

도커 컨테이너는 리눅스의 자체 기능인 cgroup, namespace, chroot 를 이용해 프로세스 단위의 가상화 공간을 제공하는 가상화 기술이다[1]. 도커 컨테이너를 구동하기 위한 도커 이미지는 컨테이너의 변경 사항을 저장하는 레이어 구조로 이루어져 있으며, 어떠한 경우라도 읽기 전용으로 사용돼 불변(Immutable Infrastructure)의 상태를 유지한다. [그림 1]은 읽기 전용의 이미지 레이어와 컨테이너 레이어, 그리고 이를 사용하기 위한 유니온 마운트 포인트의 구조를 나타낸다.

컨테이너는 여러 개의 이미지 레이어와 컨테이너 레이어를 마운트한 유니온 마운트 포인트로부터 이미지의 파일들을 사용할 수 있으며, 컨테이너의 변경사항은 이미지가 아닌 컨테이너 레이어에 저장된다.



[그림 1] 이미지와 컨테이너의 구조

RoW 와 CoW 는 파일 시스템의 스냅숏을 구성하기 위해 사용되는 기술이다. 두 방식 모두 파일들을 스냅숏으로 생성한 뒤 쓰기 작업으로 인해 파일에 변경 사항이 발생할 경우 원본 스냅숏 파일을 유지하고 변경된 사항을 스냅숏 풀(Snapshot Pool)에 저장하는 방식으로 동작한다. CoW 는 스냅숏 파일에 쓰기 작업을 수행할 때 스냅숏 풀에 원본 파일을 복사하고 쓰기 요청을 반영하기 때문에 파일을 읽는 작업 1 번, 파일을 스냅숏 풀에 복사하고 변경된 사항을 쓰는 작업 총 2 번의 쓰기 작업이 발생한다. 반면 RoW 는 원본 파일을 묶은(Freeze) 뒤 스냅숏 풀에서 블록을 할당 받아 변경점을 저장하기 때문에 1 번의 쓰기 작업만이 발생한다[3]. [그림 2]는 RoW 의 작동 원리를 도식화한 것으로, 스냅숏 파일의 변경점을 스냅숏 풀에 할당하는 것을 나타낸다.

[그림 2] RoW 의 동작 방식

2.2. 레이어 파일 시스템

도커 엔진이 사용하려는 파일 시스템은 도커 이미지를 사용하는 컨테이너 레이어를 구성하기 위해 기본적으로 RoW 또는 CoW 기능을 지원해야 한다. 도커 이미지를 스냅숏으로 사용하고 이를 불변의 상태로 유지하며, 이미지 레이어의 파일에서 변경되는 사항들은 RoW 또는 CoW 의 원리에 의해 컨테이너 레이어에 별도로 저장된다. [표 1]은 본 논문에서 설명하는 각 파일 시스템들의 이러한 특징들을 나타낸다.

AUFS 는 컨테이너가 이미지의 파일에 쓰기 작업을 수행하면 가장 상층에 위치한 레이어부터 이미지를 찾아 해당 파일을 컨테이너 레이어로 복사하는 Copy-Up 작업을 수행한다[1]. 따라서 이미지의 파일을 처음 변경할 때는 오버헤드가 존재할 수 있지만, 컨테이너 환경에서 비교적 오랜 기간 사용돼온 파일 시스템이기 때문에 안정성 및 컨테이너의 할당 속도 측면에서 PaaS 에 적합한 파일 시스템으로 평가받는다[4]. AUFS 는 데비안(Debian) 계열의 리눅스에서 도커 엔진을 사용하면 자동으로 사용하도록 설정되는 파일 시스템으로서, 리눅스 커널 모듈에 포함되어 있지 않기 때문에 레드햇 계열의 리눅스에서는 기본적으로 사용할 수 없다.

Devicemapper 는 스토리지 풀 (Storage Pool)로부터 Allocate-on-Demand 원리에 의해 이미지 레이어의 변경 사항을 블록 단위로 할당 받아 컨테이너 레이어를 구성한다[4].

```
[root@localhost ~]# ls -lsah /var/lib/docker/devicemapper/devicemapper/
...
1.8G -rw-------. 1 root root 100G Feb  8 12:27 data
4.0M -rw-------. 1 root root 2.0G Feb  8 12:50 metadata
```

Sparse 파일인 data 파일이 도커에 필요한 파일들을 저장하고 있으며, 기본적으로 100G 크기의 스토리지 풀에서 공간을 할당한다. Devicemapper 는 대부분의 리눅스 커널에서 사용할 수 있다는 장점은 있지만, 다른 파일 시스템과 달리 컨테이너와 이미지 파일을 분리된 디렉터리에 저장하지 않아 레이어 단위로 멀티테넌시 환경을 관리할 수 없으며, 파일 시스템을 블록 디바이스로 사용하는 루프백(loopback) 디바이스ㅇ를 사용할 경우 일부 성능의 저하가 발생한다는 단점이 있다[5]. Devicemapper 는 이전 도커 릴리스에서 레드햇 계열의 리눅스를 위해 사용되었던 파일 시스템이며, 리눅스 커널에 기본적으로 탑재되어 있어 대부분의 리눅스 운영 체제에서 사용할 수 있다[1].

Overlay 는 AUFS 와 매우 유사한 구조를 가지고 있지만 수직적인 이미지 레이어 구조가 아니라는 점에서 좀 더 빠른 성능을 갖는다. AUFS 와 동일하게 컨테이너가 변경하려는 이미지의 파일을 컨테이너 레이어로 복사해 사용하는 Copy-Up 작업을 수행하지만, 이미지 레이어를 계층화된 구조로 사용하지 않기 때문에 파일을 찾는 과정에서 비교적 적은 오버헤드가 발생한다[4]. Overlay 는 레드햇 계열의 운영체제 및 라즈비안(Raspbian) 에서

[표 1] 파일 시스템 특징 비교

	AUFS	Devicemapper	Overlay	ZFS
파일 쓰기 작업 원리	Copy-Up	Allocate-on-Demand	Copy-Up	Allocate-on-Demand
모듈 의존성 여부	O	X	X	O
기본 지원 운영 체제	Ubuntu	-	CentOS, RHEL, Raspbian	-

자동으로 사용하도록 설정되는 파일 시스템으로, 리눅스 커널에 기본적으로 포함되어 있어 대부분의 리눅스에서 사용할 수 있다.

ZFS 는 썬 마이크로시스템즈(Sun Microsystems)에서 개발한 128 비트 파일 시스템으로, 리눅스 커널에 포함되어있지 않아 별도의 설치가 필요하다. ZFS 는 이미지의 파일에 쓰기 작업을 수행할 때 zpool 에서 Allocate-on-Demand 원리에 의해 128kb 크기의 블록을 할당해 저장하며, 컨테이너와 이미지를 구성하기 위해 클론(Clone)과 스냅숏이라는 개념을 사용한다[6]. ZFS 는 ARC(Adaptive Replacement Cache), 압축 등 다양한 기능을 제공해 반복되는 읽기 및 쓰기 작업에 대해 우수한 성능을 가지지만, 여러 개의 컨테이너를 동시에 사용할 경우 높은 메모리 점유율을 나타낼 수 있다는 단점이 있다.

파일 시스템은 레이어를 구성하는 방법에 따라 컨테이너 생성 시간 및 파일 입출력의 성능이 다르다. 따라서 컨테이너의 성능을 최적화하기 위해서는 파일 시스템의 성능을 면밀히 조사한 뒤 애플리케이션의 특성에 따라 파일 시스템을 선택하는 것이 요구된다.

3. 파일 시스템 성능 평가

본 성능 평가의 지표는 (1) 서비스의 실시간성을 보장하기 위한 컨테이너 생성 시간, (2) 서비스의 지연을 최소화하기 위한 파일 입출력 수행 시간으로 설정한다. 이에 따라 각 파일 시스템의 성능을 평가하기 위해 (1) 컨테이너 생성 및 시작 시간, (2) 단일 파일에 대한 쓰기 및 읽기 시간, (3) 분할된 파일들에 대한 쓰기 및 읽기 시간을 측정한다. 이는 AWS(Amazon Web Service)의 t2.medium 인스턴스에서 SSD 저장 장치를 사용해 테스트했으며, Ubuntu 16.04 운영 체제의 도커 엔진 버전 17.03.0-ce 에서 수행되었다. Devicemapper 는 루프백 디바이스를 사용하였다.

3.1 컨테이너 생성 및 시작 시간

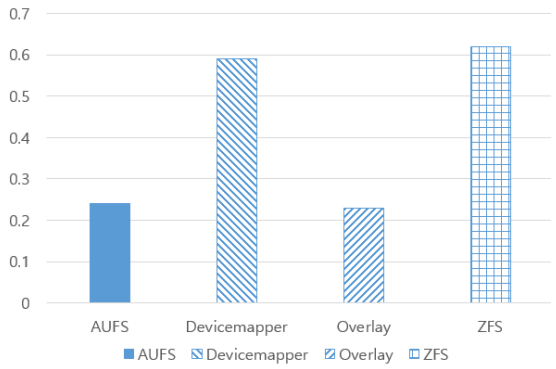
컨테이너를 연속으로 100 개를 10 개씩 10 번 생성하고 시작하는 시간을 측정함으로써 컨테이너 1 개에 소요되는 평균 시간을 계산하였으며, 사용된 명령어는 아래와 같다.

```
# docker run -itd busybox
```

스냅숏의 구조가 유사한 AUFS 와 Overlay 는 거의 동일한 수행 결과를 얻었지만 Devicemapper 와 ZFS 는 상대적으로 느린 수행 결과를 보여주었다. 그러나 ZFS 는 각 수행 결과와 평균의 오차가 적은 반면, DeviceMapper 는 수행 결과가 최소 0.5 초에서 최대 0.9 초에 달하는, 일률적이지 않은 결과를 얻었다.

[표 2] 컨테이너 10 개의 생성, 시작 평균 시간

	AUFS	Devicemapper	Overlay	ZFS
시간(s)	0.24	0.59	0.23	0.62



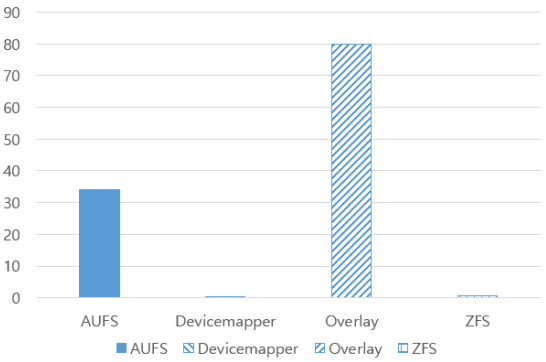
[그림 3] 컨테이너 10 개의 생성, 시작 평균 시간

3.2 단일 파일에 대한 쓰기 및 읽기 시간

각 파일 시스템에서 4G 크기의 파일에 쓰기 작업을 첫 번째로 수행한 시간을 측정하였으며, 동일한 작업을 새로운 컨테이너에서 10 회 수행해 평균 시간을 계산하였다.

[표 3] 4G 파일에 첫 번째 쓰기를 수행한 시간

	AUFS	Devicemapper	Overlay	ZFS
시간(s)	34	0.001	80	0.001



[그림 4] 4G 파일에 첫 번째 쓰기를 수행한 시간

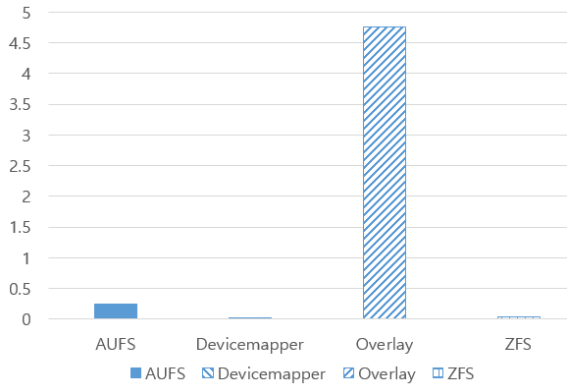
AUFS 와 Overlay 는 이미지 레이어의 파일에 처음 쓰기 작업을 수행할 때 컨테이너 레이어로 파일을 복사하기 때문에 새로운 파일을 생성하는 시간이 소요된 반면, Devicemapper 와 ZFS 는 블록 단위로 쓰기 작업을 수행하기 때문에 시간이 거의 소요되지 않았다.

3.3 분할된 파일들에 대한 쓰기 및 읽기 시간

각 파일 시스템에서 10 개의 128MB 크기 파일에 첫 번째로 쓰기 작업을 수행하는 시간을 측정했으며, 동일한 작업을 새로운 컨테이너에서 10 회 수행해 평균 시간을 계산하였다.

[표 4] 10 개의 128MB 파일에 첫 번째 쓰기를 수행한 시간

	AUFS	Devicemapper	Overlay	ZFS
시간(s)	0.25	0.013	4.76	0.036



[그림 5] 10 개의 128MB 파일에 첫 번째 쓰기를 수행한 시간

Devicemapper 와 ZFS 는 4.2 의 성능 평가에 비해 쓰기 작업의 소요 시간이 소폭 증가했으나, 이에 관련된 서비스의 실시간성을 보장하기에는 부족하지 않을 것으로 평가된다. 그러나 쓰기 작업을 수행하는 파일의 수와 크기가 늘어날수록 AUFS 와 Overlay 는 필연적으로 성능의 저하를 수반할 것으로 예상된다.

4. 결론 및 향후 계획

본 연구는 컨테이너에 적용할 수 있는 파일 시스템들의 특징을 조사하고 성능 평가를 진행하였다. AUFS, Overlay 파일 시스템은 많은 수의 컨테이너를 빠르게 생성하고 삭제해야 하는 웹 서비스 등의 환경에 적합하지만 파일의 크기가 큰 .iso 확장자의 파일, 빅데이터 분석을 위한 덤프 파일 등이 이미지 레이어에 존재할 경우 쓰기 작업 수행에 있어 실시간성을 보장할 수 없다는 단점이 있다. 이러한 관점에서는 블록 단위로 이미지 레이어의 변경점을 저장하는 Devicemapper, ZFS 와 같은 파일 시스템이 이점을 가진다.

컨테이너화(Containerize) 할 애플리케이션의 특징과 각 파일 시스템의 특징을 비교해 적합한 파일 시스템을 선택하면 읽기 및 쓰기, 컨테이너 시작 등 다양한 컨테이너 작업 수행의 성능 최적화를 이룰 수 있을 것으로 예상된다. 그러나 본 연구는 SSD 최적화, 메모리 캐싱, 데이터 압축 등을 통한 최적화를 다루지 않았으며, Devicemapper 의 루프백 디바이스에 대해서만 성능 평가를 수행하였다. 따라서 추후 연구에서 thinpool 기반의 Devicemapper, ZFS 또는 BtrFS 와 같은 파일 시스템에서의 SSD 최적화 및 캐싱 등의 성능 최적화를 진행한다.

참 고 문 헌

[1] Merkel, Dirk. "Docker: lightweight linux containers for consistent development and deployment." *Linux Journal* 2014.239 (2014): 2.

[2] Felter, Wes, et al. "An updated performance comparison of virtual machines and linux containers." *Performance Analysis of Systems and Software (ISPASS), 2015 IEEE International Symposium on*. IEEE, 2015.

[3] Xiao, Weijun, et al. "Design and analysis of block-level snapshots for data protection and recovery." *IEEE Transactions on Computers* 58.12 (2009): 1615-1625.

[4] Bellavista, Paolo, and Alessandro Zanni. "Feasibility of Fog Computing Deployment based on Docker Containerization over RaspberryPi." *Proceedings of the 18th International Conference on Distributed Computing and Networking*. ACM, 2017.

[5] Inagaki, Tatsushi, Yohei Ueda, and Moriyoshi Ohara. "Container management as emerging workload for operating systems." *Workload Characterization (IISWC), 2016 IEEE International Symposium on*. IEEE, 2016.

[6] Dawidek, Pawel Jakub, and Marshall Kirk McKusick. "porting the Solaris ZFS file system to the FreeBSD operating system." *; login:: the magazine of USENIX & SAGE* 32.3 (2007): 19-24.

"본 연구는 미래창조과학부 및 정보통신기술진흥센터의 대학 ICT 연구센터육성지원사업의 연구결과로 수행되었음" (IITP-2017-2013-0-00717)