The Journal of The Institute of Internet, Broadcasting and Communication (IIBC)
Vol. 20, No. 3, pp.97-103, Jun. 30, 2020. pISSN 2289-0238, eISSN 2289-0246

https://doi.org/10.7236/JIIBC.2020.20.3.97
JIIBC 2020-3-14

클라우드 환경에서 고성능 저장장치를 위한 동적 대역폭 분배 기법

Dynamic Bandwidth Distribution Method for High Performance Non-volatile Memory in Cloud Computing Environment

권필진[*], 안성용[**]

Piljin Kwon[*], Sungyong Ahn[**]

요 약 리눅스 Cgroups은 컨테이너 기반 클라우드 서비스 구축에서 각 컨테이너 별 시스템 자원을 할당하기 위한 핵심적인 역할을 담당하고 있다. 특히 입출력 자원의 경우 리눅스 Cgroups은 컨테이너의 가중치에 따라 입출력 대역폭을 분배하는 기법을 지원하고 있다. 그러나 성능 분석 결과에 따르면 현재 리눅스 Cgroups의 입출력 대역폭 분배 기법은 NVMe SSD와 같은 고성능 저장장치를 사용할 경우 입출력 성능이 크게 저하된다는 한계점을 가지고 있다. 따라서 본 논문에서는 리눅스 Cgroups을 위한 새로운 피드백 기반의 동적 대역폭 분배 기법을 제안하고자 한다. 제안하는 기법은 가중치에 따라 입출력 크레딧을 분배하며 고성능 저장장치의 성능 변화를 동적으로 반영해 입출력 크레딧을 계산함으로써 저장장치의 성능 저하를 최소화한다. 제안된 기법은 리눅스 커널 5.3에 구현되었으며 성능 평가 결과 정확한 입출력 대역폭 분배를 수행할 뿐만 아니라 기존 기법에 비해 최대 2배 높은 입출력 성능을 보여주었다.

Abstract Linux Cgroups takes a fundamental role for sharing system resources among multiple containers on container-based cloud computing environment. Especially for I/O resource, Linux Cgroups supports a mechanism for sharing I/O bandwidth in proportion to I/O weight. However, the current mechanism of Linux Cgroups using BFQ I/O scheduler seriously degrades the I/O performance with high bandwidth storage device such as NVMe SSDs. In this paper, we proposed a new feedback based I/O bandwidth sharing scheme for Linux Cgroups which allocates I/O credits to containers according to I/O weights and adjusts the amount of credits to performance fluctuation of NVMe SSDs. The proposed scheme is implemented on Linux kernel 5.3 and evaluated. The evaluation results show that it can share the I/O bandwidth among multiple containers proportionally to I/O weights while improving I/O performance more than twice as high as the existing scheme.

Key Words : Cloud service, Linux Cgroups, BFQ scheduler, Proportional I/O bandwidth sharing, NVMe SSD

[*]준회원, 부산대학교 정보컴퓨터공학부
[**]정회원, 부산대학교 정보컴퓨터공학부 (교신저자)
접수일자 2020년 5월 9일, 수정완료 2020년 5월 30일
게재확정일자 2020년 6월 5일

Received: 9 May, 2020 / Revised: 30 May, 2020 / Accepted: 5 June, 2020
[**]Corresponding Author: sungyong.ahn@pusan.ac.kr
School of Computer Science and Engineering, Pusan National University, Korea

Ⅰ. 서 론

오늘날 클라우드 컴퓨팅[1] 환경에서는 IaaS (Infrastructures-as-a-Service)[2]를 이용해 복수의 사용자들에게 서비스 수준 협약서 (Service Level Agreement)에 따라 CPU, 메모리, 저장장치와 같은 시스템 자원을 제공하고 사용자들은 사용한 자원에 따라 비용을 지불한다[3,4]. 특히 NVMe SSD와 같은 비휘발성 메모리 기반의 고성능 저장장치가 등장하면서 수 GB/s 수준의 높은 입출력 대역폭을 분배하는 문제가 주목을 받고 있다[5].

리눅스 Cgroups(Control groups)[6]은 프로세스 그룹 별로 CPU, 메모리, 저장장치와 같은 시스템 자원을 할당하는 리눅스 커널의 자원 할당 프레임워크로 컨테이너 기반의 클라우드 서비스 구축에서 핵심적인 역할을 담당하고 있다. 리눅스 Cgroups에서는 스토리지의 대역폭을 각 컨테이너에게 분배하기 위해 가중치 비례 자원 할당 기법(Proportional I/O Sharing Scheme)을 지원하고 있다. 현재 리눅스 Cgroups에서는 BFQ 스케줄러[7]를 이용해 가중치 비례 자원 할당 기법을 지원하고 있다. 그러나 BFQ 스케줄러는 스케줄링 오버헤드가 너무 커 고성능 저장장치에 적합하지 않다는 문제점이 있다.

따라서 본 논문에서는 멀티-큐 기반의 고성능 저장장치를 위한 새로운 가중치 비례 입출력 자원 할당 기법을 제안한다. 제안한 기법은 가중치 기반의 동적 크레딧 할당을 통해 고성능 스토리지의 대역폭을 각 컨테이너에게 효율적으로 분배한다. 본 논문에서 제안한 기법은 리눅스 커널에 직접 구현되었으며 성능 평가 결과에 따르면 가중치 비례 입출력 대역폭 할당을 수행하면서도 기존 BFQ 스케줄러와 비교할 때 2배 이상 높은 성능 향상을 달성하였다.

본 논문의 구성은 다음과 같다. 2장에서는 기존 리눅스 Cgrpups의 입출력 자원 할당 기법의 성능을 평가하고 문제점을 분석하였으며 3장에서는 본 논문에서 제안하는 고성능 저장장치를 위한 피드백 기반의 입출력 자원 할당 기법에 대해 설명 한다. 4장에서 성능평가 결과를 통해 제안한 기법의 우수성을 보이고 마지막으로 5장에서 결론을 맺는다.

표 1. 실험 환경
Table 1. Experimental environment

CPU	Intel Xeon E5-2620v4 2.1GHz, 8 core x 2
Memory	32G
NVMe SSD	Intel® SSD DC P4500 Series (1.0TB, 2.5in PCIe 3.1 x4, 3D1, TLC)
OS	Ubuntu 18.04, Linux Kernel 5.3.11

Ⅱ. 연구 배경

현재 리눅스에서는 컨테이너별 가중치에 비례해 저장장치의 대역폭을 할당하기 위해 BFQ(Budget Fair Queuing) 입출력 스케줄러를 사용하고 있다. 그러나 NVMe SSD와 같은 비휘발성 메모리 기반의 고성능 저장장치는 하드디스크(Hard Disk Drive)와 달리 입출력 스케줄링이 필요하지 않으므로 입출력 스케줄러를 이용한 입출력 자원 분배는 입출력 성능 저하를 일으킬 수 있다. 따라서 이번 장에서는 기존 리눅스에서 사용하고 있는 BFQ 스케줄러의 가중치 비례 입출력 대역폭 할당 기법에 대한 성능 평가를 진행하고 문제점을 분석 하고자 한다.

1. BFQ 스케줄러 성능 평가

BFQ 스케줄러는 각 컨테이너의 입출력 가중치에 비례해 버젯(Budget)을 할당하는 방식으로 저장장치의 입출력 대역폭을 분배한다. 그러나 본 논문에서 수행한 성능 평가 결과에 따르면 현재 BFQ 스케줄러는 NVMe SSD와 같은 고성능 저장장치의 대역폭을 복수의 컨테이너에 분배할 경우 입출력 성능이 크게 저하된다는 문제점을 가지고 있다. 성능 평가는 표 1에서 보는 것과 같이 인텔 제온 CPU와 인텔 데이터센터용 고성능 NVMe SSD를 사용해 진행되었으며 4개의 Fio[8] 프로세스를 이용해 임의 읽기(Random Read) 대역폭을 측정하였다. 그림 1에서 리눅스 Cgroups을 이용해 동일한 입출력 가

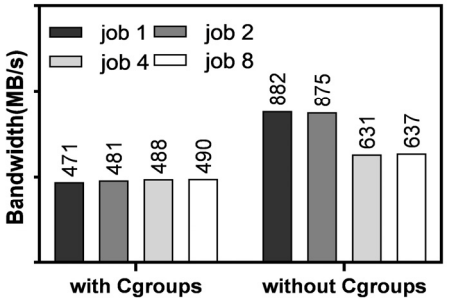

그림 1. 리눅스 Cgroups 사용에 따른 입출력 성능 변화
Fig. 1. I/O bandwidth changing while using Linux Cgroups

중치를 가진 4개의 컨테이너에서 Fio를 수행하는 경우와 컨테이너 없이 수행할 경우 측정된 입출력 대역폭을 비교하고 있다. 그림 1에서 알 수 있듯이 리눅스 Cgroup을 이용할 경우(w/ Cgroups) 입출력 대역폭이 가중치에 따라 각 컨테이너에 동등하게 분배되지만 그렇지 않은 경우(w/o Cgroups)와 비교할 때 전체 입출력 대역폭이 약 1.9배까지 감소하는 것을 확인할 수 있다.

2. 성능 평가 결과 분석

기존 리눅스 Cgroups의 입출력 대역폭 분배 기법에서 발생하는 입출력 성능 감소의 주된 원인은 BFQ 스케줄러에서 찾을 수 있다. BFQ 스케줄러는 프로세스 별로 입출력 큐를 관리하며 현재 입출력 큐에 더 이상 처리할 입출력 요청이 없는 경우 다음 두 가지 동작 중 하나를 선택한다. 첫 번째는 'keep queue'로 현재 서비스 중인 입출력 큐에 사용가능한 대역폭이 남아 있는 경우 다른 입출력 큐를 처리하지 않고 일정 시간동안 새로운 입출력 요청이 도착하기를 대기한다. 이는 현재 입출력 큐의 서비스 대역폭을 보장하는 역할을 수행한다. 반면, 'new queue'의 경우 할당된 대역폭을 모두 사용했거나 가중치 기반 대역폭 할당을 사용하지 않는 경우로 즉시 다른 입출력 큐를 서비스한다.

본 논문에서는 리눅스 Cgroups을 사용할 경우 BFQ 스케줄러의 동작을 확인하기 위해 그림 1의 실험을 진행하는 동안 'keep queue'와 'new queue' 상태 횟수를 측정하였다. 측정 결과 표 2에서 확인할 수 있듯이 리눅스 Cgroups을 사용해 입출력 가중치를 설정할 경우 각 입출력 큐의 서비스 대역폭을 보장하기 위해 'keep queue' 상태를 선택하는 비율이 크게 증가한다는 것을 확인할 수 있다. 이는 BFQ 스케줄러가 각 입출력 큐의 서비스 대역폭을 공정하게 보장하기 위해 대기하는 유휴 시간이 크게 증가함을 의미하며 이로 인해 저장장치의 입출력 대역폭이 크게 감소하였음을 알 수 있다.

BFQ 스케줄러의 이러한 동작은 하드디스크와 같은 싱글-큐 저장장치에서 동기 입출력 스트림이 비동기 입출력 스트림에 의해 계속 지연될 경우 입출력 대역폭의 불균형이 발생되는 것을 막기 위한 것이다. 그러나 하드디스크와 달리 동시에 여러 개의 입출력 요청을 처리할 수 있는 NVMe SSD와 같은 멀티-큐 저장장치에서는 불필요한 동작으로 오히려 저장장치의 성능을 저하시키는 결과를 가져온다는 것을 실험을 통해 확인할 수 있다. 따라서 본 논문에서는 고성능 저장장치의 대역폭 저하 없이 입출력 대역폭을 가중치에 따라 효율적으로 분배할 수 있는 기법을 제안하고자 한다.

표 2. Linux Cgroup 설정에 따른 BFQ 스케줄러 상태 변화
Table 2. The queue status of BFQ scheduler according to using Linux Cgroups

큐 상태	Cgroup 설정	Cgroup 미설정
new queue	504	11,845,366
keep queue	7,209,532	11,875,459

Ⅲ. 피드백 기반 가중치 비례 입출력 대역폭 할당 기법

본 장에서는 기존 BFQ 기반의 입출력 대역폭 할당 기법을 대체하기 위해 피드백 기반의 새로운 가중치 비례 대역폭 할당 기법을 제안한다. 제안된 기법에서는 각 컨테이너의 가중치에 비례해 입출력 크레딧(Credit)을 할당함으로써 기존 BFQ와 달리 고성능 저장장치의 입출력 성능을 저해하지 않으면서도 입출력 대역폭을 분배하고자 한다.

1. 디자인 개요

본 논문에서 제안하는 피드백 기반의 가중치 비례대역폭 기법(Feedback-Based Allocation of Credits)은 그림 2에서 보는 것과 같이 크게 2 가지 동작으로 구현된다. 첫 번째 동작은 크레딧 할당과 소비로 이루어진다. 그림 2의 크레딧 할당 모듈(Credit allocator)은 각 컨테이너의 가중치에 비례해 주기적으로 입출력 크레딧을 컨테이너에 할당한다. 크레딧이 할당 되면 각 컨테이너는 할당된 크레딧을 사용해 입출력 요청을 전달할 수 있으며 크레딧을 모두 소모할 경우 다음 크레딧 할당이 이루

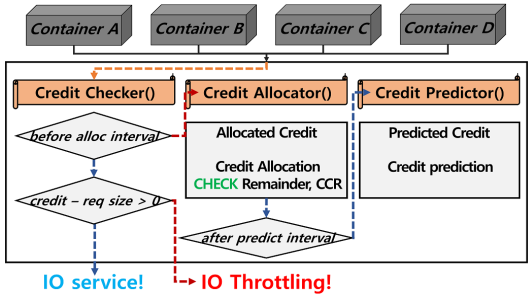

그림 2. 피드백 기반 가중치 비례 입출력 대역폭 할당 기법
Fig. 2. Overview of Feedback-Based Allocation of Credits

어질 때까지 입출력 요청을 중단한다. 제안한 기법은 기본적으로 이 동작을 통해 컨테이너 간의 가중치에 비례한 입출력 대역폭 할당을 수행할 수 있다. NVMe SSD와 같은 고성능 저장장치는 전통적인 하드디스크와 달리 입출력 스케줄링이 필요하지 않기 때문에 이러한 크레딧 기반의 대역폭 할당 기법이 매우 적합하다.

그림 2의 두 번째 모듈(Credit Predictor)은 피드백 기반의 크레딧 계산을 수행한다. 크레딧 기반의 입출력 자원 할당 기법이 고성능 저장장치의 대역폭을 저하시키지 는 것을 막기 위해서는 저장장치의 입출력 성능에 정확히 일치하는 크레딧 계산이 필요하다. 그러나 다양한 낸드플래시 메모리 기반의 NVMe SSD의 경우 내부적으로 수행하는 쓰레기수집(Garbage Collection)과 웨어 레벨링(Wear-leveling)으로 인해 입출력 성능이 변화하며 이는 저장장치의 공간 사용률이 높아질수록 GC의 빈도 및 오버헤드가 높아지면서 더 심해지는 경향을 가지고 있다.[9, 10] 따라서 본 논문에서는 저장장치의 입출력 성능 변화를 모니터링하고 이를 기반으로 다음 주기에 할당될 크레딧의 총량을 결정한다. 이러한 피드백 기반의 크레딧 계산을 통해 저장장치의 종류와 내부 상태의 변화에 상관없이 저장장치에서 제공할 수 있는 전체 입출력 대역폭을 각 컨테이너의 가중치에 비례해 할당할 수 있다.

2. 피드백 기반 크레딧 예측 기법

앞에서 언급된 것처럼 저장장치의 성능을 저하시키지 않기 위해서는 다음 주기에 필요한 크레딧의 양을 정확하게 예측하는 것이 매우 중요하다. 따라서 제안된 기법에서는 시간에 따라 변화하는 데이터를 예측하기 위해 가장 널리 사용되는 기법 중 하나인 이중 지수 평활법(Double exponential smoothing)을 통해 각 컨테이너의 기존 크레딧 사용률(Credit consumption rate)을 기반으로 다음 주기에 필요한 크레딧 양을 계산한다.[11]

$$\begin{cases} s_t^1 = s_{t-1}^1 + \alpha(x_t - s_{t-1}^1) \\ s_t^2 = s_{t-1}^2 + \alpha(s_t^1 - s_{t-1}^2) \\ c_{t+1} = (2s_t^1 - s_t^2) + (\frac{\alpha}{1-\alpha} * (s_t^1 - s_t^2)) \end{cases} \quad \cdots\cdots ①$$

수식 1에서 x_t는 크레딧 사용률로 시간에 따른 크레딧 사용량을 판단 할 수 있는 척도이다. 크레딧 사용률은 할당 주기 마다 사용한 크레딧을 사용한 시간으로 나눈 후, 크레딧 예측 주기 동안 합산한 값을 사용하였다. 따라서 이를 기반으로 계산한 s_t^1, s_t^2을 통해 다음에 사용할 수 있는 크레딧을 예측 할 수 있다. 예측된 값인 c_{t+1}의 경우, 예측 주기 동안 합산 한 값이므로 할당 주기/합산 횟수를 고려하여 다음 할당 주기에 사용할 수 있는 크레딧을 예측할 수 있다. 식 1에서 α는 측정되는 값의 추세에 따라 0에서 1사이의 값으로 설정할 수 있다. 본 기법에서는 예측 주기마다 예측 값과 실제 사용한 크레딧을 통해 오차값을 구한 후, 오차에 따라 α값을 다르게 적용하였다.

3. 크레딧 보정 기법

앞 절에서 소개한 이중 지수 평활법을 이용한 크레딧 사용량 예측은 효율적으로 작동하였으나 예측 값이 실제 크레딧 사용 패턴과 일치하지 않는 경우 이를 보정할 수 있는 기법이 필요하다. 먼저, 예측된 크레딧 양이 실제 필요한 크레딧 양에 비해 작을 경우 크레딧 할당 주기가 만기되기 전에 모든 컨테이너의 크레딧이 고갈되어 저장장치가 유휴상태에 빠지게 된다. 반면, 예측된 크레딧이 과대평가 된 경우 해당 컨테이너는 할당받은 크레딧을 모두 소모하지 못해 대역폭 할당이 가중치에 위배되는 상황이 발생할 수 있다. 따라서 다음과 같은 크레딧 보정 기법을 통해 크레딧 예측 실패에 대해 보정을 수행한다.

첫 번째, 현재 활성화 상태인 모든 컨테이너가 크레딧을 소모한 상태라면 크레딧 할당 주기를 기다리지 않고 즉시 크레딧을 배분한다. 이를 통해 모든 컨테이너의 크레딧이 고갈되어 저장장치가 불필요한 유휴 상태에 빠지는 것을 방지할 수 있다. 두 번째 보정 기법은 크레딧 할당 주기 만기 시 사용하지 못한 여분 크레딧을 계산하여 다음 주기에 사용할 수 있도록 하는 기법이며 이 기법을 통해 크레딧 예측이 실패하더라도 각 컨테이너는 가중치에 따라 입출력 대역폭을 사용할 수 있다.

표 3. 블록 입출력 워크로드의 특성
Table 3. The characteristics of block I/O workloads

워크로드 종류	읽기 쓰기 비율 (R : W)
Exchange	0.29:1
MSNMeta	2.97:1
Financial	0.3:1
MSNFS	2.04:1

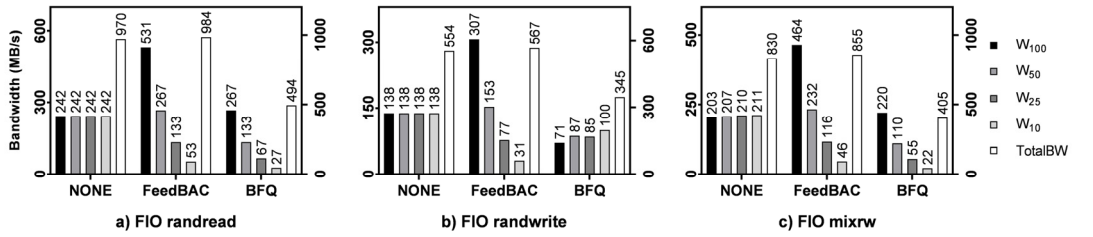

그림 3. Fio 벤치마크에서의 성능 평가 결과 (I/O depth=32, block size=4KB)
Fig. 3. Evaluation results with Fio benchmark (I/O depth=32, block size=4KB)

Ⅳ. 성능 평가

1. 실험 환경

본 논문에서 제안한 기법은 리눅스 커널 5.3에 직접 구현되었으며 표 1에 기술된 하드웨어 환경에서 성능 평가를 진행하였다. 다양한 성능 평가를 위해 합성 워크로드(Synthetic workload)뿐만 아니라 실제 입출력 워크로드를 이용해 실험을 진행하였다. 표 3에 기술된 입출력 워크로드는 블록 입출력 트레이스 재생 도구(trace-replay)[12]를 이용해 생성되었으며 UMass[13]와 SNIA[14]에서 수집한 블록 입출력 트레이스를 사용하였다. 서로 다른 가중치(100 : 50 : 25 : 10)를 가진 4개의 컨테이너를 생성해 입출력 대역폭을 측정하였으며 논문에서 제안한 FeedBAC 기법과 기존 리눅스의 None 스케줄러, BFQ 스케줄러와 성능 비교를 진행하였다.

2. 성능 평가 결과

그림 3은 Fio 벤치마크를 이용해 임의읽기, 임의쓰기, 임의읽기/쓰기(50:50) 워크로드를 수행했을 때 각 컨테이너에서 측정된 입출력 대역폭을 보여주고 있다. 그림 3에 따르면 논문에서 제안된 FeedBAC의 경우 워크로드에 관계없이 가중치에 비례한 입출력 대역폭 할당이 정확하게 이루어지는 것을 확인할 수 있다. 뿐만 아니라 전체 대역폭에서도 None 스케줄러와 비슷한 결과를 보여줌에 따라 저장장치의 성능 저하 없이 가중치 비례 입출력 대역폭을 할당할 수 있다는 것을 알 수 있다. 반면 BFQ 스케줄러의 경우 임의읽기의 경우 가중치 비례 대역폭 할당이 정확하게 이루어졌으나 임의쓰기의 경우 대역폭 할당이 가중치에 따라 공정하게 이루어지지 않았다. 이는 BFQ 스케줄러가 성능 변동 폭이 읽기에 비해 비교적 큰 쓰기 워크로드에 대해 제대로 대처하지 못한다는 것을 의미한다. 하지만 논문에서 제안한 FeedBAC 기법의 경우 피드백을 통해 저장장치의 성능 변화를 반영해 입출력 대역폭을 할당해 쓰기 워크로드에서도 대역폭 할당이 가중치에 따라 정확하게 이루어졌다. 또한 전체 입출력 성능 측면에서도 BFQ 스케줄러는 None이나 FeedBAC에 비해 크게 감소하는 것을 확인할 수 있다.

그림 4는 실제 워크로드를 이용한 성능 평가 결과를 보여주고 있다. 결과 그래프를 보면 None 스케줄러의 경우 가중치에 따른 대역폭 분배가 전혀 이루어지고 있지 않으며 BFQ 스케줄러 역시 입출력 대역폭 분배가 정확하게 이루어지지 않는다는 것을 확인할 수 있다. 반면

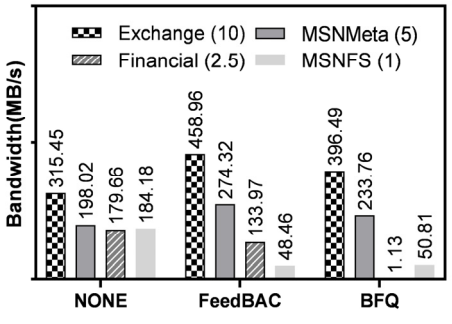

그림 4. 실제 입출력 워크로드에서의 성능 평가 결과
Fig. 4. Evaluation results with real I/O workloads

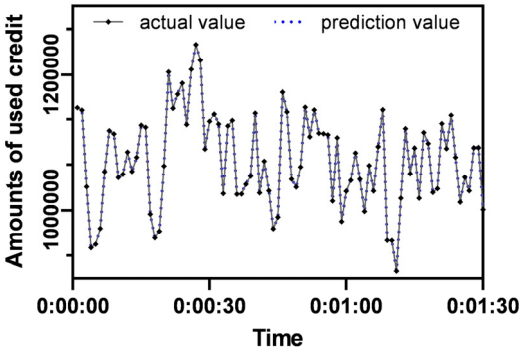

그림 5. 크레딧 예측 값과 실제 사용 값 비교
Fig. 5. Comparison between predicted and actual used credit value

논문에서 제안한 FeedBAC 기법의 경우 가중치에 비례해 각 컨테이너의 입출력 대역폭이 정확하게 분배되었을 뿐만 아니라 전체 입출력 성능 면에서도 기존의 BFQ 스케줄러 보다 우수하다는 것을 알 수 있다. 이는 논문에서 제안한 피드백 기반의 크레딧 예측 기법이 잘 작동한다는 것을 의미한다.

그림 5는 논문의 피드백 기반 크레딧 예측 기법을 평가하기 위해 그림 4의 성능 평가 중 Exchagne 워크로드를 기준으로 크레딧 예측값과 실제 크레딧 사용량의 변화를 비교해 보여주고 있다. 그래프에서 볼 수 있듯이, 예측 값과 실제 크레딧 사용량의 오차는 약 1% 이하로 이중 지수 평활법을 이용한 예측값과 거의 일치하는 것을 확인할 수 있다. 이로써 논문에서 제안한 크레딧 예측 기법이 NVMe SSD의 성능변화를 정확하게 반영하고 있음을 알 수 있다.

V. 결 론

본 논문에서는 컨테이너 가상화에서 NVMe SSD와 같은 고성능 저장장치의 입출력 자원을 효율적으로 분배하기 위한 새로운 기법을 제안하였다. 제안한 기법에서는 각 컨테이너의 입출력 가중치에 따라 입출력 크레딧을 할당함으로써 입출력 대역폭을 분배하고 피드백 기반의 크레딧 예측 기법을 통해 저장장치 성능 변화를 반영해 크레딧 양을 결정하였다. 성능 평가 결과에 따르면 제안된 FeedBAC 기법은 입출력 가중치에 비례해 입출력 대역폭을 각 컨테이너에게 정확히 분배하였으며 기존 BFQ 스케줄러에 비해 최대 2배 이상 높은 입출력 성능 개선을 확인할 수 있었다.

References

[1] J Youn et al, “The Establishment for Technology Development Plan for National Spatial Information Infrastructure Cloud Service”, Journal of KAIS, Vol. 18, No. 3 pp. 469-477, 2017.
DOI: https://doi.org/10.5762/KAIS.2017.18.3.469

[2] S. S. Manvi, and G. K. Shyam. "Resource management for Infrastructure as a Service (IaaS) in cloud computing: A survey." Journal of Network and Computer Applications, Vol. 41, pp. 424-440, 2014.
DOI: https://doi.org/10.1016/j.jnca.2013.10.004

[3] D. Serrano et al., “SLA guarantees for cloud services,” Future Generation Computer Systems, vol. 54, pp. 233-246, Jan. 2016.
DOI: https://doi.org/10.1016/j.future.2015.03.018.

[4] K. Jang et al., "A Study on Recognization for Quality Importance of Cloud Services," The Journal of the Institute of Internet, Broadcasting and Communication(JIIBC), VOL. 15 NO. 2, pp 39-44, 2015.
DOI: https://doi.org/10.7236/JIIBC.2017.17.4.143

[5] K. Lee et al, “A Study on Efficient SSD Selection Method”, Journal of KIIT. Vol. 15, No. 4, pp. 105-11, 2017.
DOI: https://doi.org/10.14801/jkiit.2017.15.4.105

[6] Llinux kernel cgroups document , URL https://www.kernel.org/doc/Documentation/cgroup-v1/cgroups.txt

[7] K. Oh, J. Park and Y. Eom, "H-BFQ: Supporting Multi-Level Hierarchical Cgroup in BFQ Scheduler", In 2020 IEEE International Conference on Big Data and Smart Computing (BigComp), pp. 366-369. 2020.
DOI:https://doi.org/10.1109/BigComp48618.2020.00-48

[8] Fio: Flexible I/O tester, URL https://github.com/axboe/fio

[9] J. Kim and S. Chung "A Study on Flash Memory Management Techniques", The Journal of The Institute of Internet, Broadcasting and Communication (JIIBC), Vol. 17, No. 4, pp.143-148, 2017.
DOI: https://doi.org/10.7236/JIIBC.2017.17.4.143

[10] E. Lee, M. Jeong, H. Bahn, "NVM-based Write Amplification Reduction to Avoid Performance Fluctuation of Flash Storage", The Journal of The Institute of Internet, Broadcasting and Communication(JIIBC), Vol. 16, No. 4, pp.15-20, 2016.
DOI: https://doi.org/10.7236/JIIBC.2016.16.4.15.

[11] J. Huang, C. Li and J. Yu, "Resource Prediction Based on Double Exponential Smoothing in Cloud Computing", in Proc. of the 2nd International Conference on Consumer Electronics, Communications and Networks (CECNet), 2012.
DOI: https://doi.org/10.1109/CECNet.2012.6201461

[12] Trace-replay. URL https://github.com/yongseokoh/trace-replay

[13] UMASS trace, URL http://traces.cs.umass.edu.

[14] D. Narayanan et al., “Migrating server storage to SSDs: analysis of tradeoffs,” in Proc. of the 4th ACM European Conf. on Computer Systems, 2009.
DOI: https://doi.org/10.1145/1519065.1519081

저 자 소 개

권 필 진(준회원)

- 2018년 2월 : 대구대학교 컴퓨터정보공학과 학사
- 2018년 9월~현재 : 부산대학교 정보컴퓨터공학부 석사과정

안 성 용(정회원)

- 2003년 2월 : 서울대학교 전기컴퓨터공학부 학사
- 2012년 2월 : 서울대학교 컴퓨터공학부 박사
- 2017년 9월~현재 : 부산대학교 정보컴퓨터공학부 조교수

※ 이 과제는 부산대학교 기본연구지원사업(2년)에 의하여 연구되었음