Journal of The Korea Society of Computer and Information
Vol. 20 No. 10, pp. 91-97, October 2015
www.ksci.re.kr
http://dx.doi.org/10.9708/jksci.2015.20.10.091

An MCFQ I/O Scheduler Considering Virtual Machine Bandwidth Distribution

Jung Kyu Park*

Abstract

In this paper, we propose a MCFQ I/O scheduler that is implemented by modifying the existing Linux CFQ I/O scheduler. MCFQ observes whether the user requested I/O bandwidth weight is well distributed. Based on the I/O bandwidth observation, we improved I/O performance of the existing bandwidth distribution ability by dynamically controlling the I/O time-slice of the virtual machine. The use of SSDs as storage has been increasing dramatically in recent computer systems due to their fast performance and low power usage. As the usage of SSD increases and prices fall, virtualized system administrators can take advantage of SSDs. However, studies on guaranteeing SLA(Service Level Agreement) services when multiple virtual machines share the SSD is still incomplete. In this paper was conducted to improve performance of the bandwidth distribution when multiple virtual machine are sharing a single SSD storage in a virtualized environment. In particular, it was observed that the performance of the bandwidth distribution varied widely when garbage collection occurs in the SSD. In order to reduce performance variance, we add a MoTS(Manager of Time Slice) on existing CFQ I/O scheduler.

▸ Keyword : SSD, Virtualization. SLA, CQF

I. Introduction

플래시 메모리 기반의 저장장치인 SSD는 빠른 성능과 저전력으로 최근 컴퓨터 시스템의 저장장치로 사용량이 급격하게 증가하고 있다. SSD의 집적도가 증가하고 가격이 하락함에 따라서 서버에서 많이 활용하고 있는 가상화 시스템에서도 SSD의 활용이 대두되고 있다. 하지만 최근 대부분의 운영체제와 시스템 소프트웨어는 오랜 기간 동안 저장장치로 사용되어온 하드디스크 기반을 가정하여 개발되어 왔으며 최근에 등장한 SSD의 물리적인 특성을 고려하지 않고 있다.[1.2].

SSD는 일반적인 시스템에서 많이 쓰이고 있지만 서버 환경에서 그 성능을 더 발휘할 수 있다. 서버 컴퓨터 시스템에서 가장 느린 하드웨어를 꼽는다면 저장장치인 하드디스크를 꼽을 수 있다. 그러므로 하드디스크를 SSD로 대체한다면 서버의 성능향상을 기대할 수 있다. 아직 하드디스크에 비해 SSD의 가격이 비싸기 때문에 하드디스크를 모두 대체하지 않고 하드디스크 위쪽 계층에 SSD를 캐쉬로 활용한다면 조금 비용을 추가하여 서버의 성능 향상이 가능하다.

최근 서버 시스템 중에서도 가장 이슈가 되고 있는 것은 클라우드 기반 서버 시스템과 가상화 환경 서버 시스템이다. 하나의 물리적인 시스템 위에 여러 개의 가상 머신을 운용할 수 있기 때문에 하드웨어 자원을 공유하면서 자원이 남는 경우를 최소한으로 줄여 시스템 자원을 효율적으로 사용할 수 있다. 이와 같이 SSD를 가상화 서버 환경 부분에 사용하면 자원을 각 가상 머신이 효율적으로 나누어 사용해야 하기 때문에 여러 가상머신의 SSD 공유는 꼭 필요한 상황이다.

가상화 환경의 서버 시스템에서 다른 자원에 비해서 CPU 와 메모리 자원의 배분은 쉬운 편이다. 그러나 저장 장치의 자원 분배는 비교적 어려운 연구 분야 중 하나다[14]. 특히 SLA(Service Level Agreement)적 측면에서 이러한 부분이 두드러지게 나타난다. 하드 디스크의 물리적인 특정(회전 지연 시간, 탐색 시간, 데이터

• First Author: Jung Kyu Park, Corresponding Author: Jung Kyu Park
*Jung Kyu Park(jkpark@dankook.ac.kr), Dept. of Computer Engineering, Dankook University
• Received: 2015. 07. 23, Revised: 2015. 08. 18, Accepted: 2015. 09. 15.
• The present research was conducted by the research fund of Dankook University in 2014

전송시간)으로 데이터를 읽고 쓰는 하드디스크와는 다르게 SSD는 물리적인 작업이 없이 전자적으로 데이터를 읽고 쓰기를 할 수 있다. 하드디스크와 SSD의 가장 큰 차이점은 SSD의 물리적인 특성에서 기인하는 가비지 컬렉션을 들 수 있다[3,4,13].

하드디스크는 기존 데이터에 위치에 새로운 데이터를 덮어쓰기가 가능하기 때문에 물리적으로 데이터를 삭제할 수 있다. 그러나 SSD의 경우 반도체 칩이 블록으로 구성되어 있고 블록은 페이지로 구성되어 있고, 데이터는 페이지 단위로 읽고 쓰기를 할 수 있고 블록단위 삭제가 가능하다. 그래서 어떤 데이터를 삭제를 할 때 FTL(Flash Translation Layer) 통해 해당 데이터가 유효하지 않다고 표시만 하고 실제로는 삭제하지 않는 경우가많다. 이러한 쓰기와 지우기가 반복됨에 따라서 플래시 내에는 유효하지 않은 페이지가 발생하게 된다. 유효하지 않은 페이지가 많이 발생하여 SSD의 플래쉬 안의 여유 공간이 부족해지면 블록 안에서 유효하지 않은 페이지들을 삭제하고 쓰기를 할 페이지들을 확보하기 위해 유효한 페이지들만 모아 다른 블록으로 복사하고 기존 블록을 삭제하는 과정을 거치게 되는데 이 과정을 가비지 컬렉션이라 부른다.

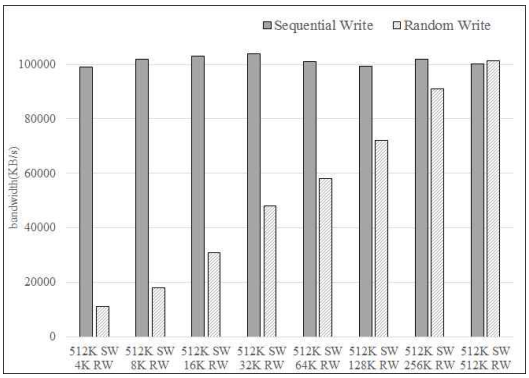


Fig. 1. Write performance test with cleaned SSD

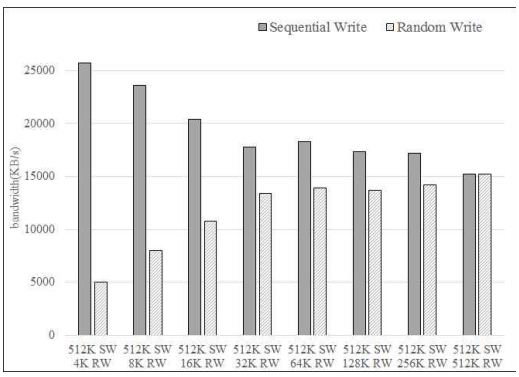


Fig. 2. Write performance test with aged SSD

가비지 컬렉션의 영향으로 인해 여러 개의 프로세스에 의해 하나의 SSD에 쓰기 작업이 발생할 경우 가비지 컬렉션이 일어나는 경우와 일어나지 않는 경우 각 프로세스의 쓰기 대역폭에 차이가 발생할 수 있다. 그림 1과 그림 2는 KVM 하이퍼바이저 가상화 환경에서 2개의 가상머신이 동작하며 가비지 컬렉션으로 인한 각 가상머신의 쓰기 성능이 어떻게 변화하는지 알아보기 위한 실험 결과이다. 실험은 1번 가상 머신에서 블록 크기를 512KB로 고정하고 1GB의 데이터를 순차 쓰기를 수행하고, 2번 가상 머신에서는 블록사이즈를 4KB부터 512KB까지 두 배씩 크게 하면서 1GB 데이터를 랜덤 쓰기를 동시에 진행하도록 구성하여 쓰기 대역폭을 조사하였다. 그림 1은 가비지 컬렉션이 일어나지 않는 상태에서 SSD, 그림 2는 가비지 컬렉션이 일어나는 상태의 SSD에서 각각 실험을 수행하였다.

그림 1 그래프에서 쓰기 대역폭에서 512KB의 순차쓰기는 랜덤 쓰기의 블록사이즈가 4KB에서 512KB로 증가함에 따라 영향을 받지 않는 것을 볼 수 있다. 즉, 가비지 컬렉션이 일어나지 않을 때는 각 가상머신의 쓰기에 영향을 주지 않는다.

하지만 그림 2의 가비지 컬렉션이 활발하게 일어나는 상황에서는 랜덤쓰기의 블록사이즈가 증가할수록 순차쓰기의 대역폭이 감소한다. 즉, 가비지 컬렉션이 일어날 때 두 개의 가상머신이 동시에 쓰기를 수행하면 랜덤 쓰기가 일으키는 가비지 컬렉션으로 인해 순차쓰기의 대역폭 감소를 발생시킬 수 있다는 것을 알 수 있다.

이러한 가비지 컬렉션은 SSD를 이용한 가상화 환경 서버에 악영향을 미칠 수 있다. 여러 가상 서버가 하나의 SSD에 동시에 접근 할 때 가비지 컬렉션이 발생할 때, 발생하지 않을 때 의 쓰기 대역폭의 비율이 다르게 변하기 때문에 비율에 따른 적절한 SLA를 보장할 수 없는 상황이 된다. 이런 이유로 SSD의 가비지 컬렉션의 특성을 고려하여 가비지 컬렉션이 일어날 때와 일어나지 않을 때 차이 없이 SLA를 보장해 줄 수 있는 해결 방법이 필요하다.

본 논문에서는 이러한 SSD의 특성을 고려한 리눅스 기본 스케줄러인 CFQ 스케줄러를 변형한 새로운 I/O 스케줄러인 MCFQ(Manager of Time Slice-CQF)를 제안한다[7]. MCFQ는 I/O 대역폭을 참조하여 현재 관리자가 의도한 대로 대역폭이 분배되고 있는지 판단한 후 타임 슬라이스 조정을 통하여 기존에 설정한 비율만큼 대역폭이 분배되도록 구현하였다.

II. Related Works

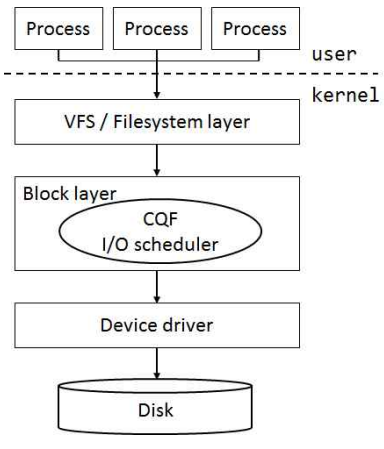


Fig. 3. CFQ I/O scheduler

기존에 하드디스크나 SSD에 대한 QoS(Quality of Service) 혹은 SLA(Service Level Agreement)를 위한 연구와 스케줄러에 대한 연구들이 있었다. 그러나 SLA를 보장해주기 위한 연구들은 최근 플래시 기반의 SSD를 고려하지 않고 하드디스크를 기반으로 하는 연구들 이었다. 기계적인 특성 없이 I/O 작업을 수행하는 SSD의 특성을 고려한 연구들은 I/O에 대한 공평성 보장을 위한 스케줄링 기법을 연구하였으나 SLA적 측면은 고려하지 않았다. 기존의 연구들은 QoS와 SLA를 고려하였으나 SSD의 중요한 특성인 가비지 컬렉션을 고려하지 않은 경우가 많다.

PARDA와 VM-PSQ는 가상화 환경이나 분산 시스템에서 I/O 자원 분배를 위한 새로운 스케줄링 기법을 제안했다 [5, 7]. PARDA 알고리즘은 분산 스토리지 환경에서 I/O 지연 시간을 측정하고 이 정보를 기반으로 I/O 큐의 길이를 조절하는 방식으로 I/O 대역폭 분배 성능과 공평성을 향상시켰다. VM-PSQ는 가상화 환경에서 I/O 타임 슬라이스와 스케줄링 토큰을 동시에 고려하여 기존 스케줄러보다 I/O 대역폭 분배 성능을 향상시켰다. 그러나 저장매체를 기존의 하드디스크에 한정시켰기 때문에 SSD에 대한 특성은 제대로 반영되지 않았다.

FIOS와 FlashFQ는 플래시 기반의 SSD 저장장치의 특성을 고려하여 공평한 I/O 자원 분배를 위한 새로운 스케줄러를 제시하였다[8, 9]. FIOS는 SSD의 읽기가 쓰기보다 빠른 특성을 이해하고 읽기와 쓰기에 대한 지연시간을 반영하는 스케줄러를 구현하여 공평한 I/O 분배에 대한 효율성 및 I/O 성능을 개선했다. FlashFQ는 SSD의 특성을 고려하여 여러 쓰레드가 I/O를 수행할 때 I/O 타임 슬라이스 분배방식을 개선하였다. 이 기법을 통해 기존의 스케줄러보다 I/O 응답시간을 줄였으며 I/O 공평성을 향상시켰다. 그러나 SLA와 SSD의 가비지 컬렉션 특성을 고려하지는 않았다.

CFQ(Completely Fair Queuing) I/O 스케줄러는 2003년도부터 리눅스 커널에 포함된 스케줄러이다. CFQ는 기본적으로 I/O 요청을 기반으로 하는 것이 아니라 타임 슬라이스 기반으로 설계되어있다.

그림 3은 리눅스의 CFQ I/O 스케줄러를 표시하고 있다. 여러 개의 프로세스에 의해 하나의 디스크 디바이스에 동시에 I/O 요청이 여러 개가 발생할 때, 각 I/O 요청의 타임 슬라이스를 공평하게 조절하여 최대한의 I/O 공정성을 유지하도록 하고 있다. CFQ는 프로세스 별로 I/O 대기 큐를 가지고 있어 프로세스들이 많은 I/O 요청 발생시킬 때 유용한 I/O 스케줄러이다. CFQ는 공평한 I/O 분배와 I/O에 대한 우선순위를 부여하는 것도 가능하다. 또한, 타임 슬라이스를 이용하여 I/O 대역폭 비율을 조정할 수 있다. KVM 방식을 사용하면 각 가상 머신은 각 프로세스로 취급되고 I/O 스케줄러에서 인식하기 때문에 이 프로세스에 I/O에 대한 가중치를 배정하는 것은 가상머신별로 I/O 대역폭 분배하는 것과 같다.

III. MCFQ Algorithm

본 논문에서 제안하는 MCFQ(Manager of Time Slice-CFQ)는 기존 리눅스의 CFQ I/O 스케줄러에서 타임 슬라이스 부분을 관리하기 위해 MoTS(Manger of Time Slice)를 추가하여 구현하였다. 그림 4는 CQF 내부에서 동작하는 MoTS를 보여주고 있다.

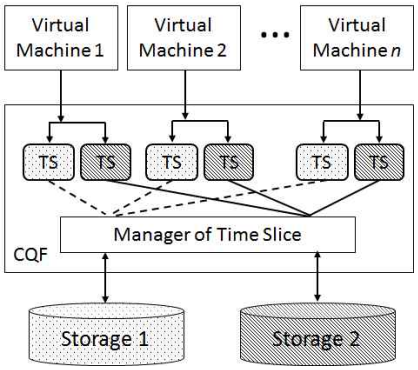


Fig. 4. Manager of Time Slice

사용자가 대역폭 분배를 위해 가상머신에 가중치를 부여했을 때 CFQ는 MoTS 쓰레드를 생성하고 작동시킨다. MoTS는 장치 독립적으로 동작하는데, 만약 하나의 하드디스크와 하나의 SSD를 여러 개의 가상머신에서 각각 공유하는 환경일 때에 CFQ는 하드디스크를 위한 MoTS 쓰레드 하나와 SSD를 위한 쓰레드 하나, 총 두 개의 쓰레드를 생성한다. 즉, 저장장치 하나마다 쓰레드를 생성하여 타임 슬라이스를 제어하도록 설계되어 있기 때문에 만약 가상머신이 저장장치별로 가중치를 다르게 줄 때에도 장치 독립적으로 타임 슬라이스를 제어할 수 있다.

MCFQ는 현재 장치에 대한 I/O 대역폭에 기반으로 타임 슬라이스를 조정한다. 그래서 하나의 디스크에 여러 개의 가상머신이 동시에 I/O를 수행할 때 각각 가상머신의 해당 저장장치에 대한 대역폭을 수집할 수 있어야 한다. 본 논문에서는 저장 장치의 자세한 I/O 대역폭 정보를 측정할 수 있도록 iostat를 활용 MoTS가 대역폭을 수집할 수 있도록 하였다[9].

1. Implementation

그림 5는 MoTS의 알고리즘 동작을 표시하고 있다. 가상머신에 의해 가상머신의 저장장치에 I/O 대역폭에 대한 가중치가 CFQ에 부여되면 MoTS 쓰레드가 동작하게 된다. MoTS 쓰레드는 사용자가 지정한 가중치만큼 대역폭이 잘 분배되는지 확인하기 위하여 iostat을 통하여 가상머신 별로 대역폭 1초마다 수집하여 이를 근거로 가장 가중치가 적게 부여된 TVM(Tiny VM)을 제외한 나머지 가상머신인 RVM(Remaining VM)의 I/O 타임 슬라이스를 조절한다. 2개 이상의 가상머신들에게 각자의 대역폭을 분배한다고 할 때, 가중치를 가장 적게 받은 TVM의

I/O 타임 슬라이스는 조절하지 않고 RVM의 타임 슬라이스를 TVM을 기준으로 하여 조정한다.

MoTS는 MoTS 쓰레드가 동작한지 BW(bandwidth Window)초가 지나지 않았다면 동작한 시간만큼의 대역폭을 참조하고, BW 초가 지나면 가상머신들의 최근 BW초간의 대역폭 값을 기반으로 하여 I/O 타임 슬라이스를 조정한다. 본 논문에서는 표 1과 같이 BW 값을 변화시켜가며 대역폭 분배성능을 실험하였다. 실험 결과를 참고하면 BW 값이 10일 때, 대역폭 분배성능이 가장 좋을 것을 알 수 있다. 대역폭을 참조하는 시간이 너무 적을 경우에는 대역폭의 값이 매 초마다 많이 변하는 워크로드의 타임 슬라이스 조절의 편차가 커지게 된다. 이와 반대로 대역폭 참조 시간이 너무 커지면 최근 대역폭의 영향이 너무 적게 반영되어 타임 슬라이스 조절이 정확하지 못해 대역폭 분배 성능이 떨어진다. 이런 이유로 본 논문에서는 10초를 BW 값으로 사용하였다.

Table 1. BW value per bandwidth distribution ratio

BW \ weight	VM1	VM2	VM3	VM4
	10	7	4	1
5	8.3	7.3	4.4	1
10	10.3	7.6	4.4	1
20	10.8	8.2	4.7	1

보정 인자 F(Factor)값은 사용자가 사전에 정한 가중치의 비율 IP(Ideal Proportionality)을 RVM의 최근 BW초간 대역폭인 BWR(BandWidth of RVM)과 TVM의 최근 10초간 대역폭인 BWT(Bandwidth of TVM)로 나눈 값으로 다시 나누어 얻을 수 있다. 실 예로, 두 가상머신 VM1과 VM2에 각각 1과 10의 가중치를 지정했다고 가정한다. 그런데 실제 최근 10초 간 대역폭의 평균값이 1대 5만큼 나왔다면 F값은 2가 된다. 다시 말해 VM2의 타임 슬라이스의 크기를 2배 늘여야 대역폭이 1대 10만큼 분배되는 것을 기대할 수 있다. 그렇지만 MoTS 알고리즘은 F값을 바로 타임 슬라이스에 적용시키지 않고, 최근 BW초간의 F값들의 평균값에 F를 곱하여 W초간의 F값이 반영된 F값을 구하고, 최근 BW초간 F값들의 평균값을 해당 가상머신의 타임 슬라이스 크기에 곱해준다. 이 과정을 통해 타임 슬라이스가 급격하게 변하는 것을 방지할 수 있고, 최근의 대역폭도 적절하게 F에 반영할 수 있다.

IV. Experiments and Results

1. Experiments environment

MCFQ I/O 스케줄러의 성능평가를 위하여 FileBench, TraceReplayer 두 종류의 벤치마크 툴을 사용하였다 [11, 12]. FileBench로 Fileserver, Varmail, TraceReplayer로 MSN, Exchange, Financial 총 다섯 종류의 워크로드로 실험을 수행하였다. 실험에서 대역폭 분배 성능 측정을 위해 KVM 가상머신 4개 설정 및 가동하였고, VM1부터 VM4까지 1부터 10까지 I/O 가중치를 다양하게 설정하였다. SSD는 최근 판매 되고 있는 삼성 840 Pro 256GB 모델을 이용하였고, 4개의 파티션에 각 30GB의 생성하여 4개의 가상 머신이 각 파티션을 독립적으로 사용할 수 있도록 하였다. 표 2에 실험 환경을 정리하였다.

2. Results

TVM : Tiny VM that has least weight of VM
RVM : Remaining VMs
BW : bandwidth windows
BWT : Mean bandwidth of TVM
BWR : Mean bandwidth of RVM
TSOVM : Time Slice of OVM

s = 0 // sec
while (1) {
  IP = weight of **RVM** / weight of **TVM**
  s++;
  if (time > BW)

$$BWR = \left(\sum_{s-BW-1}^{s} VMI\_bandwidth \right) / BW$$

$$BWT = \left(\sum_{s-BW-1}^{s} VMS\_bandwidth \right) / BW$$

$$F = \frac{IP}{BWR/BWT}$$

$$F_s = \left\{ \left(\sum_{s-bw-1}^{s-1} F_{s-1} \right) / W \right\} * F$$

$$TSOVM* = \left(\sum_{time-BW-1}^{time} F_s \right) / W$$

  else

$$BWR = \left(\sum_{1}^{s} VMI\_bandwidth \right) / s$$

$$BWT = \left(\sum_{1}^{s} VMS\_bandwidth \right) / s$$

$$F = \frac{IP}{BWR/BWT}$$

$$F_s = \left\{ \left(\sum_{s-bw-1}^{s-1} F_{s-1} \right) / W \right\} * F$$

$$TSOVM* = \left(\sum_{1}^{s} F_s \right) / s$$

  endif
  sleep(1);
}

Fig. 5. Algorithm of MoTS

MCFQ에 대한 성능평가는 가비지 컬렉션의 특성을 고려하여 쓰기 작업이 많이 일어나서 SSD가 에이징된 상태와 SSD에 데이터 없는 깨끗한 상태의 두 가지 조건으로 구분하여 실험하였다. 표 3에는 실험에 사용된 워크로드들의 특성을 정리하였다.

기존의 I/O 스케줄러인 CFQ는 대부분의 워크로드에서 에이징된 상태의 SSD와 깨끗한 상태의 SSD에서의 대역폭 분배성능이 차이가 났다. SSD에 쓰기 작업을 많이 수행하여 에이징된 SSD에 쓰기 작업을 할 경우 앞에서 언급한 것과 같이 가비지 컬렉션 연산이 발생하기 때문에 대역폭 분배 성능에 문제가 생기는 것을 실험에서 발견할 수 있었다.

Table 2. KVM environment

	W	core	mem	OS	Storage
Host		8	8GB	Fedora 14	Dedicated storage
VM1	10	1	1GB	CentOS 6.4	S840 Pro 30GB
VM2	7	1	1GB	CentOS 6.4	S840 Pro 30GB
VM3	4	1	1GB	CentOS 6.4	S840 Pro 30GB
VM4	1	1	1GB	CentOS 6.4	S840 Pro 30GB

Table 3. Properties of experiment workload

workload	write/read Ratio	average request size(KB)
Fileserver	2	72
varmail	1	24
MSN	66.7	22.5
Exchange	1.5	12.5
Financial	0.24	2.38

깨끗한 상태 SSD는 에이징된 SSD보다 대역폭 분배성능을 보였지만 모든 워크로드에서 이상적인 대역폭 분배성능을 보여주지는 못했다. 그러나 본 연구에서 제안한 MCFQ 스케줄러는 실시간으로 대역폭을 측정하여 타임 슬라이스를 조정하기 때문에 대부분의 경우에 이상적인 대역폭 분배에 근접한 결과를 보여주었다.

그림 6, 7은 fileserver와 varmail의 워크로드의 경우 기존의 CFQ는 에이징된 SSD에서는 20%대의 오차율를 보였고, 깨끗한 상태의 SSD에서도 15%근처의 오차율 보였다. 하지만 MCFQ는 fileserver 워크로드는 8%, varmail 워크로드는 2% 이내의 오차율을 보였다. 그림 8의 MSN 워크로드의 경우 SSD가 에이징된 상태 14.2%에서 7.6%로 성능개선을 보였으나, 깨끗한 상태에서는 근소하게 성능이 떨어졌다. 그림 9의 exchange 워크로드는 CFQ, MCFQ 모두 오류율 5% 이내의 좋은 성능을 보여주었다.

V. Conclusion

본 연구는 가상화 환경에서 여러 가상머신이 하나의 저장매체를 공유할 때 대역폭 분배 성능에 대한 개선을 목표로 진행되었다. 특히 최신 저장매체인 SSD에 초점을 맞추어서 SSD에서 가비지 컬렉션이 일어날 때와 일어나지 않을 때의 대역폭 분배성능이 다르다는 것을 관찰하였고, 이를 개선하기 위하여 기존의 CFQ I/O 스케줄러에 MoTS를 추가한 MCFQ I/O 스케줄러를 제안하였다.

MCFQ I/O 스케줄러는 I/O 대역폭 분배성능을 개선하기 위하여 기존 리눅스 I/O 스케줄러인 CFQ를 수정하였다. MCFQ는 먼저 각각 가상머신이 공유하는 저장매체의 파티션의 대역폭을 사용자가 의도한 대로 나누어지는지 관찰한다. 만약 사용자가 지정한 가중치보다 대역폭 분배가 잘되지 않는다면 해당 가상머신의 I/O 타임 슬라이스를 늘여주고, 대역폭 분배가 잘되면 I/O 타임 슬라이스를 줄여주는 동적 타임 슬라이스 제어 방법을 사용하였다.

벤치마크 도구를 이용하여 MCFQ I/O 스케줄러의 성능 평가한 결과 SSD의 가비지 컬렉션이 일어날 때와 일어나지 않을 때의 대역폭 분배성능이 기존의 CFQ I/O 스케줄러보다 성능 개선이 있었으며 대부분의 워크로드에서 사용자가 사전에 지정한 가중치대로 I/O 대역폭이 분배되는 것을 확인하였다.

REFERENCE

[1] X. Song, J. Yang and H. Chen, "Architecting Flash-based Solid-State Drive for High-performance I/O Virtualization," *Computer Architecture Letters*, vol. 13, no. 2, pp. 61-64, July 2013.

[2] T. Luo, S. Ma, R. Lee, X. Zhang, D. Liu, and Li Zhou, "S-CAVE: Effective SSD Caching to Improve Virtual Machine Storage Performance," *In Proc. of International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pages 103-112, Sep. 2013.

[3] F. Meng, L. Zhou, X. Ma, S. Uttamchandani, and D. Liu, "vCacheShare: Automated Server Flash Cache Space Management in a Virtualization Environment," *In Proc. of USENIX Conference on USENIX Annual Technical Conference(ATC)*, pages 133-144, June 2014.

[4] Y. Yang and J. Zhu, "Analytical modeling of garbage collection algorithms in hotness-aware flash-based solid state drives," *2014 30th Symposium on Mass*

Storage Systems and Technologies (MSST), pp. 1-10, June 2014

[5] Q. Niu, J. Dinan, Q. Lu and P. Sadayappan, "PARDA: A Fast Parallel Reuse Distance Analysis Algorithm," *IEEE 26th International Parallel & Distributed Processing Symposium (IPDPS)*, pp. 1284-1294, May 2012.

[6] D. Kang, C. Kim ; K. Kim and S. Jung, "Proportional Disk I/O Bandwidth Management for Server Virtualization Environment," *International Conference on Computer Science and Information Technology(ICCSIT '08)*, pp. 647-652, Sept. 2008.

[7] S. Seelam, R. Romero, P. Teller and B. Buros, "Enhancements to Linux I/O Scheduling," *In Proceedings of the Linux Symposium*, pp. 175-192, Sept. 2014.

[8] Q. Deng, Y. Luo, and J. Ge, "Dual threshold based unsupervised face image clustering," *In Proceedings of the 2nd International Conference on Industrial Mechatronics and Automation*, pp. 436-439, May 2010.

[9] SIMGRID Project, http://simgrid.gforge.inria.fr

[10] iostat, http://www.freebsd.org/cgi/man.cgi?iostat

[11] filebench, http://sourceforge.net/projects/filebench

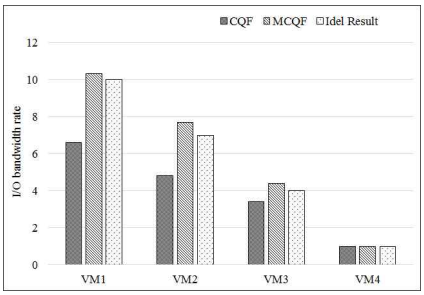
[12] ioreplay, https://code.google.com/p/ioapps/wiki/ioreplay

[13] C. Kim, J. Kim and C. Jeon, "Garbage Collection Method using Proxy Block considering Index Data Structure based on Flash Memory," *Journal of the Korea Society of Computer and Information*, vol. 20, no. 6, pp. 1-11, 2015.

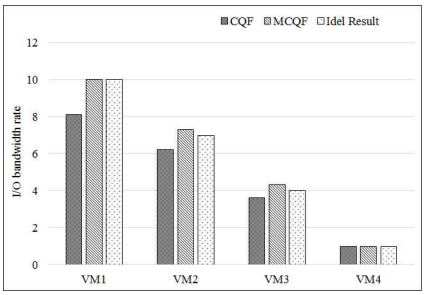
[14] S. H. Kim and J. W. Kwak, "A Virtual Machine Remapping Scheme for Reducing Relocation Time on a Cloud Cluster," *Journal of the Korea Society of Computer and Information*, vol. 19, no. 11, pp. 1-7, 2014.

Authors

Jung Kyu Park received the M.S. and Ph.D. degrees in computer engineering from Hongik University in 2002 and 2013, respectively. He has been a research professor at the Dankook Universitysince 2014. His research interests include operating system, new memory, embedded system and robotics theory and its application.

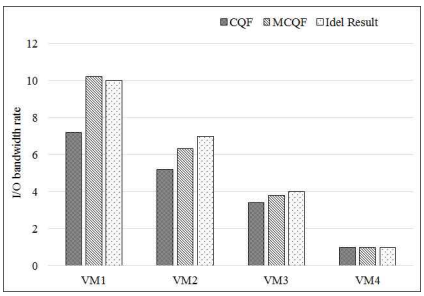


(a) Aged SSD

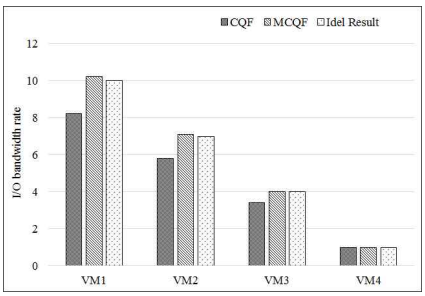


(b) Cleaned SSD

Fig. 6. Result of fileserver workload

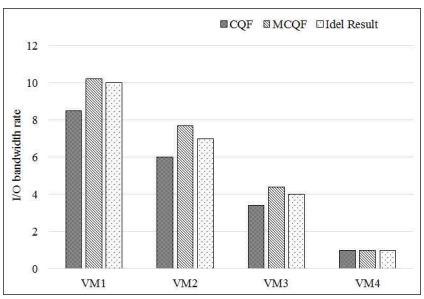


(a) Aged SSD

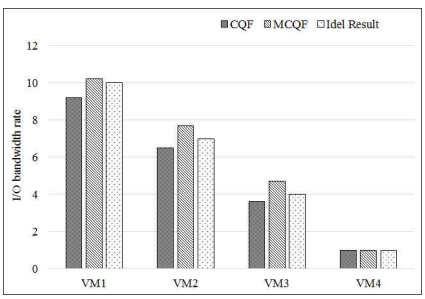


(b) Cleaned SSD

Fig. 7. Result of varmail workload

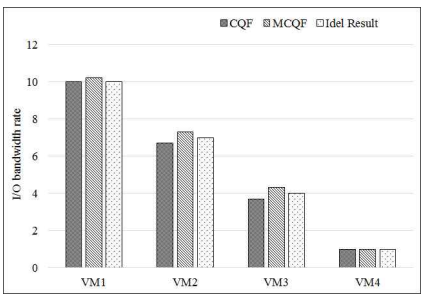


(a) Aged SSD

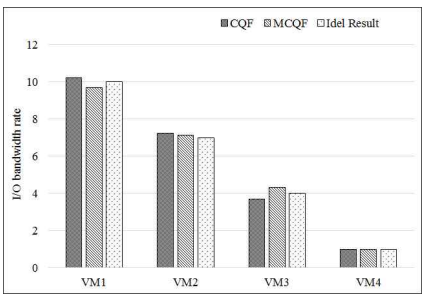


(b) Cleaned SSD

Fig. 8. Result of MSN workload



(a) Aged SSD



(b) Cleaned SSD

Fig. 9. Result of Exchange workload