ISSN 2383-630X(Print) / ISSN 2383-6296(Online)
Journal of KIISE, Vol. 43, No. 1, pp. 27-33, 2016. 1
http://dx.doi.org/10.5626/JOK.2016.43.1.27

다중 큐 SSD 기반 I/O 가상화 프레임워크의 성능 향상 기법

(Improving Performance of I/O Virtualization Framework based on Multi-queue SSD)

김 태 용 † 강 동 현 † 엄 영 익 ††
(Tae Yong Kim) (Dong Hyun Kang) (Young Ik Eom)

요 약 오늘날 가상화 기술은 가장 유용하게 사용되는 컴퓨팅 기술 중 하나이며 모든 컴퓨팅 환경에서 널리 활용되고 있다. 그러나 가상화 환경의 I/O 계층들은 호스트 머신의 I/O 동작 방식을 인지하지 못하도록 설계되어 있기 때문에 I/O 확장성 문제는 여전히 해결해야 할 문제로 남아 있다. 특히, 다중 큐 SSD가 보조 기억 장치로 사용될 경우, 증가한 잠금 경쟁과 제한된 I/O 병렬화 문제로 가상 머신은 다중 큐 SSD의 공인된 성능을 활용하지 못하는 문제가 발생한다. 이러한 성능 문제를 해결하기 위해 본 논문에서는 가상 CPU마다 전용 큐와 I/O 스레드를 할당하는 새로운 기법을 제안한다. 제안 기법은 성능 저하의 주요한 원인 중 하나인 잠금 경쟁을 효율적으로 분산시키고 또 다른 원인인 Virtio-blk-data-plane의 병렬화 문제를 해소한다. 제안 기법을 평가한 결과 최신 QEMU 보다 IOPS가 최대 155% 향상되는 것을 확인하였다.

키워드: 다중 큐, 솔리드 스테이트 드라이브, non-volatile memory express, 가상화, quick emulator

Abstract Virtualization has become one of the most helpful techniques in computing systems, and today it is prevalent in several computing environments including desktops, data-centers, and enterprises. However, since I/O layers are implemented to be oblivious to the I/O behaviors on virtual machines (VM), there still exists an I/O scalability issue in virtualized systems. In particular, when a multi-queue solid state drive (SSD) is used as a secondary storage, each system reveals a semantic gap that degrades the overall performance of the VM. This is due to two key problems, accelerated lock contentions and the I/O parallelism issue. In this paper, we propose a novel approach, including the design of virtual CPU (vCPU)-dedicated queues and I/O threads, which efficiently distributes the lock contentions and addresses the parallelism issue of Virtio-blk-data-plane in virtualized environments. Our approach is based on the above principle, which allocates a dedicated queue and an I/O thread for each vCPU to reduce the semantic gap. Our experimental results with various I/O traces clearly show that our design improves the I/O operations per second (IOPS) in virtualized environments by up to 155% over existing QEMU-based systems.

Keywords: multi-queue, solid state drive, non-volatile memory express, virtualization, quick emulator

· 이 논문은 2015년도 정부(미래창조과학부)의 재원으로 정보통신기술진흥센터의 지원을 받아 수행된 연구임(No.R0126-15-1082, (ICBMS-총괄) ICBMS(IoT, 클라우드, 빅데이터, 모바일, 정보보호) 핵심 기술 개발 사업 총괄 및 엑사스케일급 클라우드 스토리지 기술 개발)
· 이 논문은 2015 한국컴퓨터종합학술대회에서 '멀티큐 SSD 기반 IO 가상화 프레임워크의 성능 향상 기법'의 제목으로 발표된 논문을 확장한 것임

† 학생회원 : 성균관대학교 정보통신대학
taeyongkim@skku.edu
kkangsu@skku.edu
†† 종신회원 : 성균관대학교 정보통신대학 교수(Sungkyunkwan Univ.)
yieom@skku.edu
(Corresponding author임)

논문접수 : 2015년 7월 22일
(Received 22 July 2015)
논문수정 : 2015년 9월 7일
(Revised 7 September 2015)
심사완료 : 2015년 9월 22일
(Accepted 22 September 2015)

1. 서 론

오늘날 가상화 기술은 전과 달리 효율적인 최적화 기술들을 통해 가장 유용한 기술 중에 하나가 되었다. 하지만 가상화 환경에서 I/O 성능 문제는 아직까지 발전시켜야만 하는 문제로 남아 있으며, 따라서 산업계와 학계에서는 하드웨어 기반 접근법과 하드웨어적인 제약사항을 뛰어 넘는 소프트웨어적인 방법을 통해 최적화를 시도해오고 있다. 몇몇 반도체 제조사들은 하드웨어에서 가상화를 지원하기 위해 Virtualization Technology (VT)[1], Virtualization Extensions(VEs)[2], Single Root I/O Virtualization(SR-IOV)[3] 그리고 I/O Memory Management Unit(IOMMU)[4] 등 다양한 기술과 인터페이스를 제공한다. 이러한 하드웨어적인 접근 방법은 성능적인 측면에 있어서 비가상화에서의 성능에 근접할 수 있지만 그를 위한 특별한 하드웨어가 반드시 필요하기 때문에 모든 가상화 환경에 적용하기 어렵다. 예를 들어 SR-IOV와 IOMMU는 호스트 머신의 장치를 가상 머신에 직접 할당함으로써 가상 머신의 이송(VM Migration)[5,6]과 같은 가상화의 장점을 저해한다. 따라서 이러한 하드웨어적인 접근 방법의 제약사항을 극복하기 위해 많은 연구가 진행되어 왔으며, 몇몇 연구자들은 게스트 머신과 호스트 머신 사이의 통신 시 발생하는 Exit과 중복된 I/O 계층으로 인한 I/O 문맥 교환를 문제점으로 지적하였다. 그리고 그 해결 방법으로 Virtio [7], Kernel-based Virtual Machine(KVM)[8] 그리고 반가상화 I/O 시스템인 ELVIS[9]를 제안하였다.

Non-volatile Memory Express(NVMe) 솔리드 스테이트 드라이브(SSD)와 같은 PCI 익스프레스(PCIe)[10] 기반 다중 큐(Multi-queue) SSD는 그 내부적인 병렬성을 최대한 활용하여 보조 기억 장치로써 I/O 성능을 극적으로 향상시킨다. 하지만 처음에는 다중 큐 SSD를 운영체제에 적용시켰을 경우 성능적인 이점이 거의 나타나지 않았으며, 그 이유는 운영체제의 블록 계층에서 I/O 요청을 처리할 때마다 단일 요청 큐(Request Queue)에서 발생하는 잠금 경쟁(Lock Contention) 문제 때문이었다[11].

이와 유사하게 Quick Emulator(QEMU)[12] 기반 가상화 환경에서 다중 큐 SSD를 사용할 경우 가상 머신(Virtual Machine)의 I/O 성능에 미치는 영향을 확인하기 위해 FIO 벤치마크[13]를 통해 4KB 랜덤 읽기 IOPS를 직접 측정한 결과, 그림 1에서 보는 바와 같이 단일 큐 기반 SSD는 I/O를 발생시키는 프로세스의 수를 증가시킬 경우 가상화 환경과 비가상화 환경간의 성능 차이가 거의 없는 반면에 다중 큐 기반 SSD인 NVMe SSD는 비가상화 환경에서의 성능 대비 최대 74%의 성능 저하가 나타났다. 그 원인은 QEMU 내부의 잠금 경쟁 문제와 병렬화 문제 때문이며, 이를 해결하기 위해 가상 CPU당 하나의 전용 큐와 하나의 전용 I/O 스레드(Thread)를 할당하는 새로운 구조를 제안한다.

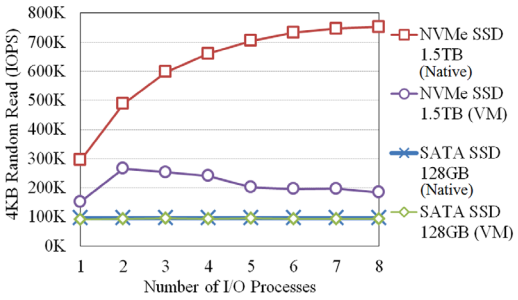

그림 1 가상화와 비가상화 환경에서 SATA SSD와 NVMe SSD의 성능 비교

Fig. 1 Performance comparison in a virtualized and a non-virtualized environment with a SATA SSD and an NVMe SSD through FIO

본 논문은 다음과 같이 구성된다. 2장에서는 새로운 구조를 제안하고, 3장에서는 제안한 구조를 대상으로 성능에 대한 실험 결과를 확인하며, 마지막으로 4장에서는 결론을 맺는다.

2. 구조 및 구현

2.1 가상 CPU 전용 큐

QEMU는 I/O 성능 향상을 위해 Virtio-blk-data-plane[14]이라는 기법을 사용한다. 이 기술을 분석해본 결과, 한 가상 CPU는 기본적으로 공유 메모리인 요청 큐에 접근할 때 반드시 단일 전역 뮤텍스(Mutex)에 락(Lock)을 걸어야만 한다. 이 상황에서 동일한 큐를 제어하려고 하는 다른 가상 CPU들은 지속적으로 락이 해제되기를 기다린다. 이러한 동작은 가상 CPU들 간의 심각한 잠금 경쟁을 유발하며, 이로 인해 내부적으로 불필요한 CPU 스케쥴링(Scheduling)이 야기된다.

따라서 우리는 기존 QEMU에 가상 CPU마다 전용 요청 큐를 할당하는 것을 제안한다. 이 방법은 가장 큰 장점은 잠금 경쟁을 최소화하여 가상 CPU의 병렬성을 높일 수 있다는 것이다. 그림 2(a)와 그림 2(b)는 QEMU를 통해 4개의 가상 CPU들을 할당한 가상 머신을 단순화시켜 가상 CPU 전용 큐의 적용 전후를 나타낸 것인데, 주로 주요 스레드들과 데이터 구조들을 묘사하였다. 두 개의 구조의 가장 큰 차이점은 요청 큐들의 수이다. 이전 연구들[14,15]에서는 가상 머신이 다수의 가상 CPU를 사용함에도 불구하고 하나 혹은 소수의 요청 큐들을 할당하여 사용했었다. 멀티코어 시스템에서 가상

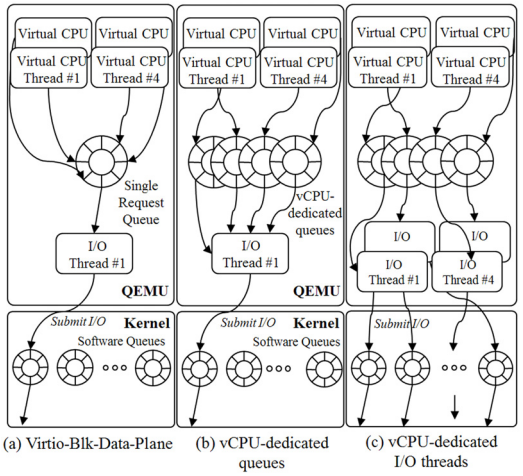

그림 2 Virtio-blk-data-plane과 가상 CPU 전용 큐 및 I/O 스레드를 가진 구조의 비교

Fig. 2 Comparison of the architecture of virtio-blk-data-plane and vCPU-dedicated queue with I/O thread

머신의 전체 성능을 향상시키기 위해 가상 머신은 요청 큐들의 수보다 많은 가상 CPU를 할당하게 된다면 이전 연구들이 제시했던 구조는 또 다시 잠금 경쟁에 시달릴 가능성이 높다. 따라서 이 잠금 경쟁을 지속적으로 줄이기 위해 요청 큐들의 수는 가상 CPU의 수와 일치해야 한다. 결과적으로, I/O를 지속적으로 발생하는 프로세스들의 수가 가상 CPU들의 수와 같을 때 락을 획득하기 위해 기다리는 시간이 50 마이크로세컨드에서 10 마이크로세컨드로 약 80% 감소하였다. 그러나 가상 CPU 전용 큐뿐만 아니라 다음에 소개할 가상 CPU 전용 I/O 스레드 역시 I/O의 병렬성 향상을 위해 절대적으로 필요하다.

2.2 가상 CPU 전용 I/O 스레드

가상 CPU 전용 큐가 전체적인 I/O 병렬성 문제를 부분적으로 해결하지만 그 근본적인 원인은 단일 I/O 스레드와 관계가 깊다. 리눅스 커널(Linux Kernel)은 일반적으로 I/O 병렬성을 위해 CPU 코어당 소프트웨어 큐와 장치에 의존적인 다수의 하드웨어 큐를 가지고 있다. 또한 I/O 요청이 들어올 때 해당 I/O를 수행하는 CPU의 번호에 따라 I/O 요청이 삽입되는 소프트웨어 큐의 위치가 결정된다. 하지만 각각의 가상 CPU가 I/O 수행을 요청함에도 불구하고 호스트 운영체제는 어떠한 가상 CPU가 해당 I/O를 요청하였는지 인식하지 못한다. 왜냐하면 가상 CPU가 아닌 별도의 단일 I/O 스레드가 실질적으로 모든 I/O 요청을 처리하기 때문이다. 이러한 이유로 가상 머신이 단일 I/O 스레드를 가진 경우 모든 I/O 요청들은 호스트 블록 계층의 단 하나의 큐에 삽입되어 처리된다. 결과적으로 그 단일 I/O 스레드는 비효율적으로 호스트의 소프트웨어 큐들을 사용하게 되며, 이는 I/O의 병렬성 향상에 치명적이다. 다중 큐 SSD는 호스트의 소프트웨어 큐를 전부 활용할 경우에 그 성능 최대화 할 수 있기 때문에 단일 I/O 스레드의 경우 다중 큐 SSD의 성능은 소프트웨어 큐들의 이용률에 의해 제한된다. 또한 그 단일 I/O 스레드는 수많은 양의 I/O 완료 동작을 수행해야 하기 때문에 필연적으로 심각한 병목현상을 초래하게 된다.

따라서 우리는 가상 CPU의 개수만큼 I/O 스레드를 할당하는 가상 CPU 전용 스레드 구조를 제안한다. 그리고 각각의 I/O 스레드는 가상 CPU 전용 요청 큐를 가상 CPU와 공유한다. 가상 CPU 전용 I/O 스레드 기법은 최종적으로 다중 큐 SSD의 하드웨어 큐들을 최대한 활용함으로써 병렬성을 향상시킬 수 있는데, 이는 각각의 I/O 스레드가 일반적으로 중복되지 않는 CPU 코어에서 수행되기 때문이다. 단일 스레드 기반 I/O 구조와 가상 CPU 전용 I/O 스레드 구조의 차이점은 그림 2(b)와 그림 2(c)에서 확인할 수 있다. 몇몇 연구자들[14,15]은 소수의 I/O 스레드들을 통해 SSD에 잠재된 모든 성능을 활용할 수 있다고 주장하였지만 가상 장치를 이용하여 직접 성능을 측정해본 결과 그것은 사실과 달랐다. 물론 I/O 집중적인 4개의 프로세스들을 이용할 경우 750K IOPS 성능을 가진 최신 NVMe SSD의 전체 능력을 사용할 수 있지만, SSD의 성능은 SSD 병렬화 구조와 인터페이스 기술의 발전으로 인하여 지속적으로 현재의 성능 한계를 넘어서고 있기 때문에 현재 성능을 만족시키는 소수의 I/O 스레드를 이용한 구조는 근시안적이다[16]. 따라서 단일 I/O 스레드로 인해 발생한 근본적인 성능 제약 사항을 해결하기 위해 가상 CPU 전용 I/O 스레드를 QEMU에 적용시켜야 한다.

가상 CPU 전용 큐와 I/O 스레드 적용 시 불필요해진 단일 전역 뮤텍스를 모두 제거하였으며, 각각의 I/O 스레드에 CPU 선호도(Affinity)를 설정하여 I/O 처리 완료(I/O Completion) 시 불필요한 문맥 교환(Context Switch)를 감소시키고 중복된 I/O Path를 단축하였다.

3. 실험 및 평가

3.1 실험 비교군

우리는 제안한 구조를 다음과 같이 다른 두 개의 기법과 비교하여 평가하였다. 우선 Baseline은 수정되지 않은 QEMU 2.1.2이며, Virtio-blk-data-plane 기법을 이용하였다. Baseline은 I/O 확장성 문제를 야기했던 단일 요청 큐와 단일 I/O 스레드 구조이다. MQ는 이전 연구자[15]가 제안한 구조를 가지고 있으며, 소수의 요청 큐와 단일 I/O 스레드를 가지고 있다. 우리는 연구자의 소

스코드를 수정 없이 사용하였으며, 실험에서는 생성된 가상 CPU의 개수와 동일하게 8개의 요청 큐를 할당하였다. 마지막으로 MT는 Baseline과 동일한 QEMU를 기반으로 가상 CPU 전용 큐와 I/O 스레드를 적용하였다.

3.2 평가 환경

모든 실험은 Intel i7-2600 3.40GHz 쿼드코어 CPU와 16GB RAM이 장착된 시스템에서 수행되었으며, 실물 장치 없이 다중 큐 동작을 시뮬레이션하는 널 블록 장치(Null Block Device)[17]를 대상으로 측정하였다. 이것은 리눅스 널 장치(Null Device)와 내부적인 I/O 동작은 유사하지만 다중 큐의 특성을 가지고 있다. 또한 실물 장치가 현재까지 도달하지 못하는 성능 범위까지 검증이 가능하다. 예를 들어 널 블록 장치는 8개의 I/O 프로세스를 통해 1200K IOPS를 달성할 수 있다. 실험 시스템에 하나의 가상 머신을 생성하며, 가상 머신은 8개의 가상 CPU와 14GB RAM을 할당한다. 실험에 사용된 CPU는 쿼드코어(Quad-core)이지만 하이퍼스레딩(Hyper Threading)을 통해 8개 스레드가 동작할 때 최대 성능을 보이기 때문에 가상 머신에서 8개의 가상 CPU를 할당한다. 추가로 가상 CPU의 개수가 물리적인 CPU 코어 수를 넘어가는 경우의 실험도 진행한다.

3.3 FIO 벤치마크

모든 I/O 작업은 요청 I/O 유형, 데이터 크기, 큐의 깊이, I/O 엔진, 캐쉬(Cache) 방식 그리고 I/O 발생 프로세스의 수 등 상세한 환경 설정을 통해 실험 결과를 명확히 할 수 있는 FIO를 통해 생성된다. 기본적으로 4가지 유형의 I/O 작업 (4KB 랜덤 읽기, 4KB 랜덤 쓰기, 32KB 순차 읽기, 32KB 순차 쓰기)을 통해 평가하며, I/O 엔진은 Libaio, I/O 깊이는 32로 설정하고 캐쉬는 사용하지 않는다. I/O 확장성을 검증하기 위해 I/O를 발생시키는 프로세스의 수는 1개부터 8개까지 달리 하였으며, 각 I/O 프로세스는 총 1 GBytes의 데이터를 전송한다.

3.4 성능 분석

우리는 제안한 새로운 구조의 성능을 평가하기 이전에 효과를 분석하기 위해 FIO를 이용하여 4KB 랜덤 읽기를 통해 IOPS와 지연 시간(Latency) 그리고 문맥 교환의 발생 횟수를 측정하였다.

첫 번째로 그림 3(a)에서 보는 바와 같이 우리가 제안한 구조는 IOPS를 효과적으로 향상시켰다. 특히 MT는 다수의 I/O 스레드를 통해서 문맥 교환의 오버헤드(Overhead)를 줄임으로써 Baseline과 비교하여 IOPS를 약 167% 향상시켰다. 그림 3(b)를 보면, 모든 지연 속도는 I/O 프로세스들의 수에 비례하여 증가하는 것을 확인할 수 있다. Baseline, MQ, MT 사이에 특이사항은 없지만 Baseline이 MQ와 MT와 비교하여 I/O 프로세스들의 수가 증가할수록 가파르게 지연 속도가 증가하는 것을 이 실험을 통해 확인하였다. 예상했던 것과 같이 MT는 지연 속도를 Baseline과 비교하여 약 59% 감소시켰다.

마지막으로 그림 3(c)에서 보는 것과 같이 단위 시간당 문맥 교환의 발생 횟수를 측정해본 결과, Baseline의 경우 문맥 교환의 발생 횟수가 다른 것과 비교하여 I/O 프로세스들의 수가 증가할수록 급격하게 증가하는 것을 확인할 수 있다. 반면에 MT의 문맥 교환 발생 횟수는 Baseline 대비 약 10% 수준이었다. 이 측정 결과는 단일 요청 큐와 단일 I/O 스레드로 인한 과도한 문맥 교환의 발생이 성능 저하의 주요 원인이었음을 증명한다.

가상화의 성능에 대한 선행 연구에서 가상 머신의 성능을 저하시키는 요인으로 Exit의 오버헤드에 초점을 맞춰왔다. 일반적으로 가상 머신이 동작하는 게스트 모드에서 운영체제의 커널이 동작하는 호스트 모드로 전환될 때 이 Exit이 발생한다. 따라서 우리는 가상화 환경에서 I/O 작업을 처리하는 동안 발생하는 Exit의 횟수와 모든 Exit 이벤트를 처리하는 시간의 총합을 측정하였으며, 그 결과물과 성능의 상관관계를 분석하였다. 그림 4는 성능 카운터(Performance Counter)를 측정하는 리눅스 표준 툴인 PERF[18]를 이용하여 측정된 Exit의 종류별 발생 횟수와 측정 중 발생한 Exit의 총

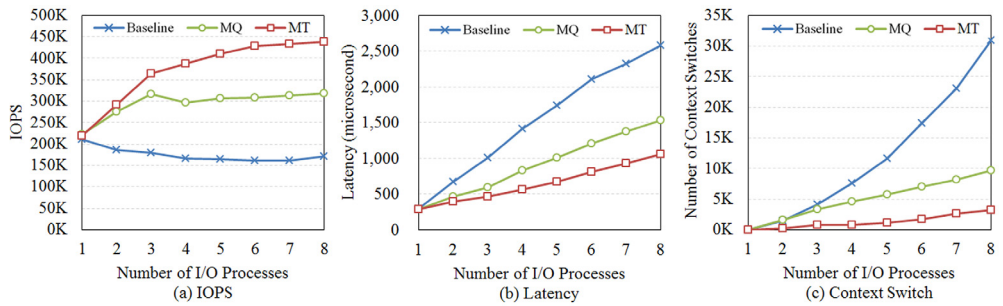

그림 3 Baseline, MQ 그리고 MT 간의 IOPS, 지연 시간, 문맥 교환 발생 횟수의 비교

Fig. 3 Comparison of IOPS, latency, and the number of context switches among Baseline, MQ, and MT

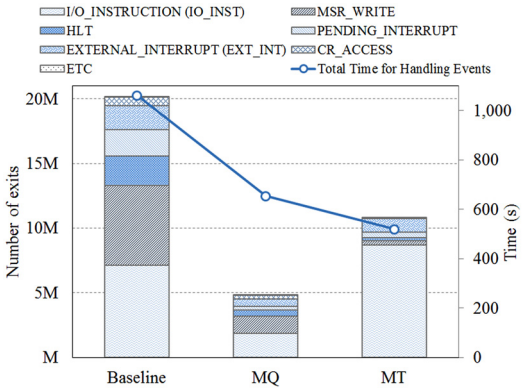

그림 4 Exit의 종류 및 발생 횟수 그리고 Exit의 처리 시간의 총합

Fig. 4 Measurement of the number of Exits and the total time for handling events

수행시간을 나타낸 결과이다. 우선 MT의 경우 Baseline과 비교하여 Exit의 발생 횟수가 약 45% 감소하였다. 이는 주로 MSR_WRITE(Model Specific Register Write), EXT_INT(External Interrupt) 그리고 HLT (Halt)의 감소로 인한 것이며 특히 MSR_WRITE의 경우 약 95% 이상 제거됨을 확인하였다. 반면에 IO_INST (I/O Instruction)는 다소 증가하는 것을 확인할 수 있는데 그 이유는 가상 CPU 전용 큐와 I/O 스레드로 인해 I/O 병렬성이 향상되었기 때문이다. Baseline과 MT만을 비교해보았을 때 Exit 이벤트 처리 시간은 Exit의 발생 횟수에 비례하는 것처럼 보인다. 하지만 MQ의 경우 Exit의 발생 횟수가 MT보다 약 65%정도 현저히 적게 발생했지만 Exit 이벤트 처리 시간과 IOPS 성능은 MT보다 높지 않다. 그 이유는 MQ의 경우 I/O가 발생할 때마다 Exit을 발생시키지 않고 일정 시간 동안 I/O를 모아두었다가 한 번에 처리하기 때문이다. 이 결과로 보아 일반적으로 Exit의 발생 횟수는 I/O 성능에 중요한 영향을 미치지만 절대적인 것은 아니며, 내부적인 I/O 병렬화 또한 I/O 성능에 중요한 역할을 한다는 것을 알 수 있다.

3.5 널 블록 장치에서의 성능

우리는 세 가지 다른 구조인 Baseline, MQ 그리고 MT를 널 블록 장치를 이용하여 성능을 측정하고 비교하였으며, 성능 측정 시 I/O를 발생시키는 프로세스의 수는 1개부터 16개까지 증가시켰다. I/O 프로세스의 수가 가상 CPU와 가상 CPU 전용 큐의 수를 넘는 경우도 검증하기 위해 프로세스의 수를 16개까지 확인했다. 성능은 3.3. FIO 벤치마크에서 언급한 4가지 유형의 I/O 작업에 대해 IOPS와 데이터 처리량의 단위로 측정하였다. 우선 그림 5에서와 같이 Baseline의 경우 I/O 프로

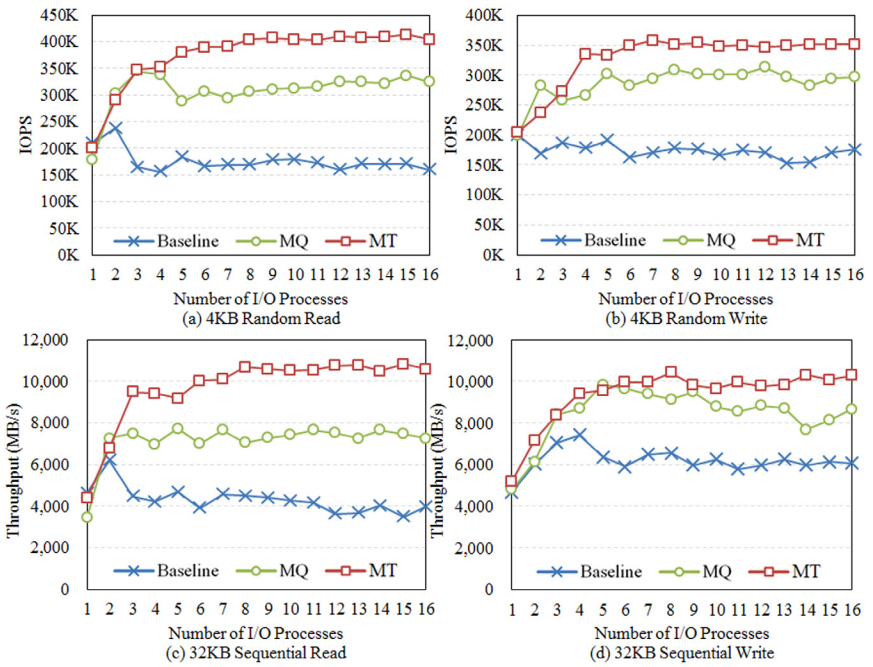

그림 5 4가지 유형의 I/O 작업에 대한 성능 측정 결과

Fig. 5 Performance comparison of four types of I/O workloads among Baseline, MQ, and MIOT using a null block device

세스의 수를 증가시킴에도 불구하고 성능이 점차 감소하였지만, 반면에 MQ와 MT는 점차 증가하였다. 특히 MT의 랜덤 읽기 IOPS(그림 5(a))는 Baseline과 비교했을 때 최대 2.55배, 그리고 MQ 대비해서는 32% 향상되었다. 그리고 다른 I/O 작업 유형에 대한 성능 역시 그림 5(b), 그림 5(c), 그림 5(d)에서 보는 것과 Baseline 대비 각각 128%, 210% 그리고 73% 증가하였다.

우리가 제안한 구조는 뛰어난 성능 향상을 보여주었지만 I/O 프로세스의 수가 8개를 넘어갈 경우 더 이상 성능이 크게 향상되지 않은 현상이 관찰되었다. 이것은 기본적으로 내재된 가상화 계층의 오버헤드로 인해 발생한 것이다. 우리는 옥타코어(Octa-core) CPU가 장착된 시스템에서 동일한 실험을 진행한 결과 쿼드코어 CPU에서 실험한 결과와 거의 동일한 결과를 얻을 수 있었다. 아쉽게도 오늘날까지 소프트웨어 기반 I/O 가상화에서 기본적인 오버헤드를 완벽하게 제거하는 것은 불가능하다. 결과적으로 우리가 제안한 구조는 랜덤 읽기에서 413K IOPS, 랜덤 쓰기에서 358K IOPS, 순차 읽기는 10.5 GB/s, 순차 쓰기의 경우 10.1 GB/s의 성능이 나타났다. 이 실험 결과는 Baseline과 비교할 때 랜덤 읽기를 기준으로 최대 155% 개선된 수치이다.

지금까지 물리적인 CPU 코어의 개수가 가상 머신의 가상 CPU 수와 동일한 경우에 대한 실험만을 진행하였으나 가상 CPU 그리고 가상 CPU 전용 큐 및 I/O 스레드의 수가 물리적인 CPU 코어의 개수보다 많은 경우 어떠한 성능을 보이는지 랜덤 읽기 성능 측정을 통해 검증해보았다. 그림 6은 가상 머신의 가상 CPU를 16개 생성한 후 랜덤 읽기에 대한 성능 평가 결과이며, 그 수치를 보면 Baseline은 8개의 I/O 프로세스 이후 급격하게 I/O 성능이 하락하는 것을 확인할 수 있었다. MQ의 경우도 7개에서 8개의 I/O 프로세스로 동작할 때 최고

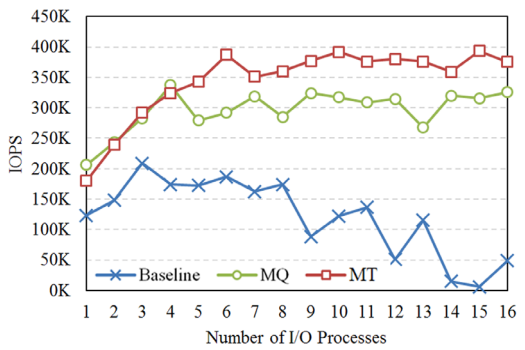

그림 6 가상 CPU의 수가 물리적인 CPU 코어의 수를 넘는 경우에서의 성능 측정

Fig. 6 IOPS when the number of vCPUs exceeds the number of CPU cores

성능을 보여주고 그 후 그 성능을 유지하거나 일부 하락하였다. 반면에 MT는 물리적인 CPU 코어의 개수를 넘는 일부 구간에서 성능을 유지하거나 다수의 구간에서 약 4%에서 9% 향상된 성능이 나타났으며, 그 결과 I/O 프로세스가 15개일 때 약 393K IOPS인 최대 성능을 보여주었다. 결과적으로 다른 두 개의 기법과는 달리 우리가 제안한 기법은 물리적인 CPU 코어를 넘는 I/O 확장 시 성능 하락 없이 지속적으로 성능을 유지함을 확인하였다.

4. 결론 및 향후 연구

우리는 최근 가상화 환경에서 다중 큐 SSD를 보조 기억장치로 사용하였을 경우 가상 머신에서 실질적인 다중 큐 SSD의 성능을 충분히 활용하지 못하고 있음을 관찰하였다. 그 이유는 I/O 가상화 프레임워크에서 다중 큐 SSD로 I/O를 요청할 때 가상 머신에서 발생하는 심각한 잠금 경쟁 때문이다. 게다가 가상 머신들은 호스트에 존재하는 다수의 I/O 큐를 효율적으로 사용하지 못하는 병렬성 문제를 가지고 있다. 이러한 문제들을 해결하기 위해 우리는 가상 CPU 전용 큐와 I/O 스레드로 구성된 새로운 구조를 제안하였다. 또한 우리가 제안한 구조를 널 블록 장치를 대상으로 다양한 종류의 I/O 작업들을 통해 성능을 측정하였으며, 그 결과 최신 I/O 가상화 기술인 Virtio-blk-data-plane과 비교하여 IOPS 성능의 경우 최대 약 2.55배 성능이 향상되었고, 데이터 처리량은 최대 약 210% 많아졌다.

모든 실험은 널 블록 장치를 대상으로 진행되었으며 널 블록 장치가 NVMe의 동작을 거의 유사하게 모사하도록 설계되었지만 실제 NVMe SSD를 대상으로 동일한 실험을 진행할 경우 본 논문과 같은 결과가 나올지 확인이 필요하다. 또한 멀티코어를 넘어 매니코어(Many-core) 환경에서 어느 정도의 성능이 나타나고 또 다른 문제점이 존재하지는 않는지 추후 연구를 진행할 계획이다.

References

[1] D. Abramson, "Intel virtualization technology for directed I/O," *Journal of Intel Technology Journal*, Vol. 10, No. 3, pp. 179-191, 2006.

[2] R. Mijat and A. Nightingale, "Virtualization is coming to a platform near you," ARM White Paper, 2011.

[3] Y. Dong, X. Yang, J. Li, G. Liao, K. Tian, and H. Guan, "High performance network virtualization with SR-IOV," *Journal of Parallel Distributed. Computing*, Vol. 72, No. 1, pp. 1471-1480, 2012.

[4] AMD I/O virtualization technology (IOMMU) specification [Online]. Available: http://developer.amd.

com/wordpress/media/2012/10/488821.pdf (downloaded 2015, Sep. 6)
[5] KVM live migration [Online]. Available: http://www.linux-kvm.org/images/5/5a/KvmForum2007$Kvm_Live_Migration_Forum_2007.pdf (downloaded 2015, Sep. 6)
[6] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, and A. Warfield, "Live migration of virtual machines," *Proc. of 2nd Symposium on Networked System Design & Implementation*, pp. 273-286, 2005.
[7] R. Russell, "VIRTIO: towards a de-facto standard for virtual I/O devices," *ACM SIGOPS Operating Syst. Review*, Vol. 42, No. 5, pp. 95-103, 2008.
[8] A. Kivity, Y. Kamay, D. Laor, UY. Lublin, and A. Liguori, "KVM: the linux virtual machine monitor," *Proc. of Linux Symposium*, pp. 225-230, 2007.
[9] N. Har'El, A. Gordon, A. Landau, M. Ben-Yehuda, A. Traeger, and R. Ladelsky, "Efficient and scalable paravirtual I/O system," *Proc. of USENIX Annual Technical Conference*, pp. 231-242, 2013.
[10] NVM Express specification 1.2 [Online]. Available: http://www.nvmexpress.org/wp-content/uploads/NVM-Express-1_2-Gold-20141209.pdf (downloaded 2015, Sep. 6)
[11] M. Bjorling, J. Axboe, D. Nellans, and P. Bonnet, "Linux block IO: Introducing multi-queue SSD access on multi-core system," *Proc. of the 6th International Systems and Storage Conference*, pp.22:1-22:10, 2013.
[12] F. Bellard, "QEMU, a fast and portable dynamic translator," *Proc. of USENIX Annual Technical Conference*, pp. 41-46, 2005.
[13] J. Axboe. Fio-Fexible IO Tester [Online]. Available: http://freecode.com/projects/fio (downloaded 2015, Sep. 6)
[14] K. Huynh, A. Theurer, and S. Hajnoczi. KVM virtualized I/O performance [Online]. Available: ftp://public.dhe.ibm.com/linux/pdfs/KVM_Virtualized_IOPerformance_Paper.pdf (downloaded 2015, Sep. 6)
[15] M. Lei. Virtio Blk multi-queue conversion [Online]. Available: http://www.linux-kvm.org/wiki/images/6/63/02x06a-VirtioBlk.pdf (downloaded 2015, Sep. 6)
[16] K. Eshghi and R. Micheloni, "SSD architecture and PCI Express interface," Inside Solid State Drives (SSDs), pp. 19-45, 2013.
[17] Null block device driver [Online]. Available: https://www.kernel.org/doc/Documentation/block/null _blk.txt (downloaded 2015, Sep. 6)
[18] Perf: linux profiling with performance counters [Online]. Available: https://perf.wiki.kernel.org (downloaded 2015, Sep. 6)

김 태 용
2008년 동국대학교 컴퓨터공학과 학사
2014년 성균관대학교 DMC공학과 석사
관심분야는 운영체제, 스토리지 시스템, 가상화 기술

강 동 현
2007년 한국산업기술대학교 컴퓨터공학과 학사. 2010년 성균관대학교 전자전기컴퓨터공학과 석사. 2013년~현재 성균관대학교 전자전기컴퓨터공학과 박사과정
관심분야는 스토리지 시스템, 운영체제

엄 영 익
1983년 서울대학교 계산통계학과 학사
1985년 서울대학교 전산과학과 석사
1991년 서울대학교 전산과학과 박사
1993년~현재 성균관대학교 정보통신대학 교수. 관심분야는 시스템 소프트웨어, 운영체제, 가상화, 파일 시스템