The Transactions of the Korea Information Processing Society
TKIPS Vol.13, No.2, pp.48~59

eISSN: 3022-7011
https://doi.org/10.3745/TKIPS.2024.13.2.48

Performance Evaluation and Analysis on Single and Multi-Network Virtualization Systems with Virtio and SR-IOV

Jaehak Lee† · Jongbeom Lim†† · Heonchang Yu†††

ABSTRACT

As functions that support virtualization on their own in hardware are developed, user applications having various workloads are operating efficiently in the virtualization system. SR-IOV is a virtualization support function that takes direct access to PCI devices, thus giving a high I/O performance by minimizing the need for hypervisor or operating system interventions. With SR-IOV, network I/O acceleration can be realized in virtualization systems that have relatively long I/O paths compared to bare-metal systems and frequent context switches between the user area and kernel area. To take performance advantages of SR-IOV, network resource management policies that can derive optimal network performance when SR-IOV is applied to an instance such as a virtual machine(VM) or container are being actively studied.This paper evaluates and analyzes the network performance of SR-IOV implementing I/O acceleration is compared with Virtio in terms of 1) network delay, 2) network throughput, 3) network fairness, 4) performance interference, and 5) multi-network. The contributions of this paper are as follows. First, the network I/O process of Virtio and SR-IOV was clearly explained in the virtualization system, and second, the evaluation results of the network performance of Virtio and SR-IOV were analyzed based on various performance metrics. Third, the system overhead and the possibility of optimization for the SR-IOV network in a virtualization system with high VM density were experimentally confirmed. The experimental results and analysis of the paper are expected to be referenced in the network resource management policy for virtualization systems that operate network-intensive services such as smart factories, connected cars, deep learning inference models, and crowdsourcing.

Keywords : Cloud Computing, Network, Resource Management, Virtualization, Hypervisor, SR-IOV, Virtio

가상화 시스템에서 Virtio와 SR-IOV 적용에 대한 단일 및 다중 네트워크 성능 평가 및 분석

이 재 학† · 임 종 범†† · 유 헌 창†††

요 약

하드웨어 자체적으로 가상화를 지원하는 기능들이 추가됨에 따라 다양한 작업 유형을 가진 사용자 어플리케이션들이 가상화 시스템에서 효율적으로 운용되고 있다. 가상화 지원 기능 중 SR-IOV는 PCI 장치에 대한 직접 접근을 통해 하이퍼바이저 또는 운영체제 개입을 최소화하여 시스템 성능을 높이는 기술로 베어-메탈 시스템 대비 비교적 긴 I/O 경로 및 사용자 영역과 커널 영역에 대한 빈번한 컨텍스트 스위칭 등 가상화 계층의 추가로 낮은 네트워크 성능을 가진 가상화 시스템에서 네트워크 I/O 가속화를 실현하게 해준다. 이러한 성능적 이점을 이용하기 위해 가상머신 또는 컨테이너와 같은 인스턴스에 SR-IOV를 접목할 시 최적의 네트워크 I/O 성능을 도출할 수 있는 네트워크 자원 관리 정책이 활발히 연구되고 있다. 본 논문은 I/O 가속화를 실현하는 SR-IOV의 네트워크 성능을 1) 네트워크 지연 시간, 2) 네트워크 처리량, 3) 네트워크 공정성, 4) 성능간섭, 5) 다중 네트워크와 같은 측면으로 세밀한 성능 평가 및 분석을 Virtio와 비교하여 진행한다. 본 논문의 기여점은 다음과 같다. 첫째, 가상화 시스템에서 Virtio와 SR-IOV의 네트워크 I/O 과정을 명확히 설명했으며, 둘째, Virtio와 SR-IOV의 네트워크 성능을 다양한 성능 메트릭을 기반으로 분석하였다. 셋째, 가상머신 밀집도가 높은 환경에서 SR-IOV 네트워크에 대한 시스템 오버헤드 및 이에 대한 최적화 가능성을 실험으로 확인하였다. 본 논문의 실험 결과 및 분석들은 스마트 팩토리, 커넥티드-카, 딥러닝 추론 모델, 크라우드 소싱과 같은 네트워크 집약적인 서비스들을 운용하는 가상화 시스템에 대한 네트워크 자원 관리 정책에 활용될 것으로 기대된다.

키워드 : 클라우드 컴퓨팅, 네트워크, 자원 관리, 가상화, 하이퍼바이저, SR-IOV, Virtio

※ 본 연구는 산림청(한국임업진흥원) 산림과학기술 연구개발사업 '2022427C10-2224-0801'의 지원에 의하여 이루어진 것임.
† 정 회 원 : 고려대학교 정보창의교육연구소 박사후연구원
†† 종신회원 : 평택대학교 스마트콘텐츠학과 조교수
††† 종신회원 : 고려대학교 컴퓨터학과 교수

Manuscript Received : October 19, 2023
First Revision : December 22, 2023
Accepted : January 20, 2024
* Corresponding Author : Heonchang Yu(yuhc@korea.ac.kr)

1. 서 론

가상화는 하드웨어, 운영체제 기반의 격리 기술을 통해 논리적 컴퓨팅 장치인 가상머신(Virtual Machine, VM) 또는 컨테이너(Container)와 같은 인스턴스 형태로 고성능 컴퓨팅 자원을 공유할 수 있게 해주는 기술이다. 이러한 가상화 기술이 적용된 가상화 시스템은 기존 싱글-테넌트(Single-tenant) 시스템 환경을 멀티-테넨트(Multi-tenant) 환경으로 운영할 수 있게 되면서 가상화는 클라우드 컴퓨팅의 중축 기술로 자리매김하였다. 하지만, 가상화 시스템에서는 물리 자원을 공유하는 사용자 서비스들에 대한 불예측적인 작업 부하, 사용자 영역과 커널 영역 간 빈번한 CPU 컨텍스트 스위칭, 베어-메탈 시스템 대비 긴 I/O 처리 과정 등 가상화 계층의 추가로 I/O 관련 오버헤드가 발생하고 있다[1-3, 7-8, 23-25]. 이러한 가상화 시스템의 오버헤드를 완화하기 위해 다양한 가상화 지원 기술들이 등장하였다.

반가상화(Para-virtualization)는 기존 특권 및 비특권 명령어에 대한 모든 제어 및 빈번한 명령어 재해석(Binary Translation)으로 인한 전가상화(Full virtualization)의 하이퍼바이저 오버헤드를 하이퍼-콜(Hyper-call)이라는 가상 인터럽트 메커니즘을 적용하여 섬세하고 경량화된 명령어 처리 과정을 통해 하이퍼바이저의 역할을 축소 시킴으로써 하이퍼바이저 오버헤드를 완화하였다.

하드웨어 지원 가상화(Hardware Assisted Virtualization, HVM)은 트랩(Trap)과 에뮬레이트(Emulate)를 기반으로 가상머신 운영체제와 하이퍼바이저간의 특권/비특권 명령어 처리에 대한 CPU의 권한 영역 확장을 통해 가상머신의 명령어를 하이퍼바이저가 아닌 CPU 자체에서 처리하는 기술로써 인텔의 VT-x, AMD의 SVM을 예로 들 수 있다. 하지만, 모든 가상머신으로부터 발생하는 비특권 명령어들을 CPU 자체에서 에뮬레이팅하여 처리하므로 CPU 부하가 높게 되어 기존 HVM에 반가상화 I/O 방식을 적용하는 하이브리드형 가상화 기술 또한 상용화되고 있다[4-6].

CPU의 Memory Management Unit(MMU)와 비슷한 역할을 수행하는 I/O Memory Management Unit(IOMMU)는 가상머신의 운영체제에 대한 가상 장치 주소 접근을 물리 장치 주소 접근으로 변환시키는 가상화 지원 장치이며, 인스턴스들은 IOMMU를 통해 PCI 주변 장치로의 직접 접근을 수행할 수 있다. 이는 곧 인스턴스가 주변 장치로의 I/O 수행에 대하여 하이퍼바이저 우회(PCI passthrough)를 가능하게 한다. PCI passthrough는 인스턴스의 메모리 주소에 대하여 PCI 주변 장치와 주메모리 간의 공유 버퍼를 형성하고, Direct Memory Access(DMA)를 통해 데이터 읽기/쓰기, I/O 처리에 대한 CPU 컨텍스트 스위칭을 줄임으로써 하이퍼바이저의 개입을 최소화하여 가상화 시스템의 I/O 성능을 높인다.

PCI Passthrough를 통한 성능적 이점을 기반으로 Network Interface Card(NIC), Graphics Processing Unit(GPU), Solid State Drive(SSD) 등과 같은 PCI 장치에 대한 Virtual Function(VF)이라는 논리 PCI 장치를 하드웨어적으로 구현하여 물리 PCI 장치를 공유할 수 있도록 하는 Single Root I/O virtualization(SR-IOV)[7, 23-25] 기술이 등장했다. 라이브러리 기반의 고속 패킷 처리 API인 DPDK(Data Plane Development Kit)[8]는 시스템의 사용자 영역에서 수행하는 응용 프로그램이 직접적으로 물리 NIC로의 직접 접근을 통해 운영체제 개입을 줄임으로써 성능 향상을 기대할 수 있는 기술 또한 상용화되고 있다.

이러한 가상화를 지원하는 기술들의 등장으로 클라우드 서버에서 다양한 사용자 작업을 처리하는 인스턴스들의 I/O 성능 향상을 위해 가상화 지원 장치를 접목하는 연구가 활발히 진행 중이다. 이 중, 클라우드로의 디지털 트랜스포메이션(Digital Transformation)으로 인하여 다수의 사용자 요청을 처리하는 사용자 서비스들이 증가하고 있는데, 이는 커넥티드-카, 원거리 헬스-케어, 인공지능, 스마트 팩토리, 멀티미디어 스트림등과 같은 데이터 집약적이며, 네트워크 집약적인 서비스들이다.

본 논문은 10GbE NIC가 장착된 Kernel-based Virtual Machine(KVM)[13] 기반 가상화 시스템에서 다양한 성능 메트릭을 기반으로 반가상화 네트워크 I/O(Virtio)과 PCI passthrough 네트워크 I/O(SR-IOV)에 대한 성능 평가 및 분석을 수행한다. 본 논문에서는 Virtio와 SR-IOV에 대한 네트워크 I/O 성능을 지연 시간, 처리량, 공정성, 성능간섭 정도와 같은 성능 메트릭을 기반으로 실험 및 평가를 진행했으며, 다양한 TCP/IP 윈도우 사이즈 크기(4 KB부터 2048 KB), 가상머신의 역할(서버 또는 클라이언트) 및 CPU 오버서브스크립션 등과 같은 가상머신의 네트워크 성능에 영향을 줄 수 있는 요소들을 실험에 개입시켜 명확한 네트워크 성능 분석을 진행한다. 또 한, 본 논문은 가상화 시스템의 네트워크 I/O 과정을 명확히 설명하며, SR-IOV에 대한 네트워크 자원 관리 최적화에 대한 방향성을 실험을 통해 확인한다.

본 논문의 구성은 다음과 같다. 2장에서는 기존 가상화 시스템의 네트워크 I/O 처리 과정에 대하여 자세히 설명하고, 3장에서는 성능 평가 및 그 결과를 분석한다. 4장은 가상화 시스템의 네트워크 성능 분석에 대한 기존 연구를 살펴보고, 5장을 끝으로 본 논문의 결론을 맺는다.

2. 가상화 시스템의 I/O 처리 과정

2.1 가상화

고성능 컴퓨팅 자원을 격리하여 다중 사용자가 공유할 수 있게 해주는 가상화 기술은 기존 시스템에 가상화 소프트웨어 레이어를 추가한 하이퍼바이저 가상화와 커널 공유를 통한 컨테이너 가상화로 구분할 수 있다.

하이퍼바이저 가상화는 하이퍼바이저가 모든 가상머신의 자원 접근을 제어하며, 가상머신에서 발생하는 특권 명령어들

은 하이퍼바이저 자체가 인지(Trap) 및 명령어 재해석 과정(Emulation)을 통해 호스트 시스템에서 인식할 수 있는 명령어로 변환하여 처리한다. 이러한 하이퍼바이저의 역할은 가상머신에게 높은 격리성을 제공하며, 다중 운영체제를 호스트 시스템에서 운용할 수 있게 해준다. 하지만, 가상머신들로부터 발생하는 특권 명령어들을 처리하기 위한 하이퍼바이저의 빈번한 개입은 가상화 시스템 자체의 성능저하를 발생시킨다[1-6, 9].

컨테이너 가상화는 리눅스 커널 자체에서 제공하는 프로세스 단위의 격리된 런-타임 환경인 컨테이너를 기반으로 가상화를 제공한다. 컨테이너를 이루는 리눅스 커널 모듈 중 Cgroup은 프로세스들에 대하여 계층적 그룹을 형성 및 관리하며, 그룹 단위로 프로세스의 I/O 수행 빈도, CPU 및 메모리 점유율 등 프로세스의 자원 점유 정도를 제어한다. 네임스페이스(Namespace)는 실질적으로 프로세스 런-타임 환경을 격리하며 파일 시스템, I/O 장치, 시스템 통신 등에 대한 추상화된 영역을 프로세스에게 할당한다. 만약, 특정 네임스페이스에 속한 프로세스가 한 개이면, 해당 프로세스는 가상화 시스템에서 자기 자신만 수행되는 걸로 인지한다. 호스트 시스템의 운영체제를 공유하는 컨테이너 가상화는 경량화된 격리성(프로세스 단위로의 격리)을 제공함으로 일반적으로 하이퍼바이저 가상화보다 높은 성능을 제공한다. 하지만, 컨테이너 가상화는 하드웨어 수준이 아닌 운영체제 수준에서 격리를 수행하므로 하이퍼바이저 가상화 대비 낮은 격리성을 가지고 있어서 보안 위협이 높으며[10-12], 호스트 시스템 운영체제에 종속적이므로 시스템 호환성이 낮다[10-11].

2.2 KVM

KVM은 오픈소스 전가상화 하이퍼바이저로 리눅스 커널 2.6.20 버전부터 공식 커널 모듈로 통합되었으며, 하드웨어 지원 가상화를 활성화하기 위해 가상화 확장을 지원하는 CPU가 필요하다. KVM은 가상머신의 명령어 집합을 CPU 가상화 확장 기능(Intel-VT, AMD-V)을 통해 명령어 에뮬레이션 과정 없이 직접적인 명령어 처리가 가능하며, QEMU[14]를 통해 I/O 장치(GPU, SSD, NIC 등)를 에뮬레이션하여 가상머신이 실제 물리 I/O 장치가 존재하는 것처럼 인식시킨다. 하지만 여전히 가상머신의 I/O 장치 접근과 같은 특권 명령어들은 트랩(VM Exit/VM Entry)을 통한 에뮬레이션이 필요하므로 오버헤드가 발생한다. 이러한 오버헤드를 완화하고자 KVM은 I/O 장치 접근 시에는 반가상화 I/O 방식인 Virtio를 채택한다.

2.3 KVM의 네트워크 I/O

전가상화는 모든 가상머신으로부터 발생하는 특권 명령어들에 대하여 I/O 장치에 대한 에뮬레이션 수행 및 명령어 재해석 과정을 수행하므로 상당한 시스템 오버헤드가 발생한다. 반가상화는 가상머신 운영체제의 커널 수정을 통해 프론트-엔드 드라이버라는 논리 드라이버를 가상머신에 적재하고, 하이퍼바이저에는 백-엔드 드라이버를 적재하여 비동기적 I/O을 수행하는 분리 드라이버 모델(Split Driver Model)을 구성한다. 논리 드라이버에서는 네트워크 I/O에 대한 인터럽트 서비스 루틴(Interrupt Service Routine, ISR)를 수행하지 않고, 특권 명령어에 대한 재해석 과정을 직접 수행하여 운영체제 통합 시스템 콜 API인 하이퍼 콜을 통해 백-엔드 드라이버에게 전달한다. 하이퍼바이저는 하이퍼 콜을 인지하고, 백-엔드 드라이버를 통해 물리 NIC로의 네트워크 I/O 관련 ISR를 수행한다. 이러한 반가상화의 분리 드라이버 모델은 하이퍼바이저 자체의 I/O 처리 관련 개입을 줄임으로써 가상화 시스템의 오버헤드를 감소시켜 전가상화보다 높은 I/O 성능을 제공한다. 반가상화를 지원하는 하이퍼바이저는 Xen을 시작으로 KVM의 Virtio, VMware의 ESXi, Oracle의 VirtualBox에서도 반가상화 I/O 처리 방식을 지원하게 되었다.

분리 드라이버 모델 기반 반가상화 I/O 방식은 가상머신 운영체제의 커널 드라이버 수정을 통해 전가상화 I/O 대비 경량화된 I/O 과정을 통해 네트워크 성능을 높였다. 하지만, I/O 장치에 대한 하이퍼바이저의 ISR 수행 과정, 프론트-엔드 드라이버와 백-엔드 드라이버 간 I/O 요청 및 응답에 대한 데이터 이동 과정 등은 여전히 가상화 시스템에서 하이퍼바이저의 빈번한 개입을 요구하며, I/O 오버헤드를 발생시킨다. 이러한 문제점을 완화하기 위해 하이퍼바이저를 우회하여 I/O을 수행하는 기술인 PCI passthrough가 등장했다. PCI passthrough는 특정 I/O 장치 드라이버와 가상머신의 프론트-엔드 드라이버의 I/O 버퍼를 맵핑하여 DMA 통신을 통해 빠른 데이터 I/O을 가능하게 한다. 하지만, I/O 장치 자체의 가상화 기능은 제공하지 않아 오직 하나의 인스턴스만이 I/O 장치에 접근할 수 있다.

이후, 다중 인스턴스들이 PCI passthrough를 통해 I/O 장치를 공유할 수 있도록 하드웨어적으로 I/O 장치 자체를 가상화하는 기술인 SR-IOV가 개발되었다. SR-IOV의 구성은 PF(Physical function)와 VF로 구분된다. PF는 실질적인 물리 PCI 장치이며, 하이퍼바이저에 의해 에뮬레이션되어 하이퍼바이저의 외부 네트워크 통신에 이용되는 네트워크 어댑터 역할을 한다. PF 드라이버는 하이퍼바이저에 적재되어 SR-IOV에 대한 전체적인 자원(가상머신에 대한 VF 할당과 반환, VF에 대한 MAC 주소 할당 등)을 관리하는 특권 계층이다. 루트 도메인(Xen의 드라이버 도메인(Dom 0) 또는 KVM의 호스트 운영체제 등)만 접근 가능하다.

VF는 PF의 부분적인 자원이며. 레지스터(Rx/Tx 버퍼, 메일박스 등)로 구성된다. VF 드라이버는 가상머신에 적재되며, VF의 Rx/Tx 버퍼와 직접적인 DMA 통신으로 데이터를 송수신한다. VF 드라이버의 특수 기능(VF MAC 주소 설정, VLAN 필터 설정, 멀티 케스트 주소 설정 등)을 변경할 시에는 메일박스(Mailbox) 메카니즘을 통해 PF 드라이버와 통신하여 변경할 수 있다.

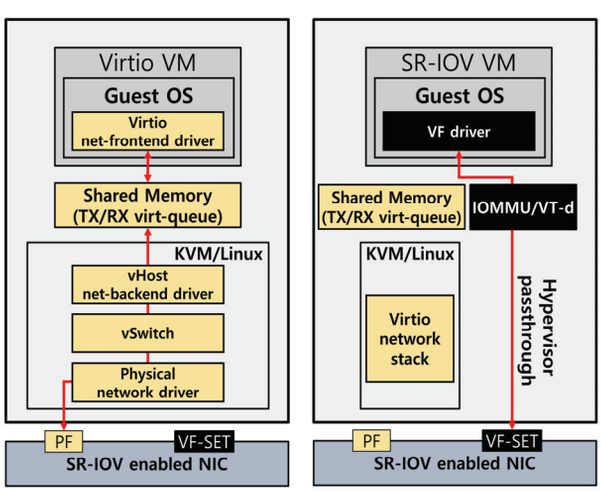
2.4 Virtio와 SR-IOV의 네트워크 I/O 과정

본 절에서는 가상화 시스템의 네트워크 I/O 과정을 상세히 설명하며, Fig. 1은 KVM 기반 가상화 시스템에서의 네트워크 I/O 과정을 보여준다. Fig. 1(a)는 분리 드라이버 모델을 기반으로 반가상화 네트워크 I/O을 수행하는 Virtio 동작 과정, Fig. 1(b)는 하이퍼바이저 우회를 통한 네트워크 I/O을 수행하는 SR-IOV의 동작 과정을 나타낸다.

Fig. 1(a)에 대하여, 물리 NIC는 외부 네트워크로부터 패킷이 수신되면 대상 가상머신으로 패킷을 전달하기 위해 하이퍼바이저로의 인터럽트를 발생시킨다. 해당 인터럽트를 인지한 QEMU 프로세스는 물리 CPU에 스케줄링 되면 KVM의 벡-엔드 드라이버의 공유 링 버퍼에 해당 패킷을 적재하고, 백-엔드 드라이버는 목적지 가상머신의 프론트-엔드 드라이버로 가상 인터럽트를 발생시킨다. 목적지 가상머신의 가상 CPU가 물리 CPU를 점유하여 가상 인터럽트의 존재를 확인하게 된다면, 공유 링 버퍼에서 수신된 패킷 기반으로 ISR을 수행한다. 가상머신이 네트워크 I/O을 요청하는 경우는 가상머신의 프론트-엔드 드라이버의 공유 링 버퍼에 전송할 패킷을 적재시킨 후, 백엔드 드라이버로의 가상 인터럽트를 발생시킨다. 그 후, QEMU 프로세스가 스케줄링되어 백-엔드 드라이버에 요청 패킷이 존재한다면 물리 NIC로의 ISR를 수행한다.

Fig. 1(b)에 대하여, 물리 NIC는 외부로부터 패킷이 수신되면 패킷을 L2 Sorter/Switch로 보낸다. 그 후, VF 대한 Mac 주소 테이블을 참고하여 목적지 VF의 Rx 버퍼에 패킷을 적재하여 VF 드라이버와 맵핑된 목적지 가상머신의 메모리로 DMA 통신을 수행한다. 패킷의 데이터가 모두 송신된 후, NIC는 하이퍼바이저로의 VF 인터럽트(MSI 또는 MSI-X)를 발생시킨다. 목적지 가상머신의 가상 CPU가 물리 CPU를 점유하면 VF 드라이버는 패킷이 존재할 시 해당 패킷에 대한 ISR을 수행한다.

가상머신에서 네트워크 I/O을 요청할 시 패킷은 직접적으로 VF 드라이버를 통해 VF의 Tx 버퍼에 패킷을 적재한 후 하이퍼바이저에게 가상 인터럽트를 발생시킨다. 호스트 영역에서 수행 중인 QEMU 프로세스가 물리 CPU를 점유하여 가상 인터럽트의 존재를 확인하면 TX 버퍼에 적재된 패킷을 외부로 송신하기 위한 ISR를 수행한다.



(a) Virtio network I/O path (b) SR-IOV network I/O path

Fig. 1. Network I/O Procedure of Virtio and SR-IOV

3. 성능 평가 수행 및 분석

본 장에서는 KVM 가상화 시스템에서 SR-IOV의 네트워크 성능을 Virtio와 비교하여 성능 평가를 진행한다. 실험에 사용되는 물리 시스템은 총 2대이며, 같은 네트워크에 속한 상태로 KVM 가상화 시스템과 비가상화 시스템으로 구성한다. 가상화 시스템은 SR-IOV 지원 10 GbE NIC을 통해 네트워크 통신을 수행한다. 성능 평가는 8대의 가상머신들에 대하여 다양한 성능 메트릭을 기반으로 실험 환경을 구성하여 진행한다. 성능 평가에 대한 네트워크 성능 메트릭은 1) 네트워크 지연시간, 2) 네트워크 대역폭, 3) 네트워크 공정성, 4) 성능간섭, 5) 다중 네트워크로 구성된다.

Table 1은 가상화 시스템과 비가상화 시스템에 대한 사양을 나타내며, 모든 가상머신은 4개의 가상 CPU와 2048 MB의 메모리 크기를 가진다. 가상화 시스템에 대하여, 작업 부하 정도에 따라서 최대 32개의 가상 CPU가 10개의 물리 CPU를 공유한다. 가상화 시스템과 비가상화 시스템 모두 리눅스 커널에서 허용하는 최대 TCP 윈도우 사이즈를 4096 KB까지 설정할 수 있도록 가상화/비가상화 시스템에 네트워크를 구축했다.

3.1 네트워크 지연 시간

본 실험은 Virtio와 SR-IOV에 대하여 가상머신의 역할(클라이언트 또는 서버)을 기반으로 CPU 작업 정도에 따른 네트

Table 1. Experiment Environment

	Virtualization System	Non-Virtualization System
CPU	Intel core i9- 10900k CPU @ 3.70GHZ x 10 without Hyperthreading	Intel core i9- 9900k CPU @ 3.40GHZ x 8 without Hyperthreading
MEMORY	DDR4 32GB	DDR4 32GB
NIC	Intel X540-T2 (10GbE) with ixgbe-5.18.13	Intel 1219-LM(1GbE)
OS	Ubuntu 20.04 LTS with Linux 5.15.0-43-generic	Ubuntu 20.04 LTS with Linux 5.15.0-43-generic
Hypervisor	KVM 6.0	-
VM	vCPU: 4, RAM: 2048MB, OS: Ubuntu 20.04 LTS with Linux 5.15.0-43-generic	-

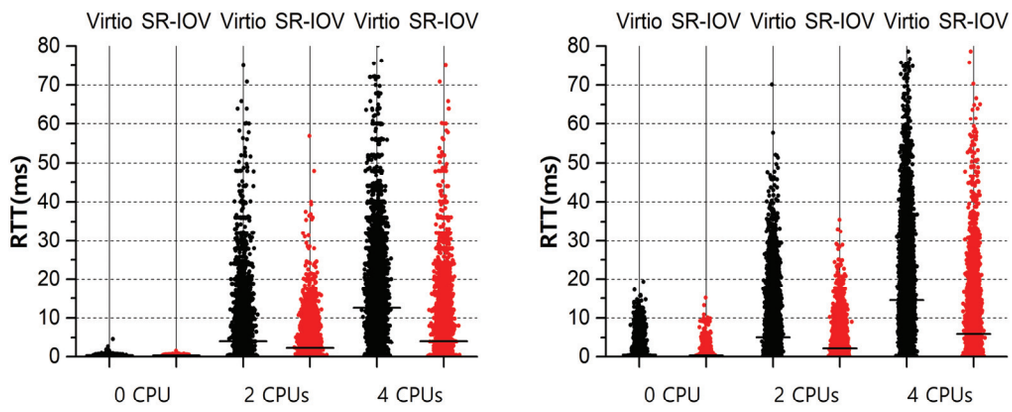


Fig. 2. Experimental Results for RTT(ms) based on CPU-intensive Degree

워크 지연 시간을 비교한다. 네트워크 지연 시간 측정은 리눅스에서 제공하는 벤치마크 툴인 Ping[15]을 이용했으며, 총 3000번의 Ping을 0.2 ms마다 수행하여 Round Trip Time (RTT, ms)를 측정한다. CPU 집약적인 작업은 stress[16] 툴을 이용하여 생성한다. Fig. 2는 8대의 가상머신들에 대하여 3000번의 Ping 수행에 대한 RTT 분포를 나타내며, Fig. 2(a)는 가상머신이 클라이언트로, Fig. 2(b)는 서버로 동작할 때의 실험 결과를 나타낸다. Fig. 2에 대하여, 0 CPU 상태는 CPU 집약적인 작업이 없는 상태를 의미하며(오직 Ping만 수행하는 상태), 2 CPUs 상태는 두 개의 가상 CPU에 CPU 집약적인 워크로드가 발생한 상태, 4 CPUs 상태는 모든 가상 CPU에 CPU 집약적인 작업이 발생한 상태를 의미한다.

Fig. 2(a)에 대하여, 오직 Ping만 수행(0 CPU 상태)시 평균 RTT는 Virtio가 0.38 ms, SR-IOV는 0.31 ms로 비교적 적은 차이를 보인다. 하지만, 2 CPUs 상태에서의 평균 RTT는 Virtio가 4.11 ms, SR-IOV는 2.19 ms로 SR-IOV의 평균 RTT가 Virtio보다 약 46% 감소했다. 4 CPUs 상태일 때는 SR-IOV가 4.11 ms로, 12.51 ms인 Virtio보다 67.15%의 평균 RTT가 감소된 것을 확인할 수 있다. Fig. 2(b)의 0 CPU 상태에서의 평균 RTT는 Virtio가 0.51 ms이고, SR-IOV는 0.28 ms로 SR-IOV가 Virtio보다 약 45.1% 감소했다. Fig. 2(b)의 2 CPUs 상태와 4 CPUs 상태에서 Virtio가 각 5.07 ms, 14.47 ms의 평균 RTT를 가지고, SR-IOV는 각 2.09 ms, 5.96 ms의 평균 RTT를 가진다. 이는 곧, 2 CPUs 상태와 4 CPUs 상태에서 SR-IOV의 평균 RTT가 Virtio 대비 58.78%, 58.81% 감소 한 것을 확인할 수 있다. 실험 결과, 전체적으로 SR-IOV가 Virtio 대비 현저히 낮은 네트워크 지연 시간을 가진다.

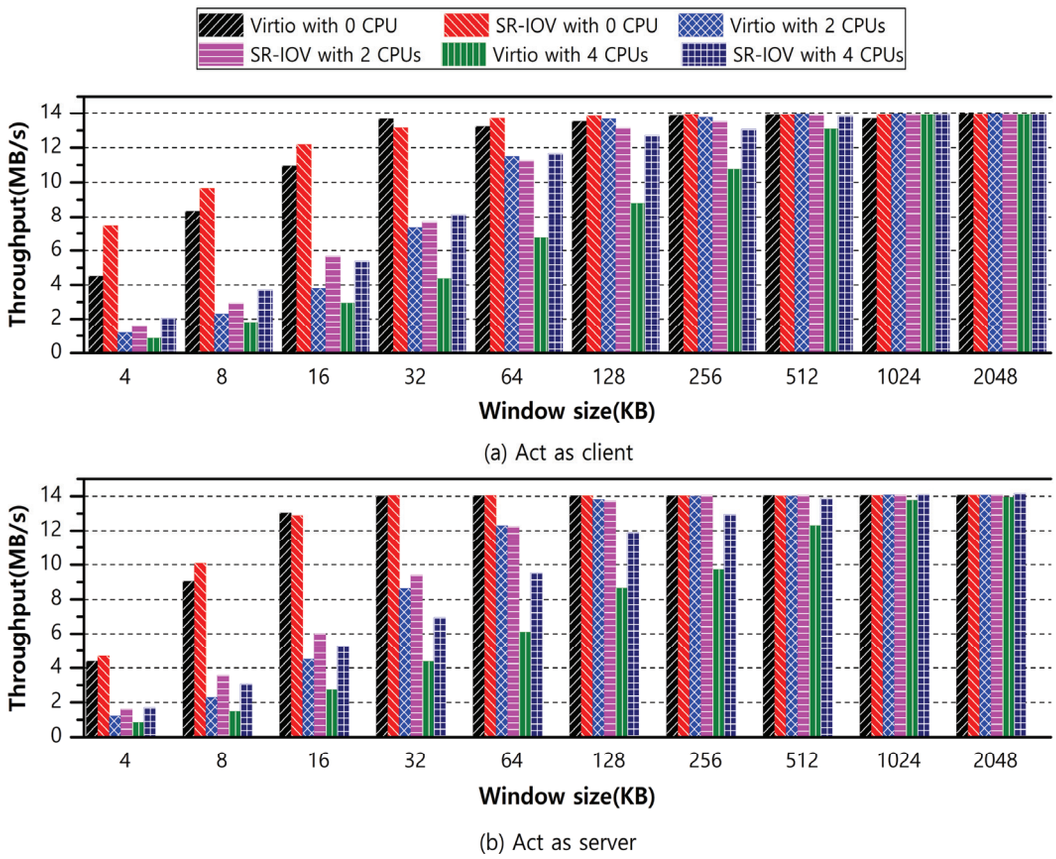
3.2 네트워크 처리량

본 실험은 TCP/IP 통신 시 다양한 윈도우 사이즈 크기를 기반으로 SR-IOV의 네트워크 처리량을 Virtio와 비교한다. 네트워크 성능을 측정하기 위해 벤치마크 툴 Iperf[17]를 이용했으며, 3.1 절과 같이 stress 툴을 이용하여 CPU 집약적인 작업을 발생시키도록 한다. Fig. 3은 8대의 가상머신에 대하여 60초 동안의 TCP/IP 통신을 윈도우 사이즈에 따라 총 15번 수행한 결과에 대한 평균 네트워크 처리량(MB/s)을 나타내며, 가상머신의 역할에 대하여 Fig. 3(a)는 클라이언트, Fig. 3(b)는 서버로 수행할 때의 실험 결과이다.

Fig. 3(a)와 (b)에서, 윈도우 사이즈 크기가 4 KB, 8 KB 일 시 전체적으로 낮은 네트워크 처리량이 발생하는데, 그 이유는 작은 TCP 버퍼 크기로 인하여 패킷 처리 과정(Tx/Rx에 대한 데이터 이동 등)이 빈번하게 발생하기 때문이다. 하지만 전체적인 실험 결과에 대하여 SR-IOV가 Virtio보다 높은 네트워크 처리량을 가진다. 특히, 4 CPUs 상태에서 윈도우 사이즈 크기가 32 KB, 64 KB일 때, Virtio와 SR-IOV의 평균 네트워크 처리량 차이가 제일 큰 것을 확인할 수 있다. Fig. 3(a)의 4 CPUs 상태에서 윈도우 사이즈 크기가 32 KB, 64 KB일 때 Virtio는 4.37 MB/s와 6.10 MB/s, SR-IOV는 6.94 MB/s와 9.52 MB/s의 네트워크 처리량을 가짐으로 SR-IOV가 Virtio 대비 각 58.81%, 56.07%만큼 네트워크 대역폭이 향상되었다. 가상머신들이 서버로 동작하는 Fig. 3(b)에서의 4 CPUs 상태에서는 Virtio가 3.13 MB/s, 6.75 MB/s이고, SR-IOV는 8.13 MB/s, 11.65 MB/s로 SR-IOV가 각 159.34%, 72.59%만큼 네트워크 처리량이 증가했다.

Fig. 3에 대해서, 4 CPUs 상태에서 SR-IOV가 Virtio 대비 네트워크 대역폭 향상 폭이 높은 것을 확인할 수 있다. 그 이유는 32개의 가상 CPU가 10개의 물리 CPU를 두고 경쟁하는 CPU 오버서브스크립션(CPU oversubscription) 상태에서 물리 CPU 경쟁 및 컨텍스트 스위칭 발생 비율 증가로 Virtio에서 발생하는 I/O 요청 및 응답에 대한 하이퍼바이저의 ISR이 지연 시간을 가지게 되기 때문이다.

윈도우 사이즈가 클수록 Virtio와 SR-IOV의 네트워크 대역폭은 비슷한데, 이는 TCP 버퍼의 크기가 커지면서 TCP/IP 통신 관련 I/O 인터럽트가 발생 비율이 낮아지면서 컨텍스트 스



(a) Act as client

(b) Act as server

Fig. 3. Experimental Results for Network Throughput(MB/s) According to Window Size and CPU-intensive Degree

위칭 빈도 또한 낮아져 SR-IOV의 PCI passthrough의 DMA 통신에 대한 성능적 이점이 낮아지기 때문이다.

3.3 네트워크 성능 공정성 정도 측정

본 실험은 Virtio와 SR-IOV의 네트워크 성능의 공정성 정도를 평가하기 위해, 8대의 가상머신에 대하여 60초 동안의 TCP/IP 통신(Iperf)을 수행하여 데이터 전송량(MB)을 비교하도록 한다. Fig. 4는 모든 가상머신에 대하여 CPU 집약 정도에 따른 총 15회에 TCP/IP 통신에 대한 데이터 전송량(MB) 분포를 나타낸다. SR-IOV의 PCI passthrough 기반 네트워크 I/O 처리는 대체적으로 반가상화 I/O 기반 Virtio 보다 높은 네트워크 성능을 제공하지만, 가상머신간 네트워크 성능에 대한 공정성은 크게 향상되지 않은 것으로 확인된다. SR-IOV는 Virtio 대비 적은 CPU 타임 슬라이스 소비와 네트워크 I/O 관련 인터럽트 발생 빈도, DMA를 기반한 패킷 읽기/쓰기 최소화 등으로 하이퍼바이저의 개입을 최소화하였음에도 불구하고, 가상머신들의 물리 NIC에 대한 공정한 접근은 보장되고 있지 않다.

이에 대한 이유는 SR-IOV의 네크워크 I/O 과정 중 가상머신의 VF 드라이버의 Tx/Rx 버퍼와 물리 NIC Tx/Rx 버퍼 간의 패킷 이동을 위한 VF 인터럽트 및 Tx/Rx 버퍼의 읽기/쓰기를 위한 가상 인터럽트는 여전히 하이퍼바이저의 개입이 필요하기 때문이다. 하이퍼바이저의 ISR를 수행하는 프로세스 또한 가상머신들의 가상 CPU 간의 물리 CPU 경쟁으로 스케줄링 지연 시간이 발생하게 되면서 불공정한 네트워크 성능이 발생한다. 리눅스의 디폴트 운영체제 스케줄러인 CFS 스케줄러는 실질적인 물리 CPU에 대한 점유 정도를 의미하는 가상-런타임을 기준으로 가상 CPU를 스케줄링하는데, 적은 가상-런타임을 가진 가상 CPU를 레드-블랙 트리 기반으로 우선적으로 스케줄링한다. 이러한 CFS 스케줄러의 vCPU 스케줄링 정책은 빈번한 I/O 대기 상태로 인해 CPU 집약적인 vCPU보다 적은 가상-런타임을 가지는 I/O 집약적인 가상 CPU에 대하여 물리 CPU 점유를 보장해 줌으로써 가상머신의 I/O 성능을 높일 수 있다[18, 19]. 하지만, Fig. 4(e)부터 Fig. 4(l)를 보면 Fig. 4(a)부터 Fig 4(d)보다 데이터 전송량 분포 정도가 고르지 못한 것을 확인할 수 있다. 이에 대한 이유는 I/O 집약적인 작업을 CPU 집약적인 작업과 동시에 처리하게 됨으로 가상 CPU의 가상-런타임 증가율이 높아지게 되어 I/O 집약적인 가상 CPU의 물리 CPU 점유 보장 정도가 낮아지게 된다. 이는 곧 CFS 스케줄러에 불예측적인 I/O 성능을 발생시킨다.

3.4 성능간섭 정도 측정

본 실험은 SR-IOV와 Virtio 네트워크에 대하여 CPU 집약적인 가상머신들로부터 발생하는 성능간섭 정도를 측정 및 분

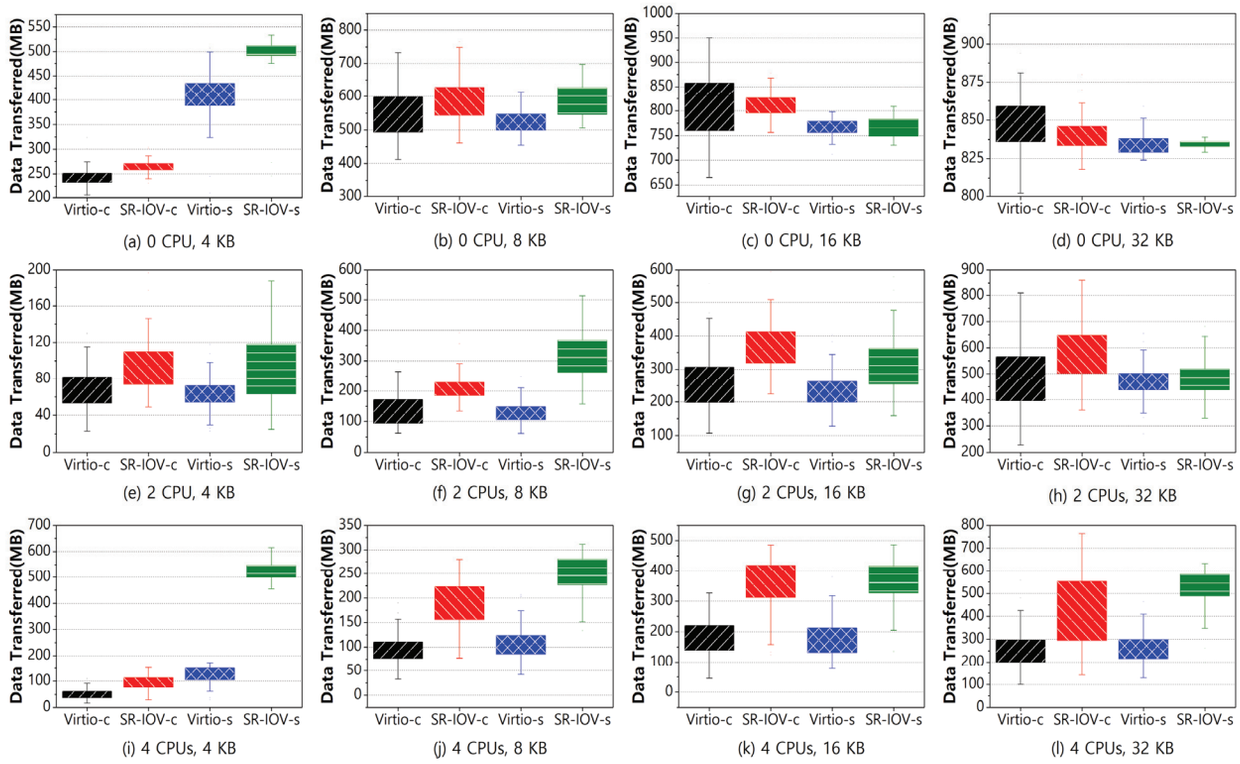


Fig. 4. Data Transmission Amount(MB) Distribution based on CPU-intensive Degree for TCP/IP Communication

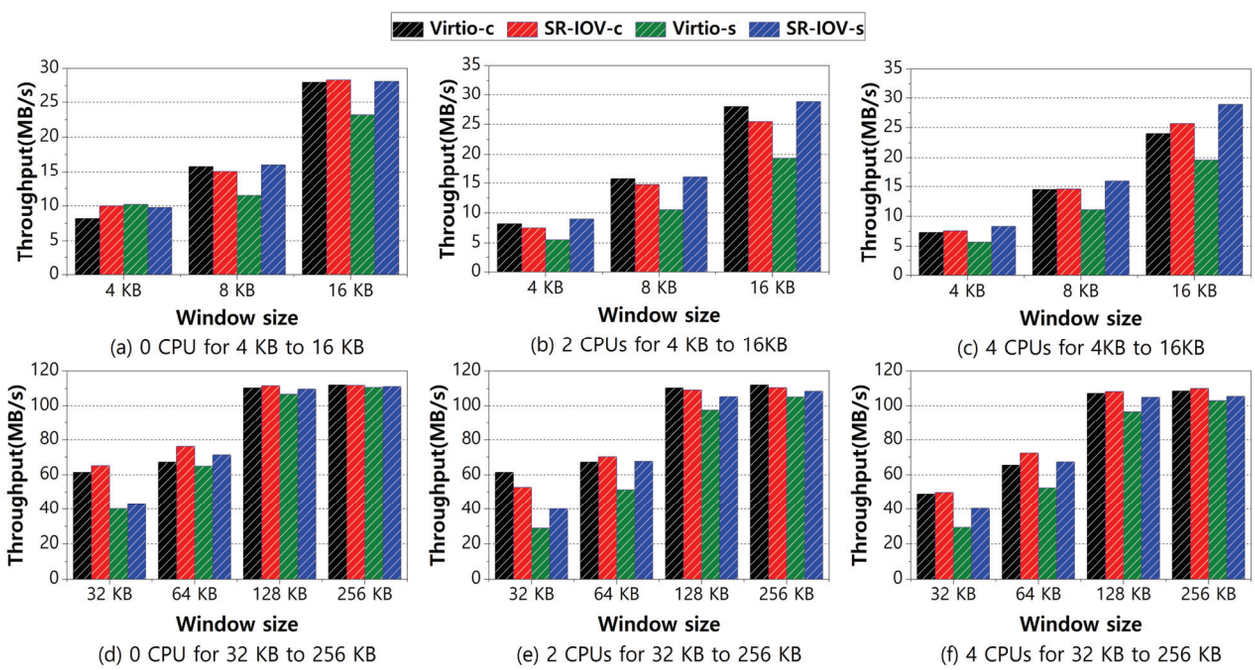


Fig. 5. Experimental Results on the Network Throughput(MB/s) of one VM According to the CPU-intensive Degree of Adjacent Virtual CPUs

석한다. Fig. 5는 7대의 가상머신이 CPU 집약적인 작업(stress)을 처리 중일 때, 1대의 가상머신에 대하여 총 15회에 대한 1분 동안의 TCP/IP 통신(Iperf) 평균 네트워크 처리량(MB/s)을 나타낸다. Fig. 5에 대하여, 네트워크 집약적인 가상머신이 클라이언트로 동작할 때(Virtio-c, SR-IOV-c), SR-IOV의 네트워크 대역폭은 Virtio와 비슷한 네트워크 처리량을 가진다. 4

CPUs 상태에서는 SR-IOV의 네트워크 처리량이 약간 더 높은 것을 확인할 수 있다. 네트워크 집약적인 가상머신이 서버로써 동작할 때(Virtio-s, SR-IOV-s)는 Virtio보다 SR-IOV가 더 높은 네트워크 처리량을 가진다.

Fig. 5에서 가상머신이 클라이언트로 동작할 때 SR-IOV의 네트워크 성능 향상 정도가 적은 이유는 다음과 같다. 가상머신에서 외부 네트워크로의 패킷을 전송하기 위해선 가상 인터럽트(I/O 요청 이벤트)를 발생시켜 하이퍼바이저가 물리 NIC의 Tx 버퍼로 패킷을 이동시켜야 한다. 하지만, CPU 집약적인 인접 가상 CPU들과의 물리 CPU 경쟁으로 인해 I/O 집약적인 가상 CPU는 스케줄링 지연 시간이 발생하게 된다. 결국, 가상머신의 네트워크 I/O 요청은 지연 시간을 가지게 되면서 SR-IOV의 PCI passthrough에 대한 성능적 이점을 기대하기 어렵게 된다. 서버로 동작하는 가상머신이 패킷을 수신할 때는 하이퍼바이저가 물리 NIC로부터의 VF 인터럽트 발생을 인지하여 즉시 수신된 패킷을 DMA 통신으로 목적지 가상머신에게 전달할 수 있으므로 Virtio보다 더 좋은 네트워크 대역폭을 보장할 수 있다.

3.5 다중 네트워크

본 실험은 VF 공유 정도에 따른 SR-IOV의 네트워크 성능을 단일 네트워크 환경과 다중 네트워크 환경으로 구분하여 측정한다. 성능 평가를 진행하기 전 가상머신의 특성을 네트워크 성능 관련 QoS가 낮은 가상머신과 높은 가상머신으로 구분한다. 가상머신 1부터 가상머신 5는 네트워크 QoS가 낮은 가상머신 집합(Low-QoS 집합)으로, 가상머신 6부터 가상머신 8까지는 높은 집합(High-QoS 집합)으로 정의한다. 가상화 시스템의 네트워크 환경은 모든 가상머신이 SR-IOV 네트워크를 사용하는 단일 네트워크 환경과 Virtio와 SR-IOV 네트워크가 동시에 사용되는 다중 네트워크 환경으로 구분한다. 다중 네트워크 환경에 대해서 가상머신 1부터 가상머신 5는 Virtio 네트워크를 할당하고, 가상머신 6부터 8까지는 SR-IOV 네트워크를 할당한다.

Fig. 6은 단일 네트워크 환경(Single)과 다중 네트워크 환경(Multi)에 대하여, Low-QoS 집합과 High-QoS 집합에 대한 총 15회에 대한 TCP/IP 통신(Iperf) 평균 네트워크 처리량을 나타낸다. Fig. 6(a)부터 Fig. 6(c)는 가상머신이 클라이언트로, Fig. 6(d)부터 Fig. 6(f)는 서버로 동작했을 때의 실험 결과를 나타낸다.

Fig. 6에 대해서, 단일 네트워크 환경은 가상머신 1부터 가상머신 8까지 SR-IOV 네트워크를 할당받았으므로 Low-QoS 집합과 High-QoS 집합의 네트워크 처리량 차이가 크지 않다. 다중 네트워크 환경에서는 High-QoS 집합이 Low-QoS 집합보다 전체적으로 높은 네트워크 처리량을 가지며, 단일 네트워크 환경의 High-QoS 집합보다 다중 네트워크 환경에서의 High-QoS 집합이 더 높은 네트워크 처리량을 가진다.

Fig. 6(a)의 High-QoS 집합의 평균 네트워크 처리량에 대해서, 윈도우 사이즈가 128 KB부터 다중 네트워크 환경은 단일 네트워크 환경보다 더 높은 네트워크 대역폭을 확보한 것을 확인 할 수 있다. 특히, 128 KB에서 단일 네트워크 환경은 14.01 MB/s, 다중 네트워크 환경은 26.24 MB/s로, 다중 네트워크 환경이 단일 네트워크 환경보다 약 87.37% 더 높은 네트워크 처리량을 가진다. Fig. 6(b)와 Fig. 6(c)에서는 CPU 작업 부하로 인해 전체적인 네트워크 대역폭은 줄어들지만, 다중 네트워크 환경의 High-QoS 집합이 여전히 단일 네트워크 환경의 High-QoS 집합보다 높은 네트워크 대역폭을 확보하고 있다.

Fig. 6(d)에서, 0 CPU 상태일 때는 패킷 이동이 빈번하게 발생하는 윈도우 사이즈(4, 8, 16 KB) 구간에서는 다중 네트워크 환경의 High-QoS 집합의 네트워크 처리량이 더 높지만, 그 외의 구간에선 단일 네트워크 환경의 High-QoS 집합과 네트워크 대역폭이 비슷하다. 하지만, 2 CPUs와 4 CPUs 상태와 같이 CPU 작업 부하 정도가 높아질수록 다중 네트워크 환경의 High-QoS 집합이 단일 네트워크 환경보다 더 높은 네트워크 대역폭 가진다. 2 CPUs 상태에서의 High-QoS 집합의 네트워크 대역폭에 대해서 윈도우 사이즈가 64 KB일 때 단일 네트워크 환경은 11.70 MB/s, 다중 네트워크 환경은 14.72 MB/s로 다중 네트워크 환경이 약 28.81% 높다. 4 CPUs 상태에선 윈도우 사이즈가 128 KB 일 때, High-QoS 집합의 네트워크 대역폭에 대하여 다중 네트워크 환경은 11.70 MB/s로 14.72 MB/s인 단일 네트워크 환경보다 약 25.81% 높다.

SR-IOV의 PF 드라이버의 중재하에 VF 대역폭은 가중 평균 방식으로 할당되는데 개별 VF 대역폭은 NIC 자체의 대역폭에 종속되어 있으며, 할당된 VF의 수가 너무 많을수록 각 VF에 할당될 수 있는 유효 대역폭 또한 줄어들기 때문이다. 이는 곧, SR-IOV는 제한적인 자원이며 VF 할당 정도에 따라 해당 시스템에 대한 SR-IOV 성능이 결정되므로, 섬세한 네트워크 자원 관리가 필요하다는 것을 의미한다.

Fig. 7은 Fig. 6의 단일 네트워크 환경과 다중 네트워크 환경에 대한 SR-IOV 네트워크 성능이 도출되는 과정을 나타낸다. SR-IOV는 IOMMU/VT-d와 PCI Passthrough를 통해 PCI 장치에 대한 가상 주소 및 물리 주소 변환 과정, DMA 기반의 데이터 이동으로 데이터 이동 부문에 대한 성능적 이점을 얻었다. 하지만, 데이터 제어 부문을 고려하면 Fig. 6과 같이 VF를 할당받은 가상머신이 증가할수록 요구되는 SR-IOV 네트워크 대역폭을 기대하기 어렵다.

Fig. 7(a)는 단일 네트워크 환경의 데이터 제어 부문에서의 시스템 오버헤드가 발생하는 상태를 나타낸다. 기존 리눅스 커널의 라운드 로빈(Round-Robin) 기반 인터럽트 처리 과정을 단순히 적용한 하이퍼바이저는 SR-IOV 가상머신이 증가할수록 VF 인터럽트 발생 비율 또한 선형적으로 증가하므로 SR-IOV 가상머신들에 대한 인터럽트 처리 지연 시간 또한 증가하게 된다. VF 인터럽트에 대한 하이퍼바이저의 ISR과 가상 인터럽트에 대한 가상머신 운영체제의 ISR은 모두 ISR 프로세

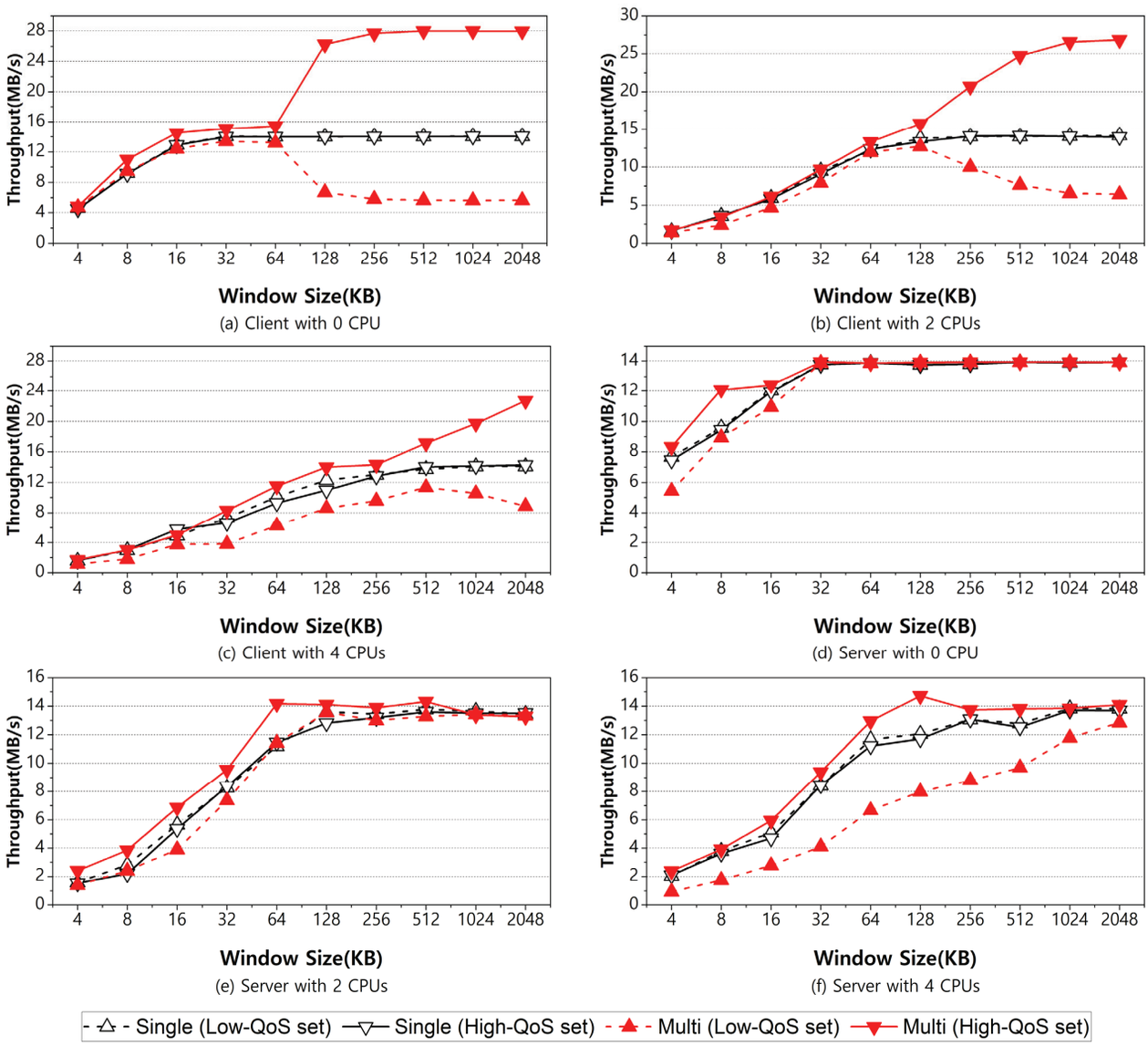


Fig. 6. Experimental Results on Average Network Throughput(MB/s) for Single Network Environment(SR-IOV) and Multiple Network Environment(Virtio+SR-IOV)

스가 물리 CPU를 점유해야지만 처리가 된다. VF 인터럽트 요청의 증가는 “인터럽트 요청은 매우 짧은 시간 안에 처리될 것”이라는 기존 리눅스 운영체제 철학을 충족시키지 못하면서, 네트워크 I/O 처리 수행에 대한 물리 CPU 경쟁이 발생하여 I/O을 수행하는 프로세스에 대한 스케줄링 지연 시간이 늘어나 SR-IOV의 네트워크 성능은 보장되기 힘들다.

Fig. 7(b)는 기존 Virtio와 SR-IOV 네크워크를 혼합해서 사용하는 경우의 가상머신들에 대한 네트워크 I/O 경로를 나타낸다. 기존 VF를 할당한 가상머신은 SR-IOV 네트워크 I/O을 수행하고, VF를 할당받지 않은 가상머신은 Virtio 네트워크 I/O을 수행한다. 이 경우, Virtio 네트워크를 할당받은 가상머신들은 커널 네트워크 스택을 이용하게 되므로 낮은 네트워크 성능을 가지지만, SR-IOV 네트워크를 이용하는 가상머신은 단일 네트워크 환경보다 VF 인터럽트에 대한 ISR 수행 기회가 높아지게 되면서 물리 NIC의 VF를 더욱 빈번하게 점유할 수 있게 된다.

이러한 다중 네트워크 환경 구성은 제한적인 자원을 가진 가상화 시스템에서 SR-IOV의 네트워크 대역폭을 최대한 보장할 수 있는 메커니즘을 제공할 수 있을 것으로 기대된다. 이 경우, 가상화 시스템에서 구동 중인 가상머신들에 대하여 특정 시간대의 네트워크 부하 정도에 따라 네트워크 집약적이지 않은 가상머신에게는 Virtio 네트워크를, 네트워크 집약적인 가상머신에게는 SR-IOV 네트워크를 동적으로 할당함으로써 네트워크 작업 부하 정도에 적응적인 자원 할당을 통해 효율적인 시스템 네트워크 최적화를 수행할 수 있을 것으로 기대된다.

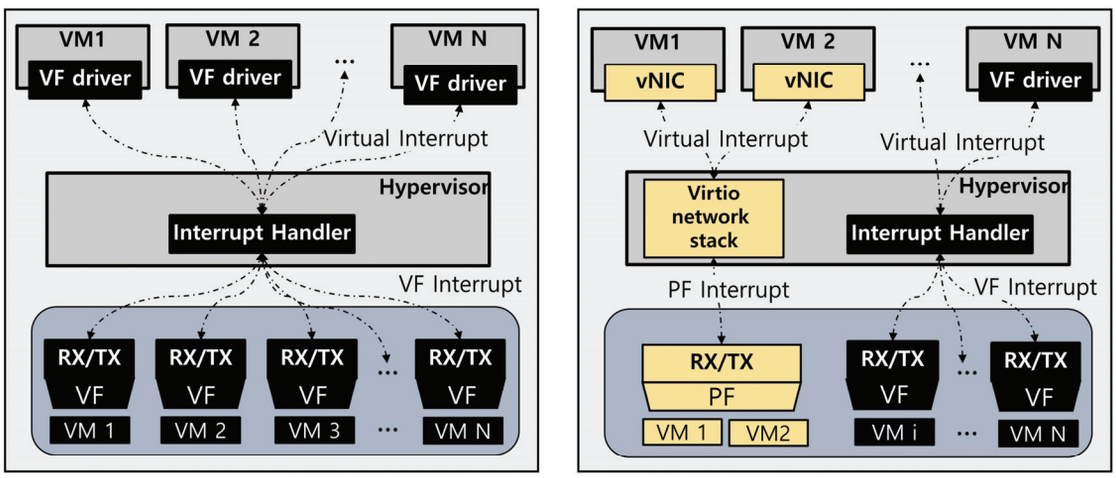


Fig. 7. Data Path in a Single-network Environment and Multi-network Environment

4. 관련 연구

소프트웨어 접근 방식의 반가상화 I/O과 하드웨어 접근 방식의 SR-IOV에 대하여 다양한 성능 분석 연구들이 진행되었다[2, 9, 20-27].

[2, 9, 20-22]은 Xen과 KVM 기반의 가상화 시스템에서 반가상화 I/O과 같은 소프트웨어적 접근 방식으로 가상화 시스템의 특성에 따른 세부적인 네트워크 성능을 분석하였으며, [2, 9]은 반가상화와 전가상화에 대한 네트워크 성능을 분석하였다. 하지만, 상위 연구들은 하드웨어적인 접근 방식에서의 네트워크 I/O에 대한 성능 분석은 진행하지 않았다.

[20-22]과 같이 비가상화 시스템과 가상화 시스템에서의 소프트웨어 접근 및 하드웨어적 접근 방식에 대한 네트워크 성능 분석에 대한 연구가 진행되었다. [23-24]은 SR-IOV의 네트워크 성능을 CPU 점유율 정도에 따른 CPU 병목현상으로 인한 시스템 오버헤드를 분석하였으며, 가상화 시스템의 네트워크 자원 최적화 방안을 설명하였다. [25-27]은 네트워크와 디스크 I/O 성능을 다양한 벤치마크 프로그램을 통해 비가상화 시스템과 가상화 시스템에서 반가상화 I/O에 대한 SR-IOV에 대한 성능적 이점을 네트워크 성능 뿐만 아니라, CPU 점유율, 메모리 접근 빈도, 인터럽트 발생 비율 등 다양한 성능 메트릭을 기반으로 평가 실험을 진행하였다 하지만, 상위 연구들은 네트워크 성능보다는 가상화 특성에 따른 시스템 오버헤드에 초점을 두어 실험을 진행했으며, 다중 가상머신이 아닌 단일 가상머신을 기반으로 성능 분석을 진행하였다.

본 논문은 SR-IOV 지원 10 GbE NIC가 장착된 KVM 기반 가상화 시스템에서 PCI passthrough를 통해 네트워크 I/O 가속화를 실현하는 SR-IOV의 네트워크 성능을 1) 네트워크 지연 시간, 2) 네트워크 처리량, 3) 네트워크 공정성, 4) 성능간섭, 5) 다중 네트워크로 구분하여 Virtio와 비교하여 성능 평가 및 분석을 진행했다. 더욱 세밀한 네트워크 성능 평가를 위해 가상머신에 대하여 클라이언트와 서버 역할로 구분하여 네트워크 성능을 분석했다. 또 한, 단일 가상머신이 아닌 다중 가상머신을 운용하여 가상머신 밀집도에 따른 공유 자원에 대한 경쟁 정도를 기반으로 가상화 시스템의 네트워크 성능 평가를 진행했다.

5. 결 론

본 논문은 KVM 가상화 시스템에서 10 GbE NIC를 이용하여 반가상화 네트워크 I/O을 수행하는 Virtio와 PCI passthrough 네트워크 I/O을 수행하는 SR-IOV에 대한 네트워크 성능을 다양한 실험 환경에서 성능 평가를 진행하였다. 평가 결과, 전체적으로 SR-IOV의 네트워크 성능이 Virtio보다 높으며, 특히 CPU 작업 부하가 높은 시스템 환경에서 네트워크 향상 정도가 높게 나타났다. SR-IOV가 하이버파이저/커널 우회를 통한 DMA 기반 네트워크 I/O 수행을 통해 가상머신이 NIC에 직접 접근을 가능하게 해줌으로써 하이퍼바이저의 I/O 개입 정도가 상당히 감소된 것을 확인하였다. 하지만, Virtio 대비 하이퍼바이저의 I/O에 대한 개입을 줄였음에도 불구하고 여전히 네트워크 불공정성 현상은 남아있으며, 해결해야 할 과제로 남아있다.

마지막으로, 시스템 네트워크 환경에 대하여 SR-IOV만을 적용한 단일 네트워크 환경을 이용할 시 VF 인터럽트 과부하, 네트워크 I/O 처리에 대한 CPU 경쟁 등으로 네트워크 I/O에 대한 ISR 지연 시간 증가로 기대되는 SR-IOV의 네트워크 성능 보장이 어려움을 확인했다. 이에 대하여, Virtio와 SR-IOV를 같이 사용하는 다중 네트워크 환경에 대한 성능 평가를 통해, 네트워크 집약적인 가상머신에게는 SR-IOV를 할당하고 그렇지 않은 가상머신에게는 Virtio를 할당함으로써 네트워크

데이터 흐름을 분산시켜 SR-IOV의 네트워크 성능을 최대한 보장할 수 있을 것으로 기대된다.

본 논문의 성능 평가 및 분석을 통해 스마트 팩토리, 커넥티드-카, 인공지능 추론, 크라우드 소싱과 같은 네트워크 집약적인 서비스들을 운용하는 가상화 시스템에 대한 네트워크 자원 관리 정책연구에 활용될 것으로 기대된다.

References

[1] Y. Dong, X. Yang, J. Li, G. Liao, K. Tian, and H. Guan, "High performance network virtualization with SR-IOV," *Journal of Parallel and Distributed Computing*, Vol.72, No.11, pp.1471-1480, 2012.

[2] G. Motika and S. Weiss, "Virtio network paravirtualization driver: Implementation and performance of a de-facto standard," *Computer Standards & Interfaces*, Vol.34, No.1, pp.36-47, 2012.

[3] X. Hu, J. Li, R. Ma, and H. Guan, "ES2: Building an efficient and responsive event path for I/O virtualization," *IEEE Transactions on Cloud Computing*, Vol.10, No.2, pp.1358-1372, 2020.

[4] J. Nakajima et al., "Optimizing virtual machines using hybrid virtualization," In: *Proceedings of the 2011 ACM Symposium on Applied Computing*, pp.573-578, 2011.

[5] Y. Dong, X. Zhang, J. Dai, and H. Guan, "HYVI: a hybrid virtualization solution balancing performance and manageability," *IEEE Transactions on Parallel and Distributed Systems*, Vol.25, No.9, pp.2332-2341, 2013.

[6] S. C. Mhatre and P. Chandran, "On making xen detect hypercalls and memory accesses for simulating virtualization-enabled processors," In: *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, pp.154-161, 2020.

[7] Y. Dong, Z. Yu, and G. Rose, "SR-IOV Networking in Xen: Architecture, Design and Implementation," In: Workshop on I/O virtualization, 2008.

[8] I. Cerrato, M. Annarumma, and F. Risso, "Supporting fine-grained network functions through Intel DPDK," In: *2014 Third European Workshop on Software Defined Networks*, IEEE. pp.1-6, 2014.

[9] H. Fayyad-Kazan, L. Perneel, and M. Timmerman, "Full and para-virtualization with Xen: a performance comparison," *Journal of Emerging Trends in Computing and Information Sciences*, Vol.4, No.9, pp.719-727, 2013.

[10] S. Sultan, I. Ahmad, and T. Dimitriou, "Container security: Issues, challenges, and the road ahead," *IEEE Access*, Vol.7, pp.52976-52996, 2019.

[11] M. Bélair, S. Laniepce, and J.-M. Menaud, "Leveraging kernel security mechanisms to improve container security: a survey," In: *Proceedings of the 14th International Conference on Availability, Reliability and Security*, pp.1-6, 2019.

[12] K. Brady, S. Moon, T. Nguyen, and J. Coffman, "Docker container security in cloud computing," In: *2020 10th Annual Computing and Communication Workshop and Conference (CCWC)*, IEEE. pp.0975-0980, 2020.

[13] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori, "kvm: the Linux virtual machine monitor," In: *Proceedings of the Linux symposium*, pp.225-230, 2007.

[14] F. Bellard, "QEMU, a fast and portable dynamic translator," In: *USENIX Annual Technical Conference*, FREENIX Track. pp.46, 2005.

[15] *ping*, Accessed in September, 2023. [Internet], https://manpages.ubuntu.com/manpages/jammy/en/man1/ping.1.html

[16] *stress*, Accessed in September, 2023. [Internet], http://manpages.ubuntu.com/manpages/jammy/en/man1/stress.1.htmls

[17] *iperf*, Accessed in September, 2023. [Internet], http://manpages.ubuntu.com/manpages/jammy/en/man1/iperf.1.html

[18] W. Jia, J. Shan, TO. Li, X. Shang, H. Cui, and X. Ding, "{vSMT-IO}: Improving {I/O} Performance and Efficiency on {SMT} Processors in Virtualized Clouds," In: *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, pp.449-463, 2020.

[19] K. Suo, Y. Zhao, J. Rao, L. Cheng, X. Zhou, and FCM. Lau, "Preserving i/o prioritization in virtualized oses," In: *Proceedings of the 2017 Symposium on Cloud Computing*, p.269-281, 2017.

[20] S. G. Soriga and B. Mihai, "A comparison of the performance and scalability of Xen and KVM hypervisors," *2013 RoEduNet International Conference 12th Edition: Networking in Education and Research*, IEEE, 2013.

[21] G. Motika and W. Shlomo, "Virtio network paravirtualization driver: Implementation and performance of a de-facto standard," *Computer Standards & Interfaces*, Vol.34, No.1, pp.36-47, 2012.

[22] S. A. Babu, M. J. Hareesh, J. P. Martin, S. Cherian, and Y. Sastri, "System performance evaluation of para virtualization, container virtualization, and full virtualization using xen, openvz, and xenserver," In *2014 Fourth International Conference on Advances in Computing and Communications*, IEEE, pp.247-250, 2014.

[23] M. Fischer and W. Florian, "Survey on SR-IOV performance," *Network*, Vol.43, 2021.

[24] A. T. de Oliveira Filho, E. Freitas, PRX. do. Carmo, DFH. Sadok, and J. Kelner, "Measuring the impact of SR-IOV and virtualization on packet round-trip time," *Computer Communications*, Vol.211, pp.193-215, 2023.
[25] R. Shea and L. Jiangchuan, "Network interface virtualization: challenges and solutions," *IEEE Network*, Vol.26, No.5, pp.28-34, 2012.
[26] J. Liu, "Evaluating standard-based self-virtualizing devices: A performance study on 10 GbE NICs with SR-IOV support," *2010 IEEE International Symposium on Parallel & Distributed Processing (IPDPS)*, IEEE, 2010.
[27] R. Ganesan, Y. Murarka, S. Sarkar, and K. Frey, "Empirical study of performance benefits of hardware assisted virtualization," *Proceedings of the 6th ACM India Computing Convention*, 2013.

이 재 학

https://orcid.org/0000-0001-5206-0257
e-mail : jaecrane2@gmail.com
2017년 한국공학대학교 컴퓨터공학과 (학사)
2023년 고려대학교 컴퓨터학과 (박사/석·박사통합)
2023년 ~ 현 재 고려대학교 정보창의교육연구소 박사후연구원
관심분야 : Cloud Computing, Virtualization, Distributed System, Resource Optimization

임 종 범

https://orcid.org/0000-0001-8954-2903
e-mail : jblim@ptu.ac.kr
2009년 백석대학교 정보통신학부(학사)
2011년 고려대학교 컴퓨터교육과(석사)
2014년 고려대학교 컴퓨터교육과(박사)
2015년 ~ 2017년 동국대학교 IT융합교육센터 초빙교수
2017년 ~ 2021년 한국공학대학교 게임공학부 조교수
2021년 ~ 현 재 평택대학교 스마트콘텐츠학과 조교수
관심분야 : Distributed & Cloud Computing, Resource Management, Artificial Intelligence Application

유 헌 창

https://orcid.org/0000-0003-2216-595X
e-mail : yuhc@korea.ac.kr
1989년 고려대학교 컴퓨터학과(학사)
1991년 고려대학교 컴퓨터학과(석사)
1994년 고려대학교 컴퓨터학과(박사)
1998년 ~ 현 재 고려대학교 컴퓨터학과 교수
관심분야 : Cloud Computing, Virtualization, Distributed System