가상화 플랫폼에서 네트워크 I/O를 지원하기 위한 구조

진현욱, 김종서
건국대학교

요 약

가상화 기술은 하나의 물리 머신에서 다수의 가상 머신을 제공하고 이들이 상호 독립적으로 수행될 수 있도록 한다. 현재 가상화는 클라우드 서버 시스템의 활용률을 높이고 전력 소모를 줄이기 위한 기술로서 각광받고 있다. 최근 이러한 가상화 기술을 스마트 임베디드 디바이스에도 적용하려는 시도가 있다. 하지만 임베디드 시스템은 일반적으로 서버 시스템에 비해서 성능이 낮은 하드웨어를 사용하므로 가상화 오버헤드에 대한 고찰이 필요하다. 본고에서는 기존에 널리 사용되는 대표적인 공개 소스 가상화 플랫폼들을 대상으로 네트워크 I/O를 지원하는 방식에 대해서 설명하고 이들의 성능을 비교한다. 또한 가상화 환경에서 네트워크 I/O 성능을 향상시키기 위한 최근 동향에 대해서도 함께 논의한다.

Ⅰ. 서 론

가상화 기술은 하나의 물리적인 시스템에서 다수의 가상 시스템 환경을 제공하고 이들이 자원 활용과 소프트웨어 오류에 대해서 상호 독립적으로 수행될 수 있도록 한다 [1]. 따라서 가상 시스템에서 수행되는 운영체제 및 상위 소프트웨어는 독립된 시스템에서 수행되고 있는 것과 같은 투명성을 제공받을 수 있다. 이러한 가상화 기술은 하나의 응용 도메인만을 담당하던 서버 시스템이 여러 응용 도메인을 책임지고 이들을 독립적으로 운영하는 것을 가능하게 했다. 그 결과 현재 가상화는 서버 시스템의 활용률을 높이고 전력 소모를 줄이기 위한 기술로서 각광받고 있다 [2].

최근 이러한 가상화 기술을 다양한 시스템에 적용하려는 시도가 진행되고 있다. 대표적인 응용 분야로는 차량용 인포테인먼트 시스템 [3][4], 스마트 모바일 디바이스 [5][6], 스마트 가전기기 [7] 등과 같은 스마트 임베디드 시스템이 있다. 가상화 기술은 이들 시스템에서 여러 소프트웨어 플랫폼을 지원함으로써 사용자의 다양한 응용 프로그램에 대한 요구사항을 만족시킬 수 있다. 또한 보안성을 향상시킬 수 있는 장점을 갖는다. 하지만 임베디드 시스템은 일반적으로 서버 시스템에 비해서 성능이 낮은 하드웨어를 사용하므로 가상화 오버헤드가 더욱 치명적일 수 있다. 따라서 이들을 위한 가상화 오버헤드에 대한 분석과 최적화가 다양하게 진행돼야 한다.

스마트 임베디드 시스템들의 중요 서비스는 대부분 네트워크에 기반하고 있다. 하지만 가상화로 인한 추가 오버헤드는 네트워크 I/O 성능을 크게 저하 시킨다 [8]. 본고에서는 기존에 널리 사용되는 Xen, VirtualBox와 같은 대표적인 공개 소스 가상화 플랫폼들이 네트워크 I/O를 지원하는 구조에 대해서 설명하고 이들의 성능을 비교한다. 이를 통해서 기존의 가상화 기술들을 스마트 임베디드 시스템에 적용할 때 발생하는 네트워크 I/O의 성능적 영향에 대한 이해를 도우려 한다. 또한 최근에는 가상화 환경에서 네트워크 I/O 성능의 한계를 인지하고 여러 연구자들이 다양한 성능 향상을 시도하고 있다. 본고는 이러한 최근 동향에 대해서도 함께 논의하고자 한다.

본고는 다음과 같이 구성되어 있다. 서론에 이어 2장에서는 가상화에 대한 기본적인 소개와 함께 가상화 기법의 분류에 대해서 설명한다. 3장에서는 서로 다른 가상화 플랫폼들이 네트워크 I/O을 지원하기 위해서 사용하는 구조를 비교 설명한다. 4장에서는 가상화 환경에서 네트워크 I/O 성능을 향상시키기 위한 최근 기술들의 동향에 대해서 설명한다. 마지막으로 5장에서 결론을 맺는다.

Ⅱ. 가상화 플랫폼 개요

하나의 물리 시스템에 다수의 가상 시스템 환경을 제공해주는 가상화의 핵심 기능은 VMM (Virtual Machine Monitor) 또는 하이퍼바이저 (Hypervisor)라고 불리는 소프트웨어 계층에 의해서 수행된다. 그리고 VMM에 의해서 생성된 가상 시스템 환경은 가상 머신 (Virtual Machine; VM)이라고 불린다. VM

에서 수행되는 소프트웨어 실행 환경은 게스트 도메인 (Guest Domain)이라고 하고, 게스트 도메인의 운영체제를 게스트 운영체제 (Guest OS)라고 한다.

일반적으로 가상화 기법은 게스트 운영체제의 수정을 필요로 하는가의 여부에 따라서 전가상화 (Full-Virtualization)와 반가상화 (Paravirtualization)로 나누어진다. 전가상화는 게스트 운영체제의 수정을 전혀 요구하지 않고 기존의 소프트웨어를 그대로 수행시킬 수 있다는 장점을 갖는다. 하지만 이를 위해서 CPU, I/O 인터페이스 등의 에뮬레이션 오버헤드가 발생한다. 반면에 반가상화는 이러한 오버헤드를 줄이기 위해서 에뮬레이션을 하지 않는 대신 게스트 운영체제와 디바이스 드라이버는 VMM이 제공해주는 서비스를 명시적으로 사용하도록 수정되어야 한다. 이들 VMM 서비스는 하이퍼콜 API (Hypercall API)를 통해서 사용하도록 되어 있다.

또한 VMM의 구현 방식에 따라 Type-1과 Type-2로 구분할 수 있다. Type-1 VMM은 하드웨어 위에서 독립적으로 수행 가능한 구현으로 베어 메탈 (Bare-Metal) 기반의 가상화라고 불리기도 한다. 반면에 Type-2 VMM은 호스트 운영체제 (Host OS)라고 불리는 기반 운영체제 위에서 수행되어 호스트형 (Hosted) 가상화라고 불린다. 일반적으로 Type-1은 호스트 운영체제를 요구하지 않으므로 Type-2보다 경량화 된 시스템을 구성할 수 있는 가능성을 갖는다. 하지만 물리 디바이스를 접근하기 위한 드라이버 모두를 VMM 내부에 구현하면 VMM의 크기가 커지고 복잡도가 증가할 수 있는 문제점이 있다. 따라서 현존하는 대부분의 Type-1 VMM 구현들은 특권을 갖는 게스트 도메인을 별도로 두고 이것이 물리 디바이스를 접근하는 드라이버를 구현하도록 하고 있다.

현존하는 대표적인 VMM 구현들로는 Citrix Xen, VMware ESX/Workstation, Oracle VirtualBox, Microsoft Hyper-V 등이 있다. Xen은 Type-1 VMM으로 반가상화 방식의 PV (ParaVirtualization) 모드와 전가상화 방식의 HVM (Hardware Virtual Machine) 모드를 지원한다. VMware는 ESX와 Workstation 제품군이 있다. VMware ESX는 Type-1 전가상화 방식을 지원한다. 최근에는 전가상화 방식의 오버헤드를 줄이기 위해서 VMM에서 VMI (Virtual Machine Interface)를 제공하여 필요 시 게스트 운영체제 및 디바이스 드라이버가 이용할 수 있도록 하고 있다. VMware Workstation과 VirtualBox는 Type-2 전가상화 VMM을 제공한다. Hyper-V는 VMware ESX와 같이 Type-1 전가상화를 구현한다. 즉, 현존하는 가상화 플랫폼 대부분은 표 1과 같이 Type-1 반가상화, Type-1 전가상화, Type-2 전가상화 중의 한 형태를 취하고 있다.

표 1. 가상화 플랫폼 분류

	전가상화	반가상화
Type-1	Xen-HVM VMware ESX Hyper-V	Xen-PV
Type-2	VirtualBox VMware Workstation	

Ⅲ. VMM에 따른 네트워크 I/O 구조 비교

본 장에서는 대표적인 VMM 구현의 예로서 VirtuaBox (Type-2 전가상화), Xen-HVM (Type-1 전가상화), Xen-PV (Type-1 반가상화)의 네트워크 I/O 지원 구조를 비교하고 이들의 성능을 측정한다. 이들 VMM은 다양한 네트워크 I/O 지원 방식을 동시에 구현하고 있으나, 본고에서는 브리지 기반의 네트워크 I/O 지원 방식을 기본 논의의 대상으로 한다. 이 방식은 가상 머신마다 고유의 IP 주소를 설정하도록 하고 동일한 물리 노드 내에서의 통신과 서로 다른 물리 노드 간의 통신을 모두 지원한다. 또한 네트워크 클라이언트 프로그램은 물론이고 서버 프로그램을 투명하게 실행시킬 수 있다.

〈그림 1〉은 VirtualBox에서 네트워크 I/O를 지원하기 위한 구조를 보인다. VirtualBox의 VMM은 게스트 도메인에게 가상 PCI 주소 공간을 제공하여 게스트 도메인의 디바이스 드라이버가 실제 물리 PCI 디바이스를 제어하는 것과 같은 환경을 제공한다. VMM이 모든 PCI 네트워크 디바이스의 고유 구조를 에

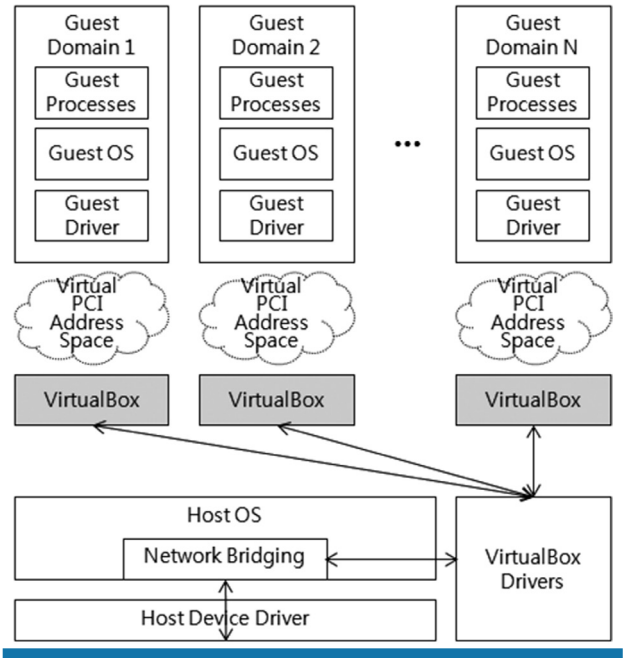

그림 1. VirtualBox의 네트워크 I/O 구조

뮬레이션하기는 힘들기 때문에 몇몇 네트워크 디바이스에 대해서만 지원하도록 구현하고 있다.

네트워크 데이터의 송수신 경로는 다음과 같다. 게스트 디바이스 드라이버의 송신 요청을 VMM이 인지하면 호스트 운영체제에 설치된 VirtualBox 드라이버의 도움으로 이를 호스트 운영체제의 브리지에 전달하게 된다. 이후 브리지의 정책에 의해서 물리 네트워크 디바이스로 전송하게 된다. 수신의 경우 호스트 운영체제에 수신된 패킷은 브리지와 VirtualBox 드라이버를 통해서 해당 가상 머신에게 전달된다. 이 때 VMM은 게스트 디바이스 드라이버가 등록한 게스트 운영체제 메모리 영역에 수신된 패킷을 위치하고, CPU 에뮬레이션에 의해서 게스트 운영체제의 네트워크 인터럽트 핸들러를 호출한다.

〈그림 2〉는 Xen의 전가상화 모드에서 네트워크 I/O를 지원하기 위한 구조를 보인다. 대부분의 구조는 VirtualBox의 경우와 동일하다. 하지만 II장에서 언급한 바와 같이 Xen은 Type-1 VMM이기 때문에 물리 디바이스를 접근하기 위한 드라이버는 특권을 갖는 게스트 도메인(Domain 0)에 포함되어 있다. Domain 0는 물리 디바이스를 접근할 수 있는 권한을 갖고 있기 때문에 그림에서와 같이 VMM의 도움 없이 디바이스를 직접 접근할 수 있다.

〈그림 3〉은 Xen의 반가상화 모드에서 네트워크 I/O를 지원하기 위한 구조를 보인다. 전반적으로 전가상화 모드와 비슷하나, Domain 0와 다른 게스트 도메인 (Domain U)간에 네트워크 I/O 제어 정보를 교환하기 위해서 가상 PCI 주소 공간 대신에 I/O 링을 사용한다. I/O 링은 Xen VMM이 가상 머신 간 통신을 위해서 제공하는 공유 메모리와 이벤트 하이퍼콜을 사용하여 구현되어 있다. I/O 링을 사용하기 위해서 게스트 디바이스 드라이버(Front-end Driver)는 수정되어야 하지만 PCI 주소 공간 및 PCI 디바이스의 기능을 에뮬레이션 하는 오버헤드를 피할 수 있다.

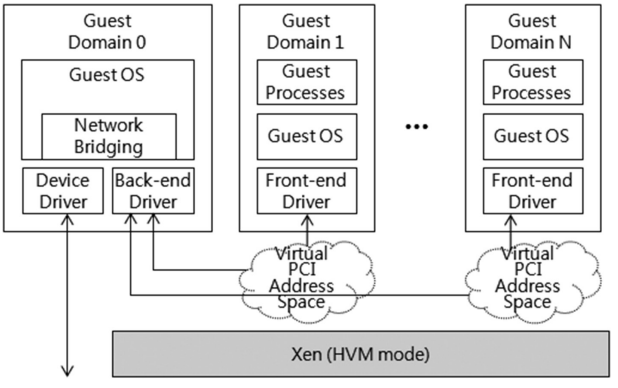

그림 2. Xen 전가상화 모드의 네트워크 I/O 구조

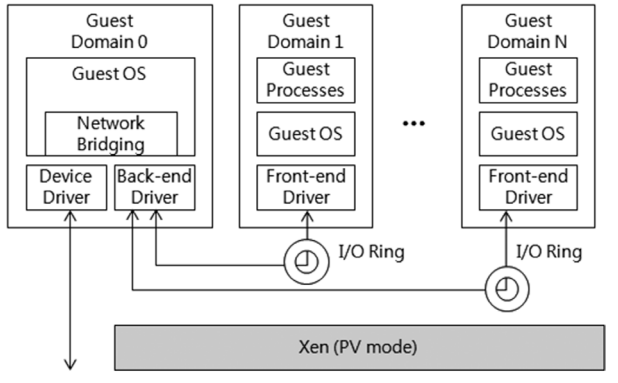

그림 3. Xen 반가상화 모드의 네트워크 I/O 구조

앞에서 언급된 VMM들의 통신 성능을 비교하기 위해서 두 노드를 Gigabit Ethernet으로 연결하고 통신 성능을 측정하였다. 두 노드 중 인텔 i3 프로세서를 장착한 산업용 임베디드 보드에만 가상화를 적용하였다. 대응 노드는 인텔 i7 프로세서를 장착한 PC를 사용하였으며 가상화를 적용하지 않았다. VMM은 VirtualBox 4.1.2와 Xen 4.1을 사용했으며, 호스트/게스트 도메인 또는 Domain 0/U는 모두 Ubuntu 11.10 (리눅스 커널 버전 3.0.24)을 사용했다. 성능 측정은 ttcp 벤치마크를 이용하여 종단 응용 프로그램 간의 TCP/IP 통신 왕복 지연시간과 TCP/IP 최대 통신 대역폭을 측정하였다.

〈그림 4〉는 서로 다른 VMM 환경에서의 통신 왕복 지연시간을 비교한다. 그림에서 without Virtualization은 두 실험 노드 모두 가상화를 전혀 사용하지 않은 경우의 성능으로 가상화에 의한 오버헤드 정도를 보여주기 위한 참조 성능이다. VirtualBox-Virtio는 IV장에서 논의하도록 한다. 그림에서 볼 수 있는 바와 같이 Type-1 반가상화를 지원하는 Xen PV 모드가 다른 가상화 환경에 비해서 가장 좋은 성능을 보여주고 있는 것을 알 수 있다. 반면에 전가상화의 경우는 Type-1(Xen HVM 모드)과 Type-2(VirtualBox)가 거의 비슷한 성능을 보여주고 있으며, 오히려 Type-2가 큰 메시지에 대해서 좋은 성능을 보여주고 있다. 이것은 전가상화의 경우 Type-1과 Type-2 모두 PCI 주소 공간과 네트워크 디바이스를 에뮬레이션하기 위한 오버헤드를 피할 수 없기 때문이다. 또한 Type-1의 경우 물리 디바이스 접근을 위한 도메인으로 Domain 0를 포함하고 있는데 이 오버헤드가 Type-2의 호스트 운영체제에 의한 오버헤드와 대등한 것으로 파악된다.

〈그림 5〉는 통신 대역폭 측정 결과를 보여준다. 통신 지연시간의 경우와 마찬가지로 Type-1 반가상화를 지원하는 Xen PV 모드가 다른 VMM에 비해서 가장 좋은 성능을 보여주고 있다. 특히 512B 이상의 메시지에 대해서는 가상화가 되지 않은 경우와 동일한 성능을 보여주고 있다. 하지만 물리 네트워크의 대역폭이 더욱 높은 경우(예, 10 Gigabit Ethernet)에는 성능 차이가 존재할 수 있음을 유념해야 한다. 전가상화의 경우는 VirtualBox가 Xen HVM 모드보다 좋은 성능을 보여주고 있다.

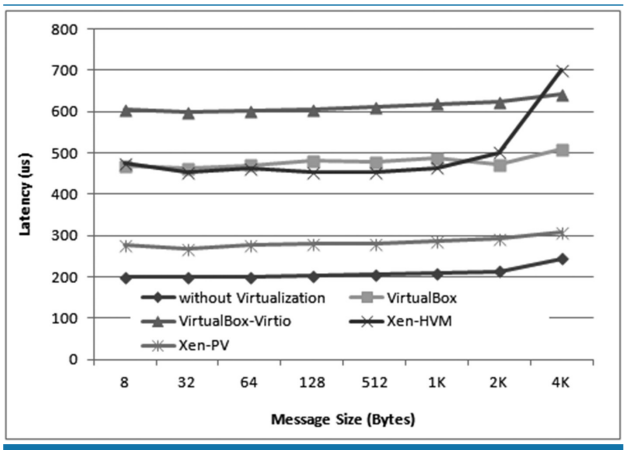

그림 1. 4D 아키텍처

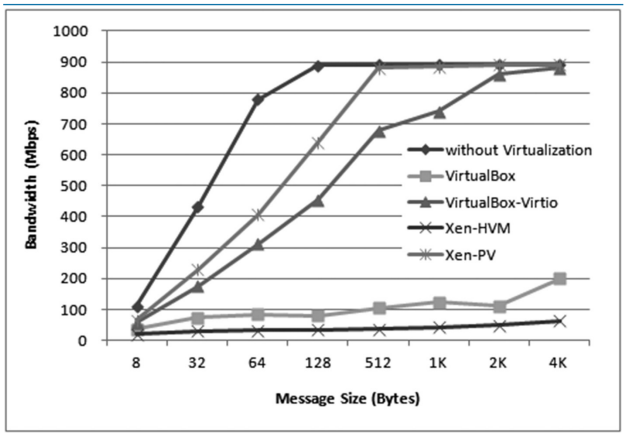

그림 5. 통신 대역폭 비교

이것은 Type-1과 Type-2의 근본적인 구조적인 차이에 의한 것이라기보다는 VMM들이 데이터 경로 및 처리를 얼마나 최적화했는가에 의한 차이로 해석해야할 것이다.

이상의 비교 내용을 요약하면 다음과 같다. 반가상화는 게스트 운영체제와 디바이스 드라이버를 수정해야 하지만 알려진 바대로 네트워크 I/O에 대해서도 전가상화보다 우수한 성능을 보여주는 것을 알 수 있었다. 그리고 일반적으로 Type-1이 Type-2보다 우수한 것으로 알려져 있으나 Type-1의 경우도 디바이스 드라이버를 위한 Domain 0를 필요로 하므로 Type-2가 갖는 호스트 운영체제에 의한 오버헤드를 크게 극복하지는 못하는 것으로 분석되었다.

Ⅳ. 네트워크 I/O 성능 향상을 위한 최근 동향

본 장에서는 최근에 네트워크 I/O 성능을 향상시키기 위해서 시도되고 있는 기술들의 동향들에 대해서 소개한다.

III장에서 분석한 바와 같이 전가상화 기법은 게스트 운영체제와 디바이스 드라이버의 수정을 요구하지 않는 반면 I/O 디바이스를 에뮬레이션 하기 위한 오버헤드가 발생한다. 이러한 오버헤드를 줄이기 위해서 디바이스 드라이버 수준에서만 반가상화 기법을 쓰는 방안이 제안되고 있다. 이 중에서 virtio는 게스트 디바이스 드라이버를 구현하기 위한 반가상화 표준 인터페이스를 정의하고 있다 [9]. 따라서 표준을 따르는 반가상화 디바이스 드라이버는 virtio를 지원하는 VMM 위에서는 수정 없이 수행 가능하다.

〈그림 4〉와 〈그림 5〉의 VirtualBox-Virtio는 VirtualBox 가상화 환경에서 virtio의 성능을 보여준다. 그림 5에서 볼 수 있는 바와 같이 메시지 크기가 4KB일 때 virtio가 기록한 대역폭은 반가상화 경우와 동일한 것을 알 수 있다. 하지만 통신 지연시간은 오히려 기존의 전가상화 기법에 비해서 조금 증가한 것을 그림 4에서 볼 수 있다. 이와 같은 현상은 게스트 도메인과 VMM 간의 전환 횟수를 줄이기 위해서 패킷 처리를 모아서 하고 있기 때문이다 [10].

전가상화 환경과 반가상화 환경 모두 네트워크 데이터 경로가 비효율적인 것을 III장에서 확인할 수 있었다. 이것은 게스트 도메인에서 직접 물리 네트워크 디바이스를 접근할 수 없으므로 호스트 운영체제 또는 Domain 0가 네트워크 I/O에 지속적으로 관여하기 때문이다. 이러한 문제를 해결하고 효율적인 네트워크 데이터 경로를 제공하기 위해서 게스트 도메인이 물리 네트워크 디바이스를 직접 접근할 수 있도록 하기 위한 Passthrough 구조가 연구되고 있다 [11][12].

Passthrough를 위해서는 네트워크 디바이스가 가상 디바이스를 구현하고 이를 각각의 VM에게 할당할 수 있는 능력을 갖고 있어야 한다. VMM은 가상 디바이스와 가상 머신을 연결하는 과정에만 관여하며, 네트워크 데이터 이동은 게스트 디바이스 드라이버가 물리 네트워크를 직접 접근함으로써 이루어진다. 물리 네트워크 디바이스 내에서 가상 디바이스들은 서로 독립된 메모리 영역을 사용하므로 가상 디바이스 간의 간섭을 피할 수 있다.

Single Root - I/O Virtualization (SR-IOV)은 Passthrough 기능을 제공하는 PCI-Express (PCIe) 네트워크 디바이스를 위한 표준이다 [13]. SR-IOV는 다수의 VM들에게 복수의 PCIe 카드로 표시될 수 있도록 하는 VF (Virtual Function)를 제공하여 여러 게스트 도메인이 PCIe 네트워크 디바이스를 충돌 없이 직접 접근할 수 있도록 한다. 즉, 물리 네트워크 디바이스의 공유를 VMM이 담당하지 않고 네트워크 디바이스가 다중화/역다중화를 구현하여 게스트 도메인에서 물리 디바이스로 직접 I/O 요청을 전달할 수 있게 한다.

Multi Root - I/O Virtualization (MR-IOV) [14]은 서로 다른 물리적 노드 간에 PCIe 디바이스 자원을 공유할 수 있게 해준다. MR-IOV는 SR-IVO와 마찬가지로 VF를 제공하지만, SR-IVO가 하나의 노드를 대상을 한다면 MR-IOV는 다수의 노드를 대상으로 한다. 노드와 노드 사이에는 PCIe 스위치가 존재하며 스위치를 통해 다른 노드의 PCIe 디바이스를 접근 할 수 있다.

네트워크 디바이스는 데이터 송수신을 위해서 시스템 메모리 영역을 접근하는데 이 때 물리 주소가 사용된다. 일반적으로 이 물리 주소는 디바이스 드라이버가 송수신 요청 시 네트워크 디바이스에게 제공해 준다. 가상화되지 않은 환경에서는 디바이스 드라이버가 커널 영역에서 수행되므로 네트워크 데이터를 위한 버퍼의 물리 주소를 쉽게 알 수 있다. 하지만 가상화 환경에서 게스트 도메인은 VMM이 마련해준 가상 물리 메모리 영역을 사용한다. 따라서 게스트 도메인의 드라이버에서 데이터 송수신을 위해서 작성된 버퍼 정보는 가상 메모리 주소를 포함하게 되고, 네트워크 디바이스가 이를 이용해서 DMA를 수행하기 어렵다. Passthrough 구조에서는 게스트 디바이스 드라이버가 직접 네트워크 디바이스에게 송수신 요청을 하기 때문에 이러한 가상 주소와 실제 물리 주소 간의 변환이 필요하다.

주소 변환을 효율적으로 지원하기 위한 기술로는 IOMMU (Input/Output Memory Management Unit)가 있다 [15]. 이것은 시스템 메모리와 I/O 디바이스 간의 MMU로서 I/O 디바이스를 위한 가상 메모리와 물리 메모리 간의 맵핑 및 관리를 하는 장치이다. 따라서 IOMMU는 Passthroug 구조에서 게스트 디바이스 드라이버가 가상 주소를 사용하는 것을 허용하며, 네트워크 디바이스가 시스템 메모리를 접근할 때 물리 주소로 변환해 줄 수 있다. 현재 여러 프로세서 제조사들이 IOMMU 기술을 자신들의 제품군에 포함시키고 있다. AMD의 AMD-Vi (AMD I/O Virtualization Technology)와 Intel의 VT-d (Virtualization Technology for Directed I/O)가 이들에 해당된다.

V. 결 론

본고에서는 가상화 플랫폼들이 네트워크 I/O를 지원하기 위해서 사용하는 기법들에 대해서 비교 설명하였다. 특히 Xen (HVM 모드, PV 모드), VirtualBox와 같이 많이 사용되는 오픈 소스 VMM을 대상으로 서로 다른 형태의 가상화 기법을 위한 네트워크 I/O 지원 구조와 그 성능을 비교하였다. 또한 가상화 환경에서 네트워크 I/O 성능을 최적화하기 위한 virtio, Passthrough (SR-IOV, MR-IOV), IOMMU 등과 같은 기술 동향을 소개했다.

현재 가상화 플랫폼을 스마트 임베디드 디바이스에 적용하려는 시도는 계속되고 있다. 이와 동시에 임베디드 시스템에서 가상화가 갖는 성능적 특성이 함께 분석되어야 할 것이다. 특히 통신 기능은 스마트 임베디드 디바이스에게 필수적인 요소이므로 가상화 플랫폼의 통신 성능 분석과 최적화는 상당히 중요한 부분이라고 할 수 있다. 따라서 스마트 임베디드 디바이스에 가상화 플랫폼을 성공적으로 적용하기 위해서는 응용 분야에 적합한 다양한 관점에서의 네트워크 성능 분석과 함께 최적화가 이루어져야 할 것이다.

Acknowledgement

본 연구는 지식경제부 및 한국산업기술평가관리원의 IT산업 원천기술개발사업의 일환으로 수행하였음. [10038768, 유전체 분석용 슈퍼컴퓨팅 시스템 개발]

참 고 문 헌

[1] K. Adams, and O. Agesen, "A comparison of software and hardware techniques for x86 virtualization," In Proc. of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-XII), Dec. 2006.

[2] A. Gavrilovska, S. Kumar, H. Raj, K. Schwan, V. Gupta, R. Nathuji, R. Niranjan, A. Ranadive, and P. Saraiya, "High-Performance Hypervisor Architectures: Virtualization in HPC Systems," In Proc. of Workshop on High Performance Virtualization (HPCVirt), 2007.

[3] S.-M. Chung and H.-W. Jin, "Isolating System Faults on Vehicular Network Gateways Using Virtualization," In Proceedings of The Sixth IEEE/IFIP International Symposium on Trusted Computing and Communications (TrustCom 2010), Dec. 2010.

[4] 한상현, 석종수, 진현욱, "차량용 인포테인먼트 시스템을 위한 혼합 파티션 지원", 한국정보과학회 2012 한국컴퓨터종합학술대회 (KCC 2012) 논문집, 2012년 6월.

[5] S. Yoo, C.-H. Hong, C. Yoo, Y. Liu, Y. Zhang, "MobiVMM: a VMM for Mobile Phones", The Workshop on Virtualization in Mobile Computing

(MobiVirt 2008), Sep. 2008.

[6] K. Barr, P. Bungale, S. Deasy,V. Gyuris, P. Hung, C. Newell, H. Tuch, and B. Zoppis, "The VMware Mobile Virtualization Platform: is that a hypervisor in your pocket?" ACM SIGOPS Operating Systems Review, Dec. 2010.

[7] F. Altschuler and V. Palatin, "Virtualization for Advanced Power Management of Consumer Electronic Devices," In Proc. of Consumer Communications and Networking Conference (CCNC 2009), Jan. 2009.

[8] L. Cherkasova and R. Gardner, "Measuring CPU Overhead for I/O Processing in the Xen Virtual Machine Monitor," In Proc. of USENIX Annual Technical Conference (USENIX ATC 2005), 2005.

[9] R. Russell, "virtio: towards a de-facto standard for virtual I/O devices," ACM SIGOPS Operating Systems Review, 2008.

[10] 이상헌, 진현욱, "가상화 환경에서 virtio를 위한 적응적 네트워크 데이터 전송 기법", 한국정보과학회 2011가을 학술발표논문집, 2011년 11월.

[11] H. Raj, and K. Schwan. "High performance and scalable I/O virtualization via self-virtualized devices." In Proc. of the 16th International Symposium on High Performance Distributed Computing, 2007.

[12] L. Xia, J. Lange, and P. Dinda, "Towards Virtual Passthrough I/O on Commodity Devices," In Proc. of First Workshop on I/O Virtualization (WIOV '08), Dec. 2008.

[13] Y.Dong, Z. Yu, and G. Rose, "SR-IOV Networking in Xen: Architecture, Design and Implementation," In Proc. of First Workshop on I/O Virtualization, Dec. 2008.

[14] B. Homolle, B. Schrader, and S. Brutt, "Multi Root I/O Virtualization (MRIOV)," In Proc. of the 1. Fachgesprach Virtualisierung, 2007.

[15] M. Ben-Yehuda, J. Mason, J. Xenidis, O. Krieger, L. Van Doorn, J. Nakajima, A. Mallick, and E. Wahlig, "Utilizing IOMMUs for Virtualization in Linux and Xen," In Proc. of the Linux Symposium, 2006.

약 력

진 현 욱

1997년 고려대학교 전산학 학사
1999년 고려대학교 전산학 석사
2003년 고려대학교 통신시스템공학 박사
2003년~2006년 미국 오하이오 주립대학교 연구원
2006년~2010년 건국대학교 컴퓨터공학부 조교수
2010년~현재 건국대학교 컴퓨터공학부 부교수
관심분야: 운영체제, 임베디드 컴퓨팅, 클라우드 컴퓨팅, 고속 네트워크

김 종 서

2011년 건국대학교 컴퓨터공학 학사
2011년~현재 건국대학교 컴퓨터공학 석사과정
관심분야: 운영체제, 플랫폼 가상화