Measuring Overhead Introduced by VMWare Workstation Hosted Virtual Machine Monitor Network Subsystem

Miquel Ferrer
Computer Architecture Department, Technical University of Catalonia
C/ Jordi Girona 1-3, Campus Nord UPC, Mòdul D6
Barcelona, Spain
e-mail: {mferrer}@ac.upc.es

***Abstract*— Use of virtual machines (VM), which concept was introduced by IBM, are increasingly due its benefits like better resource utilization and ease system manageability. Although, virtualization of I/O devices can introduce an overhead due to layer increment between VM and host. This penalty is specially wrong when the requested devices have high sustained thoughput and/or low latency. An excellent example of a device that requires both characteristics is a network interface card (NIC). In this work, VMWare Virtual Machine Monitor Network Subsystem is studied and measured to provide a measure of its introduced overhead.**

I. Introduction

Since 1960, when IBM first introduce virtual machines (VM), its use has been increasing continuously, due to its benefits that include isolation, resource sharing, etc. First developments was made over mainframes, and the term virtual machine was used to define a full protected and isolated copy of the underlying physical machine hardware. Thus, each virtual machine user is given the illusion of having a dedicated physical machine. Figure 1 shows the architecture of traditional organization of a virtual machine system. In this architecture there is a thin layer close to hardware, called virtual machine monitor (VMM), that takes a complete control of the machine hardware, and creates a virual machines, each one behaving like a complete physical machine and running its own operating system. To maximize the performance, VMM gets out of the way whenever possible and allows VMs to execute directly on the hardware, albeit in a non-privileged mode.

In the last decades use of the Intel-based PCs have been increased spectaculary due to its high performance low price ratio. In this way, Intel-based PCs can be used to develop taks assigned to mainframes in the past and can be used to develop user-level applications. Most of the benefits of mainframes virtual machines, and several new ones, can be applied to the PC platform. Although, the *traditional approach* of virtual machines can't be applied easily to PC's for the following reasons: Non-virtualizable hardware, PC hardware diversity and preexisting software.

To solve this problem new architectures are appeared. One of this is VMWare Architecture, which is shown in figure 2. Other works have studied VMWare Network Subsystem architeture and provided results of overhead introduced by this architecture. Although, these works have olny focused on CPU introduced overhead and how this overhead increment the response time of an action. In this work we focuses on data-overhead introduced by VMWare Network Subsystem architecture.

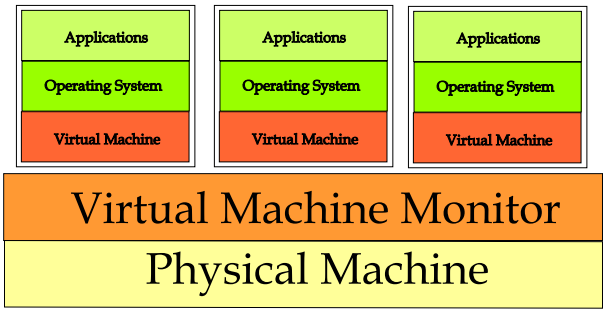

Fig. 1. VMWare's network subsystem.

The rest of this paper is organized as follows. Section 2 describes VMWare Workstation Network Subsystem Architecture, and specially in Network Interface Card NIC implementation. Section 3 presents our experiments and provides a measurements of performance of NIC virtualization of VMWare Workstation 4.5. Finally, section 4 summarizes the observed properties of this I/O device virtualization and presents some conclusions about it.

II. VMWare architecture

VMWare Workstation has a **hosted architecture** to virtualize I/O that allows it to co-exist with a pre-existing *host* operating system. Figure 2 shows this architecture. In this design there are three main portions: **VMApp**,**VMDriver** and **VMM**. VMApp is a simple application that allows users to install other operating systems int its. This application uses a driver (VMDriver) loaded into host OS to establish the privileged the virtual machine monitor (VMM). This virtual machine runs directly on the hardware. Each physical processor can run

either the host world or the VMM world. VMDriver allows control transfer between the two worlds. Although, this world switching is more heavy than normal switching. For its reason *world switching* introduces an overhead in the CPU. You can find more detailed information about this architecture in [1].

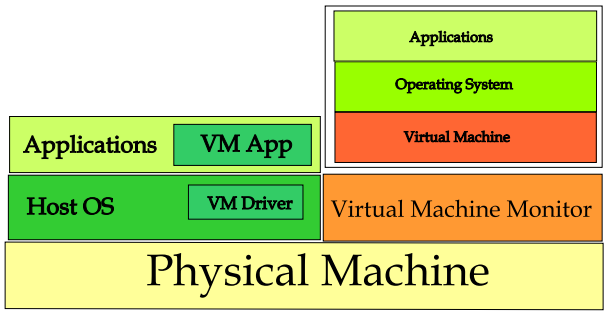

Fig. 2. VMWare's network subsystem.

A. Virtualization of I/O devices

To virtualize I/O devices, VMM intercepts all I/O operations issued by the guest operating system via special privileged IA-32 IN and OUT instructions. This virtualization can introduce overhead due to switching between host world ans VMM world. However, these overheads matter only for devices with either high sustained thoughput or low latency. An excellent example of these devices is a network interface card (NIC). Next section presents how VMWare architecture implements this device. 1

B. Virtualizing Network Card

In VMWare architecture, the virtual NIC is presented to guest as a full-fledged PCI Ethernet controller, complete with its own MAC adress. As shown in figure 3, NIC emulation can be done in two ways. It can be bridged to the same host network as a physical NIC, or it can be connected to a virtual network which is created on the host. In both cases these connections are provided by *VMNet* driver which is loaded in the host OS.

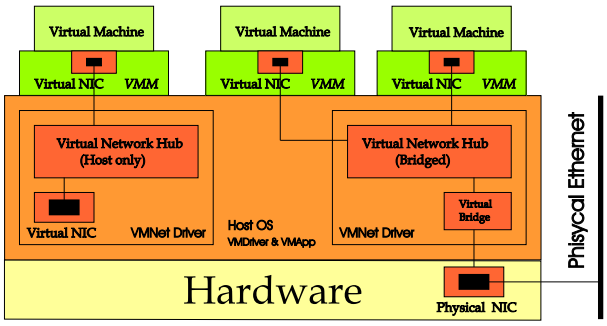

Fig. 3. VMWare's network subsystem.

In the case when the virual NIC is bridged to a physical NIC, it appears on the local Ethernet segment indistnguishably from any real machine because this implementation is a true Ehternet bridge in a srict sense. As a result, a virtual machine with a bridged virtual NIC can participate in a full manner in accessing and providing network services.

A virtual NIC that is connected to a virutal network can work in two ways. If host doesn't have Ethernet interface, this connection can be used to connect virtual machine to a private network composed by itself, host and other virtual machines. If the host have an Ethernet interface, can offer connection to an external network by providing NAT translation to virtual machines. Virtual NIC implemetation is performed combining VMM and VMApp. VMM exports a number of virtual I/O ports and a virtual IRQ that represents the virtual network adapter in the virtual machine.

III. Virtual Network Subsystem Performance

Virtualizing I/O devices can provide excellent flexibility and portability but can also introduce degradation in performance specially in high troughput low latency devices. This section analyzes VMWare Virutal Network Subsystem introduced overheads.

A. Experimental Setup

The measurements were done on two Intel-based PCs, physically connected via 10 MB/s Ethernet network with other computers. One of this PC, used to serve data to the other PC, was a dual Xeon Pentium at 2.4 GHz with hyperthreading with 1GB of memory. In this PC was installed a Red Hat 9.0 Linux with 2.4.31-smp kernel version as a host OS, VMWare Workstation 4.5 and Windows XP Professional and another Red Hat 9.0 with 2.4.31 kernel versions as a guests OS. This PC will be called PC1 in the rest of this work. The other PC, used to collect data packets and process its information, was a Pentium 4 at 2.4 GHz Laptop computer with 256 MB of memory running a Windows XP Home edition. It had installed NetAsyst 1.0 15-day trial version to capture IP datagrams send by PC1 and extract and process its data. This PC will be called PC2. Once this scenario was completed four types of transmissions were done. Two of this transmissions were done between guests operating systems installed on PC1 and PC2. Another path were between Host OS installed on PC1 and PC2. Finally, the fourth path were between two guests operating systems installed on PC1. Figure 4 shows these transmission paths. For convenience, in the next sections these transmission paths will be named in the following mannner: *SourceOS-DestinationOS*. For example, *VMLinux-Win*; *VMwin-Win*; etc.

B. Packet transmission

The first experiment was to execute a *ping* command from each OS located on PC1 to the PC2. First we transmit from *VMLinux* to *Win* four IP datagrams containing an ICMP request. The command executed to transmit only four datagrams on linux shell was *ping -c 4 192.168.1.116*. After, the same command was executed from *Linux* to *Win*. Finally,

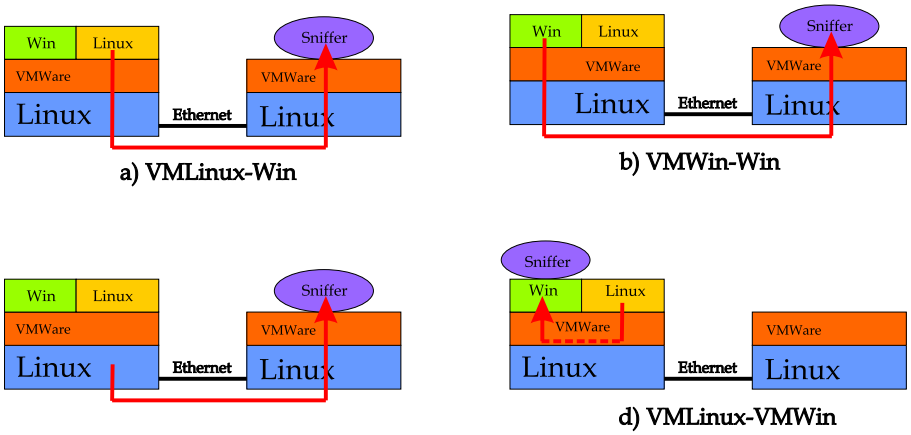

Fig. 4. VMWare's network subsystem.

TABLE I
PING COMMAND RESULTS

Source station	Dest. station	Tx Frames	Tx Data	Duration
VMLin	Win	4	392	3s 2ms
VMWin	Win	4	392	3s 17ms
Lin	Win	4	392	2s 9ms

TABLE II
PING COMMAND RESULTS

Source	Dest	Tx Frames	Tx Data	Duration
Win	VMLin	4	392	3s
Win	VMWin	4	392	3s 2ms
Win	Lin	4	392	3s 1 ms

transmission was done from *VMWin* to *Win* using *ping -l 56 192.168.1.118*. This command was used because the default amount of data transmited by windows is diferent to the quantity of data transmited by Linux. Table I shows the results obtained from this experiment. There are mainly two comments about this results. Second, as shown in table I, the time spent to transmit the same quantity of data from *VMLinux* and *Linux* is different. In the first case the time needed to transmit this 392 bytes was 3s 2ms and in the second one was 2s 9ms. This results are agree with the results presented in [1], and are due to the extra work introduced by VMWare Network subsystem. In the case of Windows, spent time is greater than spent time used by Linux and VMLin transmissions.

Table II shows the results obtained by transmit ping command from *Win* mchine to another three. These results show that spent time to receive four packets containing ICMP request are practically the same in all the destination machines. In this case the amount of time dues to overhead in virtual machine can be considered despreciable.

Next four tables show the results obtained on transmit a 40Mb of data from one computer to another using http protocol. In each case the destination was *Win* machine. The file was hosted in every source computer and taken it from destination by issuing a http request from internet explorer. Each table show the results obtained in each transmission at diferent protocol level. Thus, table III shows the reults at application level, table IV at transport level, table V at net level and table VI at MAC level.

The results show that the fastest transmission was when using *VMWin* to *Win* transmission. Comparing two linux machines, the results show that the *VMLinux* machine is faster than *Linux* machine in all three last cases. This result can be explained because VMWare have a disk buffering that can speed up the acces time to acces data.

IV. Conclusions and Future Work

The results obtained in experiments show that VMWare Hosted Network Interface Card implementation can introduce an overhead in time due to inclusion of an extra layer in the transmition path. The ration of time overhead is major when transmitted data is small and decrements when the amount of data to transmit becomes high. This cause can be eplained due to disk buffer contained in VMWare virtual machine. Although from data point of view, this interface doesn't introduce a data overhead. This results are agree of obtained in [1].

To obtain more accurate results different experiments can be done in the future. One of this can be to transmit data from one virtual machine to another to study a virtual network implementation of VMWare. Another one could be to trace the execution of VMWare to determine where time is spent.

References

[1] J. Sugerman, G. Venkitachalam and B. Lim , *Virtualizing I/O Devices on VMWare Workstation's Hosted Virtual Machine Monitor*, Proceedings of 2001 USENIX Annual Techincal Conference. Boston, June 2001.
[2] NIC Team, *ESX Server 2 white paper*, vmware.com.
[3] *VMWare datasheet*, vmware.com.
[4] *NetAsyst 1.0 datasheet*, Network Associates.

TABLE III

RESULTS FOR DATA TRANSMISSION AT APPLICATION LEVEL

Net Station 1	Net Station 2	Protocol	Frames	Bytes	Duration
Lin	Win	HTTP	27903	40738K	40s 469ms
VMWin	Win	HTTP	29849	40753K	39s 924ms
VMLin	Win	HTTP	27926	40745K	41s 559ms

TABLE IV

RESULTS FOR DATA TRANSMISSION AT TRANSPORT LEVEL

Net Station 1	Net Station 2	Protocol	Frames	Bytes	Duration
Lin	Win	TCP	43871	41616K	42s 933ms
VMWin	Win	TCP	49747	41748K	39s 926ms
VMLin	Win	TCP	44645	41638K	41s 563ms

TABLE V

RESULTS FOR DATA TRANSMISSION AT LEVEL

Net Station 1	Net Station 2	Tx Frames	Rx Frames	Tx Bytes	Rx Bytes	Protocol	Duration
Lin	Win	15966	27905	639K	41854K	IP	42s 933ms
VMWin	Win	19898	29849	796K	41947K	IP	39s 926ms
VMLin	Win	16718	27927	669K	41863K	IP	41s 563ms

TABLE VI

RESULTS FOR DATA TRANSMISSION AT MAC LEVEL

Net Station 1	Net Station 2	Tx Frames	Rx Frames	Tx Bytes	Rx Bytes	Protocol	Duration
Lin	Win	15966	27905	958K	42245K	Ethernet	42s 933ms
VMWin	Win	19898	29849	1194K	42365K	Ethernet	39s 926ms
VMLin	Win	16718	27927	1003K	42254K	Ethernet	41s 563ms