Power Consumption of Virtual Machines with Network Transactions: Measurement and Improvements

Ryan Shea*, Haiyang Wang*† and Jiangchuan Liu*
*Simon Fraser University, British Columbia, Canada
†University of Minnesota at Duluth, Minnesota, USA
Email: rws1@cs.sfu.ca, haiyang@d.umn.edu, jcliu@cs.sfu.ca

***Abstract*—There have been significant studies on *virtual machines* (VMs), including their power consumption in performing different types of tasks. The VM's power consumption with network transactions, however, has seldom been examined. This paper presents an empirical study on the power consumption of typical virtualization packages while performing network tasks. We find that both Hardware Virtualization and Paravirtualization add considerable energy overhead, affecting both sending and receiving, and a busy virtualized web-server may consume 40% more energy than its non-virtualized counterparts. Our detailed profiling on packet path reveals that a VM can take 5 times more cycles to deliver a packet than a bare-metal machine, and is also much less efficient on caching. Without fundamental changes to the hypervisor-based VM architecture, we show that the use of adaptive packet buffering potentially reduces the overhead. Its practicality and effectiveness in power saving are validated through driver-level implementation and experiments.**

I. Introduction

Fueled by promises of enhanced utilization, improved efficiency, and increased flexibility, system administrators have expanded their IT architectures to include a large number of virtualized systems [1]. Further, virtulization is one of the cornerstone technologies that makes utility computing platforms such as *cloud computing* a reality. For example the industry leader Amazon provides computing as a service through it's Xen virtualization based EC2 platform. There have been significant studies on the performance and optimization of virtual machines (VMs), including their power consumption in performing different types of tasks. Research has been done on reducing power consumption of data-centers through the placing of energy consuming jobs/VMs in strategic cooling locations, energy reduction through job and VM consolidation, and energy aware scheduling [2] [3] [4] [5] [6] [7] [8]. Other pioneering works have explored solutions to meter or cap the energy consumption of virtual machines running in a cloud environment [9] [10]. Work has also been done on power consumed during a VM migration [11]. Further, the energy consumption implications of VMs used in server consolidation has been discussed in [12].

The precise power consumption of these virtualized systems while processing typical network transactions however has seldom been examined. Physical network interface cards are not known as power hungry, except for wireless communications. Yet the virtualization of the network module can be considerably more complex than others in a machine, given that it is largely a standalone unit with its own control and storage units, and with both sending and receiving functions across multiple layers. It is known that the network module can be a severe performance bottleneck in VMs [13]. It remains unclear what the precise implications of virtualized networking overhead will be on total system power consumption.

978-1-4799-3360-0/14/$31.00 © 2014 IEEE

In this paper, we present an empirical study on the power consumption of typical computer virtualization packages, including KVM, Xen and OpenVZ, while the systems are performing network tasks. We find that both Hardware Virtualization and Paravirtualization systems add a considerable amount of energy overhead to networking tasks. Both TCP sending and receiving can be noticeably affected, and a busy virtualized web-server may consume up to 40% more energy than its non-virtualized counterparts. We have conducted detailed profiling to analyze the workflow for packet delivery in virtualized machines, which reveals that a VM can take nearly 5 times more cycles to deliver a packet than a bare-metal machine, and is also much less efficient on the systems hardware caches. The existence of hypervisors in VMs can dramatically increase interrupts and memory accesses, which in turn lead to significantly more power consumption for network transactions than a bare-metal machine does.

Our analysis further suggests that, without fundamental changes to the hypervisor-based VM architecture, the use of adaptive packet buffering potentially reduces the extra interrupts and memory copies. The practicality of our modifications has been validated through driver-level implementation as well as experiments in realworld systems. The results demonstrated that, with the adaptive buffering, we are able to improve the energy consumption of a busy web server by 16% without noticeable loss of its network performance.

The rest of this paper is organized as follows. In Section II, we offer an overview of typical virtualization techniques. The measurement configurations are introduced in Section III, followed by the results in Section IV, which also identifies where the power consumption comes from. Section V then presents effective batching solutions to mitigate power consumption, which is validated through experiments in Section VI. We then investigate more complex networked transactions in Section VII, and conclude the paper in Section VIII.

II. Overview of Virtualization

To thoroughly analyze and compare the power consumption of virtualization techniques, we need to select representative samples of virtualization packages, so as to cover the typical and state-of-the-art solutions. Broadly speaking, all current virtualization solutions can be classified into three main categories, which we discuss as follows.

Paravirtualization(PVM) Paravirtualization was one of the first adopted versions of virtualization and is still widely deployed today. PVM requires no special hardware to realize virtualization, instead relying on special kernels and drivers. The kernel sends privileged system calls and hardware access directly to a *hypervisor*, which in turn decides what to do with the request. The use of special kernels and drivers means a loss of flexibility in terms of choosing the operating systems. In particular, PVM must use an OS that can be modified to work with the hypervisor. Typical PVM solutions include Xen and User Mode Linux.

Hardware Virtual Machine (HVM) HVM is the lowest level of virtualization, requiring special hardware capabilities to trap privileged calls from guest domains. It allows a machine to be fully virtualized without the need for any special operating systems or drivers on the guest system. Most modern CPUs are built with HVM capabilities, often called *virtualization extensions*. They detect if a guest VM tries to make a privileged call to a system resource. The hardware intercepts this call and sends it to a hypervisor which decides how to handle the call. It has been noticed however that HVMs can also have the highest virtualization overhead and as such may not always be the best choice for a particular situation [13][14]. Yet paravirtualization I/O drivers can alleviate such overhead; one example of a paravirtualization driver package is the open source VirtIO [15]. Representative Virtualization solutions that are HVM include VMware Server, KVM, and Virtual-Box.

Container Virtualization Container Virtualization, also known as OS-level virtualization, creates multiple secure containers to run different applications in. It is based on the intuition that a server administrator may wish to isolate different applications for security or performance reasons while maintaining the same OS across each container. Container virtualization allows a user to share a single kernel between multiple containers and have them securely use computer resources with minimal interference from others containers. It has been shown to have the lowest overhead among all the existing virtualization techniques [13]. However, this superiority comes at the price of much less flexibility as compared to other solutions. In short, the user cannot mix different OSes. Typical container virtualization implementations include OpenVZ, Linux-VServer and Solaris Zones.

It is important to note that Hardware Virtualization and Paravirtualization both use a *hypervisor* to interact with the underlying hardware, whereas Container Virtualization does not. This distinction is crucial because the hypervisor, acting as a gatekeeper, generally improves performance isolation between guests on a host. However, it has been noted that the hypervisor can introduce measurable overhead [16], which, as we will show in our experiments, can also negatively affect the power consumption.

In our experiments, we chose to evaluate the power consumption of Xen, KVM and OpenVZ. We believe this choice is representative for the following two reasons. First, they are all open-source with publicly available documents and with cross-platform implementations. We can run their packages on the same platform without changing OS or computer hardware. This makes a fair comparison possible and the results reproducible; Second, all of them have been widely used in real-world production environments for server consolidation and Cloud Computing. As mentioned previously, Amazon EC2 is largely based on Xen; KVM has been used by Ubuntu Enterprise Cloud and Eucalyptus Cloud Service [17]; OpenVZ is a popular choice in offering Virtual Private Server(VPS) containers to the public.

III. Power Consumption Measurement: Platform and Virtualization Setup

In this section, we present the measurement configuration for the power consumption of virtual machines, including the hardware setup, the virtualization setup, and the measurement devices used.

A. Measurement Platform

For our test system, we use a modern midrange server with a Intel core i5 2400 3.1 GHz quad core CPU, 8 GB 1333 MHz DDR3 ram, a 500 GB 7200 RPM hard drive and a 1000 Mb/s Broadcom *Network Interface Card* (NIC) attached to the PCI-E bus. The choice of the Intel i5 server is motivated by the factor that the Intel's x86 architecture has long become dominating in the CPU market, which is also the basis for virtual machine implementation in such major cloud service providers as Amazon. The core i5 CPU is known to have low power consumption with well-designed state-of-the-art power management. In particular, when Intel introduced the Sandy Bridge line of processors, they also introduced the *Running Average Power Limit hardware counters* (RAPL) [18]. These highly accurate and versatile counters allow a user to configure the system to record its CPU power consumption. In all our experiments, we measure the internal power consumption using the RAPL counters while the CPU is running our network-based tasks.

Unfortunately, the RAPL counters do not measure the overall system power consumption. For example, in order to measure the power consumption of the modules other than the CPU, for example, cooling fans, hard drives, NICs, we have to use other tools. To determine the overall system's power consumption, referred to as *wall power*, we have wired a digital multi-meter (Mastech MAS-345) into the AC input power line of our system. We read the data from our meter over a PC-Link installed in the meter and collect samples every second throughout our experiments.

Finally, since our focus is on network transactions, we also configure a second system to work as a client emulator, which is a 2.8 GHz Intel Core2 Quad system, with 4 GB DDR3 ram, and a 1000 Mb/s network connection. We connect our test system and client emulator to each other through a 1000 Mb/s Linksys SD2005 SOHO switch.

B. Virtualization Setup

As explained earlier, we have chosen Xen, OpenVZ, and KVM in our experiments for their open-source nature and their extensive deployment in the real-world. We now briefly describe the system setup of these virtualization solutions.

1) Xen System Setup: We installed the Xen 4.1 Paravirtualization Hypervisor on our test system. To configure networking, we created a bridged adapter and attached our primary interface and Xen's virtual interfaces to it. The Xen virtual machine received an IP address from the DHCP server running on our gateway. Similar network configuration has also been used for the OpenVZ and KVM systems. For disk interface, we used the Xen's flat file feature. We set the number of virtual CPUs (VCPU) to 4 and the amount of RAM to 6144 MB. The virtual machine and physical host ran the 3.2.41-2-Xen kernel.

2) OpenVZ System Setup: We installed the OpenVZ container-based virtualization package from the official OpenVZ repository, following the latest guidelines. We configured our container using the Debian Squeeze template. The container was given access to 6144 MB of main memory and full access to the 4 CPU processing cores. The virtual machine host ran the ovzkernel-2.6.18 kernel, which is the latest patched kernel supported by Debian.

3) KVM System Setup: We compiled KVM version 1.2.0 from the official source repository. Once again the virtual machine was given full access to all the 4 processor cores as well as the 6144 MB of memory. The disk interface was configured as a flat file on the physical host's file system. The virtual machine and physical host ran the 3.2.41-2-amd64 kernel. To enable the best network performance, we configured KVM to use the VirtIO network drivers [15].

4) Non-Virtualized Bare-metal System Setup: Finally, as the baseline for comparison, we also had a bare-metal setup with no virtualization, i.e., the system has direct access to the hardware. The same drivers, packages and kernel were used as in the previous setup. This configuration enabled us to calculate the minimal amount of energy required to run our benchmarks. Debian kernel 3.2.41-2-amd64 was used in this test.

IV. Measurement Results and Analysis

To determine the power consumption profiles of the virtualized systems, we have performed a number of simple and complex benchmarks, and measured the corresponding CPU power consumption using the internal RAPL counter and the wall power of the systems using the AC power meter. We start from the following two benchmarks for network-oriented transactions:

Iperf Sending and Receiving: Before moving on to more complex experiments, we first used Iperf [19] to determine the energy consumption of our systems while sending and receiving high bandwidth TCP streams. To this end, we configured our test system to communicate with our client emulator, sending and receiving at their maximum rates for a duration of 60 seconds.

Apache2 Many Client HTTP Download: In this experiment, we emulated the workload experienced by a busy web server that serves a number of clients. We configured our client emulator to open 100 HTTP connections, each downloading a 20 MB file, and then recorded the energy consumption experienced by each system.

To obtain the base-line power consumption of our systems, we also measured the power consumption when the system is in an 'idle' state. To this end, we killed all non-essential processes on the system. We measured each system for 60 seconds, taking samples of the power consumption (in number of Watts consumed) every second.

To mitigate randomness, we run each experiment three time and calculate the average and standard deviation. We graph the results and display the standard deviation as the error bars on the graph. For experiments that have a deterministic running time we give the results in Watts (joules per second); for those with a non-deterministic running time, we give the results in total joules required to complete the benchmark. In each of our following experiments a single VM is running on the physical machine. Although the use of a single Virtual Machine is uncommon in practice, this configuration allows us to precisely measure the energy consumption of the virtualization system while running our test application. If multiple VMs were running on our host at the same time the energy consumption would increase as the VMs contended for the hosts resources.

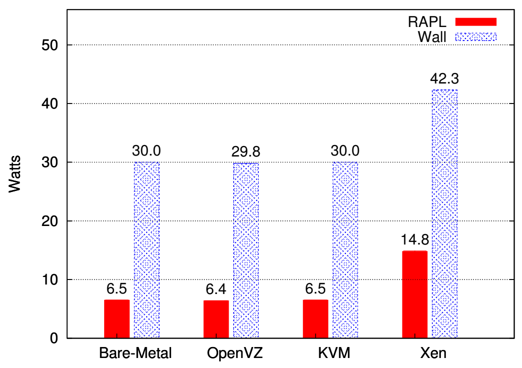

Fig. 1. Power Consumption - Idle

A. System Idle Power Consumption

Figure 1 shows the baseline power consumption of our systems when idle. The Bare Metal, OpenVZ, and KVM systems all show similar results to one another for both the wall power (the external or whole system power consumption), and RAPL (the internal or CPU power consumption). These three systems all have an average power consumption of about 30 watts at the wall, and about 6.5 watts at the CPU. The Xen system has a greater external and internal power consumption, with an average of 42.3 watts for the whole system and 14.8 for the CPU. For all of these systems, the error bars, which represent the standard deviation between tests, are very small. It is not surprising that KVM, OpenVZ, and the bare metal system have similar power consumption while idle, since they all take advantage of the standard Linux power saving system. However, in Xen, even the driver domain Domain-0 is in fact a virtual machine, which does not allow the Linux kernel to properly manage the power of the system. Our initial investigation leads us to conjecture that the Xen 4.1 hypervisor does not properly utilize the Core i5 processor's advanced *c-states* (sleep states). In the standard Linux kernel, when a processing core is idle, the system puts the core to sleep. We have forced Xen to use the c-states, yet it does not seem to enter the deep sleep states such as the c7 state, which leads to a much higher idle energy consumption. Using the Xen power management command "xenpm", we have determined our idle Xen system never enters a sleep state deeper than c1. This was observed despite the fact that we had passed the required boot time parameters to the Xen kernel and enabled the deeper sleep states in Xen power management module.

B. Power Consumption with Virtualization

Figure 2 and Figure 3 show the power consumption of our systems while running the Iperf TCP sending and receiving

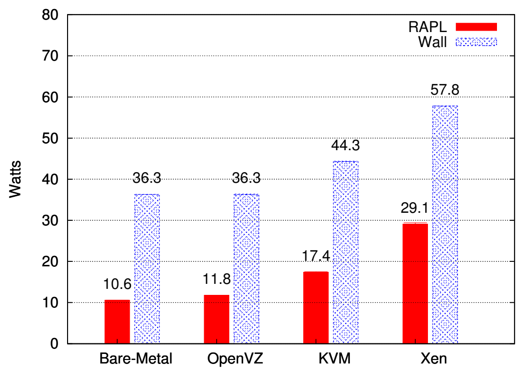

Fig. 2. Iperf - Sending 1000 Mb/s

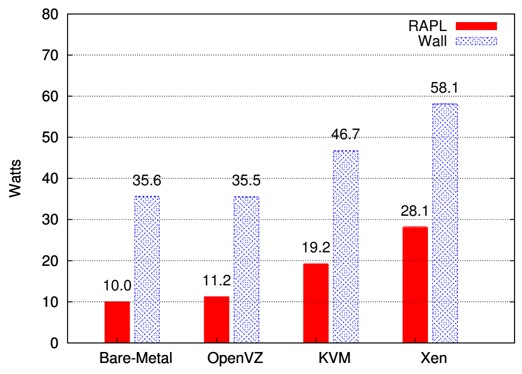

Fig. 3. Iperf - Receiving 1000 Mb/s

tests. In this experiment, all systems maintained an average TCP transfer rate of approximately 940 Mb/s while both sending and receiving. For the Iperf sending test, the Bare metal system has the lowest power consumption with an average of 36.3 watts consumed for the total system and an average of 10.6 watts for the CPU. OpenVZ has a similar power consumption as the Bare metal system for the total consumption, but a slightly higher internal power consumption of 11.8 watts. KVM has a significantly higher consumption with an average of 44.3 watts external and 17.4 watts internal. Xen has the worst performance with the highest power consumption: 57.7 watts for the whole system and 29.1 watts for the CPU. For the Iperf receiving test, with both Bare metal and OpenVZ, the results are similar but slightly better than for sending. Bare metal has slightly less power consumption with 35.6 watts external and 10.0 watts for internal consumption, as does OpenVZ with 35.5 watts external and 11.2 watts internal. KVM and Xen virtualization each performed slightly worse than on the sending test, with KVM consuming 46.7 watts external and 19.2 watts internal, and Xen consuming 58.1 watts external and 28.1 watts internal. Again, the standard deviations between the tests on the same system are very low. It is not surprising that both KVM and Xen consume considerably more power in these tests since network interface virtualization is a computationally expensive operation [20].

The results of Apache2 Many Client HTTP Download are shown in Figure 4 for each system. Our Bare metal control consumed an average of 37.9 watts for the whole system on this test, and an average of 12.1 watts for the CPU. Of the virtualized systems, OpenVZ again performed the best, with an average of 39.1 watts total consumption and 14.3 watts CPU

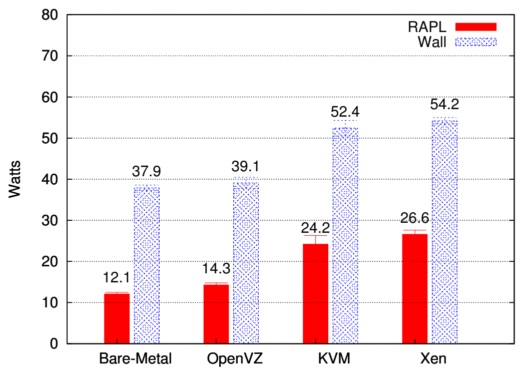

Fig. 4. Apache2 Download - 100 Clients Downloading

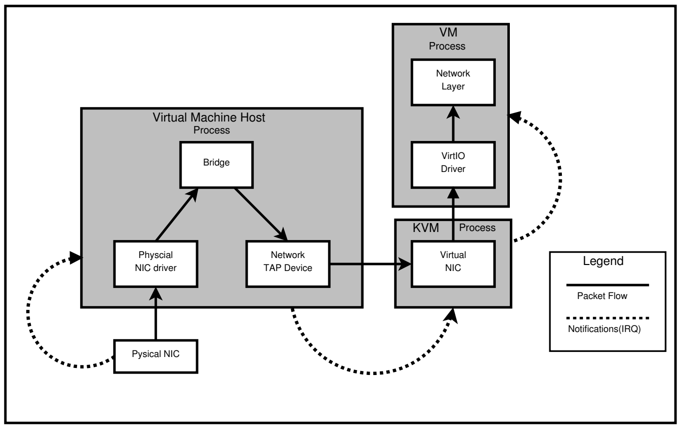

Fig. 5. KVM Network Architecture and Modifications

consumption. In this test, KVM and Xen performed similarly, with KVM consuming an average of 52.4 watts external and 24.2 watts internal, and Xen consuming an average of 54.2 watts external and 26.6 watts internal. The reason Xen and KVM have much closer energy consumption in this test is because both Xen and KVM appear to be load-balancing the apache HTTP threads over all available CPUs. Since all 4 cores now contain active threads the benefits of KVM's ability to reach deeper sleep states is much less pronounced. The standard deviations in this test are slightly higher, though remain small. These results are not unexpected, since transmitting data over a TCP based HTTP connection contains many small control messages and packets. Processing this large number of packets causes higher overhead in virtual machines since virtual packet processing is a much more expensive operation.

C. *Where Is Power Consumed ?*

To understand why there is increased energy consumption and overhead in virtual machines while processing the network tasks described above, we now take a close look at a state of the art virtualized network interface architecture. We use KVM as a representative because of its wide deployment, and because the VirtIO driver it employs is designed to work with a variety of virtualization systems, making the observations widely applicable.

We start our discussion with an overview of KVM's virtual packet processing. Figure 5, gives the typical path of a packet entering a virtualized system. The packet is first handled by the physical NIC, which copies the packet in the memory space of

	Bare-Metal	KVM VirtIO
Cycles	11.5M/Sec	51.8M/Sec
LLC References	0.48M/Sec	2.3M/Sec
IRQs	600/Sec	2600/Sec

TABLE I. IPERF RECEIVING 10 MB/S TCP TRAFFIC

	Bare-Metal	KVM VirtIO
Cycles	4.1M/Sec	33.4M/Sec
LLC References	0.13M/Sec	1.3M/Sec
IRQs	180/Sec	1700/Sec

TABLE II. IPERF SENDING 10 MB/S TCP TRAFFIC

the host physical machine and alerts the physical machine of the incoming data through the use of a hardware interrupt. The kernel on the physical machine is then scheduled to run and inspects the packet. The packet is then pushed through some form of software switch; in our experiments we used a Linux bridge. The switch then sorts the packet and sends it to the virtual NICs back-end, which in the case of KVM is a network tap device. The kernel then notifies the Virtual Machine's hypervisor process of the incoming packet(s) through a software interrupt, and the KVM hypervisor is scheduled to run. The KVM hypervisor process then copies the packet from the host's memory space into the virtual machine's memory space, and sends an interrupt to the virtual machine. Finally, the virtual machine's kernel collects the packet from the virtual NIC and passes the packet to the networking layer. When sending, the virtual machine simply pushes the packet along in the reverse direction through these steps. All of these additional steps cause a virtual machine to consume much more resources than a bare-metal machine when processing network traffic, thus potentially consuming much more energy.

The best way to show the increase in resource consumption created by virtualization is a small experiment. To this end, we designed an experiment to measure the virtualization overhead, and therefore the excessive power consumption, of sending and receiving on both virtual and hardware network interfaces. We once again used the `Iperf` network benchmark to create the TCP traffic to and from a remote host in the same subnet. We used the Linux hardware performance analysis tool `Perf` to collect system level statistics such as processor cycles consumed, last level cache references (LLC), and interrupt requests. We collected statistics for all cores in our physical system, thus our data shows resource consumption both inside and outside the VM. For each experiment, we instructed `Perf` to collect five samples each with a duration of 10 seconds and then averaged them. We tested two systems: our bare metal Linux host, and KVM with VirtIO drivers.

The results for the receiving experiment are given in Table I and the results for the sending experiment are given in Table II. Both CPU cycles and LLC references are given in millions per second. The interrupt requests are given in number per second. We can see that, in the receiving experiment, KVM with VirtIO takes nearly 5 times more cycles to deliver the packets to the VM as the bare-metal host. KVM with VirtIO is also much less efficient on cache than the bare-metal system. This is due to the fact that the VM's hypervisor must copy each packet from the memory space of the host to the VMs space. These copies use up valuable processor cycles and can also evict data from the processor cache. Next we look at

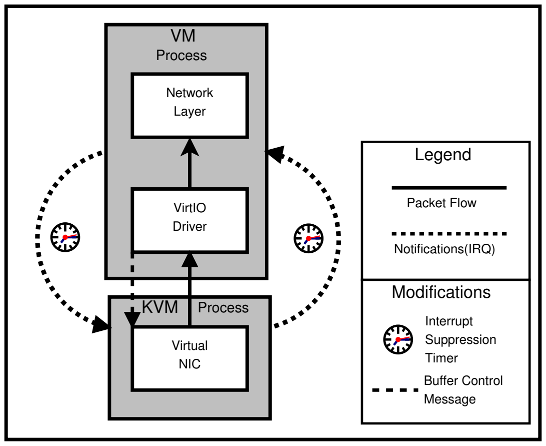

Fig. 6. Modified VirtIO Drivers

IRQs that are used by the physical device to notify the kernel of an incoming packet, and by the hypervisor to indicate to a running VM that it has received packets. Since KVM also uses interrupts to indicate to a running VM that it has received packets from the network, it comes as no surprise that they would be considerably higher than the bare-metal baseline. As such, if the number of interrupts can be reduced, we can amortize the cost of network processing by batch processing packets, thus reducing the amount of memory access and CPU cycles consumed by the VMs, which would in turn improve the power consumption for many network applications.

V. REDUCING POWER CONSUMPTION IN KVM

The results of our profiling experiment show that virtualization systems introduce considerable overhead while processing packets. From our previous description of how packets are processed in virtual systems, we propose that buffering packets will help reduce the overhead caused by the virtualization system, and therefore reduce the energy consumption.

It has been established in other contexts, such as mobile devices, that buffering sending and receiving tasks has an energy conservation benefit [21]. Much of this energy saving is due to the efficiency of batch processing of network packets and powering down the radio between network bursts. However, our novel approach in applying packet buffering to virtual packet processing achieves it's energy savings by greatly increasing the efficiency with which the hypervisor handles network traffic. Further, in [20] through hardware profiling techniques, we found that virtualized systems use more memory, cache and CPU cycles to process packets than non virtualized systems do. We implement our modifications in the VirtIO drivers of the KVM system. The choice of KVM and the VirtIO drivers is motivated by the fact that VirtIO is designed to be a driver system that is compatible with many virtualization systems. Thus, our modifications to the VirtIO drivers are likely applicable to many other virtualization platforms.

A. *Buffering Timer for VirtIO*

We first modified the VirtIO driver to buffer incoming packets, delivering them in bulk to the virtual machine. To accomplish this, we carefully inspected the VirtIO's virtual NIC device code and changed how the VM is notified that it has incoming packets. Instead of immediately sending an interrupt on a packets arrival, the KVM hypervisor can now set a timer. When the timer expires the VM is notified of the original packets as well as all other that have arrived since the timer was started.

Although we can configure the VirtIO virtual NIC device to ignore the interrupts created by the VM's transmitted packets, it is not enough to reduce the sending overhead of the VM. This is because the VirtIO drivers inside the VM perform a queue "kick" operation for every buffer of sent packets. The kick operation not only raises an interrupt to the hypervisor, but also performs an expensive locking operation to copy over the packets into the hypervisor's sending buffer. Thus, modifying the VirtIO's virtual NIC device is not enough to buffer the transmitted packets from the VM, and we must also optimize the driver residing inside the VM.

With this observation, we modified the VirtIO drivers to set a timer instead of immediately performing the kick operation. When the timer expires, a single efficient copy of all the packets from the VM's sending queue is made to the hypervisor. Also, a single interrupt is sent to the hypervisor instructing it to process these packets. Figure 6 shows the location of our modifications in the VirtIO NIC structure.

B. Synchronizing Receiving and Transmitting Delay

After modifying the VirtIO virtual NIC subsystem to allow buffering of transmitted and received packets, we next need to develop a system to synchronize the Receiving (RX) and Transmitting (TX) delays. Synchronization of the delays is needed for a number of reasons. First, we must be able to advertise to the KVM hypervisor the appropriate time to buffer the received packets. Second, the hypervisor must be instructed to only buffer the packets for systems running applications that can tolerate the buffering delay. Finally, to ensure a stable round trip time (RTT), the hypervisor and VM must balance their buffer times. To this end, we developed and implemented a new virtual "hardware" control message, which can be sent from the VM to the virtual NIC device. The control message specifies if the VM expects the hypervisor to buffer incoming packets as well as how long to buffer the packets for. Our modification allows for each VM on a physical host to enable/disable and control its own buffering independently. Figure 6 shows the path of hardware control message as well as a high level overview of our modified driver system. Current version of our source code is available at www.sfu.ca/~rws1/energy-kvm/.

To make our modifications adaptive to applications with different sending and receiving packet rates, we choose our buffering amounts based on a packet ratio estimate. We use $n_s(t_i)$ and $n_r(t_i)$ to refer the total number of sent/received packets between time slots t_i and t_{i-1} when the ith timer expires. The ratio of sent packets $P_s(t_i)$ is therefore

$$p_s(t_i) = \begin{cases} 0 & n_t{=}n_r{=}0 \\ \frac{n_s(t_i)}{n_s(t_i)+n_r(t_i)} & \text{else} \end{cases} \qquad (1)$$

Note that we can also let $p_s = n_s/(n_s + n_r + 1)$ in the real-world implementation. Based on this definition, we can get the *estimated sending packet rate* p_s^* using an exponentially weighted moving average as:

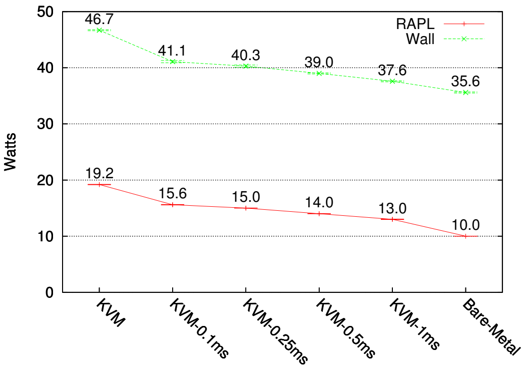

Fig. 7. Iperf Receiving 1000 Mb/s Improvement

$$p_s^*(t_i) = \alpha p_s(t_{i-1}) + (1-\alpha)p_s(t_i) \qquad (2)$$

where the real number $\alpha \in [0, 1]$ is the weighting factor. Smaller α makes the estimated packet rate more sensitive to the changes of packet rate. Larger α on the other hand, makes E immune to the short-term rate change. The size of sending buffer b_s and the size of receiving buffer b_r can therefore be obtained as

$$b_s = p_s^*(t_i)\, T_{buf} \qquad (3)$$

$$b_r = p_r^*(t_i)\, T_{buf} \qquad (4)$$

where T_{buff} is the maximum tolerable RTT specified by the user. $p_r^*(t_i)$ is the *estimated receiving packet rate*.

VI. Power Savings with Driver Modification

Using the VirtIO modifications and packet rate calculation formula described previously, we once again measure the energy consumption of network tasks on our test platform. We installed our modified drivers in our VM and configured the drivers to calculate the packet rate estimate and update the receiving and sending buffer amounts every 5 seconds. The drivers calculate the estimated packet rate using an α of 0.75, implying we slightly favor the historical samples. For all experiments, we once again run them three times and give the average as well as express the standard deviation as error bars on our graphs. We test different buffer times ranging from 0.1 ms to 1 ms, which, as compared to the typical wide area network RTT, is generally negligible.

Figure 7 shows the results for receiving of running the Iperf TCP benchmark using our modified KVM driver. In all tests the modified drivers achieve a maximum throughput of approximately 940 Mb/s. Even with a 0.1ms buffer there is an approximately 12% drop in energy consumption from the external Wall measurement, from 46.7 watts to 41.1 watts. With steadily bigger buffer sizes we continue to see steady improvement, up to an improvement of 19.5% over the unmodified KVM. For the internal RAPL measurement, we also see a steady improvement with increased buffer size, from an 18.8%

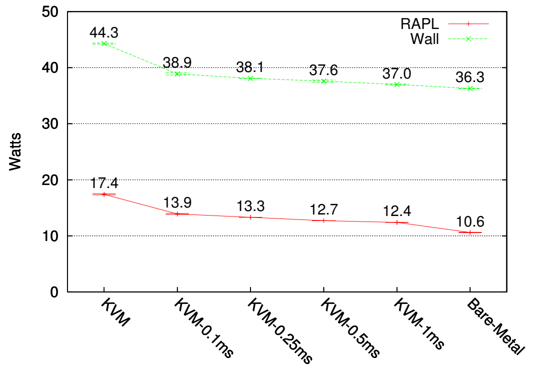

Fig. 8. Iperf Send 1000 Mb/s Improvement

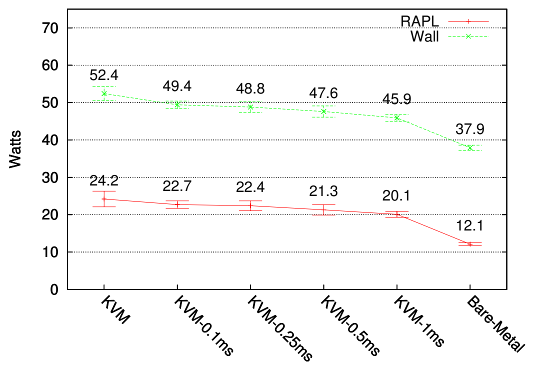

Fig. 9. Apache2 Download Improvement

improvement in energy consumption with a 0.1ms buffer to a 32.3% improvement over the unmodified driver with a 1ms buffer. Figure 8 shows the Iperf TCP benchmark results for sending. In the wall measurements, with the smallest buffer size, there is already a major improvement of 12.2%, from 44.3 watts with the basic KVM driver to 38.9 watts with our modified driver. There is a gradual but steady improvement with increased buffer size, up to 16.5% change in energy consumption with a 1ms buffer. The RAPL measurements show a very similar trend. With a 0.1ms buffer there is a 20.1% change in energy consumption, and with a 1ms buffer we see a 28.7% improvement with our modified KVM driver.

In Figure 9, we see the energy consumption of running the Apache benchmark with the basic KVM driver compared to with our modified driver using different buffer sizes. For the wall measurements, there was a fairly steady increase with increasing buffer size. With the smallest buffer size we only saw a 5.7% improvement over the basic KVM driver, however this increased to a 12.4% change in energy consumption with a 1ms buffer. The results for the internal measurements are similar. With a 0.1ms buffer we see a 6.2% change, from 24.2 watts with the basic driver to 22.7 watts with the modified driver. With a 1ms buffer we see a 17.0% improvement over the unmodified driver. It is important to note that our modifications did not reduce the data transfer rate of the HTTP downloads, thus the modified KVM system still served the files in the same amount of time as the baseline systems.

VII. Impact to Complex Network Applications

So far we have focused on the basic network transactions. We now establish the power consumption profiles of advanced and more realistic network applications, as well as the impact of buffering. Although the simple benchmarks presented previously illuminate some important characteristics of virtualized systems in terms of energy consumption, it is critical to examine more complex applications which are more commonly found in real world data-centers.

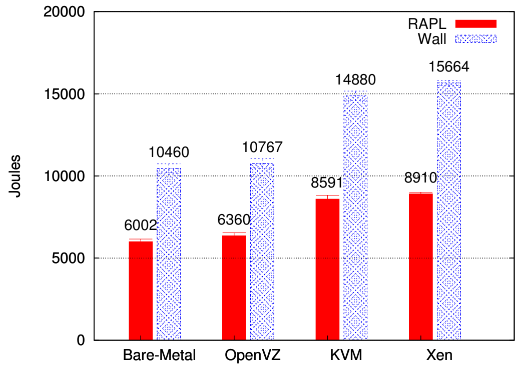

Fig. 10. RUBBoS - 100 Clients, 500,000 requests

A. Benchmark Setup

RUBBoS Bulletin Board Benchmark: To further understand the overall system power consumption, we have devised a comprehensive benchmark based on a simple 2-tier web server and database. We used the Debian repositories to install the `Apache` 2.2 Web Server and the `MySQL` Server 5.5. To create a web application representative of a real-world service, we installed the `RuBBoS` bulletin board benchmark [22]. We chose the PHP version of the RuBBoS and installed the necessary Apache extensions for PHP. We then installed the RuBBoS data into our MySQL database.

Although RuBBoS comes with its own client simulator, we used the one from the Apache benchmark instead, which has been more commonly used for web server stress testing [23]. Also, we only require the maximum request rate, which is more straightforward to extract with the Apache Benchmark. We ran the Apache Benchmark against the RuBBoS website in each of the test setups. We emulated 100 clients requesting the latest forum topics page. By using this page, the web server must perform a single SQL query and render the PHP page for each user request. We then used the Apache benchmark to calculate how long it takes to service the 500,000 requests.

Tbench Network File System Benchmark: Our final experiment employs Tbench [24], which emulates the network portion of the standard Netbench performance test. Tbench sends TCP data based on a workload profile, which simulates a network file system. We specified our test system as the Tbench server and then configured our client emulator to simulate 100 processes reading and writing the remote file system.

B. Power Consumption

Our results for the RUBBoS benchmark, the comprehensive web server test, are shown in Figure 10. The Bare metal system consumed an average of 10460 joules total and 6002 joules for the CPU. OpenVZ was comparable, with an average consumption of 10767 joules for the whole system and 6360 joules for the CPU. KVM and Xen performed worse, with KVM consuming an average of 14880 joules total and an

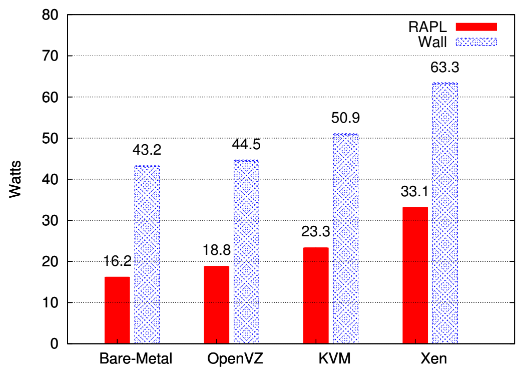

Fig. 11. Tbench, 100 network processes, read/writing remote file-system

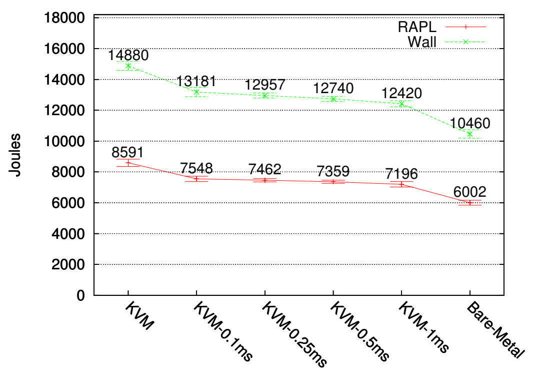

Fig. 12. RUBBoS Improvement

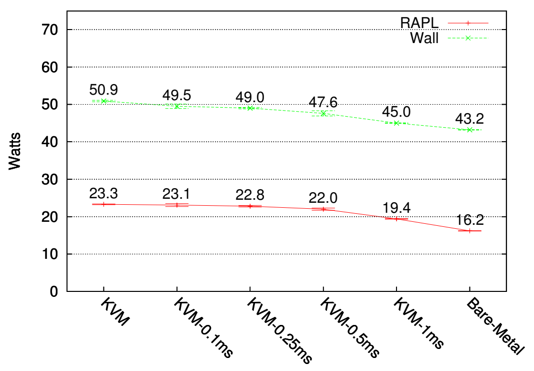

Fig. 13. Tbench, 100 network processes Improvement

average of 8591 joules for the CPU, and Xen consuming an average of 15664 joules total and an average of 8910 joules for the CPU. Unlike the previous benchmarks we have performed, the RUBBoS benchmark is not strictly a network-bound application. This multi-tier benchmark has to render a PHP page as well as perform database look ups to service each incoming request, both of which are hard on memory and CPU resources. This leads to even higher energy consumption because not only are the network accesses causing virtualization overhead, but the access to memory by the database is also contributing to the virtualization overhead.

Figure 11 shows the power used by each of our systems running the Tbench benchmark test. The Bare metal system uses 43.5 watts for the total system measured at the wall, and 16.2 watts for the internal CPU. OpenVZ performs nearly as well, consuming 44.5 watts at the wall and 18.8 for the RAPL counter. KVM also performs fairly well on this test, using 50.9 watts externally and 23.3 watts for the CPU. Finally, Xen uses 63.3 watts for the total system and 33.1 watts for the CPU. In this benchmark the Tbench server must emulate the responses to the read and write file system requests generated by the remote clients, while at the same maintaining high network throughput. This is a more complex use of the network communication infrastructure as performance is dependent on the Tbench server processing the clients' requests quickly.

C. Impact of Buffering

Using our modified drivers VirtIO drivers described previously, we test the candidate buffer time of 0.1 ms, 0.25 ms, 0.5 ms and 1 ms. For all experiments we once again run them three times and give the average as well as express the standard deviation.

Figure 12 shows the improved energy consumption while running the rubbos benchmark, comparing the unmodified KVM driver to the modified driver at our specified buffer sizes. The external measurements taken from the wall show that even with the smallest buffer size, there is a significant improvement in energy consumption. With a 0.1ms buffer, there is a 11.4% drop in energy consumption, from 14880 joules with the unmodified driver to 13181 joules with the introduction of the buffer. There is then a steady improvement with increasing buffer size, up to a 16.5% change from the unmodified KVM with a 1ms buffer. The internal RAPL measurements are similar, with the major improvement occurring with the introduction of even the 0.1ms buffer size, with a 12.1% drop in joules consumed. There continues to be an improvement in energy consumption with increasing buffer size, with our largest buffer size, 1ms, resulting in a 16.2% improvement, from 8591 joules using the standard KVM driver to 7196 joules with our modified driver. The modified drivers were not only able to reduce the energy consumption of our virtualized system but actually increased the performance as well. With a buffer of 1.0 ms our modified KVM virtual machine was able to service nearly 10% more request per second than the unmodified KVM system. As stated previously, this benchmark requires many subsystems in addition to the network, leading to even higher overhead due to non-network causes. However, we show that in this complex case, we still manage the impressive energy reduction of up to 16.2%, while improving performance. This improvement in performance is because the system resources are now freed to do useful work inside the VM such as answering queries or rendering PHP pages.

In Figure 13, we present the results of the Tbench test, comparing the power consumption of the Bare metal system to the unmodified KVM system and to a KVM system running the modified driver with varying buffer sizes. The unmodified KVM system consumes 50.9 watts at the wall, and 23.3 watts at the CPU. Running the modified KVM driver with a buffer of 0.1ms, the power consumption is 49.5 watts at the wall, and 23.1 at the RAPL counter. With a larger buffer size, the power consumption improves, down to 49.0 watts at the wall and 22.8 watts at the CPU with a buffer size of 0.25ms. With a buffer size of 0.5ms, the system consumes 47.6 watts at the wall and 22.0 watts at the CPU, an improvement of 6.5% and 5.6% respectively over the unmodified KVM. The largest buffer size we test, 1ms, does have lower power consumption but unfortunately it suffers from an approximately 6% decrease in throughput. However, the buffer sizes 0.1ms 0.25 ms, and 0.5 ms have identical performance to their KVM base-line. The loss of performance with Tbench as our buffer sizes approaches 1ms is likely due to the fact that network file systems are more susceptible to increases in latency than our web server benchmark RUBBoS, for example. Tbench serves to illustrate that many applications benefit from this buffering technique, however the amount of buffering to ensure optimal performance of the application could vary.

VIII. Conclusion and Further Discussion

In this paper we showed, through both external power meter measurements and internal RAPL hardware profiling counters, the energy overhead created by using virtualized systems. We found that due to virtualization overhead, hypervisor based virtualization systems such as KVM and Xen can consume considerably more energy when running typical network tasks. On the other hand, we find that the container virtualization system OpenVZ consumes near identical amount of power as our non-virtualized baseline system.

Our initial experiments with other complex network applications, such as the distributed memory object caching system `Memcached` [25] and Online Transaction Processing (OLTP) benchmarked by `sysbench` [26], their energy consumption does decrease but at the cost of a slight loss of performance. After inspecting these two systems, we conjecture the following reasons. First, both applications' benchmarks appear to employ a stage that incorporates blocking I/O, meaning that any delay in response will slow down the process of the entire benchmark. Second, both applications are sensitive to increases in latency, especially memcached, which is designed to be a low latency memory object caching system. However, it is likely that non-blocking I/O implementations of these application would also see improvement in energy consumption.

Further, our modifications to KVM's VirtIO driver are likely compatible with recent advances in VM packet switching such as the VALE [27] software switch. For a future work we plan to analyze the energy consumption of these advanced packet switching techniques as well as test their performance with our VM packet buffering techniques.

We show that it is possible to conserve energy without loss of performance for many typical networked applications through the use of packet buffering. Our real world practical modifications show that a busy virtualized web server can reduce it's energy consumption by over 16% by buffering its incoming and outgoing packets. However, it remains to be discovered what other networked applications can benefit from our optimizations.

Acknowledgments

H. Wang's work is supported by a Start-up Grant from the University of Minnesota at Duluth. J. Liu's research is supported by a Canada NSERC Discovery Grant, an NSERC Strategic Project Grant, and a China NSFC Major Program of International Cooperation Grant (61120106008).

References

[1] CDW, "Cdw's server virtualization life cycle report, january 2010."

[2] C. Bash and G. Forman, "Cool job allocation: measuring the power savings of placing jobs at cooling-efficient locations in the data center," in *2007 USENIX Annual Technical Conference on Proceedings of the USENIX Annual Technical Conference*. USENIX Association, 2007, pp. 1–6.

[3] A. Banerjee, T. Mukherjee, G. Varsamopoulos, and S. K. Gupta, "Cooling-aware and thermal-aware workload placement for green hpc data centers," in *Green Computing Conference, 2010 International*. IEEE, 2010, pp. 245–256.

[4] A. Beloglazov and R. Buyya, "Energy efficient resource management in virtualized cloud data centers," in *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*. IEEE Computer Society, 2010, pp. 826–831.

[5] R. Urgaonkar, U. C. Kozat, K. Igarashi, and M. J. Neely, "Dynamic resource allocation and power management in virtualized data centers," in *Network Operations and Management Symposium (NOMS)*. IEEE, 2010, pp. 479–486.

[6] A. Berl, E. Gelenbe, M. Di Girolamo, G. Giuliani, H. De Meer, M. Q. Dang, and K. Pentikousis, "Energy-efficient cloud computing," *The Computer Journal*, vol. 53, no. 7, pp. 1045–1051, 2010.

[7] L. Liu, H. Wang, X. Liu, X. Jin, W. B. He, Q. B. Wang, and Y. Chen, "Greencloud: a new architecture for green data center," in *Proceedings of the 6th international conference industry session on Autonomic computing and communications industry session*. ACM, 2009, pp. 29–38.

[8] R. Buyya, A. Beloglazov, and J. Abawajy, "Energy-efficient management of data center resources for cloud computing: A vision, architectural elements, and open challenges," *arXiv preprint arXiv:1006.0308*, 2010.

[9] A. Kansal, F. Zhao, J. Liu, N. Kothari, and A. A. Bhattacharya, "Virtual machine power metering and provisioning," in *Proceedings of the 1st ACM symposium on Cloud computing*. ACM, 2010, pp. 39–50.

[10] B. Krishnan, H. Amur, A. Gavrilovska, and K. Schwan, "Vm power metering: feasibility and challenges," *ACM SIGMETRICS Performance Evaluation Review*, vol. 38, no. 3, pp. 56–60, 2011.

[11] Q. Huang, F. Gao, R. Wang, and Z. Qi, "Power consumption of virtual machine live migration in clouds," in *Communications and Mobile Computing (CMC), 2011 Third International Conference on*. IEEE, 2011, pp. 122–125.

[12] Y. Jin, Y. Wen, and Q. Chen, "Energy efficiency and server virtualization in data centers: An empirical investigation," in *Computer Communications Workshops (INFOCOM WKSHPS), 2012 IEEE Conference on*. IEEE, 2012, pp. 133–138.

[13] P. Padala, X. Zhu, Z. Wang, S. Singhal, and K. Shin, "Performance evaluation of virtualization technologies for server consolidation," *HP Labs Tec. Report*, 2007.

[14] K. Ye, X. Jiang, S. Chen, D. Huang, and B. Wang, "Analyzing and modeling the performance in xen-based virtual cluster environment," in *2010 12th IEEE International Conference on High Performance Computing and Communications*, 2010, pp. 273–280.

[15] R. Russell, "virtio: towards a de-facto standard for virtual i/o devices," *SIGOPS Oper. Syst. Rev.*, vol. 42, pp. 95–103, July 2008.

[16] S. Soltesz, H. Pötzl, M. E. Fiuczynski, A. Bavier, and L. Peterson, "Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors," *SIGOPS Oper. Syst. Rev.*, vol. 41, March 2007.

[17] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, "The eucalyptus open-source cloud-computing system," in *Proc. of the 2009 9th IEEE/ACM CCGRID*, 2009.

[18] Intel, "Intel xeon processor e5 family: Spec update." [Online]. Available: http://www.intel.com/content/dam/www/public/us/en/documents/specification-updates/xeon-e5-family-spec-update.pdf

[19] Iperf, http://iperf.sourceforge.net/.

[20] R. Shea and J. Liu, "Network interface virtualization: challenges and solutions," *Network*, vol. 26, no. 5, pp. 28–34, 2012.

[21] M. Hefeeda and C.-H. Hsu, "On burst transmission scheduling in mobile tv broadcast networks," *IEEE/ACM Transactions on Networking (TON)*, vol. 18, no. 2, pp. 610–623, 2010.

[22] [Online]. Available: http://jmob.ow2.org/rubbos.html

[23] [Online]. Available: http://apache.org/

[24] [Online]. Available: http://linux.die.net/man/1/tbench

[25] [Online]. Available: http://memcached.org/

[26] Sysbench, http://sysbench.sourceforge.net/.

[27] L. Rizzo and G. Lettieri, "Vale, a switched ethernet for virtual machines," in *Proceedings of the 8th international conference on Emerging networking experiments and technologies*. ACM, 2012, pp. 61–72.