

A Deep Investigation Into Network Performance in Virtual Machine Based Cloud Environments

Ryan Shea*, Feng Wang[†], Haiyang Wang*[‡] and Jiangchuan Liu*

*Simon Fraser University, British Columbia, Canada

[†]University of Mississippi, Mississippi, USA

[‡]University of Minnesota at Duluth, Minnesota, USA

Email: rws1@cs.sfu.ca, fwang@cs.olemiss.edu, haiyang@d.umn.edu, jcliu@cs.sfu.ca

Abstract—Existing research on cloud network (in)stability has primarily focused on communications between Virtual Machines (VMs) inside a cloud, leaving that of VM communications over higher-latency wide-area networks largely unexplored. Through measurement in real-world cloud platforms, we find that there are prevalent and significant degradation and variation for such VM communications with both TCP and UDP traffic, even over lightly utilized networks. Our in-depth measurement and detailed system analysis reveal that the performance variation and degradation are mainly due to the dual-role of the CPU in both computation and network communication in a VM, and they can be dramatically affected by the CPU's scheduling policy. We provide strong evidence that such issues can be addressed in the hypervisor level and present concrete solutions. Such remedies have been implemented and evaluated in our cloud testbed, showing noticeable improvement for long-haul network communications with VMs.

I. INTRODUCTION

Recently, cloud computing has attracted significant interests from both industry and academia. Companies such as Amazon [1], Google [4] and Microsoft [13] have announced their own platforms to provide public cloud computing services. A great number of applications have been migrated to or developed for cloud platforms to enjoy the elastic and cost-efficient resource provisioning, where the spectrum ranges from traditional computation-centric services such as scientific computing, large scale data processing, and enterprise service, to newly emerged network-centric services including web service, file sharing, file synchronization, and video streaming.

In most cloud computing platforms, virtual machine (VM) technology is the core technology enabling a single physical machine to be split into several VMs. By this means, cloud providers can decrease operational costs through server consolidation and better utilization of computational resources. Cloud users can also easily outsource their computational requirements and purchase suitable cloud instances accordingly. The benefit of VM technology, however, comes together with overhead, affecting virtually every aspect of the computer system, especially the network subsystem. For example, it is reported that current network communications on VMs (e.g., Amazon EC2 instances) may experience significant throughput instability and abnormal delay variations [19]. Despite this pioneering research many issues remain unexplored. For example, existing research has mainly focused on performance issues due to network instability in low latency environments inside the cloud. The effect of this instability on the performance of network protocols running over higher latency,

wide area networks has seldom been explored. Also, the effect of computational load inside a cloud virtual machine and its effect on network performance and stability has yet to be quantified. It has been shown, in the vSnoop developed in Purdue University, that the TCP performance with Xen virtualization can be improved by offloading TCP acknowledgements to the driver domain [11]. We however show that not only does TCP performance degrade when the physical system is under heavy use but the packet drop rate can increase by up to 20%, indicating TCP acknowledgement offloading alone is not enough to fix the underlying performance issues. More recent research has been conducted into packet delay in virtualized system hosted on single core physical machines [20]. It remains to be determined what other factors contribute to network delay in cloud virtualized systems when they are hosted in a more complex multi-core environment.

In this paper, we systematically investigate the virtualization overhead on network performance in VM based cloud platforms, striving to not only understand the performance degradation and variation phenomena clearly but also dive deeply to uncover the root causes. To this end, we take the Xen hypervisor based cloud Amazon EC2 as a case study, conducting extensive measurements on the network performance of its large instances. Our measurement results show that the network performance degradation and variation phenomena can be prevalent and significant even within the same data center. Taking TCP traffic as an example, among 10 instances we randomly pick from Zone A in Amazon's Oregon data center, up to 8 of them have experienced certain levels of throughput degradation (with the maximum up to 50%) even when the VMs' virtual CPUs are nearly idle. When the VMs are fully utilized with computation tasks, the throughput degradation can be as much as 87%. For UDP traffic, the throughput can also fall about 19% when a VM is heavily utilized and the packet drop rate can increase to nearly 20%. In addition, we find that the round trip time on a VM fully utilized with computation tasks can rise from around 10ms to as high as 67ms.

To fully understand the root cause of these phenomena, we conduct detailed system analysis on Xen's CPU scheduler and network architecture. Our analysis reveals that the network performance variation and degradation are mainly due to the "interference" between the CPU utilization for computation and network communication, causing the network communication to be starved for CPU. In particular, the "netback" process of a VM in the driver domain cannot be scheduled on a CPU in

time, which causes the arrived network packets to stay in the receiving buffer of the driver domain for an extra period that does not exist in a normal non-virtualized machine, quickly overflowing the buffer and causing later arriving packets to be dropped. Moreover, we identify the two typical scenarios that lead to such a situation, namely, “co-runner interference” and “self interference”. The former scenario mainly happens among different VMs, where a VM or the netback process of a VM is starved due to waiting for the scheduling of other VMs. And the latter scenario happens within a VM, where the netfront process of the VM is starved because other processes (mainly computation processes) have used up the CPU resources allocated to this VM.

To verify our analysis, we further set up a Xen based cloud environment similar to Amazon EC2 on our local cloud testbed and conduct extensive experiments to carefully examine our findings. The results match well with what we have observed directly from the Amazon EC2 platform and confirm the validity of our analysis.

Based on our findings, we propose a solution to improve and stabilize the network performance. We develop a scheme to schedule the VMs mainly with computation tasks and the VMs mainly with network communication tasks to separate CPU pools. This way VMs with mainly network communication tasks can achieve excellent network performance while still keeping reasonably good computation performance, without disturbing and degrading the performance of the VMs with mainly computation tasks. We conduct an evaluation on our cloud testbed. The results further demonstrate the effectiveness of our solution.

II. RELATED WORK

In addition to the works already cited the following research has been conducted on cloud network performance. Researchers at Duke University and Microsoft Research have recently proposed a frame work for comparing public cloud providers titled *CloudCMP* [12]. CloudCMP allows organizations to compare public clouds on metrics such as computational performance, price, and storage offering. Ostermann *et al.* [14] evaluated Amazon EC2’s suitability for high performance scientific computing. The researchers found that the use of virtualization can impose significant performance penalties on many scientific computing applications. In 2011, Shieh *et al.* [17] discussed the network performance issues specific to multi-tenant data centers. The authors introduced *Seawall*, a network bandwidth allocation scheme, which can fairly share the data-center’s resources according to administrator policies. Further research related to network sharing in cloud based multi-tenant data-centers was preformed by Popa *et al.* [15]. The authors investigated issues related to minimum guaranteed and proportional bandwidth sharing in a cloud computing context. A set of bandwidth allocation strategies were proposed to both improve performance and fairness of these systems. There have also been many performance analysis testing the suitability and performance of virtualized system for specific applications and scenarios. [7][3][9][10].

III. NETWORK PERFORMANCE IN VM-BASED CLOUD

Our measurement goals are to find out how network bandwidth, delay, and stability are affected by the variations of

CPU loads. Further, we want to discover what differences in network performance exist between instances of the same class, performing the same task, in the same data center. To ensure an accurate and representative analysis we collect a wide arrangement of samples from cloud instances of the same class. Also, to ensure accuracy our experiments must not rely on a single protocol. To this end, we devise the following cloud *instance selection methods* and *measurement settings*.

For our cloud instance selection we choose the current industry leader in cloud computing Amazon EC2 as our cloud provider. We choose Amazon EC2 not only because it is the current market leader but also because of its heavy use of the open source Xen hypervisor to provide virtual machines. Amazon’s use of Xen allows us to closely investigate performance issues by looking into the architecture and source code. Also, Xen is used by numerous other large cloud providers such as Rackspace [16], making our findings more general and applicable to other clouds. To accurately measure network performance, we provision Amazon EC2 “Large” instances. The choice of Large instances is motivated by two key factors. First, they provide ample computation and memory resources for most web applications including our benchmarks and experiments. Second, they are rated to have “high” I/O capabilities, making them a natural choice for bandwidth intensive applications. The choice of large instances also allows our results to be more general since smaller instances may simply not be provisioned with enough resources to complete our benchmarks. It is also expected that smaller instances would show the same behaviors or worse since they lack much of the resources granted to their larger counterparts.

For our measurements we require the ability to target loads on the network and CPU subsystems. To this end, we choose *iperf* [8] and *sysbench* [18], respectively. *Iperf* is a widely used configurable network benchmark, allowing a user to gauge the performance of both TCP and UDP data flows. *Sysbench* is chosen for CPU benchmarking because it is widely used and can be configured to target the CPU using its prime number benchmark. Using these targeted benchmarks we can easily apply certain amount of computation loads to CPU and measure the network performance accordingly. We can then compare the performance across different scenarios and load patterns. Further, the use of these two targeted applications also allow us to view the direct consequence of CPU load on the network protocols, whereas a more complex application may suffer from application specific problems when being migrated to the cloud environment. For all measurements our CPU benchmarks are set to use the lowest Linux scheduler priority using the “nice” command. The use of the lowest nice priority for the CPU benchmark ensures that the network task *Iperf* will be given priority to run on the VM’s virtual CPU. By doing this we ensure that inside the VM the CPU benchmark does not take CPU cycles needed to process our network traffic. For operating system selection we choose Ubuntu 12-04 64 bit, and the packages are installed from the Ubuntu repository.

To fully understand the network performance degradation and variation phenomena, our measurements investigate a wide range of settings, across both TCP and UDP traffic, and for

each type of traffic, examining both sending and receiving performance. We also consider different situations such as whether there are computation loads on VMs' CPUs or not. In addition to throughput and packet drop rate, we also measure round trip time by ICMP, striving to identify potential relations among our findings. In the following subsections, we first discuss our results on TCP traffic. We then proceed with UDP traffic and present our round trip time measurement by ICMP thereafter.

A. TCP Traffic

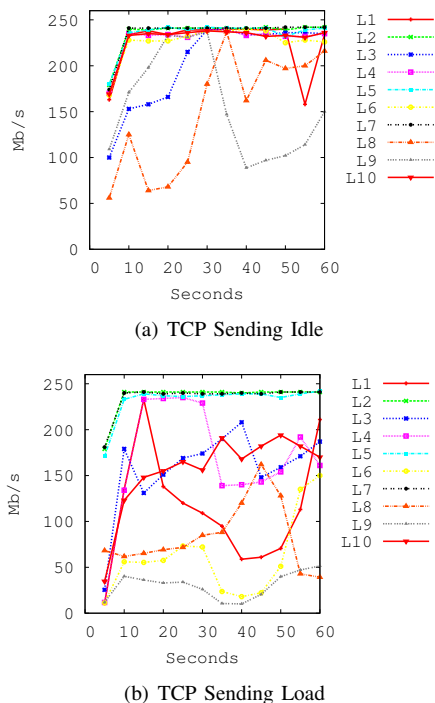


Fig. 1. Ten EC2 Large Instances Sending TCP Traffic

For our TCP measurement, we provision 10 EC2 large instances from availability Zone A in Amazon's Oregon data center. Using 10 instances we will be able to accurately find the bandwidth and delay variation between instances of the same class while the CPU is both *idle* and *fully utilized*. We define idle to be only Iperf and required operating systems services running. Iperf and the operating system consume less than 10% of the system resources while running. Fully utilized is where we also saturate the remaining CPU with our Sysbench prime number benchmark. We also configure a test system outside of the Oregon cloud. The test system has a maximum network throughput of 300 Mb/s and is used to send and receive data to and from our cloud instances. We number our EC2 large instances from L1-L10 and test each one sequentially under the following conditions: 1) Maximum achievable sending bandwidth while instances are idle and under computation load, and, 2) Maximum achievable receiving bandwidth while the instances are idle and under computation load. Iperf is configured to run for 60 seconds and report the bandwidth average every 5 seconds. The tests are run sequentially to minimize the interference from potential

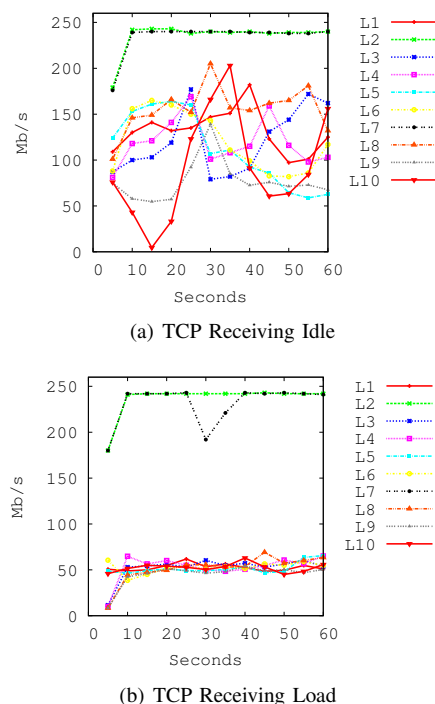


Fig. 2. Ten EC2 Large Instances Receiving TCP Traffic

changes in network conditions and topology at Amazon's data center.

The results from the sending and receiving measurements are given in Figure 1 and Figure 2, respectively. They show that even with identical specification, large variations in performance can be observed when the systems are under load, and even in many cases, when the systems are idle. We first focus on the sending scenario.

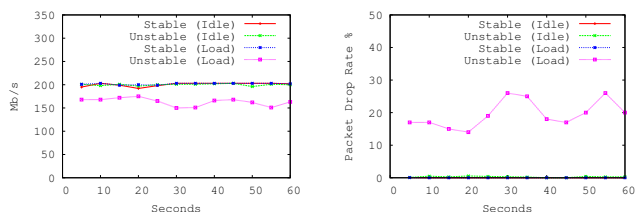
Under the sending measurement, each of our 10 instances sends data using TCP to our remote host. As shown in Figure 1 the base line performance of the 10 instances are relatively similar, with most of the instances reaching a stable sending rate of approximately 240 Mb/s. 3 of the instances however experience significant instability over the 60 seconds of measurement period. One of the unstable instances reaches an average of only 200 Mb/s, a drop of nearly 17% on throughput. The other two unstable instances show even weaker performance at less than 160 Mb/s, a drop of over 30%. Next, we introduce the CPU load using Sysbench and once again send data to our remote host. The results are given in Figure 1b. It is easy to see that although 3 of the instances show nearly identical bandwidth to their baseline idle performance, 7 out of 10 instances now show tremendous variation in terms of bandwidth. In particular, 4 of the instances drop over 30% to less than 160 Mb/s. Two instances fall more than 66% to less than 80 Mb/s, and one of the instances falls even lower to 30 Mb/s, an incredible drop of over 87%.

Next we look at the receiving measurements. The baseline performance is given in Figure 2a, where 8 out of 10 instances suffer from severe bandwidth instability even when under near idle conditions. The 2 stable instances have a receiving bandwidth of 235 Mb/s. 6 of the unstable instances achieve a performance of less than 150 Mb/s, a considerable loss of over

35% on throughput when compared to the two stable instances. Furthermore, another 2 large instances achieved even lower performance with well under 100 Mb/s, a drop of over 50%. We now focus on the measurements under computation load which can be found in Figure 2b. Once again the stable instances from the previous idle experiment remain at nearly the same bandwidth. One instance does temporarily drop its receiving rate but recovers quickly, this is possibly due to a temporary loss of packets. Interestingly, all the unstable instances now stabilize, unfortunately all 8 instances stabilize at only about 50 Mb/s. It is clear that the introduction of computation load on these instances can have a devastating effect on the bandwidth performance of many cloud instances. In addition, receiving traffic can suffer worse than sending traffic in terms of performance loss and variation.

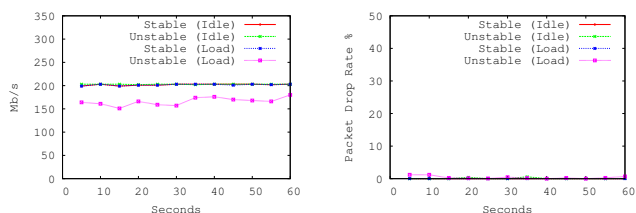
The measurements on TCP traffic already show that even when instances are all of the same class, allocated in the same data center, communicating to the same end host, and under the same conditions, there can be considerable differences in terms of network performance. In section IV we will show that overloaded physical hosts, mixed with the high overhead introduced by network interface virtualization, are likely the cause of this issue. Another finding from our measurements is that there are generally two different types of instances. One type is like the 2 instances that are rarely affected by changes in computation load inside the virtual machine. We thus call them “stable” instances. The other type is like the 8 instances that have considerably worse performance when under load. We thus refer to them as “unstable” instances. It is worth noting that preliminary measurements using other EC2 data centers, namely California and Virginia, indicate these issues are not limited to Amazon’s Oregon data center. For a more detailed investigation on the network conditions such as jitter and drop rate, we now move onto our measurements on UDP traffic.

B. UDP Traffic



(a) UDP Bandwidth(Receiving) (b) UDP Packet Drop(Receiving)

Fig. 3. EC2 Large UDP performance Receiving 200 Mb/s



(a) UDP Bandwidth (Sending) (b) UDP Packet Drop Rate (Sending)

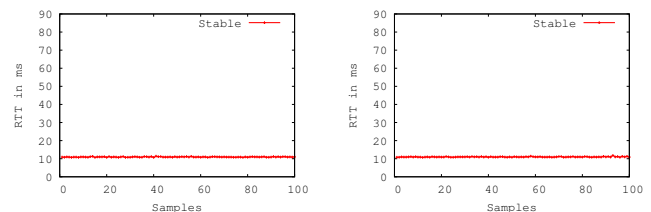
Fig. 4. EC2 Large UDP performance Sending 200 Mb/s

Unlike TCP, UDP’s stateless best effort packet delivery allows us to directly observe the maximum throughput and

data loss rate of our instances. This is because we will not have any issues with TCP flow control and retransmissions.

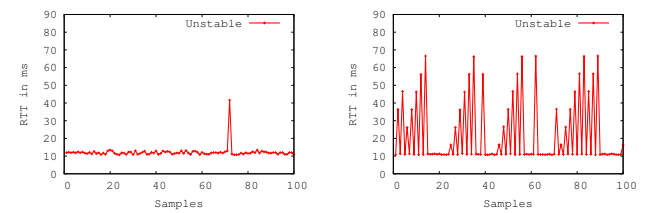
We select two representative instances from our previous experiment so we have one “stable” instance and one “unstable” instance. We test each instance while sending and receiving at their maximum rate using the same remote system described in the TCP experiment. Since we must select a data rate for UDP we decided on 200 Mb/s, which could easily be sustained by both machines while idle. Once again we use Sysbench running at the lowest Linux nice priority. We show results for data-rate and packet loss rate. Once again each test is run for 60 seconds, and each system is tested while idle and under load.

First we describe the receiving experiments, the total UDP receiving rate is given in Figure 3a and the packet drop rate is given in Figure 3b. As can be seen while the systems are idle both the unstable and stable instance maintain a receiving rate of 200 Mb/s. When we run our Sysbench CPU prime experiment concurrently with our UDP receiving task the incoming bandwidth of the unstable instance falls nearly 19% to 163 Mb/s. The stable instance on the other hand shows no noticeable drop in bandwidth while under load. We now focus on the drop rates, which can be found in Figure 3b. As can be seen our unstable instance increases from a near negligible drop rate when idle to a massive increase of nearly 20% when under load. We will show in section IV that this increase is most likely due to long CPU scheduling delays between a Xen Virtual machine and the VM’s “Netback” thread. The Netback thread runs in Dom0 of the Xen system and is responsible for moving packets to and from the Virtual Machine. Since the packets are not delivered to the VM the incoming packet queue becomes full in the physical machine which manifests itself in the large amount of data-loss observed in our unstable instance. This is happening despite the fact that the CPU bound task has been set to the lowest nice priority provided to us by the Linux operating system.



(a) RTT idle (Stable Instance) (b) RTT load (Stable Instance)

Fig. 5. ICMP Round Trip Time Stable EC2 Large Instance



(a) RTT idle (Unstable Instance) (b) RTT load (Unstable Instance)

Fig. 6. ICMP Round Trip Time Unstable EC2 Large Instance

Next we describe the sending experiments; the total UDP sending rate is given in Figure 4a and the packet drop rate is

given in Figure 4b. The throughput remains at the optimal 200 Mb/s for both our instances when no load is introduced inside the VM. However, when we activate our CPU benchmark the performance of the Unstable instances tumbles 37 Mb/s to 166 Mb/s, a drop of over 18%. The stable instance shows no measurable change in outgoing bandwidth. We now turn our attention to the packet drop rate seen in Figure 4b. The result shows no increase for the stable instance while under a CPU load and only a minor increase for the unstable large instance. When contrasted with the results of our UDP receiving experiment, we notice that although the data rates for our unstable instance falls by a similar degree, the packet drop rates are very different. In the sending experiment introducing a CPU load on our unstable instance does not noticeably increase the packet drop rate but in our receiving experiment the drop rate jumps to nearly 20%. This observation indicates that while sending, the sending buffer inside the VM becomes full, causing the the operating system to reduce the speed of the UDP application. The buffer becomes full because the netback thread is not able to get enough CPU time to forward the packets to the physical network. The reduction in rate means that no measurable increase in drop rates is found. Receiving, on the other hand, overloads a receiving buffer in Dom0 controlled by the netback thread. Since UDP has no native notion of flow-control our remote host continues to send at its full 200 Mb/s and many of these packets are dropped by the physical machine.

C. Round Trip Time by ICMP

We next devised an experiment to test the effect CPU load had on the round trip time (RTT) packets sent to our cloud instances. Once again we select a representative Stable large instance and a representative Unstable large instance. We use the send a ICMP echo request packet (ping) with 100 bytes of data and record the amount of time to receive the ICMP echo response from the cloud server. We use a local workstation to perform this test and send an echo request every 100 ms for ten seconds. We use the hping3 tool to create our packets for this experiment. Once again we test the instance while idle and while under a high CPU load generated by the Sysbench CPU prime benchmark. Again we set Sysbench to have the lowest priority. The results for the stable instance are given in Figure 5 and the unstable are given in Figure 6.

This experiment generated some interesting results. As can be seen in Figure 5a and Figure 5b the stable instance showed no measurable increase in RTT with the average of both tests being approximately 10ms. As can be seen in Figure 6a the unstable instance maintains a RTT nearly identical to the stable instance while idle. However, when the load is introduced the average RTT doubles to over 20 ms with some packets samples showing an RTT as high as 67 ms.

IV. SYSTEM ANALYSIS AND VERIFICATION

To further examine the root causes of the unstable performance experienced by our EC2 large instances in the previous section, we will now perform a system analysis and confirm our findings on our local cloud. The use of a local cloud is essential to our understanding of the underlining issues, since public cloud computing platforms such as Amazon EC2 do not allow us to monitor the physical machines directly. To this end

we devise a set of experiments on our local platform to illuminate what subsystems are causing the massive performance variation and loss experienced in our previous experiments.

A. System Analysis

To fully understand the root causes of the unstable performance experienced by our EC2 VMs we must look closely at the architecture of Xen. It is widely known that Amazon EC2 uses a version of the Xen server, for which the core source code is available and open-source thus we implement a Xen based cloud system in our local network. Of particular interest is the Xen network subsystem and the Xen CPU scheduler. We have discovered the interactions between these two subsystems leads to the unstable performance found in our previous experiments. We begin our discussion with a analysis of the CPU scheduler subsystem.

1) *Xen CPU Scheduler*: The Xen hypervisor supports a number of CPU schedulers such as Borrowed Virtual Time (BVT), Simple Earliest Deadline First (SEDF), and the Credit scheduler. The credit scheduler is the most widely deployed Xen CPU scheduler because it provides load balancing across multiple processors cores and can be configured to be work conserving or not based on the requirements of the system administrator [2]. We conjecture the credit scheduler, or a modified version of it, is likely the scheduler used by EC2. We suspect this for two reasons. First, advanced features such as global load balancing are likely required by EC2. Secondly, the credit scheduler has a fixed 30 ms maximum time slice and as our round trip time (RTT) experiment shows in Table 6b the increases in RTT are approximately multiples of 30 ms.

The Xen credit scheduler provides load-balancing for Xen virtual machines by scheduling Virtual CPU (VCPU) based on a weighted priority and optionally a maximum utilization cap. It should be noted that the use of the maximum utilization cap makes the credit scheduler behave in a non-work conserving fashion, this is because even if there are idle physical resources the scheduler will not assign them to a VM which has exceeded its utilization cap. The weighted priority is an integer assigned by the system administrator to each VM running on the Xen system. The weight represents a proportional share of the CPU, for example if virtual machine VM-1 has the default scheduler weight of 128 and VM-2 has a weight of 256, VM-2 will be scheduled twice as often on the physical CPUs. The utilization cap is provided as a percentage and corresponds to the amount of physical CPU time which can be consumed. For example, a cap of 50% corresponds to half of a physical CPU and a cap of 200% indicates the VM can fully occupy 2 physical CPUs. Figure 7 shows a simplified version of the Xen credit schedulers global load balancing. As can be seen we have a number of VCPUs belonging to different VMs running on our 4 physical CPUs. Each one is given a maximum time slice of 30 ms. Execution for these VCPUs end when the virtual machine consumes its time slice, blocks, or yields its remaining time. In the example in figure 7 VM-3's Virtual CPU-2 consumes its entire time slice and is placed at the end of the run queue. VM-1 is at the front of the run queue so its VCPU is scheduled to run on CPU-3. It is important to note that a VM that is de-scheduled due to an expired time slice or blocking is not necessarily sent to the

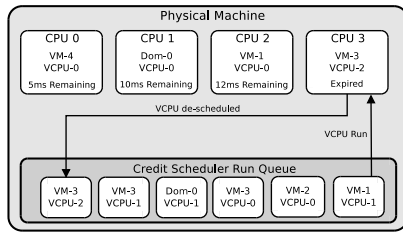


Fig. 7. Xen Credit Scheduler

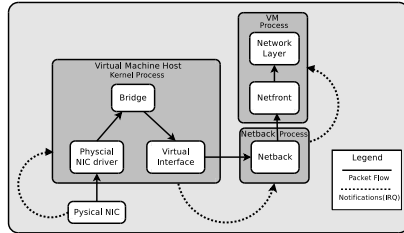


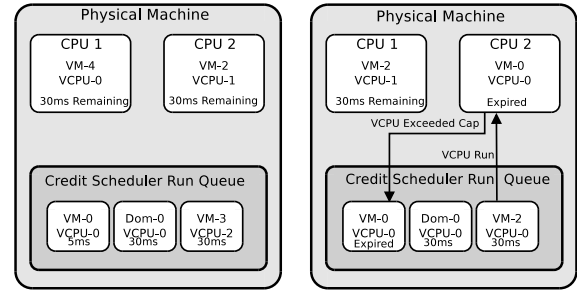
Fig. 8. Xen Simplified Network Architecture

end of the run queue. For example, a VM with a high priority weight can be placed in front of lower priority VMs. Also, a VM which blocks or yields without consuming its time slice will be placed in front of VMs which have consumed their time slice.

2) *Xen Network Architecture*: The Xen network subsystem must pass packets through a number of steps and CPU process in order to deliver the packet to a VMs' network layer. Figure 8 shows a simplified drawing of the steps involved in delivering packets to a Xen VM. On receipt of a packet in Xen, the physical NIC driver delivers the packet to the bridge; Once the packet arrives at the bridge it is transferred to the Netback module, which allocates buffers for the packet and notifies Netfront, in the Guest VM, of the incoming packet; Finally, Netfront receives the packet and passes it to the guests' network layer. As can be seen the packet must traverse several processes in multiple domains before it is finally delivered to the network layer in the virtual machine. The netback process for example runs in the driver domain (Domain-0) and is responsible for copying packets to and from the virtual machine's memory space. The netfront module runs inside the virtual machine, essentially operating as a network interface controller. Since these two modules run in different Domains there is potential that the credit scheduler can schedule these Domains far apart in time. For example, the netback process for a VM may be set to run at the front of the run queue and our VM is at the end of the queue because it consumed all its credits on a CPU task. Potentially this VM must wait 30ms now before it can process its packets. Next, we will discuss how the network architecture and CPU scheduler interact with each other to form the unstable performance experienced by our EC2 instances.

3) *Reasons for Delay and Bandwidth Instability*: We can roughly split the delay and bandwidth instability into two categories. The first category we define as "co-runner interference", which is the bandwidth and delay instability experienced when the VM's VCPUs are working in a near idle condition. For example, the VM's VCPU is only processing network traffic and not performing any computational intensive tasks. The second category we define as "self interference",

which is bandwidth and delay instability experienced when the VMs VCPU are running both bandwidth and computational intensive tasks. We start our discussion with co-runner interference.



(a) Unstable Performance (near idle VCPU) (b) Unstable Performance (Saturated VCPU)

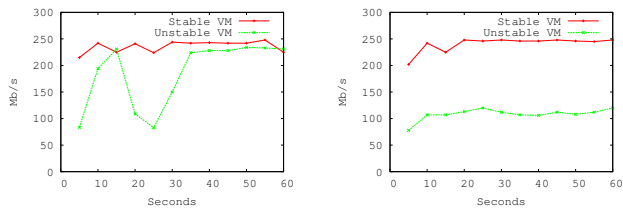
Fig. 9. Unstable Performance

Co-runner interference was a major contribution factor to the unstable TCP performance experienced in Figure 1, by 8 out of 10 of our EC2 large instances while they were idle. This instability is created by the Xen scheduler scheduling bandwidth intensive and computational intensive virtual machines to the same physical processors. Since the bandwidth intensive tasks rarely use their whole time slice they often yield and are replaced by computational VMs, which always consume their entire 30ms time slice. Also, as discussed previously the netback process for the VM runs in the driver domain (Domain-0), which can also be blocked from running for long periods by these computational intensive tasks. If either Domain-0 or our bandwidth intensive VM have to wait a multiple of 30 ms between executions on the CPU, the TCP congestion algorithm will detect a congestion event, leading to a drop in throughput. Figure 9a shows an example where VM-0 is our bandwidth intensive VM, which has only 5ms CPU processing to perform when it is scheduled to run. However, all physical CPUs are currently occupied by VMs which will use their entire time slice of 30ms, and the next two VCPUs on the run queue also have 30ms of processing. In this example VM-0 will have to wait up to 60 ms before it gets a chance to run creating a significant delay in its packet processing. These delays in processing have very little effect on our idle large instances' UDP performance seen in Table 6, this is because UDP does not control for congestion so infrequent spikes in the RTT do not effect the bandwidth. Next we discuss bandwidth/delay instability due to self interference.

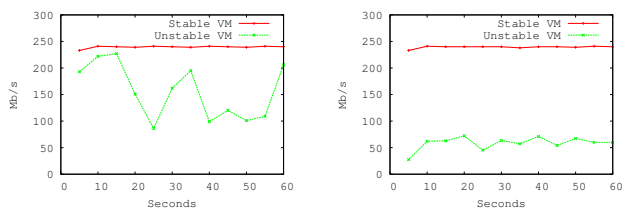
Since a VM currently has no way to indicate the priority of its tasks running on its VCPUs to the Xen hypervisor, a low priority CPU processing task can consume an entire 30ms time slice that may have actually been required for stable network performance. If a VM expends its entire CPU time slice it will likely go to the end of the credit scheduler run queue and have to wait a considerable amount of time before it reaches the front of the queue again. Figure 9b shows a bandwidth intensive virtual machine, VM-0, which has used its entire CPU time slice on a low priority computational task. VM-0 is de-scheduled and placed at the end of the run queue, where on a busy physical machine it could be 30ms or more before VM-0 is scheduled again. If the Xen system administrator has decided to use the Xen credit scheduler

CPU cap option on the VM-0 it will not be scheduled again until the next scheduler accounting period even if there is idle resources on the physical system. This is because the credit scheduler cap option enforces rigid execution rules on a VM and is not necessarily work conserving. Our previous EC2 experiments showed that self interference creates terrible performance for the eight unstable instances. Not only does the RTT increase dramatically in Figure 6 but the UDP experiments in Figure 3 make it clear that the data rate of these instances drop considerably for both sending and receiving. In the receiving UDP experiments we also see an increase in drop rate because the VM's netback thread must drop packets, likely because the VM is not emptying the Domain-0 receive buffers often enough. The UDP sending experiment sees no increase in drop-rate since it is likely the VMs sending buffers are now full inside the VM and it is not given enough CPU time to pass the data to the netback thread; because of this the VM's operating system slows the UDP sending rate of the instance.

Finally, it is important to note that the two kinds of interference are not necessarily mutually exclusive. For example a Xen virtual machine with unstable idle performance will likely have even poorer performance when it saturates its VCPUs.



(a) TCP Sending Idle (b) TCP Sending Load
Fig. 10. Local Xen Instances Sending TCP Traffic



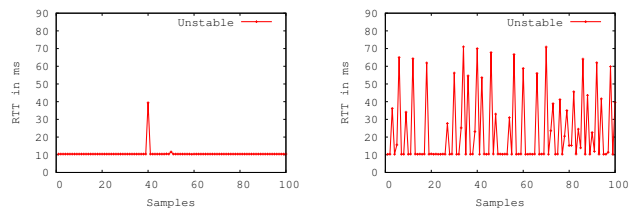
(a) TCP Receiving Idle (b) TCP Receiving Load
Fig. 11. Local Xen Instances Receiving TCP Traffic

B. Verification Experiments

Our local cloud is a cloud platform based on Citrix's Cloud Stack management software. Each node has 6 CPU cores, 16 GB of main memory and 500 GB of local storage. Networking is provided through 1000 Mb/s switches. The hypervisor used on our cloud nodes is Xen 4.1, which as of this writing is the latest stable version. The CPU scheduler algorithm used on our platform is the Xen credit scheduler. We create a cloud instance with similar memory and CPU specifications to an EC2 large instance. An EC2 large instance is configured with 7.5 GB main memory and 2 virtual CPUs with 4 EC2 compute worth of processing power. By comparing the performance of CPU benchmarks on EC2's cloud and our local cloud we have determined that 4 EC2 compute units is equivalent to 1.2 physical cores on our cloud platform. To recreate the network conditions experienced on our EC2 experiments we

limit each VM to a network rate of 250 Mb/s. We also use the network emulator netem to emulate a 10 ms round trip time between our cloud instances and the remote host. Our remote host is a physical machine located one hop away from our platform and connected through 1000 Mb/s Ethernet. Like the EC2 experiments our VMs run Ubuntu 12.04 and any additional software packages are downloaded from the official repositories. We once again use iperf to test network performance and the sysbench CPU prime benchmark to create processor load.

In order to test unstable instances we create a situation similar to that seen in Figure 9a. We create two local VMs similar to EC2 large instances and two VMs similar to EC2 small instances. We assign these 4 VMs to three of our physical systems processor cores. Previously, we calculated each large instance requires 1.2 processor cores and each small instance has 1/4th the processing power we requires 0.3 cores per small VM. The total processor requirement for these 4 VMs is 3 processor cores. To ensure proper sharing of the resources we use the Xen credit scheduler optional cap feature to ensure each VM uses only its proper proportion of CPU time. Next, we configure a single large VM and the two small VMs as computational intensive VMs by running sysbench CPU prime. The remaining large VM is our test VM on which we will test the network performance. To create a stable instance we simply provision a VM on one of our cloud servers, and pin the VM to physical processors which are not shared with others VMs.



(a) RTT idle (Unstable Instance) (b) RTT load (Unstable Instance)
Fig. 12. ICMP Round Trip Time Unstable Local Experiments

1) *TCP experiments:* Similar to our EC2 experiments, we test the maximum TCP bandwidth of our VMs. We configure a single stable VM and a single unstable VM on our local cloud using the technique previously described. We begin with the sending experiments, which are given in figure 10. As expected, the stable VM maintains a data rate of approximately 240 Mb/s both while the VM is idle and when we introduce a computational task using sysbench. The unstable VM, on the other hand, shows inconsistent performance as the TCP rate fluctuates rapidly both while the VMs vcpu is idle and while saturated. The unstable VMs idle TCP sending bandwidth is 186 Mb/s, which is almost 21% lower than our stable instance. When a computational task is introduced inside the unstable VM we see an even further drop in sending performance, which now only measures 108 Mb/s, far less than half the performance of the stable instance.

Next we look at the TCP receiving experiments shown in figure 11. Once again our stable instance maintains consistent performance, receiving at a rate of approximately 240 Mb/s both while its VCPU is idle and while it is saturated. The unstable VM once again shows inconsistent performance both

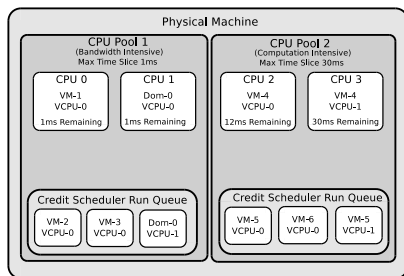


Fig. 13. Multiple Pools for Different VM classes

while idle and with a saturated VCPU. While idle the receiving performance of the unstable instance is only 156 Mb/s a drop of over 34% when compared to the stable instance. When the unstable's VCPU is saturated the performance drops by over 75% to only 59 Mb/s.

As can be seen in our experiments the TCP performance of our unstable local cloud instances are nearly identical to the TCP performance experienced by the EC2 instances in section III. This near identical performance is a very strong indication that inconsistent TCP performance experienced by our EC2 instances is likely due to VCPU scheduling issues in the Xen hypervisor and not necessarily the network conditions in the data-center.

2) *ICMP RTT experiments*: As in the EC2 experiments we wanted to investigate the effect computational work has on the round trip time (RTT) performance of our instances. We used the hping3 tool to create ICMP echo request packets with 100 bytes of data each and sent them to our instances at a rate of 10 per second. As is to be expected the stable instance showed no increase in RTT while its VCPU is idle or when it is saturated with a computational task. Figure 12 gives the unstable instance performance. When the unstable instance's VCPUs are idle we see a single increase to above 30 ms. However, as can be seen introducing a load to the VM causes significant RTT variation. Similar to what was seen in our EC2 experiments these increases are generally in approximate multiples of 30 ms. This is expected as the delay is being caused by the unstable instance exhausting its 30 ms time slice on the CPU and having to wait at least 30 ms to get another chance to run.

V. MITIGATE PERFORMANCE DEGRADATION AND INSTABILITY

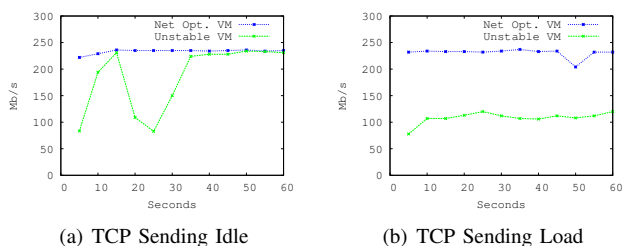


Fig. 14. Sending TCP Traffic effect of 1 ms time slice pool

Since the presence of co-runner interference is not something that can be controlled from inside the VM, we must change the scheduler in order to improve performance. Much work has been done on the Xen scheduler to increase the

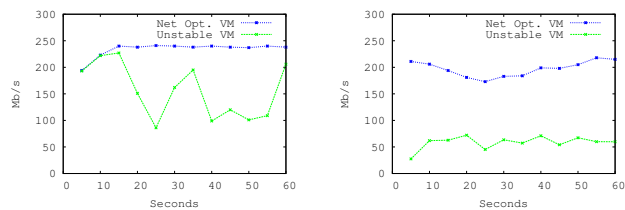


Fig. 15. Receiving TCP Traffic effect of 1 ms time slice pool

performance of latency sensitive applications. For example, work has been done to improve the response time of network based applications hosted in Xen virtual machines [5]. Also, researchers at George Washington University modified the Xen scheduler to prioritize interactive virtual machines, which were used to create virtual desktops [6]. However, these modifications often entail a tradeoff of less efficient computational performance. This lower computational performance is often due to a decrease in the maximum time slice assigned to a virtual machine. This shorter time slice means that VMs are scheduled more often but also results in higher number of processor context switches and lower processor cache efficiency, since VMs are being constantly interrupted while running. These negative side effects are likely one of the reasons why public clouds like EC2 appear to use the standard Xen credit scheduler.

To solve this impasse between the scheduling requirements of computational tasks and network intensive tasks, we propose running two different scheduling strategies in Xen at the same time, one which optimizes network performance and the other which optimizes computational tasks. The owner of the VM is free to choose which scheduling strategy (s)he wants the VM to use. Recently, the Xen maintainers have added the notion of CPU pools into the experimental version of the Xen Hypervisor. CPU pools allow us to pool physical CPUs under different configurations including scheduling rules.

To evaluate the effectiveness of this strategy we install the latest experimental version of Xen on one of our cloud platforms. We configure 2 CPU pools, Pool 1 and Pool 2, each with 3 physical cores. We configure Pool 1 to be our network optimized pool by selecting a maximum CPU time slice of 1 ms. CPU pool 2 is optimized for computational tasks so we select a maximum time slice of 30 ms. For both pools we use the standard Xen credit scheduler. Figure 13 shows the architecture of our Xen system. To recreate co-runner interference we once again create 4 VMs, 2 large VMs and 2 small VMs, and assign them to CPU pool 1. Once again, one large and two of the small VMs are configured to saturate their VCPUs with computational tasks. The remaining large VM is the machine we will test the performance on. We also assign a single VM with 3 VCPUs to pool 2 to ensure the physical system is fully utilized.

The sending results for our network optimized CPU pool experiments are given in figure 14 and the receiving results are given in figure 15. As can be seen, the use of the 1ms credit maximum time slice in our network optimized pool has greatly improved TCP performance both while under load and while idle for sending and receiving. With identical conditions to our previous local experiments, a bandwidth

optimized VM achieves a stable sending rate of 230 Mb/s while the VCPU is idle. Our unstable VM, in the same test, suffers from inconsistent performance and a data rate of only 185 Mb/s. When we introduce a computational load, our network optimized VM achieves near identical performance to its base line. This is a massive improvement as it is experiencing the same interference as the unstable VM but it is not exhibiting the same performance loss. However it does exhibit a small amount of instability, dropping slightly due to self interference and recovering quickly. The unstable VM, on the other hand, exhibits a massive drop in performance, falling to under 110 Mb/s. In our TCP receiving test our network optimized VM stays stable at a throughput of nearly 235 Mb/s while idle. The unstable VM, on the other hand, performs inconsistently, achieving a throughput of only 155 Mb/s. When a computational load is introduced both our network optimized VM and the unstable VM are unable to maintain their baseline performance. However, our network optimized VM manages a performance of nearly 87% of the baseline, receiving at a rate of approximately 200 Mb/s. From our experiments it is clear that using multiple CPU pools, some with network optimizations such as shorter VM CPU time slices, can greatly reduce co-runner interference experienced by virtual machines. Our initial experiments also show that the use of the 1 ms time-slice effects the running time of our CPU tasks by less than 8%. However, the use of bandwidth optimized CPU pools does not solve all issues, as the network receiving performance does fall when a computational task is run along side the network intensive task. It is likely combining our 1 ms CPU pool fix with a TCP acknowledgement offloading technique like the previously discussed vSnoop could improve the performance even further [11].

VI. CONCLUSION AND FURTHER DISCUSSION

In this paper we systematically investigated the effect virtualization overhead has on network performance in public cloud computing. We discover that in public cloud computing platforms such as Amazon EC2, different VMs can suffer from different amounts of network performance instability. For example, some VMs show no signs of instability while others show extreme network performance instability. Through system analysis and local experiments we have discovered that this instability is often caused by computational load on the physical systems and can be roughly split into two types, self interference and co-runner interference. We then implement practical modifications in existing software that show that this performance instability can be mitigated to a large degree.

From our observations in undertaking this set of experiments, we noted that some Amazon cloud datacentre zones have a higher percentage of instances with unstable network performance than others. For example, the majority of EC2 large instances provisioned in Amazon's Oregon datacentre Zone A showed very unstable performance. However, this datacentre recently added a new zone, Zone C; when we allocated our large instances in this zone, a smaller proportion showed unstable performance. This is likely because the new zone has fewer users, and thus currently has a lower utilization. Given time, more users will allocate resources in this zone, and this will lead to an increase in network instability.

Future research in this area should focus on the best route for deploying mitigation strategies in large scale datacentres. Our results are encouraging that network performance and stability can be greatly improved with practical modifications to existing systems.

ACKNOWLEDGMENTS

F. Wang's work is supported by a Start-up Grant from the University of Mississippi. 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] Amazon Elastic Compute Cloud. <http://aws.amazon.com/ec2/>.
- [2] L. Cherkasova, D. Gupta, and A. Vahdat. Comparison of the three cpu schedulers in xen. *Performance Evaluation Review*, 35(2):42, 2007.
- [3] J. Ekanayake and G. Fox. High performance parallel computing with clouds and cloud technologies. *Cloud Computing*, pages 20–38, 2010.
- [4] Google Cloud Platform. <http://cloud.google.com>.
- [5] S. Govindan, A. Nath, A. Das, B. Urgaonkar, and A. Sivasubramaniam. Xen and co.: communication-aware cpu scheduling for consolidated xen-based hosting platforms. In *Proceedings of the 3rd international conference on Virtual execution environments*, pages 126–136. ACM, 2007.
- [6] J. Hwang and T. Wood. Adaptive dynamic priority scheduling for virtual desktop infrastructures. In *Proceedings of the 2012 IEEE 20th International Workshop on Quality of Service*, page 16. IEEE Press, 2012.
- [7] A. Iosup, S. Ostermann, N. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema. Performance analysis of cloud computing services for many-tasks scientific computing. *IEEE Transactions on Parallel and Distributed Systems*, 22(6):931–945, 2011.
- [8] Iperf. <http://iperf.sourceforge.net/>.
- [9] K. Jackson, L. Ramakrishnan, K. Muriki, S. Canon, S. Cholia, J. Shalf, H. Wasserman, and N. Wright. Performance analysis of high performance computing applications on the amazon web services cloud. In *2nd IEEE International Conference on Cloud Computing Technology and Science*, pages 159–168, 2010.
- [10] D. Jayasinghe, S. Malkowski, Q. Wang, J. Li, P. Xiong, and C. Pu. Variations in performance and scalability when migrating n-tier applications to different clouds. In *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, pages 73–80. IEEE, 2011.
- [11] A. Kangarlou, S. Gamage, R. R. Kompella, and D. Xu. vsnoop: Improving tcp throughput in virtualized environments via acknowledgement offload. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–11. IEEE Computer Society, 2010.
- [12] A. Li, X. Yang, S. Kandula, and M. Zhang. Cloudcmp: comparing public cloud providers. In *Proceedings of the 10th annual conference on Internet measurement*, pages 1–14. ACM, 2010.
- [13] Microsoft Server and Cloud Platform. <http://www.microsoft.com/server-cloud/readynow/>.
- [14] S. Ostermann, A. Iosup, N. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema. A performance analysis of ec2 cloud computing services for scientific computing. *Cloud Computing*, pages 115–131, 2010.
- [15] L. Popa, G. Kumar, M. Chowdhury, A. Krishnamurthy, S. Ratnasamy, and I. Stoica. Faircloud: Sharing the network in cloud computing. In *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*, pages 187–198. ACM, 2012.
- [16] RackSpace. <http://www.rackspace.com/>.
- [17] A. Shieh, S. Kandula, A. Greenberg, C. Kim, and B. Saha. Sharing the data center network. In *USENIX NSDI*, volume 11, 2011.
- [18] Sysbench. <http://sysbench.sourceforge.net/>.
- [19] G. Wang and T. Ng. The impact of virtualization on network performance of amazon ec2 data center. In *INFOCOM, 2010 Proc. IEEE*, pages 1–9, 2010.
- [20] J. Whiteaker, F. Schneider, and R. Teixeira. Explaining packet delays under virtualization. *ACM SIGCOMM Computer Communication Review*, 41(1):38–44, 2011.