The Impact of Virtualization on Network Performance of Amazon EC2 Data Center

Guohui Wang T. S. Eugene Ng
Dept. of Computer Science, Rice University

***Abstract*—Cloud computing services allow users to lease computing resources from large scale data centers operated by service providers. Using cloud services, users can deploy a wide variety of applications dynamically and on-demand. Most cloud service providers use machine virtualization to provide flexible and cost-effective resource sharing. However, few studies have investigated the impact of machine virtualization in the cloud on networking performance.**

In this paper, we present a measurement study to characterize the impact of virtualization on the networking performance of the Amazon Elastic Cloud Computing (EC2) data center. We measure the processor sharing, packet delay, TCP/UDP throughput and packet loss among Amazon EC2 virtual machines. Our results show that even though the data center network is lightly utilized, virtualization can still cause significant throughput instability and abnormal delay variations. We discuss the implications of our findings on several classes of applications.

***Index Terms*—Measurement, cloud service, virtualization, networking performance**

I. Introduction

Cloud service allows enterprise class and individual users to acquire computing resources from large scale data centers of service providers. Users can rent machine instances with different capabilities as needed and pay at a certain per machine hour billing rate. Despite concerns about security and privacy, cloud service attracts much attention from both users and service providers. Recently, many companies, such as Amazon, Google and Microsoft, have launched their cloud service businesses.

Most cloud service providers use machine virtualization techniques to provide flexible and cost-effective resource sharing among users. For example, both Amazon EC2 [1] and GoGrid [11] use Xen virtualization [3] to support multiple virtual machine instances on a single physical server. Virtual machine instances normally share physical processors and I/O interfaces with other instances. It is expected that virtualization can impact the computation and communication performance of cloud services. However, very few studies have been performed to understand the characteristics of these large scale virtualized environments.

In this paper, we present an empirical measurement study on the end-to-end networking performance of the commercial Amazon EC2 cloud service, which represents a typical large scale data center with machine virtualization. The focus of our study is to characterize the networking performance of virtual machine instances and understand the impact of virtualization on the network performance experienced by users.

Observations: We measure the processor sharing, packet delay, TCP/UDP throughput and packet loss properties among Amazon EC2 virtual machine instances. Our study systematically quantifies the impacts of virtualization and finds that the magnitude of the observed impacts are significant:

1) We find that Amazon EC2 small instance virtual machines typically receive only a 40% to 50% share of the processor.
2) Processor sharing can cause very unstable TCP/UDP throughput among Amazon EC2 small instances. Even at the tens of millisecond time granularity, the TCP/UDP throughput experienced by applications can fluctuate rapidly between 1 Gb/s and zero.
3) Even though the data center network is not heavily congested, we observe abnormally large packet delay variations among Amazon EC2 instances. The delay variations can be a hundred times larger than the propagation delay between two end hosts. We conjecture that the large delay variations are caused by long queuing delays at the driver domains of the virtualized machines.
4) We find that the abnormally unstable network performance can dramatically skew the results of certain network performance measurement techniques.

Implications: Our study serves as a first step towards understanding the end-to-end network performance characteristics of virtualized data centers. The quantitative measurement results from this study provide insights that are valuable to users running a variety of applications in the cloud. Many cloud applications (e.g. video processing, scientific computing, distributed data analysis) are data intensive. The networking performance among virtual machines is thus critical to these applications' performance. The unstable throughput and packet delays can obviously degrade the performance of many data intensive applications. More importantly, they make it hard to infer the network congestion and bandwidth properties from end-to-end probes. Packet loss estimation is an example that will be discussed in section VI. The abnormal variations in network performance measurements could also be detrimental to adaptive applications and protocols (e.g. TCP vegas [5], PCP [2]) that conduct network performance measurements for self-tuning. Researchers have also recently started to deploy large scale emulated network experiments on cloud services [6], [12]. For this use of cloud services, our results point to challenges in performing accurate network link emulation in virtualized data centers. The unstable network performance

of cloud services may bias the conclusions drawn from these experiments. Given the observations from our measurement study, many applications may need to be adjusted to achieve optimal performance in virtualized data center environments.

The rest of this paper is organized as follows. In Section II, we introduce the background on Amazon EC2 and the Xen virtualization technique. In section III, we explain our measurement methodology. In Section IV, V, VI, we describe our measurement results and discuss the reasons behind our observations. In Section VII, we discuss the implications of our findings on different classes of cloud applications. We discuss the related work in Section VIII and conclude the paper in Section IX.

II. Background

A. Xen Virtualization

Xen [3] is an open source x86 virtual machine monitor which can create multiple virtual machines on a physical machine. Each virtual machine runs an instance of an operating system. A scheduler is running in the Xen hypervisor to schedule virtual machines on the processors. The original Xen implementation schedules virtual machines according to the Borrowed Virtual Time (BVT) algorithm [8].

Particularly for network virtualization, Xen only allows a special privileged virtual machine called *driver domain, or domain 0* to directly control the network devices. All the other virtual machines (called *guest domains* in Xen) have to communicate through the driver domain to access the physical network devices. The way Xen realizes this is, the driver domain has a set of drivers to control the physical Network Interface Cards (NIC), and a set of *back-end interfaces* to communicate with *guest domains*. The back-end interfaces and physical drivers are connected by a software bridge inside the kernel of the driver domain. Each *guest domain* has a customized virtual interface driver to communicate with a back-end interface in the driver domain. All the packets sent from *guest domains* will be sent to the driver domain through the virtual interfaces and then sent into the network. All the packets destined to a *guest domain* will be received by the driver domain first, and then transferred to the *guest domain*.

B. Amazon Elastic Cloud Computing (EC2)

Amazon EC2 is a component of Amazon's Web Services (AWS), which allows users to rent computers to run computer applications in the Amazon EC2 data center. Amazon EC2 uses the Xen virtualization technique to manage physical servers. There might be several Xen virtual machines running on one physical server. Each Xen virtual machine is called an *instance* in Amazon EC2. There are several types of instances. Each type of instance provides a predictable amount of computing capacity. The small instance is the primary instance type, which is configured with 1.7GB memory, 1 EC2 compute unit and 160GB instance storage. According to Amazon, "one EC2 compute unit provides the equivalent CPU capacity of a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor." For applications requiring higher computing capacity, Amazon EC2 provides several high-capacity instances which are configured with 4 to 20 EC2 compute units. The input-output (I/O) capacities of these types of instances are not specified clearly.

Allocated EC2 instances can be placed at different physical locations. Amazon organizes the infrastructure into different regions and availability zones. There are two regions, us-east-1 and eu-west-1, which are located in the US and in Europe respectively. Each region is completely independent and contains several availability zones that are used to improve the fault tolerance within the region. We suspect that each availability zone is an isolated data center which is powered by its own powerline. Different availability zones in the same region are placed very close to each other. The region us-east-1 has three availability zones, us-east-1a, us-east-1b and us-east-1c. The region eu-west-1 has two availability zones, eu-west-1a and eu-west-1b.

III. Experiment Methodology

In this section, we introduce the methodology of our measurement study. We first explain the properties we measure in our experiments, and the methodology we use to measure them.

A. Properties and Measurement Tools

Processor Sharing: Since each Amazon EC2 instance is a Xen virtual machine, an immediate question users may ask is "how does Amazon EC2 assign physical processor to my instance? Are there any processor sharing?" To answer this question, we use a simple CPUTest program to test the processor sharing property of EC2 instances. This program consists of a loop that runs for 1 million times. In each iteration, the program simply gets the current time by calling gettimeofday() and saves the timestamp into a pre-allocated array in memory. When the loop finishes, the program dumps all the saved timestamps to the disk. Normally, if the program is executed continuously, all loop iterations should take a similar amount of time. However, virtual machine scheduling can cause some iterations to take much longer than the others. If the instance is scheduled off from the physical processor, we should observe a gap in the timestamp trace. Since context switching among user processes can also cause a gap in the timestamp trace, we always run the CPUTest program as the only user process in our processor sharing experiments to minimize the impact of context switching. Therefore, from the timestamp trace of this program, we can estimate the processor sharing property of EC2 instances.

Packet round-trip delay: Given an instance pair, we use ping to measure the packet round-trip delay (or round-trip time, RTT) between them. To also measure delay variations, we send 10 ping probes per second, and continuously collect 5000 round-trip delay measurements.

TCP/UDP throughput: We developed two programs TCPTest and UDPTest to measure the TCP and UDP throughput that can be achieved by applications running on Amazon

EC2 instances. The UDPTest tool has a sender and receiver. The sender reads data from a buffer in memory and sends it as UDP packets. Since Amazon EC2 instances are Xen virtual machines, the UDP packets are sent to network through Xen driver domain. The communication between Xen driver domain and guest domain is done by copying data from memory to memory. If UDP sender sends as fast as possible, it will burst data at very high rate to the driver domain. A lot of traffic will be dropped when Xen driver domain cannot send them out in time. Therefore, in our UDPTest tool, the sender controls the sending rate to 1Gb/s by adding small idle intervals between every 128KB of data. We set the sending rate to 1Gb/s because according to our experiences, the Amazon EC2 instances are configured with Gigabit network cards. The UDP/IP packet size is 1500 bytes (i.e. the MTU of Ethernet) and the socket buffer size is 128KB. The receiver simply receives the UDP data and calculates the UDP throughput. The TCPTest tool also has a sender and a receiver. The sender reads data from a buffer in memory and sends data via a TCP connection to the receiver. The receiver also simply receives the data and calculates the TCP throughput. The TCP maximum send and receive window sizes are set to 256KB. From our experience, most of the RTTs among Amazon EC2 instances are below 0.5 ms. Therefore, if the network allows, end host could achieve throughput higher than 4Gbps by this window size.

Packet loss: We use the Badabing tool [13] to estimate the packet loss among Amazon EC2 instances. Badabing is the state-of-the-art loss rate estimation tool. It has been shown to be more accurate than previous packet loss measurement tools [13]. Packet loss estimation is considered challenging because packet loss typically occurs rarely and lasts for very short time. Badabing use active probes and statistical estimations to measure the packet loss properties. However, since we are using these tools in a virtualized environment, those estimations may not give us accurate results. We will provide detailed discussion on the packet loss estimation results in section VI.

B. Instance Type Selection

Amazon EC2 provides different types of instances for users. Our measurement experiments are mainly based on Amazon EC2 small instances and high CPU medium instances (or called medium instances). Small instances are the default instances in Amazon EC2 and they compete for physical processor resources, which creates an interesting environment for studying the impact of virtualization on network performance. High CPU medium instance is one type of high-capacity instances in Amazon EC2. Based on Amazon EC2 documents, the high-capacity instances are configured with multiple virtual cores (2 for high CPU medium instances). Each virtual core represents a CPU core that is visible inside a virtual machine. It is expected to have no processor competing among high-capacity instances. We choose medium instances as comparison with small instances to study the cases with and without processor sharing among virtual machines.

C. Large Scale Experiment Setup

We deploy large scale experiments to evaluate the system wide networking performance of Amazon EC2 instances. We set up a spatial experiment to evaluate how the network performance varies on instances at different network locations. We set up a temporal experiment to evaluate how the network performance varies on a given instance over a long time period. All the large scale experiments are deployed in the us-east-1 region. To eliminate the potential impacts from different kernel versions, we use the same OS image ami-5647a33f on all the instances.

Spatial experiment: In the spatial experiment, we request 250 pairs of small instance and 50 pairs of medium instances from each of the three availability zones us-east-1a, us-east-1b and us-east-1c. Within each availability zone, the instances are requested and measured in a round by round manner. In each round, we request a pair of instances, measure them and release them. Since we don't know the instance allocation strategy of Amazon EC2, we check the network mask of all the instances to validate that the requested instances are at different network locations. According to the network mask, the instances we have chosen cover 177 different subnets in Amazon EC2 data centers. For each instance pair, we measure the processor sharing using CPUTest on both instances. We also measure the network properties between the two instances, including delay, TCP/UDP throughput and packet loss. To avoid interference between different measurement programs, we run the programs one by one sequentially. Since the TCP/UDP throughput measurement programs are more intrusive to the network, we limit the amount of data transmitted in each test to 800 MB, which corresponds to roughly 10 seconds of measurement time. We run the Badabing tool for one minute to estimate the packet loss property for an instance pair. Since all the instance pairs in the same availability zone are measured sequentially, the measurement traffic of different instance pairs will not interfere with each other.

Temporal experiment: In the temporal experiment, we choose two small instance pairs and one medium instance pair in each of the three availability zones (us-east-1a, us-east-1b and us-east-1c). For all the nine instance pairs, we measure their processor sharing and network performance continuously for 150 hours. The measurements are done in a round by round fashion. Within each availability zone in each round, we measure the processor sharing, RTT, TCP/UDP throughput and packet loss of the three instance pairs one by one. The settings of all the measurement tools are the same as in the spatial experiment. The time interval between two adjacent rounds is set to 10 minutes. We arrange all the experiments inside the same availability zone sequentially to avoid interference between measurement traffic.

IV. Processor Sharing

We use our CPUTest program to test the processor sharing on small and medium instances in our spatial and temporal experiments. We first present a typical CPUTest timestamp trace observed on small and medium instance in Figure 1.

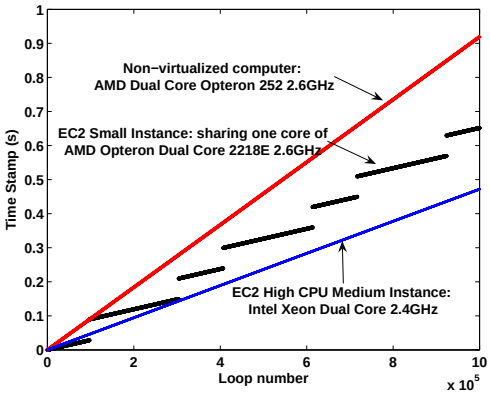

Fig. 1. CPUTest trace plot

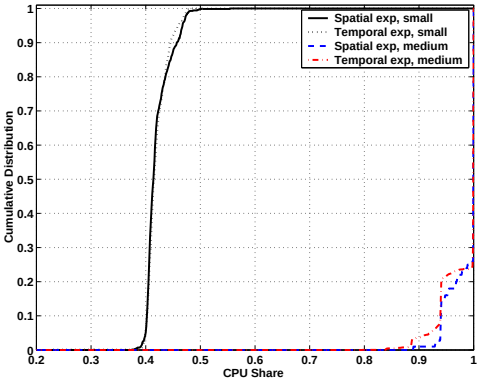

Fig. 2. The distribution of CPU share

As illustrated in Figure 1, when the CPUTest program is run on a non-virtualized machine or a medium instance, the timestamp traces produced indicate the CPUTest program achieves a steady execution rate with no significant interruption. However, the timestamp trace of the small instance shows very obvious scheduling effects. When the CPUTest program is run on a EC2 small instance, periodically there is a big timestamp gap between two adjacent loop iterations. The timestamp gaps are on the order of tens of milliseconds. In each iteration of the CPUTest program, the program only retrieves the current time and saves it to memory; there is no I/O operations. Since CPUTest is the only user program running on the instance, there shouldn't be frequent context switch between user programs. Therefore, the logical reason that explains the observed execution gap is virtual machine scheduling. The large timestamp gaps represent the periods when the instance running the CPUTest program is scheduled off from the physical processor.

In the CPUTest timestamp trace on an EC2 small instance, when the CPUTest program is running on the processor, one loop iteration normally takes less than 3 us. If one loop iteration takes more than 1 ms, we treat this time gap as a schedule off period. We define CPU sharing as the percentage of CPU time an instance gets from the Xen hypervisor. By searching through the timestamp traces produced by the CPUTest program, we can estimate the CPU sharing of EC2 instances. Figure 2 shows the distribution of CPU sharing

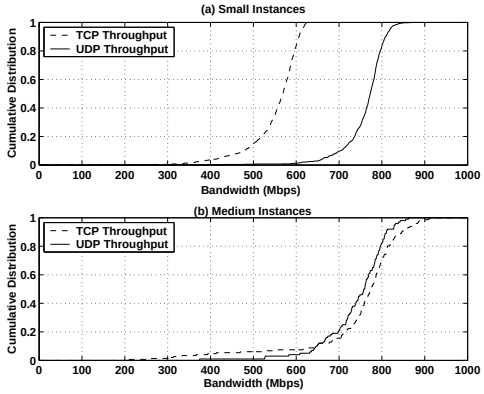

Fig. 3. The Distribution of bandwidth measurement results in spatial experiment

estimation of small and medium instances in both spatial and temporal experiments. From this graph, we can see that small instances are always sharing processors with other instances. For almost all the cases, small instances always get 40% to 50% of the physical CPU sharing. We suspect Amazon EC2 uses strict virtual machine scheduling policy to control the computation capacity of instances. Even there is no other virtual machines running on the same server, small instances still cannot use more than 50% of the processor. On the other hand, medium instances get 100% CPU sharing for most of the cases. There are only 20% of the cases where medium instances get 95% of the CPU sharing, which might be caused by the context switch between the CPUTest program and kernel service processes.

Note that the scheduling effect observed by our CPUTest program is only typical for CPU intensive applications since it does not have any I/O operations during the test period. I/O intensive applications may have different scheduling pattern. However, our results do confirm that processor sharing is a wide spread phenomenon among EC2 small instances, whereas medium instances do not competing processors with other instances.

V. Bandwidth and Delay Measurement

In this section, we discuss the bandwidth and delay measurement results of Amazon EC2 instances observed in our experiments.

A. Bandwidth Measurement

Measurement Results: In the spatial experiment, we measured the TCP/UDP throughput of 750 pairs of small instances and 150 pairs of medium instances at different network locations. In the temporal experiment, we measured the TCP/UDP throughput of 6 pairs of small instances and 3 pairs of medium instances continuously over 150 hours. Figure 3 shows the cumulative distribution of TCP/UDP throughput among small and medium instances in the spatial experiment. From these results, we can see that Amazon EC2 data center network is not heavily loaded since EC2 instances can achieve more than 500 Mbps TCP throughput for most the cases. More importantly, we can make an interesting observation from this

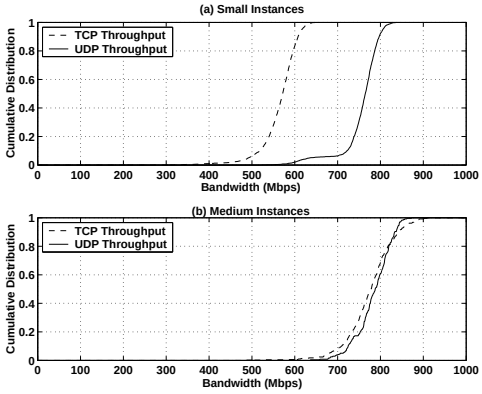

Fig. 4. The distribution of bandwidth measurement results in temporal experiment

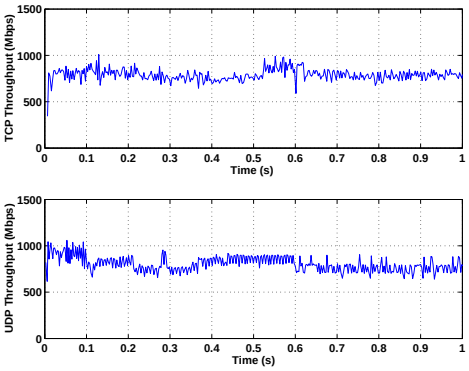

Fig. 6. TCP and UDP throughput of medium instances at fine granularity

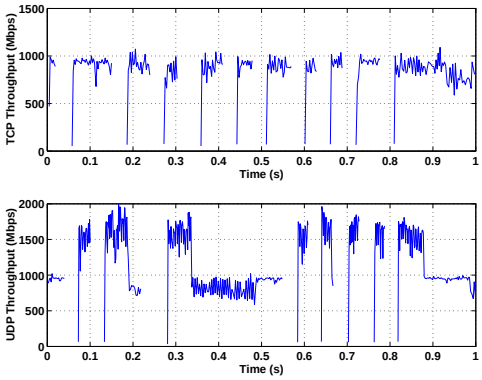

Fig. 5. TCP and UDP throughput of small instances at fine granularity

graph. Medium instances can achieve similar TCP and UDP throughput. The median TCP/UDP throughput of medium instances are both close to 760 Mbps. However, the TCP throughput of small instances are much lower than their UDP throughput. The median UDP throughput of small instances is 770 Mbps, but the median TCP throughput is only 570 Mbps. Figure 4 shows the cumulative distribution of TCP and UDP throughput of small and medium instances over the 150-hour period in our temporal experiment. We observe the same behavior from the results of the temporal experiment. Why are the TCP throughput of small instances much lower than their UDP throughput? We perform more detailed discussions and experiments to answer this question.

Discussion: Several factors can impact the TCP throughput results, including TCP parameter settings, packet loss caused by network congestion, rate shaping and machine virtualization. In our experiments, the TCP window size we use is 256KB which can achieve 4Gb/s throughput if network allows. Therefore, the low TCP throughput of small instances is not caused by TCP parameter settings. To investigate further, we study the TCP/UDP transmission at a much smaller time scale. In our TCPTest and UDPTest tool, every time when the receiver receives 256KB of data, it computes a throughput for the recent 256KB data transmission. Figure 5 demonstrates the fine-grain TCP and UDP throughput of a typical small instance pair in 1-second transmission. We consistently observe the same transmission pattern on all the small instances. To make the results clearly visible, we only pick one small instance pair and plot the throughput in 1 second period.

First, we discuss the behavior of TCP transmission. We observe the drastically unstable TCP throughput switching between full link rate at near 1 Gb/s and close to 0 Gb/s. From these transmission patterns, the relatively low average TCP throughput does not appear to be caused by any explicit rate shaping in Xen because typical rate shapers (e.g. a token bucket rate shaper) would not create such oscillations.

By looking at the TCPDUMP trace of the TCP transmission, we find that during the very low throughput period, no packet is sent out from the TCP sender. The quiet periods last for tens of milliseconds. The minimum TCP retransmission timeout is normally set to 200 ms in today's Linux kernel [14]. These quiet periods are not long enough to cause TCP retransmissions. We also confirm that there are no TCP retransmission observed in the TCPDUMP trace. This result tells us that the periodic low TCP throughput is not caused by packet loss and network congestion because if that is the case, we should observe a large number of TCP retransmissions. Considering the processor sharing behavior observed in our CPUTest experiments, we believe that the quiet periods are caused by the processor sharing among small instances. During these quiet periods, either the TCP sender instance or the receiver instance are scheduled off from the physical processor, therefore no packet can be sent out from the sender.

From Figure 5, we can observe a similar unstable UDP throughput on small instances. The difference between UDP and TCP transfers is that, in many cases, after a low throughput period, there is a period where the receiver receives UDP traffic at a high burst rate (even higher than the network's full link rate). That is why UDP throughput is higher than TCP throughput on average. We believe the reason is, during the low UDP throughput periods, the receiver is scheduled off from the processor, but the sender instance is scheduled on. All the UDP traffic sent to the receiver will be buffered in the Xen driver domain. When the receiver is scheduled in later, all the buffered data will be copied from driver domain memory to the receiver's memory. Since the data is copied from memory to memory, the receiver can get them at a much higher rate than the full link rate.

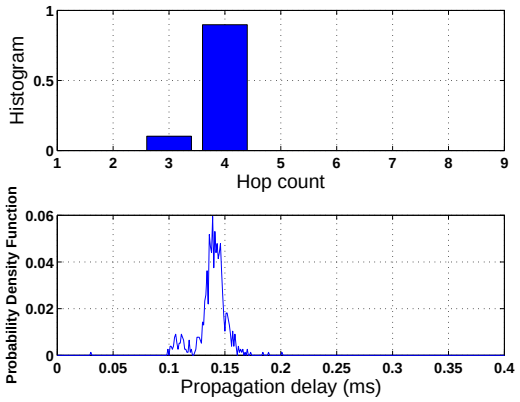

Fig. 7. The Distribution of propagation delays and hop count results in spatial experiment

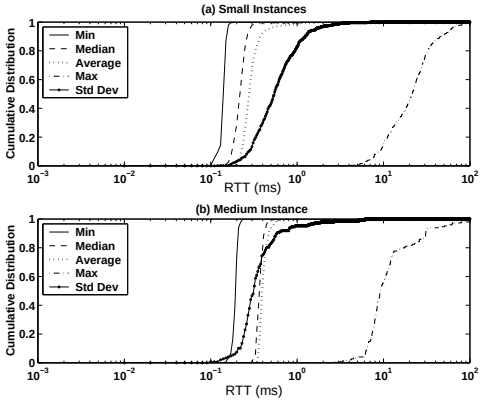

Fig. 8. The distribution of delay statistical metrics in spatial experiment

We define a *buffer burst period* by a UDP transmission period during which the receiver continuously receive data at rates higher than the full link rate. Since we control the UDP sending rate to 1Gbps, during a *buffer burst period*, the additional amount of data beyond full link rate transfer must come from the Xen driver domain buffer. We call this additional data *buffer burst data*. We can estimate the lower bound of Xen driver domain buffer size by the volume of *buffer burst data*. We analyze the UDP transmission trace of small instance pairs in our 150 hour temporal experiment. We find, in the maximum case, the *buffer burst data* is as high as 7.7 MB. It means that Xen driver domain buffer can be more than 7.7 MB. The large buffer at Xen driver domain can help reduce the packet loss and improve the average UDP throughput when instances are scheduled off from physical processors.

Figure 6 demonstrates the fine-grain TCP/UDP throughput trace for a medium instance pair. Since there is no processor sharing among medium instance pairs, the TCP/UDP throughput is relatively stable. Medium instances can achieve similar UDP and TCP throughput which are decided by the traffic load of the data center network.

B. End-to-end Delays

Measurement Results: In this section, we discuss the packet delay measurement in our experiments. In our spatial

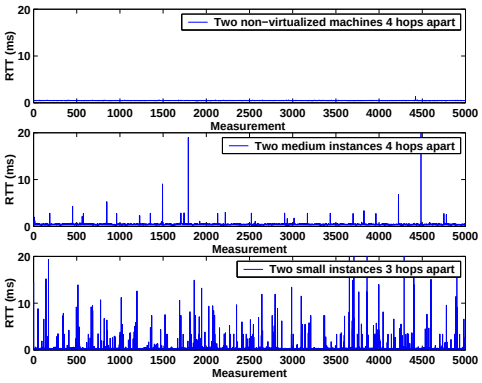

Fig. 9. Raw RTT measurements on Amazon EC2 instances and non-virtualized machines in university network

experiment, we measure the packet round trip delay (RTT) of 750 small instance pairs and 150 medium instance pairs using 5000 ping probes. Before describing the characteristics of end-to-end delays, we first discuss an interesting observation in our ping measurement results. We consistently observe very large delays (hundreds of ms) for the first several ping probe packets over all the instance pairs in our spatial experiment. We compare the ping RTT results with the RTTs measured by UDP probe packets. We find that the UDP probe RTTs have the same characteristics with the ping RTTs except that the first several UDP probe packets do not have abnormally large delays. By looking at the TCPDUMP trace of the ping packets, we believe the reason for the abnormally large initial ping RTTs is that every time ping packets are initiated between an instance pair, the first several ping packets are redirected to a device, perhaps for a security check. The routers can forward ping packets only after the security check device allows them to do so. The large delays of the first several ping packets are caused by the buffer delay at the security device. Therefore, in our RTT characteristics analysis, we remove the RTT measurement results of the first 50 ping packets to eliminate the impact of this security check on our delay measurements.

We analyze several characteristics of RTTs among EC2 instances. First, we estimate the propagation delays between instance pairs using the minimum RTTs observed in ping probes. In Figure 7, the bottom graph shows the probability distribution of propagation delays for all instance pairs. The propagation delays have a two-peak distribution. The top graph in Figure 7 shows the histogram of the hop counts for all the instance pairs. The hop counts are measured using traceroute. From this graph, we can see that in the EC2 data center, instances are very close to each other. All the instance pairs we measured are within 4 hops from each other, and most propagation delays are smaller than 0.2 ms. For all the 900 instance pairs we have measured, the instances are either 3 hops or 4 hops away from each other. This is the reason why we observe a two-peak propagation delay distribution.

For each instance pair, we compute the minimum, median, average, maximum RTTs and the RTT standard deviation from the 4950 probes. Figure 8 shows the cumulative distribution of

these RTT statistical metrics for small and medium instances (note that the x-axis is in log scale). From this graph, we can see that the delays among these instances are not stable. The propagation delays are smaller than 0.2 ms for most of the small instance pairs. However, on 55% of the small instance pairs, the maximum RTTs are higher than 20 ms. The standard deviation of RTTs is an order of magnitude larger than the propagation delay and the maximum RTTs are 100 times larger than the propagation delays. The delays of medium instances are much more stable than the small instances. But we still observe that, for 20% medium instance pairs, the maximum RTTs are larger than 10ms. Considering the Amazon EC2 data center is a large cluster of computers that are not spread over a wide area, these large delay variations are abnormal.

As a comparison, we test the RTT and between non-virtualized machines located in our university network and in Emulab. We observe much smaller delay variations on the machines in our university network and in Emulab. For example, for two machines in our university network which are 4 hops away, the minimum/average/maximum RTTs are 0.386/0.460/1.68 ms respectively, and the RTT standard deviation is 0.037 ms. For two machines in Emulab which are connected through a switch, the minimum/average/maximum RTTs are 0.138/0.145/0.378 ms, and the RTT standard deviation is 0.014 ms. For all these non-virtualized machines, the RTT standard deviation is roughly 10 times smaller than the propagation delays. To visualize this difference, we plot the 5000 RTT measurement results for non-virtualized machines, small instances, and medium instances in Figure 9. We can clearly see that, RTTs among Amazon EC2 instances have much higher variations than non-virtualized machines. The delay variations among small instances are much higher than that of medium instances.

Discussion: End-to-end delay variation are typically assumed to be caused by the queuing delays on routers when a network is congested. However, in the Amazon EC2 data center, the large delay variations are unlikely to be caused by network congestion. The reasons can be argued as follows. First, we observe very rare packet loss in the ping probes. Among all the instance pairs we have done ping probes, 98% of them did not experience any ping packet loss. The other 2% only experience roughly 1 out of 1000 packet loss. Second, in our bandwidth measurement experiments, all the instances we have measured can achieve at least 500Mb/s TCP throughput. All these results imply that the Amazon EC2 data center network is not congested.

Considering the processor sharing and large Xen driver domain buffer observed in previous sections, our conjecture is that the large delay variations among EC2 instances are caused by the long queuing delay at the Xen driver domain. Since small instances are sharing processors with other instances, when the receiver instance is scheduled off, the probe packets will be buffered at the Xen driver domain until the receiver is scheduled on. This long buffering delay causes very high delay variations among small instances. Although medium instances do not share processors, they are still sharing Xen driver domain with other instances. Let us suppose a medium instance A is sharing Xen driver domain with another instance B. Since there is a large buffer in the driver domain, instance B could burst a big trunk of data into the driver domain buffer. In this case, the packet from A could be put into a long queue in Xen driver domain, which leads to relatively long queuing delay. Since packets don't need to wait for the processor scheduling for medium instances, the delay variations on medium instances are generally smaller than small instances.

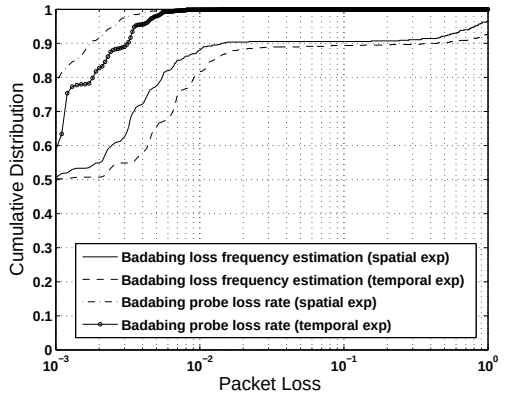

Fig. 10. Badabing packet loss results in spatial and temporal experiments

VI. Packet Loss Estimation

Estimation Results: In this section, we describe the packet loss estimation results observed in our experiments. Badabing estimates the packet loss characteristics of an end-to-end path by estimating if each 5ms time slot is a lost episode. Figure 10 shows the overall cumulative distribution of packet loss frequency estimated by Badabing in our spatial and temporal experiments (note the x-axis is in log scale). Here the packet loss frequency is defined as $(loss\_time\_slot/total\_time\_slot)$. From this graph, we can see that Badabing reports abnormally high packet loss frequency in the Amazon EC2 data center. In both spatial and temporal experiment results, more than 10% of the Badabing measurements report very high packet loss frequency ($> 10\%$). This packet loss frequency is extremely high since normally packet loss happens very rarely ($< 0.1\%$). To cross validate, we look at the probing packet traces of all the Badabing measurements, the cumulative distributions of Badabing probe loss rate are also plotted in Figure 10. The Badabing probe loss rate is defined as $(lost\_probes/total\_probes)$ in Badabing measurements. From the distribution of Badabing probe loss rate, we can see that probe packet loss actually happens very rarely in both spatial and temporal experiments. For 98% of the measurements, the probe loss rates are smaller than 0.005 and for 60% of the measurments, the probe loss rates are smaller than 0.001. The very high packet loss frequency reported by Badabing is suspicious. We perform a more detailed discussion on the Badabing estimation results.

Discussion: Badabing estimates the loss characteristics of end-to-end paths by detecting the *loss episodes*. *loss episode*

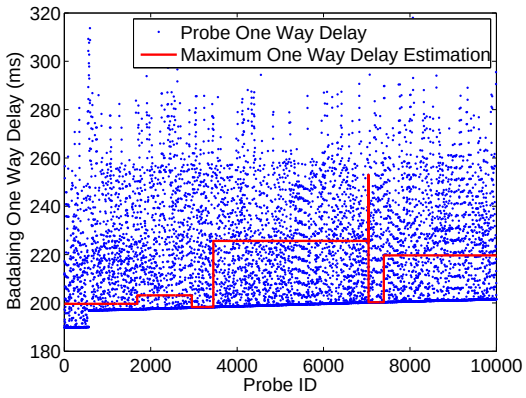

Fig. 11. Badabing probe packet one way delay and maximum OWD estimation

is defined as the time series indicating a series of consecutive packets (possibly only of length one) were lost [16]. There is a sender and a receiver in the Badabing tool. At each 5ms time slot, the sender sends a probe with 30% probability. Each probe includes 3 probing packets and all the probe packets are timestamped. When the receiver receives a probe packet, it simply remembers the packet sequence number and the timestamps. Badabing assumes time synchronization between the sender and receiver. However, since Badabing estimates loss episode based on delay differences, the time synchronization does not have to be perfect. The estimation algorithm marks a time slot as lossy or not lossy based on the one way delay and lost packets in the time slot. The criteria is that if there is a lost packet within τ time of the current time slot, or the one way delay is larger than $max\_owd \times (1-\alpha)$, the time slot is marked as lossy. Here, the max_owd is the maximum one way delay of the path, which is estimated by the one way delay of the most recent successful packet when a packet loss happens. By default, τ is set to 50 ms and α is set to 0.005. Here, Badabing is making an implicit assumption that when an end-to-end path is in loss episodes, the one way delays (OWD) of this path will be higher than its one way delays when the path is not in loss episodes. This assumption makes sense in the wide area Internet environments.

However, in our previous results, we have observed very high delay variation even though the data center network is not congested. These large delay variations are very likely to be caused by the machine virtualization. The problem is that, the delay variations caused by virtualization can be much larger than the delay variations caused by network congestion. Many of these large delay variations can cause Badabing to mark a time slot as lossy. Therefore, in this environment, Badabing will have a much higher false positive rate. That is why Badabing reports very high packet loss frequency on many instance pairs in our experiments. To demonstrate this effect, we plot the one way delay and corresponding maximum OWD estimation for 10,000 probes on a small instance pair in Figure 11. During the 10,000 Badabing probes, there are only 7 packets lost. Every time when a packet loss happens, Badabing will estimate a new maximum OWD based on one way delay of most recent successful packets. From the graph, we can see that the estimated maximum OWDs are not very large. However, in many cases, the one way delay variations caused by virtualization can be much larger than the estimated maximum OWDs. All these cases will cause false positive detections in the Badabing results.

The discussion of Badabing results reveals that, in the virtualized data center environment, there are additional difficulties to infer network properties using statistics. Some valid assumption in traditional network environments may not hold in virtualized data centers.

VII. Implications

We have found that the networking performance between Amazon EC2 instances demonstrate very different characteristics from traditional non-virtualized clusters, such as the abnormal large delay variations and unstable TCP/UDP throughput caused by end host virtualization. In this section, we discuss the implications of our findings on applications running in cloud services.

Network measurement based systems in cloud: As discussed in the previous section, the large delay variation can completely skew the packet loss estimation results of the Badabing tool. Badabing is just one example of the problem. The fundamental problem is that the simple textbook end-to-end delay model composed of network transmission delay, propagation delay, and router queuing delay is no longer sufficient. Our results show that in the virtualized data center, the delay caused by end host virtualization can be much larger than the other delay factors and cannot be overlooked. Other than the Badabing tool we discussed in the paper, the large delay variations can also impact many other protocols and systems that rely on the RTT measurement to infer network congestion, such as TCP vegas [5] and PCP [2]. Therefore, if the cloud service users want to build systems relying on the network measurements to make decisions, they need to be aware of the virtual machine scheduling characteristics of the virtualized data center environment.

Network experiments in cloud: Emulab is a widely used facility for networking researchers to deploy emulated network experiments. However, Emulab is based on a relatively small computer cluster. In many cases, researchers cannot find enough machines to deploy their large scale experiments. Recently, researchers have proposed to deploy large scale network experiments on the Amazon EC2 service (e.g. the Cloudlab project [6]). However, as far as we know, there is no quantitative result about the feasibility of this idea. Our measurement results provide some insights on this problem. To deploy network experiments on Amazon EC2, the challenge is to emulate different kinds of network links between EC2 instances. The processor sharing and unstable network performance bring challenges to the link emulation. First, because all the small EC2 instances are sharing processors with other instances, it is very hard for them to set timers precisely to perform accurate rate shaping. In addition, the large delay variations and unstable throughput make it hard to emulate

stable high speed links among small instances. Using high capacity instances might be able to reduce the problem, but further experimental study is needed to understand this issue.

Scientific computing in cloud: Unstable network throughput and large delay variations can also have negative impact on the performance of scientific computing applications. For example, in many MPI applications, a worker has to exchange intermediate results with all the other workers before it can proceed to the next task. If the network connections to a few workers suffer from low throughput and high delay variations, the worker has to wait for the results from the delayed workers before it can proceed. Therefore, MPI applications will experience significant performance degradation. MapReduce applications [7] may experience the same problem when a large amount of data is shuffled among all the mappers and reducers. To improve the performance of these scientific computing applications on cloud service, we may need to customize their job assignment strategies to accommodate the unstable networking performance among virtual machines.

VIII. Related Work

A few studies have evaluated the performance of cloud services. In [10], Garfinkel has reported his experience of using Amazon EC2, S3 (simple storage service) services. The focus is mainly on the performance of the Amazon S3 service. This paper measures the throughput and latency of the S3 service from Amazon EC2 and other Internet locations. In contrast, the focus of our study is on the networking performance of Amazon EC2 instances and the impact of virtualization on the network performance. In [15], Walker evaluates the performance of Amazon EC2 for high-performance scientific computing applications. The author compares the performance of Amazon EC2 against another high performance computing cluster NCSA, and reports that Amazon EC2 has much worse performance than traditional scientific clusters. This paper is focused on the application level performance of Amazon EC2. The findings of our study can help to explain some of the observations in [15].

In [9], Ersoz *et al.* measure the network traffic characteristics in a cluster-based, multi-tier data center. This study is not based on a real commercial data center service. Instead, the authors measure the network traffic using an emulated data center prototype. A more recent study [4] has measured the network traffic workload of production data centers. The focus of this work is to investigate the data center traffic pattern and explore the opportunities for traffic engineering. No virtualization is considered in these studies. As far as we know, our work is the first study focusing on the networking performance of Amazon EC2 instances and on understanding the impact of virtualization on the data center network performance.

IX. Conclusion

In this paper, we present a quantitative study of the end-to-end networking performance among Amazon EC2 instances from users' perspective. We observe wide spread processor sharing, abnormal delay variations and drastically unstable TCP/UDP throughput among Amazon EC2 instances. Through detailed analysis, we conclude that these unstable network characteristics are caused by virtualization and processor sharing on server hosts. The unstable network performance can degrade the performance of and bring new challenges to many applications. Our study provides early results towards understanding the characteristics of virtualized data centers. As future work, we will study how applications can be customized to achieve good performance over virtualized environments.

Acknowledgments

This research was sponsored by the NSF under CAREER Award CNS-0448546, NeTS FIND Award CNS-0721990, by Microsoft Corp., and by an Alfred P. Sloan Research Fellowship. Views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of NSF, Microsoft Corp., the Alfred P. Sloan Foundation, or the U.S. government.

References

[1] "Amazon ec2," http://aws.amazon.com/ec2/.
[2] T. Anderson, A. Collins, A. Krishnamurthy, and J. Zahorjan, "Pcp: Efficient endpoint congestion control," in *Proceedings of USENIX NSDI'06*, May 2006.
[3] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *Proceedings of SOSP'03*, Oct. 2003.
[4] T. A. Benson, A. Anand, A. Akella, and M. Zhang, "Understanding data center traffic characteristics," in *Proceedings of SIGCOMM WREN Workshop*, Aug. 2009.
[5] L. S. Brakmo and L. Peterson, "Tcp vegas: End-to-end congestion avoidance on a global internet," *IEEE Journal of Selected Areas in Communication*, vol. 13, no. 8, Oct. 1995.
[6] "Cloudlab," http://www.cs.cornell.edu/ einar/cloudlab/index.html.
[7] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," in *Proceedings of USENIX OSDI'04*, Dec. 2004.
[8] K. J. Duda and D. R. Cheriton, "Borrowed-virtual-time(bvt) scheduling: supporting latency-sensitive threads in a general purpose scheduler," in *Proceedings of SOSP'99*, Dec. 1999.
[9] D. Ersoz, M. S. Yousif, and C. R. Das, "Characterizing network traffic in a cluster-based, multi-tier data center," in *Proceedings of ICDCS'07*, Jun. 2007.
[10] S. L. Garfinkel, "An evaluation of amazon's grid computing services: Ec2, s3 and sqs," Computer Science Group, Harvard University, Technical Report, 2007, tR-08-07.
[11] "Gogrid," http://www.gogrid.com/.
[12] C. Raiciu, F. Huici, M. Handley, and D. S. Rosen, "Roar: Increasing the flexibility and performance of distributed search," in *Proceedings of SIGCOMM'09*, Aug. 2009.
[13] J. Sommers, P. Barford, N. Duffield, and A. Ron, "Improving accuracy in end-to-end packet loss measurement," in *Proceedings of SIGCOMM'05*, Aug. 2005.
[14] V. Vasudevan, A. Phanishayee, H. Shah, E. Krevat, D. Andersen, G. R. Ganger, G. A. Gibson, and B. Mueller, "Safe and effective fine-grained tcp retransmissions for datacenter communication," in *Proceedings of SIGCOMM'09*, Aug. 2009.
[15] E. Walker, "Benchmarking amazon ec2 for high-performance scientific computing," in *USENIX ;login: magzine*, Oct. 2008.
[16] Y. Zhang, N. Duffield, V. Paxson, and S. Shenker, "On the constancy of internet path properties," in *Proceedings of IMW'01*, Nov. 2001.