Workload-Aware Provisioning in Public Clouds

Yunjing Xu[1], Zachary Musgrave[2], Brian Noble[3], Michael Bailey[3]

[1]Square, Inc., [2]Yelp, Inc., [3]Department of Computer Science and Engineering, University of Michigan

Abstract

Public cloud services rely on virtualization to support multi-tenancy—customers from different organizations are allowed to share the data center infrastructure. Unfortunately, today's public clouds fail to provide sufficient isolation. Hardware resources are often multiplexed between virtual machines that belong to different customers, and they can cause performance interference to each other. This article characterizes the interference on an important metric, the network latency between virtual machines, and shows that Amazon's EC2 cloud, a leading public cloud provider, suffers from a long tail latency problem. The root cause of this problem is co-scheduling of CPU-bound and latency-sensitive tasks. We leverage these observations in Bobtail, a system that allows cloud customers to proactively detect and avoid these bad neighboring virtual machines without any help from cloud service providers.

I. Introduction

With the emergence of cloud computing in the mid 2000s, computing resources became public utility; this concept dates back to the early 60s [2]. Among cloud computing paradigms, Infrastructure-as-a-Service (IaaS) employs a pay-as-you-go model that allows anyone with a valid credit card to rent a large amount of computing resources from cloud data centers and only pay for what they use, without an upfront investment in hardware infrastructure.

Public IaaS clouds, such as Amazon Elastic Cloud Compute (EC2) [3], often are used to build Internet-scale Web applications, such as Netflix, Yelp, and Pinterest [4]. The impact of public clouds on the consumer Internet is therefore enormous. For example, as of April 2012, sites built on Amazon's cloud alone attract one third of all Internet users every day and contribute to more than 1% of all Internet consumer traffic [5].

A distinguishing feature of public clouds is *multi-tenancy*—hardware infrastructure is shared by various users of different organizations. Thus, instead of allowing direct hardware access, cloud providers use *virtualization* to give users access to computing resources in the form of virtual machines (VMs), they retain full control of all underlying hardware infrastructure. Ideally, virtualization should give users the illusion of dedicated hardware access and should provide strong isolation between the virtual machines that share physical machines, the data center network, and other layers of the cloud infrastructure, so that they cannot interfere with one another.

This article is derived from the conference paper, Bobtail: Avoiding Long Tails in the Cloud [1], published in NSDI'13 by the same authors. This work was completed while Yunjing Xu and Zachary Musgrave were both graduate students at the University of Michigan.

Unfortunately, such isolation is routinely violated because of contention for the shared resources multiplexed between guest VMs in public clouds, and it may results in performance interference between one another [6]–[8]. For example, the performance of a workload with temporal locality in its memory access pattern relies heavily on the efficiency of various levels of CPU caches, but its neighboring VMs on the same physical machine may run workloads causing frequent cache eviction; this behavior forces repeated main memory access for identical, recently used content [9].

Mitigating the performance interference between guest virtual machines in public clouds is challenging because virtualization creates a *semantic gap* [10] between the guests who manage application workloads and the hosts who manage the cloud infrastructure. Optimizations at one layer are made without understanding the mechanisms or even intentions at another, and they tend to operate at cross purposes. For instance, from the cloud's guests' perspective, the extent of resource contention is determined by the resource schedulers that are host-controlled and operating underneath all guest VMs, while from the host's perspective, applications' resource usage patterns may affect their scheduling policies, but only guest VMs have the knowledge of such patterns.

Therefore, in order to achieve effective mitigation, the first step is to characterize the impact of performance interference and study its root causes. The potential impact of performance interference may exist for both the throughput and latency of network I/O, disk I/O and computational jobs. As network I/O latency becomes increasingly important for Internet-scale user-facing applications [11], [12], this article characterizes the impact and root cause of performance interference on inter-VM network latency.

Our characterization studies show that the performance interference is a consequence of resource contention between guest VMs. Therefore, this article also explores novel techniques to reduce conflicts of resource usage and mitigate VM interference. By definition, avoiding hardware sharing completely would eliminate all possible contention [6], but it also defeats the economic model of cloud computing. Instead, to mitigate the performance interference, while still preserving the benefits of resource multiplexing, a restricted form of sharing, Bobtail [1], is designed to seek processor sharing with only compatible workloads to therefore reduce the conflicts. This approach only requires knowledge about guest application workloads above the virtualization semantic gap—it allows guest VMs to reduce network latency without any infrastructure changes.

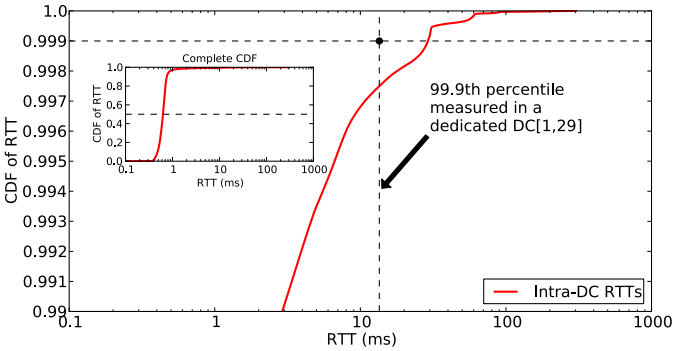


Fig. 1. CDF of RTTs for small, medium, and large instances within one of Amazon's EC2 data centers, compared to measurements taken in a dedicated data center [11], [12]. While the median RTTs are comparable, the 99.9th percentiles in EC2 are twice as bad when compared to dedicated data centers.

II. Observations

A. Tail Latency Characterization

To characterize the impact of performance interference on network latency, we tested network round-trip-times (RTTs) between virtual machines in several data centers of a leading public cloud provider, the US east region of Amazon's EC2 [3], for five weeks. Virtual machine in EC2 come with different configurations and price tags. Figure 1 shows the Cumulative Distribution Function (CDF) of RTTs for a combination of small, medium, and large virtual machine instances (the units of resources to rent). Specifically, we instantiated 20 instances of each type of virtual machines and measured the RTTs against them from dedicated testing nodes. The inset of the figure shows that median RTTs within a single data center, at ∼0.6ms, compare well to those found within a dedicated data center at ∼0.4ms [11], [12].

While mean or median metrics are useful for high-throughput applications like MapReduce [13], worst-case performance matters much more to applications like the Web that require excellent user experience [12]. Because of this, researchers use the RTTs at the 99th and 99.9th percentiles to measure flow tail completion times in dedicated data centers [11], [12]. Figure 1 also shows the 99th to 100th percentile range of its inset figure. Unfortunately, its results paint a different picture of latency measurements in Amazon's data centers. The 99.9th percentile of RTT measurements is *twice as bad* as the same metric in a dedicated data center [11], [12]. Individual nodes can have 99.9th percentile RTTs up to four times higher than those seen in dedicated data centers.

To explore potential factors that might create extra long tails, we launched 16 instances within the same data center and measure the *pairwise* RTTs between each instance. Figure 2 shows measurement results at the 99.9th percentile in milliseconds. Rows represent source IP addresses, while columns represent destination IP addresses.

Were host location on the network affecting long tail performance, we would see a symmetric pattern emerge on the heat map, since network RTT is a symmetric measurement. Surprisingly, the heat map is asymmetric—there are vertical bands which do not correspond to reciprocal pairings. To a large degree, the destination host controls whether a long tail

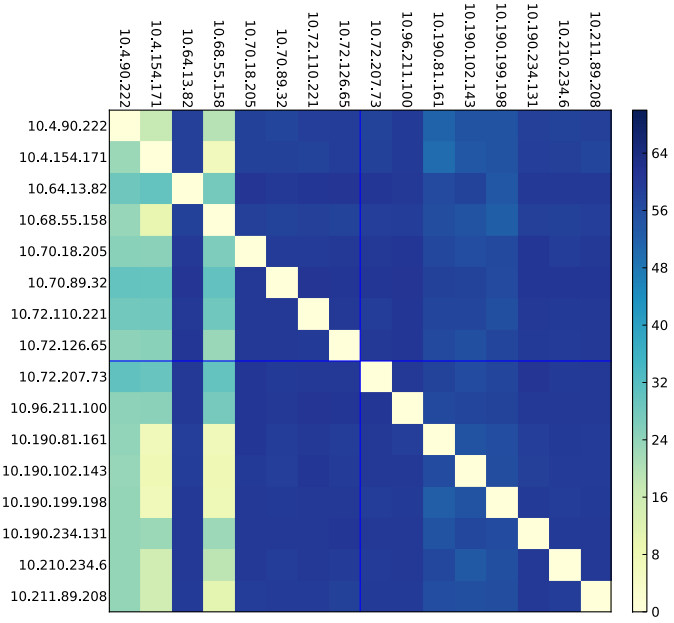


Fig. 2. A heat map of the 99.9th percentile of RTTs, shown for 16 small pairwise instances in milliseconds. Bad instances, represented by dark vertical bands, are bad consistently. This suggests that the long tail problem is a property of specific nodes instead of the network.

exists. In other words, *the extra long tail problem in cloud environments is a property of nodes, rather than the network.*

Interestingly, the data shown in Figure 2 is not entirely bleak: there are *both* dark and light bands, so tail performance between nodes varies drastically. Commonly, service nodes are allowed only 10ms to return their results [11]. Therefore, we refer to nodes that fulfill this service as *good* nodes, which appear in Figure 2 as light bands; otherwise, they are referred to as *bad* nodes. Under this definition, we find that RTTs at the 99.9th percentile can vary by *up to an order of magnitude* between good nodes and bad nodes. In particular, the bad nodes we measured can be two times worse than those seen in a dedicated data center [11], [12] for the 99.9th percentile. This is because the latter case's latency tail is caused by network congestion, whose worst case impact is bounded by the egress queue size of the bottleneck switch port, but the latency tail problem we study here is a property of nodes, and its worst case impact can be much larger than that caused by network queueing delay. This observation will become more clear when we discuss the root cause of the problem in § III-A.

To determine whether bad nodes are a pervasive problem in EC2, we spun up 300 small instances in each of four data centers in the US east region. We measured all the nodes' RTTs and we found between 40% and 70% bad nodes within three of the four data centers.

Interestingly, the remaining data center sometimes does not return bad nodes; nevertheless, when it does, it returns 40% to 50% bad nodes. We noticed that this data center spans a smaller address space of only three /16 subnets compared to the others, which can span tens of /16 subnets. Also, its available CPU models are, on average, newer than those found in any of the other data centers; Ou *et al.* present similar

findings [14], so we speculate that this data center is newly built and loaded more lightly than the others.

We also want to explore whether the long latency tail we observe is a persistent problem, because it is a property defined by node conditions rather than transient network conditions. We conducted a five week experiment comprised of two sets of 32 small instances: one set was launched in equal parts from two data centers, and one set was launched from all four data centers. Within each set, we selected random pairs of instances and measured their RTTs over the five weeks. For the sake of space, the details of our measurement results are omitted, but in short, we observed that the long tail latency is a relatively stable property for the instances used in the five-week measurement period.

III. Root Cause Analysis

We know that the latency tail in EC2 is two to four times worse than that of a dedicated data center. We also know that, as a property of nodes instead of the network, it persists. Then, what is its root cause? Wang *et al.* reported that network latency in EC2 is highly variable, and they speculated that virtualization and processor sharing make up the root cause [15].

However, the coexistence of good and bad instances suggests that processor sharing under virtualization is *not sufficient* to cause the long tail problem by itself. We will show in this section that only *a certain mix of workloads* on shared processors can cause this problem, and we demonstrate the patterns of such a bad mix.

A. Root Cause Explained

If processor sharing under virtualization does not always cause the extra long tail problem, when does it? To answer this question, we conduct five controlled experiments with the Xen hypervisor [16], which was reported to be used by EC2 for virtualization [15].

For this set of experiments, we varied the workload types running on five virtual machines (VMs) sharing a local workstation. In the first experiment, we ran a server that serves measurement requests in all five guest VMs; we used another non-virtualized workstation in the same local network to make measurement requests to all five servers, once every two milliseconds, for 15 minutes. During the experiment, the local network was never congested. In the next four experiments, we replaced the measurement servers on the guest VMs with a CPU-intensive workload, one at a time, until four guest VMs are CPU-intensive and the last one, called the *victim VM*, remained latency-sensitive.

Figure 3 shows the CDF of our five experiments' RTT distributions from the 99th to the 100th percentile for the victim VM. While four other VMs also run latency-sensitive jobs (zero VMs run CPU-intensive jobs), the latency tail up to the 99.9th percentile remains under 1ms. If one VM runs a CPU-intensive workload, this result does not change. Notably, even when the victim VM *does share* processors with one CPU-intensive VM and three latency-sensitive VMs, the extra long tail problem is *nonexistent*.

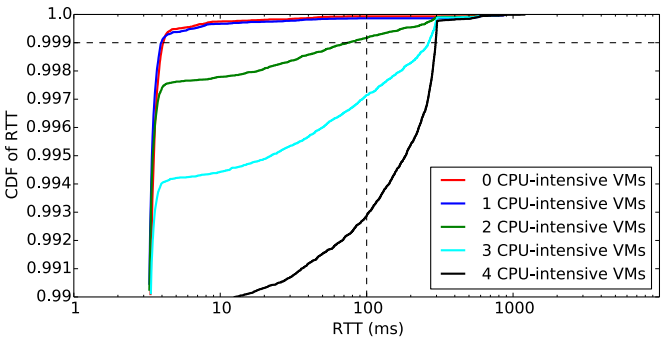


Fig. 3. CDF of RTTs for a VM within controlled experiments, with an increasing number of co-located VMs running CPU-intensive workloads. Sharing does not cause extra long latency tails as long as physical cores outnumber CPU-intensive VMs, but once this condition no longer holds, the long tail emerges.

However, the 99.9th percentile becomes *five times* larger once two VMs run CPU-intensive jobs. This still qualifies as a good node under our definition (<10ms), but the introduction of even slight network congestion could change that. To make matters worse, RTT distributions increase further as more VMs become CPU-intensive. Eventually, the latency-sensitive victim VM behaves just like the bad nodes we observe in EC2.

The results of our controlled experiments assert that virtualization and processor sharing are not sufficient to cause high latency effects across the entire tail of the RTT distribution; therefore, much of the blame rests upon co-located workloads. We show that having one CPU-intensive VM is acceptable; why does adding one more make things five times worse?

There are two physical cores available to guest VMs; if we have one CPU-intensive VM, the latency-sensitive VMs can be scheduled as soon as they need to be, while the single CPU-intensive VM occupies the other core. Once we reach two CPU-intensive VMs, it becomes possible that they occupy both physical cores concurrently while the victim VM has a measurement request pending. Unfortunately, Xen's VM scheduler does not appear to let the victim VM preempt the CPU-intensive VMs often enough. Resulting from these unfortunate scenarios is an extra long latency distribution. In other words, *sharing does not cause extra long latency tails as long as physical cores outnumber CPU-intensive VMs; once this condition no longer holds, the long tail emerges.*

This conclusion helps understand why one data center in EC2's east region has a higher probability of returning good instances than the others. If we break down VMs returned from this data center by CPU model, we find a higher likelihood of newer CPUs, which have six cores instead of four, these older CPUs are more common in the other three data centers. One potential explanation is that six-core machines allow more headroom to tolerate CPU-intensive VMs, i.e., they are less susceptible to the long tail latency problem.

IV. Avoiding the Long Tails

While sharing is inevitable in multi-tenant public clouds, we set out to design a system, Bobtail, to find instances where processor sharing does not cause extra long tail distributions for network RTTs. Cloud customers can use Bobtail as a utility

to decide on which instances to run their latency-sensitive workloads without any help from their cloud providers.

A naive approach might be to conduct network measurements with every candidate. But however accurate it might be, such a design would not scale well to handle a large number of candidate instances in parallel. On the other hand, the most scalable approach involves conducting testing locally at the candidate instances, which does not rely on any resources outside the instance itself. Therefore, all operations can be done quickly in parallel and scale linearly. This approach trades accuracy for scalability.

Based on our root cause analysis, such a method exists because the part of the long tail problem we focus on is a property of nodes instead of the network. Accordingly, if we know the workload patterns of the VMs co-located with the victim VM, we should be able to predict if the victim VM will have a bad latency distribution locally without any network measurement.

In order to achieve this, we must infer how often long scheduling delays happen to the victim VM. Because the long scheduling delays caused by the co-located CPU-intensive VMs are not unique to network packet processing and any interrupt-based events will suffer from the same problem, we can measure the frequency of large delays by measuring the time for the target VM to wake up from the `sleep` function call. Delay in processing the timer interrupt serves as a proxy for delays in processing all hardware interrupts.

Algorithm 1 Instance Selection Algorithm

```
num_delay = 0
for i = 1 → M do
    sleep for S micro seconds
    if sleep time ≥ 10ms then
        num_delay++
    end if
end for
if num_delay ≤ LOW_MARK then
    return GOOD
end if
if num_delay ≤ HIGH_MARK then
    return MAY USE NETWORK TEST
end if
return BAD
```

Based on the results of our controlled experiments, we can design an instance selection algorithm to predict *locally* if a target VM will experience a large number of long scheduling delays. Algorithm 1 shows the pseudocode of our design. While the algorithm itself is straightforward, the challenge is to find the right threshold in EC2 to distinguish the two cases (`LOW_MARK` and `HIGH_MARK`) and to draw an accurate conclusion as quickly as possible (loop size M).

Our current policy is to be conservative in choosing the thresholds. In order words, we want to reduce the possibility of labeling bad nodes as good incorrectly (i.e., false positives). The cost of such conservatism is that we may label good nodes as bad incorrectly (i.e., false negatives), and as a result we must instantiate even more nodes to reach a desired number. To return N good nodes as requested by users, our system needs to choose from a pool of $K * N$ instances and find the best N instances of that set. The details of parameterization can be found in our conference paper [1]. The intuition is that because the candidates launched directly by EC2 contains 40% to 70% bad nodes (§ II-A) and our instance selection algorithm is not perfect, we need a relatively large pool of candidates, from which we can pick the ones with the lowest probability of producing long latency tails. Empirically, we find $K = 3$ to 4 are reasonable choices in practice, and we set `LOW_MARK` to be 13 and `HIGH_MARK` five times as `LOW_MARK`.

After Bobtail fulfills a user's request for N instances whose delays fall below `LOW_MARK`, we can apply the network-based latency testing to the leftover instances whose delays fall between `LOW_MARK` and `HIGH_MARK`; this costs the user no more than the one-time over-provisioning of VM instances but provides further value using the instances that users already paid for by the hour. Many of these nodes are likely false negatives which, upon further inspection, can be approved and returned to the user. In this scenario, scalability is no longer a problem because we no longer need to make a decision within minutes. Aggregate network throughput for testing can be thus much reduced.

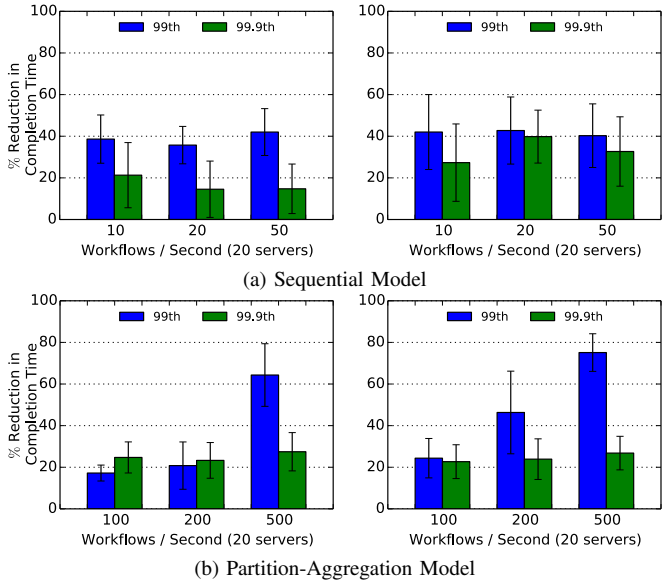
V. Evaluation

In this section, we compare the latency tails of instances selected by Bobtail with those launched directly via the standard mechanism using two common workload patterns—sequential and partition-aggregation workloads. In sequential workloads, a client calls some number of servers in series to complete a single, timed observation. In partition-aggregation workloads, a client calls all workers in parallel for each timed observation.

For both workload patterns, we compare 40 small instances launched directly by EC2 to 40 small instances selected by our system from the same data center. To select 40 good instances, we use Bobtail to choose from a pool of 160 candidate instances. That is, we launched $K = 4$ times as many instances to find the desired number of good ones. In addition, we launch four extra large instances for every 40 small instances to run the measurement clients. We did this because of the observation that extra large instances do not experience the extra long tail problem; we therefore can blame the server instances for bad latency distributions.

A. Sequential Model

Our traffic models for both sequential and partition-aggregation workloads have inter-arrival times of client requests forming a Poisson process. For sequential workloads, we apply the workload model to 10-node, 20-node, and 40-node server groups. In this case, the client sends small requests, and the servers reply with a message size randomly chosen from among 1KB, 2KB, and 4KB. For each workflow, instead of sending requests to all the servers, the client will randomly choose one server from the groups of sizes 10, 20, and 40. Then, it will send 10 synchronous requests to the chosen server; the total time to complete all 10 requests is



(a) Sequential Model

(b) Partition-Aggregation Model

Fig. 4. Reduction in flow tail completion time for sequential and partition-aggregation models by using Bobtail in two data centers in EC2's US east region. The mean reduction time is presented with a 90% confidence interval.

then used as the workflow RTT. The workflow rates for the sequential model include 10, 20, and 50 workflows per second.

Figure 4(a) shows our improvement under the sequential model with 20 servers per group. Bobtail brings a 35% to 40% improvement to sequential workloads at the 99th percentile across all experiments, and it roughly translates to an 8ms reduction. While not shown in the figure, there is a similarity in the reduction of flow completion time with different numbers of server nodes, which shows that the tail performance of the sequential workflow model only depends on the ratio of bad nodes among all involved server nodes. Essentially, the sequential model demonstrates the average tail performance across all server nodes by randomly choosing one server node each time with equal probability at the client side.

B. Partition-Aggregation Model

For the partition-aggregation model, we use the same 10, 20, and 40-node groups to evaluate Bobtail. In this case, the client always sends requests to all servers concurrently, and the workflow finishes as the slowest response returns; servers always reply with 2KB of random data. The RTT of the slowest server is effectively the RTT of the workflow.

Figure 4(b) shows improvement under the partition-aggregation model with 20 servers involved. The improvement brought by Bobtail at the 99th percentile varies from less than 20% to over 60%, and the improvement at 99.9th percentile is always around 20%. In addition, while not shown in the figure, the reduction in tail completion time diminishes as the number of servers involved in the workload increases.

VI. Discussion

The key challenge of mitigating performance interference in public clouds is dealing with the semantic gap between guests and hosts. Bobtail provides a guest-centric solution that allows cloud users to avoid long latency tails without changing any of the underlying infrastructure. Since the tail latency distribution has become an important metric and received lots of attention among practitioners [17], public cloud users can leverage Bobtail today to improve their applications. We obtained such an ability by carefully characterizing the impact of the performance interference and analyzing its root cause entirely from the perspective of cloud users.

Alternatively, cloud providers can provide host-centric solutions by modifying their cloud infrastructure and placement policy in a way that is completely transparent to their users. For example, new versions of Xen's credit scheduler [18] may help alleviate the long tail latency problem. More generally, providers can allocate fewer VMs on each physical machine to relieve resource contention, at the cost of hardware utilization. On the other hand, cloud providers can also challenge the semantic gap by breaking the virtualization abstraction. If they can infer the types of workloads running in the guest VMs, they will be able to overhaul their VM placement policy to allocate different types of VMs in different regions in the first place. In this arrangement, cloud providers need to make sure that the VMs allocated in the same region exhibit compatible resource usage patterns so that they do not cause as much performance interference to each other.

References

[1] Y. Xu, Z. Musgrave, B. Noble, and M. Bailey, "Bobtail: Avoiding Long Tails in the Cloud," in *Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI'13)*, Lombard, IL, USA, April 2013.

[2] S. Garfinkel, *Architects of the Information Society: Thirty-Five Years of the Laboratory for Computer Science at MIT*, H. Abelson, Ed. MIT Press, 1999.

[3] A. W. S. LLC, "Amazon Elastic Compute Cloud (EC2)," http://aws.amazon.com/ec2/.

[4] Techcrunch, "There Goes The Weekend! Pinterest, Instagram And Netflix Down Due To AWS Outage," http://techcrunch.com/2012/06/30/there-goes-the-weekend- pinterest-instagram-and-netflix-down-due-to-aws-outage/.

[5] C. Labovitz, "How Big is Amazon's Cloud?" http://www.deepfield.net/2012/04/how-big-is-amazons-cloud/.

[6] E. Keller, J. Szefer, J. Rexford, and R. B. Lee, "NoHype: Virtualized Cloud Infrastructure without the Virtualization," in *Proceedings of the 37th International Symposium on Computer Architecture (ISCA'10)*, Saint-Malo, France, June 2010.

[7] Y. Koh, R. C. Knauerhase, P. Brett, M. Bowman, Z. Wen, and C. Pu, "An Analysis of Performance Interference Effects in Virtual Environments," in *Proceedings of the 2007 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS'07)*, San Jose, CA, USA, April 2007.

[8] Y. Mei, L. Liu, X. Pu, and S. Sivathanu, "Performance Measurements and Analysis of Network I/O Applications in Virtualized Cloud," in *Proceedings of the 3rd IEEE International Conference on Cloud Computing (CLOUD'10)*, Miami, FL, USA, June 2010.

[9] J. Ahn, C. Kim, J. Han, Y. ri Choi, and J. Huh, "Dynamic Virtual Machine Scheduling in Clouds for Architectural Shared Resources," in *Proceedings of the 4th USENIX conference on Hot Topics in Cloud Computing (HotCloud'12)*, Boston, MA, USA, June 2012.

[10] P. Chen and B. Noble, "When Virtual Is Better Than Real," in *Proceedings of the Eighth Workshop on Hot Topics in Operating Systems (HOTOS'01)*, Washington, DC, USA, May 2001.

[11] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data Center TCP (DCTCP)," in *Proceedings of the ACM SIGCOMM 2010 conference (SIGCOMM'10)*, New Delhi, India, August 2010.

[12] D. Zats, T. Das, P. Mohan, D. Borthakur, and R. Katz, "DeTail: Reducing the Flow Completion Time Tail in Datacenter Networks," in *Proceedings of the ACM SIGCOMM 2012 conference (SIGCOMM'12)*, Helsinki, Finland, August 2012.

[13] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," in *Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation (OSDI'04)*, San Francisco, CA, USA, March 2004.

[14] Z. Ou, H. Zhuang, J. K. Nurminen, A. Ylä-Jääski, and P. Hui, "Exploiting Hardware Heterogeneity within the Same Instance Type of Amazon EC2," in *Proceedings of the 4th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud'12)*, Boston, MA, USA, June 2012.

[15] G. Wang and T. S. E. Ng, "The Impact of Virtualization on Network Performance of Amazon EC2 Data Center," in *Proceedings of the 29th conference on Information communications (INFOCOM'10)*, San Diego, CA, USA, March 2010.

[16] 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 the 19th ACM Symposium on Operating Systems Principles (SOSP'03)*, Bolton Landing, NY, USA, October 2003.

[17] S. Newman, "Three Latency Anomalies," http://amistrongeryet.blogspot.com/2010/04/three-latency-anomalies.html.

[18] xen.org, "Xen Credit Scheduler," http://wiki.xen.org/wiki/Credit_Scheduler.