The Unobservability Problem in Clouds

Anshul Gandhi†, Parijat Dube*, Alexei Karve*, Andrzej Kochut*, Harsha Ellanti†
† Stony Brook University, * IBM Research

***Abstract*—The cloud is not transparent. Users of cloud computing cannot control or monitor important information about their VMs or services, such as placement, true resource allocation, virtualization overhead, etc. Likewise, cloud providers cannot obtain important information about their users' deployment such as the application model, the role of each VM, etc. While such information is not required to be revealed, we claim that this lack of information prevents users from fully understanding their resource availability, and limits the feasibility of various performance management solutions. We refer to this lack of information as the "Unobservability" problem.**

In this paper, we describe the unobservability problem and present various use cases from our experience managing a medium-scale cloud deployment with several hundred VMs and experiments on EC2 that highlight the severe impact of unobservability on performance, and the limitations it imposes on users and cloud providers. We show that, interestingly, unobservability often diminishes the potential benefits of cloud computing. To address unobservability, we present and evaluate a practical solution to the unobservability problem that reveals important unobservable information without requiring any instrumentation or changes to the cloud.

I. Introduction

Cloud computing offers several benefits over traditional physical deployments such as low cost, elasticity, high availability, and access to useful services such as automated deployment and monitoring. Importantly, the ability to pay-as-you-go makes it lucrative for small and medium businesses to economically deploy their dynamic applications in the cloud. However, cloud computing does have its shortcomings.

In this paper, we focus on the "**unobservability**" problem which relates to the inability of those involved in cloud computing, such as Cloud Service Providers (CSPs) and users, from observing each others' configuration and settings. In other words, we focus on the *lack of transparency* in the cloud. The unobservability problem typically manifests itself by restricting the information available to users and CSPs. For example, cloud users typically do not know the identity of the physical servers that are hosting their VMs or the specifics of the virtualization technology used by the hosts. Likewise, CSPs are often not aware of the services and software employed by the cloud users in their VMs or the role of each VM in the user's application deployment.

A good example of challenges related to unobservability in clouds is running applications in the IBM Bluemix [1] environment. IBM Bluemix is a cloud application development platform allowing users to rapidly develop, test, and publish cloud applications. It is powered by the IBM IaaS Cloud [2] and also offers base IaaS services such as OpenStack-based VMs [3]. Because of the well established cloud layering principles, applications written in Bluemix do not have insight into infrastructure specifics on which they execute, such as the identity and configuration of the physical machine on which they are hosted and the resource contention at the host. Obtaining such insights and information could help developers predict and optimize the performance of their applications.

Note that the cloud is not required to be transparent. In fact, by design, the cloud is expected to be opaque to some extent. The opaque nature of the cloud allows CSPs to offer low-cost services to users by independently managing VM placement and resource allocation as aggressively as needed without revealing their decision logic. Likewise, users do not have to reveal any information about their intended use of cloud resources, thus protecting them from CSPs and other users. However, this lack of transparency *limits the full potential of the cloud*. We present various examples using public and private clouds that highlight the severity and spread of unobservability in clouds. *We find that, interestingly, unobservability often hampers the many benefits of cloud computing such as elasticity and availability*.

While the unobservability problem is more widespread in public clouds, it also affects private clouds. Users in a managed dedicated cloud, such as IBM's Cloud OpenStack Services [4] which runs on SoftLayer [2] Bare Metal Servers, typically cannot control their VM placement and scheduling. Of course, the private cloud administrator can expose such functionality to specific users, if needed. Likewise, the CSP cannot observe all user operations that are being performed within the VMs. Our experience managing a medium-scale private OpenStack-deployed cloud (with KVM hypervisors) has revealed several cases of unobservability that significantly impact performance.

We assert that addressing, even partially, the unobservability problem can significantly improve cloud adoption among users. As a concrete example, consider the autoscaling functionality that helps users leverage the elastic nature of cloud computing. Many CSPs today, including Amazon [5], RightScale [6], and cloud software solutions such as OpenStack [7], offer rule-based solutions (not necessarily for free) to users for autoscaling their deployments. Such rule-based autoscaling solutions place the burden of determining the scaling thresholds and rules on the users as the CSPs do not have any information about the users' deployments. If the unobservability problem can be addressed, however, then CSPs can offer a fully automated autoscaling functionality to users.

Unfortunately, the unobservability problem is difficult to address because of the nature of clouds. For example, while instrumenting user VMs can improve visibility for CSPs, it is not desirable, especially for privacy reasons. Fortunately, it

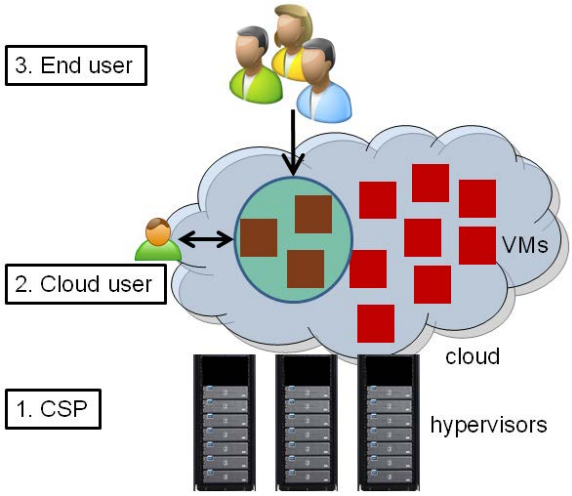


Fig. 1. Abstract cloud model with the three distinct entities.

is possible to *infer* some missing information by analyzing the performance of cloud-deployed VMs. We describe one such inference solution to the unobservability problem in this paper, and employ it to enable CSP-driven autoscaling of user applications. Importantly, our inference solution does *not* require any changes to the cloud platform or user deployment.

The rest of the paper is organized as follows. We describe the unobservability problem in detail in Section II. We then present, in Sections III, IV, V, and VI, several use cases based on our experience with a medium-scale OpenStack-deployed cloud platform (hosting *several hundred VMs*) and our experiments on EC2 that highlight the unobservability problem in public and private clouds. We then present a potential solution to the unobservability problem in Section VII based on inference techniques that requires no changes to the cloud platform or application. We discuss related work in Section VIII, and present our conclusions in Section IX.

II. The Unobservability Problem

The unobservability problem refers to the lack of visibility and control among the entities in a cloud computing environment. Consider the cloud deployment example in Figure 1. Here, there are three distinct entities that interact with the cloud. The first is the **CSP**, for example, AWS [8], Microsoft Azure [9], and Google Cloud Platform [10]. The second is the **cloud user**, for example, Netflix [11] and BestBuy [12], who purchase services, such as VMs or storage, from the CSPs. The third is the **end user** of these customers, for example, Netflix customers, who contract various cloud-based services through cloud users. Note that end users could also directly purchase services from CSPs. Based on this model, there are at least four different types of unobservability problems that can be identified:

- **(Type I) Cloud user-cloud user:** Cloud users whose deployments are co-located on the same physical machines are typically unaware of each others' resource consumption or application model. This unobservability often leads to physical resource contention and performance interference, and is an important research topic in itself [13], [14], [15].

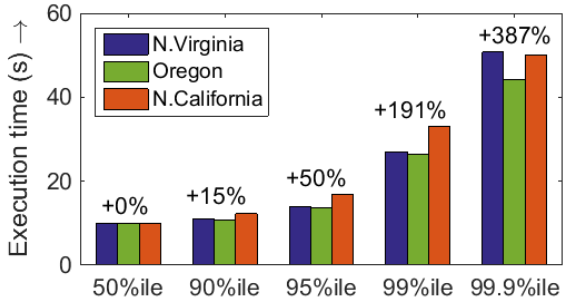


Fig. 2. Performance variation on EC2. Public cloud VMs exhibit significant variation in performance, probably due to resource contention from (unobservable) colocated VMs.

- **(Type II) Cloud user-CSP:** In this case, the cloud user cannot observe the configuration and control logic used by the CSP. For instance, the cloud user is typically unaware of (and has limited control on) the placement of her VMs on the physical machines in the cloud, is unaware of the exact amount of resources allocated to her VMs at all times (due to contention), and is also unaware of the decision logic that guides placement and allocation of new VMs.

- **(Type III) CSP-Cloud user:** The CSP typically does not encroach into the cloud user's deployment. For example, the CSP cannot observe the software and services employed by the user, or their configuration settings. However, the CSP can monitor a few parameters, such as resource utilization. The lack of transparency in this case limits the assistance that the CSP can provide to the cloud user (see Section VI-A). Note that Type III and Type II unobservability occur simultaneously in the same scenario.

- **(Type IV) End user-cloud user:** End users who employ services offered by cloud users face the same unobservability problems as experienced by cloud users in Type II. That is, the end user cannot observe the model and deployment of the cloud user. Thus, the end user cannot determine how the cloud service will respond in case of a load spike or a gradual increase in traffic. Likewise, the end user cannot ascertain if and when the service will be unavailable or slow due to, for instance, maintenance or resource contention.

In the following sections, we present several use cases that highlight the Type I, Type II, and Type III unobservability problems. A good example of Type IV unobservability is discussed in Deng et al. [16].

III. Performance variation (Type I)

In this section, we present our empirical results illustrating the variability in performance experienced by public cloud VMs. We set up 15 micro instances on EC2 [17], 5 each on

```
d, 0.0%wa, 0.0%hi, 8.0%si, 0.0%st
85401588k free,   350696k buffers
100663288k free, 37821740k cached
 %CPU  %MEM    TIME+   COMMAND
 100.0  0.0   9062:16 qemu-kvm
  99.7  0.2   0:48.22 qemu-kvm
  99.0  0.0   0:49.88 vhost-6272
  99.0  0.2   1:00.73 qemu-kvm
  99.0  0.3   1:03.20 qemu-kvm
  99.0  0.0   0:50.83 vhost-7192
```

Fig. 3. When VM network traffic is high, the hypervisor `vhost` processes consume significant CPU, as shown in the `top` output above, thereby affecting the CPU capacity available to the VMs.

the N.Virginia, Oregon, and N.California regions. We then run a CPU-bound microbenchmark repeatedly on all 15 VMs over the span of a few days, and record execution times.

Figure 2 shows the various percentiles of execution times recorded for EC2, broken down by region. The percentage values above the bars represent the increase in execution time relative to the 50%ile (median) value. We see that the performance varies significantly - the top 5% execution times are a factor 1.5 larger than the median, and the top 1% are almost a factor 3 larger than the median. Ideally, there should be no difference in execution time for various percentiles. However, even if we allow for some minor variations, the substantial difference in execution times for the higher percentiles cannot be easily explained. We attribute this variation in performance to possible resource contention due to colocated instances. Similar performance degradation has been reported in prior work [15], [18], [14]. These are examples of Type I unobservability (cloud user-cloud user).

IV. Virtualization overhead (Type II)

We now illustrate several examples of Type II unobservability (cloud user-CSP) by analyzing the overhead of virtualization and its non-trivial effects on VM performance based on our experience managing a multi-node OpenStack deployment. For this case study, we use our SoftLayer-hosted OpenStack deployment with 24 compute nodes (KVM hypervisors) named kvm001-kvm024 and 3 controllers named ops001-ops003. Each compute node has 32 Intel(R) Xeon(R) CPU E5-2650 v2 (2.60GHz) and 128GB RAM.

A. Networking overhead

The network traffic created by VMs can impact the CPU availability of the compute nodes in a non-trivial manner. In order to examine the effect of VM network traffic on compute node resources, we employ the network-intensive Netperf benchmark [19] on 375 client and server pairs, for a total of 750 VMs (each VM is allocated 1 CPU) deployed over time in batches. With this high network load, in addition to the `qemu-kvm` processes, there are additional corresponding `vhost` processes running on the compute nodes that handle (among other things) the virtual network traffic, and use high

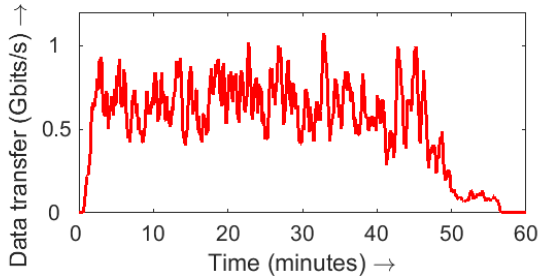


Fig. 4. High network traffic at the hypervisor due to image transfers (shown above) can significantly slowdown other hypervisor operations, thus affecting the performance of hosted VMs.

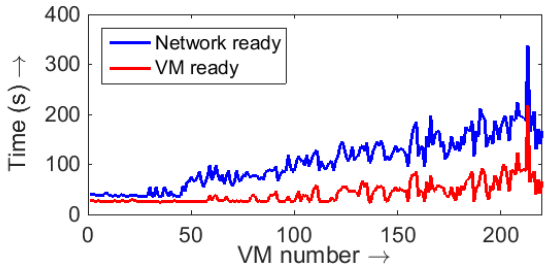


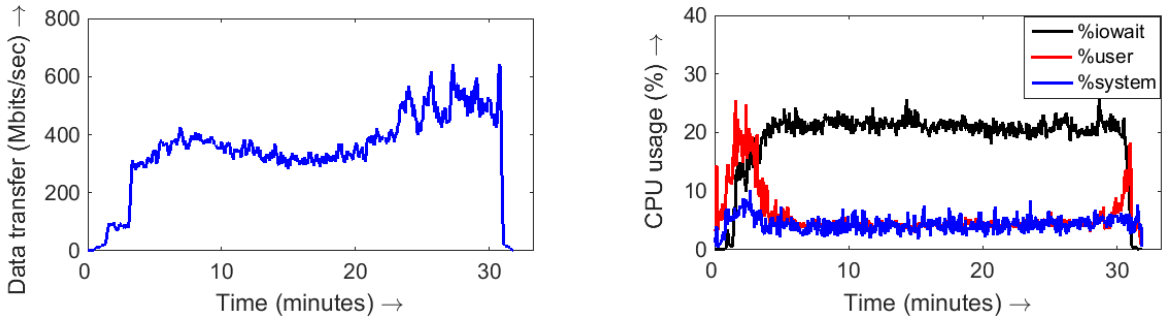
Fig. 5. I/O-intensive applications running on the VMs can overwhelm the hypervisor and affect cloud operations, such as delaying the launch of new VMs as depicted above.

CPU as shown in Figure 3. Due to the high `vhost` overhead, each VM effectively uses 2 CPUs (1 for VM, 1 for `vhost`). This dynamic additional CPU usage is not accounted for by the scheduler, resulting in possible over-allocation of resources. Due to Type II unobservability, the future VM users on these compute nodes will receive compromised CPU resources. Note that existing VMs also experience some performance degradation due to the CPU contention at the compute nodes.

As another example, consider the case where multiple VMs are *simultaneously* booted. Before booting a VM with the specified image, the image must be copied from the repository by the controller to the compute node hosting the VM. In this case, the controller is bottlenecked when the `glance-api` process is busy copying images (especially large images) from `glance` to the compute nodes during cold start. We illustrate this example by simultaneously booting 30 VMs whose (30GB) images were pre-loaded with the DayTrader [20] benchmark application. Figure 4 shows the high network traffic transmitted over eth1 from one of the controller nodes to the compute nodes. Note that each of the 3 controllers copies the images to 8 compute nodes, for a total of 24 compute nodes. The cold start (for simultaneous boot) takes anywhere between 20 minutes for the start of the first VM instance to 45 minutes for the start of the last VM instance. During this entire time, the controller is network bottlenecked, significantly slowing down any other tasks that the controller is responsible for, such as the OpenStack API services and schedulers.

B. I/O overhead

The I/O activity of VMs can likewise affect the compute nodes hosting the VMs (in addition to affecting colocated



(a) I/O transfer rate.

(b) CPU usage.

Fig. 6. I/O overhead due to hypervisor activities can lead to wasted CPU cycles. Figure 6(a) shows the high I/O activity at the hypervisor due to the simultaneous provisioning of several VMs, resulting in under-utilization of the CPU as depicted by Figure 6(b).

VMs). In order to analyze I/O overhead, we employ the Filebench benchmark [21] on several VMs. We sequentially launch 220 VMs, and run the benchmark on the provisioned VMs. The iowaits on the compute nodes when running Filebench overwhelm the server, delaying the launch of new VMs. Figure 5 shows the provisioning time, in order, of all 220 VMs. We see that the newer VMs (on the right) take much longer to boot, almost a factor 5 longer than the first few VMs, due to the server being overwhelmed by iowaits. The "VM ready" indicates the time when OpenStack considers the VM to be ready, and "Network ready" indicates the time when we can successfully `ssh` into the VM. The delay in launching new VMs is caused by the overhead involved in servicing multiple VM I/O requests (Filebench) simultaneously.

As another example, consider the case where multiple VMs are being provisioned on a compute node. We simultaneously boot 30 VMs on a single node using a RedHat 6.5 image. While the image is being expanded to the 60GB instance disk size, multiple (interleaved) I/O writes to the physical disk occur simultaneously. Figures 6(a) and 6(b) show the aggregate I/O write rate and the CPU usage at the compute node during the provisioning of the 30 VMs. We see that the node spends a significant fraction of time waiting for I/Os to complete. During this time, the CPU is under-utilized. The VMs take as much as 30mins to boot because of the slow I/O. By contrast, a single (isolated) RedHat image VM boot takes only 70s.

C. Overhead of controller processes

In OpenStack, cloud management processes such as the `openstack-nova-compute`, `libvirt` daemon, and `neutron-dhcp-agent` may be installed and running on the compute nodes. The scheduler does not account for their (phantom) resource usage, and this overhead can impair the resources allocated to the VMs deployed on these nodes.

V. Limited control over VM placement and resource allocation (Type II)

The private cloud is not immune to the unobservability problem. Common examples of lack of transparency in private clouds include VM placement logic and resource allocation, which we discuss below. In general, private clouds can be just as vulnerable to unobservability problems as public clouds. However, the private cloud can have more transparency as the users and cloud administrator(s) are typically from the same or affiliated organizations. In our experience managing an OpenStack-based private cloud at IBM, we find that there are some significant unobservability problems, specifically of Type II (cloud user-CSP), that affect private clouds.

A. VM placement

OpenStack users can launch new VMs by issuing nova `boot` commands. Unfortunately, the VM placement on physical nodes cannot be controlled by users. The placement is either dictated by the OpenStack scheduler logic (round-robin, by default), or is manually enforced by the administrator (via OpenStack directives such as `HostAggregates` and `AvailabilityZones`). While the default round-robin policy helps balance load among the available compute nodes, it can have undesirable performance effects for users. For instance, the round-robin policy does not help users who wish to launch multiple VMs on the same compute node to reduce network latency. Likewise, users who wish to launch additional VMs on the same compute nodes as their previous VMs cannot do so unless they know exactly which compute node their previous VMs were on. Because of Type II unobservability, such users cannot optimize their placement.

Consider the concrete example where we wish to provision multiple DayTrader application VMs *sequentially* on a private cloud. For this, we use our 30GB custom DayTrader image. The first VM that we launch takes about 4 minutes to boot because of the cold start required to copy the 30GB image from the `glance` repository to the target compute node. Now, if successive VMs are launched on this same compute node (which already has a copy of the image), the boot time is only about 40 seconds. However, because of the default round-robin scheduling policy, our successive VMs are launched on new compute nodes, thus requiring the full 4 minutes of provisioning time each.

B. Resource allocation

Cloud users can typically choose their instance size from a selection of pre-configured "flavors". These flavors describe

the number of CPU cores, the amount of memory, and the disk size that the VM will have. Unfortunately, VMs with the *same* flavor can have very *different* configurations due to the difference in hardware specifications of the underlying compute nodes. For example, our private cloud consists of two different types of compute nodes: one with 4 cores at 3.4GHz, and the other with 8 cores at 2GHz. When a user launches a 2-core flavor VM, then depending on the scheduling logic, the VM might be launched on the 3.4GHz node or the 2GHz node. Thus, the configuration (and performance) of the VM will be different depending on the compute node that it is launched on. Consider another example where we have only one type of compute node, but the node has hyper-threading which enables 8 threads for 4 cores. If a user requests a 1-core flavor VM, then depending on the availability, the VM might be launched on a hyperthreaded core that is being shared (two threads on a core) by other VMs. Thus, configuration can be affected even if the cloud has homogeneous hypervisors.

VI. Inefficient management solutions (Type III)

We now present more complex case studies that highlight the limitations imposed by unobservability on management solutions. Such unobservability problems significantly impact the efficacy of performance management solutions on cloud deployments. We present two such scenarios, namely cloud autoscaling and virtual desktop services, in the following subsections. We then present a solution in Section VII that can help address some of the challenges that unobservability poses for management solutions.

A. Cloud autoscaling

Autoscaling is a functionality that automatically scales (up or down) an application deployment in response to variations in workload so as to meet user-specified performance targets. Cloud users are typically interested in autoscaling their cloud-deployed applications while minimizing their VM rental costs. However, due to Type II unobservability (cloud user-CSP), new VMs launched by the users might not provide the expected level of performance (see Sections V-A and V-B). Further, determining the autoscaling rules and thresholds is a challenging task [22], [23] that requires a deep understanding of the application and the necessary performance modeling expertise to develop the scaling rules. Small and medium businesses and casual cloud users typically lack the resources required to overcome these hurdles. By contrast, the CSP is ideally suited to schedule and place new VMs on the user's behalf as it can monitor the resource consumption of all hypervisors and typically employs several system administrators to manage resource allocation. Unfortunately, due to Type III unobservability (CSP-cloud user), the CSP cannot determine when to scale the user's deployment [24], [25]. Consequently, CSP-offered autoscaling solutions today require the user to determine the scaling thresholds and rules, thus placing the burden of developing the scaling logic on the users.

Clearly, in this scenario, even a limited amount of information sharing between the users and the CSP can greatly improve the efficacy of cloud autoscaling. In Section VII we present a solution that leverages limited information sharing to improve the efficacy of cloud autoscaling without requiring any significant changes to the cloud or user application.

B. Virtual desktop service

Consider a virtual desktop service, such as VMware Horizon [26]. This service allows end-users to connect to VMs running desktop operating systems on servers in a remote data center. The only device local to the user is a "thin-client" - an inexpensive machine with limited computing power designed to render screen images received from the desktop server and forward keyboard and mouse events to the server. The device provides GUI interaction but does not necessarily perform any end-user computing. The data exchange between the "thin-client" and data center is facilitated using remoting protocols such as VNC.

Since the actual computation happens on commodity servers in the remote data center, the allocated virtual resources can be reused by end-users for other tasks when desktop users are not active. This results in increased average data center utilization, thus contributing to reduction of energy consumption and maintenance costs. However, reusing allocated resources requires direct access to underlying hypervisors. In a private cloud, the resource usage for each VM can be obtained directly from the hypervisor, and any configuration (relative VM shares) and operation changes (power and reset) to the VMs can be relayed via available APIs. By contrast, in a public cloud environment, the CSP assumes all responsibility for allocation of resources, placement of the VMs, and workload management of the physical servers. Deploying a virtualization-aware solution such as the one discussed above is not feasible in such opaque environments. Either the solution must rely on the CSP's physical resource management or the CSP must incorporate solution space knowledge and information in its management decisions. The latter is not feasible because of Type III unobservability.

VII. Possible Solution: Inference

We believe that the unobservability in clouds is largely caused by business needs. We argue that limited sharing of information in clouds across different entities will not jeopardize business interests. Instead, such sharing of information can provide collective benefits to all entities. This limited sharing can be exploited to develop, for instance, solutions for allocation and management of cloud resources, as we show in Section VII-B.

A. Inference

In order to enable sharing of information *without* requiring any changes at the user or CSP level, we propose using *inference* techniques. Inference techniques, such as those based on

machine learning or statistical methods, estimate unobservable system parameters based on the available limited information. For example, by examining the network latency of a VM, as reported by `ping`, we can deduce the location of the VM in a geographically distributed public cloud such as AWS [8]. Likewise, a change in device throughput or execution time as experienced by a cloud user can provide some information about the load on colocated VMs.

B. Evaluation

1) Deducing VM location: We first start with a simple evaluation. Consider a CSP that allows users to launch new VMs. If the CSP employs several geographically distributed data centers to provide capacity for hosting VMs, then the location of a newly launched VM might depend on the scheduling policy used by the CSP, which is not known to the user due to Type II unobservability (cloud user-CSP). However, by examining the `ping` times to the VM, one can infer the possible location of the physical server hosting the VM. For example, our experiments on Amazon EC2 [17] show that a `ping` time in excess of 200ms to a VM from New York suggests that the VM is located in the Asia Pacific region of EC2 (Tokyo, Sydney, or Singapore).

2) Inferring the bottleneck in a distributed system: Consider a private cloud where the cloud administrator is trying to debug the performance of a user's distributed cloud application. In this case, the administrator cannot directly monitor the user's deployment due to Type III unobservability (CSP-cloud user). However, if the user is willing to provide some information, then the CSP can infer the bottleneck VM.

We evaluate this inference approach on a small-scale cloud deployment. As a cloud user, we set up a popular web application benchmark, RUBiS [27] (Apache + Tomcat + MySQL). RUBiS supports multiple request classes, such as browse, store, and home (index.html). Now, as the CSP, we use a queueing network model with missing parameters (such as per-tier service times and end-to-end network delays) to model RUBiS. Here, the parameters are unobservable from the CSP's perspective (Type III unobservability), but are known to the user as the user can easily benchmark her application. We employ a workload generator to create varying load for RUBiS and collect CPU utilization, per-class request rate, and per-class response time. We allow the CSP to observe these values; however, the CSP cannot observe per-tier service times and network delays (limited information sharing). In order to infer the missing parameters, we (CSP) use a Kalman filter, a popular online estimation tool. Our inference approach estimates the service times for the browse request at the Apache frontend web tier, the Tomcat application tier, and the MySQL database tier to be 1.3ms, 5.7ms, and 1.1ms, respectively. These rightly suggest that the browse requests are bottlenecked at the application tier. We validate this finding by directly accessing the application and individually scaling each of the three tiers in response to increased load to determine the bottleneck tier. For the store requests, our estimated service times for the three tiers are 2.3ms, 3.1ms, and 16.9ms, respectively. These rightly suggest that the store requests are bottlenecked at the database tier. Note that the bottleneck tier for an application cannot simply be detected by looking at the CPU utilization, since the utilization at a tier is a result of *multiple* request classes. Further, the database tier exhibits performance degradation even at a moderately low CPU utilization of 15%, as in our experiments.

Using our inference approach, we also estimate the network delays for each of the request types. Our estimates for the end-to-end network delays for the browse and store request classes are 14.1ms and 0.5ms. The browse request involves multiple (`GET`) queries between the application tier VM and the database tier VM, both of which are hosted on different hypervisors. Thus, the end-to-end network delay is significant. The store request involves a single (`PUT`) query between the application tier VM and the database tier VM, resulting in a lower delay. The fact that browse requests in the RUBiS application are bottlenecked at the application tier and incur significant network delays was also observed by previous studies (for example, Malkowski et al. [28]) that employed *offline* benchmarking.

The above is an example of how inference can be used to address unobservability by providing limited information sharing. In this case, the user allowed the CSP to only observe end-to-end performance to infer the bottleneck tier.

3) CSP-managed autoscaling: Due to Type II unobservability (cloud user-CSP), autoscaling is typically left to the users, as discussed in Section VI-A. However, inference, with the help of limited information sharing, enables the CSP to control and manage the autoscaling of a user's cloud deployment. In our inference approach above, the CSP can infer the bottleneck tier and estimate the network delays and per-tier service times for a user-deployed cloud application. If the arrival rate to the user's cloud application increases, the CSP can leverage the inferred information to suggest scale-out of the bottleneck tier to maintain acceptable response times. Such a service (CSP-managed autoscaling) can then be offered to cloud users for an additional price. We employ this approach to successfully autoscale the application tier when the browse workload intensity increases under RUBiS.

C. Extensions

We now formalize our proposed inference approach. Figure 7 shows a solution architecture that leverages information provided by the user (application logic, performance) and the CSP (cloud infrastructure) for dynamic management of cloud resources to conform to user-specified performance SLAs. Whereas application performance can be concretely measured in terms of standard metrics, like throughput, average and/or percentile end-to-end delay, etc, the application logic is a loosely defined term. Depending on the type of application and the willingness of the user to share information, appli-

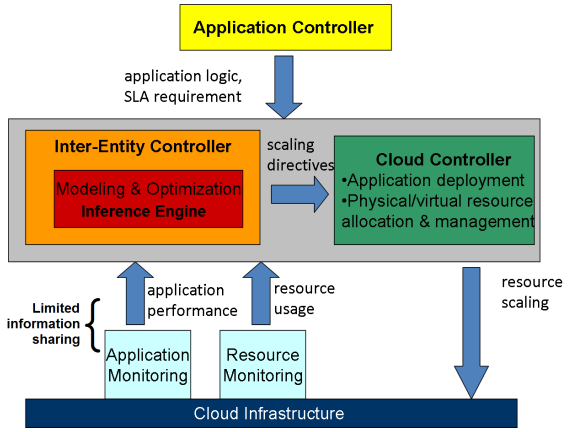


Fig. 7. Solution architecture for CSP-managed autoscaling of user applications in the cloud via our inference approach and limited information sharing.

cation logic can have different meanings. For transactional workloads, the application logic can include the different tiers and their interconnection topology, and the different service classes and their flow within the tiers. In this case, multi-class queueing networks can effectively model the application and its end-to-end performance [29], [30]. For other workloads, such as data analytics, application logic may include only per-tier delays. In this case, the application can be modeled as a black-box. Statistical [31], [32], [33] and machine learning [34], [35], [36], [37] techniques can then be employed for application performance modeling.

VIII. Related Work

There has been some very recent work that has focused on the unobservability problem. Kocsis et al. [38] propose running light-weight benchmarks in the background to estimate resource availability and measure platform characteristics from a user's perspective to address Type I unobservability (cloud user-cloud user). While this approach helps mitigate unobservability, it incurs overhead due to the additional load created by the benchmarks, and only provides limited (sampled) information. IC^2 [39] is a user-level solution that addresses Type I unobservability (cloud user-cloud user). IC^2 monitors application performance periodically to detect interference. When interference is detected, the application parameters are reconfigured to optimize performance based on the monitored level and type of interference. There are also a lot of other related works [15], [18], [14], [40], [41], [42], [43], [44] that focus on the unobservability problem from the perspective of a (co-located) cloud user (Type I) by measuring performance interference in cloud instances.

DeepDive [13] proposes to understand user application behavior in clouds by cloning the application setup (via JustRunIt [45]) in a sandboxed environment. DeepDive and JustRunIt focus on the unobservability problem from the CSP's perspective (Type III). The JustRunIt framework's primary limitation is the requirement of an intrusive proxy to replicate incoming user requests. Such an approach might not be feasible for customers, such as financial companies, who wish to protect their workloads. Stay-Away [46] regulates the performance of a time-sensitive application by proactively throttling other co-located batch applications. However, the authors assume that the class of the application running on a user VM is known, and that the VM performance can be throttled. Unfortunately, Type III unobservability (CSP-cloud user) makes it difficult to obtain this information from the users, unless limited information sharing is assumed.

Deng et al. [16] consider the unobservability problem from the perspective of an end user (Type IV) who wants to make use of an online cloud-based service but cannot assess the robustness of the service due to unobservability. The authors assume a simple linear model for application response time and employ statistical methods to derive the coefficients of the model.

Solutions to the unobservability problem typically rely on modeling and inference. Nathuji et al. [15] use a MIMO feedback approach to create an online model of the application using its VM resource allocations as input and application performance as output. Pesto [47] employs a black-box approach combined with analytical queueing results to estimate the I/O needs of an application. Our proposed inference solution in Section VII also relies on modeling to estimate unobservable system parameters to aid CSPs and cloud users.

IX. Conclusion

In this paper, we present the unobservability problem in clouds and discuss its many facets. We present empirical case studies based on our experience managing a medium-scale cloud deployment at IBM consisting of several hundred VMs and our experiments on EC2. These case studies highlight the many issues with unobservability that users often either ignore or take for granted. We assert that at least some of these problems can be addressed by techniques that infer unobservable parameters in public and private cloud deployments.

An overarching goal of this paper is to make the case for (some form of) transparency in public clouds. We believe that there are cases where transparency helps both the users and the CSP. For example, by providing some information about the application setup to CSPs, the users can benefit from better performance since the CSP can now provide better VM placement or suggest scaling when needed. Of course, transparency can potentially expose the CSP and users to security and privacy risks that must be assessed before sharing information. We will examine security issues in future work.

The unobservability problem can also be thought of as a by-product of commercializing clouds as it is the result of a business choice made by CSPs to deliver best-effort service at low cost to a large audience. Specific users who care more about performance and transparency will likely avoid (economical) clouds and employ (expensive) dedicated servers. However, we believe that public and private clouds

can be made more appealing to such users by providing some form of transparency, and by leveraging inference solutions such as the one presented in Section VII.

REFERENCES

[1] "IBM Bluemix," https://console.ng.bluemix.net.
[2] "SoftLayer Bare Metal Servers," http://www.softlayer.com.
[3] "IBM OpenStack," https://console.ng.bluemix.net/solutions/open-architecture#vms.
[4] "IBM Cloud OpenStack Services," https://open.ibmcloud.com.
[5] "Amazon Auto Scaling," http://aws.amazon.com/autoscaling.
[6] "Auto-Scaling Arrays," http://www.rightscale.com/products/automation-engine.php.
[7] "OpenStack Heat," https://wiki.openstack.org/wiki/Heat.
[8] "Amazon Web Services," http://aws.amazon.com.
[9] "Microsoft Azure," http://azure.microsoft.com.
[10] Google Cloud Platform, https://cloud.google.com.
[11] "5 Lessons Weve Learned Using AWS," http://techblog.netflix.com/2010/12/5-lessons-weve-learned-using-aws.html.
[12] "Best Buy Slashes App Development Time and Resources with Google App Engine," https://cloud.google.com/customers/best-buy.
[13] D. Novaković, N. Vasić, S. Novaković, D. Kostić, and R. Bianchini, "DeepDive: Transparently Identifying and Managing Performance Interference in Virtualized Environments," in *Proceedings of the 2013 USENIX Annual Technical Conference*, San Jose, CA, USA, 2013, pp. 219–230.
[14] S. K. Barker and P. Shenoy, "Empirical Evaluation of Latency-sensitive Application Performance in the Cloud," in *Proceedings of the 2010 ACM Conference on Multimedia Systems*, Phoenix, AZ, USA, pp. 35–46.
[15] R. Nathuji, A. Kansal, and A. Ghaffarkhah, "Q-clouds: Managing performance interference effects for QoS-aware clouds," in *Proceedings of the 5th European Conference on Computer Systems*, Paris, France, 2010, pp. 237–250.
[16] N. Deng, Z. Xu, C. Stewart, and X. Wang, "From the Outside Looking In: Probing Web APIs to Build Detailed Workload Profiles," in *Proceedings of the 9th International Workshop on Feedback Computing*, Philadelphia, PA, USA, 2014.
[17] "Amazon Elastic Compute Cloud," http://aws.amazon.com/ec2.
[18] Y. Koh, R. Knauerhase, P. Brett, M. Bowman, Z. Wen, and C. Pu, "An Analysis of Performance Interference Effects in Virtual Environments," in *Proceedings of the 2007 International Symposium on Performance Analysis of Systems Software*, San Jose, CA, USA, 2007, pp. 200–209.
[19] Rick Jones, "Netperf," http://www.netperf.org/netperf.
[20] Apache, "DayTrader," http://geronimo.apache.org/GMOxDOC20/daytrader.html.
[21] "Filebench: File system benchmark," http://sourceforge.net/projects/filebench.
[22] A. Gandhi, M. Harchol-Balter, R. Raghunathan, and M. Kozuch, "AutoScale: Dynamic, Robust Capacity Management for Multi-Tier Data Centers," *Transactions on Computer Systems*, vol. 30, 2012.
[23] A. Gandhi, Y. Chen, D. Gmach, M. Arlitt, and M. Marwah, "Hybrid Resource Provisioning for Minimizing Data Center SLA Violations and Power Consumption," *Sustainable Computing: Informatics and Systems*, vol. 2, pp. 91–104, 2012.
[24] A. Gandhi, P. Dube, A. Karve, A. Kochut, and L. Zhang, "Adaptive, Model-driven Autoscaling for Cloud Applications," in *Proceedings of the 11th International Conference on Autonomic Computing*, Philadelphia, PA, USA, 2014, pp. 57–64.
[25] ——, "Modeling the Impact of Workload on Cloud Resource Scaling," in *Proceedings of the 26th International Symposium on Computer Architecture and High Performance Computing*, Paris, France, 2014.
[26] "VMware Horizon 6," http://www.vmware.com/products/horizon-view.
[27] "RUBiS: Rice University Bidding System," http://rubis.ow2.org.
[28] S. Malkowski, M. Hedwig, and C. Pu, "Experimental Evaluation of N-tier Systems: Observation and Analysis of Multi-bottlenecks," in *Proceedings of the 2009 IEEE International Symposium on Workload Characterization*, Austin, TX, USA, 2009, pp. 118–127.
[29] B. Urgaonkar, G. Pacifici, P. Shenoy, M. Spreitzer, and A. Tantawi, "An analytical model for multi-tier internet services and its applications," in *Proceedings of the 2005 International Conference on Measurement and Modeling of Computer Systems*, Banff, Alberta, Canada, 2005, pp. 291–302.
[30] G. Pacifici, M. Spreitzer, A. Tantawi, and A. Youssef, "Performance management for cluster-based web services," *IEEE Journal on Selected Areas in Communications*, vol. 23, no. 12, pp. 2333–2343, 2005.
[31] B. Xi, Z. Liu, M. Raghavachari, C. H. Xia, and L. Zhang, "A Smart Hill-climbing Algorithm for Application Server Configuration," in *Proceedings of the 13th International Conference on World Wide Web*, New York, NY, USA, 2004, pp. 287–296.
[32] H. Nguyen, Z. Shen, X. Gu, S. Subbiah, and J. Wilkes, "Agile: Elastic distributed resource scaling for infrastructure-as-a-service," in *Proceedings of the 10th International Conference on Autonomic Computing*, San Jose, CA, USA, 2013, pp. 69–82.
[33] A. Verma, L. Cherkasova, and R. H. Campbell, "Profiling and evaluating hardware choices for mapreduce environments: An application-aware approach," *Performance Evaluation*, vol. 79, pp. 328 – 344, 2014.
[34] J. Rao, X. Bu, C.-Z. Xu, L. Wang, and G. Yin, "Vconf: A reinforcement learning approach to virtual machines auto-configuration," in *Proceedings of the 6th International Conference on Autonomic Computing*, Barcelona, Spain, 2009, pp. 137–146.
[35] X. Dutreilh, A. Moreau, J. Malenfant, N. Rivierre, and I. Truck, "From data center resource allocation to control theory and back," in *Proceedings of the 3rd IEEE International Conference on Cloud Computing*, Miami, FL, USA, 2010, pp. 410–417.
[36] P. Lama and X. Zhou, "Aroma: Automated resource allocation and configuration of mapreduce environment in the cloud," in *Proceedings of the 9th International Conference on Autonomic Computing*, San Jose, CA, USA, 2012, pp. 63–72.
[37] C. Delimitrou and C. Kozyrakis, "Quasar: Resource-efficient and qos-aware cluster management," in *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, Salt Lake City, UT, USA, 2014, pp. 127–144.
[38] I. Kocsis, A. Pataricza, Z. Micskei, A. Kövi, and Z. Kocsis, "Analytics of resource transients in cloud-based applications," *International Journal of Cloud Computing*, vol. 2, no. 2, pp. 191–212, 2013.
[39] A. K. Maji, S. Mitra, B. Zhou, S. Bagchi, and A. Verma, "Mitigating interference in cloud services by middleware reconfiguration," in *Proceedings of the 15th International Middleware Conference*, Bordeaux, France, 2014, pp. 277–288.
[40] N. Vasić, D. Novaković, S. Miučin, D. Kostić, and R. Bianchini, "DejaVu: Accelerating Resource Allocation in Virtualized Environments," in *Proceedings of the 17th International Conference on Architectural Support for Programming Languages and Operating Systems*, London, England, UK, 2012, pp. 423–436.
[41] S. Zhuravlev, S. Blagodurov, and A. Fedorova, "Addressing Shared Resource Contention in Multicore Processors via Scheduling," in *Proceedings of the 15th International Conference on Architectural Support for Programming Languages and Operating Systems*, Pittsburgh, PA, USA, 2010, pp. 129–142.
[42] S. Govindan, J. Liu, A. Kansal, and A. Sivasubramaniam, "Cuanta: Quantifying effects of shared on-chip resource interference for consolidated virtual machines," in *Proceedings of the 2nd ACM Symposium on Cloud Computing*, Cascais, Portugal, 2011, pp. 1–14.
[43] J. Mars, L. Tang, R. Hundt, K. Skadron, and M. L. Soffa, "Bubble-Up: Increasing Utilization in Modern Warehouse Scale Computers via Sensible Co-locations," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, Porto Alegre, Brazil, 2011, pp. 248–259.
[44] X. Zhang, E. Tune, R. Hagmann, R. Jnagal, V. Gokhale, and J. Wilkes, "CPI^2: CPU Performance Isolation for Shared Compute Clusters," in *Proceedings of the 8th ACM European Conference on Computer Systems*, Prague, Czech Republic, 2013, pp. 379–391.
[45] W. Zheng, R. Bianchini, G. J. Janakiraman, J. R. Santos, and Y. Turner, "JustRunIt: Experiment-based Management of Virtualized Data Centers," in *Proceedings of the 2009 USENIX Annual Technical Conference*, San Diego, CA, USA, 2009, pp. 243–258.
[46] N. Rameshan, L. Navarro, E. Monte, and V. Vlassov, "Stay-away, protecting sensitive applications from performance interference," in *Proceedings of the 15th International Middleware Conference*, Bordeaux, France, 2014, pp. 301–312.
[47] A. Gulati, G. Shanmuganathan, I. Ahmad, C. Waldspurger, and M. Uysal, "Pesto: Online Storage Performance Management in Virtualized Datacenters," in *Proceedings of the 2nd ACM Symposium on Cloud Computing*, Cascais, Portugal, 2011, pp. 19:1–19:14.