UIE: User-centric Interference Estimation for Cloud Applications

Seyyed Ahmad Javadi, Sagar Mehra, Bharath Kumar Reddy Vangoor, Anshul Gandhi
Computer Science Department, Stony Brook University
{sjavadi, smehra, bvangoor, anshul}@cs.stonybrook.edu

***Abstract*—Interference is one of the key deterrents to cloud adoption, and is known to cause severe degradation in application performance, costing service providers in lost revenues. In this paper, we present UIE, a *user-centric* approach to detecting and, importantly, *estimating* the degree of interference experienced by user applications in the cloud. UIE employs queueing theory to model the impact of resource contention on application performance. By leveraging UIE, users can estimate the true amount of resources, including CPU, network, and I/O, allocated to their application at any given time, *without* any assistance from the cloud provider or hypervisor.**

I. Introduction

Cloud computing provides economical and elastic resources, such as VMs or storage, to users for their software deployment needs; the low resource cost is realized via multi-tenancy, which allows for physical resources to be shared among several users via virtualization. Unfortunately, multi-tenancy results in undesirable performance effects, the most severe of which is *interference*, which is caused by lack of effective performance isolation among users of colocated resources. Interference can result in significant performance degradation, increasing application response times by *factors of 5-20X* as suggested by prior work [1]–[3].

Existing work on addressing performance interference typically focuses on hypervisor-centric solutions. However, in public clouds, the cloud provider (who can monitor the hypervisors) does not communicate with users and is oblivious of user application requirements. Ideally, the solution should allow cloud users to detect *and* estimate the degree of interference experienced by their applications *without* requiring any assistance from the cloud provider or the colocated application owners.

Analyzing interference from a user's perspective is challenging because of the user's lack of visibility into colocated user applications [4]. Note that while *detecting* interference from within the VM is possible [1], [3], [5], it is much more difficult to *estimate the degree of interference*, which can be defined as the resource utilization of all the colocated VMs. This is especially true for non-CPU resources, such as disk I/O and network bandwidth, that are shared, not necessarily equally, among all colocated applications.

In this paper, we propose a user-centric technique, UIE, to estimate the degree of interference by *inferring* the utilization of resources due to colocated applications. UIE provides accurate estimations of per-device interference, including CPU, and the more challenging I/O and network, *without* requiring any additional instrumentation or information about colocated applications. UIE achieves this goal by monitoring application performance over time and developing analytical models that estimate actual resource allocation of the application. This information can then be exploited by the user to either proactively migrate their VMs or to initiate scale-out to avoid imminent SLO violations.

We evaluate UIE by accurately estimating the degree of interference for two popular cloud applications: (i) web servers (Apache [6]), and (ii) data processing (YARN). Our experimental results on an OpenStack cluster show that UIE can estimate interference with less than 10% error.

II. Experimental setup

We use an OpenStack Icehouse-based private cloud with three Dell C6100 physical machines, referred to *PMs* (each with two 6-core CPUs and 24 GB memory), as our experimental setup. All PMs are connected to a network switch via a 1Gb Ethernet cable. Our experiments reveal that the maximum achievable bandwidth is about 931 Mb/sec (we flood the network using a simple load generator and measure the peak observed bandwidth under various request rates). For disk, we use the `dd` linux tool to benchmark I/O bandwidth. Our experiments reveal that the maximum achievable write bandwidth for our PMs is about 140MB/sec.

We launch several VMs on this setup to deploy our applications. Unless otherwise specified, each VM has 4 vCPUs, 8GB memory, and 80GB disk space. We use Apache web server [6] and YARN (Hadoop2) as our target applications for performance interference.

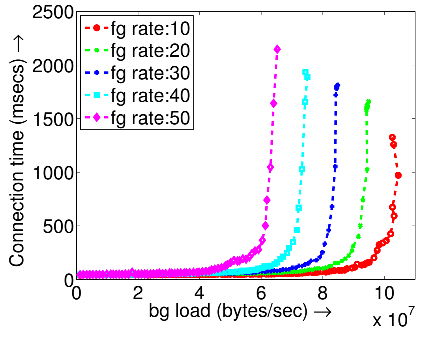
III. Performance interference

We now analyze interference using the experimental setup described above. We use benchmarks, referred to as **bg** (background), to create interference for our primary applications, Apache and YARN, referred to as **fg** (foreground).

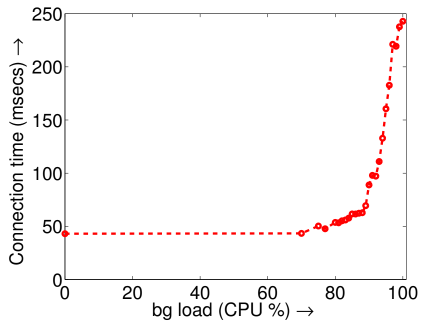
A. Apache Performance

We focus on network and CPU contention for the Apache setup, as those are the typical bottlenecks for Apache [7].

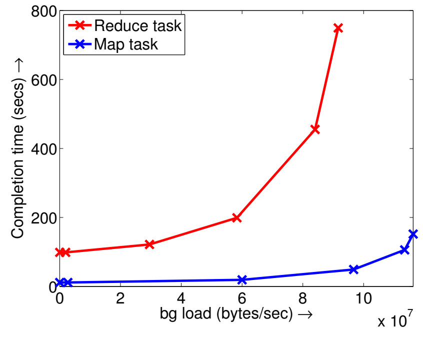
Network contention: For the Apache setup, we host a 1MB html file at a VM, VM_{fg}, running Apache, on host PM1. This file is retrieved via an httperf [8] client running on a different VM (and on a different PM, PM2). PM1 is used as



(a) Apache performance under network interference (using 1MB file).



(b) Apache performance under CPU interference.



(c) Terasort (YARN) performance under disk I/O interference.

Figure 1: Impact of interference on application performance.

the target physical machine which will experience resource contention. To create interference, we again use the httperf benchmark. Another VM, VM_{bg}, also on PM1, hosts several files of different sizes. These are retrieved by another httperf client outside of PM1 (specifically, on a VM on PM3). This client floods the link connecting PM1 to the network by sending a given rate of http requests to download html files from VM_{bg} on PM1. We monitor the connection time as seen by httperf for VM_{fg}. **Connection time** here is defined as the time between sending the http request from httperf and receiving the requested web page, in full, from VM_{fg}.

Figure 1a shows the observed fg connection time (for VM_{fg}) as a function of generated bg load, for a 1MB download file size and different fg request rates. The bg load on VM_{bg} (x-axis) is controlled via the bg request rate for a 1MB file. The results indicate that the fg connection time is not much affected by interference at low resource contention. However, once the contention is high, connection time rises rapidly. Results are qualitatively similar when the bg file size is changed to 2MB and 3MB. The rapid rise in connection time is to be expected as there is a finite available bandwidth (931 Mb/sec) that is being shared by VM_{fg} and VM_{bg}.

CPU contention: For CPU contention, we change VM_{fg} (on PM1) to now have only 2 vCPUs. To create sufficient bg CPU interference, considering the fact that our PMs have 12 physical cores, we launch 3 bg VMs on PM1, each of which has 4 vCPUs. The stress-ng tool [9] is used to create CPU load on these bg VMs. Note that, along with VM_{fg}, PM1 is now hosting VMs that require 14 vCPUs on its 12 physical cores.

Figure 1b shows the observed average fg connection time as a function of bg CPU usage, averaged across all three co-hosted bg VMs. In our setup, since the fg Apache is not performing any significant compute or processing operations, the connection time does not increase much with bg load, up to 80%. However, after this point, there is a steep increase in the connection time due to limited CPU resources (and oversubscription of these resources).

B. YARN Performance

We now study resource contention for the YARN setup. For YARN, our experiments reveal that network usage is transient, and not as significant as disk I/O and CPU usage. We thus focus on disk I/O and CPU interference.

Disk I/O contention: To evaluate disk I/O contention in YARN, we launch 3 VMs on PM1: slave-1, slave-2, and VM_{bg}. The Master VM is launched on a different PM (PM2). As the fg application, we launch Terasort (from the HiBench suite) using a 5GB input data set, which results in 42 Map tasks and 5 reduce tasks. To create I/O contention, the VM_{bg} uses stress-ng [9] to create 4 processes that continually write data to the (shared) disks. Each PM has 2 physical disks, set up in a RAID 1 configuration, and the replication factor of HDFS is set to 1.

Figure 1c shows the average Map and Reduce task completion times for Terasort as a function of the bg load. We control the bg load on VM_{bg} by varying the data write rate for the stress-ng benchmark. The results indicate that the fg performance (completion times, in the case of YARN) is significantly affected by resource contention at the disk. Recall, from Section II, that the maximum disk I/O bandwidth is about 140 MB/sec, which is being shared by VM_{fg} and VM_{bg}.

CPU contention: For CPU contention, we launch 7 VMs on PM1: slave-1 and 6 bg VMs. All of the PM1 VMs are configured with 4 vCPUs, 4GB memory, and 80GB disk, to allow CPU overcommit without violating the memory overcommit values set in OpenStack. As the fg application, we launch Kmeans (from the HiBench suite) using a roughly 4GB input data set, which results in 31 Map tasks and 1 reduce task. To create CPU contention, all 6 bg VMs use stress-ng [9] to create CPU load on PM1. Along with slave-1, PM1 is now hosting VMs that require 28 vCPUs on its 12 (oversubscribed) physical cores.

Our experimental results (omitted due to lack of space) show that the average Map and Reduce task completion times for Kmeans increase as a function of the average bg CPU usage.

IV. UIE

Our experimental results in the previous section show that interference can severely impact performance, often in a non-linear manner, as illustrated by the steeply rising fg connection/completion times at high bg loads. The motivation behind our UIE approach is to accurately predict this performance degradation *without* observing the bg application.

The key idea behind UIE is to model interference using concepts from *queueing theory*. Interference is essentially the result of contention at the shared physical resource. Queueing theory is well suited to model performance in such shared environments. Unfortunately, in this user-centric scenario, we *do not know* the value of all the system parameters, such as bg load. In order to apply performance modeling in such environments, we employ inference techniques, specifically multiple linear regression, to *estimate* the model parameters based on online observations.

A. Performance modeling

Consider the Apache experimental results for network contention in Section III-A. Here, the total network bandwidth, say B_{total}, is being shared among the fg and bg applications. The fg and bg bandwidth requirements, B_{fg} and B_{bg}, respectively, can be expressed as the product of request rate and file (download) size. That is, $B_{fg} = fg_{request\ rate} \times fg_{file\ size}$, and $B_{bg} = bg_{request\ rate} \times bg_{file\ size}$. Queueing theory [10] suggests that the mean connection time, T, in this case is inversely proportional to $(1-(B_{fg}+B_{bg})/B_{total})^{\alpha}$, for some parameter α. In the case of exponential job sizes, $\alpha = 1$, but α can take higher values for other distributions and service disciplines [10]. Based on this approximation, we model connection time as:

$$T = \theta_0 + \frac{\theta_1}{\left(1 - \frac{B_{fg} + B_{bg}}{B_{total}}\right)} + \frac{\theta_2}{\left(1 - \frac{B_{fg} + B_{bg}}{B_{total}}\right)^2}, \quad (1)$$

where the θ variables are coefficients. We find that using two exponent terms (with coefficients θ_1 and θ_2), along with an intercept term (with coefficient θ_0), suffices for modeling performance. Note that B_{total} in Eq. (1) is a constant that depends on the network configuration. B_{fg} and T can be measured by the fg user. B_{bg}, unfortunately, cannot be measured by the fg user as it depends on the bg application, which *cannot* be observed. Most of the prior work on interference detection assumes that the bg application and its behavior can be measured. From the fg user's perspective, this is infeasible. Thus, the challenge now is to determine the θ coefficients *without* assuming knowledge of B_{bg}.

B. Inference

Interference in cloud environments is usually dynamic in nature. Thus, if we observe the fg performance over a long time interval, there will be instances where the bg load is low. We leverage this notion, which was also observed in prior work [1], to obtain (T, B_{fg}) pairs which correspond

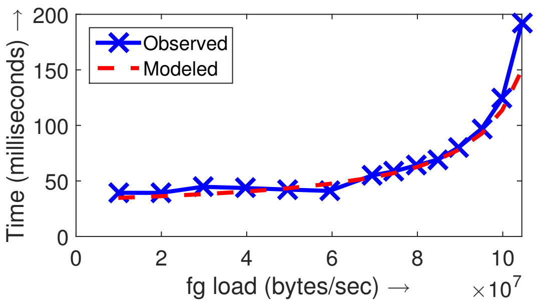


Figure 2: Regression modeling for Apache performance with no bg load. Modeling error is about 6%.

to low B_{bg} values. Our results in Section III show that when the bg load is low, the impact on fg performance is negligible; we thus set $B_{bg} = 0$ for these cases. We now use these collected (T, B_{fg}) pairs and perform multiple linear regression on Eq. (1) with $B_{bg} = 0$ to determine the θ coefficient values. Figure 2 shows our regression results. We see that our estimated connection time, T, values (dashed red line) accurately track the observed (solid blue line) values. The modeling error on training data is about 6%; regression coefficients are $\theta_0 = 20.3$, $\theta_1 = 13.2$, and $\theta_2 \approx 0$.

C. Interference estimation

We now employ UIE to estimate interference (B_{bg}) using our performance model. Based on the observed connection time, T, and the monitored fg load, B_{fg}, we solve Eq. (1) to determine bg load, B_{bg}, by substituting for B_{total} and the θ coefficient values obtained from the above regression analysis. UIE uses this approach to estimate bg load for all the experiments in Section III, as discussed below. Note that training data used in Section IV-B is different from the experimental test data discussed below.

D. Results

Apache network contention: The estimation (or test) error for UIE under Apache's network contention experiments (Figure 1a) is an *impressively low 5.4%*. This is illustrated in Figure 3 which plots the actual and estimated bg load for the experiments in Figure 1a. Since interference does not impact Apache's performance at low loads, we consider UIE's estimation error for those data points where the impact of bg load is non-negligible; this corresponds to roughly 50% of the data (around 315 experiments) in Figure 1a.

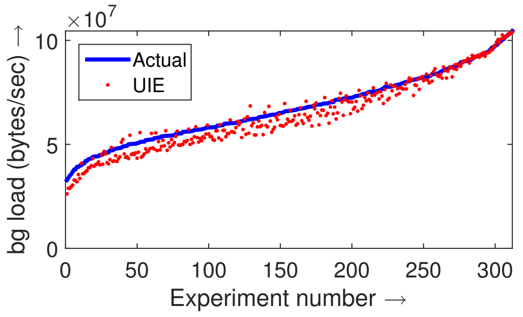


Figure 3: Estimated bg load using UIE for the data in Figure 1a. Estimation error is about 5.4%.

Application	Interference	Modeling Accuracy
Apache	Network	94.0%
Apache	CPU	91.1%
YARN (Terasort Map)	disk I/O	91.9%
YARN (Terasort Reduce)	disk I/O	94.9%
YARN (Kmeans Map)	CPU	94.9%
YARN (Kmeans Reduce)	CPU	95.0%

Table I: Modeling accuracy for all applications.

It is interesting to note, from Figure 3, that the difference in actual and estimated bg load is lower for higher bg loads. This can be explained as follows. Looking at the two curves in Figure 2, we see that a constant change in the y-value, say Δy, corresponds to a much smaller change in the x-value, say Δx, at higher loads than it does at lower loads. This is because of the increasing nature of the connection time curves. Fortunately, this behavior results in high estimation accuracy for the regime where bg load, or interference, is high. This is often the regime that practitioners are interested in, as it causes the most severe performance degradation.

Apache CPU contention: UIE can be applied to the case of Apache CPU contention (Figure 1b) in a similar manner. For CPU contention, we use cpu % as the load unit instead of bytes/sec. Thus, Eq. (1) now takes the form:

$$T = \theta_0 + \frac{\theta_1}{\left(1 - \frac{CPU_{fg}\ +\ CPU_{bg}}{100}\right)} + \frac{\theta_2}{\left(1 - \frac{CPU_{fg}\ +\ CPU_{bg}}{100}\right)^2}, \quad (2)$$

where CPU$_{fg}$ and CPU$_{bg}$ are average CPU usage percentages in the range $[0, 100]$. Note that CPU$_{fg}$ + CPU$_{bg}$ < 100 as these correspond to VMs on the same PM. The modeling error in this case is 8.9% (listed in Table I).

YARN disk I/O contention: We employ the same non-linear relationship as in Eq. (1) to model disk I/O interference for the Map and Reduce tasks under YARN. The modeling error for Terasort experiments (Figure 1c) is 8.1% and 5.1% for Map and Reduce task completion times, respectively.

YARN CPU contention: Using the same non-linear relationship as in Eq. (2) to model CPU interference, the modeling error for Kmeans experiments is 5.1% and 5.0% for Map and Reduce task completion times, respectively.

V. Related Work

Casale et al. [5] propose a user-centric technique to detect CPU contention by analyzing the CPU steal metric (time spent waiting for the hypervisor). However, this technique only considers CPU-level interference. IC^2 [3] employs decision trees using VM-level statistics to detect interference. IC^2 focuses only on cache-intensive interference, and the decision tree classifier requires a substantial amount of time (nearly 30 hours) for offline training. CPI^2 [11] employs statistical approaches to analyze an application's cycles per instruction (CPI) metric in order to detect and mitigate processor resources interference. However, recent work [1], [2] has shown that CPI is not always a good metric for interference detection as it is representative of throughput and not latency. CRE [1] makes use of collaborative filtering to detect interference in web services by monitoring the response times. While CRE can accurately detect interference in most cases, it does not estimate the amount of interference. By contrast, UIE detects *and* estimates interference.

VI. Conclusion

We presented UIE, a *user-centric* approach for not only detecting, but also *estimating*, the amount of interference experienced by cloud applications. The fact that UIE is user-centric allows users to analyze interference and take actions without requiring any assistance from the hypervisor. UIE accurately detects the amount of physical resources, including network and I/O bandwidth, and CPU, allocated to a user application, thus supporting several classes of applications including network-intensive web services and I/O-intensive data processing frameworks. UIE is the first step in our eventual goal of providing interference-aware SLO compliance for cloud applications. As next steps, we will leverage UIE to provide performance management to cloud-deployed applications in interference-ridden environments.

Acknowledgment

This research was supported by NSF CSR Grant 1464151.

References

[1] Y. Amannejad, D. Krishnamurthy, and B. Far, "Detecting Performance Interference in Cloud-based Web Services," in *Proceedings of IFIP/IEEE International Symposium on Integrated Network Management*, 2015, pp. 423–431.

[2] J. Mukherjee, D. Krishnamurthy, and J. Rolia, "Resource Contention Detection in Virtualized Environments," *IEEE Transactions on Network and Service Management*, vol. 12, no. 2, pp. 217–231, 2015.

[3] 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*, 2014, pp. 277–288.

[4] A. Gandhi, P. Dube, A. Karve, A. Kochut, and H. Ellanti, "The Unobservability Problem in Clouds," in *Proceedings of the 2015 IEEE International Conference on Cloud and Autonomic Computing*, Cambridge, MA, USA, 2015.

[5] G. Casale, C. Ragusa, and P. Parpas, "A Feasibility Study of Host-level Contention Detection by Guest Virtual Machines," in *Proceedings of the 5th International Conference on Cloud Computing Technology and Science*, 2013, pp. 152–157.

[6] "The Apache HTTP Server Project," See http://httpd.apache.org.

[7] Y. Hu, A. Nanda, and Q. Yang, "Measurement, Analysis and Performance Improvement of the Apache Web Server," in *Proceedings of the 18th International Performance, Computing and Communications Conference*, Scottsdale, AZ, USA, 1999, pp. 261–267.

[8] D. Mosberger and T. Jin, "httperf—A Tool for Measuring Web Server Performance," *ACM Sigmetrics: Performance Evaluation Review*, vol. 26, pp. 31–37, 1998.

[9] "Stress-ng," http://kernel.ubuntu.com/ cking/stress-ng.

[10] M. Harchol-Balter, *Performance Modeling and Design of Computer Systems*. Cambridge University Press, 2013.

[11] 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*, 2013, pp. 379–391.