Intelligent, Performance Interference-aware Resource Management for IoT Cloud Backends

Faruk Caglar*, Shashank Shekhar†, Aniruddha Gokhale† and Xenofon Koutsoukos†
*Dept of Computer Eng, Meliksah University, Kayseri, TURKEY
†Dept of EECS, Vanderbilt University, Nashville, TN, USA.
Email: fcaglar@meliksah.edu.tr and {shashank.shekhar,a.gokhale,xenofon.koutsoukos}@vanderbilt.edu

***Abstract*—Emerging Internet of Things (IoT) applications often demonstrate unpredictable Big Data processing workloads at the cloud backends making it hard for cloud service providers (CSPs) to employ existing resource overbooking schemes effectively. Ad hoc approaches to resource overbooking can lead to performance interference among the virtual machines (VMs) hosted on the physical resources causing performance unpredictability for VM-hosted performance-sensitive IoT applications. Balancing these conflicting needs requires an intelligent strategy for hosting applications such that the performance interference effects are minimized while still allowing resource overbooking. Such a strategy must be online because application workloads may change at run time. To address these challenges, this paper presents iSensitive, which is an intelligent, performance interference-aware resource management scheme for IoT cloud backends. iSensitive first classifies the VMs based on their historic mean CPU, memory, and network usage features. Subsequently, it learns the desired VM patterns of collocating the classified VMs by employing machine learning techniques. These extracted patterns document the lowest performance interference level on the specified host machines making them amenable to hosting performance-sensitive applications while still allowing resource overbooking. Our approach is validated by emulating a publicly available usage trace of a large data center owned by Google and benchmark tools running real-world applications. Experimental results evaluating iSensitive illustrates its advantages in deploying VMs to aptly-suited host machine compared to traditional schemes, such as first-fit bin packing.**

***Keywords*—Performance interference, cloud, resource monitoring, KVM hypervisor**

I. Introduction

Cloud service providers (CSPs) always look to maintaining high resource utilization of cloud computing resources while keeping energy costs low and increasing revenues. To that end they often resort to resource overbooking techniques [1], [2], [3], [4]. The idea behind resource overbooking is to commit more resources, such as CPU and memory, than are actually available on the physical host machines. Statistical multiplexing is the key intuition behind the overbooking strategy which exploits the fact that users of traditional cloud-hosted applications often request more resources than what their applications actually need and and that too almost never at the same time thereby providing an opportunity to the cloud provider to overbook. Contemporary hypervisors, such as Xen [5], KVM [6], and VMware ESX Server [7], provide the necessary support to make overbooking feasible to implement in practice.

Performance-sensitive Internet of Things (IoT) applications, such as the Industrial Internet applications, are now increasingly hosted in the cloud to perform Big Data analytics on real-time sensor streams. A key trait of this class of applications is its unpredictable arrival pattern of streaming data, which makes it hard to bound the number of requested resources ahead of time. Consequently, statistical multiplexing may not be as effective and hence resource overbooking may adversely impact application performance because multiple virtual machines (VMs) collocated due to resource overbooking can trigger significant performance interference [8], [9], [10], [11], [12], [13].

Performance isolation is an important consideration in this context, however, there is no perfect solution to provide a virtualized environment where there is no performance interference between VMs [14], [15]. Although there exists prior work on performance isolation among VMs collocated in an overbooked host machine [13], it is still a challenging task to monitor and consider performance interference for VM placement and shield the VMs from its neighbors due to the nature of resource sharing, resource overbooking practices employed, configuration of the underlying scheduling mechanisms, other co-hosted neighbor applications, and the fluctuating workload characteristics in the cloud. Therefore, an application running in one VM might still impact the performance of another application running in a separate VM on the same host machine. Specifically, network- and compute-intensive applications might be adversely impacted.

Addressing the performance interference challenges that stem from resource overbooking while satisfying the performance and response-time requirements of IoT applications will require effective trade-offs in the placement of VMs on host machines by carefully considering the actual workload characteristics of the VMs. Due to the changing dynamics of the IoT application workloads on the VMs and also because VMs often tend to migrate from one physical machine to another for a variety of reasons, traditional offline heuristics such as bin packing will not be applicable for interference-aware VM placement in cloud computing. Consequently, we have focused on a system architecture that considers monitoring of performance interference variations in addition to the VM placement strategy by incorporating both the performance interference effects and the workload characteristics of the collocated VMs on the target host machine.

Our prior work to date involving VM-based cloud resource management has considered power-performance trade-offs [16], physical server consolidation using VM overbook-

*Work conducted by first author while at Vanderbilt University

ing [1], and auto tuning of the hypervisor parameters [**?**] but none of these works account for dynamic resource management that considers performance interference between collocated VMs. This is a critical need for IoT applications hosted in the cloud backends.

This paper presents a combined online performance interference monitoring and VM placement technique based on a machine learning approach, which is codified in a middleware called *iSensitive*. Our approach first analyzes the performance differences in the *Base*, *Non-Overbooked*, and *Overbooked* environments and compares the impact of resource utilization on performance interference. Using these insights we then use machine learning to learn about the desired VM placement patterns and encode these patterns in the iSensitive middleware enabling it to make runtime VM placement decisions. Our recent work [17] described preliminary work in this regard. For this paper, we made substantial enhancements to our preliminary work. For instance, we completely redesigned our machine learning-based model learning approach. Moreover, we provide extensive empirical validation of our approach.

Specifically, we make the following research contributions:

- We provide insights into how application performance and resource utilization are impacted by performance interference (See Section III-A).
- We present a machine learning-based performance interference-aware virtual machine placement technique and performance monitoring middleware called iSensitive that can be deployed in resource-overbooked IoT cloud backends (See Section III).
- We show experimental results to validate our claims that iSensitive can find the aptly suited host machine that minimizes performance interference while still allowing overbooking (See Section IV).

The rest of this paper is organized as follows: Section II describes relevant related work comparing it with iSensitive; Section III presents the iSensitive approach; Section IV shows empirical results validating the iSensitive approach, and Section V presents concluding remarks alluding to future work.

II. Related Work

This section presents related work on VM placement and techniques that address performance interference, and compares it with our iSensitive approach.

Novakovic et al. [9] propose DeepDive which transparently identifies and manages performance interference. DeepDive comprises three subsystems: (1) warning system, (2) interference analyzer, and (3) placement manager. The warning system conducts interference analysis at a lesser reliability whereas the interference analyzer is employed and starts measurements when the warning system suspects a VM is causing performance interference. The placement manager migrates the VM to another physical host by first mimicking the behavior of the VM being migrated using a synthetic benchmark. This benchmark is executed on all target hosts for a short period of time before the migration takes place to see whether interference will occur again on the target host. The authors mention this as a costly process due to cloning the VM on a separate machine and running it in an isolated environment. The experimental results show how throughput and latency of the Cassandra NoSQL database is impacted in the Amazon EC2 environment. The measured performance of this application on identical EC2 VMs indicates the performance impacts of collocated VMs. iSensitive differs from DeepDive in that it first learns the best collocated VM patterns and conducts the placement decision based on it without trying to run a mimicked VM on each machines concurrently. Additionally, modeling the exact behavior of an application is a challenging task where a variety of applications are hosted in cloud environment.

In [18], the performance interference effects of background workloads (i.e. collocated VMs) on the same host machine are analyzed by evaluating the performance of latency-sensitive online games and gaming servers. It provides experimental results of network jitter and throughput fluctuations in the Amazon EC2 environment. Q-Clouds [10] is a QoS-aware framework to manage performance interference in the cloud. Q-Clouds provisions additional resources to alleviate performance interference. It applies an online feedback mechanism to build a model for capturing interference interactions and uses it for resource management. Moreover, the system employs a staging server to determine the resource requirements and leaves a head room, *i.e.*, slack resource, for performance management. Q-Clouds allows specifying different levels of QoS, known as Q-states, to increase the resource utilization. Despite these ideas, the slack resources still lead to under utilization of the server resources. Frequent resource allocations due to the feedback mechanism can also cause performance overhead for the hypervisor.

Zhu et al. [19] proposed an interference model which predicts application QoS. It considers time-variant interdependence between the different levels of resource contention. The authors develop a resource usage profile as a vector of matrices for different performance metrics and then apply a consolidation algorithm to accommodate applications to minimize interference and achieve QoS. We believe that this work focuses on developing simplistic models for complex resource utilization relationships, whereas iSensitive uses k-means clustering to group the VMs in different classes to capture the complex relationships and then apply machine learning to determine performance interference.

TRACON [20] is a task and resource allocation framework for data-intensive applications. It develops three interference prediction models: weighted mean method model, linear model and non-linear model using statistical machine learning for reasoning. It then employs an interference-aware scheduler for reducing performance interference. The focus of this technique is network I/O-intensive applications whereas our approach focuses on a variety of application types.

Kambadur et al. [21], studied the methodology and several complexities behind measuring performance interference in data centers stemming from resource contention and proposed a new technique based on finding the performance interference between base application and co-runners on the same machine. In this work, the authors have measured the performance interference to identify interference relationships and classes but have not demonstrated its application. We have leveraged some of the insights and parameters from this work in iSensitive.

Moreno et al. [22] proposed a method for interference-aware virtual machine placement by analyzing its impact on energy efficiency in data centers. The combined interference score utilized in this work requires the knowledge of maximum throughput of each workload running on a host machine when mixed with other workload types. This might require employing some applications to reside on VMs to populate this information from the workload which may result in high overhead when a host runs multiple different types of workloads. In contrast, iSensitive discovers and extracts the best VM patterns by employing machine learning algorithms to learn performance interference levels. iSensitive also differs from this work based on its VM classification features that uses network utilization.

Maji et al. [23], propose an approach named IC2, which handles the performance interference problem from a different perspective. IC2 mitigates the interference by application (e.g. web server or database) reconfiguration through a machine learning-based technique. IC2 handles interference at the application level rather than hardware level. IC2 targets the Apache web server and PHP runtime configuration parameters, and considers only three key parameters affecting application performance. IC2 also takes application-level indicators to detect the interference where hardware-level parameters are not accessible. IC2 improves the application response time to a certain degree. Even though application reconfiguration appears to be an attractive strategy to mitigate the interference, the complexity of reconfiguration increases as the number of application types hosted in the cloud increases.

III. Design and Implementation of iSensitive

We now present iSensitive providing a rationale for the design decisions. iSensitive is a cloud backend resource management solution suited for IoT applications that addresses performance interference concerns in the cloud while allowing resource overbooking. The iSensitive solution is premised on the hypothesis that resource overbooking causes performance interference and hence degrades application performance.

A. Problem Context: Relating Performance Interference to Application Performance and Resource Usage

To justify our premise, we first provide insights on how performance interference stems from resource overbooking resulting in contention for resources, which in turn adversely impacts the application performance and resource utilization. Our hypothesis was validated on a cloud platform that hosts VMs managed by the KVM hypervisor.

The experiments were conducted under three distinct setups named *Base*, *Non-Overbooked*, and *Overbooked*. The workload in the VMs in these setups are generated through applications randomly picked from the *phoronix test suite* (http://www.phoronix-test-suite.com/. To quote their web site: "The Phoronix Test Suite is the most comprehensive testing and benchmarking platform available that provides an extensible framework for which new tests can be easily added. The software is designed to effectively carry out both qualitative and quantitative benchmarks in a clean, reproducible, and easy-to-use manner." These benchmarking applications were representative of various dynamically generated IoT application loads in the VMs. *Virt-top* and *jMeter* tools were utilized to log various resource usage and performance metrics. The generated data were also adequate to make decisions.

- **Base**: In this setup, only one VM having 1 virtual CPU (vCPU) and 512 MB of RAM running an Apache Web Server resides on the host machine. Web requests from 50 concurrent users are posted to the web server from a separate host machine located in the same network cluster.
- **Non-Overbooked**: In non-overbooked setup, the requested resources that are available on the host machine are equal to the available resources. We created 12 VMs each of which comprises 1 vCPU and 512 MB of memory. Each VM in this setup hosts a benchmarking application randomly picked from the *phoronix test suite* except for the one VM that runs the Apache Web Server. The benchmarking application is run continuously in the VM. One VM out of 12 handled the web requests initiated from 50 concurrent users from a separate host machine in the same network cluster. This setup is intended to mimic the continuous stream of IoT sensor information reaching the cloud for analytics.
- **Overbooked**: The overbooking ratio used for the CPU resources is 2 which means that the requested CPU resource is two times greater than that is available on the host machine. Thus, we created 24 VMs each one having 1 vCPU and 512 MB of memory. Again, as was the case in the non-overbooked scenario, each VM comprises a benchmarking application which was randomly selected from the *phoronix test suite*. One VM out of the 24 managed the web requests initiated from 50 concurrent users.

1) Impact of Performance Interference on Application Performance: Figure 1 shows the comparison of web server throughput in the *Base*, *Non-Overbooked*, and *Overbooked* scenarios. As can be seen, the application throughput performance in the *Base*, *Non-Overbooked*, and *Overbooked* scenarios are 179, 128, and 88, respectively, requests per second handled successfully. As expected, *Base* provides the best application performance since there is no other VM contending for the resources. The *Non-overbooked* environment is not as good as the *Base*, but better than the *Overbooked* case because there are more VMs contending for available resources which triggers more performance interference between VMs in the *Overbooked* scenario.

Figure 2 compares the response time for the web server in the *Base*, *Non-Overbooked*, and *Overbooked* test setups. As can be seen, the web server's response time illustrates a similar trend for the *Base*, *Non-Overbooked*, and *Overbooked* environments by providing best, good, and poor performance, respectively. The performance of the web server in the *Overbooked* has significant jitter than *Non-Overbooked*. The *Base* scenario does not show jitter and response time is continuously steady. These results are also supported by the results in Figure 1.

Figure 3 shows the comparison of web server response time percentiles in the *Base*, *Non-Overbooked*, and *Overbooked*

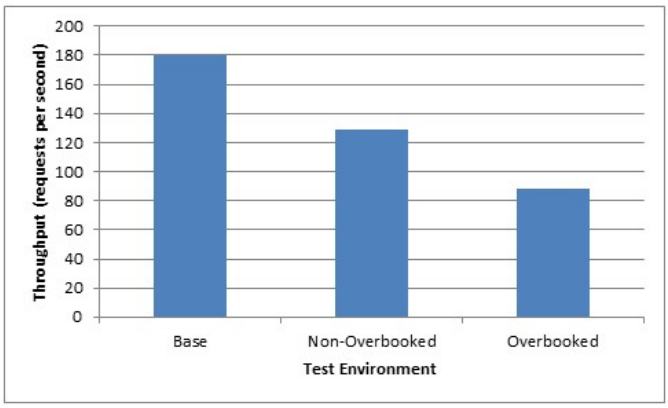

Fig. 1: Comparison of Web Server Throughput in Base, Non-Overbooked, and Overbooked Scenarios

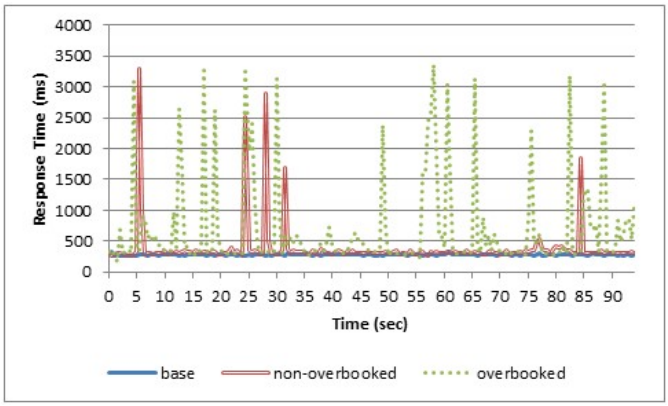

Fig. 2: Comparison of Web Server Response Time in Base, Non-Overbooked, and Overbooked Environments

test setups. For all of the percentiles in Figure 3, *Base* provides the best, then *Non-Overbooked*, and then *Overbooked*. Additionally, there is no contradiction with the results from Figures 1 and 2.

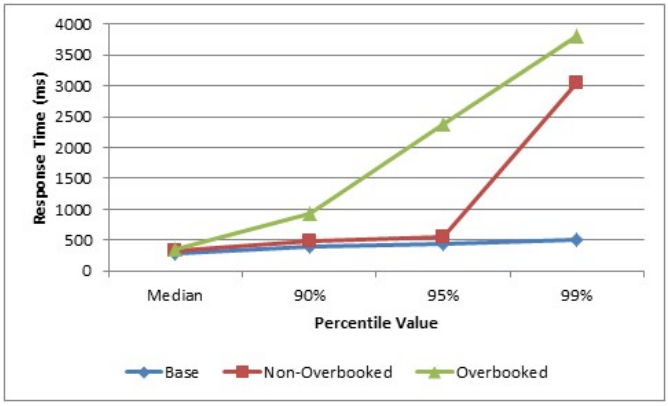

Fig. 3: Comparison of Web Server Response Time Percentiles in Base, Non-Overbooked, and Overbooked Environments

2) Impact of Performance Interference on Resource Utilization Performance: The CPU utilization of the VM hosting the web server is depicted in Figure 4. The root cause of the performance degradation between the three different setups is clearly seen here in that the CPU utilization in *Non-Overbooked* and *Overbooked* environments is not as good as that of the *Base*. Even though the physical resources in the *Non-Overbooked* environment were not overbooked, there may still be some instant spikes due to the hypervisor overhead or resource contention. Moreover, the jitter in the *Overbooked* scenario is considerably high. Even though, CPU resources on the host machine were not 100% utilized in both *Non-Overbooked* and *Overbooked*, the performance interference was unavoidable.

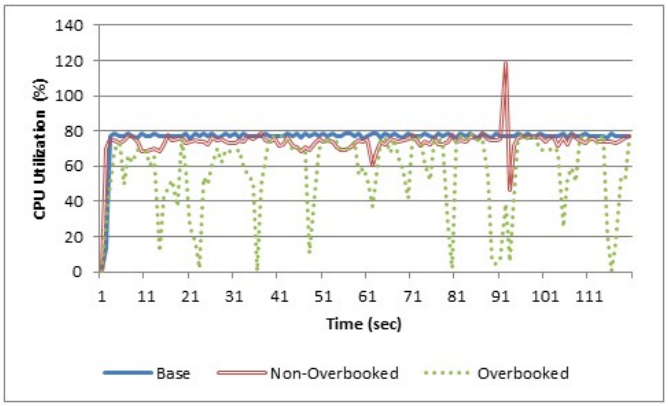

Fig. 4: Comparison of Web Server CPU Utilization in Base, Non-Overbooked, and Overbooked Environments

B. Problem Statement

Resource contention and hence performance interference is unavoidable in virtualized environments due to the nature of resource sharing. We have validated this hypothesis empirically where we analyzed how performance interference stems from resource overbooking and how contention impacts the application performance running in the VMs.

Consequently, it is imperative that performance interference be considered as a first class issue particularly for resource-overbooked environments that aim to host IoT applications, which is the focus of this work. Moreover, since virtual machines in the cloud can migrate from one host machine to another one in a data center and between data centers because of reasons such as consolidation, load balancing, thermal effects, and noisy neighbors, VM migration must also be considered as a key issue.

Therefore, to mitigate the performance interference between collocated VMs, the resource usage profiles of VMs hosting applications must be examined and the VM placement decision in the cloud must incorporate this information in identifying the collocated VMs. Additionally, VM profiling should continue at run-time because of the fact that workloads might change dynamically.

C. Intuition behind the iSensitive Approach

Recall that IoT application workloads on cloud backends do not have predictable arrival patterns. Thus, collocating VMs that perform IoT data analytics with other VMs in an

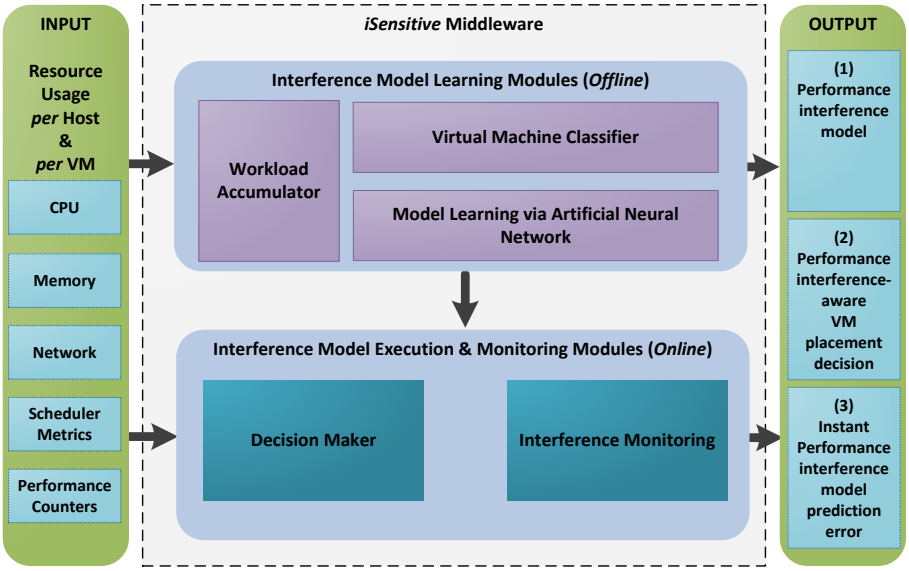

Fig. 5: Conceptual Design of iSensitive illustrating Input, Output, and System of Interest

arbitrary manner focusing only on resource overbooking is not acceptable since it can lead to performance interferences and hence degraded performance for the IoT applications. Thus, we need an approach for predicting the type of incoming workloads thereby enabling effective VM collocation placement decisions.

To that end, in this paper we present a model predictive approach to address performance interference issues at runtime. The intuition behind our approach is as follows: a predictive approach needs to learn from historical data. Thus, our first step requires obtaining real-world historical resource usage data for VMs as well as host machines in the cloud so as to generate a system performance interference model. High quality and fine-grained historical data is crucial to gain a sense of the resource usage patterns and how they change over time. Based on this analysis, we can then create an accurate model of the system that can be employed at runtime.

In the second step, which is an online step, the learned model from the first step is used to make decisions on VM placement that will minimize the performance interference in the resulting deployment. Our approach is generic enough and can be applied by cloud service providers in their data centers for their expected workloads and hardware platform.

Figure 5 shows the algorithmic design and building blocks of our framework called iSensitive that adopts the solution approach described above. As shown, iSensitive comprises two distinct modules: (1) Interference Model Learning Module (*offline phase*), and (2) Interference Model Execution and Monitoring Module (*online phase*). The Interference Model Learning Module in turn comprises three main components: (1) Virtual Machine Classifier, (2) Model Learning via Artificial Neural Network, and (3) Workload Accumulator which presently feeds from a Synthetic Workload Generator. The Interference Model Execution and Monitoring Module consists of two primary components: (1) Decision Maker, and (2) Interference Monitoring.

D. Gist of the Two Phases of iSensitive

Since resource utilization is a key indicator of performance interference as seen from Figure 4, iSensitive utilizes different resource usage metrics, such as CPU usage, memory usage, network I/O usage, internal scheduler metrics, hardware and kernel-level performance counters for VMs and physical host as input to the system. These metrics are retrieved with the help of (1) *perf,* performance analyzing tool in Linux, (2) *mpstat,* Linux command for processor related statistics, and (3) *libvirt,* toolkit to interact with the underlying virtualization system. The *virtual machine classifier* clusters VMs into similar sets of objects by employing the k-means algorithm and the silhouette method. These classes of VMs are then used by the *artificial neural network* to extract the "best collocated VM patterns", which are those patterns that lead to minimal performance interference on the host machines. In other words, a performance interference model of a host machine is generated.

After the neural network is trained, the *decision maker* is employed to find the aptly suited host machine having the minimal performance interference by utilizing the trained model. *Interference monitoring* is responsible for comparing the actual performance interference value and its predicted value. If the difference is greater than a threshold value, then that collocation pattern is saved for future model refinements.

Note that for our work, we have assumed that the physical host machines in the cloud data center are homogeneous and therefore a model generated for one host machine is applicable to all other physical hosts. If a data center comprises heterogeneous machine types, then performance interference models for each different host machine type must be created.

TABLE I: Benchmark Applications Utilized by iSensitive

Test Suite Name	Application Name	Application Description	Resource Intensiveness
Phoronix Test Suite	pts/build-apache	Timed Apache Compilation	Processor
Phoronix Test Suite	pts/compress-gzip	Gzip Compression	Processor
Phoronix Test Suite	pts/compress-pbzip2	Parallel BZIP2 Compression	Processor
Phoronix Test Suite	pts/espeak	eSpeak Speech Engine	Processor
Phoronix Test Suite	pts/n-queens	N-Queens	Processor
Phoronix Test Suite	pts/openssl	OpenSSL	Processor
Phoronix Test Suite	pts/tachyon	Tachyon	Processor
Phoronix Test Suite	pts/tscp	TSCP	Processor
Phoronix Test Suite	pts/stresscpu2	StressCPU2 Stress-Test	Processor
Phoronix Test Suite	pts/sample-program	Sample PI program	Processor
Phoronix Test Suite	pts/ramspeed	RAMspeed SMP	Memory
Phoronix Test Suite	pts/stream	Stream	Memory
Netperf	TCP_STREAM	TCP Stream Performance	Network
Netperf	TCP_RR	TCP Request Response	Network
Netperf	UDP_STREAM	UDP Stream Performance	Network
Httperf	TCP	Web Workload Generator	Network
Sysbench	OLTP	Database Server Performance	Disk

E. iSensitive Offline Phase

The details of the offline phase of iSensitive are presented below.

1) Workload Accumulator: Recall that to produce a predictive model of our system, our system needs to be trained using raw data and classification. Thus, our first objective is to obtain such a raw data. One of the key components supplied by iSensitive is a workload accumulator to enable collection and storage of performance data. Presently, it feeds from a python-based synthetic workload generator that communicates with a cloud manager to instantiate, deploy, start and destroy virtual machines. It imitates the lifecycles of real-world VMs. However, in an actual deployment, the data will be coming from the hosts comprising the data center.

We have exploited the VM lifecycle events (e.g., create, destroy, migrate) and their resource configurations (e.g., number of virtual CPUs) from a real-world trace made available in the Google Cluster Trace [24]. To generate a training data set as realistic as possible to the real-world workload in the cloud data center, we mimicked the VM event data of five randomly chosen hosts from the Google Trace with ids 2790227930, 2113205802, 4550520892, 257335557, 317488481. We did not use more number of hosts for the VM lifecycle events due to the effort and scale needed in setting up experiments to generate the training sets.

Since the details of applications and their types running in the VMs of the Google Trace are not provided by Google, we could only exploit the lifecycle events and their resource configurations. To produce the right mix of different types of application workloads, therefore, we created a test suite using some of the popular benchmarking tools and real world applications stressing different aspects of a system as shown in Table I. For every VM, the tool randomly picks a test from the uniform distribution of tests in the suite and executes it. iSensitive's monitoring tools collect the performance metrics and generate the training data set.

2) Virtual Machine Classifier: Once the raw training data is created, the virtual machine classifier component clusters the VMs based on their CPU, memory, and network usage by using *k-means clustering* [25]. k-means is an unsupervised learning algorithm that is used to classify the VMs in different classes based on their resource usage profiles. It provides good results with large datasets such as the one used in our approach. Note that we do not consider disk intensive applications in this work. Therefore, these three main resource usage metrics are utilized for profiling VMs.

To decide the best number of clusters, the *Silhouette* method [26] is employed for the cluster data we utilize. The Silhouette method fits well with the k-means clustered data and is hence employed in our approach to analyze the VM clusters. The higher the silhouette value is, the better the classification is. The best cluster number for our raw data set is found to be 5 with a maximum mean silhouette value of 0.6560 over other cluster numbers. The resulting cluster center points found by the virtual machine classifier component is shown in Table II.

TABLE II: Identified Cluster Center Points for each Cluster

Cluster Number	CPU (%)	Memory (%)	Network IO (MBps)
Cluster 1 (C1)	12.83	44.60	1.04
Cluster 2 (C2)	199.03	27.54	9.19
Cluster 3 (C3)	76.29	42.75	1.46
Cluster 4 (C4)	128.05	17.07	234.06
Cluster 5 (C5)	104.08	36.68	121.48

3) Model Learning via Artificial Neural Network (offline phase): iSensitive relies on the historical data to model and capture the relationships between input and output parameters to discover the patterns of VM combinations and the resulting degree of performance interference. To capture the non-linear relationships between performance interference among the VMs and the large set of input factors for various classes of VMs, we have applied the back propagation-based artificial neural network (ANN) [27]. It is a supervised machine learning technique used to predict the performance interference, which is otherwise difficult to estimate in our complex model.

Concretely, the ANN is trained to capture the relationships on how the different types and numbers of VMs of the

same cluster found by the virtual machine classifier impact performance interference. The input parameters for the ANN are as follows:

N_1 = Total number of VMs of *Class 1*

N_2 = Total number of VMs of *Class 2*

N_3 = Total number of VMs of *Class 3*

N_4 = Total number of VMs of *Class 4*

N_5 = Total number of VMs of *Class 5*

C = CPU overbooking ratio

PIL = Performance Interference Level

The reason to choose the number of VMs of each class is to capture the relationships between the different VM combinations along with host machine CPU overbooking ratio and discover the regularities in how these patterns affect the performance interference level (PIL) on a host machine.

We modeled the performance interference level as the sum of cache miss ratio, scheduler waiting time, scheduler io waiting time, and guest cpu usage percentages as follows.

PIL = *Cache Miss Ratio + Scheduler Wait Time % + Scheduler IO Wait Time % + Guest %*

The metrics in the performance interference model are some of the significant metrics capturing the contention at shared resources that might cause crucial performance degradation.

- **Cache Miss Ratio**: Represents the ratio of total system-wide last-level cache (LLC) misses to total number of retired instructions. It captures the contention occurring at the LLC cache and is a promising metric to model performance interference of memory and cache-intensive applications. The value is represented as per hundred instructions in order not to dominate the other values.
- **Scheduler Wait Time %**: Represents the waiting time incurred at the scheduler's run queue which means that a VM is not able to access the physical CPU even though it is in the runnable state due to the CPU contention and causes increased latencies. The scheduler waiting time value may be very high for resource-overbooked environments. Therefore, this is also a promising metric capturing interference occurring at the scheduler.
- **Scheduler IO Wait Time %**: Represents the waiting time incurred due to the IO operations. The VM is in the idle mode while the system is waiting for an outstanding IO operation. This metric helps to capture the contention for IO-bound applications and allows the to incorporate IO-level interference into model.
- **Guest %**: Expressed as the percentage of CPU time spent by all the VMs on the host machine. This is also an important metric to capture how busy are the CPU resources to serve the VMs. Thus, a less busy host machine with all the guests will ultimately have less contention at the CPU-level.

For a finer granularity in the performance interference model, it is better to continue to update the trained model at run-time. This makes the performance interference model much more accurate right after an unforeseen workload pattern is experienced. The online model learning part of this work needs to be addressed in a future work.

F. iSensitive Online Phase

The details of the online phase of iSensitive are presented below.

1) Decision Maker: When a VM placement request is made or if a VM must be migrated, the decision maker component is responsible to iterate over all the host machines in the cluster, run the trained ANN, and return the host machine info which will provide the lowest performance interference level. The VM can then be placed in the machine despite the cloud service provider utilizing overbooking strategies.

2) Interference Monitoring: When it is enabled, the interference monitoring module keeps track of the error rate between actual and predicted performance interference level at run-time. Recall that iSensitive is trained offline with the historic data and utilizes the trained model for run-time predictions. However, there is always a possibility to encounter different workload patterns that were not known by the trained model. This will cause the system to incur high prediction errors. Therefore, the *interference monitoring* component is responsible for two tasks: (1) if the prediction error is greater than a configured threshold value, the actual workload pattern on the host is logged for re-training, (2) if a VM is way off from the actual cluster center points, it is also logged for re-clustering. These logged data are later used for re-training the performance interference model.

G. iSensitive Distributed System Architecture

The iSensitive distributed system architecture that can be integrated with cloud infrastructure software, such as OpenStack, is depicted in Figure 6. As can be seen in the figure, iSensitive comprises a Virtual Machine Manager (V-Man) for each VM residing on a physical host, a Host Manager (H-Man) for each physical host, and a Cloud Manager (C-Man) to orchestrate the cluster of host machines in the cloud data center.

The V-Man is responsible for collecting resource information, such as memory utilization, for VMs. The reason to employ the V-Man inside a VM is because we were unable to retrieve some of the metrics at the host level. For example, the actual memory in use by a VM cannot be collected when the host machine is virtualized by the Xen hypervisor and accessed through the libvirt API. This is because of concerns such as reliability and various operating systems run by VMs. Therefore, the statistics which are only known by the VM's kernel must be retrieved by an agent such as V-Man running inside the VM.

The primary goal of the H-Man is to accumulate statistics associated with each VM as well as the physical host machine and post these information to the C-Man. The CPU, memory,

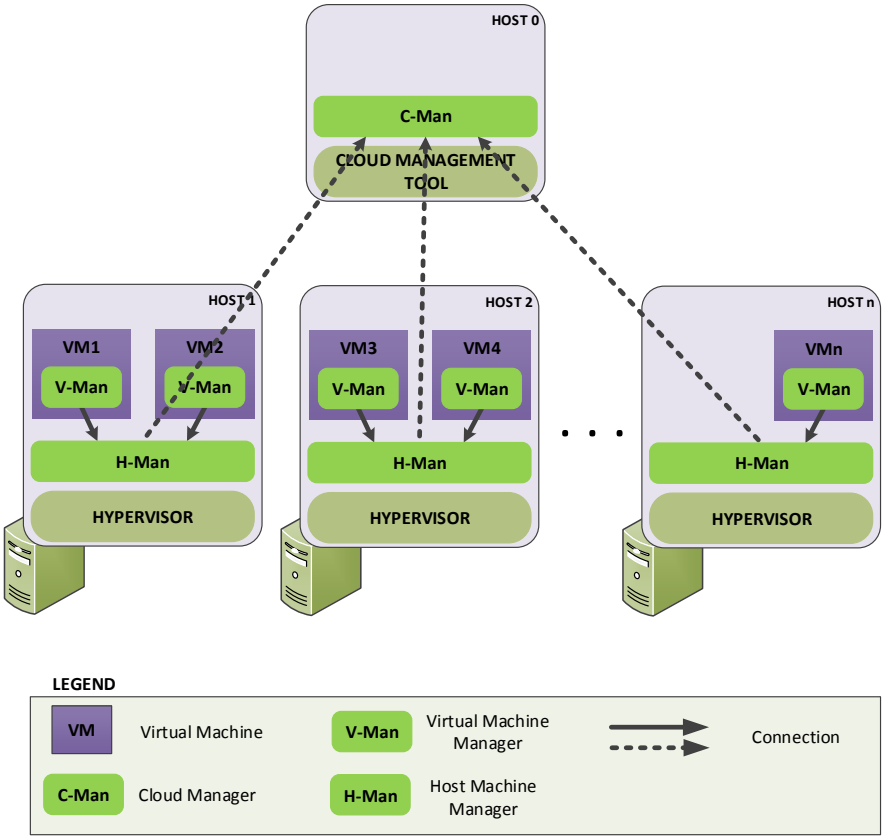

Fig. 6: iSensitive System Architecture Diagram

and network utilization of each VM and intrinsic scheduler parameters such as waiting time, I/O waiting time, and guest CPU percentage are some of the information being sent to the C-Man. Another critical information being sent to the C-Man is the cluster number of the VM. The cluster number of the VM is found by the minimum Euclidean distance from the actual resource usage information of a VM (i.e. CPU, memory, and network utilization) to the center points of clusters found by the virtual machine classifier component.

Another challenge handled by the H-Man is handling instant spikes of resource usage. Based on the configuration, if the instant spikes cause the VM's cluster number to change for five consecutive cycles (i.e. 30 secs of interval between each cycles), then iSensitive changes the actual cluster number with this new cluster number. The number of consecutive cycles triggering the change to the cluster number is a configuration parameter and can be tuned to another setting.

The C-Man is responsible for making decisions to place a VM on an aptly suited host machine in the cloud. All the host and VM-level statistics are collected through the collection of H-Mans and an overall view of the cloud environment is then defined by the C-Man. Eliminating hot spots, performance concern of a collocated high priority VM, server consolidation, and an antagonist, noisy neighbor VM are few of the primary reasons to migrate VMs. Whenever a VM needs to be migrated from one host machine to another one, the C-Man employs the decision maker and finds the target host machine where this VM should be migrated to.

IV. Validating the iSensitive Approach

This section presents empirical validation of the iSensitive's performance interference-aware virtual machine placement algorithm in our private data center. We compare the performance improvements stemming from the use of iSensitive in migrating an Apache web server VM-based application to one of the common approaches applied by data centers, e.g., first-fit bin-packing heuristics with preference for least occupied host [28], in our case, one with minimum CPU overbooking ratio. The experiments demonstrate that the iSensitive approach finds the aptly suited host machine with minimum performance interference level at the host-level and provides better performance to the applications running in the VM being migrated.

A. Experimental Setup

The experiments were conducted in our private data center comprising a cluster of 7 homogeneous host machines. The host machines were managed by OpenNebula [29] cloud management software version 4.6.2. Table III provides the hardware and software configurations while Table IV provides the virtualization configuration for each host machine in our private data center.

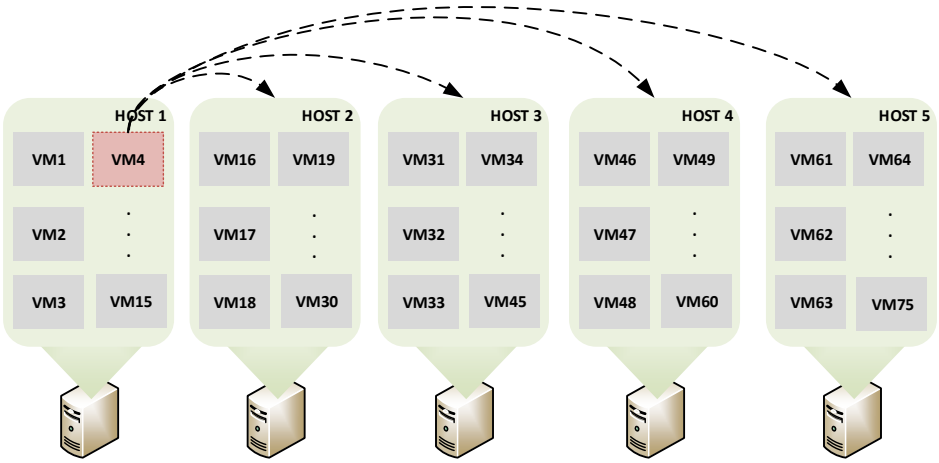

Fig. 7: Experimental Setup

TABLE III: Hardware and Software Specification of the Experiment Host

Memory (GB)	32
Hard Disk (GB)	500
Processor Type	AMD Opteron 4170 HE
CPU Socket Count	2
Core Count per Socket	6
Base Speed (MHz)	2100
L1 Cache Size (KB)	128
L1 Cache Count	6
L2 Cache Size (KB)	512
L2 Cache Count	6
L2 Cache Speed (MHz)	2100
L3 Cache Size (KB)	6144
Integrated Memory Controller Speed (MHz)	2200
Operating System	Ubuntu 14.04 64-bit

TABLE IV: Virtualization Specification of the Experiment Host

Hypervisor	KVM
Kernel	Linux 3.13.0-24
Qemu Virtualizer	2.0.0
Guest Virtualization Mode	HVM
Guest Operating System	Ubuntu 14.04 64-bit

Figure 7 describes the setup created to validate the effectiveness of the iSensitive approach. We created 15 VMs per host each having 2 vCPUs and 512 MB of memory on 5 different host machines. Each VM and host machine in this setup employs the V-Man and H-Man, respectively, as explained in Section III. Two more bare-metal host machines with the same configuration as in Table III were used. One machine was used to deploy C-Man and the other to send out client requests generated by the Apache jMeter load generator tool. jMeter sends out HTTP requests from 50 concurrent users to the Apache web server residing in the VM being tested. Virtualizing the server machine hosting the client application may also have resource contention causing inconsistent test results. Hence we decided not to use a VM for hosting the client, thereby providing more robust and consistent results.

Recall from Section III-E2 that we had found 5 as the ideal number of clusters for the raw data we had generated. iSensitive attempts to classify the VMs into one of these clusters. For the initialization of the experiments, randomly picked workloads from the benchmarking suite described in Section III-E1 were run on the 15 VMs on each of the five hosts. The host overbooking ratio for all 5 host machines was set to 2.5 for fairness between host machines. After the VMs were up and running, the workload on each VM was classified into one of the clusters. The resulting number of VMs per cluster number on each of the host machines was as shown in Table V.

TABLE V: Number of VMs in Each Cluster for Each Host

Host Name	C1	C2	C3	C4	C5
Host 1	7	1	7	0	0
Host 2	9	1	5	0	0
Host 3	6	1	7	1	0
Host 4	15	0	0	0	0
Host 5	7	1	5	0	1

The experiments were conducted by selecting one of the VMs from Cluster 3 on Host 1 and requesting a migration decision from iSensitive and comparing it to the migration decision using first-fit. The performance results were collected for a period of two minutes before and after migration and were found to be sufficient for analysis.

B. Application Performance Improvement using iSensitive

We analyzed the performance of the target VM for the three cases, i.e. performance before the migration and performance

after migration on the hosts decided by iSensitive and the first-fit heuristic. As mentioned earlier, the target VM was chosen from Host 1. iSensitive suggested its new location to be Host 4, whereas first-fit heuristic found it to be Host 2. For both the scenarios, we migrated the VM on these hosts and present the performance results before and after the migration.

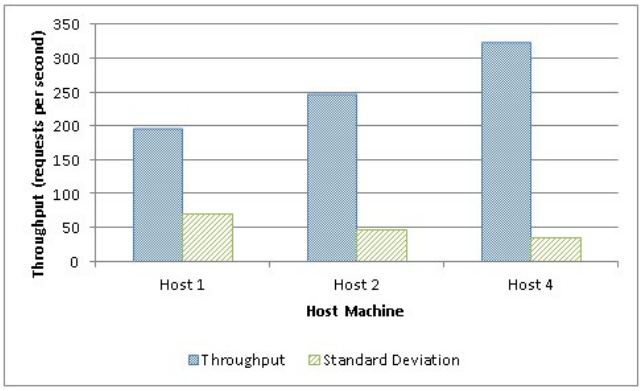

Fig. 8: Comparison of Web Server Throughput on Hosts 1, 2 and 4

Figure 8 presents the throughput for the three different scenarios. We observe that before the migration (see the bar for Host 1), the mean throughput of the Apache web server was 197 requests per seconds with a standard deviation value of 71, suggesting a sluggish performance. Hence, a decision to migrate is taken. After the migration to Host 2 (using first-fit), we see an improvement of 25% in throughput and the standard deviation value reduced to 47.7. Compared to first-fit's improvement, if the VM is migrated to Host 4 as suggested by iSensitive, we see a performance improvement of 64% percent in mean throughput and a lower standard deviation of 36.5 value (see the bar for Host 4). This shows that for our experimental use-case, the performance improvement due to iSensitive's placement decision is 39% better for mean throughput than the commonly used strategy of first-fit heuristics.

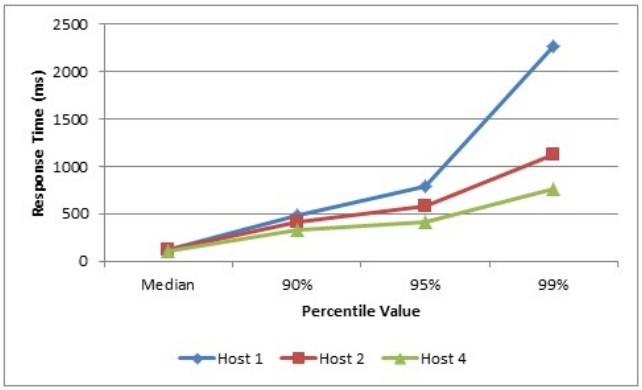

Fig. 9: Comparison of Web Server Response Time Percentiles on Actual, Before, and After Migrating to a Host Machine

Figures 9 and 10 depict the response time results for the same scenarios. In Figure 10, we can observe that the response time reduces significantly for iSensitive-suggested migration.

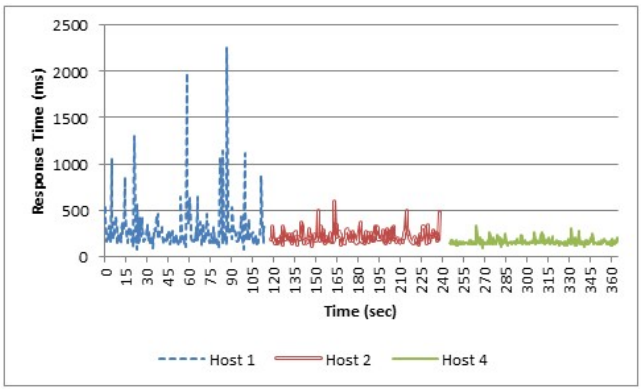

Fig. 10: Comparison of Web Server Response Time Over Time on Actual, Before, and After Migrating to a Host Machine

Figure 9 confirms the same where we see performance improvements of 14.8, 21.6, 27.8, 31.9 % for mean, 90th percentile, 95th percentile and 99th percentile, respectively, for the iSensitive approach over the first-fit heuristics approach. These two different performance indicators of throughput and response time improvements confirm the efficacy of the iSensitive solution.

We also measured the CPU overhead of using iSensitive in the runtime phase. It was found to be less than 1% for V-Man and C-Man and ˜5% for H-Man. These values are for the 2.5 overbooking ratio where 15 V-Mans are connected to 1 H-Man with a 1 seconds heartbeat interval, and 5 H-Mans connected to 1 C-Man with a 15 seconds heartbeat interval. This shows that iSensitive has low overhead, however, in future we would like to perform more scalability experiments on a larger cluster.

V. Conclusion

This paper presented iSensitive, which is a performance interference-aware virtual machine placement middleware to support performance-sensitive cloud-hosted applications, such as Big Data processing of IoT sensory data. The approach comprises two steps. In the first step, which is an offline step, raw usage data of a data center is used to glean away VM workload patterns for clustering decisions and key insights into performance interference caused due to VM collocation. To that end, a clustering-based VM placement approach was designed by utilizing back propagation artificial neural network. These insights are used in the second step, which is an online step, in finding an aptly suited host machine for VMs to minimize the performance interference effects and reduce the performance degradation in cloud-hosted IoT applications.

In this work, we have not yet considered disk-intensive applications but this will form a dimension of our future work. Disk utilization needs to be considered by the *virtual machine classifier* component in the future releases of iSensitive. Additionally, analyzing iSensitive's energy efficiency properties is left as future work.

The presented work and the algorithmic structure of iSensitive is generic enough to be used by cloud service providers for their platforms. They will need to learn the models based on historical data observed in their environment. Moreover, the

building blocks of the iSensitive distributed system architecture can seamlessly integrate with the cloud infrastructure software.

ACKNOWLEDGMENTS

This work was supported in part by the National Science Foundation CAREER CNS 0845789 and AFOSR DDDAS FA9550-13-1-0227. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of NSF and AFOSR.

REFERENCES

[1] F. Caglar and A. Gokhale, "iOverbook: Managing Cloud-based Soft Real-time Applications in a Resource-Overbooked Data Center," in *The 7th IEEE International Conference on Cloud Computing (CLOUD' 14)*. Anchorage, AL, USA: IEEE, Jun. 2014, pp. 538–545.

[2] L. Tomas and J. Tordsson, "An autonomic approach to risk-aware data center overbooking," *Cloud Computing, IEEE Transactions on*, vol. 2, no. 3, pp. 292–305, July 2014.

[3] I. S. Moreno and J. Xu, "Neural network-based overallocation for improved energy-efficiency in real-time cloud environments," in *Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC), 2012 IEEE 15th International Symposium on*. IEEE, 2012, pp. 119–126.

[4] S. A. Baset, L. Wang, and C. Tang, "Towards an understanding of oversubscription in cloud," in *Proceedings of the 2nd USENIX conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services*. USENIX Association, 2012, pp. 7–7.

[5] T. Abels, P. Dhawan, and B. Chandrasekaran, "An overview of xen virtualization," *Dell Power Solutions*, vol. 8, pp. 109–111, 2005.

[6] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori, "kvm: the Linux virtual machine monitor," in *Proceedings of the Linux Symposium*, vol. 1, 2007, pp. 225–230.

[7] A. Muller and S. Wilson, "Virtualization with vmware esx server," 2005.

[8] 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*. ACM, 2014, pp. 301–312.

[9] 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 Conference on Annual Technical Conference*, ser. USENIX ATC'13. Berkeley, CA, USA: USENIX Association, 2013, pp. 219–230. [Online]. Available: http://dl.acm.org/citation.cfm?id=2535461.2535489

[10] 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*. ACM, 2010, pp. 237–250.

[11] O. Tickoo, R. Iyer, R. Illikkal, and D. Newell, "Modeling virtual machine performance: challenges and approaches," *ACM SIGMETRICS Performance Evaluation Review*, vol. 37, no. 3, pp. 55–60, 2010.

[12] X. Pu, L. Liu, Y. Mei, S. Sivathanu, Y. Koh, and C. Pu, "Understanding performance interference of i/o workload in virtualized cloud environments," in *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*. IEEE, 2010, pp. 51–58.

[13] X. Zhang, E. Tune, R. Hagmann, R. Jnagal, V. Gokhale, and J. Wilkes, "Cpi2: Cpu performance isolation for shared compute clusters," in *Proceedings of the 8th ACM European Conference on Computer Systems*, ser. EuroSys '13. New York, NY, USA: ACM, 2013, pp. 379–391.

[14] J. Hwang, S. Zeng, F. Wu, and T. Wood, "A component-based performance comparison of four hypervisors," in *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on*. IEEE, 2013, pp. 269–276.

[15] Y. Koh, R. Knauerhase, P. Brett, M. Bowman, Z. Wen, and C. Pu, "An analysis of performance interference effects in virtual environments," in *Performance Analysis of Systems & Software, 2007. ISPASS 2007. IEEE International Symposium on*. IEEE, 2007, pp. 200–209.

[16] F. Caglar, S. Shekhar, and A. Gokhale, "iPlace: An Intelligent and Tunable Power- and Performance-Aware Virtual Machine Placement Technique for Cloud-based Real-time Applications," in *17th IEEE Computer Society Symposium on Object/component/service-oriented real-time distributed Computing Technology (ISORC '14)*. Reno, NV, USA: IEEE, Jun. 2014, pp. 48–55.

[17] ——, "Performance Interference-aware Virtual Machine Placement Strategy for Supporting Soft Real-time Applications in the Cloud," in *3rd International Workshop on Real-time and Distributed Computing in Emerging Applications (REACTION), IEEE RTSS 2014*. Rome, Italy: IEEE, Dec. 2014, p. 6.

[18] S. K. Barker and P. Shenoy, "Empirical evaluation of latency-sensitive application performance in the cloud," in *Proceedings of the first annual ACM SIGMM conference on Multimedia systems*. ACM, 2010, pp. 35–46.

[19] Q. Zhu and T. Tung, "A performance interference model for managing consolidated workloads in qos-aware clouds," in *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*. IEEE, 2012, pp. 170–179.

[20] R. C. Chiang and H. H. Huang, "Tracon: Interference-aware scheduling for data-intensive applications in virtualized environments," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 2011, p. 47.

[21] M. Kambadur, T. Moseley, R. Hank, and M. A. Kim, "Measuring interference between live datacenter applications," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE Computer Society Press, 2012, p. 51.

[22] I. S. Moreno, R. Yang, J. Xu, and T. Wo, "Improved energy-efficiency in cloud datacenters with interference-aware virtual machine placement," in *Autonomous Decentralized Systems (ISADS), 2013 IEEE Eleventh International Symposium on*. IEEE, 2013, pp. 1–8.

[23] 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*. ACM, 2014, pp. 277–288.

[24] C. Reiss, J. Wilkes, and J. Hellerstein, "Google cluster-usage traces: format+ schema," *Google Inc., White Paper*, 2011.

[25] J. A. Hartigan and M. A. Wong, "Algorithm as 136: A k-means clustering algorithm," *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, vol. 28, no. 1, pp. 100–108, 1979.

[26] P. J. Rousseeuw, "Silhouettes: a graphical aid to the interpretation and validation of cluster analysis," *Journal of computational and applied mathematics*, vol. 20, pp. 53–65, 1987.

[27] R. Hecht-Nielsen, "Theory of the backpropagation neural network," in *Neural Networks, 1989. IJCNN., International Joint Conference on*. IEEE, 1989, pp. 593–605.

[28] R. S. Camati, A. Calsavara, and L. Lima Jr, "Solving the virtual machine placement problem as a multiple multidimensional knapsack problem," *ICN 2014*, p. 264, 2014.

[29] D. Milojičić, I. M. Llorente, and R. S. Montero, "Opennebula: A cloud management tool," *IEEE Internet Computing*, vol. 15, no. 2, pp. 0011–14, 2011.