2019 IEEE International Conference on Cloud Engineering (IC2E)

Host Hypervisor Trace Mining for Virtual Machine Workload Characterization

Hani Nemati*, Seyed Vahid Azhari†, and Michel R. Dagenais‡
Department of Computer and Software Engineering,
Polytechnique Montreal, Quebec, Canada
Email: {*hani.nemati,†azharivs,‡michel.dagenais}@polymtl.ca

***Abstract*—The efficient operation and resource management of multi-tenant data centers hosting thousands of services is a demanding task that requires precise and detailed information regarding the behaviour of each and every virtual machine (VM). Often, coarse measures such as CPU, memory, disk and network usage by VMs are considered in grouping them onto the same physical server, as detailed measures would require access to the guest operating system (OS) which is not feasible in a multi-tenant setting.**

In this paper, we propose host level hypervisor tracing as a non-intrusive means to extract useful features that can provide for fine grain characterization of VM behaviour. In particular, we extract VM blocking periods as well as virtual interrupt injection rates to detect multiple levels of resource intensiveness. In addition, we consider resource contention rate due to other VMs and the host, along with reasons for exit from non-root to root privileged mode revealing useful information about the nature of the underlying VM workload. We also use tracing to get information about rate of process and thread preemption in each VM extracting process and thread contention as another feature set. We then employ various feature selection strategies and assess the quality of the resulting workload clustering. Notably, we adopt a two stage feature selection approach in addition to a one shot clustering scheme. Moreover, we consider inter-cluster and intra-cluster similarity metrics such as silhouette score to discover distinct groups of workloads as well as workload groups with significant overlap. This information can be used by 1) data center administrators to gain deep visibility into the nature of various VMs running on their infrastructure, 2) performance engineers to assist root cause analysis of VM issues and 3) IaaS providers to help in resource management based on VM behavior.

***Index Terms*—VM Clustering, Workload characterization, performance analysis, tracing, vCPU states, K-Means, Machine Learning.**

I. Introduction

In the few last years, cloud computing becomes an emerging technology that allows users to have access to pool of resources from anywhere and anytime with ease. Many enterprises are beginning to adapt VMs due to the fact that always access to the latest applications and no infrastructure needs to be maintained. Despite its merits, from IaaS provider point of view, the process of debugging and troubleshooting such infrastructure with hundreds of hosts and thousands of VMs is extremely complex. Specially, when one issue in a VM could lead to performance degradation of other VMs. Moreover, monitoring such infrastructure comes with scalibility issue due to the amount of data that should be analyzed. Hence, for such a complex environments, there is a need to elaborate more sophisticated and automated techniques for performance analysis of virtualized environments.

We argue that we could reduce the complexly of analyzing and monitoring VMs by leveraging the VM clustering technique. Using this technique, VMs that behave almost the same will be grouped and the amount of monitoring data will be reduced. As a result, the administrator should deal with ten VMs instead of thousands of VMs. As a very simple example, consider the scenario when one system resource is being fully utilized. The clustering technique could point to the group of VMs that suffer more from contention on the specific resource. Then the resource management could be easier by adding more resource or migrating VMs to another host. Detailed information about behavior of VM could not be revealed only by considering resource usage, it needs more complex method for feature extraction.

In addition, because of security issues and added overhead, the infrastructure provider does not have access to VM internally. Thus, in case of performance issue the IaaS provider could not provide useful insight. So, an agent-less technique is required for collecting metrics and feature extraction.

In this paper, we propose an unsupervised clustering approach with an agent-less technique for feature extraction and selection. Each VM is considered as a black-box with independent characteristics and with no prior knowledge of applications or processes running on the VMs. Unsupervised clustering methods are popular since they do not have dependency on labelled data and could detect various performance issue. We show that VMs could be clustered based on their similarity in term of executed workload. Our feature extraction method is based on hypervisor trace mining using existed static tracepoints in Kernel and KVM module along with an added dynamic tracepoint to obtain information from running VM processes. We analyze the states of each virtual CPU (vCPU) along with virtual interrupt injection rate to VM. Moreover, we consider resource contention rate due to other VMs and host along with the VM exit reason. Then, we adopt a two phase clustering scheme using K-Means. We consider inter-clustering and intra-clustering similarity metrics like silhouette score to depict the distinct groups of workloads. The outcome of our analysis could be used by IaaS administrator to improve the scalability, resource management, and root cause analysis of VM.

The main contributions of this work are three-folds: **First,** our feature extraction method limits its data collection to host

978-1-7281-0218-4/19/$31.00 ©2019 IEEE
DOI 10.1109/IC2E.2019.00024

hypervisor level, without internal access to VMs. All tracing and analyzing part is hidden to the VM. This is critical, as access to the VMs in public cloud is restricted. **Second,** a two phase feature selection and clustering is proposed. This method provides both coarse and fine grain workload characterization with detailed hits into performance issue. **Third,** we experiment on actual software (e.g., mySQL, Apache, etc.) in order to build our database to group VMs with regard to different scenarios such as overcommitment of resource and other possible problems. Our database is available online for further enhancements and to the benefit of other developers [1]. Furthermore, we implemented different graphical views as follows: VM similarity plot depicts the level of similarity between VMs. We also build a graphical view for vCPU that presents a timeline for each vCPU along with different states for the VM for further analysis after grouping the VMs.

The rest of this paper is organized as follows: Section II presents a summary of other existing approaches for VM feature extraction and clustering VMs. Section III introduces some background information about virtualization technology and presents the different states of applications inside the VMs and their requirements. It also presents the algorithm used to detect different state of VM vCPU. Moreover, our approach for feature extraction is presented in this section. Our VM workload clustering model is explained in section IV. Section V presents our experimental results along with a discussion about essential behavior of different clusters. We compare these approaches in terms of overhead, ease of use, and limitation, in section VI. Section VII concludes the paper with directions for future investigations.

II. Related Work

In this section, we review the available techniques for clustering VMs and feature extraction.

There are several techniques to gather metrics from VM. We categorized them into agent-based and agent-less techniques. In the agent-based, a daemon should execute inside the VM all the time and send the collected metrics to an external module for analysis. Many works like [2][3] collect their metrics from VM by injecting agent into VM. The agent usually uses existing API in Linux or simply analyze the output of Linux tool such as `iostat`[4] and `vmstat`[5]. On the other hand, agent-less techniques collect metrics with-out having access to the VMs. Virtual Machine Introspection (VMI) [6] is a technique for analyzing the VM memory space. Many works like [7][8] uses VMI technique for analyzing performance of VM. Another method to analyze VM without having agent is done in [9][10][11]. They used hypervisor level tracing for analyzing VM.

Canali *et al*. [12][13] proposed a method to detect similarities in VM behavior based on Smoothing Histogram-based clustering (SH-based Clustering). Then, they applied their method on EC2 Amazon datatbase which is coming from benchmarking PHP applications. They used basic system metrics (e.g., CPU usage, Network usage, Disk usage, Memory usage) coming from hypervisor API to feed their clustering algorithm. However, they focused on reducing the number of instances to be monitored for monitoring scalibility rather than detecting similar VMs for resource management. As they have shown, they could cluster VMs based on their usage and not based on actual behavior. The work in [14] clusters VMs based on K-means clustering algorithm in order to detect unexpected behavior. However, detecting anomalies and malware is hard by using only resource metrics. Having access to the VM and using tracing could be a acceptable solution for finding malware. Tracing VM is being used in [15] to detect Denial of Service (DoS) attacks. They used Support Vector Machine (SVM) to distinguish changes in VM environment.

VM clustering based on resource usage for VM placement is targeted in [16]. They used SH-based clustering algorithm to cluster VMs and then consider each group instead of a host as a bin-packing problem for VM deployment. Clustering based on correlation of resource usage is proposed in [17]. As we mentioned before, these approaches work for clustering based on resource utilization and does not consider behavior of processes inside VM. Another work that used tracing for anomaly detection is called Perfcompass and is proposed in [3]. PerfCompass is a statistical approach for finding external and internal fault for VMs. They used specially `syscalls` events to identify faulty VM.

Zhang *et al*. in [2] proposed an density-based clustering method (DBSCAN) for clustering VMs. For feature selection an entropy-based feature selection technique is used for data dimension reduction. However, to gather simple system metrics (e.g., CPU utilization, Memory usage, and Disk usage) they need an agent inside VM.

To the best of our knowledge, there is no pre-existing technique for feature extraction and analyzing the state of the vCPU inside the VM. Our technique can cluster VMs based on VM workload without internal access. Moreover, compared to other solutions, our method brings less overhead, and simplifies deployment in terms of tracing, since it limits its data collection to the host hypervisor level.

III. VM Features and Types Selection

A. VMX Operation

Intel processors (like AMD processors) provide support for virtualization by Virtual Machine eXtensions (VMX) instructions set (AMD supports Secure Virtual Machine (SVM) instructions set). There are two types of VMX operation: **VMX root** operation and **VMX non-root** operation. Privileged instruction executed by VM causes a VM exit to VMX root mode to pass the control to Virtual Machine Monitor (VMM) in order to execute the instruction in a safe environment. After handling the privileged instruction, the VM resumes to VMX non-root. The transition between VMX root to VMX non-root is called VM entry and the transition between VMX non-root to VMX root is called VM exit. The VMM keeps track of transitions by updating an in-memory data structure called Virtual-Machine Control Structure (VMCS). VMM creats different VMCS for each vCPU and uses `VMREAD`, `VMWRITE` and `VMCLEAR` instructions to update VM information. Each

VM exit comes with the exit number which showing the reason of exit. The number could be used to reveal information about running application inside VM.

B. Virtual Interrupt Injection

Virtual Machine Extensions (VMX), instructions on processors with x86 virtualization, supports virtual interrupt (`virq`) injection into VM by filling VM-entry interrupt-information field in Virtual Machine Control Structure (VMCS). Interrupt injection includes software interrupt, internal interrupt, and external interrupt. Then, the processor uses interrupt information field to deliver virtual interrupt through the VM Interrupt Descriptor Table (IDT). Usually, the virq will be injected to guest during next VM exit by emulating the `LAPIC` register. Once the VM resumed, the pending interrupt is executed by looking up the VM's `IDT`. After handling the interrupt, during the next VM exit, the processor performs End Of Interrupt `EOI` virulization to update guest interrupt status. Looking at injected interrupt reveals useful information about running application inside the VM. We elaborate more on the extracted metrics from injected `virq` in the next subsection.

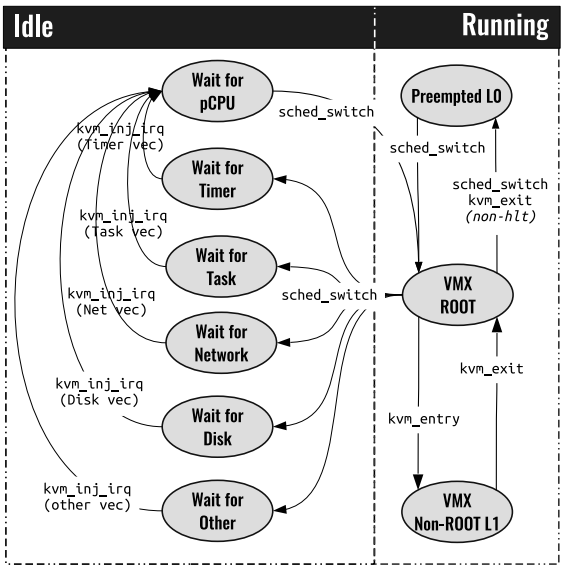

Fig. 1. Virtual Machine virtual CPU State Transition

C. VM Machine States Analysis

Figure 1 shows different possible states of virtual CPU of a VM and their condition to reach them. Like physical CPU (pCPU) each vCPU could be either in running state or idle state. As we mentioned before, while running a privileged instruction the VM exit to VMX root mode to execute the instruction and then with vm entry it resumes the VM. In addition, from VM point of view, VM assumes that it is the only one using the whole resources. In fact, VM shares its resources and that causes resource contention. The preemption state is when the host scheduler decides to take pCPU and gives it to another VM or process. It halts executing VM for a while without notifying the VM. Consequently, the VM process is in running state inside VM but it does not execute any instruction. The preemption state is one of the state that could not be shown from inside VM. Other than running states, analyzing idle states bring useful information. When a vCPU is being scheduled out voluntary, it execute `hlt` instruction which is a privileged instruction. This instruction causes an exit to VMX root and the host scheduler realize that the vCPU thread should be schedule out of pCPU. In this state the vCPU waits for a signal to be waken up. The host scheduler wakes up the vCPU thread mainly for following reasons. **First**, a process inside the VM sets a timer and the timer is fired (Timer Interrupt). **Second**, inter process communication wakes up a process (IPI interrupt). **Third**, a process inside VM waits for incoming packet from remote process (Network Interrupt). **Fourth**, a process inside the VM waits for disk (disk interrupts). Other than Timer, IPI, Network, and Disk interrupts, the VM could wait for another device (with different virq number) which could be labeled as wait for other. The wait for X states along with running state will be used by our feature extraction mechanism to collect information about running VM.

In order to detect the reason of waking up, we introduce Wake-up Reason Analysis (WRA). The pseudocode for the WRA algorithm is depicted in Algorithm 1. The WRA algorithm receives a sequence of events as input and detects the reason of wait for a vCPU of given VM. Whenever a VM has something to execute, it asks for pCPU from host scheduler. Receiving `sched_wakeup` event shows that the VM ask for a pCPU and the state of vCPU is updated to waiting for pCPU. Event `sched_in` shows a vCPU is scheduled in on pCPU, but still it is in waiting state to run. We name this state wait for vCPU. Before running the vCPU on pCPU the hypervisor injects interrupt into the VM. These interrups are the key of our analysis. The event related to interrupt injection is `vm_inj_virq`. The vec field in this event is compared with the Task, Timer, Disk, and Network interrupt number. The reason of waiting is updated based on the *vec* number. The different interrupt numbers are being retrieved by following the pattern of using device. This pattern is different between hypervisors.

The `vcpu_enter_guest` event changes the vCPU state to running (Line 18). We also store the CR3 value in the payload of this event. CR3 points to the page directory of a process in any level of virtualization, and could be used as an unique identifier of a process. As we mentioned before, a running vCPU could be preempted either in host level (L0) or guest level (L1). If the last exit is `hlt`, which means that the VM runs the idle thread, the state will be modified as wait for reason. In the other case of exit number, the state will be changed to preempted (Line 5). The preemption L1 happens when the process inside the VM is preempted by guest scheduler. In this case the cr3 will be changed with out running `hlt` instruction (Line 16).

Algorithm 1: Wakeup Reason Analysis Algorithm (WRA)

```
if event == sched_in then
    vCPU_j^State = vCPU-root;
    last_exit = getLastExit(vCPU_j) ;
    if last_exit != hlt then
        vCPU_k^State = Preempted_L0
    else
        vCPU_k^State = waiting_for_reason
else if event == vm_exit then
    putLastExit(exit_number);
    vCPU_j^State = vCPU-root ;
else if event == sched_wakeup then
    vCPU_j^State = pCPU-waiting ;
else if event == vcpu_enter_guest then
    lastCR3 = getLastCR3(vCPU_j);
    if lastCR3 != cr3 then
        vCPU_j^internal = Preempted_L1
    vCPU_j^State = vCPU-root;
    vCPU_j^State = vCPU-non-root ;
    putLastCR3(cr3) ;
else if event == sched_out then
    vCPU_j^State = idle ;
else if event == vm_inj_virq then
    if vec == Task Interrupt then
        Update State for vCPU_j^State to Task;
    else if vec == Timer Interrupt then
        Update State for vCPU_j^State to Timer;
    else if vec == Disk Interrupt then
        Update State for vCPU_j^State to Disk;
    else if vec == Network Interrupt then
        Update State for vCPU_j^State to Network;
    else
        Update State for vCPU_j^State to Other
```

D. Feature extraction and feature selection

Feature extraction is a method in which the input data transforms onto a low dimensional subspace. In this case, the size of the feature space decreases without losing a lot of information. It gives a simple representation of complex data that leads to more suitable input for learning algorithm. In the feature selection process a subset of features are selected to feed the learning algorithm. The advantage of feature selection is when the original features are very diverse and could mislead the learning algorithm. Our feature extraction and selection has several components that are shown in Figure 2 and are explained below. The events will be gathered by our agent-less tracing mechanism and will be extracted by our tracing analyser. Then, the collected data will be sent to our clustering algorithm and the result will be shown by our report engine.

1) Tracer: In order to analyse the workload of each VM and collect information, we use a very lightweight tracing tool called Linux Trace Toolkit Next Generation (LTTng)[18].

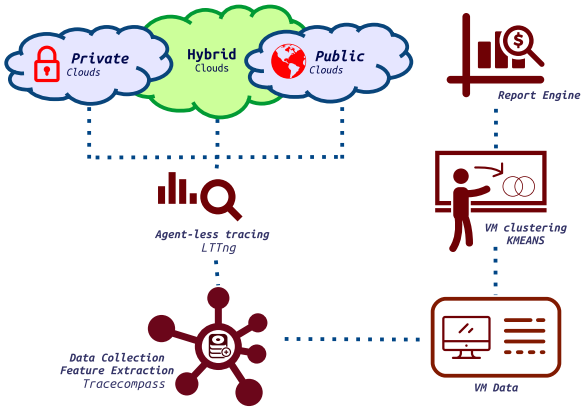

Fig. 2. Architecture of our implementation

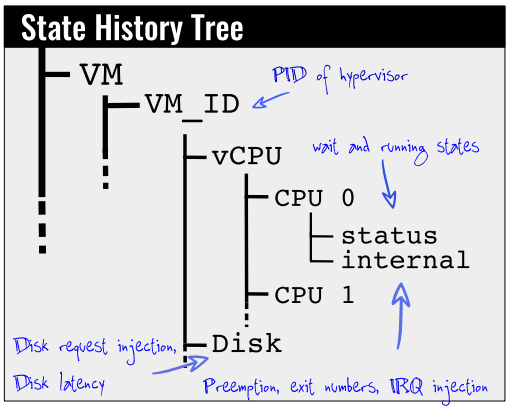

Fig. 3. State History Tree used to store different information of VM

LTTng is an open source tracing toolkit for Linux that allows to to gather events related to interaction between user space and kernel space. Additionally, the Linux kernel and KVM module are instrumented by difficult static tracepoints that LTTng collects them and sends them to the trace analyser. The tracepoint that we use for workload analysis is shown in Table I. `sched_wakeup` and `sched_switch` are existing Linux kernel tracepoints and are related to the process scheduler. `vm_exit` and `vm_inj_virq` are existing KVM module tracepoints and are related to virtualization technology. In order to complete our analysis, we create a new tracepoint, named as `vcpu_enter_guess`, for the host using kprobe, which can retrive VR3 and SP from VMCS at each VM entry. Note that, all of our tracepoints are in the host level and we do not need to have access to guest level. Consequently, we add less overhead to the VM.

2) Trace Analyser and Data collection: We implemented WRA algorithm in Tracecompass [19] as a separate module. Tracecompass is an open source tool for viewing and analyzing traces. Although, tracecompass has several pre-built analysis modules to extracted useful information from huge traces, it does not provide agent-less VM analysis. Using WRA

TABLE I
NEEDED TRACEPOINTS FOR FEATURE EXTRACTION

Tracepoint	Description
sched_wakeup	Wakeup and resume a task
vcpu_enter_guest	vCPU enters guest mode
vm_exit	vCPU exits guest mode
vm_inj_virq	Inject virtual interrupt into VM
sched_switch	Scheduling out or in a process

TABLE II
NOTATIONS

Term	Features collected by tracing
W_{Disk}	Wait for Disk average time
W_{Net}	Wait for Net average time
W_{Timer}	Wait for Timer average time
W_{Task}	Wait for Task average time
E_{Root}	Root mode average time
$E_{non-Root}$	Non-Root mode average time
f_{Disk}	The frequency of wait for Disk
f_{Net}	The frequency of wait for Net
f_{Timer}	The frequency of wait for Timer
f_{Task}	The frequency of wait for Task
I_{Disk}	Virtual disk irq injection rate
I_{Net}	Virtual network irq injection rate
I_{Timer}	Virtual timer irq injection rate
I_{Task}	Virtual task irq injection rate
FP_{VMVM}	The frequency of two VMs preempt each other
FP_{HostVM}	The frequency of host preempts a VM
FP_{VMProc}	The frequency of VM processes preempt each other
$FP_{VMThread}$	The frequency of VM threads preempt each other
N_{Exit}	The frequency of different VM exit reason
f_{Read}	The frequency of read from Disk
f_{Write}	The frequency of write to Disk
B_{Read}	Total Block Disk Read
L_{Read}	Total Latency Disk Read
B_{Write}	Total Block Disk Write
L_{Write}	Total Latency Disk Write

algorithm, we extract some useful information out of our collected trace and store it in a database named as State History Tree (SHT). The SHT is a tree shaped disk database that is used to store state of various components as integer, long, string values associated with a time interval. The structure of the SHT used for our analysis is depicted in Figure 3. For example, each VM has different vCPU named by their CPU ID. The vCPU has different states as shown in 1. Once the SHT is constructed by WRA algorithm, we query the SHT attributes to extract metrics. For each VM, we consider 25 metrics. The complete list of the metrics is provided in Table II along with a short description. The $W_X, X \in \{Disk, Net, Timer, Task\}$ metrics show the average time to wait for a vCPU waits for a signal to be waken up. The $E_X, X \in \{Root, non-root\}$ metrics are the average time that a VM is running in VMX root mode and VMX non-root mode. The $f_X, X \in \{Disk, Net, Timer, Task\}$ show the frequency of waking up a vCPU by different signals. W_X, E_X and f_X are calculated from status attribute of each VM. The $I_X, X \in \{Disk, Net, Timer, Task\}$ metrics are the frequency of injecting virtual interrupt into the VM.

3) VM Clustering and Report Engine: In order to cluster VMs based on their behaviour, we used Kmeans clustering algorithm. The metrics will be selected and will be sent to our VM clustering algorithm. Then, the output of clustering algorithm will be represented as a similarity matrix of different VMs.

IV. VM WORKLOAD CLUSTERING MODEL

We apply the Kmeans clustering algorithm [20] to our set of sample vectors derived via the feature extraction process. Note that this is an unsupervised process as compared with classification and requires no training data set. A clustering approach groups VM workloads based on the actual workload distribution in the data center. Consequently, it potentially "discovers" interesting groups of VMs, as opposed to assigning a predefined label to them. Each sample i represents some VM workload denoted by VM_i as a vector of features, i.e.,

$$VM_i = (f_{i1}, f_{i2}, ..., f_{ij}, ..., f_{iM}) \tag{1}$$

To allow for meaningful comparison, sample vectors are individually normalized using the $L2$ norm such that they become of equal magnitude [21], that is,

$$\sum_{m=1}^{M} f_{im}^2 = 1, \ \forall i = 1..N. \tag{2}$$

The Kmeans clustering algorithm will discover clusters $C_1, ..., C_K$ of similar VMs over a sample matrix, $VM_{N,M}$, which includes one sample vector per row. In order to do this, we use the Euclidean distance metric over feature vectors to measure similarity among VM samples. Two VMs are grouped into the same cluster if their Euclidean distance is small. More formally, the Kmeans approach to clustering is based on the notion of centroids (a.k.a. prototypes), which are hypothetical sample points placed at the center of each cluster. Given a certain number of centroids, K, Kmeans will determine cluster membership for all samples such that the sum of squared distance from each sample VM_i to its closest centroid C_j is minimized over all set of samples. That is,

$$min \sum_{i=1}^{N} \sum_{k=1}^{M} (f_{ik} - c_{jk})^2 \tag{3}$$

where the solution to (3) will be $C_j = (c_{j1}, ..., c_{jk}, ..., c_{jM}), j = 1..K$, which are the hypothetical coordinates of the j^{th} centroid that is the closest to VM_i.

A distinct advantage of Kmeans is that each centroid (a.k.a. prototype) can be used to represent the workload of its cluster as discussed in Section V. This provides condensed useful information of the bahavior of many VMs.

Furthermore, we need a measure to assess the suitability of a given clustering. There are an array of such measures introduced in the literature, which essentially characterize a clustering through its cohesion and/or separation. A cohesive clustering is one in which individual clusters consist of tightly packed very similar samples. Separation, on the other hand, evaluates how distinct clusters are well apart. Consequently, we adopt the "silhouette" score [22] which considers both

TABLE III
LIST OF APPLICATIONS FOR WORKLOAD GENERATION

Application Behavior	Application Name	Application Description
CPU	Sysbench, Stress, Burn, Chess, Compress, encode, interBench, openSSL, smallpt, infinitLoop	Bunch of well-konwn benchmarking tools for Linux to stress CPU including computing prime number, compressing files, video and audio encoding, mathematic operations
Disk	Sysbench, Compress, dbench, dd, interBench, hdparam, fsMarker, aio, ioZone, pgbench-disk, pqlight, stream, tioBench	Include different testing toolkits for Disk I/O. These toolkits provide realistic scenario for reading/writing with different options to evaluate Disk I/O.
Net	Wget, iperf, netperf, netping, netSpeedTest, scp, ab, httperf	Include real network applications in Linux. It covers web server, file transfer server, downloading file.

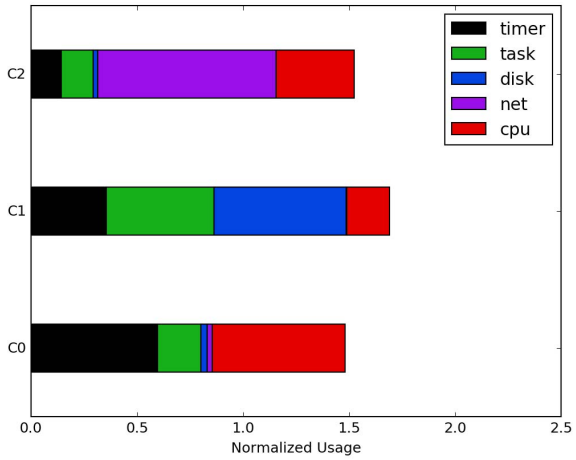

Fig. 4. Characteristic of Workloads Discovered in First Stage of Clustering

cohesion and separation and is computed for a given sample VM_i as,

$$s_i = \frac{out_i - in_i}{max(in_i, out_i)} \quad (4)$$

Here, in_i is the average distance of VM_i to all other VM samples in its cluster, while out_i is the minimum average distance between VM_i and all other samples in clusters not containing VM_i. The value of silhouette coefficient lies between -1 and 1, where a positive large value is more desirable. We would like to have in_i as close as possible to zero for a very cohesive cluster, and to have out_i as large as possible representing very large separation from the closest neighbor cluster. A negative value $(in_i > out_i)$ is undesirable because it corresponds to when average distance to in-cluster samples exceeds minimum average distance to out-cluster samples. It should be noted that, while the quality of a clustering algorithm plays an important role in finding good clusters, having cohesive and well separated clusters also largely depends on the nature of the data and the distance metric. The silhouette coefficient can be averaged over a cluster or the entire samples to yield a silhouette score per cluster or for the entire clustering, respectively.

We employ a two stage clustering approach, where the entire set of VMs are first grouped into a small number of clusters. We then apply a second stage clustering to each cluster discovered in the first stage to yield our final VM groups. Hence, each VM would be characterized by two numbers (Ci, Cj) standing for the cluster it belongs to in the first and second stage, respectively. We believe this two stage approach provides better grouping of VM workloads by first performing a coarse grain characterization of VMs followed by more detailed clustering. It should be noted that, our clustering selection criteria is always guided by total silhouette score. That is, at any stage we select the clustering with maximum total silhouette score as the desired solution.

V. EXPERIMENTAL EVALUATION

In this section, we presents results for the two stage clustering approach. In order to cluster VMs we generate different types of workload with well-known Linux applications (e.g., apache, mySQL, dd, Sysbench , etc.). The list of application for generating workload is depicted in Table III. We divided these application in three categories based on generated load. Then, we mixed them to generate over than 60 different workloads and scenarios.

For the results presented in this paper we have used VM execution time in VMX root and VMX non-root mode along with virtual interrupt injection rate for network, disk, task, and timer interrupts as our feature vector. As we show in following section, this feature vector could reveal the behavior of the VM. The VM process communicates with each other and also with host devices by injecting interrupts. At the first stage we found that applying Kmeans to find three coarse clusters would yield the best total silhouette score of 0.42 and per cluster silhouette of $C0 = 0.38, C1 = 0.35, C2 = 0.48$. Figure 4 shows the three centroids obtained via the first stage clustering. Note that the feature vectors are normalized and their absolute values are of no significance.

Interestingly, three distinct groups of VMs are discovered in Figure 4. Cluster C0 is discovered to be CPU intensive as the prototype usage pattern includes large CPU and Timer interrupt injection rate. The large timer injection rate corresponds to many host OS-internal counter ticks (scheduler ticks) which is characteristic of CPU intensive tasks that do not yield CPU. Whenever a VM enters into guest mode, the KVM module clears CPU preemption flag. In this state, no one could preempt the VM. In order to be able to yield the pCPU, the scheduler kicks the VM with injecting external interrupts into vCPU. Injecting external interrupt causes an exit to VMX root mode (with exit number one). In VMX root mode, KVM module sets the preemption flag in order to authorize host scheduler to be able to preempt the VM. As a result for CPU intensive applications, the rate of virtual timer interrupt is high.

On the other hand, clusters C1 and C2 represent disk and network intensive VMs, respectively. An interesting observation is that unlike the network intensive VMs of C2, disk

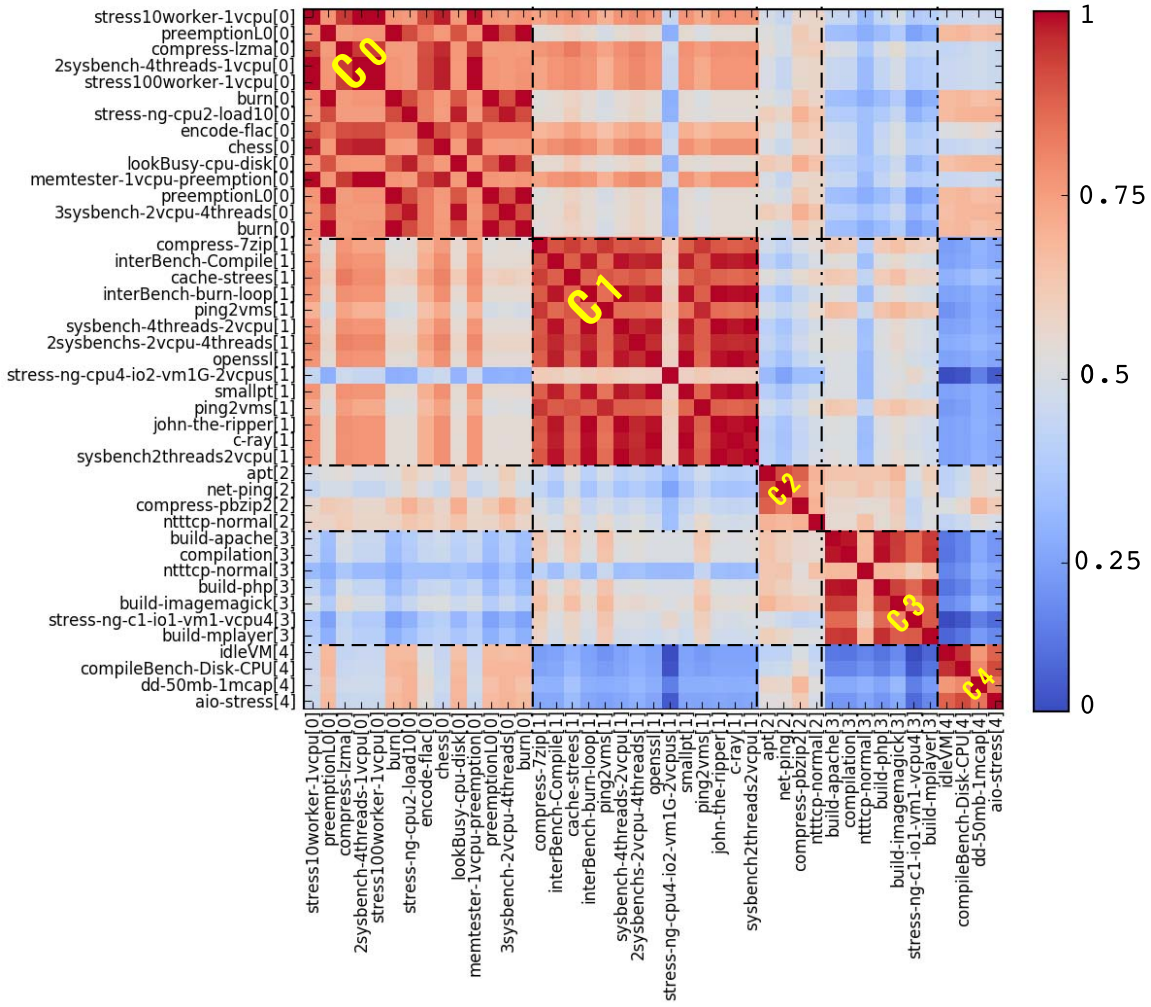

Fig. 5. Similarity Plot For the Five Workload Clusters Discovered in the First Group (Silhouette = 0.59)

intensive VMs in C1 make significant use of task interrupt. This is an indication of disk VMs waiting more frequently for some task to finish I/O as opposed to network VMs which wait for some process on another machine. The result of our first stage of clustering can be used by cloud infrastructure administrator to guide resource management. A host with too many CPU intensive VMs causes preemption on physical CPU. This could affect running VM process which leads to latency. A host with too many disk intensive VMs also could cause disk contention. Using our clustering method the administrator could balance the host resource utilization and reduce resource contention.

We next apply Kmeans to each of the first stage clusters independently and select the clustering with largest silhouette score. For our particular set of workload, this approach resulted in a total of 10 clusters. The first stage cluster C0 turns out to be best characterized as five distinct groups of VMs with a total silhouette score of 0.59. Figure 5 illustrates the similarity plot for the five clusters found in the second stage. Each point on both X and Y axis correspond to a particular VM which is labeled once on the Y axis. The second stage cluster index corresponding to this VM is enclosed in brackets.

Each point on the similarity plot presents the normalized euclidean distance defined as in (5) between VM feature vectors intersecting at that point. Note that d_{ij} is the actual euclidean distance between VM i and j, while d_{max} and d_{min} correspond to the maximum and minimum distances, respectively. It follows that the similarity score of any two VM is between zero and one. Interestingly, if VMs are sorted according to their cluster index, the similarity plot should resemble a block diagonal shape for cohesive and well separated clusters.

$$sim_{ij} = 1 - \frac{d_{ij} - d_{min}}{d_{max} - d_{min}} \quad (5)$$

The similarity plot of Figure 5 illustrates such a block diagonal pattern asserting that the clustering is good. This is also

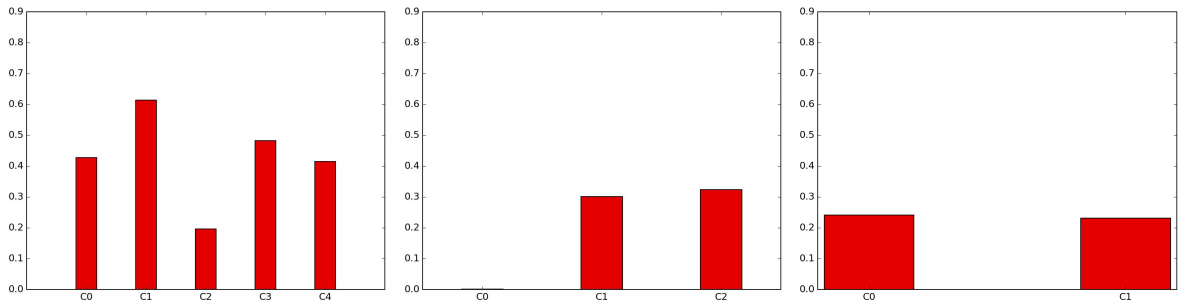

Fig. 6. Silhouette Score of VM Clusters Discovered in Second Stage of Clustering

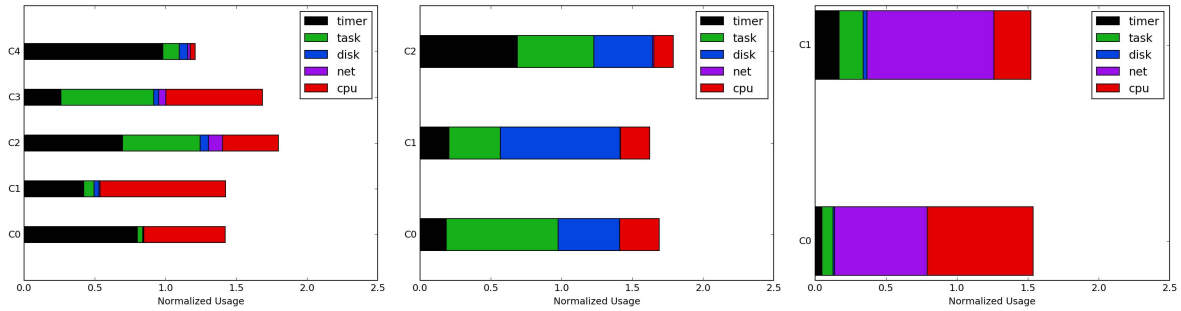

Fig. 7. Characteristic of Workloads Discovered in Second Stage of Clustering

reflected in the silhouette score for each cluster as plotted in Figure 6a showing positive and mostly large scores. According to Figure 7a there are a set of five different workload groups discovered in this case, for which the similarity plot is as in Figure 5. These groups mainly differ on the amount of their CPU usage as well as reliance on timer and task interrupts. For instance, there are two groups, (C0,C2) and (C0,C3) which make intensive use of the timer interrupt, reflecting multiple process dependencies. On the other hand, (C0,C4) is almost entirely timer intensive with minimal CPU usage indicating that the VMs in this group represent mostly idle workloads. In fact this is true, since the ”Idle” workload is put into this cluster. Instances in this group could be in any host with out causing any resource contention.

Let us elaborate on a few interesting observations from the similarity plot for C0. The first interesting observation is regarding VMs in cluster (C0,C3) which have high task dependency. It means that they are either dependent on other running processes or dependent on their own threads. For example, all of VMs that build an application are in this group since building a file depends on compiling other files. Figure 8 shows an example of this group. As it is shown, processes waits for other processes to finish their job.

Moreover, there is a moderate similarity among VMs in clusters (C0,C1) and (C0,C0). This is visible from the moderately warm areas at the intersection of these two clusters. Further inspection of the VMs reveals that these two groups are both CPU intensive with negligible task dependency. It means that both group have almost single threaded processes. This is clearly observed from the centroids for C1 and C0 in Figure 7a, further showing a sharp contrast in CPU and timer composition among the two clusters. VMs in cluster (C0,C0) have special behaviour. Figure 9 depicts one instance of this cluster. As it is shown, it executes small amount of code and then waits for timer to be fired. Other VMs could be affected by this catastrophic behaviour. Some instances of this group do not utilize the pCPU and preempt other vCPU running on the pCPU. Then all instances with preemption in host level are in this group. In contrast, VMs in cluster (C0,C1) are super CPU intensive and utilize the pCPU all the time. Despite this overlap, Figure 6a shows that the silhouette score for (C0,C1) is significantly larger than (C0,C0), suggesting that C1 is a better cluster. This is because (C0,C1) is a more cohesive cluster compared with (C0,C0), as suggested by the very warm texture of (C0,C1). This implies that the VMs in this group are very similar and closely resemble the prototype centroid representing this cluster. This information can be useful for the cloud administrator when dealing with deployment of these very similar workloads. VM deployment strategy should satisfy both IaaS provider and VM user. It should guarantee an efficient usage of host resources while avoiding resource contention between VMs.

As another interesting observation, one could spot two particular VMs in (C0,C3) and (C0,C1) which are quite dissimilar to their co-cluster VMs. These are the `ntttcp-normal` workloads, which make moderate use of network but are put into these clusters due to making significant use of timer and task as well as CPU resources. While it is debatable to put

these workloads in the same group with non-network VMs, the similarity plot can easily visualize such peculiarities for the system administrator.

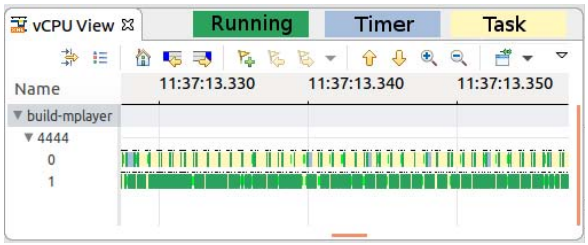

Fig. 8. vCPU view for build-mplayer VM that has high task dependency and high cpu utilization

Fig. 9. One of the instances from cluster (C0,C0)

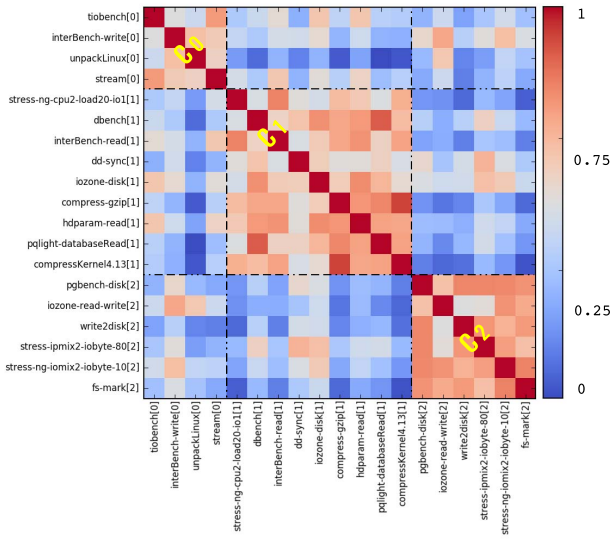

Fig. 10. Similarity Plot For the Three Workload Clusters Discovered in the Second Group (Silhouette = 0.40)

The second and third stage one clusters are also further processed into three and two groups of workload by our approach. Figures 10 and 11 provide similarity plots for C1 and C2, respectively. As for (C0,C3) and (C0,C1), (C2,C1) also includes a rather dissimilar VM, namely, `host-slow-disk` which although highly network intensive but also makes moderate disk usage. `host-slow-disk` VM executes `scp` to copy a big file from one VM to the host. For this special VM, we put cap on disk to slow down the reading data. It is true, since many IaaS providers like Amazon limit number of Block I/O access in order to reduce contention on disk. As a result, we could call the VM more network intensive rather than disk intensive. Therefore, it lies some where between the disk intensive and network intensive groups but more close to the latter one.

Another peculiar case is (C1,C0) with a cluster silhouette score of almost zero as indicated in Figure 6b. Inspecting the similarity plot of Figure 10 for C1 reveals a rather loose cluster in this case (i.e., C0). While Figure 7b shows that the centroid for this cluster (C0) is highly task dependent, further inspection of the feature vectors extracted for the VMs in (C1,C0) reveal that they are rather diversely affected by timer interrupt and CPU usage. Such cases with low cluster quality are generally not of interest from a resource management point of view. However, they can be still used to investigate the root cause of performance issues through their representative prototype workload.

VMs in cluster (C1,C2) reads or write huge bunch of file at once and then wait for some time and then continue. In contrast, VMs in cluster (C1,C0) reads small data. Reading small file extremely harmful for other VMs since it issue bunch of disk request to host level and cause contention.

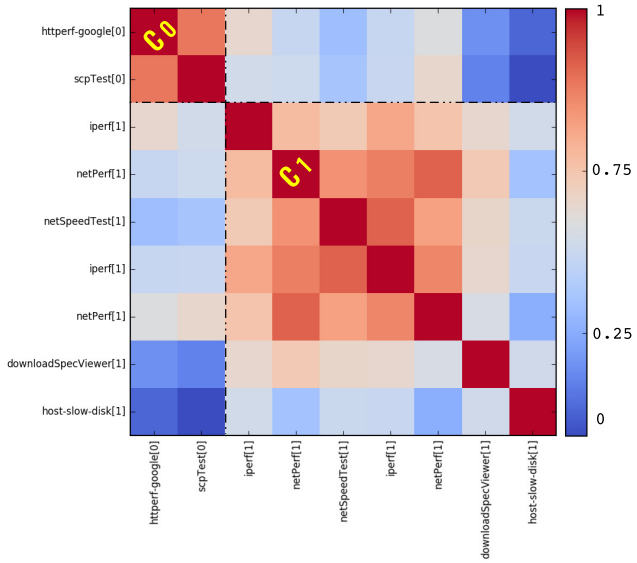

Fig. 11. Similarity Plot For the Two Workload Clusters Discovered in the Third Group (Silhouette = 0.46)

Let us now elaborate on the workload prototypes discovered in the second stage, which are shown in Figures 7a, 7b, and 7c, corresponding to first stage clusters C0,C1,C2, respectively. Recall that C0 included CPU intensive VMs, while C1 and C2 were discovered to be disk and network intensive, respectively. We start from the network intensive group C2, which is discovered to have only two clusters of workloads; (C2,C1) which is highly network intensive and (C2,C0) with also significant CPU usage. The latter group should be allocated

TABLE IV
OVERHEAD ANALYSIS OF WRA ALGORITHM COMPARING WITH AGENT-BASED FEATURE EXTRACTION METHOD

Benchmark	Baseline	FAE	WRA	Overhead	
				FAE	WRA
File I/O (ms)	450.92	480.38	451.08	6.13%	0.03%
Memory (ms)	612.27	615.23	614.66	4.81%	0.01%
CPU (ms)	324.92	337.26	325.91	3.65%	0.30%

adequate CPU to prevent bottlenecks, whereas the former group can be co-located with CPU intensive VMs on a server which has otherwise no network usage. Hence, while VMs in (C2,C0) are compatible with those of all other first stage groups, (C2,C1) may only be co-located on physical servers hosting (C1,C2) or (C0,C4).

As for the disk intensive group, we have discovered three clusters with contrasting behavior in the composition of disk and task usage as well as timer interrupt rate. More specifically, we observe two contrasting set of workloads (C1,C1) with large disk usage but small inter-task dependency and (C1,C0) with moderate disk usage and significant task dependency. Alternatively, (C0,C0) and (C0,C1) make high use of CPU and timer. This indicates that these VMs contain contending processes which are CPU intensive in nature.

VI. OVERHEAD ANALYSIS

In this section, the overhead of WRA algorithm is compared with the agent-based feature extraction (AFE) method. In order to compare the two approaches, we enabled the tracepoints that were needed for extracting the same feature from inside of VM. Also, it is worth mentioning that our WRA algorithm needs only to trace the host. As shown in Table IV, the FAE approach adds more overhead in all tests, since it needs to trace the VMs and the overhead of virtualization will be added. We used the Sysbench benchmarks to reveal the overhead of both approaches, since sysbench is configured for Memory, Disk I/O and CPU intensive evaluations. Our approach has negligible overhead for CPU and Memory intensive tasks at 0.3% or less.

VII. CONCLUSIONS

Service virtualization in cloud is enumerated as the major player of nowadays market. Thus, IaaS providers deal with various types of workload coming from different services. Accordingly, for troubleshooting and debugging, the IaaS provider should monitor and analyze thousands of VMs which is a extremely complex process. This complexity could be reduced by automatically finding the group of VMs that could be the cause of issue or be affected by the issue. As a result, the admin deal with a few VMs instead of thousands of VMs. Not only clustering VM could help finding issue, but also it could assist the resource management process.

In this paper, we have introduced a host based VM feature extraction method to extract meaningful information and provide fine grain characterization of VM behaviour. We used K-means clustering technique to group VMs which have similarity in term of workload behavior. The two phase clustering technique let us to apply more directive features in the phase two in order to find root cause of an issue. To validate our work, we built a database of actual software. The experimental results showed that our method could group VMs with similarity. As an example, it grouped the VMs that had resource contention on CPU. Our benchmarks depict that the accumulated overhead in our approach was around 0.3%. In contrast, the overhead of other approaches ranged from 3.65% to 6.13%. As a future work, our current technique can be enhanced to use other existing feature for further analysis of VM.

ACKNOWLEDGMENT

We would like to gratefully acknowledge the Natural Sciences and Engineering Research Council of Canada (NSERC), Ericsson, Ciena, and EffciOS for funding this project. We thank Genevieve Bastien for her comments.

REFERENCES

[1] "Vm workload database," https://github.com/Nemati/VMClassificationDataBase, accessed: 2018-11-03.

[2] X. Zhang, F. Meng, and J. Xu, "Perfinsight: A robust clustering-based abnormal behavior detection system for large-scale cloud," in *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, July 2018, pp. 896–899.

[3] D. J. Dean, H. Nguyen, P. Wang, X. Gu, A. Sailer, and A. Kochut, "Perfcompass: Online performance anomaly fault localization and inference in infrastructure-as-a-service clouds," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 6, pp. 1742–1755, June 2016.

[4] "iostat," http://linux.die.net/man/1/iostat, accessed: 2018-02-12.

[5] "vmstat," http://linuxcommand.org/man_pages/vmstat8.html, 2005, accessed: 2018-02-12.

[6] "Libvmi, virtual machine introspection library," http://libvmi.com/, accessed: 2018-2-25.

[7] S. T. Jones, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Vmm-based hidden process detection and identification using lycosid," in *Proceedings of the Fourth ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, ser. VEE '08. New York, NY, USA: ACM, 2008, pp. 91–100. [Online]. Available: http://doi.acm.org/10.1145/1346256.1346269

[8] L. Litty and D. Lie, "Manitou: A layer-below approach to fighting malware," in *Proceedings of the 1st Workshop on Architectural and System Support for Improving Software Dependability*, ser. ASID '06. New York, NY, USA: ACM, 2006, pp. 6–11. [Online]. Available: http://doi.acm.org/10.1145/1181309.1181311

[9] H. Nemati and M. R. Dagenais, "virtflow: Guest independent execution flow analysis across virtualized environments," *IEEE Transactions on Cloud Computing*, pp. 1–1, 2018.

[10] ——, "Vm processes state detection by hypervisor tracing," in *2018 Annual IEEE International Systems Conference (SysCon)*, April 2018.

[11] ——, "Virtual cpu state detection and execution flow analysis by host tracing," in *2016 IEEE International Conferences on Big Data and Cloud Computing (BDCloud), Social Computing and Networking (SocialCom), Sustainable Computing and Communications (SustainCom) (BDCloud-SocialCom-SustainCom)*, Oct 2016, pp. 7–14.

[12] C. Canali and R. Lancellotti, "Detecting similarities in virtual machine behavior for cloud monitoring using smoothed histograms," *Journal of Parallel and Distributed Computing*, vol. 74, no. 8, pp. 2757 – 2769, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0743731514000343

[13] ——, "Exploiting ensemble techniques for automatic virtual machine clustering in cloud systems," *Automated Software Engineering*, vol. 21, no. 3, pp. 319–344, Sep 2014. [Online]. Available: https://doi.org/10.1007/s10515-013-0134-y

[14] M. Abdelsalam, R. Krishnan, and R. Sandhu, "Clustering-based iaas cloud monitoring," in *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*, June 2017, pp. 672–679.

[15] A. Abusitta, M. Bellaiche, and M. Dagenais, "An svm-based framework for detecting dos attacks in virtualized clouds under changing environment," *Journal of Cloud Computing*, vol. 7, no. 1, p. 9, Apr 2018. [Online]. Available: https://doi.org/10.1186/s13677-018-0109-4

[16] C. Canali and R. Lancellotti, "Exploiting classes of virtual machines for scalable iaas cloud management," in *2015 IEEE Fourth Symposium on Network Cloud Computing and Applications (NCCA)*, June 2015, pp. 15–22.

[17] ——, "Exploiting classes of virtual machines for scalable iaas cloud management," in *Journal of Communications Software and Systems*, vol. 8, no. 4, 2012, pp. 102–109.

[18] M. Desnoyers and M. R. Dagenais, "The lttng tracer: A low impact performance and behavior monitor for gnu/linux," in *OLS (Ottawa Linux Symposium) 2006*, 2006, pp. 209–224.

[19] "Trace compass," https://projects.eclipse.org/projects/tools. tracecompass, 2018, accessed: 2018-02-12.

[20] J. Macqueen, "Some methods for classification and analysis of multivariate observations," in *Proc. 5th Berkeley Symp. Mathematical Statist. Probability*, 1967, pp. 281–297.

[21] I. S. Dhillon, Y. Guan, and B. Kulis, "Kernel k-means: Spectral clustering and normalized cuts," in *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '04. New York, NY, USA: ACM, 2004, pp. 551–556. [Online]. Available: http://doi.acm.org/10.1145/1014052.1014118

[22] L. Wang, T. Tan, H. Ning, and W. Hu, "Silhouette analysis-based gait recognition for human identification," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 25, no. 12, pp. 1505–1518, Dec 2003.