A Machine Learning Auditing Model for Detection of Multi-Tenancy Issues Within Tenant Domain

Cleverton Vicentini[1,2], Altair Santin[1], Eduardo Viegas[1], Vilmar Abreu[1]
[1]Graduate Program in Computer Science / Pontifical Catholic University of Parana, Curitiba, Parana, Brazil
[2]Federal Institute of Parana, Curitiba, Parana, Brazil
{cleverton, santin, eduardo.viegas, vilmar.abreu}@ppgia.pucpr.br

Abstract—**Cloud computing is intrinsically based on multi-tenancy, which enables a physical host to be shared amongst several tenants (customers). In this context, for several reasons, a cloud provider may overload the physical machine by hosting more tenants that it can adequately handle. In such a case, a tenant may experience application performance issues. However, the tenant is not able to identify the causes, since most cloud providers do not provide performance metrics for customer monitoring, or when they do, the metrics can be biased. This study proposes a two-tier auditing model for the identification of multi-tenancy issues within the tenant domain. Our proposal relies on machine learning techniques fed with application and virtual resource metrics, gathered within the tenant domain, for identifying overloading resources in a distributed application context. The evaluation using Apache Storm as a case study, has shown that our proposal is able to identify a node experiencing multi-tenancy interference of at least 6%, with less than 1% false-positive or false-negative rates, regardless of the affected resource. Nonetheless, our model was able to generalize the multi-tenancy interference behavior based on private cloud testbed monitoring, for different hardware configurations. Thus, a system administrator can monitor an application in a public cloud provider, without possessing any hardware-level performance metrics.**

Keywords— Multi-tenancy Interference; Cloud Computing; Machine Learning Classifier; Stream Processing, Provider and Client (tenant) Auditing

I. Introduction

Cloud computing aims at providing scalable and on-demand computing resources, such as processing, storage, and network [1]. The key aspects of cloud computing model are elasticity and multi-tenancy. Elasticity enables users to scale and pay for the consumed resources only, eliminating the need to maintain their own hardware infrastructure. In contrast, multi-tenancy enables several cloud computing clients (tenants) to share the same physical hardware infrastructure [2].

Multi-tenancy is achieved through the virtualization of the physical resources, which is performed by a software known as *hypervisor*, such as Xen [3] and KVM [4]. The *hypervisor* acts as the middleware between the tenants and the physical hardware. In this context, tenants are typically represented as virtual machines (VMs), which access the same physical hardware resources according to the *hypervisor* policies [5].

Thus, as a single physical resource (e.g., CPU) can be shared amongst several tenants, its performance may get impacted. For example, a cloud provider that hosts more tenants than the physical hardware can handle (typically referred as *overbooking of resources*), to maximize the profits. However, this *overbooking of resources* is invisible to the cloud computing customer, who is only able to monitor the virtualized (tenant) resources, which are accessible only within the *hypervisor* domain [6].

The *overbooking of resources* in general, will result in the increase or a highly variable response time for the application running in the tenant domain. However, for a near real-time application (e.g., stream processing platforms) it implies processing time constraints; in this case a highly variable application response time increase may inhibit its use on a cloud computing platform [7].

Current approaches to deal with *overbooking of resources* in general, aim at changing the tenant disposal policies in the hardware infrastructure, typically for the energy saving purposes [8] for a cloud provider. Thus, these approaches consider the perspective of the cloud provider only. On the other hand, approaches for auditing cloud Service Level Agreements (SLAs) consider the cloud client perspective, focusing on the availability of resources [9], but not their performance. Moreover, this multi-tenancy issue may not be correctly reported by a public cloud provider owing to a possible conflict of interests [10].

In the light of this situation, this study proposes a cloud independent auditing model for the identification of multi-tenancy issues from the tenant (client) perspective. This work presents two insights for the identification of multi-tenancy issues. First, we apply a two-tier auditing model: resource-based and application-based. The purpose of the two-tier auditing model is to provide metrics (features) to enable the identification of deviation between the used resources (virtualized) and the application performance. Second, based on the extracted features, we employ machine learning techniques that enable to identify multi-tenancy issues in public clouds (e.g., Amazon), where the provided performance metrics could be biased. Through our technique one is able to use a model trained in a controlled environment, to monitor her application behavior in a public cloud testbed.

The contribution of this study is twofold. First, through the evaluation of a stream processing platform (Apache Storm), we show that multi-tenancy significantly degrades the application performance (increasing the response time by up to 9 times). Second, we propose a model that enables to identify when an

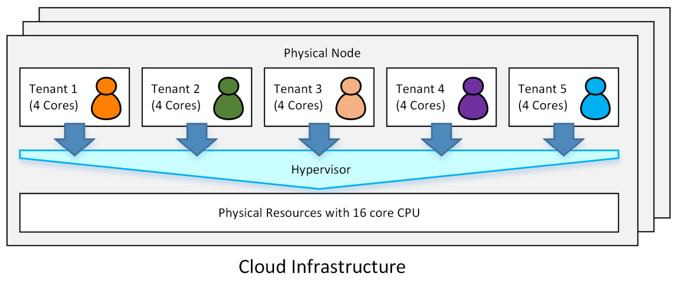


Figure 1 – Typical *overbooking of resources* at hardware level in cloud environments.

application experiences multi-tenancy interferences within the tenant domain, in a cloud independent manner. In this manner, our proposal is able to generalize the application behavior (and the expected resources usage) through a two-tier auditing model by employing machine learning techniques. More specifically, we are able to train our model in a controlled environment, generalize the system behavior enabling the monitoring of the same application in a public cloud domain, while also considering both different application workloads and hardware configurations.

The remainder of this paper is organized as follows. Section II presents the work preliminaries. Section III presents our two-tier auditing model. Section IV describes our prototype. Section V depict prototype evaluation. Section VI the related works, and finally Section VII draws concludes.

II. PRELIMINARIES

In this section, we discuss the background for our proposal, and some experiments showing the multi-tenancy impact on cloud infrastructures.

A. Machine Learning

Machine learning techniques are extensively used for the classification of events into groups (classes) [11]. These techniques in general rely on supervised learning through pattern recognition techniques to do the classification. In such a case, the learner is provided a set of previously labeled events, represented as a feature vector, from the considered classes. The classifier algorithm then learns the behavior of each class to obtain a model. Finally, through the model, the user is able to classify new events into the considered classes.

As the classification is performed through the extracted set of features, the model may wrongly classify similar events [11]. Thus, an important stage of the model building process is its evaluation. In such a case, the model is evaluated regarding its expected accuracy rates, typically measured in terms of false-positive (FP) and false-negative (FN) terms.

The FP rates denote the rate of examples from a modeled class wrongly classified as not belonging to it, while the FN rate represents the opposite classes error. In this manner, the data used during the evaluation process must represent all the production (real-world) environment properties as close as possible, since the accuracy rates will be calculated from that.

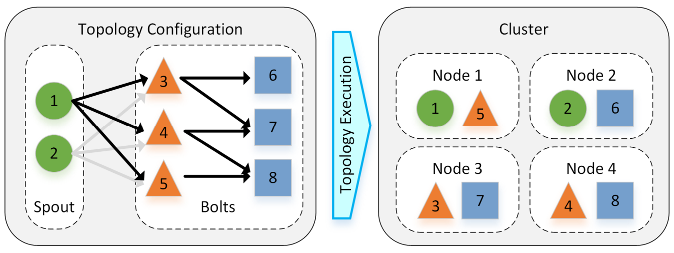


Figure 2 – Typical stream processing platform execution architecture.

B. Cloud Computing

Cloud computing can be deployed either on-premises in a controlled (private and community), or in a public environment [1]. In the former, the cloud customers have access to the cloud physical infrastructure and its management interface (e.g., Eucalyptus HPE [12]). Thus, a customer is able to manage his tenant's allocation disposal. However, in public clouds, the management of resources is done by the cloud providers [1], who offer the cloud resources as a service for profit purposes.

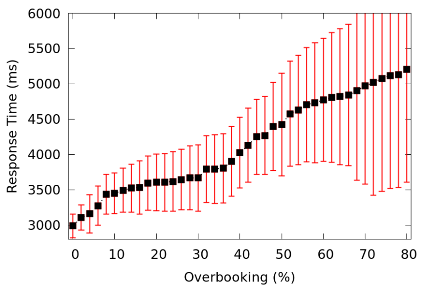
As the customer in a public cloud does not have control over the physical infrastructure, the cloud provider provides a contract for the client, known as Service Level Agreement (SLA) [9]. An SLA defines the customer guarantees such as availability (probability that a system is operational at a given time) and data security amongst others.

The majority of the public cloud providers provide SLAs specific to the resource's availability, which can be easily monitored by the cloud customer through Service Level Indicators (SLI) [9, 13]. In contrast, when the performance of the resources is desired, the client must trust the cloud provider as the SLAs typically do not provide such guarantees. Moreover, when performance metrics are provided (e.g., CPU steal time[1]), a conflict of interests may occur if the cloud provider changes the performance metrics for profit purposes [9, 13] or because of its inability to monitor the given resource when it is overloaded.

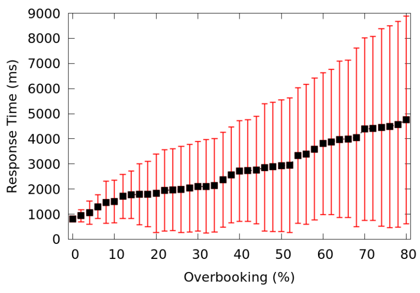
A typical scenario in which the expected performance of resources can be degraded is shown in Figure 1. In such a case, the cloud provider overbooks the node physical resources using multi-tenancy, either to maximize profits or because of operator configuration error, in a scenario commonly referred as *overbooking of resources*. In the Figure 1, several tenants (Tenant 1 to N) compete to access the node physical resources through the *hypervisor*. However, as the number of provided resources (e.g., virtualized CPU cores) is greater than the number of available physical resources, the tenants will have their performance degraded. In this case, the *hypervisor* will not be able to service all resource access requests in real-time. This leads to a possible increase in the application response time. Nonetheless, the customer is not able to identify such a scenario within the tenant perspective because he is only able to access the virtualized resources usage, but not the physical ones.

In this context, the *overbooking of resources* may significantly degrade or even inhibit the deployment of processing architectures in cloud environments. For instance,

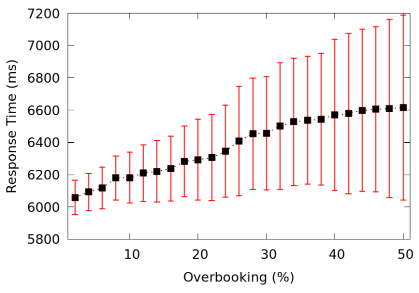
[1] CPU time that was requested by the tenant, which the *hypervisor* was not able to provide.



(a) ***CPU-bound topology***; overbooking degree is measured through CPU steal-time.



(b) **Disk-bound topology**; overbooking degree is measured through CPU wait time.



(c) **Network-bound topology**; overbooking degree is measured at hypervisor-level through the monitoring over the tenant's network usage.

Figure 3 – Multi-tenant interferences in the Apache Storm performance, showing the average and standard deviation response time; the physical resources degradation is transparent to the client in the tenant perspective.

Stream processing platforms, which often demand processing time constraints are an example of the above.

C. *Stream Processing Platform*

Big data stream (near real-time) processing platforms are distributed systems designed to process an amount of data that is unfeasible to manage with conventional processing and storage tools [14]. These distributed platforms require adequate infrastructure to support the high computational demand. Thus, they are often deployed in cloud computing environments.

The data processing in stream processing platforms (e.g., Apache Storm [15]) is performed through the definition of topologies (similar to a Hadoop job). A topology defines the processing configuration to process the data from diverse sources. A sample topology configuration shown in Figure 2 has two main components: (i) *Spouts* and (ii) *Bolts*. *Spouts* read data from sources (internal or external) and generate internal data streams. *Bolts* consume the *spouts'* data and perform related processing. *Bolts* may also consume data generated by another *bolt*.

Each unit generated by *spouts* or *bolts* is called a tuple. A topology has also several executors (threads), which are predetermined to perform tasks for the *spouts* or *bolts*. A topology in execution, formed by *spouts* and *bolts* is also known as a logical abstraction of the Apache Storm environment. When a topology is submitted (Topology Execution, Figure 2), the Storm scheduling policy distributes the topology executors in a cluster as a default configuration in a round-robin manner.

In order to achieve near real-time stream processing, Apache Storm assumes that the processing time required for processing each tuple is small. Thus, for each new tuple generated is allocated in a memory buffer until an executor is ready to process it. In order to deal with this memory buffer growth, Apache Storm employs a *backpressure* mechanism [15], which suppress the tuple generation allowing the memory buffer to decrease its size.

The *backpressure* mechanism in an *overbooking of resources* context may significantly slow the overall topology processing capacity, because, if the topology has one node that is experiencing multi-tenancy issues, the *backpressure* mechanism will suppress the tuple generation from all executors that are sending tuples to it, greatly increasing the application response time.

D. *Multi-tenancy in Stream Processing Platforms*

In this subsection, we evaluate the Apache Storm performance when deployed in a cloud computing environment with multi-tenant issues. The purpose of this evaluation is to identify the performance impact in the contexts with and without multi-tenancy issues.

Thus, to provide the desired fine-grained control over the resources, a private cloud deployment was considered through the Eucalyptus HPE version 4.2.2 [12]. The testbed is a five-node cluster in which four nodes are used as the Eucalyptus node controllers; they are responsible for the virtual machines instantiation. One of the nodes is used for the cloud infrastructure management. Each computer is equipped with an 8-core Intel Core i7 processor, 16 GB RAM, and is connected through a Gigabit Ethernet interface. For the virtual machines instantiation, the KVM *hypervisor* was used.

To achieve generality and impartiality, the evaluated workload was generated through the well-known *word count* problem [15], which has pre-defined Storm topologies and is commonly used in related literature. Three distinct Apache Storm topologies were evaluated according to their resource-bound nature: CPU, disk, and network:

- *CPU-bound Topology*: A spout generates random sentences from a book [16]; a bolt (split) divides the sentences into words; and a bolt (count) counts words occurrence.
- *Disk-bound Topology*: A spout generates random sentences from a book [17]; a bolt (split) divides sentences into words; and a bolt (count) counts words occurrence and writes them to disk.
- *Network-bound Topology*: A spout generates random messages up to 10 KB; a bolt (consumer) receives such messages (*Throughput Test*) [18].

Furthermore, all topologies spouts randomly vary the frequency of the generated messages to generate a more realistic workload. Thus, also varying the usage of the processing resources overtime. Each of the aforementioned topologies is composed of 8 spouts and 32 bolts. The processing cluster is

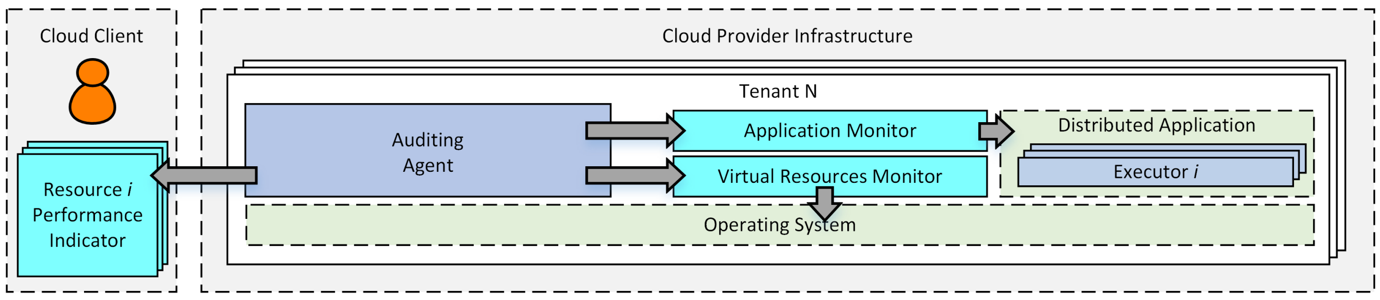


Figure 4 – Two-tier Auditing Model architecture.

composed of four nodes, which are instantiated as virtual machines (VM).

For each VM was allocated 8 cores and 8 GB of memory. To generate an *overbooking of resources* scenario, we have mirrored the processing cluster, as belonging to another cloud tenant. Thus, each physical machine hosts two virtual machines, generating an *overbooking of resources* of at most factor one (only 8 physical cores are available to 16 allocated virtual cores, while one disk and one NIC (Network Interface Card) is divided by two tenants). In order to better measure the *overbooking of resources* impact, the testbed scenario was executed for 72 h for each of the evaluated topologies.

Figure 3 shows the relation between the topologies average response time for each tuple (total processing time for each message issued from the spouts) and the *overbooking of resources* degree, regarding each of the evaluated topologies. Considering the *CPU-bound topology*, the average response time is only 2989 ms when the overbooking of resources factor is zero. However, as the overbooking degree increases, the response time also increases, increasing by up to 74% with an 80% of *overbooking of resources* degree, and also significantly increasing the variation in the response time.

The same behavior could be evidenced for each of the evaluated topologies, increasing the response time in average to 482% and 9% for the Disk-bound and Network-bound topologies, respectively. However, regardless of the average topologies response time increase, the standard deviation significantly increases. In such a case, the response time regardless of the considered topology, will significantly vary between requests.

Thus, the *overbooking of resources* poses a significant challenge for real-time applications, when deployed in cloud environments. The reason for that, is because the increase of response time, in the average, for both the load processing and the variation between such processing, may inhibit its use in the cloud context. Moreover, such processing interferences are invisible to the cloud customer, since in most cases the public cloud providers do not provide any performance metric of the physical resources.

III. TWO-TIER AUDITING MODEL

Owing to a possible conflict of interest in cloud provider and the customer, the identification of multi-tenancy issues based solely on the cloud provider metrics (e.g., CPU steal time) may be biased. The cloud provider may be either unable to measure it accurately, change or even reduce the provided resources without the cloud client consent, decreasing the application performance.

If the client does not monitor the resources performance, she is not able to know that her application is having a poor performance owing to cloud provider fault, because the client is only able to access the virtual resources performance, and does not have access to the physical ones. Nonetheless, if the client monitors only the application performance, she is not able to know if the application performance is running low owing to the cloud provider fault or because of the highly application demand for resources.

In the light of this, our proposal relies in a two-tier auditing model: *resource-based* and *application-based*. The model assumption is that it is possible to identify multi-tenancy issues through the identification of deviations between the used resources (virtualized, within the tenant domain) and the application performance. The model architecture is shown in Figure 4.

A. Architecture

In the model architecture (Figure 4), we consider the perspective of a cloud client who wishes to monitor his application running in a distributed fashion, e.g., Apache Storm (Figure 2). The proposal considers a set of tenants (virtual machines) running over a cloud provider infrastructure. Each tenant has a distributed application with a set of *executors*. In each monitored tenant, two monitors are executed, the *Application Monitor* and the *Virtual Resources Monitor*.

The *Application Monitor* is responsible to periodically collect application performance metrics from each executor, e.g., processed units in the last 10 s (see Section IV). While the *Virtual Resources Monitor* periodically collects metrics over the virtual resources within the tenant domain, e.g., CPU load (see Section IV). Periodically, both monitors send the collected metrics for an *Auditing Agent* to perform the actual resources performance auditing.

For instance, consider a cluster of tenants executing an Apache Storm topology as a distributed application. In such a case, a specific tenant may execute two spouts and three bolts, for instance. The *Application Monitor* will periodically collect application performance metrics for the five executors (two spouts and three bolts). While the *Virtual Resources Monitor,* will periodically collect a single virtual resources metrics, regarding the specific tenant. Then, periodically the performance metrics of five applications and one virtual resources are sent to the auditing agent.

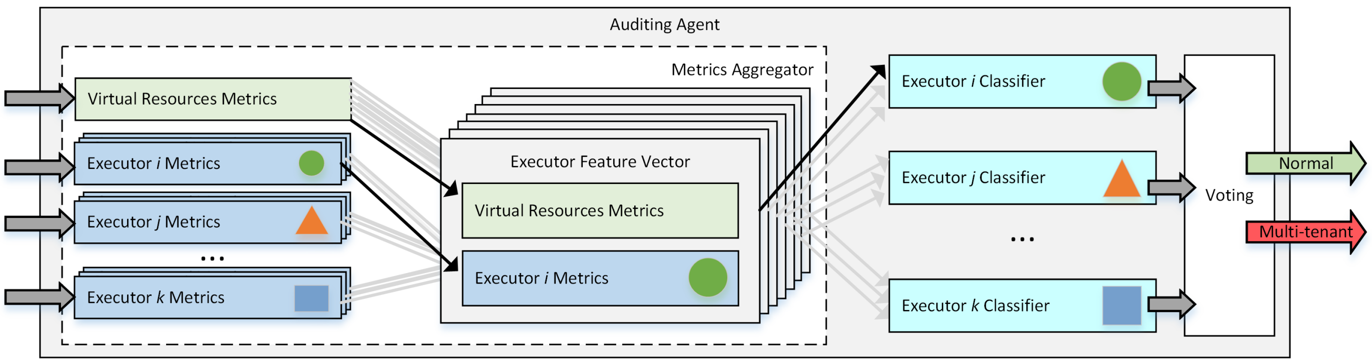


Figure 5 – Application-based Auditing Agent scheme.

The proposed architecture enables to easily scale according to the number of tenants. This because each tenant is able to monitor itself, without requiring a centralized entity, e.g., a master node. Moreover, the features are collected within each tenant domain, making it difficult for the cloud provider to provide the collected metrics incorrectly. This occurs mainly because our model relies in a two-tier auditing scheme: *resource-based* and *application-based*. Thereby, even if the cloud provider modifies the virtual resource metrics, the application-based features can still be trusted. Therefore, the customer is still able to identify performance issues.

B. *Auditing Agent*

The identification of multi-tenant interferences in distributed applications is a challenging task. This because the performance of such applications varies greatly over time, according to the generated workload (normally according to the client requests) and the monitored executors, which perform different computations when compared to their pairs. In order to enable the identification of multi-tenancy issues, the *auditing agent* employs machine learning techniques over the collected metrics from the *application monitor* and the *virtual resources monitor*.

The purpose of the *Auditing Agent* is to identify whether the last collected performance metrics were obtained in a multi-tenant interference free context (Normal) or not (Multi-tenant conditions). The *Auditing Agent* scheme is shown in Figure 5.

The agent takes the performance metrics collected by the monitors as an input. For each executor performance metric (Figure 5, Executor Metrics), the agent builds an executor feature vector composed of both the executor performance metric and the corresponding virtual resource metrics. Note that an executor feature vector is built for each collected executor performance metric.

After building the feature vectors, the classification process is performed in a component dependent manner because each component performs a different computation. Thus, the executor feature vectors are supplied to the classifier according to their component type: *spout, split,* or *bolt*. Finally, the tenant class (Normal or Multi-tenant) is assigned through a voting scheme. In this manner, the resources provided to a tenant are classified as Normal only if the majority of its executors are classified as Normal, otherwise, it is assumed to have Multi-tenant issues.

C. *Model Building Process*

The proposed model collects only information in a tenant domain, which mitigates a possible conflict of interests. However, in order to properly identify multi-tenancy issues, the classifier relies in the prior knowledge of the event classes, e.g., the executor feature vectors classes (Normal or Multi-tenant) must be previously labeled to enable the model building from the training[2] sentences in machine learning. Although some cloud providers provide performance metrics that may enable to identification of performance issues, the proposed model building process assumes that such metrics may be biased.

Our model performs the model training in a controlled environment in order to provide correct class labeling. The customers execute the application in their private cloud (controlled environment), and use the obtained model to monitor their application in the public cloud during production deployment.

During the model training process in the private cloud, the client is able to manage the physical resources usage in a fine-grained manner. In such a case, it becomes possible to evaluate the client application behavior under several scenarios of multi-tenancy issues (as shown in Section II).

Finally, with the training dataset built, the client is able to train the component's classifiers (Figure 5) and properly evaluate the system accuracy. However, in this case, with the correct event labeling, the testbed is performed in a private cloud, under the client management.

IV. Prototype

The model prototype is shown in Figure 6. A distributed stream processing framework was considered. To this end, the Apache Storm [15] was deployed in the monitored tenants (see Section II.D), in which, each tenant has slots, which vary according to the evaluated hardware configuration. Each slot holds a set of executors.

For the metrics collection process, the *application monitor* (Figure 6) periodically requests the metrics for each of the tenant's executors to the Apache Storm REST UI [19]. On the other hand, the virtual resources metrics collect the information

[2] Although unsupervised machine learning techniques can be applied, they often assume that the least occuring patterns are outliers. However, in our context it is not possible to infer the pattern classes according to their occurrences, i.e. multi-tenancy issue may occur at either none or up to all of the measured cases.

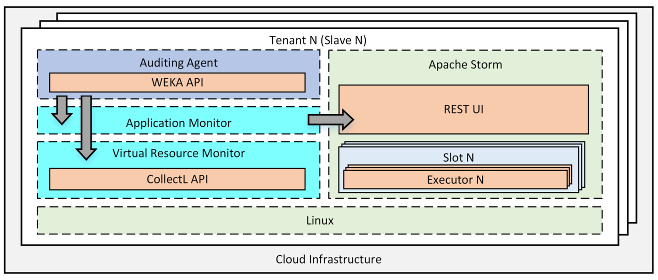


Figure 6 – Model prototype architecture.

through the CollectL API [20]. Each monitor is responsible to build the extracted features by computing the metrics in a 10 sec. interval, defined after experimental evaluation tests. A total of 7 *application-based* and 11 *virtual-resources-based* metrics are extracted; the features are shown in Table 1. The *virtual-resources-based* feature set was defined considering only the virtual resources features.

Finally, the *auditing agent* machine learning algorithm was implemented using the Weka API [21]. The agent receives the metrics collected from both monitors every 10 s and builds a Weka feature vector. The classifier training process is described in Section V.A.

V. Evaluation

Two cloud testbeds were considered during the evaluation process: private and public. For the private cloud deployment, the same testbed shown in Section II.D was considered, i.e., an Eucalyptus HPE infrastructure with four node controllers, 8-core Intel CPU, and 16 GB of memory for each node. On the other hand, the Amazon AWS [31], Google Cloud Platform [32], and Microsoft Azure Platform [33] were evaluated for the public cloud testbed.

Our evaluation aimed at answering four research questions: *(V.i) What is the minimum multi-tenancy interference needed according to each resource, for our model be able to properly classify the nodes? (V.ii) How challenging is the classification in different hardware configurations? (V.iii) How does our model perform in a public cloud environment, using the model obtained in a controlled environment (private cloud testbed)? (V.iv) How could our model be used in public cloud infrastructures without having access to neither hypervisor-level metrics nor private cloud testbeds?*

The next subsections show the model building process used in our work, and its evaluation in both private and public clouds.

A. Model Building Process

As discussed in Section III.C, the classifiers are trained through the testbed in the private cloud. Three topologies shown in Section II.D were evaluated, in which each topology is bound to a specific resource: CPU, disk, or network. Similarly, for each evaluated topology, its mirror is executed in parallel in another tenant on the same physical host. In this way, the *overbooking of resource* degree may periodically vary from a factor of zero to 1.0 (remember that each physical node hosts two tenants with eight virtual cores each). Each of the evaluated topologies was executed for 48 h. The secondary topology (executed in the

TABLE I. *Extracted Features Obtained Within the Tenant Domain*

Feature Group	Features
Application-based	Number of input tuples; Number of output tuples; Average delay; Average processed tuples per second; Difference from last number of input tuples; Difference from last number of output tuples; Difference from last average delay; Difference from last average processed tuples per second
Virtual-Resources-based	CPU load; Average CPU load last 1 min; Average CPU load last 5 min; KB read from disk; KB written to disk; Disk write requests; Disk read requests; Network packets received; Network packets sent; Network data received; Network data sent;

secondary tenants) was executed for 24 h, starting at the 12th hour until the 36th hour.

In this manner three distinct training datasets were built: CPU, disk, and network. Each dataset contains the feature vectors (Table 1) from all tenants in the corresponding scenario. For the classification process, two distinct classifiers were evaluated because of their fast classification skill: the Naïve Bayes (NB) and the Decision Tree (DT). The supervised discretization process from M. Fayadd [21] was used for the NB, while the J48 [21] algorithm was used for the DT.

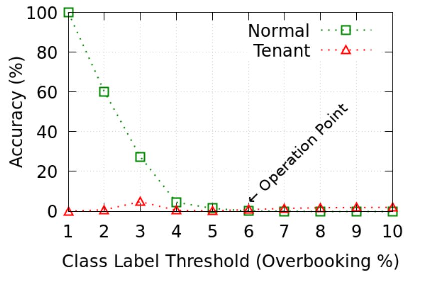
B. Private Cloud Infrastructures

To answer question *V.i*, the first evaluation aimed at defining the best threshold between the Normal and the Multi-tenant classes. To this end, the instance labels were defined according to the *overbooking of resource* degree, in which a tenant is considered Normal only if its *overbooking of resource* degree is lower than the defined threshold, else Tenant. The classifiers were trained with two nodes and tested with the remaining two nodes. Figure 7 shows the relation between the FP (rate of Normal instances wrongly classified as Tenant) and FN (rate of Tenant wrongly classified as Normal) rates.

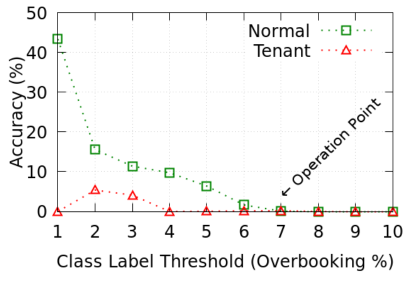
It can be noted that our two-tier auditing model is able to correctly classify Normal and Tenant nodes for all the evaluated topologies, and for both the evaluated classifiers. Regarding the CPU-bound topology, our model was able to classify when a node was experiencing more than 6 % (Figure 7.a) and 7% (Figure 7.d) of multi-tenancy issues, presenting FP rates of only 0.05% and 0.01%, and FN rates of 0.81% and 0.43 % for the NB and the DT, respectively. The same occurs for the disk-bound and network-bound topologies with a reasonable detection when the overbooking degree is over 7% for the NB classifier and 5% and 6% for the DT classifier respectively. The evaluated classifiers presented a similar performance regarding their chosen operation point.

Finally, it can be noted that the accuracy is relative to the overbooking threshold, since the application performance and the resources usage difference becomes more significant. The minimal multi-tenancy interference needed for each classifier and topology is marked as Operation Point in Figure 7, chosen when both FP and FN rates reach less than 1%.

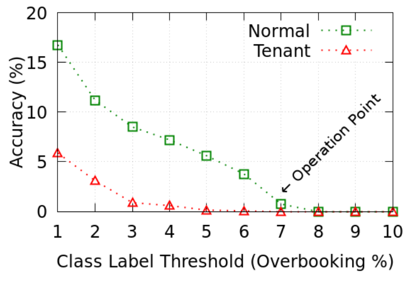
To answer question *V.ii*, we have deployed the CPU-bound topology testbed described in Section V.A with different virtual machine configurations, varying it from 8 to 1 virtual CPU



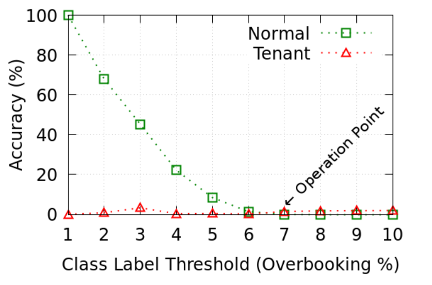
(a) NB for CPU-bound topology (8 virtual Cores)



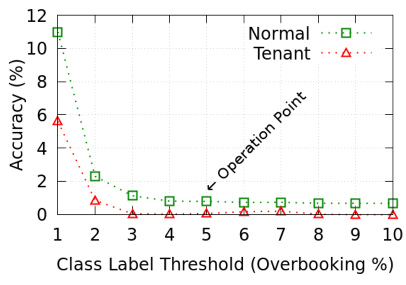
(b) NB for disk-bound topology



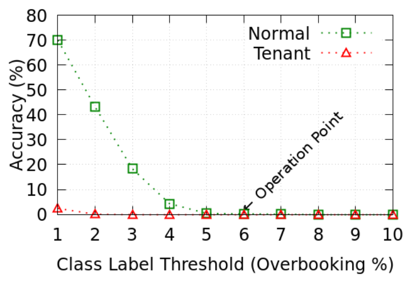
(c) NB for network-bound topology



(d) DT for CPU-bound topology (8 virtual Cores)

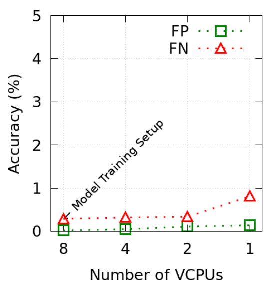


(e) DT for disk-bound topology

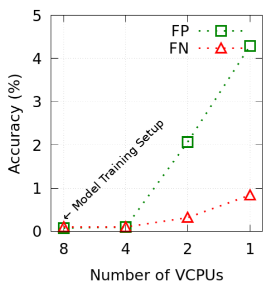


(f) DT for network-bound topology

Figure 7 – *Overbooking of resources* granularity-Accuracy tradeoff; operation points for multi-tenancy detection are marked. Operation points are defined when both Normal and Tenant accuracy rates are lower than 1%.



(a) NB for different hardware configurations

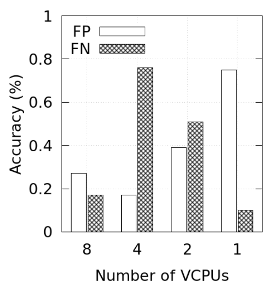


(b) DT for different hardware configurations

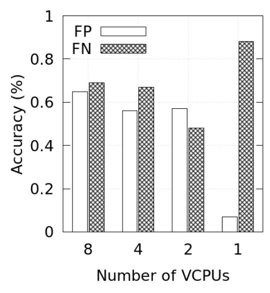
Figure 8 – Auditing model accuracy when trained with 8 virtual CPU cores and evaluated in different hardware configurations for the CPU-bound topology.

cores. Figure 8 shows the FP and FN rates with different hardware configurations for both NB and DT classifiers, considering that it was trained in the 8 virtual-core CPU testbed and evaluated in the remaining configurations. It can be noted that the NB classifier outperforms the DT classifier when a different hardware configuration is considered. Moreover, the FP and FN rates increase according to the difference between the training environment (Figure 8, 8 VCPUs) and the evaluated hardware configuration. When VMs with a single VCPU testbed are considered, the FP rate increases by 0.12% points and by 4.21% points for the NB and DT, respectively. However, when a more similar environment is considered, the FP and FN rates do not change significantly.

Thus, it becomes possible to note, that our proposed auditing model is able to perform the detection, presenting similar detection rates, even with different hardware configurations. Thus, if the hardware configuration is not changed significantly, the detection rates remains similar.



(a) NB for different hardware configurations in a public cloud testbed.



(b) DT for different hardware configurations in a public cloud testbed.

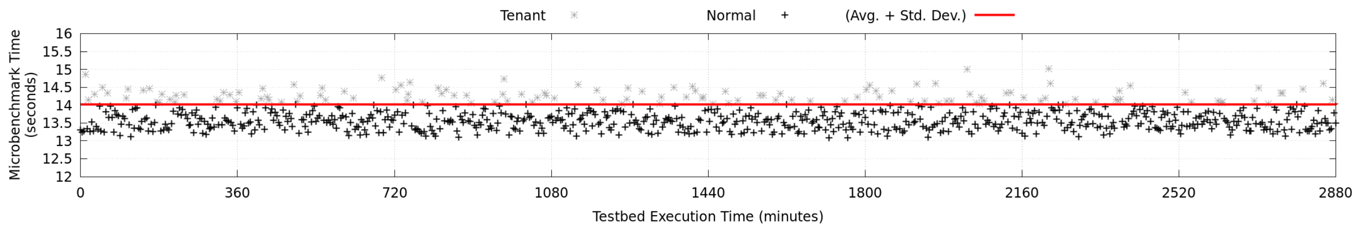
Figure 9 – Auditing model accuracy in different hardware configurations for the CPU-bound topology in the public cloud environment; the classifiers were trained in the private cloud testbed.

C. *Public Cloud with Performance Metric*

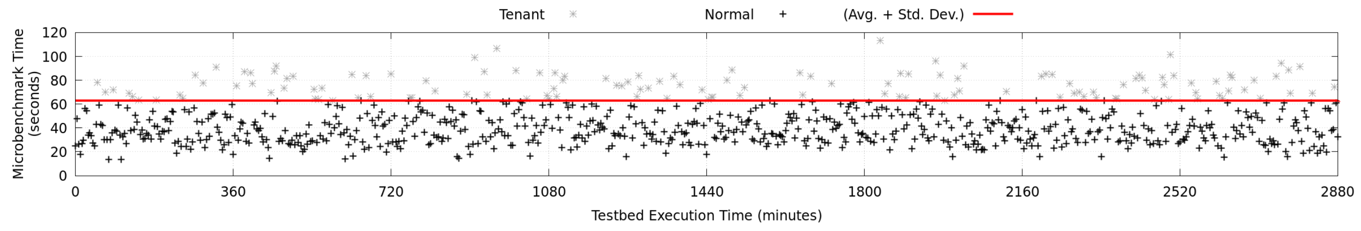
The evaluation of multi-tenant issues in public cloud environments is a challenging task. This is because the cloud client is not able to manage the physical machine resources. To answer question *V.iii* and in face of such a challenge, we have considered the evaluation of the CPU-bound topology in the Amazon AWS [31] cloud. In order to provide a ground truth (correct prior event labels) we have used the Amazon provided CPU steal time metric[3]. The same testbed described in Section V.A was deployed, but the hardware configuration was also varied.

Figure 9 shows the auditing model performance in the public cloud environment when using the model trained in the private cloud testbed. The proposed auditing model was able to generalize the application behavior from a private cloud testbed

[3] For evaluation purposes we assumed the Amazon CPU steal time is not biased. A realistic assumption considering it is a major public cloud provider.



(a) Google Cloud Platform



(b) Microsoft Azure Platform

Figure 10 – Assessment of overbooking degree through periodic CPU micro-benchmarks. Tenant class is assumed if micro-benchmark is above average plus standard deviation during the testbed execution time.

to a public one. The FP and FN rates were lower than 1% for both the evaluated classifiers in all considered hardware configurations, presenting a similar performance when compared to the private cloud testbed (Figure 8.a and 8.b).

In this manner, the proposed two-tier auditing mechanism is able to detect when an application is experiencing multi-tenancy interferences i.e., if the overbooking degree is at least more than 6% in both public and private clouds, considering different type of resources. Moreover, a system administrator is able to train the proposed mechanism in a private cloud, with a different hardware configuration, and properly monitor his application in a public cloud environment, even if the cloud provider does not provide any kind of performance metrics (Section V.D), since the FP and FN rates are lower than 1 %, and the proposed auditing scheme does not rely on any hypervisor-level metric (see Table 1).

D. Assessing Public Cloud Providers Without Hypervisor-level Metrics

As mentioned earlier, most public cloud providers do not provide any type of hypervisor-level metrics, which could be used to detect interferences caused by the sharing of physical resources in cloud computing. Nonetheless, as our scheme relies on a private cloud testbed to perform the model building process, the system administrator may have difficulties in applying such a scheme for application monitoring. In this context, the identification of multi-tenancy issues becomes a challenging task. Therefore, to enable the proper evaluation of our model in such contexts, we employed a similar approach as that applied by Schad et al. [22] for the identification of multi-tenancy interferences.

TABLE II. *SYSTEM ACCURACY WITHOUT PERFORMANCE METRICS*

Cloud Provider	**Classifier**	**Accuracy Rates (%)**	
		FP	FN
Google Cloud Platform	DT	0.00	1.30
	NB	0.00	1.88
Microsoft Azure	DT	0.48	0.00
	NB	0.50	0.00

Our assumption is that, as proposed in [22], multi-tenancy issues can be detected through micro-benchmarks. However, unlike their proposal that implies wasted processing cycles, caused by the need to perform periodic micro-benchmarks, and also to answer question *V.iv*, we conducted two tests using different cloud providers: Google Cloud Platform [32] and Microsoft Azure Platform [33]. The same experiment, with the same testbed configuration as described in section V.C, was performed for 48 h. However, during the experiments, a periodic CPU micro-benchmark was performed in parallel within the tenant. The Sysbench [34] was used as a CPU micro-benchmark tool, which computes a set of prime numbers.

Each micro-benchmark execution time was used as a measure to evaluate whether or not the tenant is experiencing multi-tenancy interferences. To decrease the testbed interference caused by the micro-benchmark execution, Apache Storm was configured to use only seven CPU cores instead of all (eight virtual CPU cores). When scheduled, we ran the micro-benchmarks in the eighth core.

As an evaluation approach, we considered the measurement class according to Eq. 1, where $bench._{current}$ denotes the current micro-benchmark time, $bench._{avg}$ the average time during the testbed execution, and $bench._{stddev}$ the standard deviation time.

$$f(class) = \begin{cases} Normal,\ bench._{current} < bench_{avg} + bench_{stddev} \\ Tenant,\ bench._{current} \geq bench_{avg} + bench_{stddev} \end{cases} \quad (1)$$

It is important to note that the approach used for the class assignment process can be customized according to the administrator needs. For instance, one could monitor the system performance over a long interval, and manually establish when the system has experienced performance degradation.

Figure 10 shows the time distribution demanded for each micro-benchmark execution, and their assigned class labels (Eq. 1), for both Google Cloud Platform (Figure 10-a) and Azure (Figure 10-b).

After the class assignment process, the first 24 h of each testbed was used for the training process, and the remaining 24 h for the test phase. Table 2 lists the FP and FN rates for the evaluated classifiers. The worst case was the NB classifier on Google Cloud Platform with an FN rate of 1.88 % and FP rate of zero.

Our proposed model was able to detect multi-tenancy interferences within the tenant domain even in the absence of hypervisor-level metrics. For production usage, our model enables the identification of multi-tenancy issues without the use of micro-benchmarks, as it presents significantly low FN and FP rates.

VI. Related Works

Performance impact in cloud computing environments have been reported by a number of works [23, 24]. Schad et al. [22] conducted a series of experiments on Amazon EC2 instances and reported a coefficient of variation of 24%, 20%, and 19% for CPU-bound, disk-bound and network-bound applications, respectively. The reported results are similar to our findings in our private cloud testbed (section II).

However, current approaches to deal with processing impact caused by overbooking of resources typically consider the cloud provider perspective. Tomás et al. [23] addressed overbooked datacenters by establishing different VM priorities in their work; high priority VMs would get pinned to a specific physical CPU, while lower ones would share unpinned CPUs. The author's approach required access to hypervisor-level features, thus was not feasible for public cloud environments. Another typical approach relies on VM migration [25]. For instance, Zhang et al. [25] designed a VM migration for over committed clouds. The authors aimed at balancing the over commitment ratio amongst the nodes. In their work, the cloud client perspective is not considered.

Moreover, when the identification of host overload is considered, it also assumes the cloud provider perspective. Anton B. and Rajkumar B. [24] aimed at identifying overloaded hosts for VM migration. The authors identified CPU overload through a utilization threshold on the physical resource. Similarly, in Breigtgand et al. [26], they overcommitted the physical nodes according to their current processing load, during the VM allocation. In their work, a node was assumed to be idle according to a simple CPU processing threshold. In this manner, both approaches are not applicable for highly variable tasks such as stream processing frameworks.

A more realistic approach was employed by Bobroff et al. [27], in which the authors relied on a time-series forecasting mechanism to minimize the number of physical hosts and SLA violations. The goal of minimization of physical hosts while providing SLA guarantees was also chased by Breigtgand and Epstein [26]. In their work, the authors identified network performance SLA violations at cloud provider level by employing a stochastic bin packing modeling over the VMs network usage. The cloud client perspective was not addressed by their works.

Some works attempt to classify the processing load in cloud computing environments [28]. In Dabbagh et al. [28] the authors attempted to predict the VMs' resources usage through a weighted sum over the recent observed utilization samples. Their approach focused on VM placement and migration at cloud provider-level. A machine learning technique was used by Segalin et al. [30] for establishing whether a VM should be reallocated to a more robust hardware or not. However, in their approach, the authors assume that the overcommit does not occur.

To the best of our knowledge, this is the first work that addresses the conflict of interest between the client and the cloud provider for over commitment detection. To this end, our work detects performance issues within the tenant domain, without relying on any of the cloud provider metrics.

VII. Concluding Remarks

The overbooking of resources in cloud computing environments is a common approach used by cloud providers, given that it is not possible to forecast each tenant demand of physical resource. Therefore, in order to maximize the hardware resource usage, the providers might favor the interference problem. Owing to a possible conflict of interest between the client and the cloud provider, the identification of this kind of issue based solely in the cloud provider metrics is a naïve approach.

This work addressed the overbooking of resources detection in the tenant domain, from the cloud client perspective. To this end, we employed a two-tier auditing model which aims at identifying deviations between the application performance and the resources usage. Our model does not rely on any metric that could be biased by the cloud provider (e.g., CPU steal time) during the detection stage.

The evaluation, using the Apache Storm as case study, has shown the feasibility of our proposal. Our model enables the system administrator to train his system in a controlled environment in a private cloud testbed and generalize it for a public cloud environment. Our model was able to reach FP and FN rates below 1% when detecting overbooking degrees of over 6%. Finally, our scheme has also enabled the monitoring of the application behavior in a public cloud environment, even in the absence of hypervisor-level metrics, and a private cloud testbed for the model building process.

Acknowledgment

This work was partially sponsored by the Brazilian National Council for Scientific and Technological Development (CNPq), grants 310671/2012-4 and 404963/2013-7. Cleverton Vicentini and Eduardo Viegas wish to thank CAPES for the scholarship, grants 1451212 and 88887.122831/2016-00, respectively.

References

[1] P. Mell, T. Grance, The NIST definition of cloud computing, National Institute of Standards and Technology, 2009. doi:10.6028/NIST.SP.800-145

[2] S. Subashini, V. Kavitha, "A survey on security issues in service delivery models of cloud computing," Journal of Network and Computer Applications, vol.34, no.1, 2011, pp. 1–11, doi:10.1016/j.jnca.2010.07.006.

[3] The Xen Project. [Online] Available: www.xenproject.org [Accessed: October 2017]

[4] KVM - Kernel-based Virtual Machine. [Online] Available: www.linux-kvm.org [Accessed: October 2017]

[5] M. Rosenblum and T. Garfinkel, “Virtual Machine Monitors: current technology and future trends,” IEEE Computer, vol. 38, no. 5, 2005, pp. 39–47, doi:10.1109/MC.2005.176.

[6] S. A. Baset, L. Wang, and C. Tang, “Towards an understanding of oversubscription in cloud,” 2nd USENIX Conf. Hot Top. Manag. Internet, Cloud, Enterp. Networks Serv. USENIX Assoc. pp. 7–7, 2012.

[7] T. Wood, L. Cherkasova, K. Ozonat, and P. Shenoy, “Profiling and modeling resource usage of virtualized applications,” in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2008, vol. 5346 LNCS, pp. 366–387.

[8] J. S. Chase, D. C. Anderson, P. N. Thakar, A. M. Vahdat, and R. P. Doyle, “Managing energy and server resources in hosting centers,” Acm Sigops, vol. 35, no. 5, p. 103, 2001.

[9] J. Seidel, O. Waldrich, W. Ziegler, R. Yahyapour, and R. Yahyapour, “Using SLA for resource management and scheduling-a survey,” Network, vol. 8, pp. 335–347, 2007.

[10] R. Weingärtner, G. B. Bräscher, and C. B. Westphall, “Cloud resource management: A survey on forecasting and profiling models,” Journal of Network and Computer Applications, vol. 47. pp. 99–106, 2015.

[11] X. Zhu, “Semi-Supervised Learning Literature Survey,” Technical Report 1530, Univ. of Wisconsin-Madison, 2006.

[12] HPE Helion Eucalyptus. [Online] Available: http://www8.hp.com/us/en/cloud/helion-eucalyptus-overview.html [Accessed: October 2017]

[13] A. Sahai, V. Machiraju, M. Sayal, A. Van Moorsel, and F. Casati, “Automated SLA monitoring for web services,” in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2002, vol. 2506, pp. 28–41.

[14] NIST. Big Data Interoperability Framework: Volume 1, Definitions, NIST Special Publication 1500-1, 2015, pp. 1-32, doi:10.6028/NIST.SP.1500-1

[15] Apache Storm, 2016. [Online]. Available: http://storm-project.net/ [Accessed: October 2017]

[16] Word Count Topology, 2013. [Online]. Available: https://github.com/apache/storm/blob/master/examples/storm-starter/src/jvm/storm/starter/WordCountTopology.java [Accessed: October 2017]

[17] J. Xu, Z. Chen, J. Tang, and S. Su, “T-storm: Traffic-aware online scheduling in storm,” in Proceedings - International Conference on Distributed Computing Systems, 2014, pp. 535–544.

[18] Storm Throughput Test, 2012. [Online]. Available: github.com/stormprocessor/storm-benchmark/blob/master/src/jvm/storm/benchmark/ThroughputTest.java [Accessed: October 2017]

[19] Storm UI REST, 2014. [Online]. Available: github.com/Parth-Brahmbhatt/incubator-storm/blob/master/storm-ui-rest-api.md [Accessed: October 2017]

[20] Collectl [Online]. Available: collectl.sourceforge.net/ [Accessed: October 2017]

[21] Weka [Online]. Available: weka.sourceforge.net [Accessed: October 2017]

[22] J. Schad, J. Dittrich, and J.-A. Quiané-Ruiz, “Runtime measurements in the cloud,” Proc. VLDB Endow., vol. 3, no. 1–2, pp. 460–471, 2010. x

[23] L. Tomas, E. B. Lakew, and E. Elmroth, “Service level and performance aware dynamic resource allocation in overbooked data centers,” 2016 16th IEEE/ACM Int. Symp. Clust. Cloud Grid Comput., pp. 42–51, 2016.x

[24] A. Beloglazov and R. Buyya, “Managing overloaded hosts for dynamic consolidation of virtual machines in cloud data centers under quality of service constraints,” IEEE Trans. Parallel Distrib. Syst., vol. 24, no. 7, pp. 1366–1379, 2013.

[25] X. Zhang, Z. Y. Shae, S. Zheng, and H. Jamjoom, “Virtual machine migration in an over-committed cloud,” in Proceedings of the 2012 IEEE Network Operations and Management Symposium, NOMS 2012, 2012, pp. 196–203

[26] D. Breitgand, Z. Dubitzky, A. Epstein, O. Feder, A. Glikson, I. Shapira, and G. Toffetti, “An adaptive utilization accelerator for virtualized environments,” in Proceedings - 2014 IEEE International Conference on Cloud Engineering, IC2E 2014, 2014, pp. 165–174.

[27] N. Bobroff, A. Kochut, and K. Beaty, “Dynamic placement of virtual machines for managing SLA violations,” in 10th IFIP/IEEE International Symposium on Integrated Network Management 2007, IM ’07, 2007, pp. 119–128.

[28] M. Dabbagh, B. Hamdaoui, M. Guizani, and A. Rayes, “Efficient datacenter resource utilization through cloud resource overcommitment,” in Proceedings - IEEE INFOCOM, 2015, vol. 2015–August, pp. 330–335.

[29] Z. Rehman, O. K. Hussain, and F. K. Hussain, “User-side cloud service management: State-of-the-art and future directions,” J. Netw. Comput. Appl., vol. 55, pp. 108–122, 2015.

[30] D. Segalin, A. O. Santin, J. E. Marynowski, and L. Segalin, “An approach to deal with processing surges in cloud computing,” in Proc. of Int. Comput. Softw. Appl. Conf., vol. 2, 2015, pp. 897–905, doi: 10.1109/COMPSAC.2015.138.

[31] Amazon AWS [Online]. Available: aws.amazon.com [Accessed: October 2017]

[32] Google Cloud Platform [Online]. Available: cloud.google.com [Accessed: October 2017]

[33] Microsoft Azure [Online]. Available: azure.microsoft.com [Accessed: October 2017]

[34] Sysbench - Modular, cross-platform and multi-threaded benchmark tool for evaluating OS parameters. [Online] Available: https://launchpad.net/sysbench [Accessed: October 2017]