Detecting Similarities in Virtual Machine Behavior for Cloud Monitoring using Smoothed Histograms

Claudia Canali, Riccardo Lancellotti (✉)
Department of Engineering “Enzo Ferrari”
University of Modena and Reggio Emilia

Corresponding author (✉):
Riccardo Lancellotti
Department of Engineering “Enzo Ferrari”
University of Modena and Reggio Emilia
Via Vignolese 905/b
Modena, 41125, Italy
Tel: +39 059 205 6256
Fax: +39 059 205 6129
Email: riccardo.lancellotti@unimore.it

Detecting Similarities in Virtual Machine Behavior for Cloud Monitoring using Smoothed Histograms

Claudia Canali, Riccardo Lancellotti (✉)

Department of Engineering "Enzo Ferrari"
University of Modena and Reggio Emilia
{*claudia.canali, riccardo.lancellotti*}*@unimore.it*

Abstract

The growing size and complexity of cloud systems determine scalability issues for resource monitoring and management. While most existing solutions consider each Virtual Machine (VM) as a black box with independent characteristics, we embrace a new perspective where VMs with similar behaviors in terms of resource usage are clustered together. We argue that this new approach has the potential to address scalability issues in cloud monitoring and management. In this paper, we propose a technique to cluster VMs starting from the usage of multiple resources, assuming no knowledge of the services executed on them. This innovative technique models VMs behavior exploiting the probability histogram of their resources usage, and performs smoothing-based noise reduction and selection of the most relevant information to consider for the clustering process. Through extensive evaluation, we show that our proposal achieves high and stable performance in terms of automatic VM clustering, and can reduce the monitoring requirements of cloud systems.

Keywords: Cloud Computing, Virtual Machine clustering, Bhattacharyya Distance, Histogram Smoothing, Spectral Clustering

1. Introduction

The cloud computing paradigm has emerged in the last few years as a way to cope with the demands of modern application exploiting virtualization techniques in large data centers. Cloud data centers based on Infrastructure as a Service (IaaS) paradigm typically host several customer applications, where each application consists of different software components (e.g., the tiers of a multi-tier Web application). Each physical server in a cloud data center hosts multiple virtual machines (VMs) running different software components with complex and heterogeneous resource demand behavior. Many customers are outsourcing services and moving their applications from internal data centers to cloud platforms exploiting *long-term commitments*, purchasing several VMs for extended periods of time (for example, integrating a data center with the Amazon so-called *reserved instances*). As this scenario is, and is expected to be in the next future, a significant part of the cloud ecosystem [1], we assume in the present study that customer VMs do not change frequently the software component they are running and that a single software component is typically deployed on several different VMs for reliability and scalability purposes.

As cloud data centers grow in size and complexity to accommodate an increasing number of customers, the process of monitoring VMs resource usage to support management strategies in cloud systems becomes a major challenge due to scalability issues. As VMs are traditionally considered as independent black boxes, management strategies require to collect information about each single VM of the data center. This means that gathering data about VMs exhibiting similar behaviors results in the collection of redundant information, thus hindering the scalability of monitoring tasks for the cloud system.

We claim that automatically clustering together VMs with similar behaviors may improve the scalability of the monitoring process. However, this approach opens novel issues about how to represent VMs behavior and to measure their similarity. The main contribution of this paper is the proposal of a technique, namely *Smoothing Histogram-based clustering* (or *SH-based clustering* for short), to group VMs showing similar behavior in a cloud data center. The proposed technique exploits histograms of resource usage to model VM behaviors, and applies a smoothing algorithm to cope with the quantization error introduced by the histogram-based representation. The VMs similarity is determined through the Bhattacharyya distance [2], that is a statistical technique measuring the similarity of discrete probability distributions. A further qualifying contribution of the *SH-based* clustering technique is the automatic selection of the specific information that is useful for the clustering process, to avoid considering data that do not carry any meaningful information and may degrade the clustering performance due to the presence of spiky or noisy behaviors.

To the best of our knowledge, the automatic clustering of VMs with similar behavior is a problem only recently analyzed in [3, 4]. In [3] clustering is based on the correlation coefficients among resource usage, which leads to highly sensitive performance with respect to the length of resource usage time series. In [4] the authors exploit an approach based on the Bhattacharyya distance requiring a separate clustering step for every VM resource, thus resulting in a non-negligible computational cost of the clustering process. On the other hand, the technique proposed in this paper outperforms the previous attempts in terms of both quality and computational cost of the clustering solution. A preliminary version of the present study was published by the authors in [5]; however, the *SH-based* proposal is a clear step ahead with respect to the original work in terms of methodological improvements (that is, use of histogram smoothing and analysis of a wider set of information to describe VM behavior) and novel experimental testbed.

We apply the proposed technique to two case studies: a first dataset coming from a cloud provider hosting VMs running Web servers and DBMS, and a second dataset obtained from a synthetic benchmark deployed on a cloud infrastructure. We show that our technique achieves high and stable performance in clustering VMs on the basis of their resource usage monitored over different time periods; in particular, the proposed clustering is effective even when the VM resource monitoring covers short periods of time (e.g., one day). Furthermore, our results demonstrate that blindly feeding every available information into the clustering process does not necessarily improve the clustering performance, demonstrating the advantage of automatically selecting relevant information.

The remainder of this paper is organized as follows. Section 2 describes the proposed technique for VM clustering. Section 3 discusses the application of the *SH-based* technique to a cloud data center. Section 4 describes the experimental testbeds used to evaluate our technique, while Section 5 presents the results of the experimental evaluation. Finally, Section 6 discusses the related work and Section 7 concludes the paper with some final remarks.

2. SH-based clustering technique

Management strategies in cloud data centers typically try to predict VM workload over a planning period of time (e.g., hours or days) based on resource usage patterns observed on past measurements, that are usually carried out with a fine granularity (e.g., 5-minute intervals) [6, 7]. Since management strategies consider each VM as a stand-alone object with independent resource usage patterns, the amount of information that needs to be collected represents a challenge for the scalability of the monitoring system.

The SH-based clustering technique aims to address this scalability issue by automatically grouping similar VMs based on resource behavior. The main goal is to cluster VMs of the same customer application which are running the same software component (e.g., VMs belonging to the same tier of a Web application), and therefore show similar behaviors in terms of resource usage. Then, the monitoring system can exploit a fine-grained data collection about few *representative* VMs as a representation of the behavior of a larger VM cluster [3].

In the rest of this section we describe and formalize the *SH-based* technique to automatically cluster similar VMs in a IaaS cloud system. The proposed technique is based on the following steps:

- Extraction of a *quantitative model* to describe the VM behavior through selected useful information
- *Smoothing* step to remove noisy contributions from the VM behavior description
- Definition of a *distance matrix* representing VMs similarities
- *Clustering* based on the distance matrix to identify classes of similar VMs

Each step is now described in detail, providing insight on the main design choices and their motivation.

2.1. VM behavior quantitative model

We now formally define the quantitative model chosen to represent the behavior of VMs and discuss some critical design choices involved in this step. We call the usage of a resource on a VM a *metric* and we use the probability distributions of the metrics to describe the VM behavior. Specifically, we represent such probability distributions using *histograms*. For formalization purposes, we consider N VMs and M metrics, so that $n \in [1, N]$ is a generic VM and $m \in [1, M]$ represents a generic metric.

We now explain how the histogram is built. Let $(\mathbf{X}_1^n, \mathbf{X}_2^n, \dots, \mathbf{X}_M^n)$ be a set of time series, where $\mathbf{X}_m^n$ is the time series consisting of the samples for metric m

on VM n. The corresponding probability density function $p(\mathbf{X}_m^n)$ is represented through normalized histograms. Each histogram consists of a specified number of *bins*, where each bin is associated to an interval of values the samples can take and represents the sample density for the interval, that is the fraction of samples in the time series falling within the interval.

If B_m is the number of bins considered for metric m, the histogram for metric m on VM n is the set $\mathbf{H}_m^n = \{h_{b,m}^n \forall b \in [1, B_m]\}$, where $h_{b,m}^n$ is the density associated to the b-th histogram bin and defined as:

$$h_{b,m}^n = \frac{|\{x \in \mathbf{X}_m^n : x > X_m^l(b), x \leq X_m^U(b)\}|}{|\mathbf{X}_m^n|}$$

where $|\{x \in \mathbf{X}_m^n : x > X_m^l(b), x \leq X_m^U(b)\}|$ is the number of samples in the range $(X_m^l(b), X_m^U(b)]$ and $|\mathbf{X}_m^n|$ is the number of samples in the time series. The bin upper and lower bounds are defined as: $X_m^l(b) = Xmin_m + (b-1)\Delta x_m$ and $X_m^U(b) = Xmin_m + b\Delta x_m$, where $Xmin_m$ is the minimum value of metric m for every VM, $Xmax_m$ is the maximum value of metric m for every VM, and Δx_m is the width of a bin for metric m, that is $\Delta x_m = \frac{Xmin_m - Xmax_m}{B_m}$. Figure 1 provides a graphical example of the above defined histogram.

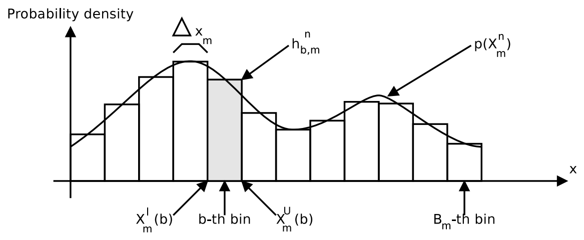


Figure 1: Histogram example

This definition ensures that for each metric m the number of bins is the same for every VM, which is required to compare histograms of different VMs. It is worth to note that different statistical techniques are popularly used to automatically estimate the number of bins of an histogram, such as Scott, Sturges and Freedman-Diaconis rules [8, 9]. In this paper, we consider the Freedman-Diaconis [9] rule, which was proven to be the best choice to generate resource usage histograms capturing VMs behavior [5]. This rule is particularly suitable to cope with samples not following a normal distribution because it makes use of the inter-quartile range of the data to determine the bin number.

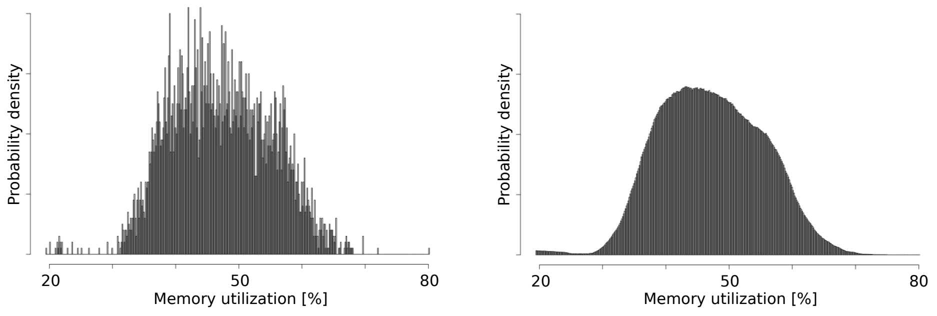
The representation of VMs behavior by means of metric histograms opens some issues related to the selection of actually useful information that may bring significant contributions to capture VMs behavior.

A first consideration is that the use of histograms takes into account the distribution of values in the time series, but may fail to capture dynamic behaviors in the resource usage. Hence, we consider to exploit additional information such as the *derivative* values extracted by the metric time series: for each histogram $\mathbf{H}_m^n$, we define another histogram $\mathbf{H}_{m'}^n$, which is built on the time series $\mathbf{X}_{m'}^n$ of the discrete derivative m' of the metric m. Such histogram aims to capture the dynamic aspects of the corresponding resource usage for the VM n. Each value in the derivative time series is computed as the difference between the corresponding value and its predecessor in the original time series.

A second consideration concerns the selection of *which* metrics are useful for actually capturing the VM behavior. A naïve approach of just feeding into the clustering process as much information as possible may be counter-productive, because non-significant data may have an effect comparable to noise and adversely affect the performance of clustering. Human intervention in the information selection process is not a viable option because it would hinder the applicability of the technique to large-scale data centers. Hence, we define an automatic process to select relevant metrics. In the management of data centers, CPU and memory are typically considered as representative VM metrics [10, 11], but we already demonstrated that they are not sufficient for VM clustering [3]. For metric selection, we rely on two statistical properties: the *autocorrelation function* (ACF) and the *coefficient of variation* (CV). We perform a first selection based on the values of the ACF of each time series: a quick decrease of the ACF means that the observed metric exhibits low (or null) autocorrelation. This is the case of metrics characterized by random perturbations and/or spikes varying in time and intensity which may be detrimental for VM clustering purposes [12]. Hence, we choose to retain metrics showing a slow decrease of the ACF, which have a strong dependency among its values. However, a slow decreasing of ACF may be caused by two conditions: (a) the metric is characterized by trends or periodical patterns that are likely to be relevant to describe the VM behavior; (b) the metric values show very low variations during the observation period, that are unlikely to be useful for capturing differences in VM behaviors. To eliminate metrics corresponding to the latter condition, we consider the CV of the metric time series. Specifically, a very low CV ($\ll 1$) indicates metrics whose values vary into very small ranges, and which do not provide any meaningful informative contribution for VM clustering.

2.2. *Histogram smoothing*

A further step in our technique is to apply a *smoothing* process to the metric histograms. Figure 2(a) shows a typical metric histogram $\mathbf{H}_m^n$ presenting an irregular shape. Irregularities are typically related to the interaction between a finite number of samples in the time series and the boundaries in the bins. Therefore, they may be considered as a quantization noise that is not useful to describe the VM behavior, and is potentially harmful for our purpose of grouping together similar VMs. We aim to remove this contribution from the VM behavior description exploiting a smoothing technique. Figure 2(b) shows the smoothed version of the original histogram in Figure 2(a). Smoothing algorithms have already been exploited in other research fields, such as artificial vision, to remove noisy components from histogram representations [13, 14]. However, smoothing has never been used in the specific context of VM behavior modeling.



(a) Non-smoothed histogram (b) Smoothed histogram

Figure 2: Histogram smoothing example

To model the weights used for the smoothing algorithm, we use a Gaussian function $G(x)$, which represents a Gaussian probability density with average value $\mu = 0$, and with standard deviation σ^2 ($\sigma^2 = 0.25$ in the example of Figure 2).

$$G(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{x^2}{2\sigma^2}} \tag{1}$$

From this Gaussian function we obtain an histogram $\mathbf{G}$. The smoothed histogram is obtained by summing the generic b-th bin of a metric histogram $\mathbf{H}^n_m$ with the values of all the nearby bins, each weighted according to the bins of the histogram $\mathbf{G}$:

$$hs^n_{b,m} = \frac{\sum_{i=0}^{B_m} h^n_{i,m} \cdot g_{i-b}}{\sum_{i=0}^{B_m} g_{i-b}}$$

where $hs^n_{b,m}$ is the b-th bin of the smoothed histogram $\mathbf{HS}^n_m$ for metric m and VM n, $h^n_{i,m}$ is the value of the original histogram bin and g_i is a bin of the histogram $\mathbf{G}$ obtained from the Gaussian function.

2.3. Distance matrix

The third step of the technique consists in building a distance matrix to define similarities among VMs starting from the smoothed histograms representing the VMs behavior. To build the distance matrix we exploit the Bhattacharyya distance [2], which measures the similarity between two datasets based on their probability distributions. The Bhattacharyya distance $D_m(n_1, n_2)$ computed according to metric m between VMs n_1 and n_2 is defined as:

$$D_m(n_1, n_2) = -ln(\sum_{b=1}^{B_m} \sqrt{hs_{b,m}^{n_1} \cdot hs_{b,m}^{n_2}})$$

where $hs_{b,m}^{n_1}$ is the b-th bin value in the smoothed histogram $\mathbf{HS}_m^{n_1}$ of metric m for VM n_1, while $hs_{b,m}^{n_2}$ refers to VM n_2. Since the histograms are normalized, the Bhattacharyya distance may take values ranging from 0 (identical histograms) to ∞ (histograms where the product of every pair of bins is 0), as shown in Figure 3.

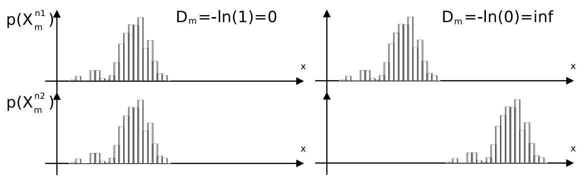


Figure 3: Bhattacharyya distance example

However, for clustering purposes a single metric is not sufficient, so we need to combine together more metrics. To this aim, we define the multimetric-based distance as the sum of squares of the distances for each metric, that is:

$$D(n_1, n_2) = \sum_{m=1}^{M} D_m(n_1, n_2)^2 a_m + \sum_{m'=1}^{M} D_{m'}(n_1, n_2)^2 a_{m'} \tag{2}$$

where $D_m(n_1, n_2)$ is the Bhattacharyya distance between n_1 and n_2 according to metrics m, and $D_{m'}(n_1, n_2)$ is the distance computed according to derivative values $m^{'}$; the boolean variables a_m and $a_{m'}$ have value 1 or 0 depending on whether metric m and its derivative $m^{'}$ are considered or not. Finally, we build the distance matrix $\mathbf{D}$ using the distances between every pair of VMs.

2.4. Clustering

The final step of the technique aims to obtain a clustering solution from the distance matrix $\mathbf{D}$. To this aim, we need to transform $\mathbf{D}$ into a *similarity* matrix $\mathbf{S}$. This step is carried out by applying a Gaussian kernel operator, that is a common approach to translate distance into similarity [15]: specifically, we define the similarity as $s_{i,j} = e^{\frac{-d_{i,j}^2}{\sigma^2}}$, where $d_{i,j}$ is an element of the distance matrix $\mathbf{D}$ and σ is a blurring coefficient of the kernel function. Preliminary analyses on the impact of the σ coefficient on the clustering results suggest that the choice of the parameter is not critical for the performance of the clustering algorithm. We choose $\sigma = 0.6$, that is a default value for Gaussian kernels in statistical software tools [16].

To cluster together elements of a set starting from a similarity matrix, traditional algorithms such as k-means or kernel k-means are not viable options because they expect as input a set of coordinates for each element to cluster. On the other hand, spectral clustering is a widely adopted solution which is explicitly designed to manage as input a similarity matrix or matrix-based representation of graphs [17].

The spectral clustering algorithm computes the Laplacian operator from the input similarity matrix $\mathbf{S}$. The eigenvalues and eigenvectors of the Laplacian are then used to extract a new coordinate system that is fed into a k-means clustering phase [18]. About this last phase of the clustering process, we must recall that the k-means algorithm starts with a random set of centroids. To ensure that the k-means result is not affected by local minimums, we iterate the k-means multiple times, then we compare the ratio between inter-cluster distances (sum of squares of distances between elements belonging to different clusters) and intra-cluster distances (sum of squares of distances between elements of the same cluster). Finally, we select the best solution across multiple k-means runs as the solution that maximize inter-cluster distances and minimize intra-cluster distances. The output of the clustering is a vector $\mathbf{C}$, where the n-th element c^n is the ID of the cluster to which VM n is assigned.

Once the clustering is complete, we need to select for each class some representative VMs that will be monitored with fine granularity. To this purpose, it is worth to note that the output of the k-means internal phase of spectral clustering provides as additional output the coordinates of the centroids for each identified cluster. In this case, few representative VMs can be selected as the VMs closest to the centroids. Specifically, we should consider that more than two representatives (at least three) are selected for each class due to the possibility that a representative unexpectedly changes its behavior with respect to its class: quorum-based techniques can be exploited to cope with byzantine failures of representative VMs [19].

3. Application of the SH-based clustering technique to cloud data centers

We now discuss how the *SH-based clustering* can be integrated in a typical IaaS cloud data center. Specifically, this section aims to demonstrate the applicability of the proposed technique, providing also an insight on the potential savings in terms of data collected by the monitoring system.

3.1. Management strategy in a cloud data center

Management strategies in IaaS cloud data centers must guarantee an efficient use of the system resources while avoiding overload conditions on physical servers. We recall that we consider a scenario where cloud customers rely on long-term commitments and we assume that VMs do not change frequently the software component they are running (e.g., changes occur with periods in the order of few weeks or months). In this scenario, we consider a cloud management strategy that consists of two separate mechanisms [20]: (a) a reactive VM relocation that exploits live VM migration when overloaded servers are detected [21];

(b) a periodic global consolidation strategy that places customer VMs on as few physical servers as possible to reduce the infrastructure costs and avoid expensive resource over-provisioning [10, 7, 22].

The *SH-based* clustering technique is essential to the scalability of the global consolidation strategy because it can group together VMs with similar behavior in terms of resource usage. The clustering technique is periodically applied to the VMs of each customer. After the clustering process, few representative VMs are selected for each identified cluster, as discussed in the previous section. To reduce the amount of data collected by the monitoring system, only the representative VMs of each class are monitored with fine granularity to collect information for the global consolidation task, while the resource usage of the other VMs of the same class is assumed to follow the representatives behavior. A quantification of the reduction in the amount of information required to support global consolidation policy is provided in the last part of this section. We do not enter in the details of the specific algorithm or periodicity used for the server consolidation because these choices depend on the workload characteristics in terms of variability and patterns. However, this does not reduce the generality of the proposed solution, because the *SH-based* clustering technique may be integrated with any consolidation strategy. As regards the non representative VMs of each class, they are monitored with coarse-grained granularity to identify behavioral drifts that could determine a change of class. At the same time, sudden changes leading to server overload are handled by the reactive VM relocation mechanism.

3.2. Reference scenario

Let us now describe an example of cloud data center exploiting the proposed technique for monitoring and management. Figure 4 depicts the reference scenario. The scheme represents a cloud data center with several physical servers, namely *host nodes*, each running several VMs. A *monitor* process on each host node periodically collects samples of the VM resources usage using the *hypervisor* APIs. The collected data are sent to the *time series aggregator* running on the host node. The time series aggregator selects the data to be communicated to the *clustering engine*, which executes the proposed technique to automatically cluster VMs, and to the *cloud controller*, which is responsible for running the consolidation strategy. Live VM migration in the case of sudden host overload [21] and programmed migrations to implement global consolidation strategies are carried out by the *local manager* on each host node.

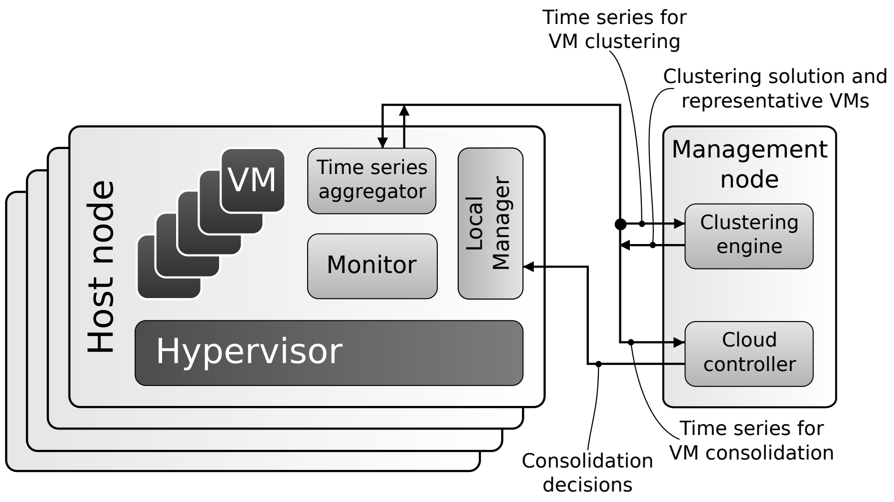


Figure 4: Cloud Data Center

We now describe the dynamics occurring in the considered cloud system to support VM clustering and server consolidation. The process of VM clustering starts from the collection of time series X_m^n describing the resources usage for each VM n and for each metric m over a certain period of time. The monitor processes are responsible for this data collection. Then, the time series aggregators of each host send the data to the clustering engine, which executes the SH-based technique to obtain a clustering solution C and selects the representative VMs for each identified class.

The information on VM classes and representatives are sent to the time series aggregators on each host node and to the cloud controller. The time series aggregators selectively collect the resource time series of the representative VMs of each class, then send the data to the cloud controller. This latter component carries out the consolidation task, exploiting the resource usage of the representative VMs to characterize the behavior of every VM of the same class. The consolidation decisions are finally communicated to the local managers on each host node to be executed.

It is worth to note that the process of VM clustering is carried out periodically with a frequency that allows to cope with changes in the VM classes (e.g. few weeks). Furthermore, the clustering may be triggered when the number of exceptions in VMs behavior exceeds a given threshold, where for exception we mean newly added VMs or clustered VMs that change their behavior with respect to the class they belong to. However, a precise determination of the activation period or strategy of the clustering process is out of the scope of this paper.

3.3. Reduction of monitored data

To understand the benefits for the monitoring system achievable through the proposed *SH-based* clustering technique, it is interesting to determine the potential reduction in the amount of data collected to support consolidation. To this aim, let us consider for example a cloud data center with 1000 VMs belonging to 5 customers, each running a three-tiered Web application (that is, resulting in 15 clusters of similar VMs).

Assuming that the global consolidation strategy considers $\overline{K}$ metrics for each VM that are collected with a frequency of 1 sample every 5 minutes, we have to manage a volume of data $288 \cdot \overline{K}$ samples per day per VM. Considering our example, we have to monitor 1000 VMs, for a total amount of data in the order of $288 \times 10^3 \cdot \overline{K}$ samples per day. After the clustering, we continue to monitor every

5 minutes only a few representative VMs per class, while the remaining VMs can be monitored with a coarse-grained granularity, for example of 1 sample every few hours. Assuming to select 3 representatives for each of the 15 VM classes, the amount of data to collect after clustering is reduced to $13 \times 10^3 \cdot \overline{K}$ samples per day for the class representatives; for the remaining 955 VMs, assuming to collect one sample of the $\overline{K}$ metrics every 6 hours for VM, the data collected is in the order of $3.8 \times 10^3 \cdot \overline{K}$ samples per day. Hence, we observe that the amount of data collected can be potentially reduced by a factor of 17, from $288 \times 10^3 \cdot \overline{K}$ to $16.8 \times 10^3 \cdot \overline{K}$.

4. Experimental testbed

To evaluate the performance of the proposed *SH-based* clustering technique we consider two case studies: (a) a dataset coming from a virtualized testbed hosting a benchmark e-business application with synthetic workload; (b) a real dataset coming from a Web-based application hosted on an enterprise data center. Let us describe in details the two case studies and the considered performance indicator.

4.1. EC2 Amazon case study

The first case study, namely *EC2 Amazon*, is based on a dataset coming from a virtualized testbed running an e-commerce application. The considered application, based on the RUBiS benchmark [23, 24], is deployed over the Amazon Elastic Computing infrastructure. The benchmark uses a PHP-based application server, a DBMS and a set of emulated browsers, issuing both HTTP and HTTPS requests. By default we use 100 emulated browsers, but in some experiments we change their number from 50 to 200 to consider different workload intensities. The benchmark is replicated on 36 VMs (we use the *micro* instances of VM provided by Amazon EC2), with 12 VMs dedicated to the emulated browsers (by default, 100 threads of emulated browsers for each VM), 12 to Web servers and 12 to DBMS. The monitoring system periodically collects samples about the usage of 13 VM resources. Each sample provides an average value computed over the period between subsequent samplings. In this scenario, the virtualized infrastructure is monitored through a framework explicitly designed for cloud platforms [25]. The complete list of the metrics collected by the monitoring system is provided in Table 1 along with a short description.

Table 1: VM metrics for EC2 Amazon case study

	Metric	**Description**
X_1	BlockOut	Rate of blocks written to storage [Blk/s]
X_2	CtxSwitch	Rate of context switches [Cs/s]
X_3	CPUIdle	CPU idle time [%]
X_4	CPUSystem	CPU utilization (syst. mode) [%]
X_5	CPUUser	CPU utilization (user mode) [%]
X_6	CPWait	CPU waiting time [%]
X_7	Interrupts	Rate of interrupts [Int/s]
X_8	MemBuff	Size of filesystem in memory (Read/Write access) [MB]
X_9	MemCache	Size of filesystem in memory (Read only access) [MB]
X_{10}	MemFree	Size of free memory [MB]
X_{11}	NetRxPkts	Rate of network incoming packets [Pkts/s]
X_{12}	NetTxPkts	Rate of network outgoing packets [Pkts/s]
X_{13}	ProcRun	Number of running processes

As the considered application is supporting a synthetic workload, the patterns of client requests are stable over time without the typical daily patterns that characterize Web traffic. For this reason, we collect samples only for 12 hours: longer time series would not provide additional information from a statistical point of view in this steady state scenario. On the other hand, having a complete control on the monitoring infrastructure allows us to change the sampling frequency for the metrics of each VM: specifically, we consider sampling frequencies ranging from 30 seconds to 5 minutes.

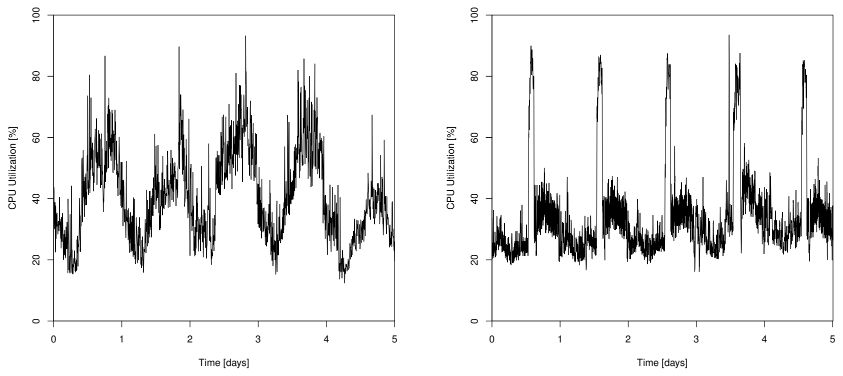
4.2. Enterprise data center case study

In the second case study, we consider 110 VMs belonging to one customer Web application which is hosted on the cloud data center and is deployed according to a multi-tier architecture. Specifically, the 110 VMs are divided in two classes: Web servers and back-end servers (that are DBMS). We collect data about the resource usage of every VM for different periods of time, ranging from 1 to 40 days. The samples are collected with a frequency of 5 minutes. For each VM we consider 10 metrics describing the usage of different resources related to CPU, memory, disk, and network. The complete list of the metrics is provided in Table 2 along with a short description.

Table 2: Virtual machine metrics

Metric		**Description**
X_1	SysCallRate	Rate of system calls [req/sec]
X_2	CPU	User CPU utilization [%]
X_3	DiskAvl	Available disk space [%]
X_4	CacheMiss	Cache miss [%]
X_5	Memory	Physical memory utilization [%]
X_6	PgOutRate	Rate of memory pages swap-out [pages/sec]
X_7	InPktRate	Rate of network incoming packets [pkts/sec]
X_8	OutPktRate	Rate of network outgoing packets [pkts/sec]
X_9	AliveProc	Number of alive processes
X_{10}	ActiveProc	Number of active processes

It is worth to note that the workload intensity considered in this case study changes dynamically over time. Figure 5(a) and 5(b) show a 5-days time series of CPU utilization for a DBMS and a Web server, respectively: both graphs clearly show the presence of daily patterns in the VMs CPU measurements.



(a) DBMS (b) Web Server

Figure 5: Time series of CPU utilization

4.3. Performance Indicator

To evaluate the performance of the proposed clustering technique, we need a measure indicating how many VMs are correctly identified. To this aim, we consider the clustering *purity*, which is one of the most popular measures for clustering evaluation [26] and represents the fraction of correctly identified VMs. Purity is determined by comparing the output $\mathbf{C}$ of the clustering algorithm with the ground truth vector $\mathbf{C}^*$, which represents the correct classification of VMs into clusters. It is worth to note that, since the spectral clustering includes an internal phase of k-means starting with a set of randomly-generated cluster centroids, in our experiments we run the final clustering step 10^3 times, then we select as the clustering output vector $\mathbf{C}$ the solution that maximizes inter-cluster and minimizes intra-cluster distances.

Purity is defined as:

$$purity = \frac{|\{c^n : c^n = c^{*n}, \forall n \in [1, N]\}|}{N}$$

where $|\{c^n : c^n = c^{*n}, \forall n \in [1, N]\}|$ is the number of VMs correctly clustered and N is the total number of VMs.

5. Performance evaluation

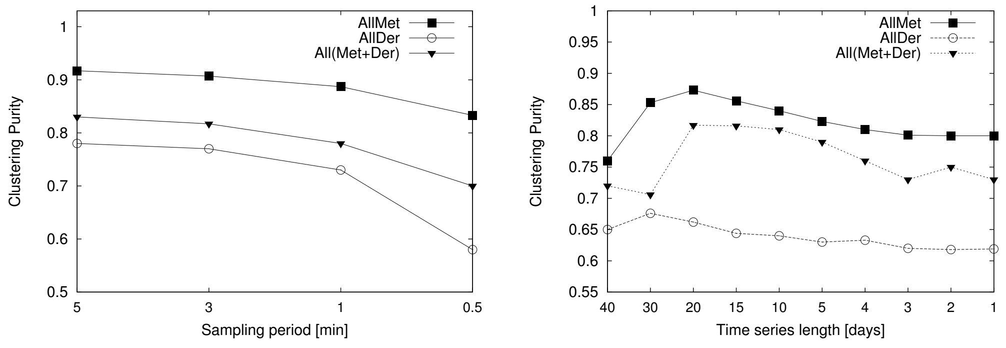
In this section we present several experiments to evaluate the effect of histogram smoothing and the performance of the *SH-based* clustering technique. We first evaluate the clustering purity achieved by the *SH-based* technique when applied to the two described case studies. This evaluation specifically aims to analyze the impact on the final clustering results of the critical design choices involved in the first two steps of the *SH-based* technique described in Section 2, which mainly concern: inclusion of derivative values, automated metric selection and application of histogram smoothing. Furthermore, we analyze the impact of variability in workload intensity on the final clustering results. Finally, we compare the performance of the *SH-based* technique with that of alternative VM clustering approaches proposed in previous studies [3, 4].

5.1. Impact of derivative values

In this first experiment we evaluate the opportunity of including derivative values of the metric time series in the computation of the Bhattacharyya distance between different VMs. To this aim, we analyzes for both case studies the clustering purity achieved by considering different information as input for the computation of the distance matrix $\mathbf{D}$ in Eq. 2. Specifically, we consider three cases: *AllMet* uses only the metric histograms to compute the Bhattacharyya distance, without derivative values (that is, $a_m = 1$, $a_{m'} = 0 \, \forall m \in [1, M]$ in Eq. 2); *AllDer* exploits only the histograms of the derivative values ($a_m = 0$, $a_{m'} = 1$); *All(Met+Der)* uses both histograms of metrics and derivative values ($a_m = a_{m'} = 1$).

Figure 6 shows the clustering purity as a function of sampling period (from 0.5 to 5 min) for the EC2 Amazon case study and of time series length (from 1 to 40 days) for the Enterprise Data Center scenario.

We observe that for both case studies the *AllDer* curve achieves very poor re-



(a) EC2 Amazon

(b) Enterprise Data Center

Figure 6: Clustering purity including derivative values

sults. The *All(Met+Der)* curve performs better than *AllDer*, but the achieved purity is always lower than *AllMet* case, showing the negative influence of including derivative values. The derivative values, that we included in the attempt to capture the dynamic behavior of VM metrics, not only fail to provide any improvement, but significantly decrease the capability of correctly clustering VMs. This poor result is explained by the fact that the use of derivative values tends to exacerbate the effect of the oscillating behaviors in the metric distributions, which do not bring any meaningful contribution to the VMs description but have a detrimental effect on the clustering results. This effect is confirmed by the analysis of the results in Figure 6(a), that is referred to the EC2 Amazon scenario. In this case we observe that the clustering purity of the *AllDer* curve significantly decreases for smaller sampling periods, because considering derivative values in case of high sampling frequency intensifies the negative effect of metric spiky behavior on the final clustering results.

A last observation is that the EC2 Amazon scenario provides a clustering purity that is generally higher if compared to the Enterprise Data Center scenario, even if the overall data collection time is limited to 12 hours. This result is motivated by the use of a synthetic workload for the EC2 Amazon case study, which is characterized by regular access patterns that increase the accuracy of the clustering algorithms.

Summarizing the results, the lesson learned by this experiment is that feeding additional information into the clustering process does not necessarily improve the achieved performance. On the other hand, it is of key importance for the clustering to select the appropriate data, which may bring significant contribution to the representation of the VM behavior.

5.2. Impact of metric selection

We now evaluate the impact on clustering performance of the automated selection of relevant VM metrics to compute the Bhattacharyya distance. The automated selection is based on the Autocorrelation Function (ACF) computed for different values of the time-lag and on the coefficient of variation (CV) of each metric. Specifically, we evaluate how fast the ACF decreases as the time-lag increases: we discard metrics having a percentage decrease of ACF greater than 50% for time-lag equal to 1 [12]. Among the remaining metrics, we operate a further selection based on the coefficient of variation (CV), discarding metrics having a value of CV $\ll 1$, as discussed in Section 2. The ACF and CV values are reported in Tables 3 and 4 for EC2 Amazon and Enterprise Data Center scenario, respectively. In particular, third and fourth columns of the tables report the percentage decrease of the values of ACF computed with time-lag equal to 1 and 5. We highlight with gray background the rows corresponding to discarded metrics, emphasizing with bold font the ACF and CV values that determine the elimination from the set of relevant metrics.

For example, in the case of the Enterprise Data Center (Table 4), we first discard metrics X_4 and X_6 because they show a quick decrease of ACF even for short time-lag (68% and 85% for time-lag equal to 1, respectively), meaning that the metrics are likely to be characterized by random perturbations and/or spikes

Table 3: Statistical properties of VM metrics for EC2 Amazon scenario

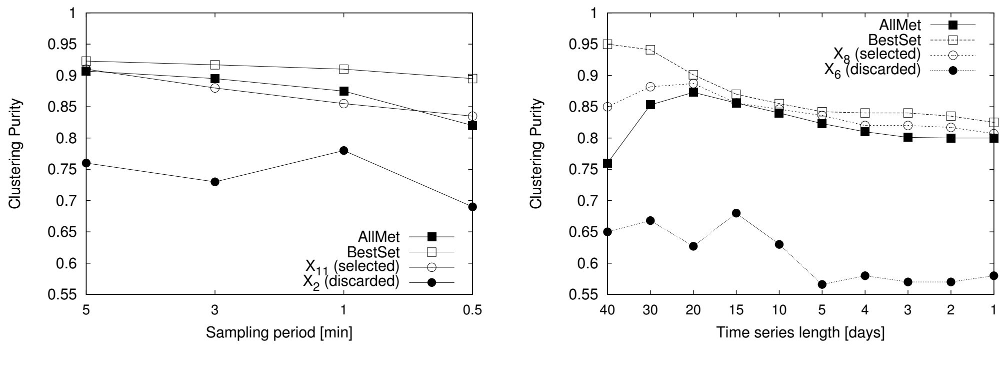
Metric		**ACF decrease**		**CV**
		Lag=1	**Lag=5**	
X_1	BlockOut	**66%**	**85%**	12.48
X_2	CtxSwitch	**57%**	**75%**	0.68
X_3	CPUIdle	15%	23%	**0.12**
X_4	CPUSystem	15%	22%	0.67
X_5	CPUUser	16%	22%	0.63
X_6	CPWait	**53%**	**74%**	20.07
X_7	Interrupts	2%	4%	**0.13**
X_8	MemBuff	4%	18%	0.35
X_9	MemCache	23%	35%	**0.08**
X_{10}	MemFree	19%	25%	1.14
X_{11}	NetRxPkts	1%	3%	0.52
X_{12}	NetTxPkts	2%	4%	0.54
X_{13}	ProcRun	**90%**	**97%**	11.76

Table 4: Statistical properties of VM metrics for Enterprise data center scenario

Metric		**ACF decrease**		**CV**
		Lag=1	**Lag=5**	
X_1	SysCallRate	11%	20%	0.87
X_2	CPU	14%	23%	1.09
X_3	DiskAvl	1%	3%	**0.17**
X_4	CacheMiss	**68%**	**81%**	0.60
X_5	Memory	5%	9%	0.54
X_6	PgOutRate	**85%**	**93%**	23.13
X_7	InPktRate	12%	19%	1.29
X_8	OutPktRate	13%	21%	1.22
X_9	AliveProc	1%	3%	**0.07**
X_{10}	ActiveProc	17%	22%	0.67

varying in time and intensity [12]. It is worth to note that metric X_6 has a CV value $\gg 1$, thus confirming that it is characterized by spiky and highly variable behavior [27]. On the other hand, metric X_4 is characterized by random perturbations that determines its lower variance (CV < 1). Then, we discard X_3 and X_9 because they have a CV $\ll 1$, indicating that these metrics show very low variations during the observation period, thus providing a not meaningful informative contribution to differentiate the behavior of VMs belonging to separated classes. The same selection process is applied to the EC2 Amazon scenario (Table 3).

To demonstrate that the selected metrics bring a relevant contribution to VM description, we now evaluate the clustering purity achievable by considering only the set of selected metrics, that we call *BestSet*, and compare it with the results obtained for the entire set of metrics, namely *AllMet*. Figures 7(a) and 7(b) show the results for EC2 Amazon and Enterprise Data Center case study, respectively. To understand the different contributions of selected and discarded metrics, for both case studies we also present the clustering purity achieved by computing the Bhattacharyya distance just on single metrics: for example, we consider the metrics X_{11} (NetRxPkts, selected) and X_2 (CtxSwitch, discarded) for the EC2 Amazon scenario, and the metrics X_8 (OutPktRate, selected) and X_6 (PgOutRate, discarded) for the Enterprise Data Center case study.



(a) Amazon EC2 (b) Enterprise Data Center

Figure 7: Clustering Purity with Metric Selection

We first observe that the use of the *BestSet* leads to two important achievements with respect to the entire set of metrics: better performance and stability of the clustering results for both case studies. Discarding the metrics based on ACF and CV values allows us to avoid the negative effect that adds variability to the performance of the *AllMet* curve. Such negative impact is particularly evident if we observe Figure 7(b) (Enterprise Data Center scenario): for almost every time series length the single selected metric X_8 achieves better results than the *AllMet* curve, whose performance is decreased by the negative effect of metrics which are discarded from the *BestSet*, such as metric X_6. It is also worth to note that the clustering purity for *AllMet* decreases for long time series (left part of Figure 7(b)). This result is apparently counter-intuitive, because it should be easier for the clustering process to correctly associate VMs to the belonging class when longer sequences of characterizing measurements are available. However, the reason of this behavior can be found in the presence of multiple local maxima (modes) in the distributions of long metric time series. The multi-modal nature of these distributions, that was pointed out by statistical analysis carried out on metric time series, tends to hinder the performance of the clustering process. However, this experiment shows that an appropriate selection of metrics significantly reduces the sensibility of the clustering results to the length of the time series.

5.3. Impact of histogram smoothing

In this experimental evaluation we aim to show the benefit of applying smoothing to metric histograms to eliminate noisy contributions and capture the similarities between VMs behavior. To this purpose, we present two experiments: first, we evaluate the change in the Bhattacharyya distance when computed on non-smoothed and smoothed histograms; second, we evaluate the impact of applying histogram smoothing on the final clustering purity of the *SH-based* technique.

5.3.1. Impact on Bhattacharyya distance

To understand how smoothing affects the Bhattacharyya distance, we consider the histograms referring to the same metric of the same VM, monitored during two subsequent periods of observation. In particular, we compare the Bhattacharyya distance computed on the non-smoothed and smoothed histograms.

Let us start with an example that considers the memory utilization (X_5) of a Web Server taken from the Enterprise Data Center case study. Figures 8(a) and 8(c) present the histograms of the memory utilization for the first and second 2-days period, respectively. Figures 8(b) and 8(d) show the corresponding smoothed histograms. It is worth to note that in this and following experiments we use a value of σ^2 equal to 0.25 for the smoothing function in Equation 1. The choice of this value is motivated by preliminary tests, not reported here for space reasons, where we evaluate the impact of this parameter on the Bhattacharyya distance of non-smoothed and smoothed histograms for multiple metrics and VMs: we found that any value of σ^2 that is $0.1 < \sigma^2 < 0.4$ provides a significant noise reduction, while preserving the main shape of the histograms.

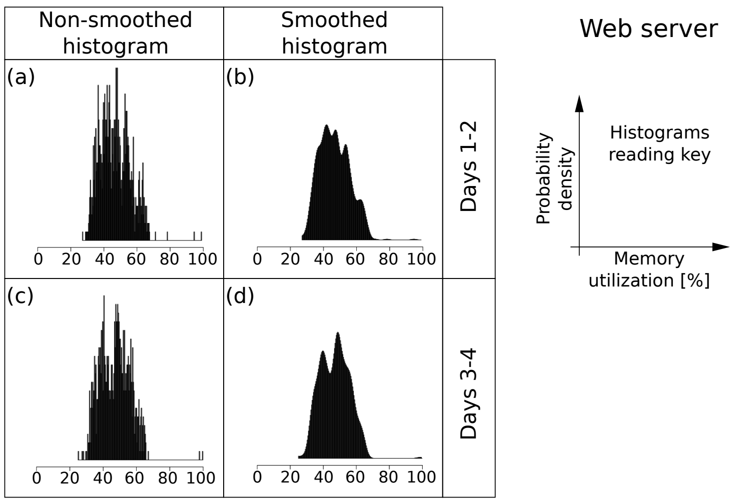


Figure 8: Histogram smoothing example

If we observe the non-smoothed histograms in the left part of Figure 8, we note that the noise due to quantization when building the histogram may hinder the possibility to capture the similarity between the resource usage in the two observed periods. This hypothesis is confirmed by the computation of the Bhattacharyya distance: the distance between the non-smoothed histograms is equal to 0.496, while it decreases to 0.297 when computed between the smoothed histograms, with a reduction of almost 40%.

We now extend our analysis to consider all the metrics included in the *BestSet* for the Enterprise Data Center scenario. For each metric, we perform the same evaluation on 110 VMs monitored for two subsequent periods of 2 days. Table 5 shows the percentage of reduction of the Bhattacharyya distance achieved when smoothing is applied to the original metric histograms. Specifically, the first and second rows of the table show for each metric the average and the standard deviation of the achieved distance reduction, respectively.

Table 5: Reduction of Bhattacharyya Distance through Histogram Smoothing

	Metric					
	SysCallRate	**CPU**	**Memory**	**InPktRate**	**OutPktRate**	**ActiveProc**
Avg	37%	46%	43%	45%	41%	34%
StdDev	8%	9%	9%	8%	7%	6%

We observe that for all the metrics of the *BestSet* the application of the histogram smoothing leads to a significant reduction of the Bhattacharyya distance between histograms resulting from subsequent observations of the same VM, with a gain ranging from 34% to 46% on average. These results confirm that the smoothing process may effectively reduce the noisy component in the metric measurements while preserving the main informative contribution of the histograms for a better determination of VM similarities.

5.3.2. Impact on clustering purity

Now we evaluate the impact of the smoothing process on the final performance of the *SH-based* clustering technique. To this purpose, we apply the smoothing technique to the histograms of the metrics included in the *BestSet*, and we refer to the results as to *BestSet-Smooth*.

Figure 9 shows the clustering purity of the considered sets of metrics with and without smoothing for the EC2 Amazon case study. We also show the results for the *AllMet* case to highlight the gain achieved through the different steps of the proposed technique.

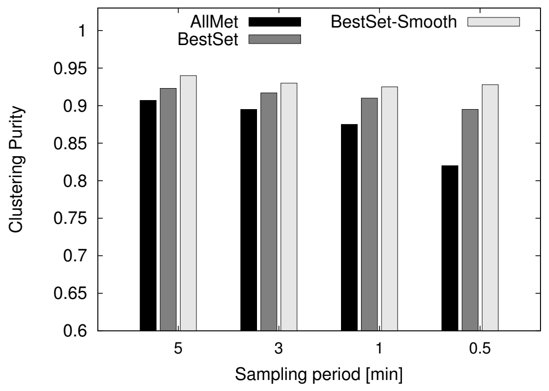
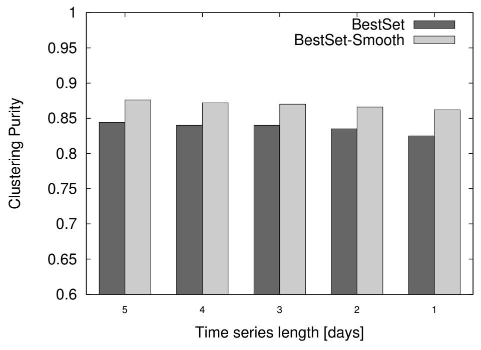


Figure 9: EC2 Amazon: clustering with BestSet smoothing

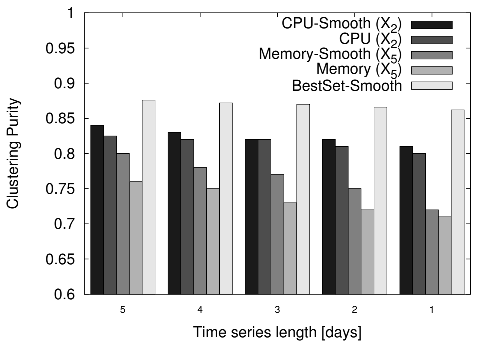
We clearly see that the application of the histogram smoothing leads to an increase in the clustering purity and a greater stability of the results with respect to the sampling frequency used to collect metric measurements, proving that reducing the contribution of the quantization noise may improve the final performance of the clustering process.

As regards the different sampling frequencies, we note that slightly better performance is achieved for a sampling period of 5 minutes (purity up to 0.94), even if the results are quite stable for every considered period. Hence, we can conclude that the *SH-based* clustering technique is suitable for being applied in systems where resource usage samples are collected every 5 minutes. Such value of the sampling period is commonly used for data monitoring in distributed systems [25, 7, 10] because it allows to reduce the amount of collected data with respect to shorter periods.

We now pass to evaluate the effect of histogram smoothing for the Enterprise Data Center scenario. Fig. 10(a) presents the comparison between *BestSet-Smooth* and *BestSet* focusing on short time series, ranging from 5 to 1 days. The graph confirms that the use of smoothed histograms may actually improve the performance of the clustering process, with an increase of the clustering purity up to 5% for the shortest time series.



(a) BestSet vs. BestSet-Smooth



(b) BestSet-Smooth vs. single metrics

Figure 10: Enterprise Data Center: clustering with BestSet smoothing

We also compare the clustering purity of the *BestSet-Smooth* approach with the results obtained by considering just the single metrics X_2 (CPU) and X_5 (Memory), with and without smoothing, as shown in Fig. 10(b). We consider these metrics because they are typically the only ones considered in cloud data center management [28, 29, 7]. Figure 10(b) shows the small benefit achievable by applying the smoothing technique to each single metric. However, results in Fig. 10(b), together with non reported experiments considering the other single metrics composing the *BestSet*, confirm that the clustering purity of the *BestSet-Smooth* outperforms the results of each single smoothed metric. This means that the use of single metrics, including CPU and memory, is not so efficient to capture the VMs behavior for clustering purposes. Hence, this experiment confirms the need to consider multiple metrics to correctly characterize VM behavior, but also demonstrates the importance of removing from the VM behavior description the useless data which may decrease the clustering performance, such as the noisy contribution removed through the smoothing process.

5.4. *Impact of workload variability*

We now aim to evaluate the impact on clustering results of workload variability. The experiments carried out on the Enterprise data center scenario already show that the proposed clustering technique may handle the inherent variability of daily patterns. We now aim to further investigate the impact of workload variability by considering aperiodic workload changes for the EC2 Amazon case study. To this purpose, we consider a number of emulated browsers ranging from 50 to 200 (we recall that 100 is the default value used in previous experiments): for each considered workload intensity we collect samples for 12 hours, then we apply the clustering technique. Moreover, we create a *dynamic workload* where the number of emulated browsers grows from 50 to 200 during a single collection period of 12 hours.

Table 6 shows the clustering purity achieved by the proposed technique, referred as *BestSet Smoothed*, in this experiment. For sake of comparison, the table also shows the results of the *BestSet Not Smoothed* clustering.

Table 6: Clustering purity for different workload intensities

Clustering	**Number of Emulated Browsers**					
Approach	**50**	**75**	**100**	**150**	**200**	**Dynamic**
BestSet Smoothed	0.943	0.938	0.940	0.936	0.942	0.949
BestSet Not Smoothed	0.922	0.918	0.921	0.913	0.911	0.901

We observe that the proposed technique shows a great stability for every workload intensity, and confirms the gain over the not smoothed approach. For the dynamic workload (last column of Table 6), we note a slightly higher gain of the smoothed approach, close to 5%. This result may be explained by considering the multi-modal shape that characterizes the histograms of the VM metrics when a dynamic workload is applied: in this case the impact of the histogram quantization noise is likely to worsen the clustering results in absence of a corrective smoothing process. From this experiment we conclude that the proposed technique is able to cope with variability in the workload, as long as the arriving requests are evenly distributed among the VMs of the same cluster, for example through load sharing mechanisms.

5.5. Comparison with existing VM clustering approaches

Recent studies [3, 4] of the authors propose alternative techniques to automatically cluster VMs in cloud systems depending on their behavior. In [3] we exploit the correlation coefficients between resource usage time series to determine VM similarities, while in [4] we rely on cluster ensemble techniques. Specifically, in the latter study we compute the Bhattacharyya distance and perform a clustering step for every single VM metric, then we apply a quorum-based ensemble technique to determine a global similarity matrix between every pair of VMs, and finally apply a last step of clustering.

Both these techniques present strengths and weaknesses. The computational cost of the *Correlation-based* approach [3] is relatively small, requiring the simple computation of the correlation matrix to determine the behavioral representation of each VM; however, the clustering purity decreases rapidly for short metric time series (that is, collected for few days) as well as in presence of time periods during which VMs are idle. On the other hand, the *Ensemble-based* approach [4] presents high clustering purity at the price of high computational costs due to histograms computation and multiple executions of the clustering step. The *SH-based* technique proposed in this paper aims to achieve a clustering purity which is stable for different time series and comparable with the *Ensemble-based* approach; as regards its computational cost, it is higher with respect to the *Correlation-based* approach due to the smoothed histograms computation, but it is expected to be lower than that of the *Ensemble-based* technique thanks to the presence of a single clustering step.

We now provide a quantitative comparison of VM clustering techniques in terms of performance and computational cost by applying our proposal and existing approaches to the Enterprise Data Center scenario. Table 7 shows the clustering purity achieved by the three clustering techniques for time series length ranging from 10 to 1 days.

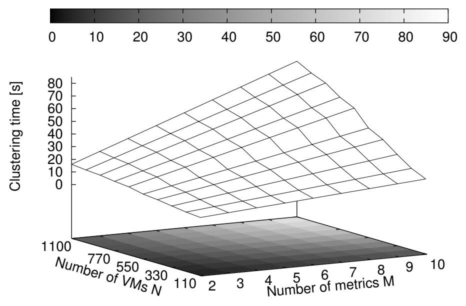
Table 7: Clustering purity of alternative clustering techniques

Clustering technique	**Time series length [d]**					
	10	**5**	**4**	**3**	**2**	**1**
Correlation-based	0.809	0.797	0.771	0.752	0.729	0.714
Ensemble-based	0.864	0.862	0.861	0.852	0.850	0.841
SH-Based	**0.880**	**0.876**	**0.872**	**0.870**	**0.867**	**0.863**

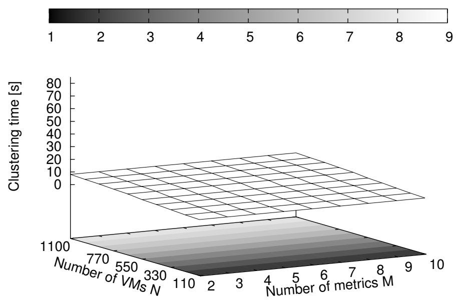
We observe that the proposed *SH-Based* technique shows higher clustering purity with respect to both alternatives for every time series length. With respect to the *Correlation-based* approach, the gain in clustering purity increases for short time series, exceeding 15% for 1 day time series. This effect is due to the rapid degradation of the clustering purity for short time series of the *Correlation-based* approach, as already noticed in [3]. With respect to the *Ensemble-based* approach, the *SH-based* technique achieves similarly stable and slightly higher performance thanks to the application of automated metric selection and histogram smoothing. However, the main advantage of the proposed technique over the *Ensemble-based* approach is the lower computational cost, as discussed below.

In this last experiment we compare the execution time of the clustering *SH-Based* and *Ensemble-based* techniques. We limit the evaluation of the execution time to the clustering phase of the techniques, because it is the only centralized step that cannot be easily distributed and parallelized on multiple nodes. Moreover, it is worth to note that the additional cost of histogram smoothing to determine the behavioral representation of each VM in the *SH-Based* approach is negligible with respect of the initial creation of the histogram itself.

The clustering execution time is evaluated with respect to the number of considered metrics and VMs to cluster: for each of the 110 VMs of the Enterprise Data Center scenario we consider metric time series with the length of one day (24 hours): in this way, we emulate the presence of an increasing number of VMs to cluster, ranging from 110 to 1100. Fig. 11 shows the clustering time as a function of the number of metrics and VMs. It is worth to note that, to carry out a fair comparison, we consider for the two approaches the same number of metrics, even if the *SH-Based* actually limits the metrics to a selected set, while the *Ensemble-based* typically exploits a greater number of metrics.



(a) Ensemble-based



(b) SH-based

Figure 11: Ensemble-based vs Smoothed-BestSet Clustering Time

The execution time required by the *Ensemble-based* approach reaches values that are one order of magnitude higher with respect to the proposed technique when 10 metrics are considered, as in [4]. The high computational costs of the *Ensemble-based* approach are due to multiple executions of the clustering step, which is performed $M + 1$ times, where M is the number of considered metrics. On the other hand, the *SH-Based* approach performs the clustering step just once, so the execution time of this step does not depend on the number of considered metrics.

6. Related Work

Scalability issues concerning resource monitoring and management in cloud systems have received a lot of attention by academic and industrial research in the last few years.

Current solutions for monitoring large data centers typically exploit frameworks for periodic collection of system status indicators. Most frameworks for multi-cloud monitoring rely on standard building blocks such as Ganglia[1], Cacti[2] or Munin[3]. However, monitoring a large number of VMs remains a challenge unless some technique is used to reduce the amount of data to collect and store, as done by the *SH-based* clustering technique proposed in this paper, which explicitly addresses this problem.

Similar scalability issues can be observed for cloud management. Most management solutions are inherited from literature on distributed systems, and can be divided in two categories: reactive on-demand solutions that can be used to avoid and mitigate server overload conditions, and periodic solutions that aim to consolidate VMs exploiting optimization algorithms. The two approaches can be combined together [20]. Examples of reactive solutions are [28] and [29], that propose a mechanism based on adaptive thresholds regarding CPU utilization values. A similar approach is described also in Wood *et al.* [21] with a rule-based approach for live VM migration that defines threshold levels about the usage of few specific physical server resources, such as CPU-demand, memory allocation, and network bandwidth usage. We believe that this type of solution can be in-

[1]http://ganglia.sourceforge.net/

[2]http://www.cacti.net

[3]http://munin-monitoring.org/

tegrated in our proposal at the level of the local managers on each host node. An example of periodic VM consolidation solutions is proposed by Kusic *et al.* in [22], where VM consolidation is achieved through a sequential optimization approach. Similar solutions are proposed in [10, 11]. However, these approaches are likely to suffer from scalability issues in large scale distributed systems due to the amount of information needed by the optimization problem. Solutions like our proposal, aiming to reduce the amount of data to collect and consider for the management of cloud data centers, may play a major role for the applicability of consolidation strategies to large cloud systems.

Solutions that exploit clustering to improve scalability of monitoring in cloud and multi-cloud systems have been recently proposed by the authors [3, 4, 5]. In [3] we rely on correlation coefficients between resource usage to determine VM similarities, while the proposal in [4] exploits cluster ensemble techniques to improve accuracy and stability of the results. However, the technique based on correlation provides poor performance for short metric time series (few days), while the ensemble-based approach has high computational costs due to multiple executions of the clustering step. Section 5.5 shows a quantitative analysis of the gain achieved by the *SH-based* proposal both in terms of performance and computational cost. Finally, in [5] we present a first application of the automatic metric selection approach to a multi-cloud context. This paper represents a major improvement with respect to the preliminary study in [5] because it proposes a way to reduce the impact of quantization noise on the computation of the Bhattacharyya distance thanks to the histogram smoothing. Furthermore, this paper presents a more extensive experimental evaluation of the clustering technique on multiple workloads, and sensitivity analysis with respect to sampling frequency and inclusion of derivative values besides metrics time series.

Prior studies proposing clustering of cloud VMs are rather limited. Zhang *et* *al.* [30] propose a method for VM clustering in cloud systems. However, they consider only storage resources in order to perform storage consolidation strategies, while our proposal focuses on the more general problem of global server consolidation and takes into account multiple resources to describe VMs behavior. The study in [31] investigates similarities in VM images used in public cloud environments. While this study focuses on the static images of cloud VMs to provide insights for deduplication and image-level cache management, our approach leverages similarities in the dynamic VMs behavior to improve scalability of cloud monitoring.

One of the main research area where clustering based on statistical representation of features has been applied is the field of image processing and computer vision. For example, the use of histograms and Bhattacharyya distance has been exploited to compare the similarity of images [32]. In a similar way, smoothing techniques on histograms have been proposed for image processing as a way to improve image identification and retrieval [13, 14]. However, the application of such techniques to cloud computing is a major innovation that opens new problems, such as the need for VM metric selection, that are specific to this application realm.

7. Conclusions

We propose a novel technique, namely *SH-based*, for automatic clustering of VMs sharing similar behavior to improve the scalability of monitoring process in cloud data centers. The proposed technique exploits the Bhattacharyya distance to determine similarities among VMs based on their resource usage and introduces a smoothing process to remove noise from VM behavior representation. Our proposal also considers multiple VM metrics and automatically selects which of them actually bring a meaningful contribution to the clustering process. We evaluate the performance of the *SH-based* technique using two case studies, and we discuss the potential benefit in terms of reduction of the amount of data collected by the cloud monitoring system. Our experiments show that the purity achieved by automatic VMs clustering ranges between 95% and 86% for every considered scenario. Furthermore, we compare the achieved results with those of alternative approaches to VM clustering, showing how the proposed technique surpasses the existing alternatives in terms of both clustering quality and computational cost.

References

[1] D. Durkee, Why cloud computing will never be free, Queue 8 (4) (2010) 20:20–20:29.

[2] A. Bhattacharyya, On a measure of divergence between two statistical populations defined by their probability distributions, Bulletin of the Calcutta Mathematical Society 35 (1943) 99–109.

[3] C. Canali, R. Lancellotti, Automated Clustering of VMs for Scalable Cloud Monitoring and Management, in: Proc. of Conference on Software, Telecommunications and Computer Networks (SOFTCOM), Split, Croatia, 2012.

[4] C. Canali, R. Lancellotti, Exploiting ensemble techniques for automatic virtual machine clustering in cloud systems, Automated Software Engineering (2013) 1–26 Available online. doi:10.1007/s10515-013-0134-y.

[5] C. Canali, R. Lancellotti, Automatic virtual machine clustering based on Bhattacharyya distance for multi-cloud systems, in: Proc. of International Workshop on Multi-cloud Applications and Federated Clouds, Prague, Czech Republic, 2013, pp. 45–52.

[6] B. Addis, D. Ardagna, B. Panicucci, M. Squillante, L. Zhang, A hierarchical ap-

proach for the resource management of very large cloud platforms, Dependable and Secure Computing, IEEE Transactions on 10 (5) (2013) 253–272.

[7] T. Setzer, A. Stage, Decision support for virtual machine reassignments in enterprise data centers, in: Proc. of Network Operations and Management Symposium (NOMS'10), Osaka, Japan, 2010.

[8] D. W. Scott, On Optimal and Data-Based Histograms, Biometrika 66 (3) (1979) 605–610.

[9] D. Freedman, P. Diaconis, On the histogram as a density estimator:L2 theory, Probability Theory and Related Fields 57 (4) (1981) 453–476.

[10] D. Ardagna, B. Panicucci, M. Trubian, L. Zhang, Energy-Aware Autonomic Resource Allocation in Multitier Virtualized Environments, IEEE Trans. on Services Computing 5 (1) (2012) 2 –19.

[11] C. Tang, M. Steinder, M. Spreitzer, G. Pacifici, A scalable application placement controller for enterprise data centers, in: Proc. of 16th World Wide Web Conference (WWW'07), Banff, Canada, 2007.

[12] Y. Baryshnikov, E. Coffman, G. Pierre, D. Rubenstein, M. Squillante, T. Yimwadsana, Predictability of Web-Server Traffic Congestion, in: Proc. of IEEE Workshop on Web Content Caching and Distribution (WCW), Sophia Antipolis, France, 2005.

[13] J.-H. Chang, K.-C. Fan, Y.-L. Chang, Image and Vision Computing 20 (3) (2002) 203 – 216.

[14] J. Kautsky, N. K. Nichols, D. L. Jupp, Smoothed histogram modification for image processing, Computer Vision, Graphics, and Image Processing 26 (3) (1984) 271 – 291.

[15] I. S. Dhillon, Y. Guan, B. Kulis, Kernel k-means: spectral clustering and normalized cuts, in: Proc. of International Conference on Knowledge Discovery and Data Mining, Seattle, USA, 2004.

[16] A. Karatzoglou, A. Smola, K. Hornik, A. Zeileis, kernlab - An S4 package for kernel methods in R, Tech. Rep. 9, WU Vienna University of Economics and Business (Aug 2004).

[17] A. K. Jain, Data clustering: 50 years beyond K-means, Pattern Recognition Letters 31 (8) (2010) 651 – 666.

[18] U. Luxburg, A tutorial on spectral clustering, Statistics and Computing 17 (4) (2007) 395–416.

[19] M. Castro, B. Liskov, Practical Byzantine Fault Tolerance, in: M. I. Seltzer, P. J. Leach (Eds.), OSDI, USENIX Association, 1999, pp. 173–186.

[20] Z. Gong, X. Gu, PAC: Pattern-driven Application Consolidation for Efficient Cloud Computing, in: Proc. of Symposium on Modeling, Analysis, Simulation of Computer and Telecommunication Systems, Miami Beach, 2010.

[21] T. Wood, P. Shenoy, A. Venkataramani, M. Yousif, Black-box and gray-box strategies for virtual machine migration, in: Proc. of Conference on Networked systems design and implementation (NSDI), Cambridge, 2007.

[22] D. Kusic, J. O. Kephart, J. E. Hanson, N. Kandasamy, G. Jiang, Power and Performance Management of Virtualized Computing Environment via Lookahead, Cluster Computing 12 (1) (2009) 1–15.

[23] E. Cecchet, A. Chanda, S. Elnikety, J. Marguerite, W. Zwaenepoel, Performance comparison of middleware architectures for generating dynamic Web content, in: Proc. of 4th Middleware Conference, 2003.

[24] L. Hu, K. Schwan, A. Gulati, J. Zhang, C. Wang, Net-cohort: detecting and managing VM ensembles in virtualized data centers, in: Proc. of the 9th international conference on Autonomic computing (ICAC '12), ICAC '12, ACM, San Jose, California, USA, 2012, pp. 3–12.

[25] M. Andreolini, M. Colajanni, M. Pietri, A scalable architecture for real-time monitoring of large information systems, in: Proc. of IEEE Second Symposium on Network Cloud Computing and Applications, London, UK, 2012.

[26] E. Amigó, J. Gonzalo, J. Artiles, F. Verdejo, A Comparison of Extrinsic Clustering Evaluation Metrics Based on Formal Constraints, Journal of Information Retrieval 12 (4) (2009) 461–486.

[27] S. Casolari, S. Tosi, F. Lo Presti, An adaptive model for online detection of state changes in Internet-based systems, Performance Evaluation 69 (5) (2012) 206–226.

[28] A. Beloglazov, R. Buyya, Adaptive Threshold-Based Approach for Energy-Efficient Consolidation of Virtual Machines in Cloud Data Centers, in: Proc. of MGC Workshop, Bangalore, India, 2010.

[29] D. Gmach, J. Rolia, L. Cherkasova, A. Kemper, Resource pool management: Reactive versus proactive or let's be friends, Computer Networks 53 (17).

[30] R. Zhang, R. Routray, D. M. Eyers, et al., IO Tetris: Deep storage consolidation for the cloud via fine-grained workload analysis, in: IEEE Int'l Conference on Cloud Computing, Washington, DC USA, 2011.

[31] K. R. Jayaram, C. Peng, Z. Zhang, M. Kim, H. Chen, H. Lei, An empirical analysis of similarity in virtual machine images, in: Proc. of the Middleware 2011 Industry Track Workshop, Middleware'11, ACM, Lisbon, Portugal, 2011, pp. 6:1–6:6.

[32] O. Michailovich, Y. Rathi, A. Tannenbaum, Image Segmentation Using Active Contours Driven by the Bhattacharyya Gradient Flow, Trans. on Image Processing 16 (11) (2007) 2787–2801.