Noname manuscript No.
(will be inserted by the editor)

A Technique to Identify Data Exchange between Cloud Virtual Machines

Nicola Bicocchi · Claudia Canali · Riccardo Lancellotti

the date of receipt and acceptance should be inserted later

Abstract Modern cloud data centers typically exploit management strategies to reduce the overall energy consumption. While most of the solutions focus on the energy consumption due to computational elements, the optimization of network-related aspects of a data center is becoming more and more important, considering also the advent of the Software-Defined Network paradigm. However, an enabling step to implement network-aware Virtual Machine (VM) allocation is the knowledge of data exchange patterns. In this way we can place in well-connected hosts (or on the same physical host) the couples of VMs that exchange a large amount of information. Unfortunately, in Infrastructure as a Service data centers, a detailed knowledge on VMs data exchange is seldom available without the deployment of a specialized (and costly) monitoring infrastructure. In this paper, we propose a technique to infer VMs communication patterns starting from input/output network traffic time series of each VM. We discuss both the theoretical aspect of such technique and the design challenges for its implementation. A case study is used to demonstrate the viability of our idea.

Keywords Cloud Computing, Communication Patterns, Correlation Coefficient, Gossip Protocol

1 Introduction

Resource management in cloud data centers is a critical task to reduce energy consumption in IT infrastructures. A large corpus of literature focuses just on reducing the number of powered-on hosts [5,7,21]. However, a more advanced approach to the problem aims at taking into account energy expenditures for data exchanges, for example by placing highly interacting VMs ont he same physical host [20,9].

This evolution towards a network-aware VMs allocation is even more evident as certain trends, such as software-defined networks, become widespread. However, network-aware allocation requires to map network interactions between each couple of VMs in a data center. In the present study, we assume the point of view of an Infrastructure as a Service (IaaS)

Department of Engineering "Enzo Ferrari"
University of Modena and Reggio Emilia
Via P. Vivarelli 10
41125, Modena, Italy
E-mail: {nicola.bicocchi, claudia.canali, riccardo.lancellotti}@unimore.it

cloud provider. In such vision, VMs are just opaque objects that do not offer any insight on the software they are running. Hence, it is unlikely to have a data traffic matrix between the VMs, relying just on monitoring services in industry[1] or proposed in scientific literature [3]. Indeed, most monitoring systems just provide the input/output traffic rate of each VM, without a per-source/per-destination breakdown. A specialized monitoring infrastructure able to provide a complete traffic matrix among VMS may be developed and deployed over the data center, as in [22,25]; however, in this case overhead and scalability are likely to represent an issue. This approach, indeed, involves on one side the overhead of dedicated resources of the infrastructure devoted to the monitoring of every communication between couples of VMs, and on the other side the presence of huge amount of data to be stored and processed, that would limit the scalability of the system [22]. For this reason, in this book chapter we propose a solution that is not relying on the presence of a data traffic matrix but is able to infer communication patterns starting from the input/output network time series of each VM.

The contribution of this chapter is twofold. First, we outline a methodology that accepts as the input the time series of the number of network packets inbound and outbound for each VM, and provides information about which VMs intensively communicate with each other. The proposed methodology, which was initially proposed in [8,17], is explicitly tailored to cope with the most challenging scenarios, that is the case where multi-tier applications are deployed over a cloud infrastructure relying on vertical replication. Second, we discuss how such methodology can be implemented adopting solutions, such as gossip protocols and distributed algorithms, that can provide high scalability even in large data centers. A qualifying point of our proposal is exploiting correlation of data traffic time series to infer the interaction between VMs. Our study compares multiple correlation metrics to identify the best solution to cope with the scenario of replicated multi-tier applications. Our experiments, carried out using a benchmark application over a real cloud infrastructure, demonstrate the viability of our approach to discover interacting VMs within a cloud data center.

The remainder of this paper is organized as follows. Section 2 describes the reference scenario of this paper. Section 3 models the problem and outlines the proposed methodology. Section 4 details an implementation of the methodology making use of distributed agents and a gossiping protocol. Section 5 describes the experimental results used to validate our proposal, while Section 6 discusses the state of the art in scientific literature. Finally, Section 7 concludes the paper and details final remarks.

2 Reference Scenario

We now outline the reference scenario for this chapter that is depicted in Figure 1.

The upper part of Figure 1, namely *Virtual view*, represents how the cloud infrastructure is perceived by the cloud customer. We assume to have a collection of VMs interacting among themselves to support a multi-tier application. The application is deployed over s replicated vertical stacks. Each vertical stack contains the r tiers of the application, hosted on separate VMs. The communication patterns in this scenario involve just VMs within the same vertical stack. No communication occurs between VMs belonging to two different vertical stacks. We assume that incoming request load is balanced across the vertical stacks. Hence, VM belonging to different vertical stacks but to the same tier are likely to show similar network traffic patterns (referring to Fig. 1 this may occur, for example, for $\mathrm{VM}_{1,1}$

[1] https://aws.amazon.com/cloudwatch/

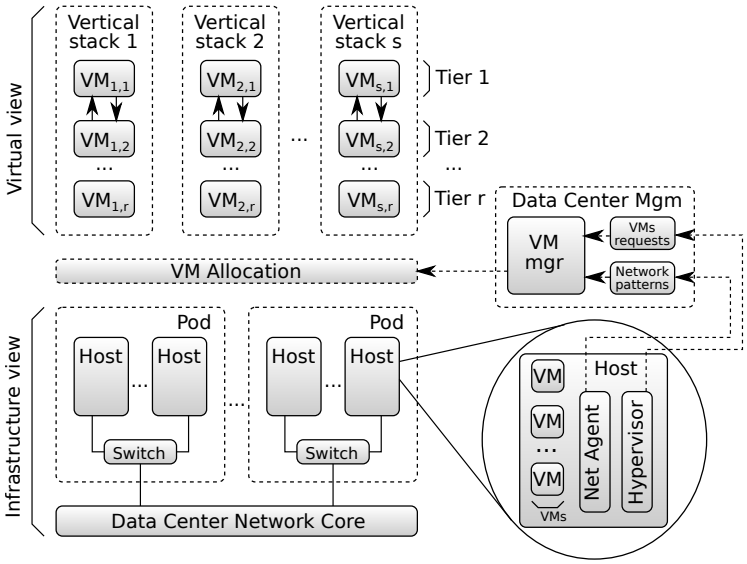


Fig. 1: Reference scenario

and $VM_{2,1}$), the final effect is that we may have correlation in network utilization even if no communication occurs between the two VMs. Dealing with this effect is one of the main challenges of our research.

The lower part of the figure represents the physical infrastructure of the data center (namely *Infrastructure view*). The data center is composed of manageable subareas, called PODs [4]. The physical hosts are interconnected using a hierarchically-structured network infrastructure (actual topologies in a data center may be more complex such as the fat-tree topology [1]) and, as a consequence, we may experience different communication costs when allocating VMs over the hosts. This heterogeneous cost of communication motivates our proposal for a network-aware VM placement [9].

Figure 1 provides also the detail of a physical host. Each host is equipped with an *Hypervisor*, that supports also the collection of data about the VMs resource usage. Typical resources, such as CPU, memory, I/O utilization, are made available for the data center management, while network utilization is processed by the *Network Agent* that is described in details in Section 4. Finally, the dashed box on the right side of the figure describes the Data Center Management: such component receives the resource utilization of each VM and collects the information generated by the network agent about VMs interaction. These data are then used to take decision about the allocation of VMs over the physical infrastructure, creating a mapping between the virtual and infrastructure views.

3 Methodology Description

The proposed methodology to support a network-aware VMs allocation in the previously described scenario has a twofold goal. First, we aim to identify couples of VMs exchanging data: the solution of the problem of allocating VMs can be improved by adding in the objective function of the optimization problem a component to capture VMs interactions [6,9]. Second, we aim to gain an insight on how the applications are deployed in the data center

to support more sophisticated management strategies. As an example, we can consider that the availability of the services can be increased by replicating and spreading the interacting VMs over the data center infrastructure.

We now propose a formalization of the previously introduced basic principles. Let $\mathcal{N}$ be a set of VMs in a data center. P_j^{out} and P_j^{in} are the two time series of packet rate in output from ad in input to VM $j \in \mathcal{N}$, respectively. For both time series let τ be the sampling interval. Our goal is to use this description of network traffic in the data center to infer which VMs exchange data among themselves. Specifically, we rely on the correlation between input and output time series. In our analysis we assume that there is no re-organization of the stacks composing the Web applications (i.e., there is no re-organization of the communication patterns during the collection of the time series). We introduce a methodology based on three steps:

1. Traces interpolation and synchronization;
2. Correlation matrix creation;
3. Interacting VMs identification.

3.1 Traces interpolation and synchronization

We will detail the mechanism to collect samples about the network data transfer in Section 4. For the goal of this analysis it is important to point out that the data for each VM will be a collection of records with the fields `<timestamp, pkt_in, pkt_out>`, with `pkt_in` and `pkt_out` being the number of packets received and transmitted in the last τ seconds, respectively. The last two values contain the number of packets received and transmitted in the last τ seconds. However, the gossip protocol used to distribute data and the absence of explicit synchronization between monitor for each VM results in time series from different VMs that can be not synchronized.

A set of preliminary experiments suggest that feeding not synchronized traces in the correlation analysis may lead to poor results for identifying interacting VMs. This motivates our choice to introduce a preliminary step in our methodology to synchronize the input data. To reach this result we exploit data interpolation as follows:

- we define a starting time t_0 and we remove every sample before that time. As a result, each time series will start in the time interval $[t_0, t_0 + \tau]$. Furhtermore, me make sure that every time series contains T samples;
- we define a synchronization time t_0^* as the average value of the starting times of each trace. We define a set of new sampling times as $t_0^*, t_0^* + \tau, \ldots, t_0^* + i\tau, \ldots, t_0^* + T\tau$;
- for each time series $P_{j_1}^{out}$, we define a synchronized time series $P_{j_1}^{*out}$ using cubic interpolation. In our prototype the implementation of the cubic interpolation is provided by the Python *Pandas*[2] framework. A similar procedure is carried out also for the time series of inbound packets $P_{j_2}^{in}$.

Figure 2 provides an example of the above described process. We start with two time series (left part of the figure). The samples in the series are shown as squares and circles, with each sample occurring after τ seconds from the previous one. The center of the figure illustrates the spline interpolation of the samples to obtain a continuous function from the samples, that is used in a re-sampling process synchronized to start at time t_0. Finally, on the right, we have the output time series of synchronized samples.

[2] http://pandas.pydata.org/

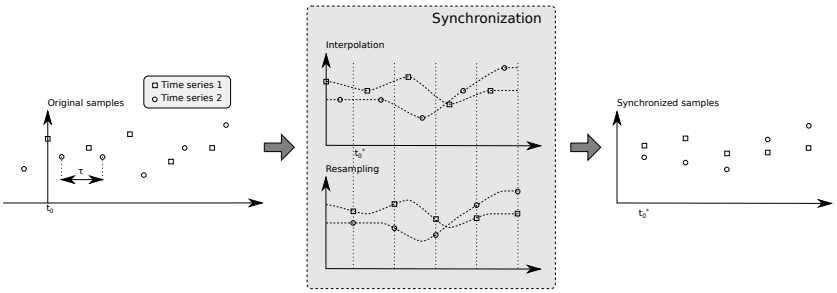


Fig. 2: Interpolation and synchronization

In order to maintain the readability of the paper, in the following we will adopt a simplified notation using $P_{j_1}^{*out}(i)$ and $P_{j_2}^{*in}(i)$ to indicate the i-th sample of a synchronized time series on VMs j_1 and j_2.

3.2 Correlation matrix creation

The correlated time series are used to create a correlation matrix $\mathbf{C}$ taking into account input and output packet rates of each pair of VMs. We explicitly focus on the correlation between output packet of a VM and input packet of another VM with the goal to identify the VMs exchanging data among themselves. Our approach is consistent with existing proposals in literature, such as [25, 29], that consider not just communication but also the direction of the data flow with the goal to optimize the data center communication infrastructure. It is also worth noting that this process is inherently parallel, so it is possible to leverage the gossip protocol described in Section 4 to locally generate portions of the $\mathbf{C}$ matrix and then propagate this information to make sure that the Data Center Management function has the complete matrix available.

We define the generic element c_{j_1,j_2} of the matrix $\mathbf{C}$ as:

$$c_{j_1,j_2} = \mathrm{Cor}(P_{j_1}^{*out}, P_{j_2}^{*in}) \tag{1}$$

where j_1 and j_2 are two generic VMs and $\mathrm{Cor}(\cdot)$ is the correlation function between time series. The resulting matrix $\mathbf{C}$ may be not symmetric. An example to explain this case is when we have a uni-directional data transfer, for example from j_1 to j_2. In this case the correlation between $P_{j_1}^{*out}$ and $P_{j_2}^{*in}$ is high, but the opposite correlation between $P_{j_2}^{*out}$ and $P_{j_1}^{*in}$ is mainly related to ACKs and is likely to be low. Several alternatives to quantify correlation can be found in literature. Our study compares the use of two metrics: *Pearson* and *Spearman* correlation functions [23]. The first is the most popular correlation metric, while the Spearman coefficient was identified in preliminary tests as a promising alternative to operate on time series sharing long-term trends, that is common in our reference problem.

The Person Correlation coefficient ρ is defined as:

$$\rho(P_{j_1}^{*out}, P_{j_2}^{*in}) = \frac{E[(P_{j_1}^{*out} - \mu(P_{j_1}^{*out}))(P_{j_2}^{in} - \mu(P_{j_2}^{*in}))]}{\sigma(P_{j_1}^{*out})\sigma(P_{j_2}^{*in})} \tag{2}$$

with $\mu(\cdot)$ being the mean value and, $\sigma(\cdot)$ the standard deviation. We use the notation $E[\cdot]$ for the average function, that appears in Eq. 2 to estimate the covariance.

The Spearman coefficient ρ_s is the correlation between two time series containing the *ranks* of the original time series values:

$$\rho_s(P_{j_1}^{*out}, P_{j_2}^{*in}) = 1 - \frac{6\sum_{i=0}^{T} r(P_{j_1}^{*out}(i)) - r(P_{j_2}^{*in}(i))}{T(T^2-1)} \quad (3)$$

where $r(\cdot)$ is the rank of a sample in the same time series and T is the number of samples. Operating on ranks rather than on values improves the ability to identify small fluctuations in values that may distinguish two time series sharing the same long-term trends, which is common when we have a workload equally distributed among replicated vertical stacks (as in our reference scenario).

3.3 Identification of interacting VMs

The last part of the proposed methodology is the identification of couples/groups of interacting VMs using the correlation matrix $\mathbf{C}$.

This step is part of the operation of the Data Center Management function and can focus on either identifying single couples of interacting VMs or on groups of VMs belonging to the same vertical stack (as in Figure 1).

With respect to the first task, that is identifying couples of VMs exchanging large amount of data, a first solution is to apply a threshold on the correlation matrix to identify *correlated* and *uncorrelated* time series of network resource utilization. In a nutshell, every couple of time series with a correlation c_{j_1,j_2} higher or equal than the threshold is correlated. On the other hand, if c_{j_1,j_2} is below the threshold, we assume that the two time series are not uncorrelated.

However, if we have a broader scope and we aim at identifying the vertical stacks (that are of VMs running different tiers of the same application), we must *cluster* together VMs that show a similar behavior in terms of network traffic patterns. To this aim we consider the correlation matrix as an *affinity matrix* between VMs and a clustering algorithm is applied to group together the VMs. Multiple clustering algorithms have been proposed in literature. However, we must focus on just a subset of them that accept an input in the form of an affinity/distance matrix (instead of a feature vector). Specifically, in the remaining of this paper we consider and compare: *spectral clustering* [19], *affinity propagation*., and *agglomerative clustering* [11].

Spectral clustering uses the Laplacian operator applied to the input similarity matrix. The eigenvalues and eigenvectors of the result are used to create a new coordinate system. The original samples projected into this new space of coordinates are then clustered using the k-means clustering algorithm [19]. Affinity propagation bases its operation on simulated message exchange between samples. The algorithms is quite fast as it identifies a representative for each cluster and assigns elements to each cluster based on the affinity with their representatives [14]. Finally, agglomerative clustering creates a *dendogram* starting with on cluster for each VM and then merging at each iteration the two most similar clusters. Depending on the cluster we want to obtain, we can then cut the dendogram at any point. In our experiments, we use the implementation of the three algorithms provided by the *SciKit-Learn* library[3].

[3] http://scikit-learn.org/

3.4 Scalability and computational complexity

We now discuss the scalability of the proposed technique with respect to the network size. In particular, we consider the computational complexity of the three steps composing the technique with respect to the number of hosts n.

The synchronization of each trace depends linearly on the network size, that is has a complexity that is $\mathcal{O}(n)$ (assuming that at most a finite number of VMs can run on each host). The creation of the correlation matrix has complexity $\mathcal{O}(n^2)$, that can be reduced to $\mathcal{O}(n)$ if we parallelize the task so that each host computes a slice of the global matrix (as described in the following of the paper). Finally, the most critical part of the technique is the clustering process that has a complexity ranging from $\mathcal{O}(n^2)$ to $\mathcal{O}(n^3)$, depending on the chosen algorithm.

We can thus conclude that the overall computational cost of the technique is polynomial. Hence, we can expect limited scalability problem with respect to the problem size.

4 Implementation

In this Section, we discuss an actual implementation of the proposed methodology. The goal of identifying clusters of VMs communicating to each other requires, as shown in Section 3, the knowledge of communication patterns of all VMs. This knowledge can be considered as an aggregate property of the whole system. From an engineering perspective, it is possible to calculate aggregate functions with either reactive or proactive approaches [15, 13].

Reactive approaches are based on queries issued by specific nodes in the network. The answers are returned directly to the issuer while the rest of the network may or may not receive the answer. An example of this approach could be a dedicated node periodically polling all VMs and making the needed computation in a centralised fashion.

Proactive approaches, instead, provide the value of aggregate functions to all nodes in the network in an adaptive fashion. By the term adaptive we mean that, if changes due to network dynamism or to variations in the input values arise, the output of the aggregation protocol should be capable of tracking these changes. Proactive protocols are frequently used in completely decentralized solutions of complex tasks.

Given the significant increase in size and complexity of modern data centers [12], we focus on a solution that exploits a reactive protocol based on gossiping. The nature and functioning of gossip-based aggregation is detailed in the following section.

4.1 Gossip-based aggregation

Let us consider a generic network with a set of $\mathcal{M}$ nodes, where each node corresponds to a physical host. Each node interacts with a small number of other nodes (neighbors). The basic protocol is based on the *push-pull* scheme illustrated in Algorithm 1. Time is divided into time slots of length τ, corresponding to the monitoring intervals. Each node $p \in \mathcal{M}$ executes two different threads. At every time slot, the active thread starts an information exchange with one random neighbor q by sending it a message containing the local status s_p and waiting for its remote status s_q. It is possible to extend the basic scheme to have interaction with multiple neighbors for each time slot. The passive thread waits for messages sent by an initiator and always replies with the local status. The term $push - pull$ refers to the fact that each information exchange is performed in a symmetric manner: both nodes

send and receive their status. It is worth noticing that the local status s_p could represent both properties of the node p itself or values measured by it. In the considered application, the status information basically consists in the time series of network utilization samples of both the local node and of the known neighbors. The methods `send()` and `receive()` are used for this message exchange.

The method `getNeighbors(`ω`)` showed in Algorithm 1 can be considered as a service underlying the aggregation protocol. It returns a uniform random sample over the entire set of neighbors. Furthermore, the parameter $\omega \in [0, 1]$ can be used for specifying how many neighbors have to be returned. In particular, with $\omega = 1$ the method will return all the available neighbors while with $\omega = 1/neighbors$ only one neighbor will be returned. The method `update` computes a new local status based on the current local status and the remote status received during the information exchange. The output of `update` as well as the semantic of the node status completely depends on the aggregation function needed by the application.

This gossip scheme tends to impose a uniform load to the system [15]. Each node executes the same amount of operations. Incidentally, reducing τ or increasing ω reduce convergence times of the algorithm (at expense of more messages exchanged). Therefore, applications can constantly manage the trade-off aggregation accuracy and communication costs.

Algorithm 1 Push-pull gossip protocol executed by node p

Ensure: Active thread
  for once every τ at random time in time-slot **do**
    $q[] \leftarrow$ `getNeighbors(`ω`)`
    for $q \in q[]$ **do**
      `send(`s_p`,`q`)`
      $s_q \leftarrow$ `receive(`q`)`
      $s_p \leftarrow$ `update(`s_p, s_q`)`
    end for
  end for
Ensure: Passive thread
  for ever **do**
    $s_q \leftarrow$ `receive(`$*$`)`
    `send(`s_p`,sender(`s_q`))`
    $s_p \leftarrow$ `update(`s_p, s_q`)`
  end for

4.2 Network Agent

Given the above definition of gossip-based aggregation, we now provide some details on how to implement it for the considered scenario. As pointed out in Section 2, each physical host is equipped with an *Hypervisor*, that manages the locally hosted VMs, and with a *Network Agent* that is a gossiping node responsible for the data exchange about the network resource utilization. We recall that the status of each VM is represented as a time series of tuples in the form `<timestamp, pkt_in, pkt_out>`. Given a sampling period $\tau = 30s$, a time series s_j comprising 240 elements would describe the latest 2 hours of network usage of VM_j. The status of a node consists of the status of each VM managed by that host

hypervisor that is, for node i the status is $s_i = \{s_j, j \in \mathcal{N}_i\}$, where $\mathcal{N}_i$ is the set of VMs hosted on physical host i.

The problem being addressed consists in making available at each host, in a timely fashion, the status of all the physical hosts comprising the considered data center, that is $s_i, \forall i \in \mathcal{M}$. The gossip protocol is used to spread data concerning the VMs running on one physical host to all others.

Let us consider two hosts p and q and their respective status s_p and s_q. At each gossiping iteration, as detailed above, nodes p and q exchange their statuses (i.e., the status s of all the VMs they are running). It is worth noticing that the nodes, as an example we consider node p, might already have a received the status s_q from a former exchange with either q or another node. In this case, the aggregation function selects the most recent version of s_q and discards the other. As a consequence, each physical host can keep track of an extended status comprising both s_p and s_q. If executed on the hosts of a data center, this algorithm allows each host to receive the status of all VMs running within the data center. To better clarify the details of the data exchange, Figure 3 provides a view of the message involved in the operations of the gossip protocol. A gossip message contains both information about the local host and propagates data about remote hosts. Each status for an host consists in the status of the hosted VMs, that, in turn, contains the samples time series.

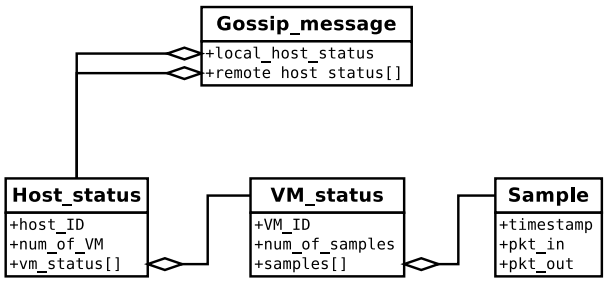


Fig. 3: Message format (UML notation)

It is also worth noticing that despite gossip-based communication schemes have proven several desirable properties over flooding, they are slower to converge [13]. Thus, for the sake of being able of successfully correlating time series, it is relevant that the time needed to spread network data across all the hosts is smaller than the network sampling period τ (i.e., in our case is 30s). If this condition is not met, gossiping can be suitably tuned by adjusting the ω parameters.

Finally, another benefit of the proposed approach comes from a reduction of the computational complexity of the clustering process. In fact, in case of reactive approaches a dedicated node has to collect the whole set of VMs statuses and find eventual correlations. In the proposed approach, instead, each node can compute correlations concerning only the VMs it hosts. Basically, the network agent on each host computes the correlation between each hosted VM and every other VM. This creates a set of columns in the final correlation matrix $\mathbf{C}$, that are then forwarded to the Data Center Management system for the identification of interacting VMs and/or clustering operations.

5 Experimental results

5.1 Setup description

The proposed approach is tested using an experimental setup exploiting the TPC-W benchmark[4]. The considered traces are referred to a TPC-W run with 12 VMs organized in 4 vertical stacks and covering a 12 hours period during which the samples are collected every 30 seconds (however, we consider also a granularity of 1 and 2 minutes for samples collection in our experiments).

To define the metrics to compare the different correlation functions, we need to distinguish between two different goals: first, identifying the interacting couples of VMs; second, clustering together VMs to identify vertical stacks.

For the first goal, that is identifying interacting VMs, we rely on a classic measure for classification problems: the *F-measure*, which is the harmonic mean of *Precision* and *Recall*. Precision is defined as $P = \frac{TP}{TP+FP}$ while recall is $R = \frac{TP}{TP+FN}$, with TP and TN true positives and negatives and FP and FN being false positives and negatives, respectively. F-measure is: $F = 2\frac{P \cdot R}{P+R}$.

For the second goal, we take into account one of the most popular metrics for clustering evaluation, that is clustering *purity* [2]. Purity is the fraction of correctly identified VMs and is used to evaluate the performance of the clustering step. In order to measure the purity, we consider the output of the clustering $\mathbf{S}$ and we compare it with a vector $\mathbf{S}^*$ that contains the correct classification of VMs (that, for each VM, contains the correct vertical stack it belongs to). We can thus provide a formal definition of purity as:

$$purity = \frac{|\{s^j : s^j = s^{j*}, \forall j \in \mathcal{N}\}|}{|\mathcal{N}|}$$

with s^j being the vertical stack to which VM j is assigned in by the clustering algorithm, s^{j*} being *correct* vertical stack of VM j, and $\mathcal{N}$ is the set of considered VMs.

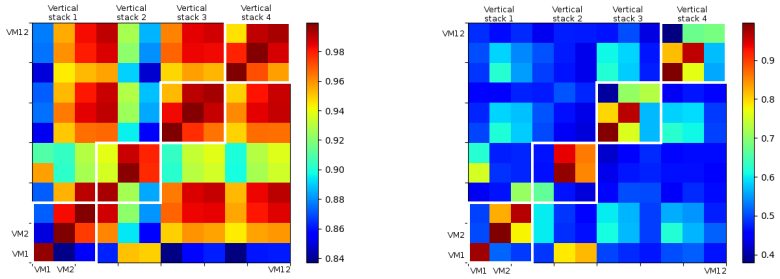
5.2 Correlation coefficients analysis

The first experiment aims at comparing the use of Pearson and Spearman correlation coefficients, that are used to compute the correlation matrices over every couple of time series.

Heat maps are used to represent the two correlation matrices in Figure 4: the two time series of input and output packets are presented for each VM (numbered 1 to 12 in the figure). An outline of the vertical stacks (the four white square frames on the main diagonal of the matrix) are also shown in each picture. The correlation should ideally present this behavior: high values within the four squares delimiting each vertical stack and low values outside the squares.

The observation of Figures 4a and 4b offers interesting insights. On one hand, the Pearson correlation coefficient (Fig. 4a) is characterized by values very high (close to 1, as shown by the predominance of orange-red hues). We also note some redish shades located far from the main diagonal outside from the vertical stack frames: this is particularly critical because it suggests the presence of high correlation between couples of VMs not belonging to the

[4] www.tpc.org/tpcw/



(a) Pearson correlation coefficient (b) Spearman correlation coefficient

Fig. 4: Heatmap of the correlation matrices

same vertical stack. If we observe the results achieved by the Spearman correlation coefficient (Figure 4b), we note that the use of this coefficient allows to better distinguish between VMs of different stacks and VMs belonging to the same vertical stack (as shown by the red areas located near to the diagonal), thus revealing a better capacity to identify VMs that are actually exchanging data.

5.3 Identification of communicating VMs

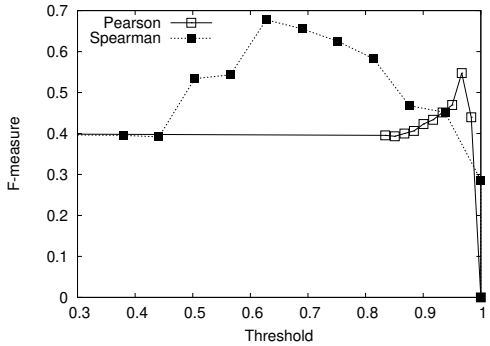


Fig. 5: F-measure comparison

In this experiment we evaluate the effectiveness of using the two correlation coefficients to identify couples of communicating VMs on the basis of a threshold. Specifically, we analyze the sensitivity of the VMs classification for different threshold values.

The comparison of the F-measure for the two correlation coefficients is shown in Figure 5 for increasing values of the threshold. The observation of the graph reveals that the use of the Spearman correlation coefficient may provide a double advantage: a) better performance with respect to the Pearson coefficient, characterized by higher maximum F-measure values; b) higher stability of the performance, with an accuracy higher than 0.5 achieved by

different threshold values (0.5 to 0.8), while the Pearson function achieves the best values for a single peak close to 0.96 as threshold value.

5.4 VMs Clustering

In this set of experiments we compare the use of the correlation coefficients to create the input affinity matrix for the three considered clustering algorithms (spectral clustering, affinity propagation and agglomerative clustering).

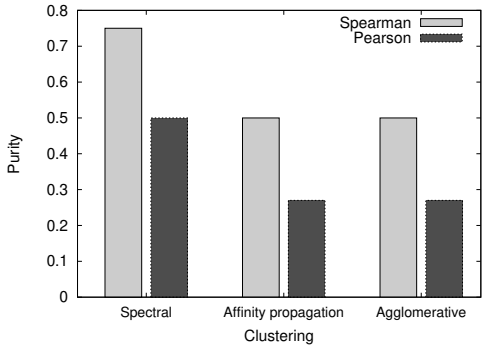


Fig. 6: Clustering purity comparison

Figure 6 represents the clustering purity of the clustering algorithms for each of the two considered correlation coefficients. From the figure, it is evident that the use of the Spearman coefficient allows the clustering algorithms to achieve significantly higher purity if compared to the alternative. Moreover, we note that the best performing algorithm of the three considered alternatives is the spectral clustering. The different performance of the algorithms may be explained by considering the example in Figure 7, that represents a spectral clustering output on the top against the actual clustering (that is, the ground truth) shown on the bottom. We may observe two kinds of errors in the clustering output: first, two VMs are swapped (VMs 1 is assigned to the cluster to which VM 4 belongs, and vice versa); second, VM 12 is misplaced. On the other hand, the poor performance of the affinity propagation algorithm are worsen by an additional problem: the wrong estimation the clusters number. Then, we note that the agglomerative clustering causes a high number of cluster swaps between VMs.

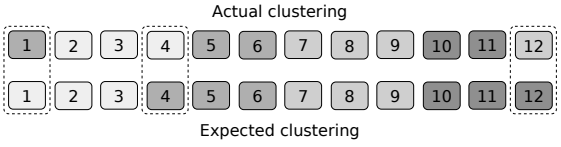


Fig. 7: Clustering example

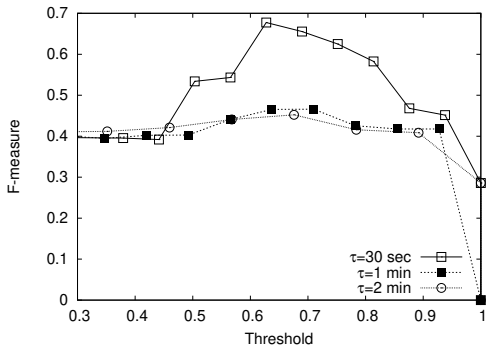


Fig. 8: Sensitivity to sampling period τ

5.5 Comparison of time sampling intervals

The last experiment evaluates the impact of data sampling granularity on the quality of the classification (that identifying correlated and non-correlated time series) and of the VMs clustering. Let us start the evaluation from the classification process. Figure 8 compares the F-measure obtained by using the Spearman correlation coefficient as the basis for VMs classification: the considered sampling period τ of network utilization ranges from 30 seconds to 2 minutes. We note that the impact on the classification F-measure is very significant, even with the small considered change in the sampling granularity. A coarse-grained data collection has, indeed, two main effects. First, the sensitivity to the threshold value of the Spearman correlation is reduced for large sampling periods. The increased robustness is a caused by the smoothing of the time series. Indeed, the time series taken into account basically only shows the most evident fluctuations: the stability to a wider range of threshold values is motivated by the ease of identifying low-frequency and highly evident fluctuations (which is basically the main way to distinguish two vertical stacks) even over a wider range of threshold values. Second, the accuracy significantly decreases for increasing sampling intervals if we observe the threshold range of values corresponding to higher F-measure ([0.6-0.8]). The motivation of this effect is quite intuitive: a smoother time series of the network resource utilization is the result of an increased sampling period. In this ways, we still are able to appreciate the main effects of the correlation; however, we miss the high frequency fluctuations, thus experiencing a reduction in the classification performance.

The final analysis evaluates the effect on the purity of the clustering solution of the sampling frequency. Figure 9 shows the comparison among the clustering purity of the three clustering algorithms making use of the Spearman coefficient of correlation. The graph offers two validations of previous observations. First, better performance of the spectral clustering with respect to other algorithms is confirmed, with the results showing a gain in the purity values ranging from 97% to 31% over the alternatives. Second, the increase of the sampling period τ significantly decreases the performance even for the spectral clustering algorithm, causing a reduction in the purity of 33% as τ increases from 30 seconds to 1 minute. This effect can be explained by the same motivations that cause a decrease in the F-measure values for the detection of communicating VMs: the distinction of different vertical stacks is more challenging in case of smoother time series.

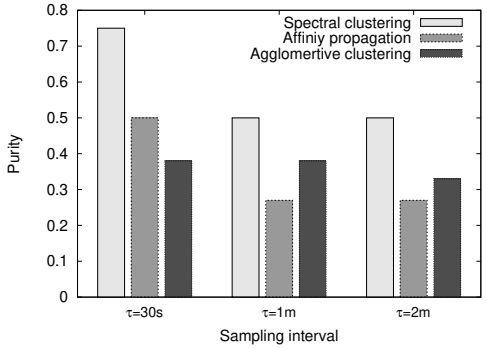


Fig. 9: Clustering purity comparison

6 Related work

The problem of describing network utilization patterns and using this information to improve the performance and efficiency of data transfer has been widely investigated in literature.

In particular, the problem of monitoring network traffic patterns to identify data flows has been tackled over the years, with special focus on the links over geographically-distributed networks, that may become bottlenecks in the data delivery. A large corpus of literature defines this problem as *network tomography* [10]. Among these studies, [10] exploits metrics based on single links to produce a topological description of the network flows in a wide-area network. The same goal is considered in [26], where a bayesian approach is used to describe and match the characteristics of data flows to reconstruct routing between origins and destinations. Papagiannki *et al.* focus on extracting the routing topology of a geographically-distributed network, with special attention on increasing scalability of the process thanks to a distributed approach, where computation is carried out at the point of presence and at the peering points of the network [24]. A more conservative approach is presented in [16], where Kowalski and Warfield derive from the theory on telephonic networks a probabilistic model to correlate the distance between two nodes and the traffic between the two. All these studies are related to our proposal, but the considered scenario, and the solution to address the problem, are more oriented towards wide-area networks, with complex and irregular graphs topology rather than on data center. Furthermore, these studies do not take into account the cloud scenario, where VMs may migrate over the infrastructure. Our focus on a modern data center with virtualization functions of computing and, possibly, of networking functions places our paper clearly aside from the above-mentioned studies.

Another branch on literature proposes the application of the Software-Defined Network paradigm to the problem of network monitoring. For example, [27] proposes NetFPGA, a system that exploits programmable hardware to monitor networks extracting useful statistics form the network links. A similar proposal is made in [28], that introduces a way to leverage the characteristics of SDN appliances to monitor large networks on a geographic scale, and in [18] that aims at improving the scalability of the process by selecting the most relevant flows in a network. Even if our approach could integrate and use information collected from a SDN infrastructure, we recall that our focus is more oriented to a cloud-based scenario, where the deployment of SDN appliances is still at its infancy, so we can't fully rely on it.

Other studies are complementary to our work, as thay can take advantage from our effort. For example, [22] focuses on exploiting the knowledge of data exchange between

VMs in a cloud environment to improve the performance of the cloud-hosted applications, while in [25] the final goal is to increase the efficiency of network resource usage.

Finally, we recall that preliminary versions of this research where published in [8,17]. However, in this paper we provide a more detailed discussion on the issue of scalability and we propose a new software architecture to implement the considered theoretical model.

7 Conclusions

This paper focused on the problem of identifying groups of VMs exchanging data traffic in a cloud data center when we do not have access to detailed information among the data exchange between each specific couple of VMs. Our reference scenario is a multi-tiered application deployed over vertical stacks that are replicated horizontally. This case is a very common and challenging scenario in modern cloud systems where replication is used to achieve scalability. To discover communicating VMs we described a methodology that rely on the correlation between the time series of VMs network packet flows and exploits clustering to identify the vertical stacks of VMs. We evaluate and compare different correlation coefficients to find the best performing solution. From our experiments it is clear that solutions based on tanking techniques (such as the Spearman correlation coefficient) provide better performance than the alternatives. Furthermore, we compare three clustering algorithms for detecting the vertical stack: our experiments show that spectral clustering clearly outperforms the other alternatives. As a last analysis, we perform a sensitivity evaluation about the sampling period of network data: the results demonstrate that a fine-grained collection is more suited to find communicating VMs than a coarse-grained approach.

Acknowledgements The authors acknowledge the support of the project *S^2C: Secure Software-defined Cloud* funded by University of Modena and Reggio Emilia.

References

1. Al-Fares, M., Loukissas, A., Vahdat, A.: A scalable, commodity data center network architecture. In: Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication, SIGCOMM '08, pp. 63–74. ACM, New York, NY, USA (2008). DOI 10.1145/1402958.1402967. URL http://doi.acm.org/10.1145/1402958.1402967
2. Amigó, E., Gonzalo, J., Artiles, J., Verdejo, F.: A Comparison of Extrinsic Clustering Evaluation Metrics Based on Formal Constraints. Journal of Information Retrieval **12**(4), 461–486 (2009)
3. Andreolini, M., Colajanni, M., Pietri, M.: A scalable architecture for real-time monitoring of large information systems. In: Proc. IEEE Symposium on Network Cloud Computing and Applications. London, UK (2012)
4. Ballani, H., Costa, P., Karagiannis, T., Rowstron, A.: Towards predictable datacenter networks. ACM SIGCOMM Computer Communication Review **41**(4), 242–253 (2011)
5. Beloglazov, A., Abawajy, J., Buyya, R.: Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. Future generation computer systems **28**(5), 755–768 (2012)
6. Boru, D., Kliazovich, D., Granelli, F., Bouvry, P., Zomaya, A.Y.: Energy-efficient data replication in cloud computing datacenters. Cluster Computing **18**(1), 385–402 (2015)
7. Canali, C., Lancellotti, R.: Exploiting Classes of Virtual Machines for Scalable IaaS Cloud Management. In: Proc. of the 4th Symposium on Network Cloud Computing and Applications (NCCA) (2015)
8. Canali, C., Lancellotti, R.: Identifying Communication Patterns Between Virtual Machines in Software-Defined Data Centers. SIGMETRICS Performance Evaluation Review **44**(4), 49–56 (2017)
9. Canali, C., Lancellotti, R., Shojafar, M.: A Computation- and Network-Aware Energy Optimization Model for Virtual Machines Allocation. In: Proc. of International Conference on Cloud Computing and Services Science (CLOSER 2017). Porto, Portugal (2017)

10. Castro, R., Coates, M., Liang, G., Nowak, R., Yu, B.: Network tomography: Recent developments. Statistical science pp. 499–517 (2004)
11. Day, W.H., Edelsbrunner, H.: Efficient algorithms for agglomerative hierarchical clustering methods. Journal of classification **1**(1), 7–24 (1984)
12. Dayarathna, M., Wen, Y., Fan, R.: Data center energy consumption modeling: A survey. IEEE Communications Surveys & Tutorials **18**(1), 732–794 (2016)
13. Eugster, P.T., Guerraoui, R., Kermarrec, A.M., Massoulieacute;, L.: Epidemic information dissemination in distributed systems. Computer **37**(5), 60–67 (2004). DOI http://doi.ieeecomputersociety.org/10.1109/MC.2004.1297243
14. Frey, B.J., Dueck, D.: Clustering by passing messages between data points. science **315**(5814), 972–976 (2007)
15. Jelasity, M., Montresor, A., Babaoglu, O.: Gossip-based aggregation in large dynamic networks. ACM Transactions Computer Systems **23**(3), 219–252 (2005)
16. Kowalski, J.P., Warfield, B.: Modelling traffic demand between nodes in a telecommunications network. In: Proc. of ATNAC95 (1995)
17. Lancellotti, R., Canali, C.: A correlation-based methodology to infer communication patterns between cloud virtual machines. In: Proc. of the 10th EAI International Conference on Performance Evaluation Methodologies and Tools (VALUETOOLS), pp. 251–254. Taormina, Italy (2017)
18. Li, D., Dai, N., Li, F., Xing, C., Dai, F.: Estimating SDN traffic matrix based on online informative flow measurement method. In: Proc. of 2017 Fifth International Conference on Advanced Cloud and Big Data (CBD), pp. 75–80 (2017)
19. Luxburg, U.: A tutorial on spectral clustering. Statistics and Computing **17**(4), 395–416 (2007)
20. Marotta, A., Avallone, S.: A Simulated Annealing Based Approach for Power Efficient Virtual Machines Consolidation. In: Proc. of 8th International Conference on Cloud Computing (CLOUD). IEEE (2015)
21. Mastroianni, C., Meo, M., Papuzzo, G.: Probabilistic Consolidation of Virtual Machines in Self-Organizing Cloud Data Centers. IEEE Transactions on Cloud Computing **1**(2), 215–228 (2013). DOI 10.1109/TCC.2013.17
22. Meng, X., Pappas, V., Zhang, L.: Improving the Scalability of Data Center Networks with Traffic-aware Virtual Machine Placement. In: Proc. of the 29th Conference on Information Communications (INFOCOM). San Diego, California, USA (2010)
23. Myers, L., Sirois, M.J.: Spearman Correlation Coefficients, Differences between. John Wiley & Sons, Ltd (2014). DOI 10.1002/9781118445112.stat02802. URL http://dx.doi.org/10.1002/9781118445112.stat02802
24. Papagiannaki, K., Taft, N., Lakhina, A.: A distributed approach to measure ip traffic matrices. In: Proc. of the 4th ACM SIGCOMM Conference on Internet Measurement, pp. 161–174. ACM (2004)
25. Sonnek, J., Greensky, J., Reutiman, R., Chandra, A.: Starling: Minimizing Communication Overhead in Virtualized Computing Platforms Using Decentralized Affinity-Aware Migration. In: Proc. of 39th International Conference on Parallel Processing (ICPP). San Diego, CA (2010)
26. Tebaldi, C., West, M.: Bayesian inference on network traffic using link count data. Journal of the American Statistical Association **93**(442), 557–573 (1998)
27. Yu, M., Jose, L., Miao, R.: Software defined traffic measurement with opensketch. In: Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13), pp. 29–42. USENIX, Lombard, IL (2013)
28. Yuan, L., Chuah, C.N., Mohapatra, P.: Progme: Towards programmable network measurement. IEEE/ACM Transactions on Networking **19**(1), 115–128 (2011)
29. Zhang, Y., Ansari, N.: Hero: Hierarchical energy optimization for data center networks. IEEE Systems Journal **9**(2), 406–415 (2013)