Supporting Autonomic Management of Clouds: Service Clustering with Random Forest

Rafael Brundo Uriarte, Francesco Tiezzi and Sotirios A. Tsaftaris

***Abstract*—A promising solution for the management of services in clouds, as fostered by autonomic computing, is to resort to self-management. However, the obfuscation of underlying details of services in cloud computing, also due to privacy requirements, affects the effectiveness of autonomic managers. Data-driven approaches, in particular those relying on service clustering based on machine learning techniques, can assist the autonomic management and support decisions concerning, e.g. the scheduling and deployment of services. Unfortunately, applying such approaches is further complicated by the coexistence of different types of data within the information provided by the monitoring of cloud systems: both *continuous* (e.g. CPU load) and *categorical* (e.g. VM instance type) data are available. Current approaches deal with this problem in a heuristic fashion. In this paper, instead, we propose an approach that uses all types of data, and learns in a data-driven fashion the similarities and patterns among the services. More specifically, we design an unsupervised formulation of Random Forest to calculate service similarities and provide them as input to a clustering algorithm. For the sake of efficiency and to meet the dynamism requirement of autonomic clouds, our methodology consists of two steps: off-line clustering and on-line prediction. Using datasets from real-world clouds, we demonstrate the superiority of our solution with respect to others and validate the accuracy of the on-line prediction. Moreover, to show applicability of our approach, we devise a service scheduler that uses similarity among services, and evaluate its performance in a cloud test-bed using realistic data.**

***Index Terms*—Cloud Computing, Autonomic Computing, SLA, Random Forest, Similarity Learning.**

I. Introduction

In cloud computing, virtually everything can be provided as an on-line, on-demand service. The characteristics of clouds, such as scalability, heterogeneity and dynamism, make clouds considerably complex. A prominent approach to cope with this complexity is autonomic computing [1], which aims at equipping systems with capabilities to autonomously adapt their behaviour according to dynamic operating conditions. To achieve self-management, the system entities in charge of enacting autonomic strategies, the so-called *autonomic managers*, require knowledge about the operating environment as well as the system itself.

The abstraction of the underlying complexity provided by clouds (achieved, e.g. via virtualization) restricts the knowledge available to autonomic managers and consequently limits their range of actions. Data-driven approaches based on machine learning techniques, which do not require human knowledge and intervention, can assist the operation of autonomic managers. Thus, we advocate the use of a *clustering* approach, which generates knowledge consisting of groups (i.e. clusters) of services with similar characteristics. This form of knowledge can be exploited by autonomic managers for different purposes, such as: optimisation of resources, service scheduling and anomalous behaviour detection.

A critical aspect that complicates any data-driven knowledge extraction approach is that information (called *features*) on services consists of both *categorical* (e.g. virtual machine instance type) and *continuous* (e.g. CPU load) types of data. This problem is typically addressed in a heuristic fashion: either only one data type is used, which reduces distinguishability, or combinations of data types are constructed by human experts. Both solutions do not cope well with the dynamism of the autonomic cloud: when new types of services are introduced they may not be distinguishable or a human intervention might be necessary again. Moreover, dealing with the potentially enormous amount of data concerning cloud services, as they are more frequently deployed in a *Big Data* fashion, is another important challenge.

In this paper, we tackle the challenge of providing a truly autonomic and effective management of services in clouds through similarity-based knowledge, which is calculated by using *all* types of service features. To this aim, we propose a learning methodology relying on the *Random Forest* (RF) algorithm [2]. We learn the service similarities from the definition of services or their monitoring information, and provide them to a clustering algorithm. In particular, in order to achieve efficiency and meet the dynamism requirement of autonomic cloud systems, our methodology consists of two main steps:

1) *off-line clustering*, to learn similarities and obtain the clusters;
2) *on-line prediction*, to predict to which of the computed clusters an incoming new service belongs.

Our methodology was designed to support important characteristics of Big Data: it addresses its volume, velocity and variety of data with easily parallelizable training and prediction phases, supports categorical and continuous variables, has constant prediction time and a mechanism to detect changes in the patterns of services.

The main contributions of this paper are:

- An analysis of the specificities of the Autonomic Cloud (AC) domain and the definition of the requirements of a clustering approach for AC services;
- An off-line approach that relies on the RF algorithm to learn the similarities among all observed services, which

R.B. Uriarte is with IMT Institute for Advanced Studies Lucca, Italy - rafael.uriarte@imtlucca.it

S.A. Tsaftaris is with The University of Edinburgh, United Kingdom - S.Tsaftaris@ed.ac.uk

F. Tiezzi is with University of Camerino, Italy - francesco.tiezzi@unicam.it

are structured as a matrix that is then provided to an off-the-shelf clustering algorithm to identify clusters;
- A cluster parsing to reduce the size of the matrix (and memory footprint);
- An on-line prediction approach that uses the re-sized matrix to assign new services to clusters with low computational requirements;
- A performance and accuracy analysis of the proposed methods using real-world datasets;
- A use-case implemented in a cloud test-bed, which demonstrates the benefits of the proposed solution through a novel scheduling algorithm that employs the similarity of services to allocate them in the cloud resources.

This paper is a revised and extended version of [3]. While the general aim remains the same, the technical development has been significantly extended. Among the novelties, we highlight the extension of our methodology to detect new tendencies in the services, which is a key element to define when to retrain the forest. Retraining is indeed necessary to preserve accuracy of predictions as new services are deployed in the system. This extension complements our solution and would facilitate its deployment in practice. Also the validation of the proposed methodology was extended with several new experiments. We show in practice how our solution improves the performance of the cloud in terms of reduction of Service-Level-Agreement (SLA) violations and evaluate the retraining strategy we propose.

The rest of the paper is organised as follows. Section II discusses the potential uses of the similarity knowledge and illustrates the requirements of the AC domain. Section III presents the proposed methodology. Section IV describes the application of the methodology to real-world datasets and in the use-case. Section V reviews related works, while Section VI draws conclusions and hints at directions for future work.

II. Similarity Knowledge and AC Domain Requirements

In this section, we discuss the benefits of the similarity knowledge, i.e. a measure of how alike two services are, and describe the requirements of the AC domain for the solutions seeking to learn such knowledge from the data. We estimate a measure of similarity by relying on service definitions, such as SLAs, and monitoring information. In the domain, this knowledge is versatile and can either be directly used by autonomic managers or provided to a clustering algorithm. Clustering algorithms use the similarity among services to group them in a way that a service is more similar (in the considered aspects) to the services in its group than to services in other groups.

These groups, called *clusters*, have a wide range of application in clouds. For example, they can be employed in autonomic management for service profiling, which helps to identify and label clusters according to the patterns of the services in each cluster, or for supporting service scheduling, behaviour prediction and efficient identification of possible SLAs violations. Another use concerns anomalous behaviour detection, which aids autonomic managers to detect failures or intrusions, by assuming that the majority of the services are normal and by looking at the cluster with the most dissimilar services.

TABLE I
Correspondence between autonomic cloud characteristics and requirements for clustering algorithms.

AC Characteristics	Requirements
Security, Heterogeneity, Dynamism	Mixed Types of Features
Large-Scale, Dynamism	On-line Prediction
Security, Heterogeneity, Dynamism, Virtualization	Similarity Learning
Large-Scale, Multi-Agent	Parallelism
Heterogeneity	Large Number of Features

The correlations among services, necessary to cluster services, are difficult to extract from raw data and performance features, especially in the autonomic clouds, since its characteristics limit the available information. We discuss below its most relevant characteristics and their impact on this task expressed in terms of requirements; we summarise their correspondence in Table I.

Approaches to improve security are commonly based on data cryptography, data anonymisation and control of cross-layer transmission of information. To learn from such data a clustering algorithm has to support mixed types of features (e.g. discrete, continuous, symbolic). Moreover, virtually everything can be provided as a service in a cloud (*heterogeneity*). Therefore, using only either categorical or continuous features may lead to clusters that do not distinguish different services, which hence provide inferior performance. Alternative approaches to deal with mixed types of features can rely on the use of pre-processing techniques, such as discretisation, normalisation or standardisation, and on hand crafting of new features that combine categorical and continuous ones. However, these solutions require human expert intervention and the full understanding of the dataset and the relationships among features, which are also hindered by security restrictions, virtualization and service heterogeneity. Moreover, they require building a new heuristic every time the autonomic manager faces a new service type. Therefore, a data-driven similarity learning approach is required in this domain.

On the other hand, due to the heterogeneity and complexity of cloud services (indeed, services may be characterised by, e.g. 100 features), the clustering algorithm should process them in an acceptable time and should not be invasive on the system. Therefore, the clustering algorithm needs to support a large number of features.

Clouds usually are *large-scale*. To cope with the potentially big amount of services within an acceptable time, a clustering algorithm should run in parallel. This also allows to move the processing elements close to the data sources, thus reducing the impact on single resources and avoiding unnecessary network traffic, which is particularly convenient when data is costly to transmit and store. These are indeed typical requirements when big data are processed.

Clouds are inherently *dynamic*. New services are constantly added to the system, as the requested resources vary over time (also due to the pay-per-use business model often employed in clouds). This would require to re-cluster all services each time a new service is added, which however is impracticable due to the large scale of clouds and the inconstant arrival rate of services. Hence, it is necessary to predict on-line the cluster to which a new service belongs.

Finally, autonomic computing is *agent-based*. A clustering algorithm can then benefit from a multi-agent architecture in order to conveniently parallelise its workload.

III. Autonomic Management of Clouds with Clustering

To achieve a meaningful measure of similarity among services in the context of autonomic clouds, we assume no prior knowledge. Since multi-dimensional correlations are difficult to extract from raw data and performance features, we use clustering methods to learn similarities and identify patterns. From the range of available clustering algorithms, we have considered those that (i) can handle mixed data types (continuous and categorical) without human expert intervention, (ii) are fast both in the training and in the prediction phase, and (iii) offer high performance (with regard to accuracy).

In this section, we first discuss our choice for unsupervised clustering with the RF algorithm to address the above requirements, and then we proceed in defining our methodology based on RF to learn similarities among services and to cluster services using the obtained similarity knowledge.

A. Clustering as unsupervised machine learning

Machine learning techniques can be categorised in supervised and unsupervised.

Supervised approaches infer a function from a labelled training set consisting of observations for which class memberships are known. In our context, an observation corresponds to a service and is denoted as a $1 \times f$ vector $\mathbf{x}$, containing the values (continuous or categorical) of the f (in number) features (characteristics) of the services. Classes are the "categories" of the observations: for instance, "small" and "large" for virtual machines. The function inferred on the basis of the training set is then used to determine the class of non-labelled observations. Evidently, the reliance on a training set, and in particular its size and labelling quality, directly affects the performance of the supervised learning technique. This dependency hinders the adoption of this technique in the context of autonomic clouds, as manually labelling services, beyond its substantial cost, is particularly difficult to perform due to the heterogeneity, dynamism and security aspects in the domain.

In view of these limitations, in this paper we adopt *unsupervised learning*, which is used to find structure and patterns in data without the need of labelled training data. More specifically, we propose a combination of a similarity learning step, to discover a proper measure of similarity among observations (corresponding, in our case, to services in the cloud), and a clustering algorithm, to group the observations according to this measure of similarity. In light of the domain requirements (illustrated in Section II), as the means to obtain such notion of similarity we devise a solution based on the RF algorithm [2].

B. Service Clustering with Random Forest in the AC Domain

The RF algorithm relies on an ensemble of independent decision trees and was initially developed for regression and classification. Decision trees are tree-like structures where at each node an input feature is tested and a decision is progressed to the left or right branches to a subsequent node. A terminal node (leaf) contains the final decision (e.g. in the case of classification a class label). RF has a training and a prediction step. In its training step, RF uses bootstrapping aggregation (i.e. re-sampling from the dataset) and random selection of features to train t decision trees (where t is a number defined by the human or autonomic manager of the cloud). In the prediction step, the observations are parsed through all t trees and the classes of the observations are defined aggregating the decision of each tree (in the simplest case a majority rule is used). For details on the classification and regression algorithms we direct the reader to [2]. In summary, the main benefits of RF are:

- it supports mixed types of features in the same dataset;
- due to feature selection, it effectively handles data with a large number of features;
- it is one of the most accurate learning algorithms [4];
- it is efficient and scalable [4];
- it is easily parallelisable.

In [5], Breiman and Cutler proposed an unsupervised version of RF. The algorithm works as follows: (i) the training dataset (original data) is labelled as class one; (ii) the same number of synthetic observations are generated by sampling at random from the univariate distributions of the original data (synthetic data); (iii) the synthetic data are labelled as class two; (iv) the trees are trained with the original and synthetic data; and (v) only the original data are parsed through the trees, which yield the references of the leaves in which the observations ended up.

What is particularly relevant for our purpose is that this algorithm generates an intrinsic similarity measure. Intuitively, the more times two observations end up on the same leaf, the more similar they should be, as they followed the same decision nodes.

More formally, the similarity between two observations $\mathbf{x}_j$ and $\mathbf{x}_k$ (where indices j and k are used to refer to two distinct observations) is calculated as follows. Each observation is parsed through all t trees of the forest; the leaves in which the observations end up are annotated as l_j^i and l_k^i respectively, where i is the index of the tree. Let I represent an indicator function, which yields 1 if two observations end in the same leaf in that tree and 0 otherwise. Thus, the similarity between two observations is defined as:

$$S(\mathbf{x}_j, \mathbf{x}_k) = \frac{1}{t} \sum_{i=1}^{t} I(l_j^i, l_k^i) \tag{1}$$

The similarity of all pairs of n observations is calculated, generating the Similarity matrix $\mathtt{S}_{n\times n}$. The Dissimilarity matrix $\mathtt{D}_{n\times n}$, which is generated from the similarity matrix as $\mathtt{D}_{n\times n} = \sqrt{1-\mathtt{S}_{n\times n}}$, is symmetric, positive and lies in the interval [0,1]. This matrix requires a considerable amount of fast memory when dealing with large datasets. To address this issue, Breiman proposed the use of the references of the leaves in which the observations ended up in each tree, which results in the Leaves matrix ($\mathtt{L}_{n\times t}$ matrix), where t is the number of trees, and usually $n >> t$. Therefore, the forest can be built in parallel and the system can generate the dissimilarity matrix when necessary.

To cluster the observations, the dissimilarity matrix is passed as input to a compatible clustering algorithm, for example, the Partitioning Around Medoids (PAM) [6]. Otherwise, the dissimilarity matrix can be transformed into points in the Euclidean space to be used as input by other clustering algorithms, e.g. the standardised version of K-means [7]. The disadvantages of this extra step is the computational cost and the time necessary to perform the transformation operation.

Due to the scale of autonomic clouds and high arrival rate of new observations, a very low prediction time is necessary. The existing unsupervised RF approaches need to re-execute the whole clustering process for each new observation just to obtain the similarity of the new observation, which obviously is impracticable due to the high overhead of this process. In Section IV-B we compare the performance of our solution to the standard off-line algorithm.

The alternative is to use on-line RF algorithms, which learn similarities and cluster observations in an instantaneous fashion without requiring all data a priori. Unfortunately, the most known on-line adaptations of the algorithm are computationally demanding, cannot make fast predictions and are devised for supervised learning, which is not easily adaptable and does not suit the requirements of the cloud domain (we refer to Section V for more details).

In light of these limitations and of the domain requirements, we propose a novel on-line prediction algorithm based on RF.

C. On-Line Prediction with RF

We propose an efficient on-line prediction solution tailored to fulfil the requirements of AC (summarised in Table I). This solution takes advantage of the design of the clustering algorithm and pre-processes the clustering components to enable a fast and low memory implementation.

The outcome of the RF similarity learning is the $\mathtt{L}_{n\times t}$ matrix, where n is the number of observations and t is the number of trees. As n grows, this matrix may grow significantly and will have a large memory footprint as this matrix is necessary for the clustering of new services. We propose a solution which, in the prediction phase, requires only a new Medoids matrix, $\mathtt{M}_{m\times t}$, where m is the number of medoids (essentially the number of clusters) and t the number of trees. Since $m << n$ (typically $m \leq 20$), this matrix has a very small memory footprint.

Our solution, termed **RF+PAM**, combines the strengths of similarity learning of RF with the computational benefits of PAM and is divided in off-line training and on-line prediction, which are coupled and thus are presented here together.

The training phase, as depicted in Figure 1, consists of the following steps:

(i) given as input the training set, which is composed of the original n observations and synthetic data (as described in the previous section), build a forest with t trees;
(ii) parse the observations (original data) through the resulting forest and yield two matrices: the Dissimilarity matrix ($\mathtt{D}_{n\times n}$) containing the dissimilarity of all pairs of observations, and the Leaves matrix ($\mathtt{L}_{n\times t}$), containing the references of the leaves in which the observations ended up;
(iii) give the Dissimilarity matrix generated in the previous step to the PAM clustering algorithm, which yields the m medoids for the dataset, i.e. the observation of each cluster which maximises the inter-cluster dissimilarity (the representative of each cluster);
(iv) from the Leaves matrix (generated by step (ii)) the observations (vectors) that were chosen as the representatives of each cluster (i.e. the medoids) are selected to create a much smaller matrix, the Medoids matrix ($\mathtt{M}_{m\times t}$).

The advantage of this approach with respect to the standard RF is that we need to store only the forest and the medoids matrix ($\mathtt{M}_{m\times t}$), which has a much smaller memory footprint.

In the prediction phase of RF+PAM, a new observation is assigned to the cluster of the most similar medoid, i.e. the medoid which the new observation ended up in the same leaf most often, considering all trees of the forest. Figure 2 depicts the main steps of the prediction phase, which are described below:

(i) when a new observation $\mathtt{x}_{1\times f}$ arrives, it is parsed through the forest, which outputs the leaves vector $\mathtt{l}_{1\times t}$ containing the references of the leaves in which $\mathtt{x}_{1\times f}$ ended up in each tree;
(ii) together with the leaves vector and the Medoids matrix obtained in the training phase, the dissimilarity between each medoid and the new observation is calculated (dissimilarity vector $\mathtt{d}_{1\times m}$);
(iii) the new observation is assigned to the cluster whose medoid has the least dissimilarity to the new observation, that is the cluster of the medoid which has the least value in the vector generated by the previous step.

The benefits of RF+PAM are several: it can be trained in parallel; it handles, in a data-driven fashion, mixed data types; and it can provide predictions in a rapid and efficient manner. In Section IV, we will demonstrate the accuracy and effectiveness of our approach comparing it with clustering based approaches that have been used in the context of service management, but adapted to the problem of service clustering. Since these methodologies rely mostly on the K-means clustering algorithm, to isolate and quantify the exact benefit of similarity learning, we also considered a version of our RF based approach, termed **RF+K-means**, which utilises the K-means algorithm for clustering services and a similarity measure obtained by RF. Note that we do not necessarily advocate the use of RF+K-means, but we explained it below

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2016.2569000, IEEE Transactions on Network and Service Management

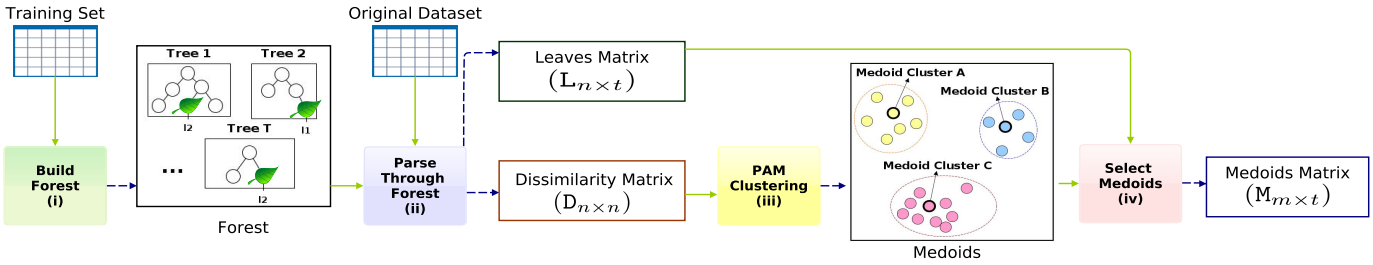


Fig. 1. Training phase of the proposed RF+PAM solution; notably, only the forest and the Medoids matrix are stored for the prediction phase. Green arrows indicate the inputs of each step and blue dotted arrows their outputs.

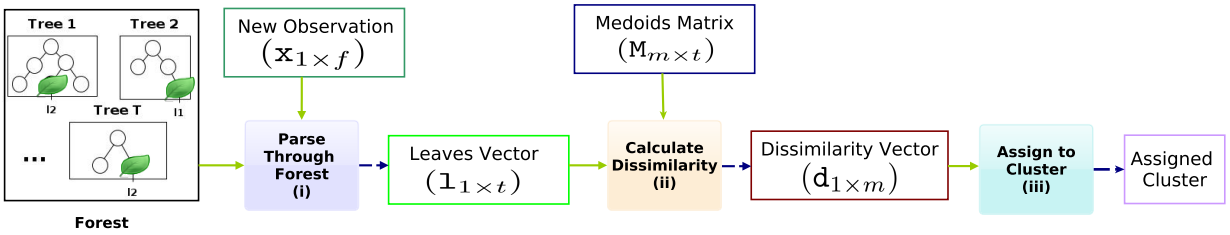


Fig. 2. Prediction phase of the proposed RF+PAM solution. Green arrows indicate the inputs of each step and blue dotted arrows their outputs.

for completeness and for the purpose of providing a fair comparison with methodologies in the literature. We also use it as a way to showcase the superiority of relying on PAM in the context of autonomic service management.

The training phase of RF+K-means uses the same initial steps of RF+PAM to obtain the dissimilarity matrix. However, it needs an extra step before clustering. Since the standardised version of K-means uses the Euclidean distance to cluster observations, the dissimilarity matrix is first transformed into a set of points in the Euclidean space using the Multidimensional Scaler (MDS) algorithm [8]. Thus, the distances between the observations are approximately equal to their dissimilarity. Next, the observations are clustered using K-means, which returns the cluster assignments of the observations. The outcomes of this phase, which need to be stored, are the leaves matrix, the forest and the clustering assignments.

The on-line prediction phase of RF+K-means is similar to the RF+PAM solution. It is composed of the following steps: (i) it parses the new observation through the trees; (ii) it calculates the dissimilarity between the new observation and *all original data* using the leaves matrix and the result of step (i), which consists of the references of the leaves in which the new observation ended up; and (iii) it assigns the new observation to the cluster with the least average dissimilarity between the new observation and all the observations in that cluster. Notably, also this solution calculates only the dissimilarity of the new observation with respect to the original data, i.e. it does not require the re-calculation of the whole dissimilarity matrix.

Although the differences between RF+K-means and RF+PAM are subtle, the impact is significant. RF+PAM is faster and has lower memory requirements, as it uses the medoids matrix, which is much smaller than the leaves matrix used by RF+K-means. Despite that the requirement of the MDS step in the RF+K-means can open the road to the wide range of algorithms that need a Euclidean distance for clustering, it is computationally demanding.

We conclude this subsection by discussing the issue of determining the number of clusters. Such issue is ambiguous and problem dependent. In many cases, the expert of the domain defines this number according to the particularities of the problem and to the objectives of clustering. For example, in cloud service clustering, if a cloud provider offers a fixed number of different services (e.g. specific applications), the number of clusters is likely to be this number.

Nevertheless, for a complete autonomic solution, there exist many techniques for the automatic determination of the number of clusters. They usually emphasise the intracluster compactness and intercluster separation, and consider the effects of other factors such as the geometric or statistical properties of the data [9]. For a review on 30 of these techniques, we refer to [10]. RF+PAM is particularly effective for most of these techniques, as only the PAM clustering and the medoid selection (steps (iii) and (iv) of the training), which are not computationally demanding, must be re-executed for different numbers of clusters.

D. When to Retrain?

Since we divided our methodology into training and prediction, and the training phase is off-line, naturally we would expect at some point the need to retrain the forest. Retraining requires the definition of a mechanism to recognise when a forest should be rebuilt. Although this definition is mostly problem-specific and depends on the available resources and accuracy requirements of managers, we analyse some approaches to decide when to retrain RF+PAM.

In RF+PAM, managers may define simple metrics for retraining, such as time intervals (e.g. every day) or a ratio between the number of new observations and the number of observations used to train the forest (e.g. when the number

of predicted observations is 10 times the number of training observations, then retrain). However, these solutions may not consider the evolution in observation patterns. Statistical tools which verify such changes, e.g. ADWIN2 [11], can be used to detect new tendencies, but they do not support customisation according to the user needs (for example, how different is the new pattern of observations). Therefore, they do not provide the means to define the trade-off between the cost of retraining and prediction accuracy.

We thus propose the use of the dissimilarity among new observations and the cluster medoids to define a measure of the evolution of observations. The intuition behind the proposal is that if new observations are far from *all* medoids, these medoids (and their clusters) do not represent well the observations anymore.

More formally, we named this dissimilarity-based measure as $alpha$:

$$\alpha = \frac{p}{b}$$

where, p stands for the average dissimilarity during the prediction phase between the new observations and their closest medoid, considered within a time window (e.g. the dissimilarity of all observations which arrived in the last 30 seconds); and b represents the average dissimilarity between the training set and their closest medoid. Note that the computational cost of measuring α is small. More specifically, b is calculated only once, i.e. only during the training phase, simply by: (i) selecting in the dissimilarity matrix the rows of the medoids; (ii) selecting the dissimilarity values corresponding to the observations within a particular medoids's cluster; and (iii) averaging these values. p, instead, is a natural by-product of the prediction of each observation (service) and only requires their cumulative sum within a time window.

The need for retraining, in other words what threshold to use on α is problem-specific. For instance, in cases where even small errors are costly and the services' patterns change often, even small number of different new services might require the retraining of RF+PAM. The proposed $alpha$ enables the manager to adapt it according to their needs and the context. On one hand, the manager defines a threshold such that, when $alpha$ is higher than this threshold, the system rebuilds the forest. On the other hand, the manager defines how much a single service affects the $alpha$ measure by setting the time window for p, i.e. it is calculated considering only the new observations within this time window. Although this solution needs inputs from the manager, which requires insights on the dataset, it also provides enough flexibility to achieve a fair trade-off between prediction quality and retraining costs.

IV. Validation

Our approach has been implemented as an open-source multi-agent framework written in Python[1]. Our tool has both a standalone and a distributed version. In the latter, agents can be placed in different resources and run in parallel, in order to speed up the RF training step taking advantage of cloud resources.

Our experiments are purposely designed to: (i) demonstrate the importance of similarity learning and appreciate clustering quality compared to other methodologies when evaluated on a common dataset; (ii) validate the quality of on-line prediction, which was trained with less data, comparing to a version which had all the data available; and (iii) present a use case, in which we developed a scheduler based on RF+PAM, to demonstrate the practical applicability of our solution in a cloud setting.

We use two datasets in the experiments, the first 12 hours of a publicly available dataset released by Google [12] and a dataset from a grid platform.

Specifically, the Google dataset contains traces from one of Google's production clouds with approximately 12500 servers. The data consists of monitoring data of services in 5 minutes intervals. To illustrate the content of the dataset we list some features: CPU and memory usage, number of tasks, assigned memory, unmapped page cache memory, disk I/O time, local disk space, task requirements and priority. The complete list of the features can be found in [12].

The second dataset, made available by the Grid Workload Archives [13], contains the traces of a grid of the Dutch Universities Research Testbed (DAS-2)[2] with approximately 200 nodes. This dataset consists of requests of resources to run services and has over 1 million observations. Among the features available are: Average CPU Time, Required Time, User ID, Executable ID and Service Structure.

A. Demonstrating the quality of RF based similarity learning

In this section, we evaluated the use of RF for unsupervised similarity learning in the autonomic cloud domain in an off-line setting, i.e. all observations are available for the training of the forest. In particular, we compared the clustering quality of our solution with two methodologies that used the Google's cloud dataset. Since these methodologies use K-means, for a fair comparison and to illustrate the importance of similarity learning, we compare here with the variant of our approach, RF+K-means, which relies on K-means.

The first methodology (**Mt1**) [14] is divided into four steps: (i) selection and preparation of the features; (ii) application of the off-the-shelf K-means clustering algorithm to construct preliminary classes; (iii) definition of the break points for the qualitative coordinates based on the results of the second phase and (iv) merging of close adjacent clusters.

While applying Mt1 in the Google dataset, the authors selected the *CPU* and *Memory* features, transformed into normalised per hour values, and the *Duration* was normalised and converted into seconds. In the second step, they heuristically defined 18 classes that represent the combination of: *Small*, *Medium*, *Large* for *CPU* and *Memory*, and *Small* and *Large* for *Duration*, and clustered the data points using K-means. In the third step, they employed these definitions and the clustering results to define the break points to separate the observations and, in the fourth step, they merged adjacent classes ending up with 8 clusters. Evidently, Mt1 cannot be deployed as a

[1]Our implementation is available in http://sourceforge.net/projects/rfpam/ along with the framework employed in our use-case, and the results and additional information about the experiments reported herein.

[2]http://www.cs.vu.nl/das2/

general solution for autonomic clouds given the necessary expert interventions. However, since it uses the same dataset, it was considered here for comparison.

The second methodology (**Mt2**) [15] is defined as follows: (i) selection of the continuous (numerical) features; (ii) creation of new features based on the existing ones (even if redundant); (iii) normalisation of data and (iv) clustering the data using K-means.

Mt2 has been applied to the considered dataset by defining the number of clusters as 8. It is clear that in Mt2 the categorical values are ignored and that the careful selection of the features is critical; this deviates from the approach proposed in this paper, which aims at offering a robust and flexible solution that can accommodate many different settings.

Both methodologies employ K-means for clustering. Therefore, for a fair comparison and to demonstrate the gain from defining a dissimilarity matrix (i.e. learning the similarity between observations), we use as clustering algorithm K-means rather than PAM. Hence, as already described in Section III-C, we used the dissimilarity matrix, generated by the unsupervised RF similarity learning, as the input for the MDS algorithm, and the resulting Euclidean points as input for K-means clustering.

For all experiments, we defined the number of clusters as 8 (as Mt1 and Mt2 did). We considered two variants of the original dataset, dropping certain features in each case: *Dataset 1* prepared for Mt1 (see the methodology definition), and *Dataset 2* which contains only all continuous features of the original dataset (i.e. categorical ones are excluded), which is used by Mt2. Then, we applied our methodology based on RF to both datasets to compare its cluster quality with the other two methodologies.

Clustering quality measures. Unlike supervised classification where several measures to evaluate performance exist, clustering has no widely accepted measure. For Mt1, the authors used the Coefficient of Variation (CV), i.e. the ratio of the standard deviation to the mean. However, since each data dimension has a different CV, which requires an unwieldy multi-dimensional comparison with large dimensions, such interpretation is far from straightforward [15]. Therefore, in alignment with approaches in the clustering literature, here we report some of the most popular indicators for evaluating and comparing clustering results.

Connectivity indicates the degree of connectedness of the clusters. The measure has a value between 0 and ∞, with 0 being the best. *Dunn* index is the ratio of the shortest distance between data points in different clusters by the biggest intra-cluster distance (a high Dunn index is desirable). *Silhouette* measures the degree of confidence in the assignment of an observation; better clustering has values near 1, while bad clustering -1 (in the literature some works point out that over 0.75 is the best class for an observation). These indicators (and others) are analysed in [16], which recommends the Silhouette measure for the evaluation of noisy datasets.

Table II summarises the results of the experiments on the methodologies detailed above. These results show that RF+K-means performed considerably better on both datasets, considering any of the evaluation criteria, when comparing to the other two methodologies. Similarity learning here outperforms the other approaches, leading to better defined clusters, even when projected to the Euclidean space with MDS. These results also demonstrate that our approach works well in the considered application domain. We should also note that, for a fair comparison, only the continuous features of the datasets were used, although our RF solution is able to handle also categorical ones.

TABLE II
QUALITY OF CLUSTERING WITH RF+K-MEANS.

	Dataset 1		Dataset 2	
	Mt1 [14]	**RF+K-means**	**Mt2** [15]	**RF+K-means**
Connec.	53.33	33.42	32.26	25.89
Dunn	0.01	0.08	0.06	0.15
Silhou.	0.67	0.98	0.89	0.99

B. Evaluating the RF based on-line prediction

To assess the performance of the on-line prediction of RF+PAM, we conducted experiments to verify the agreement between two set-ups of the algorithm: a *benchmark* set-up where all the data are available for training/prediction, and another set-up where only a subset of data is available for training and the remaining data is used for testing. We use the former set-up to obtain a ground truth cluster assignment, since all information is available and we cannot expect the algorithm (with less data for training) to perform better than that. We evaluate the on-line prediction by measuring whether unseen observations (not included in the training set) ended up in the same cluster as assigned by the benchmark set-up that gave the ground truth. Thus, accuracy in this context is measured as the agreement in the cluster assignment.

In the experiment, we first use all observations and obtain the cluster assignments for the benchmark set-up. We proceed carrying out a K-Fold cross-validation strategy to evaluate the agreement. K-Fold cross-validation divides the dataset in K partitions. It reserves one partition for testing and uses the other $K-1$ for training the trees and learning the similarities and clusters. We execute the following steps K times, every time using a different K-th partition:

1) Train a forest using the data in the $K-1$ partitions and obtain cluster assignments;
2) Predict the cluster assignment of the observations belonging to the K-th partition using the on-line RF methodology;
3) Compute the Adjusted Rand Index (see below for details) between the results of steps 2 and the ground truth of the benchmark set-up.

To illustrate the power of PAM, we compare the results of the above process, using RF+PAM and RF+K-means. A measure of quality for comparison of clustering methodologies is the Adjusted Rand Index (ARI), which quantifies the agreement of the clusters produced by each methodology. The maximum value 1 indicates that two results are identical (complete agreement); value 0 indicates results equivalent to random; the minimum value -1 indicates completely different results. For more details, we refer to [17].

TABLE III
CLUSTERING AGREEMENT RESULTS BETWEEN THE GROUND CLUSTER AND THE PROPOSED METHODOLOGIES.

	Google Dataset		DAS-2 Dataset	
K	**RF+PAM**	**RF+K-means**	**RF+PAM**	**RF+K-means**
100	0.81 (0.32)	0.50 (0.37)	0.70 (0.23)	0.52 (0.21)
50	0.75 (0.19)	0.45 (0.19)	0.68 (0.17)	0.54 (0.18)
20	0.73 (0.09)	0.43 (0.11)	0.67 (0.11)	0.47 (0.08)
10	0.70 (0.06)	0.43 (0.13)	0.63 (0.09)	0.44 (0.09)
5	0.69 (0.05)	0.42 (0.06)	0.61 (0.07)	0.41 (0.01)

Table III presents the results of the experiments considering both Google and DAS-2 datasets. The results are averaged over all K-Folds and presented along with the standard deviation (reported within parenthesis). RF+PAM performs significantly better in the tests. This difference is due to the reliance of the K-means version on MDS to lower the dimensions and construct a Euclidean distance. Since many features are used, the dimensionality reduction step and embedding the observations in linear space (from unfolding the higher dimensional manifold), achieved with MDS, lead to poorer separability of the clusters.

Notably, these datasets are examples of real-monitoring data from the cloud domain and are not (manually) prepared (via, e.g. transformation or removal of categorical features). When comparing the results of the two datasets we see clear improvements with high dimensional data (Google's dataset). It indicates that RF is able –without heuristic or manual expert intervention to prepare the dataset– to benefit from the additional information contained in the features to cluster the data (through similarity learning) and can, dynamically, adapt to scenarios where the relation among features changes.

To further highlight the importance of the categorical features in aiding cluster separability, we repeated the previous experiments excluding categorical features from the datasets. In this case the ARI of the on-line and off-line solutions using the datasets without the categorical variables was 10 - 15% when compared to use all features. This suggests that the information contained in these features is important to improve clustering quality and the performance of the on-line solution. In this regard, we re-emphasise the importance of accommodating a mix of categorical and continuous features in autonomic cloud computing.

C. Cloud Use Case

To demonstrate the applicability of the on-line RF+PAM methodology in a practical real AC setting, we propose a scheduling algorithm based on the similarity between services. Intuitively, the scheduler assigns an incoming service to the node executing the most dissimilar services, thus avoiding race conditions for the node's resources. For each node, the scheduler averages the dissimilarity between the new service and the services running in that node, then it assigns the service to the node with the highest average dissimilarity. In these experiments, the calculation of the dissimilarity among services is based on their SLAs.

The scheduling steps are detailed in Algorithm 1. The scheduler receives as parameter the new service (*nSer*) and the list of nodes (*node_list*), which also contains the list of the services running in each node. Then, it clusters the new service and calculates, for each node, the dissimilarity between the new service and all services running in that node. According to the RF+PAM methodology, this dissimilarity is calculated between the new service and the cluster medoids of the running services. Then, if there is at least one service running in the node, the total dissimilarity is divided by the number of services. Otherwise, since no service will compete for the same resource, the dissimilarity for the node is defined as 1.1 to prioritise it in the assignment phase (as the maximum dissimilarity is 1).

Algorithm 1 Calculate the dissimilarity between a new service and the services running in the nodes of the cloud.

```
1:  procedure CALCULATE_DISSIMILARITY(nSer, node_list)
2:      nSer.c ← CLUSTER_SERVICE(nSer.SLA)
3:      for node in node_list do
4:          node_dissi ← 0
5:          for s in node do
6:              d ← dissimilarity(nSer, s.c)
7:              node_dissi ← node_dissi + d
8:          if node_dissi > 0 then #Average Dissimilarity
9:              node_dissi ← node_dissi / len(node.Sers)
10:         else #No Services in the node, best case
11:             node_dissi ← 1.1
12:         nodes_dissi.append([node,node_dissi])
13:     ASSIGN_SER(nSer, nodes_dissi)
```

In the assignment phase, the scheduler assigns the service to the node with most dissimilar services, after verifying whether it has enough resources to run the service. When no node is available, the service joins a waiting list. When a service terminates, the scheduler selects the compatible service from the waiting list with the highest dissimilarity to the services running in the node (not considering the terminated ones).

We employed these concepts in a framework that coordinates the execution of services. In our use case, services are applications executed in a cloud defined by SLAs, which specify service descriptions and quality of service requirements using the SLAC language [18]. The framework has a central scheduler that receives service requests and schedules them. More specifically, when it receives a new service, it communicates with the RF+PAM implementation to request the clustering of the service, which is performed based only on the features specified in its SLA. Other information, such as monitoring data, can be used for the clustering process; however, since in our use case most services have a short duration, collecting and analysing monitoring data would generate a high overhead. With the clustering knowledge, the scheduler assigns the service to the corresponding node and sends it to an agent deployed in that node. Afterwards, it configures the monitoring system (in our framework we exploited Panoptes [19]) to monitor the services and to send the collected information to the scheduler. Finally, the scheduler uses this monitoring information to manage the services, as depicted in Figure 3.

The experiments were conducted in a cloud using the

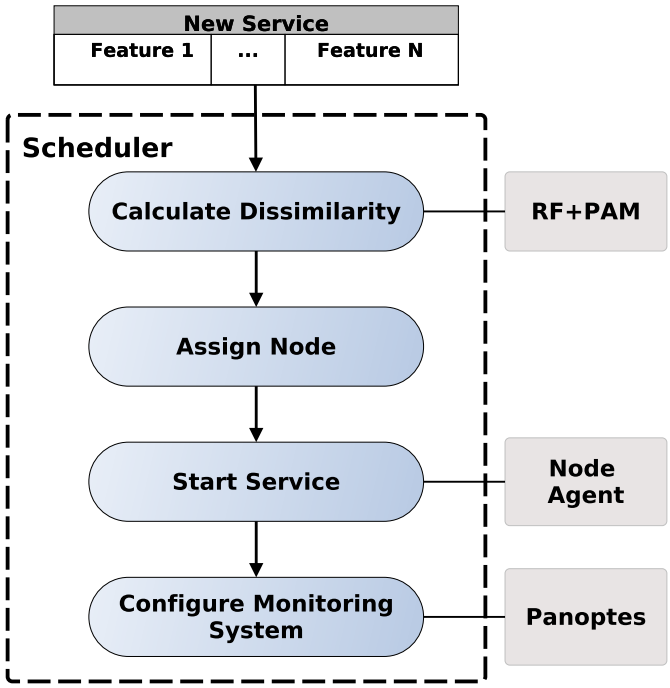


Fig. 3. Similarity Scheduling in the developed framework.

OpenNebula[3] tool and 2 physical machines with different hardware configurations, providing 14 heterogeneous VMs in which the agents of the framework are employed to execute services.

To assess the performance of the Dissimilarity scheduling, three other scheduling algorithms were used. In the first scheduler, named *Isolated*), each service runs without any other service in the same VM, thus having all resources available for the execution of the service. This algorithm was implemented to serve as the lower bound of the results, i.e. the best possible case since the services are executed without interference from each other. The second algorithm, the *Least Loaded*, assigns services to the node with the least number of services. Finally, the third scheduler (named *Random*) assigns the services randomly to the nodes. Notably, all four algorithms rely on the resource admission control, so that services are assigned only to machines that have enough resources to run them.

In the experiments, the services are generated based on the distribution of the Google's cloud dataset [12] at the beginning of every round of tests and the same services are executed using all four described algorithms. Each service has an associated SLA, which is generated along with the service, based on an estimation of the resources necessary to finish the service within the completion time. The created features are: CPU, RAM, Requirements, Disk Space, Completion Time and Network Bandwidth. The services in the experiments are of different types, such as web crawling, word count, machine learning algorithms, number generation and format conversion, which are close to real-world applications [20].

In real-world clouds, services arrive in variable intervals. In our scenario, we assume that the services' arrival is a Poisson process, i.e. the time between consecutive arrivals has an exponential distribution with parameter λ. Intuitively, the higher λ is, the more often services arrive; e.g. for λ set to 0.2 a service arrives in average every 5 seconds, while for λ set to 1 the same happens on every second. We created three different scenarios varying the value of λ to analyse the performance of the algorithms with different loads: Low ($\lambda = 1$), Medium ($\lambda = 2$) and High ($\lambda = 3$) arrival rates.

The algorithms were run 10 times, each execution using the same services for all algorithms. Since we considered different numbers of services, we executed this method for each number of services in each of the three scenarios.

Figure 4 shows the total execution time of services in the different scenarios considering different number of services.Notably, such overall time does not correspond to the time necessary to execute (possibly in parallel) all services, but it is calculated by summing the execution time of each single service. The Dissimilarity scheduler performs significantly better than the Random (in average 59%) and the Least Loaded (in average 36%). Note also that Dissimilarity is in average only 14% worse than lower bound, i.e. the Isolated scheduler: a scheduler that assumes ideal, but impractical and wasteful, conditions where a service is executed alone.

The results of all experiments suggest that the Dissimilarity scheduler performs better when the cloud is not overloaded since it has more options to allocate services in the node with the most dissimilar ones. However, even with high arrival rates (worst case scenario for this scheduler) and with a high number of input services, our solution performs significantly better than the Random and the Least Loaded ones as it allocates the services that use different resources together. This approach reduces the competition for the resources of the node, thereby improving the cloud's performance.

In real-world deployments, other aspects of services, such as service priority or SLA violation probability, must be considered for designing a scheduler. Yet, our results suggest that more complex schedulers can benefit from integrating dissimilarity scheduling in their solutions.

1) SLA Violations: The performance comparison previously discussed provides insights on the performance advantages of our scheduler. Here, we show the impacts of RF+PAM on a production system from the viewpoint of quality of service. To this aim, we used the same set-up to compare the algorithms considering the number of SLA violations. Moreover, we exploited the SLAC Framework [18] to support the specification and evaluation of SLAs.

For each service, an associated SLA is created based on an estimation of the resources necessary to complete the service within the specified completion time. This estimation was performed beforehand through a benchmarking of each type of service using different resources of the cloud. The SLA metrics in the service definition are: CPU, RAM, Requirements, Disk Space, Completion Time and Network Bandwidth.

After a service is scheduled, the framework configures the monitoring system to collect information about the new service and to send the collected information to the SLAC Framework, which periodically evaluates its SLA.

[3]http://opennebula.org

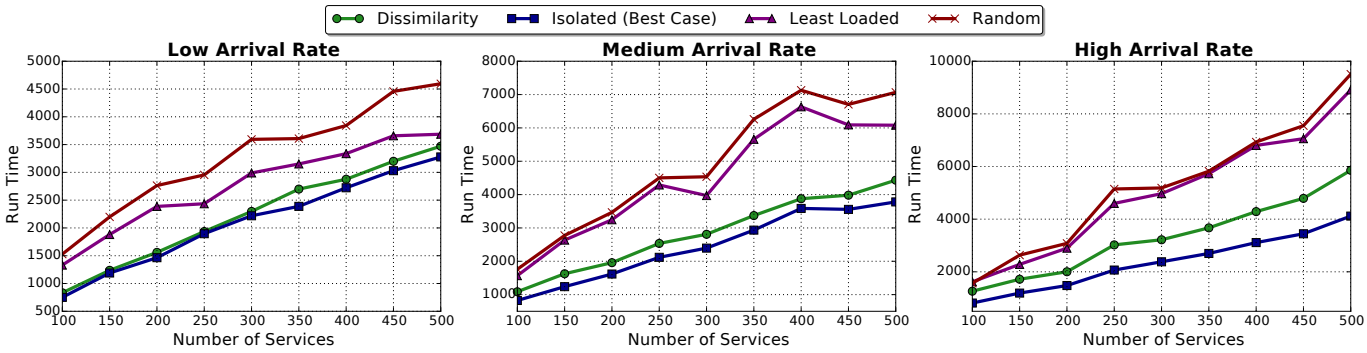


Fig. 4. Total run time of the scheduling algorithms in different scenarios.

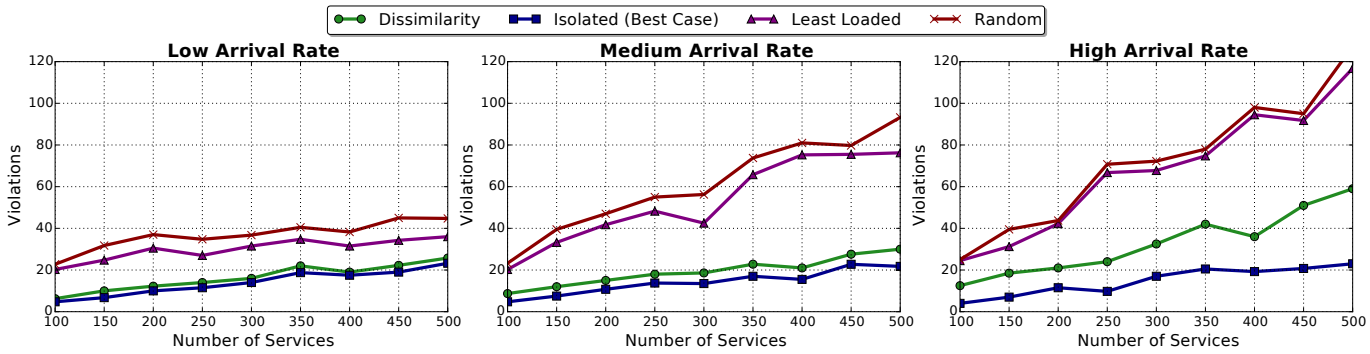


Fig. 5. Total number of SLA violations for the scheduling algorithms in different scenarios.

TABLE IV
AVERAGE NUMBER OF SLA VIOLATIONS AND VIOLATION REDUCTION OF THE DISSIMILARITY COMPARED TO THE LEAST LOADED SCHEDULER IN DIFFERENT SCENARIOS.

Scenario	Random	Least Loaded	Dissimilarity	Violation Reduction
Low	36.8	30.0	16.3	45%
Medium	60.9	53.2	20.1	62%
High	72.1	67.8	32.9	51%

The results of these experiments are shown in Figure 5 and Table IV. Using the Dissimilarity methodology, the SLA violations were reduced up to 62% and on average by 52%. The experiments again confirm that the Dissimilarity scheduler performs better when the cloud is not overloaded.

We also performed experiments using different numbers of clusters, namely 4, 6 and 10, in the same use case. However, due to the nature of our use case, in which we try to identify the server with most different services, the results of the scheduler with different numbers of clusters are similar to the results of these experiments. Therefore, we do not show these additional results in the paper.

2) Detecting new Tendencies on Services via the alpha Measure : Our methodology can be beneficial for the autonomic management of the cloud system also in the long run, where the arrival of many new services may affect the accuracy of predictions. Indeed, as explained in Section III-D, the accuracy can be preserved by retraining the forest. Thus, to provide insights of the proposed retraining measure $alpha$, we conducted further experiments using the previous scenario.

In these experiments, we calculate the measure α of the new services and the number of SLA violations. To emphasise the need of retraining, after a fixed amount of time we purposely modify the distribution of the SLA metrics of the generated services to create a new tendency. This is done to demonstrate in practice the changes in α and their impact on the number of SLA violations. Also, in the experiments, to show that after retraining the approach well adapts to the new tendency, we retrain the forest after detecting the change using α. In particular, we retrain the forest using the new services together with the services used in the training phase, using a buffer of 2 minutes.

We compare our Dissimilarity algorithm with the Random one, because the latter does not need retraining and does not rely on prior knowledge, i.e. it should be unaffected by the change of patterns. The test considered 600 services, in a the High arrival rate scenario, and using a time window of 30 seconds. We run the experiments 5 times and averaged the results.

The results are reported in Figure 6. The number of violations with the Dissimilarity scheduler after the new tendency (change of patterns) grew considerably. After retraining, the number of violations was considerably reduced, achieving similar performance as to the one prior to the new tendency. This suggests that the algorithm adapted well to changing tendencies. Violations with the random algorithm, instead, remained almost stable: it performed poorly before and maintained poor performance even with the new tendency. In any case, our Dissimilarity scheduler always performed better than

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2016.2569000, IEEE Transactions on Network and Service Management

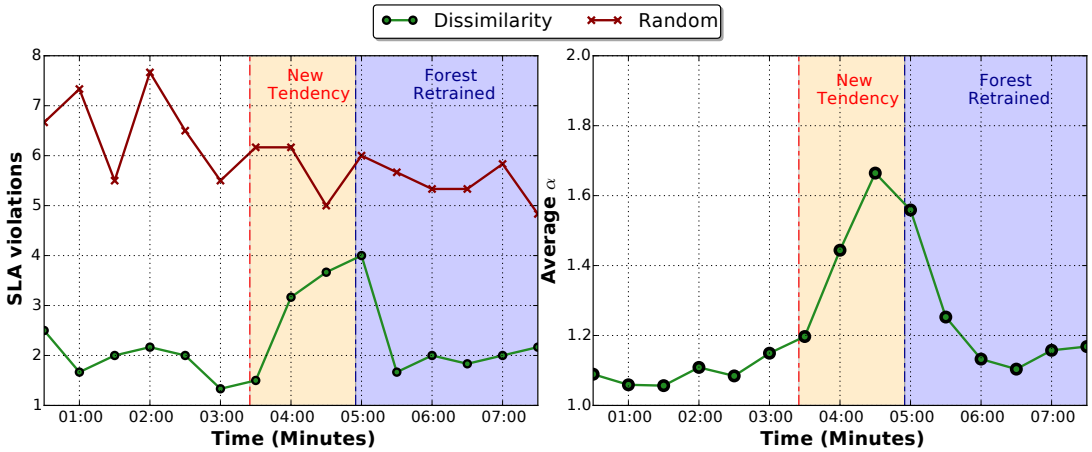


Fig. 6. Number of SLA violations and α measure trends in presence of patterns change and after retraining the forest.

the Random one. Remarkably, the increase in SLA violations for the Dissimilarity scheduler is also captured within the α retraining measure (Figure 6, plot in the right-hand side), which is completely agnostic of SLA function and concept. Observe how the α measure changes trend (increases) as soon as new patterns appear and it is correlated with the number of violations. This shows that α can be used as a suitable proxy for the number of SLA violations, and that by setting a threshold the manager can decide at which point to retrain the forest in order to prevent higher number of SLA violations. Interestingly, similar analysis can be done for any metric suitable to the application (for example, power consumption or renting cost) and a manager can set suitable thresholds on α for that. This demonstrates the versatility of the proposed framework.

V. Related Works

A. Methodology

Among the unsupervised solutions, according to the domain requirements, we look for an algorithm that can process data fast, handle mixed types, support a high number of features, and, ideally, process data in an on-line fashion. For an extensive review on the existing unsupervised learning techniques we refer to [21] and, specifically for on-line clustering, to [22], [23].

Clustering: Few existing clustering solutions handle mixed types of data (e.g. [24], [25], [26]). In particular, the majority of the existing on-line clustering algorithms which handle mixed data types cannot handle cases with a large number of features. For example, the HClustream [26] algorithm presents poor performance results even with 10 features [27].

Another common approach to deal with mixed data types is to devise data-driven solutions that can learn similarities among observations (we refer to [28] for a detailed review on them). However, these solutions either require information a priori about the data (known as supervised similarity learning), which is not available in our context, or are computationally intensive and hence do not scale well to large-scale systems.

Random Forest: Related works which propose or employ unsupervised RF (e.g. [29], [30], [31], [32], [5]) need to re-execute the whole clustering process for each new observation to obtain the dissimilarity, which obviously is impracticable due to the high overhead of this process (it may take several minutes to cluster a single new observation). In Section IV-B we demonstrate quality of our clustering methodology in comparison to the standard off-line solutions. The most known on-line adaptations of the RF algorithm ([33], [34], [35], [36], and even the most recent one [37]) are computationally demanding and cannot make fast *predictions*. Moreover, all these approaches are devised for supervised learning, which is incompatible and difficult to adapt to the unsupervised approach since: (i) they create pruned trees with maximum depth; and (ii) the observations in intermediate leaves should be re-parsed on every new split and the observations re-clustered.

B. Applications of Clustering and Similarity in Clouds

Here, we discuss the relevant literature in the cloud domain that uses a notion of similarity to support decision systems with knowledge. Note that this paper's focus is not on establishing a scheduler, a service profiling methodology, etc., but to propose a service clustering methodology supporting autonomic management of clouds, and to show its applicability in the domain. Thus, we may think the reviewed approaches using the concept of dissimilarity as possible applications of our methodology.

Scheduling: In the *service scheduling* field, several works, e.g. [38], [39], [40], [41], [42], use a measure of similarity. However, they consider only numerical features and, as discussed in Section II, the domain requires the support of different types of features. In [20], the authors manually combine features and employ a supervised Incremental Naive-Bayes classifier to assign a service. However, this approach depends on the hand-crafted combination of features, which is problem-specific, and on several parameters defined by the administrators.

In [43], the authors propose an interesting scheduling approach, named Paragon, that supports heterogeneity of re-

sources and services, and analyse the interference of co-scheduling services into the same resources. They employ a different notion of similarity based on the performance of services, borrowing ideas from collaborative filtering. Similarity is considered based on how a service x has run on hardware specification A. By pooling (on training time) these values, they can extract recommendations for scheduling assignment relying on SVD – a powerful decomposition technique, which is computationally demanding and whose distributed versions, beyond recent developments in randomised SVD, are not common. Instead, by relying on random forest (and trees), our approach can be trained in parallel and trees have many inherent mechanisms to deal with missing values built-in, not requiring iterative refinements as the approach in Paragon necessitates. Moreover, the solutions can be used together since the similarity notion of Paragon can also be built upon our solution.

Service Profiling: Most approaches in this are are problem-specific, e.g. [44], [45] focus only on VMs. Hence, they cannot cover the diversity of the services and the heterogeneity of clouds. The solution of Kahn et al. [46] on *workload characterisation* clusters workload patterns by their similarity. However, their similarity clustering algorithm is based on simple heuristic metrics to accommodate VMs, which does not cope with the dynamism of the AC domain.

Anomalous Behaviour Detection: In [47], [48], the authors use a heuristic notion of similarity to cluster service requests and detect anomalous behaviours. Similarly, Wang et al. [49] propose a methodology to detect anomalies for Web applications in which the similarity among the workloads is used to detect problematic requests. However, those works do not consider different types of features.

In summary, most works in cloud which employ a notion of similarity implicitly assume: homogeneity on the resources and services; preparation and normalisation of the data for the clustering process; and good representation of the relations of data features. Our clustering solution, instead, does not rely on these assumptions and is not problem-specific. Thus, it can be used with any kind of service. Therefore, we advocate that our solution, or an adaptation of our approach, could significantly improve decision-making in autonomic clouds.

VI. Conclusions

In this paper, we developed a methodology to feed autonomic cloud managers with knowledge on the similarities among services and their clusters. This knowledge has a wide range of applications in the domain, e.g. for anomalous behaviour detection, service profiling and service scheduling.

To feed the autonomic managers with such knowledge, we devised a novel clustering methodology based on RF and PAM. We validated it through several experiments, which used real-world cloud datasets. Our methodology shows significant benefits: superior performance, low memory footprint, support for mixed types of features, support for a large number of features and fast on-line prediction. To demonstrate its applicability in the domain, we implemented and tested a scheduling algorithm for a test-bed cloud, which uses the notion of similarity to assign incoming services.

As future work, we will investigate the characteristics of RF, such as variable importance and feature selection, to improve our methodology. Moreover, we plan to apply our solution to the management of services, utilising RF+PAM to dynamically calculate the SLA violation risk. Finally, we intend to improve the dissimilarity scheduler in order to consider other aspects of the cloud domain, including the sensitivity of the services to hardware configurations and priorities.

VII. Acknowledgements

This work has been partially sponsored by the MIUR PRIN project CINA (#2010LHT4KM), by CNPq through the Science Without Borders programme, and by a Marie Curie Action: "Reintegration Grant" (grant #256534) of the EU's Seventh Framework Programme. We would like to thank Rocco De Nicola and the CCGrid's reviewers for their encouraging and fruitful comments.

References

[1] P. Horn, "Autonomic Computing: IBM's Perspective on the State of Information Technology," 2001.

[2] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[3] R. B. Uriarte, S. Tsaftaris, and F. Tiezzi, "Service clustering for autonomic clouds using random forest," in *Proc. of CCGrid 2015*. IEEE, 2015, pp. 515–524.

[4] R. Caruana and A. Niculescu-Mizil, "An empirical comparison of supervised learning algorithms," in *Proc. of the 23rd ICML*. ACM Press, 2006, pp. 161–168.

[5] L. Breiman and Adele Cutler, "Random forests Manual V3.1," 2003. [Online]. Available: https://www.stat.berkeley.edu/~breiman/Using_random_forests_V3.1.pdf

[6] L. Kaufman and P. Rousseeuw, *Finding groups in data: an introduction to cluster analysis*. New York, NY: Wiley, 1990.

[7] S. Lloyd, "Least squares quantization in PCM," *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 129–137, Mar. 1982.

[8] T. Cox and M. Cox, *Multidimensional scaling*. London;UK: Chapman & Hall, 1994.

[9] L. Wang, C. Leckie, K. Ramamohanarao, and J. Bezdek, "Automatically determining the number of clusters in unlabeled data sets," *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, no. 3, pp. 335–350, 2009.

[10] G. W. Milligan and M. C. Cooper, "An examination of procedures for determining the number of clusters in a data set," *Psychometrika*, vol. 50, no. 2, pp. 159–179, 1985.

[11] A. Bifet and R. Gavaldà, "Learning from Time-Changing Data with Adaptive Windowing," *SDM*, pp. 443–448, 2007.

[12] C. Reiss, J. Wilkes, and J. L. Hellerstein, "{Google} cluster-usage traces: format + schema," Google Inc., Mountain View, USA, Technical Report, nov 2011. [Online]. Available: https://googleclusterdata.googlecode.com/files/Google%20cluster-usage%20traces%20-%20format%20%2B%20schema%20(2011.10.27%20external).pdf

[13] A. Iosup, H. Li, M. Jan, S. Anoep, C. Dumitrescu, L. Wolters, and D. H. Epema, "The Grid Workloads Archive," *Future Generation Computer Systems*, vol. 24, no. 7, pp. 672–686, Jul. 2008.

[14] A. K. Mishra, J. L. Hellerstein, W. Cirne, and C. R. Das, "Towards characterizing cloud backend workloads," *ACM SIGMETRICS Performance Evaluation Review*, vol. 37, no. 4, p. 34, Mar. 2010.

[15] Y. Chen and A. Ganapathi, "Analysis and lessons from a publicly available google cluster trace," ECS Department, University of California, Berkeley, Tech. Rep., 2010. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2010/EECS-2010-95.pdf

[16] J. Handl, J. Knowles, and D. B. Kell, "Computational cluster validation in post-genomic data analysis." *Bioinformatics (Oxford, England)*, vol. 21, no. 15, pp. 3201–12, Aug. 2005.

[17] L. Hubert and P. Arabie, "Comparing partitions," *Journal of classification*, vol. 218, pp. 193–218, 1985.

[18] R. B. Uriarte, F. Tiezzi, and R. D. Nicola, "SLAC: A Formal Service-Level-Agreement Language for Cloud Computing," in *Proc. of the 7th IEEE/ACM UCC*. IEEE, Dec. 2014, pp. 419–426.

[19] R. B. Uriarte and C. B. Westphall, "Panoptes: A monitoring architecture and framework for supporting autonomic Clouds," in *Proc. of the 16th IEEE/IFIP NOMS*. Krakow, Poland: IEEE, 2014.

[20] R. Nanduri, N. Maheshwari, A. Reddyraja, and V. Varma, "Job Aware Scheduling Algorithm for MapReduce Framework," in *Proc. of the 3rd IEEE CloudCom*. IEEE, Nov. 2011, pp. 724–729.

[21] R. Xu and D. Wunsch, "Survey of clustering algorithms." *IEEE Transactions on Neural Networks*, vol. 16, no. 3, pp. 645–78, May 2005.

[22] A. Quiroz, M. Parashar, N. Gnanasambandam, and N. Sharma, "Design and evaluation of decentralized online clustering," *ACM Transactions on Autonomous and Adaptive Systems*, vol. 7, no. 3, pp. 1–31, Sep. 2012.

[23] M. Gaber, A. Zaslavsky, and S. Krishnaswamy, "Mining data streams: a review," *ACM Sigmod Record*, vol. 34, no. 2, pp. 18–26, 2005.

[24] A. Ahmad and L. Dey, "A k-mean clustering algorithm for mixed numeric and categorical data," *Data & Knowledge Engineering*, vol. 63, no. 2, pp. 503–527, Nov. 2007.

[25] M.-S. Yang, P.-Y. Hwang, and D.-H. Chen, "Fuzzy clustering algorithms for mixed feature variables," *Fuzzy Sets and Systems*, vol. 141, no. 2, pp. 301–317, Jan. 2004.

[26] C. Yang and J. Zhou, "HClustream: A Novel Approach for Clustering Evolving Heterogeneous Data Stream," *Proc. of the 6th IEEE ICDMW*, pp. 682–688, 2006.

[27] C. Aggarwal, J. Han, J. Wang, and P. Yu, "A framework for projected clustering of high dimensional data streams," in *Proc. of the 30th VLDB*, vol. 30, 2004, pp. 852–863.

[28] L. Yang and R. Jin, "Distance metric learning: A comprehensive survey," Michigan State Universiy, Tech. Rep., 2006. [Online]. Available: https://www.cs.cmu.edu/~liuy/frame_survey_v2.pdf

[29] H. Albehadili and N. Islam, "Unsupervised Decision Forest for Data Clustering and Density Estimation," *CoRR*, vol. abs/1507.0, 2015.

[30] E. Allen, S. Horvath, F. Tong, P. Kraft, E. Spiteri, A. D. Riggs, and Y. Marahrens, "High concentrations of long interspersed nuclear element sequence distinguish monoallelically expressed genes." in *Proc. of the National Academy of Sciences of the United States of America*, vol. 100, no. 17, Aug. 2003, pp. 9940–5.

[31] T. Shi, D. Seligson, A. S. Belldegrun, A. Palotie, and S. Horvath, "Tumor classification by tissue microarray profiling: random forest clustering applied to renal cell carcinoma." *Modern pathology : an official journal of the United States and Canadian Academy of Pathology, Inc*, vol. 18, no. 4, pp. 547–57, Apr. 2005.

[32] T. Shi and S. Horvath, "Unsupervised Learning With Random Forest Predictors," *Journal of Computational and Graphical Statistics*, vol. 15, no. 1, pp. 118–138, Mar. 2006.

[33] O. Hassab Elgawi, "Online random forests based on CorrFS and CorrBE," *Proc. of IEEE Computer Vision and Pattern Recognition Workshops*, pp. 1–7, 2008.

[34] A. Saffari, C. Leistner, J. Santner, M. Godec, and H. Bischof, "On-line Random Forests," in *Proc. of 12th IEEE ICCV*. IEEE, Sep. 2009.

[35] H. Abdulsalam, D. B. Skillicorn, and P. Martin, "Streaming random forests," in *Proc. of the 11th IDEAS*, 2007, pp. 643–651.

[36] O. Hassab Elgawi and O. Hasegawa, "Online incremental random forests," *International Conference on Machine Vision*, 2007.

[37] B. Lakshminarayanan, D. Roy, and Y. Teh, "Mondrian forests: Efficient online random forests," *ArXiv e-prints. Preprint arXiv: 1406.2673*, 2014.

[38] B. Lu and J. Chen, "Grid resource scheduling based on fuzzy similarity measures," in *Proc. of the 2nd IEEE Cybernetics and Intelligent Systems*, 2008, pp. 940–944.

[39] K. Song, S. Ruan, and M. Jiang, "A Flexible Grid Task Scheduling Algorithm Based on QoS Similarity," *Journal of Convergence Information Technology*, vol. 5, no. 7, pp. 161–166, Sep. 2010.

[40] H. Sun, P. Stolf, J.-M. Pierson, and G. D. Costa, "Multi-objective Scheduling for Heterogeneous Server Systems with Machine Placement," in *Proc. of the 14th IEEE/ACM CCGrid*. IEEE, May 2014.

[41] A. Quiroz, H. Kim, M. Parashar, N. Gnanasambandam, and N. Sharma, "Towards autonomic workload provisioning for enterprise Grids and clouds," in *Proc. of the 10th IEEE/ACM International Conference on Grid Computing*. IEEE, Oct. 2009, pp. 50–57.

[42] M. Dabbagh, B. Hamdaoui, S. Member, and M. Guizani, "Energy-Efficient Resource Allocation and Provisioning Framework for Cloud Data Centers," *IEEE Transactions on Network and Service Management*, vol. 12, no. 3, pp. 377–391, 2015.

[43] C. Delimitrou and C. Kozyrakis, "Paragon: QoS-aware Scheduling for Heterogeneous Datacenters," in *Proc. of the 8th ASPLOS*, 2013, pp. 77–88.

[44] T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif, "Black-box and Gray-box Strategies for Virtual Machine Migration." in *Proc. of the 4th NSDI*, 2007.

[45] A. V. Do, J. Chen, C. Wang, Y. C. Lee, A. Y. Zomaya, and B. B. Zhou, "Profiling Applications for Virtual Machine Placement in Clouds," *Proc. of the 4th IEEE Cloud*, pp. 660–667, Jul. 2011.

[46] A. Khan, X. Yan, S. Tao, and N. Anerousis, "Workload characterization and prediction in the cloud: A multiple time series approach," in *Proc. of the IEEE/IFIP NOMS*. IEEE, 2012.

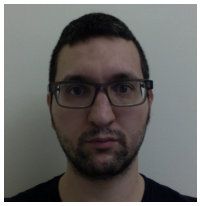
[47] H. Mi, H. Wang, Y. Zhou, M. R.-T. Lyu, and H. Cai, "Toward Fine-Grained, Unsupervised, Scalable Performance Diagnosis for Production Cloud Computing Systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 6, pp. 1245–1255, Jun. 2013.

[48] A. Kind, M. Stoecklin, and X. Dimitropoulos, "Histogram-based traffic anomaly detection," *IEEE Transactions on Network and Service Management*, vol. 6, no. 2, pp. 110–121, 2009.

[49] T. Wang, J. Wei, W. Zhang, H. Zhong, and T. Huang, "Workload-aware anomaly detection for Web applications," *Journal of Systems and Software*, Mar. 2013.

Rafael Brundo Uriarte is currently holding a Post-doctoral Research Fellow in the Computer Science and Engineering Department at IMT Institute for Advanced Studies, Lucca, Italy, where he received his Ph.D. in March, 2015. His research interest include cloud computing, autonomic computing, machine learning and distributed systems. He has published in internationally recognised conferences and journals, such as the IEEE Communication Magazine, IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing (CCGrid) and International Conference on Utility and Cloud Computing (UCC). For further information, please, visit the web-site: http://rafaeluriarte.com/

Sotirios Tsaftaris is a Chancellor's Fellow (at a Senior Lecturer level) at the University of Edinburgh. He is also an Adjunct Professor with the Department of Electrical Engineering and Computer Science at Northwestern University. Previously was with IMT Institute for Advanced Studies, Lucca, Italy and the Director of the Pattern Recognition and Image Analysis Unit. His research interests are image analysis, image processing, data mining and machine learning, and distributed computing. He has published extensively, particularly in interdisciplinary fields, with more than 100 journal and conference papers in his active record. Additional information is available at http://tsaftaris.com



Francesco Tiezzi is Associate Professor at the Computer Science Division of the University of Camerino. His research activities focus on foundational study of distributed systems, and on application of formal methods for developing and analysing them. Particular attention is paid to the definition of formal bases for technologies supporting service-oriented and autonomic computing. Tiezzi is the author of more than sixty publications in international journals, books or conference proceedings. He has served as Program Chair of WWV'11-'12, MOSPAS'14 and SAC:CM'15-'16, and has been member of the program committees of many international conferences and workshops. Additional information is available at http://tiezzi.unicam.it