Service Clustering for Autonomic Clouds Using Random Forest

Rafael Brundo Uriarte and Sotirios Tsaftaris
IMT Institute for Advanced Studies Lucca, Italy
{rafael.uriarte, s.tsaftaris}@imtlucca.it

Francesco Tiezzi
University of Camerino, Italy
francesco.tiezzi@unicam.it

***Abstract*—Managing and optimising cloud services is one of the main challenges faced by industry and academia. A possible solution is resorting to self-management, as fostered by autonomic computing. However, the abstraction layer provided by cloud computing obfuscates several details of the provided services, which, in turn, hinders the effectiveness of autonomic managers. Data-driven approaches, particularly those relying on service clustering based on machine learning techniques, can assist the autonomic management and support decisions concerning, for example, the scheduling and deployment of services. One aspect that complicates this approach is that the information provided by the monitoring contains both *continuous* (e.g. CPU load) and *categorical* (e.g. VM instance type) data. Current approaches treat this problem in a heuristic fashion. This paper, instead, proposes an approach, which uses all kinds of data and learns in a data-driven fashion the similarities and resource usage patterns among the services. In particular, we use an unsupervised formulation of the Random Forest algorithm to calculate similarities and provide them as input to a clustering algorithm. For the sake of efficiency and meeting the dynamism requirement of autonomic clouds, our methodology consists of two steps: (i) off-line clustering and (ii) 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 the applicability of our approach, we devise a service scheduler that uses the notion of similarity among services and evaluate it in a cloud test-bed.**

I. Introduction

In the cloud computing domain, virtually everything can be provided as an on-line, on-demand service [1]. Together with scalability, heterogeneity and dynamism make clouds advantageous for consumers, but from the provider's perspective, they make clouds difficult to manage and coordinate. Moreover, security and privacy mechanisms also hinder management and optimisation of cloud systems.

A prominent approach to cope with the complexity of cloud systems is autonomic computing [2], which aims at equipping such systems with capabilities to autonomously adapt their behaviour according to dynamic operating conditions. To achieve such 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 provided by clouds restricts the knowledge available to autonomic managers, and, consequently, limits their range of actions. Data-driven approaches, without human knowledge and intervention, can assist the operation of autonomic managers. In particular, machine learning techniques, such as *clustering*, generate knowledge consisting of groups (i.e. clusters) of services with similar resource usage patterns. 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 this approach is that the information about services (called *features*) contains both *categorical* (e.g. virtual machine instance type) and *continuous* (e.g. CPU load) types of data. Current approaches address this problem in a heuristic fashion: they either use only one data type, which reduces distinguishability, or construct combinations of data types by human expert intervention. Both do not cope well with the dynamism of autonomic cloud: when new types of services are introduced they may not be distinguishable or a human intervention is necessary again.

In this paper, we tackle the challenge of providing a truly autonomic and effective management of services in clouds through similarity-based knowledge, calculated by using *all* types of service features. To achieve this, we propose a learning methodology relying on the *Random Forest* (RF) algorithm [3]. We learn from the definition of services or monitoring information and provide the similarities to a clustering algorithm. In particular, for the sake of efficiency and meeting the dynamism requirement of autonomic clouds, our methodology consists of two steps: (1) *off-line clustering*, to learn similarities and obtain the clusters; and (2) *on-line prediction*, to predict to which of the computed clusters an incoming new service belongs.

The main contributions of this paper are: (i) the analysis of the specificities of the Autonomic Cloud (AC) domain and the definition of the requirements of a clustering approach for AC services; (ii) an off-line approach that relies on the RF algorithm to learn the similarities among all observed services, essentially a matrix, which is then provided to an off-the-shelf clustering algorithm to identify clusters; (iii) a cluster parsing to reduce the size of the matrix; which is then used by (iv) the on-line prediction to reduce computational requirements; (v) the performance and accuracy analysis of the proposed methods using real-world datasets; and (vi) 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.

The rest of the paper is organised as follows: Section II discusses the potential uses of the similarity knowledge, introduces the use case 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 Domain Requirements

In this section, we discuss the benefits of the similarity knowledge, present a use case and describe the requirements of the AC domain for the solutions seeking this knowledge.

Similarity is a measure of how alike two services are. We focus on the estimation of this measure based on services definitions, such as Service-Level-Agreements (SLAs), or monitoring information. In the domain, this knowledge is versatile and can either be provided to a clustering algorithm or directly used by autonomic managers. Therefore, it can deal with a wide range of application scenarios. For example, clustering can be employed in autonomic management for service profiling, which dynamically groups services based on their behaviour. It can be used for service scheduling, behaviour prediction, and efficient identification of SLAs violations during their enforcement. Another use concerns anomalous behaviour detection, which aids autonomic managers to detect, e.g., failures and intrusions, by assuming that the majority of the services are normal and looking at the cluster with the most dissimilar services.

Let us consider a motivating example of how similarity knowledge can be used directly by an autonomic manager. A provider enables its consumers to deploy any kind of service in the cloud. Due to security concerns, the autonomic manager relies exclusively on the descriptions and SLAs of services, and on their monitoring information. Let us assume that the autonomic manager notices that service A, which had only one available CPU, violated a term of its SLA (concerning, e.g., the completion time). Afterwards, a new service, named B, was found to be similar to A and is clustered in the same group of service A. Instead of assigning only one CPU to service B, the autonomic manager can then decide to assign two CPUs to avoid SLA violations, as occurred with service A.

To practically demonstrate the benefits of the similarity knowledge, in Section IV-C, we describe the implementation of a scheduling algorithm that assigns new services to the nodes running the most dissimilar services in a cloud test-bed. Therefore, the services allocated together tend to access different resources and, thus, this algorithm reduces the competition for the node's resources (e.g. CPU or disk) and improves performance.

Despite the utility of the similarity and clustering algorithms for the domain, designing or adapting them to the autonomic domain is challenging [4]. Moreover, the cloud domain is characterized by specific properties that hinder the clustering task. We describe below the most relevant characteristics of the autonomic clouds and their impact on this task. Table I summarises these characteristics and relate them to the requirements for service clustering in the domain.

TABLE I
Correspondence between autonomic cloud characteristics and the 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	Loosely-Coupled Parallelism
Heterogeneity	Large Number of Features

Data *security* is one of the biggest barriers for cloud adoption. Approaches to improve security in the domain are commonly based on data cryptography and control of cross-layer transmission of information. To process the data converted with these security measures, a clustering algorithm needs to support different types of features (e.g. discrete, continuous, symbolic). Moreover, as these techniques obfuscate the features of the data, they hinder the manual combination of data types. Therefore, a data-driven similarity learning approach is required.

Cloud systems contain a *virtualization* layer. A potential risk that this layer brings to the domain is the fine-control over the monitoring of resources [5], limiting the management of such systems. In light of this loose control and of the uncertainty added by virtualization, the data is heterogeneous and also often incongruent [6]. These characteristics pose significant challenges towards manual combination of data types.

To offer seemly infinity pool of resources, cloud providers deploy *large-scale* clouds. The massive operational data generated in these environments requires a considerable amount of resources to be processed. The knowledge discovery process should not be invasive, i.e. should not impact on the performance of the cloud services provision. Accordingly, a clustering algorithm should run in parallel to cope with the large quantity of services within an acceptable time (low overhead) and to divide its computational load and to operate close to the data sources, thus reducing the impact in single resources and avoiding unnecessary network traffic.

Clouds are inherently *dynamic*. New resources are constantly added and removed from the infrastructure. Furthermore, the types of services and the requested resources vary over time (also due to the pay-per-use business model employed in clouds). Considering the number of services in the domain (large-scale clouds), the number of clustering requests and their inconstant arrival rate, it is impracticable to re-cluster all services on each request. Hence, on-line prediction for new services is a requirement for clustering algorithms. Moreover, this dynamism is enabled also by the loose coupled nature of the cloud infrastructure; therefore, the parallelisation of the clustering algorithms should also be loosely coupled.

Virtually everything can be provided as a service in the cloud domain. Due to such *heterogeneity*, some types of services might require monitoring data types that may not be

easily converted to continuous numerical data types, which are the ones commonly accepted by the clustering algorithms. Moreover, using only categorical or continuous data types may lead to clusterings that do not distinguish different services and thus may provide inferior performance. Instead, using pre-processing techniques, such as discretisation, normalisation or standardisation, and hand crafting new data types that combine categorical and continuous data types, require human expert intervention and full understanding of the dataset and the relationships among data types. Devising such heuristic solutions in the autonomic cloud domain is even more complex, considering that clouds are dynamic. Indeed, this would require to build a new heuristic every time the autonomic manager faces a new service type. Moreover, these techniques are hindered by security restrictions, virtualization and the variety of services in the cloud domain. To overcome these limitations and, most of all, to avoid manual expert intervention, mixed types of features should be handled by the clustering algorithm.

On the other hand, due to the heterogeneity and complexity of cloud services (e.g. services with 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.

Finally, autonomic computing employs *agents* to enact the self-* properties. A clustering algorithm can benefit from this arrangement by parallelizing 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 seek those that: (i) can handle mixed data types (continuous and categorical) without human expert intervention, (ii) are fast both in the training and prediction phases and (iii) offer superior performance.

In the following, we first discuss our choice for unsupervised clustering with RF to address the above requirements, and then we proceed in defining our methodology for learning similarities based on the RF algorithm, and for clustering services using such obtained similarities.

A. Clustering as unsupervised machine learning

The lack of available labelled datasets and the dynamism of the cloud limit the adoption of supervised learning approaches. Thus, in this paper we adopt *unsupervised learning*, which does not require labelled training data and is used to find structures and patterns in data. For an extensive review on the these solutions we refer to [7] and, specifically on on-line clustering, to [8], [9]. Among the unsupervised solutions, we seek an algorithm that can process data fast, can handle mixed types, and ideally could process data in an online fashion.

Few existing clustering solutions handle mixed types of data (e.g. [10], [11], [12]). Moreover, 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 [12] algorithm presents poor performance results even with 10 features [13].

Another common approach to deal with mixed data types is to devise data-driven solution that can learn similarities among observations[1] (we refer to [14] 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 do not scale well.

Thus, in this paper, we propose a combination of a similarity learning step to discover a proper measure of similarity among observations and a clustering algorithm to group the observations according to this measure of similarity. In light of the domain requirements, as the means to obtain such notion of similarity, we adopt the Random Forest (RF) algorithm [3].

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. It 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 user). 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. For details on the classification and regression algorithms we direct the reader to [3]. In summary, the main characteristics of RF are:

- it can handle mixed 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 [15];
- it is efficient and scales well [15];
- the algorithm is easily parallelizable;
- generated forests can be saved for future use (in our case, on-line prediction).

In [16], Breiman and Cutler proposed an unsupervised version of RF. Intuitively, 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 is labelled as class two; (iv) the trees are trained with the original and synthetic data; and (v) the original data is 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 principle used is the following: the more times two

[1]The features of a member of the data set form a one dimensional vector (the *observation*). In this work, an observation corresponds to a service.

observations end up on the same leaf, the more similar they should be.

More formally, the similarity between two observations $x_m,\ x_n$ (m, n are the indices of the 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_m^i and l_n^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:

$$S(x_m, x_n) = \frac{1}{T}\sum_{i=1}^{T} I(l_m^i = l_n^i) \quad (1)$$

The similarity of all pairs of observations is calculated, which results in an $N \times N$ matrix, named SIM, where N is the number of observations. The *dissimilarity matrix* (which is generated from the similarity matrix by applying $DISSIM_{nm} = \sqrt{1 - SIM_{nm}}$) 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, generating a $N \times T$ matrix (where N is the number of observations and T the number of trees, where 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 used as input to a compatible clustering algorithm, for example, the Partitioning Around Medoids (PAM) [17]. Otherwise, the dissimilarity matrix can be transformed into points in the Euclidean space to be used as input to other clustering algorithms, e.g. the standardized version of K-means [18]. 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 the possible high arrival rate of new observations, the domain requires very low prediction time. The unsupervised RF algorithm (successfully used in [19], [20], [21], [16]) needs to re-execute the whole clustering process for each new observation, which is impracticable in our domain because of the high overhead of this process. The alternative is to use online RF algorithms, which learn similarities and cluster observations in an instantaneous fashion without requiring all data a priori. Unfortunately, the most known adaptations of this approach ([22], [23], [24], [25], and even the most recent one [26]) are computationally demanding and cannot make a fast prediction[2]. Finally, RF has been used for similarity learning in [27] but this solution requires labelled data, which is not available in the AC domain.

Therefore, we propose a novel on-line prediction algorithm based on RF, to fulfill the requirements of the domain.

C. On-Line Prediction with RF

We propose a fast and minimal footprint on-line prediction solution tailored to fulfill the requirements of AC (summarised in Table I). This solution takes advantage of the design of the clustering algorithm and pre-processes the trees in order to permit a fast and low memory implementation.

The outcome of the classical RF similarity learning is the $N \times T$ matrix, where N is the number of observations and T is the number of trees. As N grows, this matrix may grow significantly and have a large memory footprint. We propose a solution which, instead, requires an $M \times T$ matrix, where M is the number of clusters. 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) the forest is built using the training set, which is composed of the original and synthetic data (as described in the previous section); (ii) the original data is parsed through the resulting forest, which yields the dissimilarity and the $N \times T$ matrices; (iii) the dissimilarity matrix is given as an input 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; and (iv) since medoids correspond to actual observations, only the results of the medoids are selected from the $N \times T$ matrix, enabling us to store only the forest and this smaller $M \times T$ matrix, which consists of the references to the leaves where the medoids ended up in each tree.

In the prediction phase of RF+PAM, the new observations are parsed through the forest, and a dissimilarity matrix for the new observations with respect to each medoid is generated. Finally, each new observation is assigned to the cluster whose medoid has the least dissimilarity to the new observation. Intuitively, a new observation is assigned to the cluster of the medoid which this observation ended up in the same leaf most times, considering all trees, i.e. the most similar medoid.

Since we separate training from prediction, and our training happens off-line, naturally we would expect at some point to retrain the forest. The retraining requires the definition of a mechanism to recognize when a forest should be rebuilt. However, this mechanism is problem-specific and depends on the available resources and accuracy requirements. In our context, we propose a simple but effective threshold: a user defined ratio between the number of new observations and the total number of observations used to train the forest.

The benefits of RF+PAM are manifold: it can be trained fast and in parallel; it handles, in a data-driven fashion, mixed data types; and it can provide predictions in a rapid

[2]We use a batch mode RF implementation for training and, thus, we need the observations a priori. We find this to be an adequate solution since the training happens in parallel and when the system has available resources. However, as the amount of monitoring data increases, off-line training can be demanding. We could adapt an on-line RF algorithm for the training phase and still use the on-line prediction algorithm we propose. However, adapting such algorithms is not trivial, as they create intermediate leaves on the trees, which are split when a minimum gain is reached. This approach is incompatible with unsupervised learning as: (i) it creates 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.

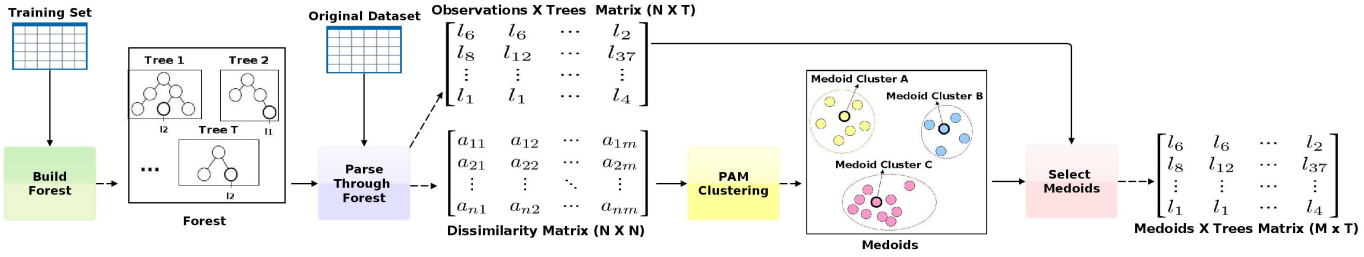

Fig. 1. Training phase of the proposed RF+PAM; notably, only the forest and the $M \times T$ matrix are stored for the prediction phase.

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 utilizes 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 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 standardized 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 [28]. 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 needs to be stored are the $N \times T$ matrix, the forest and the clustering assignments.

The on-line prediction phase of RF+K-means is composed of the following steps: (i) parse the new observations through the trees; (ii) calculate the dissimilarity between the new observations and *all original data* using as the input the $N \times T$ matrix of the original data and the result of step (i), which consists of the references of the leaves in which the new observations ended up; and (iii) assign each new observation to the cluster with the least average dissimilarity between the new observation and all the observations in that cluster. Notably, this solution calculates only the dissimilarity of the new observation 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 $M \times T$ matrix, which is much smaller than the $N \times T$ matrix used by RF+K-means. Moreover, 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.

IV. Experiments

To understand the implications of the solutions described in the previous section, we have implemented them in an open-source multi-agent framework written in Python[3]. This tool has both a standalone and a distributed version. In the latter, agents can be placed in different resources to speed up the RF training step and, thus, take advantage of cloud resources.

Our experiments are purposely designed to: (i) demonstrate the importance of similarity learning and appreciate the clustering quality compared to other methodologies using the same dataset; (ii) validate the quality of on-line prediction, which has been trained with less data, comparing to a version, which has all the data available; and (iii) present a use case to demonstrate the applicability of our solution in the domain.

For datasets, we use the first 12 hours of a publicly available dataset released by Google [29] and of a dataset from a grid.

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 of the available features: CPU and memory usage, number of tasks, assigned memory, unmapped page cache memory, disk I/O time, local disk space, task's requirements and priority. The complete list of the features can be found in [29].

The second dataset, made available by the Grid Workload Archives [30], contains the traces of a grid of the Dutch Universities Research Testbed (DAS-2)[4] with approximately 200 nodes. This dataset consists of the request of resources to run services and has over 1 million observations. Among the features available in the dataset there are: Average CPU Time, Required Time, User ID, Executable ID and Service Structure.

A. Demonstrating the importance 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

[3]Available in http://code.google.com/p/unsupervised-randomforest/ along with the framework employed in our use case.

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

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 use here RF+K-means.

The first methodology (**Mt1**) [31] 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 general solution for autonomic clouds given the necessary man-made interventions. However, since it uses the same dataset, it was considered here for comparison.

The second methodology (**Mt2**) [32] 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, 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 did Mt1 and Mt2). 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 apply our methodology based on RF to both datasets to compare its cluster quality with the other two methodologies.

Clustering quality measures: Notably, 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, this requires an unwieldy multi-dimensional comparison with large dimensions, the interpretation of which is far from straightforward [32]. Therefore, in alignment with approaches in the clustering literature, here we report some of the most popular indicators for the comparison of 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 [33], which recommends the Silhouette measure for the evaluation of noisy datasets.

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

	Dataset 1		**Dataset 2**	
	Mt1 [31]	**RF+K-means**	**Mt2** [32]	**RF+K-means**
Connectivity	53.33	33.42	32.26	25.89
Dunn Index	0.01	0.08	0.06	0.15
Silhouette	0.67	0.98	0.89	0.99

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 dataset, considering any of the evaluation criteria, when comparing methodologies applied to equivalent datasets. 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 features.

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, with only a subset available for training and the remaining set used for testing. We use the set-up with all the data to obtain a ground truth cluster assignment, since all information is available and we cannot expect the algorithm (with less data to train) 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. Thus, accuracy in this context is measured as the agreement in the cluster assignment.

TABLE III
CLUSTERING AGREEMENT RESULTS.

	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)

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 methodologies;
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 [34]).

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-world monitoring data from the cloud domain and are not (manually) prepared (e.g. transformation or removal of 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 obtain clustering (through similarity learning) and can, dynamically, adapt to scenarios where the relation among features change.

C. Cloud Use Case

To demonstrate the applicability of the on-line RF+PAM methodology in the domain, 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 highest average dissimilarity.

The scheduling steps are detailed in Algorithm 1. The scheduler receives as parameter the new service and the list of nodes, 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 prioritize 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) #c = cluster
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 one).

We employed these concepts in a framework that coordinates the execution of services. In our use case, services are applications defined by a SLA (service description and quality of services' requirements) using the SLAC language [35], which are executed in a cloud. The framework has a central scheduler that receives service requests and schedules according to their SLAs. When it receives a new service, it communicates with the RF+PAM implementation to request

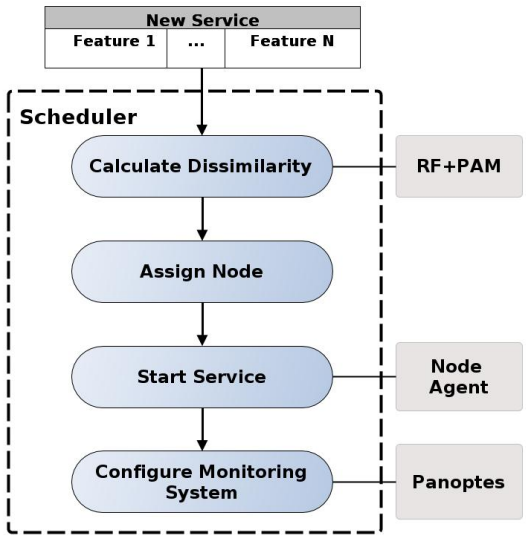

Fig. 2. Similarity Scheduling of the new services in the developed framework.

the clustering of the service. Then, the scheduler assigns the service to a node and sends it to an agent deployed in that node. Afterwards, it configures the Panoptes system [36] 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 2.

The experiments were conducted in a cloud using the OpenNebula[5] tool and 6 physical machines, providing 9 heterogeneous VMs in which the agents of a framework are employed to execute services requested by consumers.

To assess the performance of the dissimilarity scheduling, two other scheduling algorithms were used. In the first (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 other services. The second (named *Random*) assigns the services randomly to the nodes. Notably, all three algorithms have the resource admission control and 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 [29] at the beginning of every round of tests and the same services are executed using all three 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 [37].

[5]http://opennebula.org

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 the λ, 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 vary the value of λ in the experiments to analyse the performance of the algorithm with different loads. On every experiment, we generated 100 services and run all algorithms to schedule these services. This procedure was repeated 10 times for every λ.

Figure 3 (a) shows overall runtime of the services given the same input using all three algorithms and with different arrival rates. On the other hand, Figure 3 (b) shows the reduction of this total run time of the Dissimilarity scheduler in comparison to the Random. The Dissimilarity scheduler performs significantly better than the Random, in particular reducing in almost 30% the total run time for λ set to 0.8. The lower bound, i.e. the Isolated scheduler, is in average around 20% better than the Dissimilarity. However, the Isolated scheduler requires each service to be executed alone in the resource, which is impracticable in real-world deployments as it would lead to low resource usage (idle resources) and high service waiting time.

Furthermore, we tested the performance of the three algorithms by fixing λ as 1 and varying the number of input services (from 50 to 250). Figure 4 presents the results, which show that the improvement of the Dissimilarity is similar to the previous experiment even with higher number of services.

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 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.

V. Related Works

In this section, we discuss the relevant literature in the cloud domain that uses a notion of similarity to support decision systems with knowledge. In the *service scheduling* field, several works, e.g. [38], [39], [40], [41], 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 our use case, we propose a service scheduling algorithm, which uses the knowledge on similarities among services to avoid race conditions in the cloud resources. A similar approach was presented in [37]; the authors manually combine features and employ

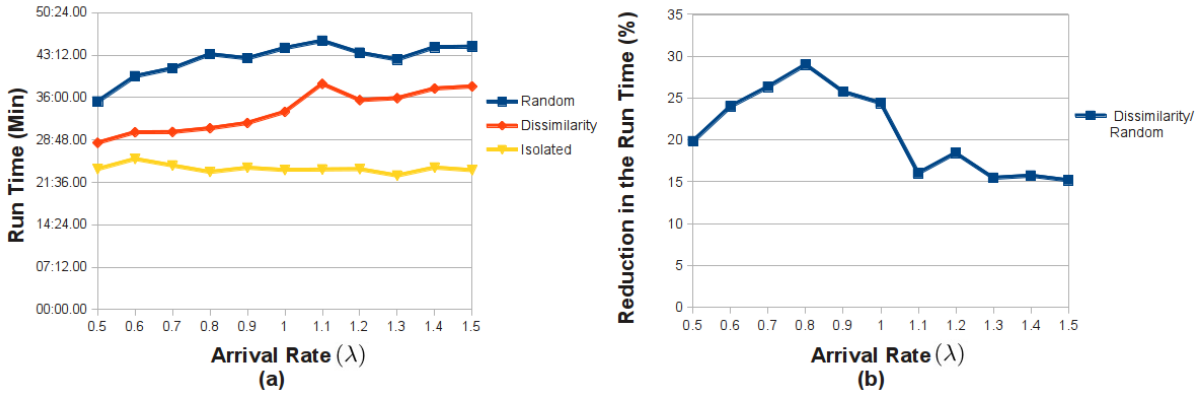

Fig. 3. Total run time of the scheduling algorithms with different arrival rates (a), and reduction (%) in the total run time with the Dissimilarity scheduler in comparison to the Random algorithm in this setting (b).

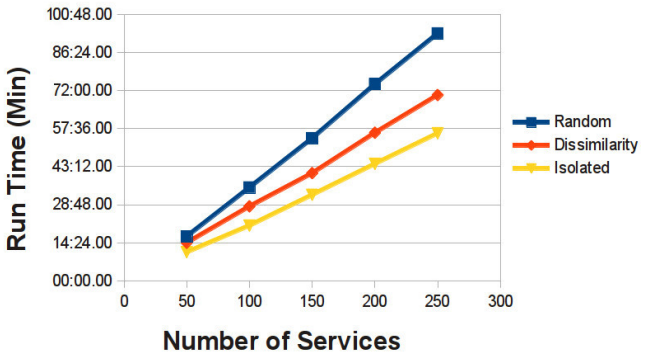

Fig. 4. Total run time of the scheduling algorithms with different numbers of input services.

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.

Regarding the *service profiling* field, most approaches are problem-specific, e.g. [42], [43] focus only on VMs. Hence, they cannot cover the diversity of the services and the heterogeneity of clouds. The solution of Kahn et al. [44] 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.

In the *anomalous behaviour detection* field, [45] uses a heuristic notion of similarity to cluster service requests and detect anomalous behaviours. Similarly, Wang et al. [46] propose a methodology to detect anomalies for Web applications in which the similarity among the workloads is used to detect problematic requests. However, both 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 the decision-making in autonomic clouds.

VI. Conclusions

The characteristics of autonomic clouds hinder their management and the decision-making process, as they obfuscate several details of the provided services and of the infrastructure. In this paper, we developed a methodology to feed autonomic cloud managers with knowledge on the similarities among services. 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 to mixed types of features, support to a large number of features and fast on-line prediction. Finally, to demonstrate its applicability in the domain, we implemented and tested a scheduling algorithm, which uses the notion of similarity to assign incoming services.

As future works, we will investigate the characteristics of RF, such as variable importance and feature selection, to improve our methodology. Moreover, we plan to apply the solution to the management of services, utilizing RF+PAM to dynamically calculate the SLA violation risks.

VII. Acknowledgements

This work has been partially sponsored by the EU project ASCENS (#257414), 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 reviewers for their encouragements and fruitful comments.

REFERENCES

[1] S. Tai, J. Nimis, and A. Lenk, "Cloud Service Engineering Categories and Subject Descriptors," in *Proc. of the 32nd ACM/IEEE ICSE*, 2010, pp. 475–476.

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

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

[4] M. Parashar and S. Hariri, "Autonomic computing: An overview," in *Unconventional Programming Paradigms*. Springer Berlin Heidelberg, 2005, pp. 247–259.

[5] I. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud Computing and Grid Computing 360-Degree Compared," in *Proc. of the 4th Grid Computing Environments Workshop*. IEEE, 2008, pp. 1–10.

[6] A. Cuzzocrea, G. Fortino, and O. Rana, "Managing Data and Processes in Cloud-Enabled Large-Scale Sensor Networks: State-of-the-Art and Future Research Directions," in *Proc. of the 13th IEEE/ACM CCGrid*. IEEE, May 2013, pp. 583–588.

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

[8] 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.

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

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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

[15] 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.

[16] 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

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

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

[19] 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.

[20] 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.

[21] 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.

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

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

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

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

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

[27] C. Xiong, D. Johnson, R. Xu, and J. J. Corso, "Random forests for metric learning with implicit pairwise position dependence," *Proc. of the 18th ACM SIGKDD*, p. 958, 2012.

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

[29] 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: http://googleclusterdata.googlecode.com/files/Googlecluster-usagetraces-format+schema(2011.10.27external).pdf

[30] 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.

[31] 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.

[32] 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

[33] 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.

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

[35] 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.

[36] 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.

[37] 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.

[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] 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.

[43] 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.

[44] 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.

[45] 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.

[46] 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.