Mapping of Quality of Service Requirements to Resource Demands for IaaS

Ioannis Bouras[1], Fotis Aisopos[1], John Violos[1], George Kousiouris[1], Alexandros Psychas[1], Theodora Varvarigou[1], Gerasimos Xydas[2], Dimitrios Charilas[2] and Yiannis Stavroulas[2]
[1]*Dept. of Electrical and Computer Engineering, NTUA, 9 Heroon Polytechniou Str, 15773 Athens, Greece*
[2]*Cognity S.A., 42 Kifissias Av., 15125 Marousi, Athens, Greece*

Keywords: Infrastructure as a Service, Cloud Computing, Quality of Service, Artificial Neural Networks, Resource Selection.

Abstract: Deciding and reserving appropriate resources in the Cloud, is a basic initial step for adopters when employing an Infrastructure as a Service to host their application. However, the size and number of Virtual Machines used, along with the expected application workload, will highly influence its operation, in terms of the observed Quality of Service. This paper proposes a machine learning approach, based on Artificial Neural Networks, for mapping Quality of Service required levels and (expected) application workload to concrete resource demands. The presented solution is evaluated through a comercial Customer Relationship Management application, generating a training set of realistic workload and Quality of Service measurements in order to illustrate the effectiveness of the proposed technique in a real-world scenario.

1 INTRODUCTION

Cloud adoption, either through a Software as a Service (SaaS), a Platform as a service (PaaS) or an Infrastructure as a Service (IaaS) approach, enables an application owner to take advantage of the Cloud computing benefits, such as stability and scalability. However, a Cloud provider can act in a twofold manner in this process, e.g. a SaaS provider can also act as an adopter, deploying his software on an external IaaS to further migrate instability risks. When it comes to choosing an IaaS provider for the deployment of any kind of application, a major consideration that Cloud adopters have to take, relates to the amount of the resources that should be reserved, in order to ensure the smooth operation of the application. This is far from a simple decision, as the resources selected and the expected workload during a period are decisive for many application-related Key Performance Indicators (KPIs), such as HTTP API calls processing times.

Moreover, in the case of a SaaS application deployed in an IaaS, keeping those KPIs in certain levels is most important, given that the SaaS provider should also keep its own Quality of Service (QoS), satisfying its customers SLAs. Despite the fact that, for this process a number of approaches exist (e.g. control based loops to regulate resources based on KPI levels), the overall definition of the various offered SLAs (in comparison also to the cost) needs an a priori analysis of the QoS, based on anticipated application workload and used resources. Thus, the need for a mapping mechanism arises, which would translate the applications high-level terms (Workload and QoS parameters), to respective infrastructure-level resource requirements.

The current work evaluates the concept of Mapping Models, i.e. models based on Artificial Neural Networks (ANNs) taking care of the aforementioned parameter translation. The design parameters of those models, are tuned employing a Genetic Algorithm (GA) in order to generate an optimum or a close to optimum ANN architecture, for the given dataset. The training of the ANN models is performed using detailed application and infrastructure data, such as average application traffic over a period, number of requests, memory (RAM) allocated for an image etc. This ensures the effectiveness of predictions, for various application and resource setups.

Therefore, this paper provides the following contributions:

- A black box ANN-based model that maps high-level IaaS adopter demands to low level parameters, by predicting Quality of Service metrics of a

Bouras, I., Aisopos, F., Violos, J., Kousiouris, G., Psychas, A., Varvarigou, T., Xydas, G., Charilas, D. and Stavroulas, Y.
Mapping of Quality of Service Requirements to Resource Demands for IaaS.
DOI: 10.5220/0007676902630270
In *Proceedings of the 9th International Conference on Cloud Computing and Services Science (CLOSER 2019)*, pages 263-270
ISBN: 978-989-758-365-0

deployed application of specific nature, based on specific resource and expected workload setups.

- The generation of a representative, experimental dataset, extracted from a real business Software as a Service application from the Customer Relationship Management (CRM) domain.
- Evaluation of the aforementioned Mapping Models accuracy and comparison with mainstream Machine Learning approaches in terms of effectiveness, discussing also the efficiency dimension.

The rest of this document is organized as follows: Section 2 presents recent related works on mapping QoS to resource demands for clouds. Section 3 presents the ANN-approach followed in the current work, explaining the Genetic Algorithm used, as well as the Mapping Models inputs and outputs. Section 4 discusses the experimental CRM use case examined and the extraction of a meaningful dataset for the Mapping Models, as well as the evaluation results. Lastly, Section 5 concludes the current work.

2 RELATED WORK

Considering Cloud provisioning as a mean to optimise an application performance and results, a basic preliminary step involves the selection of a service provider. However, once selecting the appropriate provider, the most crucial step of the cloudification process is resource selection (Psychas et al., 2018). Given some application runtime restrictions (e.g. maximum HTTP API calls processing times), certain Quality of Services requirements arise. The problem concerning the decision on the amount of resources to be reserved in the IaaS layer to satisfy or ensure those QoS requirements has been raised and thoroughly studied throughout the previous years (Geeta and Prakash, 2017).

2.1 QoS-aware Resource Management by IaaS

Starting from an IaaS provider perspective, work in (Papagianni et al., 2013) attempted to tackle the generic problem of providing an automated solution for dynamically managing physical resources of a data center used as substrate for IaaS platforms. The authors employed a Self-Tuning Regulation (STR) adaptation scheme, using an AutoRegressive with eXogenous terms (ARX) model to dynamically analyse resource and QoS monitoring values at run time. This model was evaluated with a set of three distinct multi-tier applications, assessing the performance of the resource management framework by measuring the number of violations with respect to the SLO constraints. Authors in (Guazzone et al., 2011) proposed a method for efficient mapping of user requests for virtual resources onto a shared substrate interconnecting previously isolated islands of computing resources, based on hard and soft QoS provisioning schemes. The approach promotes a unified management and control framework for delivering efficiently cloud IaaS, formulating the optimal networked cloud mapping problem as a mixed integer programming (MIP) problem with constrains related to cost efficiency of the resource mapping procedure, while abiding by user requests for QoS-aware virtual resources.

Collazo-Mojica (Collazo-Mojica et al., 2012) proposed a resource usage prediction engine based on a multivariate linear regression model, which provides possible resource allocations in IaaS based on heuristics. During runtime, the application performance is being monitored, in order to provide scaling suggestions. The approach is validated with a CPU-bound cloud application running on Amazon EC2 with a resulting average relative error of 17.49%. Works in (Kritikos et al., 2016) and (Anastasi et al., 2017) provide some interesting approaches on matching resources offered by an IaaS to respective non-functional parameters (like execution duration) for the application. Lastly, (Ran et al., 2017) presents an online overload probability estimation model aiming at minimizing the total computing cost in IaaS, while keeping the QoS levels agreed in customer SLAs without any priori knowledge of the workload.

2.2 Predicting Resource Needs for SaaS

The issue of resource reservation from the perspective of SaaS providers, has also been extensively studied in numerous works which propose algorithms that minimize infrastructure cost and SLA violations. Authors in (Reig et al., 2010) used fast analytical and adaptive predictors for resource needs in order to prevent SLA violations concerning execution time. Authors mainly target SaaS providers, handling heterogeneous workloads and attempt to predict the amount of CPU and memory needed, based on reference execution times and using simple machine learning models, like Linear Regression and REPTree, aiming at low computational complexity. Work in (Wu, 2011) examines an EPR/CRM SaaS example, including parameters like Number of Records and Response time in their respective SLAs. Authors aim at interpreting customer requirements to infrastructure layer parameters, focusing on the VM level, similarly to the current case study. They propose two SLA-based profit maximization algorithms, based on a plain Mapping Strategy of customer QoS requirements to resources.

Moreover, (Sun et al., 2017) introduced ROAR (Resource Optimization, Allocation and Recommendation System), a modeling framework to simplify, optimize, and automate SaaS resource allocation decisions to meet QoS goals for web applications. A textual domain-specific language (DSL) called the Generic Resource Optimization for Web applications Language (GROWL) is defined to specify the high-level and customizable load testing plan and QoS requirements without low-level configuration details. ROAR derives the appropriate cloud resource configurations to orchestrate tests against each resource allocation option, before using the results to recommend a cost-optimized resource allocation to meet the QoS goals.

2.3 Mapping Models Approach

The Mapping Models tool, developed in the context of the CloudPerfect Innovation action project (CloudPerfect:, 2017), uses a sophisticated machine learning approach, employing genetically optimized Artificial Neural Networks. This approach was selected, as the relation between the parameters (inputs and outputs) of the problem are rather complex, presenting non-linear behavior. Its goal is to translate high level QoS requirements of a SaaS provider to low level resource demands for the IaaS layer. The current work is based on (Kousiouris et al., 2011) and (Kousiouris et al., 2014), aiming to provide an innovative solution, tested using a CRM SaaS application deployed over a commercial IaaS testbed.

3 MAPPING QUALITY OF SERVICE TO RESOURCES

Mapping Models provide a generic supervised machine learning approach for correlating the defined requirements (QoS) of a SaaS application for various workloads, to resource level attributes (as these are needed to be expressed in the SLA with the IaaS provider). Specifically, the number of resources needed by a specific application instance can be estimated given the respective application level characteristics, such as the defined QoS restrictions and the applied workload.

Mapping Models are based on Artificial Neural Networks in GNU Octave [1] and built on a per software component basis with application-specific terms that vary depending on each use case and in different hardware configurations available per Cloud Provider. As a first step, the SaaS application administrator (IaaS adopter) needs to define the application parameters (workload, resources and KPIs) to be used as model inputs and outputs respectively. This process is necessary in order to adapt in each examined case. ANNs are non-linear approximators that can be used, in a black-box manner, to capture the dependency of the output from the applied input, as shown in Figure 1.

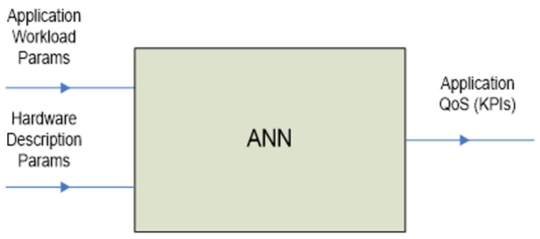

Figure 1: ANN black box approach inputs and outputs.

The implementation utilizes the GNU Octave 'newff' function for ANN building, which is based on feed forward back propagation networks trained with the Levenberg-Marquardt algorithm. The core of the Models Creation mechanism currently consists of an ANN factory, based on Octave tool. For the training phase, the ANN factory needs a representative dataset of executions provided by the IaaS adopter in an Octave file format, which will be used from a macroscopic point of view. This training dataset will be used to determine the network weights and identify complex, linear or non-linear dependencies of the output from the inputs.

The design parameters of the ANN are configured by a Genetic Algorithm (GA), in order to optimize their selection. In general, selecting these parameters is a tedious process, based on human experience. However, it can not be easily repeated and applied in an automated environment, thus in the current approach the GA solution was followed. The design parameters that are included in the optimization process include the number of hidden layers, the number of neurons per hidden layer and the type of transfer functions of each network layer, from the available three types in Octave (tansig, logsig and purelin)[2]. As illustrated in Fig. 2, the neuro-evolution process starts with an intial population of random architectures of ANNs. On each generation the GA selects the fittest

[1]https://www.gnu.org/software/octave/

[2]http://radio.feld.cvut.cz/matlab/toolbox/nnet/backpr52 .html/

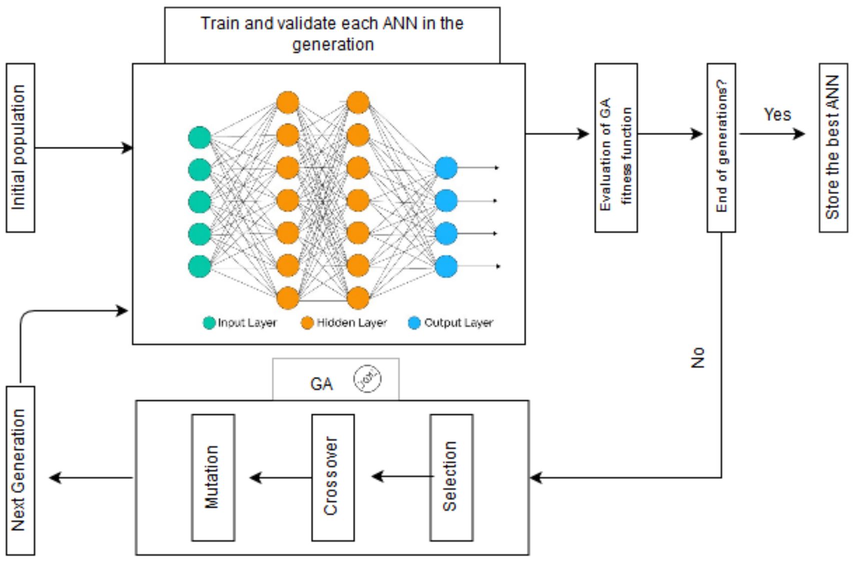

Figure 2: The neuro-evolution process employed by the Mapping Models.

ANNs to be crossovered based on a probability system regarding their performance (Selection process). From this procedure, new architectures of ANNs will be created and after the necessary mutation process (in order to maintain some amount of randomness in the GA) a new generation of ANNs is produced, ready to be trained and validated. This process continues until the predifined number of generations is reached. Furthermore, proper data preprocessing ensures the inclusion of the minimum and maximum values in the training subset in order to avoid extrapolation risks.

The evolutionary performance criterion for the GA i.e the Fitness Function, is the error on the training set. However, variations with the performance criterion as the error on the intermediate validation set, had been tried out in the original version of the algorithm (Kousiouris et al., 2013), which seemed to yield towards worse results. In each generation, the generated models are evaluated and selected, based on their prediction performance on the intermediate validation set, while the final selection is performed according to the prediction performance on the third and final subset (test set) of the initial data (approximately 30% of them), which is not used during the training process.

The overall process of model creation consists of the following steps:

- Create KPI monitoring endpoints in application structure
- Define different types of used resources
- Gather dataset with rows including values for workload, resource type and KPI values respectively
- Feed the file in the tool UI
- Launch the training and validation phase for the model
- Check prediction outcomes and save candidate model
- Use model for prediction of new cases/combinations

The Mapping Models tool consists of 3 main scripts, the model-creator, the model-selection and the ann-script, along with a number of helper functions for e.g. (de)normalization. Inside the tool, the process is as follows:

- The model-creator script is responsible for the UI part, receiving input files and configuration paths from the user (e.g. path to model repository, model ID, number of input columns of the provided csv file etc). File manipulation as well as identification is based on the model ID, which needs to be unique. Furthermore, the IaaS adopter is asked whether the KPI is ascending or descending and whether they prefer over or under-provisioning, a feature that will be taken under consideration in the selection of the final model.
- Upon finalization of the setup, the model-selection script is launched, which is responsible for retrieving the data, normalizing them, splitting them into three subsets (65% training, 15% intermediate validation and 20% final test set) and setting up the GA part of the code before calling the ann-script inside the GA function. Furthermore, it is responsible for identifying, at the end of the process, the best model by applying the test set and concluding on the error.

- The ann-script is the core function that receives a proposed ANN architecture by the GA (a chromosome) and is responsible for creating the respective network based on the parameters, as well as training and measuring its error on the intermediate validation set.

Upon finalization of the process, the UI part picks up the control and presents the outcomes (error of all the candidate models and detailed error of the best model in all validation cases), asking the user whether the outcome was satisfactory, in which case it stores the model for future use. An example is illustrated in Figure 3.

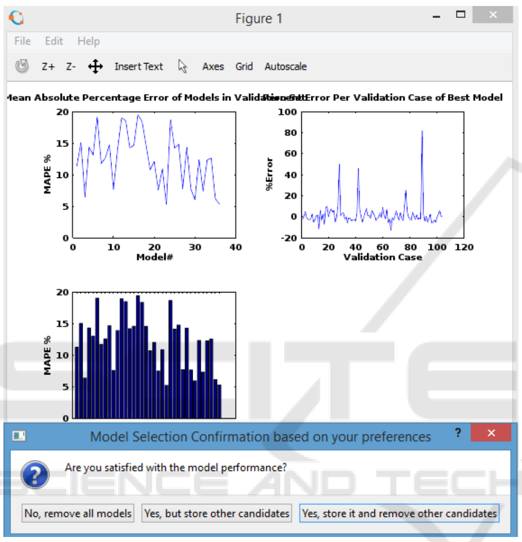
Figure 1
File Edit Help
Z+ Z- Insert Text Axes Grid Autoscale
Mean Absolute Percentage Error of Models in Validation Set
Error Per Validation Case of Best Model
MAPE %
Model#
%Error
Validation Case
Model Selection Confirmation based on your preferences
Are you satisfied with the model performance?
No, remove all models
Yes, but store other candidates
Yes, store it and remove other candidates

Figure 3: Mapping model performance.

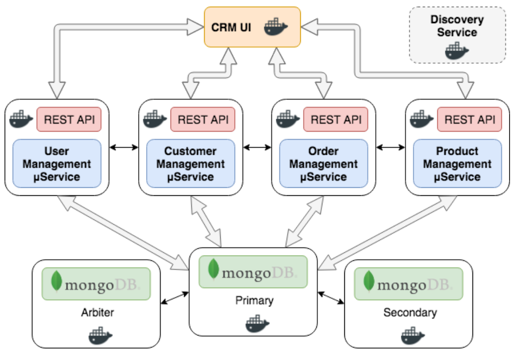

Figure 4: Vertical structure of the CRM application.

4 PREDICTING QoS FOR A CRM SaaS APPLICATION

For the purpose of validating the described mapping approach, we employ a use case scenario involving a multi-tenant SaaS application from the Customer Relational Management (CRM) domain, deployed on the Cloud at the IaaS level. This application, initially deployed on a commercial on-premises infrastructure, is moved to the IaaS layer in a horizontal manner. At a glance, agents of the CRM application (users) log in and perform customers, products and orders management.

4.1 CRM Application Architecture

The CRM application follows the modern Microservices architecture, according to which the application logic is broken into several independent services, each of which is charged with performing a specific task. This allows for the application to be easily scalable and become distributed in multiple hosting environments.

Figure 4 illustrates the application components, along with their interconnections in a vertical structure. For the purposes of the current experimentation we focus on the 4 Microservices that provide the core functionality of the CRM application: Users, Products, Customers and Orders. Due to its nature, the CRM application can be scaled up (modifying CPU cores and memory) or out (adding more instances). In the frame of this work, we experimented with various test cases in order to assess a set of scalability scenarios in terms of cost and performance. These scenarios mainly considered scaling out, an example of which is depicted in Figure 5.

In such cases, additional instances of degraded, in terms of performance, components are deployed in order to maintain the applications overall availability.

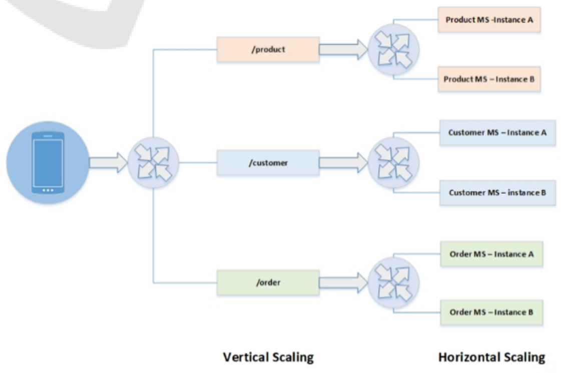

Figure 5: Scalability use of the CRM application.

A major performance indicator for any application that users interact with in real-time, is the processing time of user actions, i.e. the time that the system does not allow further interactions until the previous action is processed. In our case, the user does not interact directly with the individual Microservices, but with the UI one, which in turn spawns requests to the core Microservices. Thus, we defined as a KPI the average processing time of each individual core Microservice: Users, Products, Customers and Orders one.

Different workloads and host hardware configuration mainly affect the processing time of each Microservice. To be able to predict processing time based on given traffic and hardware we utilized the Mapping Models. For the following experiments, we deployed all Microservices as docker containers in a single dedicated host server.

4.2 Dataset Creation

Building a dataset for the Mapping Models required the generation of enough paradigms of real user scenarios. For that purpose, we composed a simulation of a common users scenario in the CRM (we call this scenario lifecycle):

1. User logs in
2. User serves M customers
3. Each customer performs N orders
4. Each order is saved/updated K times
5. User logs out

We built a testbed application (the orchestrator) based on a smooth combination of bash scripts, GO and NodeJS codebase to load the CRM with a variety of lifecycle traffic and retrieve statistical measurements (memory usage, request execution time etc.) from each individual CRM Microservice.

Table 1: Lifecycle script input parameters.

Parameter	Values
experiment duration (sec)	30, 60, 120
#virtual users distributed in the above duration	1, 10, 30, 50, 60, 90, 100, 120, 150, 200, 300
#customers (M)	2, 4
#orders per customer (N)	2, 4
#updates per order (K)	7

Table 2: Containers RAM (MB) per Service component per batch execution.

	Batch			
Service	1	2	3	4
Orders	700	896	1384	1792
Customers	700	896	1384	1792
Products	1024	1596	2048	3096
Users	700	896	1384	1792

The script was executed by creating different number of concurrent users each time. Table 1. shows the parameter variations in the script configuration. All combinations of these values (132) formulated a batch that was executed under a different hardware setup each time. To achieve hardware variation in the experiments, we modified the overall memory allocated to each Docker container, creating four different configurations for each batch Table 2.

Thus, the Mapping Models dataset was generated by running the lifecycle batches (132) under the 4 different hardware profiles, providing 396 paradigms for each Service component. Table 3 shows some indicative values for the various Microservices, where:

- **RAM:** the overall memory allocated for the respective docker container
- **#Users:** total number of virtual users created during the scenario run
- **#Request:** the number of requests served by the component
- **#RPS:** requests per second for the component
- **Processing Avg:** average Microservice processing time

During the execution phase, we used the simplest possible configuration, i.e. a single instance of each application Microservice. The Microservice performance is monitored throughout the scenario execution time, and an average processing time is provided as the final Quality of Service meter. Each thread simulates the lifecycle scenario described in Section B of the current document as UI actions, which are translated in a number of requests to the supporting service components.

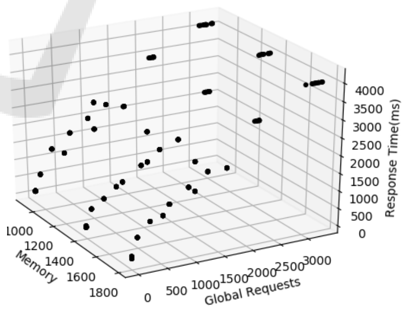

Figure 6: 3D view of the Customers service parameters.

As observed in the initial dataset results, most requests during the lifecycle are generated in Orders Microservice, while the RAM increment has a positive effect, only in Products and Users ones. Fig. 6 illustrates the main parameters of a Microservice dataset in the 3D space.

Table 3: Mapping Models Dataset generated by the orchestrator.

Microservices	RAM (MB)	#Users	#Requests	RPS	Processing Avg (msec)
Orders	700	10	70	3.04	23.59
Customers	700	10	70	3.04	23.59
Products	1024	10	20	0.87	104.70
Users	700	10	20	0.87	55.30
Orders	1792	10	70	2.92	23.03
Customers	1792	10	10	0.42	66.3
Products	3096	10	20	0.83	80.10
Users	1792	10	20	0.83	37.45

4.3 Evaluation of Experimental Results

To validate the accuracy of the Mapping Models approach, we compared them with three mainstream and effective machine learning techniques, namely Support Vector Machines (SVMs), Random Forests (RFs) and Linear Regression (LR). In order to achieve the desired level of generalization and prediction perfomance, the aforementioned models were tuned using exhaustive Grid Search on a wide range of hyperparameter values, combined with k-fold cross-validation as the model validation technique. In specific, for the SVMs we used the RBF kernel, tuning the C and gamma parameters. In Random Forests we tuned the number of estimators, the maximum depth of the tree and the minimum number of samples required to be at a leaf node. Linear Regression, being a simple model, performed poorly on the given dataset and its results will not be presented. The code for these experiments was written in python using the scikit-learn library (Pedregosa et al., 2011).

Table 4: Percentage Errors Analytical Comparison.

Service	ANN	Random Forest	SVM
Orders	9.6	18.17	20.83
Customers	6.22	10.51	9.33
Products	13.39	13.52	12.82
Users	8.37	9.38	9.38

The design parameters of each ANN model which were configured, after 30 generations, by the Genetic Algorithm (GA) with 3 candidate transfer functions and a maximum of 10 layers and 30 neurons per layer are illustrated in Table 5.

In order to keep the models simple, a different model was built and trained for each different Microservice of the CRM application (given that they are independent) resulting to better performance and to a comprehensive insight of the behavior of each separate Service Component. The validation metric for these experiments was the Mean Absolute Percentage Error (MAPE) of each model for the same test set.

An overview of the resulting MAPEs is illustrated in Fig. 7, with the numbers for each Service separately provided in Table 4. As it can be observed in the provided results, the ANN, optimized by the Genetic Algorithm, clearly outperforms the competition for all services, being however less efficient during the training phase, due to the optimization step introduced by the use of the GA.

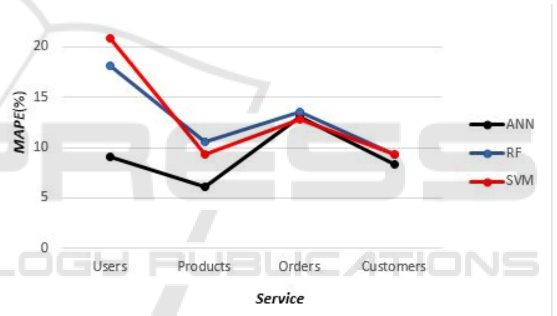

Figure 7: Mapping Models, SVM and Random Forest Performance Comparison.

5 CONCLUSIONS

This work presented a machine learning approach, employing Artificial Neural Networks and a Genetic Algorithm, in order to analyze resource, workload and Quality of Service parameters trade-offs for a SaaS application adopting resources from the IaaS layer. This was evaluated using a real business CRM application dataset, and compared to several existing mainstream approaches. Evaluation results illustrated the effectiveness of the ANN solution, in contrast with other techniques. The application of the GA makes the training of the current solution less efficient, adding an extra step, however during runtime no significant time deviation can be observed from other approaches.

In terms of future work, the authors plan to include more hardware parameters (e.g. CPU cores)

Table 5: Neural Network parameters selected by GA.

Optimized ANN details Format			
Service	**Layer Number**	**Neuron per layer**	**Transfer functions per Layer**
Orders	4	5-3-2-1	tansig-logsig-logsig-purelin
Customers	3	5-3-1	tansig-logsig-purelin
Products	3	5-3-1	tansig-logsig-purelin
Users	3	5-3-1	tansig-tansig-purelin

and build an extended dataset for the aforementioned application. This will provide a more thorough view of resources and QoS relation and will result into a more concrete analysis, taking into account also an efficiency comparison of various approaches.

ACKNOWLEDGEMENTS

This work has been supported by the CloudPerfect project and funded from the European Unions Horizon 2020 research and innovation programme, topic ICT-06-2016: Cloud Computing, under grant agreement No 73225.

REFERENCES

Anastasi, G., Carlini, E., Coppola, M., and Dazzi, P. (2017). Qos-aware genetic cloud brokering. In *Future Generation Computer Systems 75, 1 - 13.*

CloudPerfect: (2017). Enabling cloud orchestration, performance and cost efficiency tools for qoe enhancement and provider ranking. In *http://cloudperfect.eu/.*

Collazo-Mojica, X., Sadjadi, S., Ejarque, J., and Badia, R. (2012). Cloud application resource mapping and scaling based on monitoring of qos constraints. In *SEKE.*

Geeta and Prakash, S. (2017). A review on quality of service in cloud computing. In *Big Data Analytics, Advances in Intelligent Systems and Computing.* Springer Singapore, pp. 739 748.

Guazzone, M., Anglano, C., and Canonico, M. (2011). Energy-efficient resource management for cloud computing infrastructures. In *IEEE Third International Conference on Cloud Computing Technology and Science, pp. 424 431.*

Kousiouris, G., Kyriazis, D., Gogouvitis, S., Menychtas, A., Konstanteli, K., and Varvarigou (2011). Translation of application-level terms to resource-level attributes across the cloud stack layers. In *IEEE Symposium on Computers and Communications (ISCC), pp. 153 160.*

Kousiouris, G., Menychtas, A., Kyriazis, D., Gogouvitis, S., and Varvarigou, T. (2014). Dynamic, behavioral-based estimation of resource provisioning based on high-level application terms in cloud platforms. In *Future Generation Computer Systems 32, 27 40.*

Kousiouris, G., Menychtas, A., Kyriazis, D., Konstanteli, K., Gogouvitis, S., Katsaros, G., and Varvarigou, T. (2013). Parametric design and performance analysis of a decoupled service-oriented prediction framework based on embedded numerical software. In *IEEE Transactions on Services Computing 6, 511 524.*

Kritikos, K., Magoutis, K., and Plexousakis, D. (2016). Towards knowledge-based assisted iaas selection. In *IEEE International Conference on Cloud Computing Technology and Science (CloudCom), pp. 431 439.*

Papagianni, C., Leivadeas, A., Papavassiliou, S., M. V., Cervell-Pastor, C., and Monje, A. (2013). On the optimal allocation of virtual resources in cloud computing networks. In *EEE Transactions on Computers 62, 1060 1071.*

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in python. In *Journal of Machine Learning Research 12, 2825 2830.*

Psychas, A., Violos, J., Aisopos, F., Evangelinou, A., Kousiouris, G.and Bouras, I. V. T., Xidas, G., Charilas, D., and Stavroulas, Y. (2018). Cloud toolkit for provider assessment, optimized application cloudification and deployment on iaas. In *Future Generation Computer Systems.*

Ran, Y., Yang, J., and Zhang, S., X. H. (2017). ynamic iaas computing resource provisioning strategy with qos constraint. In *IEEE Transactions on Services Computing 10, 190 202.*

Reig, G., Alonso, J., and Guitart, J. (2010). Prediction of job resource requirements for deadline schedulers to manage high-level slas on the cloud. In *Ninth IEEE International Symposium on Network Computing and Applications, pp. 162 167.*

Sun, Y., White, J., Eade, S., and Schmidt, D. (2017). Roar: A qos-oriented modeling framework for automated cloud resource allocation and optimization. In *Journal of Systems and Software 116, 146 161.*

Wu, L., G. S. B. R. (2011). Sla-based resource allocation for software as a service provider (saas) in cloud computing environments. In *Proceedings of the 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGRID 11. IEEE Computer Society, Washington, DC, USA, pp. 195 204.*