INTELLIGENT PROACTIVE FAULT TOLERANCE AT THE EDGE THROUGH RESOURCE USAGE PREDICTION

Theodoros Theodoropoulos[1], John Violos[2], Stylianos Tsanakas[2], Aris Leivadeas[3], Konstantinos Tserpes[1], Theodora Varvarigou[2]

[1]Department of Informatics and Telematics, Harokopio University of Athens, Tavros, Greece, [2]School of Electrical and Computer Engineering, National Technical University of Athens, Zografou, Greece, [3]Department of Software and IT Engineering, École de technologie supérieure, Montreal, Canada

NOTE: Corresponding author: John Violos, violos@mail.ntua.gr

Abstract – *The proliferation of demanding applications and edge computing establishes the need for efficient management of the underlying computing infrastructures, urging the providers to rethink their operational methods. In this paper, we propose an Intelligent Proactive Fault Tolerance (IPFT) method that leverages the edge resource usage predictions through Recurrent Neural Networks (RNNs). More specifically, we focus on the process faults, which are related with the inability of the infrastructure to provide Quality of Service (QoS) in acceptable ranges due to the lack of processing power. In order to tackle this challenge we propose a composite deep learning architecture that predicts the resource usage metrics of the edge nodes and triggers proactive node replications and task migration. Taking also into consideration that the edge computing infrastructure is also highly dynamic and heterogeneous, we propose an innovative Hybrid Bayesian Evolution Strategy (HBES) algorithm for automated adaptation of the resource usage models. The proposed resource usage prediction mechanism has been experimentally evaluated and compared with other state of the art methods with significant improvements in terms of Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE). Additionally, the IPFT mechanism that leverages the resource usage predictions has been evaluated in an extensive simulation in CloudSim Plus and the results show significant improvement compared to the reactive fault tolerance method in terms of reliability and maintainability.*

Keywords – *Edge computing, evolution strategy, hyper-tuning, fault tolerance, recurrent neural networks*

1. INTRODUCTION

During the last decade, the scientific community witnessed the emergence of applications that are intertwined with a set of demanding QoS requirements. Extended Reality (XR) applications is one instance of this type of application. XR applications are associated with various QoS requirements [1] that are based on the ability to provide an immersive end-user experience. These requirements may include aspects such as latency and bandwidth. Studies have shown that for an end-user experience to be acceptable in terms of immersion, the end- to-end latency shall not surpass the 15ms mark, and the bandwidth should be able to reach up to 30 Gbps [2]. On top of that, the desired integrity of the aforementioned immersive experiences may be jeopardized by faults in task processing, due to potential disruptions in service delivery. Therefore, it is of paramount importance for this class of applications to be able to exhibit fault tolerance capabilities. Furthermore, this class of applications presents various demanding requirements in terms of computational resources, since they incorporate the rendering of 3D models and detailed graphics. Because of these computational requirements, monolithic development architectures would result in prohibitively bulky and expensive end-user equipment in order to facilitate the required computational resources.

Cloud computing is able to partially alleviate the burden that is imposed on the end-user devices by providing computational resources that these applications may run on via the Internet. Cloud computing is based on the use of shared computational resources that may span multiple locations. Therefore, part of the computational burden is transferred to these shared resources. Unfortunately, the distance between the end-user devices and the cloud servers may result in high latency and low available bandwidth. Thus, the need to bring processing and data closer to the devices where it's being generated [3] was created. In the case of XR applications, these devices may include smart objects, mobile phones, network gateways, sensors and a plethora of immersion devices. This distributed computing paradigm, defined as edge computing aims to establish decentralized topologies and allow the relocation of various computational and storage resources closer to the edge of the network. By doing so, it is expected to provide service delivery and content caching in better response times and transfer rates. The aforementioned devices may vary wildly in terms of computational prowess. As a result, it is necessary to make sure that the computations that take place at the edge are not demanding and do not exceed the computational capabilities of the involved devices.

When contemplating the nature and requirements of modern-day applications it is of major importance for

the workload execution to be resilient and meet the QoS standards set by the industry. The devices at the edge of the network are subjected to significant fluctuations in the amount of offloaded tasks over time [4]. Hence, it is of paramount importance that these fluctuations do not affect the performance of the system and cause process faults [5]. In addition to that, edge computing environments are characterized by extreme heterogeneity and dynamicity in regards to the tasks and the processing nodes involved. This unprecedented situation gave birth to the need for an IPFT method which should be robust to infrastructure and workload changes.

Monitoring and predicting the capacity under which the edge nodes are operating in terms of resource metrics such as CPU, RAM, bandwidth and disk can be a valuable piece of information with regards to implementing fault tolerance policies. Resource metrics have high serial and cross-correlation values making the use of time series methods rational [6]. Regression-based RNN [7], which leverage time series characteristics through Gated Recurrent Units (GRUs) [8] or Long Short-Term Memory (LSTM) [9], can be used in order to accurately predict the resource metrics.

In order to handle the extreme heterogeneity and dynamicity of the edge environments, we provide a systematic methodology for building deep learning models in an automatic way using historical data. Common approaches that are based on manual trial and error methodologies in regards to creating acceptable deep learning architectures require many working hours to be spent by deep learning experts every time the deployed applications, user behaviour or the edge infrastructure change. On the other hand, the available deep learning automation methods still have significant shortcomings such as low effi- ciency and high computational requirements [10]. A potential solution to these drawbacks could be the extension of evolutionary algorithms, as well as their combination with other models for hyper-tuning.

The facts mentioned above motivated us to propose an IPFT method that focuses on processing faults; faults related with resource shortage and the resulting incompetence in regards to processing capabilities that impede the underlying infrastructure to execute tasks within acceptable QoS ranges. Our research goals are to propose a composite deep learning architecture suitable for predicting in a unified way the ability of the edge nodes to execute the incoming workload and an appropriate operational pipeline that guarantees advanced fault tolerance. The cornerstone of this pipeline is the ability to operate in a proactive manner. Whenever a bottleneck in task execution is expected to occur, proactive measures like task migration and node replication should be triggered. A composite deep learning architecture should leverage the time series characteristics of the edge resources and the involved tasks which can be provided by monitoring systems (i.e. Prometheus) [11]. Finally, we propose the HBES optimization algorithm in order to provide a composite

deep learning model which is nearly optimal. The four major contributions of our research are:

- The proposal of the IPFT method that achieves high reliability and maintainability with very good performance in terms of timely fault detection and repair.

- A discussion of how a specific category of faults, the process faults are related and can be predicted by the resource utilization metrics of processing edge nodes

- The proposal and analysis of the theoretical principles of a composite deep learning model for edge resource usage prediction that includes two channels. One with feedforward and one based on RNN layers.

- The proposal of an innovative hybrid hyperparameter optimization model that combines the evolution strategy with the Bayesian optimization algorithms in order to gauge a close to optimal composite deep learning architecture.

The rest of the paper is structured as follows: Section 2 highlights the related work in fault tolerance, resource usage prediction, time series, deep learning and hyperparameter optimisation techniques. Section 3 explains how a proactive fault tolerance mechanism can leverage resource usage predictions. Section 4 provides an analysis of the RNN multi-output regression approaches, the composite deep learning architectures and the HBES method. Section 5 describes the experimental setup in a real edge computing dataset, the simulation of IPFT in CloudSim Plus and the evaluation results. Section 6 concludes the paper, reports the current limitations and suggests future directions.

2. RELATED WORK

Fault tolerance mechanisms are mainly divided in two categories; reactive and proactive. The reactive approach decreases the influence of failures in the edge infrastructure after a failure has actually occurred. The main reactive fault tolerance methodologies are reactive replication, resubmission, retry and use of checkpoints. For instance, a state of the art replication-based fault tolerance mechanism in large-scale graph-parallel systems was proposed in [12], which works by supporting cheap maintenance of the vertex states. This mechanism replicates the vertices with normal message exchanges, and provides fast in-memory reconstruction of the failed vertices from replicas in other machines. The replication increases the reliability of the system and the chance that the task will finish correctly, at the expense of additional resources for redundancy.

A retry approach that uses idempotent HTTP methods has been proposed in [13] for offloading and execution failures. This retry strategy has the advantage that it utilizes the least resources of the computing environment and minimizes the user time, but at the expense of increasing the response time, since HTTP methods may be retried multiple times until they complete successfully. In terms of checkpoints, a reactive fault tolerance approach for the

serverless paradigm was investigated in [14]. Specifically, the authors, through checkpoints and live container migration, have succeeded in saving resources in constrained devices. Unfortunately, also in this case, the execution time is increased since it includes the recovery time of the failed servers.

In the proactive fault tolerance approach a potential fault is predicted in order to avoid its influence on the task execution. Tian et al. [15] use the tree-based model which is a statistical analysis technique to diagnose the high risk cloud tasks and apply virtual machine migration techniques. This approach even if it significantly improves the reliability and efficiency, has generalization limitations, since it cannot automatically adapt to new computing environments. Machine learning and online learning methods have also been used in combination with microservices architecture and IoT systems to detect fault patterns and pre-emptively mitigate the faults [16]. Another ad-vanced proactive model has recently been proposed in [17]. This model performs multi-step predictions in or-der to estimate the process faults and the QoS degrada-tion in different time granularities. This approach utilizes an encoder-decoder model and gauges the ability of the infrastructure to process the incoming tasks at different production rates.

From the above it is evident that the limited computational capacity of processing nodes sets barriers to the edge computing and IoT applications [18]. We can overcome these barriers through efficient resource management [19]. This process includes the guarantee of well-defined QoS metrics and an accurate workload prediction [20]. Some of the workload prediction models leverage RNN and specifically LSTM, formulating the resource usage metrics as data sequences. But none of them focus on how a proactive fault tolerance method can leverage the resource usage predictions.

For instance, the authors in[6] have used an Autoregressive Integrated Moving Average (ARIMA) to avoid resource under-provisioning or over-provisioning in data centers. ARIMA has the limitation that it models linear dependencies and it is based on the stationarity assumption. However, as noticed in [21] the workload to be processed at the edge has trend, seasonality and nonlinearities in the execution behaviours, which limit the application of statistical linear models. These limitations are overcome by machine learning models such as K-Means, decision tree and K-nearest neighbors [22]. While there is a lot of classical machine learning models publicly available, for our experiments we selected XGBoost [23] because it is popular for winning Kaggle and other prestigious machine learning competitions [24]. XGBoost mostly uses gradient boosted decision trees and is available as an open-source software library.

The limitation of the machine learning approaches is that every time the edge-cloud infrastructure, the user behaviour, or the application changes, new models should be trained from scratch with human assistance. Automated machine learning achieves an automatic way to guide the learning process of models, maximizing the performance, and minimizing the computational budget without human involvement. In the domain of cloud computing, the Application and User Context Resource Predictor (AUCROP) [25] has been proposed for automated usage of classical machine learning algorithms. In addition, a general-purpose automated machine learning meta-model for data preprocessing, regression and hyper-parameter tuning through the Bayesian optimization is the auto-sklearn [26].

Keras-Tuner [27] is the approach from Keras to automate the hyper-parameter tuning, also named hyper-tuning. Keras is one of the most popular frameworks in the deep learning community. Keras-Tuner has the advantage that the hyper-models, the range of hyper-parameters and the tuning process is smoothly integrated in Keras but it supports only the optimizers: (a) random search, (b) hyper-band which is a random search with early stopping, and the (c) Bayesian optimization. In our experimental evaluation we used Keras-Tuner, AUCROP and auto-sklearn. In this work we also extend the research in the hyper-tuning combining evolution strategy with the Bayesian optimization. Thus, we propose the innovative HBES method as a prominent automated deep learning solution that tackles the heterogeneity and the dynamicity of edge computing environments.

More specifically, our work aims to extend the resource usage prediction method by proposing a proactive fault tolerance method that leverages the resource usage predictions and tackles the above-mentioned limitations as follows: Firstly, the IPFT mechanism requires a minimum number of replicas of the execution nodes since it requests a replication only after a fault prediction. Secondly, the IFPT does not include the time overhead of task rescheduling after a fault since the replication and rescheduling of the task will take place proactively and in a timely manner. Thirdly, the IPFT leverages deep learning RNN models in order to overcome the limitations of statistical models and adapts to non-linear and non-stationary resource metrics. Lastly, the introduced HBES qualifies the generality of the whole process which is a common limitation of many methods in the pertinent lit-erature.

3. LEVERAGING RESOURCE USAGE PREDICTIONS FOR PROACTIVE FAULT TOLERANCE

3.1 Resource usage prediction in edge computing

The management and orchestration of edge computing infrastructures can be improved by leveraging various resource utilization metrics. The most notable of these metrics are CPU, RAM, bandwidth, and disk I/O. At the same time, the edge computing paradigm is characterized by the dynamic behaviour and the heterogeneity of the processing edge nodes, which are obliged to operate within

some specific constraints dictated by the QoS requirements.

As the decision-making in a dynamic and heterogeneous environment is a rather complex process, it requires every available source of information to be used. Prediction of the resource consumption metrics, by leveraging time series characteristics of historical data, constitutes one of the most valuable pieces of information. It serves as a strong indicator for the availability of the processing nodes in order to receive additional workload or to predict potential QoS degradation in future time steps. Accordingly, the publicly-available monitoring tools like Prometheus, OpenTSDB, Nagios and InfuxDB can provide the resource metrics in a stream format or in a time series database like PromQL. These time series databases can be used to produce datasets which are suitable for the RNN model training.

The dynamic behaviour of edge nodes is attributed to the fluctuation of application requests and their workload. The number of requests per time interval changes within various time-frames and is affected by many periodic phenomena. Furthermore, the edge is characterized by a high heterogeneity, since the edge nodes can have different hardware and software characteristics, such as memory, computing power, etc. This heterogeneity becomes more apparent when taking into consideration the var- ious flavours of Raspberry Pis, Arduinos, sensor motes, and other microcontrollers that coexist and collaborate within the same edge infrastructure.

At the same time, application owners can set strict performance requirements for the edge nodes in terms of availability, throughput and different types of potential delays. Thus, edge providers are struggling to get the QoS metrics within the acceptable ranges specified. Consequently, in order to guarantee that the infrastructure will have sufficient computational capacities to handle fluctuating demands, a fault tolerance mechanism should proactively take decisions considering the amount of resources and the availability of the processing nodes.

In the scientific literature there are different categories of faults that correspond to specific fault tolerance mechanisms. The major categories of the faults are: (a) network faults, (b) physical faults, (c) process faults and (d) service expiry faults [28]. Among these faults, process faults can be very severe. In more detail, process faults occur in processes because of resource shortages which lead to longer task execution delays or even execution stoppages. This type of fault can eventually lead to performance degradation that is not acceptable for real-time and/or time-sensitive applications.

Thus, it becomes evident that the execution of the tasks has a solid impact on resource utilization and vice versa. This fact has led us to examine the resource utilization metrics in order to take proactive fault tolerance decisions to reduce the adverse effects of process faults.

3.2 Proactive fault tolerance

Given that the modus operandi of the edge computing paradigm relies on a vast number of compute nodes operating simultaneously, it is extremely important to consider component failures as an inevitability. By doing so, it is important to ensure that the infrastructure will continue operating without interruptions and QoS deteriora- tion. The main way of ensuring this service continuity, is by triggering migration policies and by utilizing backup components, which automatically replace the failed ones in a manner which guarantees the QoS.

The replication process of a node, such as a virtual machine, requires a certain waiting period, which would provoke QoS degradation. Thus, fault tolerance should be achieved by following a proactive approach. At any given time, the network should contain a specific number of computational nodes, which can remain idle until one of the already working components ceases to function properly. Given that redundant computational nodes may be requested, it is important to keep this redundancy to a minimum. However, by utilizing machine learning algorithms, it is possible to extract information regarding the behaviour patterns of the services and the process faults that occur. Hence, this enables the fault tolerance functionality to manifest in a manner which will ensure that the operations will continue to take place uninterrupted and that the overall redundancy cost will be kept to a minimum.

The proposed IPFT model provides fault predictions by using data features which are associated with the resource usage in distributed edge environments. The IPFT monitors the resource consumption that takes place on each processing node in order to reveal, at run-time, insufficient processing capabilities that may result in potential QoS degradations. In case that the deployed resources cannot satisfy the increasing amount of demands within a specified time-frame, the IPFT will then trigger mitigation policies such as proactive node replication and task execution migration.

The multichannel neural network part of the IPFT mechanism of a node takes into consideration the state of the other available nodes when predicting its future state. This way, in case of a predicted fault, it can perform task migration to the nodes that are already up and running, provided that there is enough computational capacity available. By doing so, we avoid immediately resorting to node replication which could lead to a cost and time deployment increase for setting up a new node. As mentioned before, the replication of a node requires a certain waiting period, which can have grave ramifications on the performance of the edge environment. The IPFT can predict the future needs for node replication in a time horizon longer than the replication time. Thus, the timely triggering of node replication processes and the corresponding task migration prevents the occurrence of the process faults. As illustrated in Fig. 1.1, when a specific processing node is predicted to present high resource
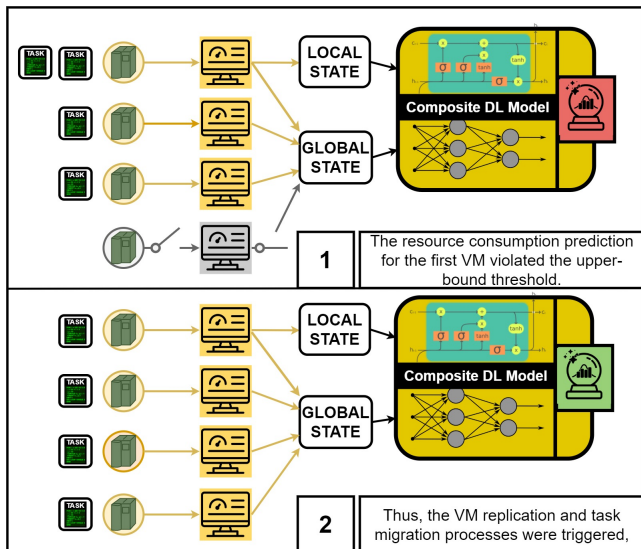
Fig. 1 – The IPFT triggers virtual machine replication and task migration based on resource utilization predictions.

utilisation, then a node replication process will be triggered (Fig. 1.2). This way, the future tasks to come will be accommodated by the new node, avoiding tasks rejections and long execution times.

The IPFT operates in a manner which consists of four main stages, as one can see in Fig. 2. In stage 1, it monitors the resource metrics of the processing nodes. In stage 2, it predicts the maximum resource usage that is expected to take place within a specified time-frame. With regards to the stage 3, there are three distinct scenarios which dictate how the rest of the operation shall be carried out. The stage 3 processes are carried out independently for each of the processing nodes. The three distinct scenarios and their interactions with the corresponding actions taken at stage 4 can be summarized as follows:

- Stage 3A. If the resource usage prediction is lower than a specific lower-bound threshold, then the processing node is considered under-utilized and thus a decommission request for this specific node is issued. During stage 4A, the tasks that were assigned to this node are redirected to alternative locations and the decommission process is completed.

- Stage 3B. If the resource usage prediction is between the lower-bound threshold and the upper-bound threshold, then the resource consumption rate is considered ideal and thus there is no need to perform any additional actions.

- Stage 3C. If the resource usage prediction is higher than a specific upper-bound threshold, then the processing node is considered over-utilized and thus a replication request is issued. Stage 4C takes place after the creation of the new processing node. During stage 4C, a fraction of the tasks that were assigned to the over-utilized processing node are redirected to the newly created one.

The IPFT works jointly with a workload balancing mechanism that gives higher priority to the nodes with low resource utilization predictions and lower priority to the nodes with higher resource utilization predictions. This is a common practice to many task offloading mechanisms that use different criteria to balance the workload [29]. As an example the MinMin task offloading method prioritizes the smaller tasks to be executed in the nodes that will be available sooner. While the MaxMin task offloading method prioritizes the larger tasks to be executed in the processing nodes that will be available sooner. The size of the tasks is estimated based on the number of its million instructions or their estimated completion time. In this way, IPFT modifies the behaviour of the task offloading mechanisms taking into consideration the resource usage predictions for the migration of the tasks.

3.3 Threshold-based decision making

Task migration and node replication occur when a resource usage prediction metric is higher than a specified threshold value. This type of threshold-based approach is used in many decision-making mechanisms in cloud/edge computing [30]. The IPFT involves two thresholds. If the value of the resource utilization prediction is higher than the upper-bound threshold, the IPFT invokes a node replication process. If the prediction value which corresponds to a specific processing node is lower than lower-bound threshold, then this particular processing node is turned off (e.g. for reducing the total energy consumption [31]).

The appropriate selection of these two thresholds is integral to the performance of the IPFT. A high value in the upper-bound threshold will result in a system which is not sensitive and reactive enough to the workload fluctuations. While a low value in the upper-bound threshold will make the system to react and trigger unnecessary node replications. Similarly, the lower-bound threshold should be appropriately fine-tuned. A low value will make the infrastructure to continue using under-utilized processing nodes. A high lower-bound threshold will also turn off processing nodes that are necessary to the smooth operation of the edge infrastructure [32].

In order to identify the optimal threshold values, we propose the use of a grid-search approach that iteratively tries sequential thresholds in order to converge close to the optimal values. These values maximize the fault tolerance evaluation metrics of reliability and maintainability, which will be extensively discussed in the upcoming experimental section. The selection of the threshold values is heavily dependant on the characteristics of each application and the underlying physical infrastructure. The literature provides recommendations in similar problems and mechanisms which, despite providing suboptimal results, serve as valuable guidelines towards establishing some standards regarding how these bounds are chosen [33].
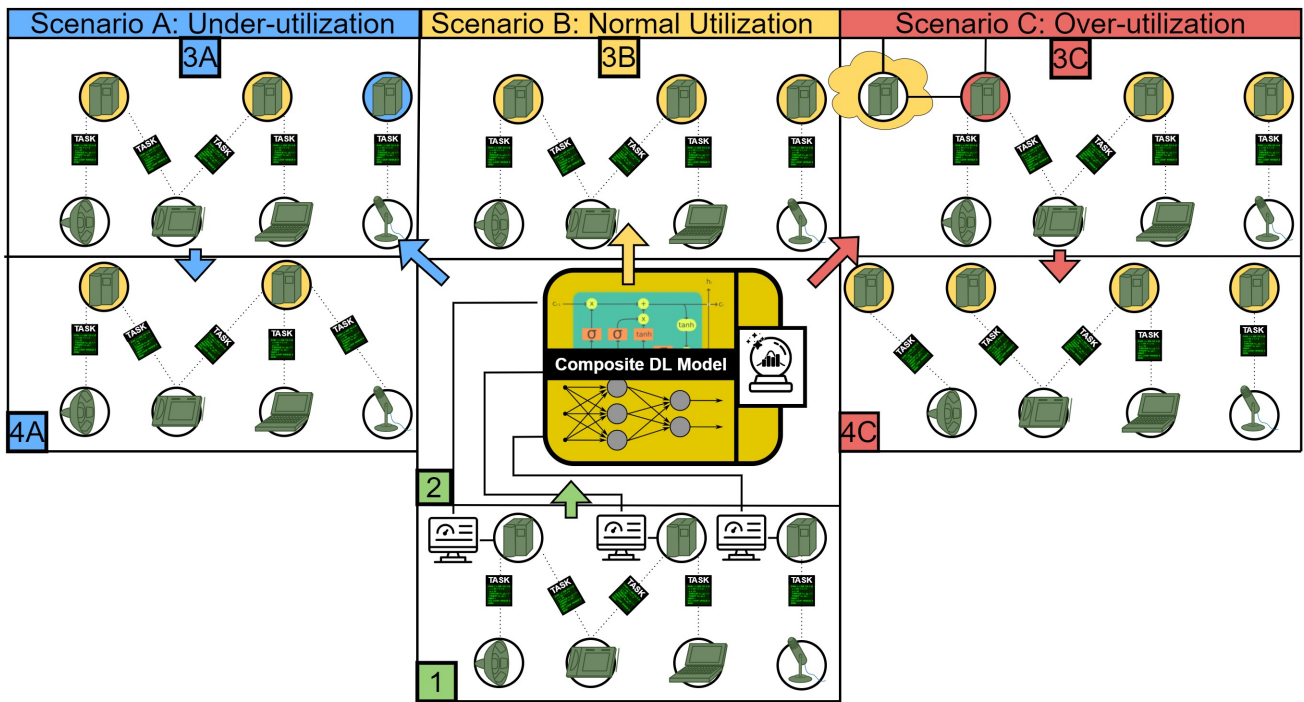
Fig. 2 – The four stages of the IPFT pipeline which span over three distinct scenarios.

Given that the predictions are quite accurate and the threshold values are chosen optimally, the resulting system is expected to be highly robust and able to provide satisfying availability. For the best results, the resource utilization predictions should derive from the resource consumption behaviour of each individual processing node, while taking into consideration the overall resource consumption behaviour of the entire edge computing infrastructure. In the next section we will describe the deep learning model that predicts the resource utilization, as well as the way it simultaneously leverages the resource consumption metrics of each individual node and the entire edge infrastructure.

4. A COMPOSITE DEEP LEARNING ARCHITECTURE FOR RESOURCE USAGE PREDICTION

A composite deep learning network with the HBES model is proposed to provide accurate resource utilization predictions for the IPFT method. The composite deep learning network is designed to satisfy the particularities of the edge infrastructure and the resource usage metrics. Since resource metrics like CPU, RAM, disk, and bandwidth have sequential dependence, RNN can provide an appropriate type of neural layers. RNN combines the advantages of deep learning with the characteristics of time series forecasting. There are different types of RNN architectures and the two most prominent are the GRU and LSTM. Each individual processing node is examined separately for future possible process faults. However, in order to trigger the node replication, the deep learning model of each node should be aware of its own status and the whole
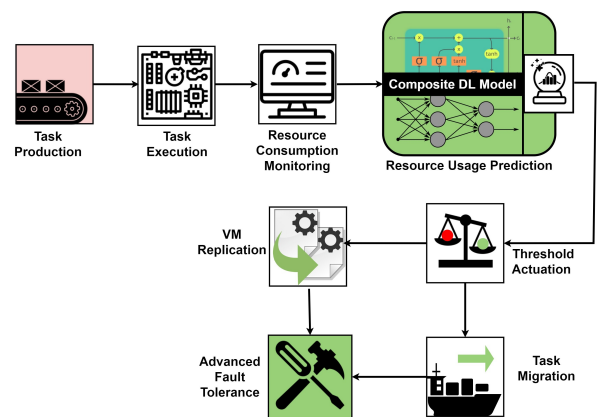

Fig. 3 – A pipeline from task production to advanced fault tolerance.

edge infrastructure status. In this paper, the edge node which is examined is also called local and the whole edge infrastructure is called global. Because the local and the global status affect each other we propose the use of a composite deep learning model that combines the two in order to provide the local resource utilization predictions. The way that these two different sources of information are combined will be explained in the next subsection.

The workflow of the resource usage prediction in an edge computing environment is depicted in Fig. 3. In the beginning, the edge devices generate tasks which are partially or fully executed to the edge computing nodes. During the task execution the nodes are monitored in order to keep the resource utilization metrics. The resource utilization metrics are provided to the composite deep learning model in order to predict the resource utilization in the next time horizon. Next, the resource

usage predictions can be used by a thresholding method in order to trigger node replication and task migration. In the following subsections, we will describe the theory behind the key parts of composite neural networks, the RNN and the HBES for the resource usage prediction.

4.1 Two channel architecture for resource usage prediction

The most commonly used architecture, when it comes to neural networks, comprises of layers that are stacked one after the other in a serial manner. Input data is fed to the first layer, and one by one each of the layers transform the data and feed it to the next one. When building a neural network however, we have the option of putting layers or even a series of layers in parallel, to take advantage of complex input structural properties. These parallel sections of the network are processing different parts of the data independently and subsequently concatenate their output to send it to the final section of the network. We decided to use an architecture, such as the one described above, utilizing two separate parallel series of layers (two channels) [34]. This allows us to better handle the local and global data metrics that are available.

The main advantage of using a multichannel neural network architecture is the ability to use different kinds of layers in the input stage of the model. As an example, many use cases from the literature use a convolutional network and a feedforward network at the input stage to accommodate data having both numerical and image properties. Multiple channel neural network architectures have the advantage that they can group together the data whose properties present high correlation [35]. This happens by having different sequences of layers handling different kind of data.

A multichannel neural network can be more time and resource-consuming to train and infer compared to a vanilla serial neural network, because of its more complicated structure. We also cannot properly evaluate the channels, or understand their individual effect when it comes to the final output of the model, since training is performed start-to-finish without giving us more details about the process.

RNNs are a type of neural network that do particularly well when facing problems with sequential data. They are used to solve problems such as text prediction or voice recognition, and they are deemed effective when dealing with time series-based problems as well. Consequently, we chose to include an RNN in our model as the first input channel, since it is going to handle the sequential data of a single node (i.e. the one that the model is actively trying to predict). The input is the monitoring measurements with an interval record of one minute. The channel consists of one or two RNN layers followed by a feedforward layer, and sends the results to the output part of the network.

For the second channel of the model, we chose to implement a feedforward network. This part is being fed with the global monitoring state, as well as a transformed timestamp feature that refers to the day of the week and the part of the day. It is obvious that the chance that a particular node to be overloaded in the immediate future is tied closely to the state of the other nodes that are collaborating with it to handle the requests. The channel of the feedforward network includes dense layers, that also have dropout layers in between. This architectural decision works as a regularization measure that stops our model from overfitting during training. Once the channels handle the input part of the model, their concatenated results are given to the output section of the network. The goal of this combination is to take into account the processed data and output a prediction for the resource utilization of a particular node, in a sufficient time horizon after the state/last input we fed to the model. This is a feedforward neural network as well, with dense and dropout layers until the output layer. The choices regarding the merging of the two channels and the concatenation size of the input section, are handled by the hyper-parameter selection algorithm.

In order to train itself on a dataset, the model tries to predict the future state of a particular node based on the complex input referring to both the local node in question and the global overall state. Once the model outputs a prediction, it compares the prediction to the actual output value utilizing a loss function. Loss functions such as MAE or RMSE, are a way to tell how good or bad a regressive model is at predicting the correct values. Based on the error calculated, the model shifts its weights and parameters using a variant of gradient descent, back-propagating through the network affecting all layers. A function called optimizer is describing how the procedure of weight-shifting/network-optimizing takes place. Both the architecture, and the training of a neural network provide us with many options and hyper-parameters to tweak. In order to ensure a quick and optimal choice of hyper-parameters for our model we implemented the HBES algorithm that will efficiently make these decisions.

4.2 Recurrent neural networks for time series data

Artificial neural networks can be defined as function approximators, mapping lower level data representations to higher and disentangled data representations. RNN [7] is a type of artificial neural network, which facilitates dynamic temporal behaviour, captures data sequences, and maintains the previous input states.

The RNN architectural paradigm is based on various neuron-like nodes organized into successive layers, where each node is connected with nodes of the next successive layer and also has recurrent connections. Utilizing this particular concept, information regarding previous data inputs is allowed to affect future outputs, thus making RNN a solid option for time series modelling while taking into account contextual information. By monitoring edge computing infrastructures, we gather sequential data and predict the future resource usage metrics with RNN, based on their current and previous values.

The main problem that RNN encounters is the vanishing gradient problem. This problem emerges during the training stage of the RNN, when the gradients become vanishing small preventing the weight updates of the RNN. Various gate-related architectures have been introduced in order to tackle the vanishing gradient problem. Through the use of gates, the network is able to properly maintain relevant information and to successfully pass it down to the next time steps. The two most notable ones are the LSTM networks and the GRU [8] networks.

4.3 Long short-term memory

The LSTM network architecture was created to tackle the problem of vanishing gradients in RNN. The importance of using this complicated architecture can be highlighted by pointing out how much context can be offered in finding a solution. When dealing with a time series data problem, such as predicting resource usage, we can get much more useful information if we look at historical data of our machines' usage, rather than just glancing at their current state. This way, we can better understand concepts like trend, which can only be explained over time. LSTMs, just like regular RNNs, utilize the hidden state to connect the consequent nodes so as to enable better understanding of temporal data. However, they also use a cell state, which is another connection between the nodes. Each LSTM cell can read from the cell state, write to it or reset it via the use of gates. There are three gates in total, each activated by a sigmoid function. This ensures that the model remains differentiable, since sigmoid offers smooth curves in the range of 0 to 1. Each one of the gates takes as inputs the actual input as well as the hidden state of the previous time step. In addition to the gates, a vector called $\tilde{C}$ is responsible for carrying the candidate information that can be added to the cell state. $\tilde{C}$ utilizes a tanh layer, which is in charge of limiting the vanishing gradient phenomenon. To this extent, the cell information can be kept longer without vanishing. The way this is achieved is by keeping the gradients zero-centred, between the values of -1 and 1.

The input gate is handling incoming data, and controlling whether the memory cell should be updated. It is applied to $\tilde{C}$, and the result is then added to the cell state. The sigmoid activation of the gate is used to either mitigate or enhance the effect that the new information should have on the cell state.

The forget gate is the entity that is responsible for selecting the information that is deemed less important, and it removes it from the cell state by soft-resetting its values. Additionally, by utilizing the sigmoid function, it produces a scaled output for every value that is saved in the cell state.

Finally, the output gate is the final layer before the new hidden state is produced. It uses the sigmoid function as a filter to be applied to the cell state after it goes through a tanh layer first. After this process is completed, both the
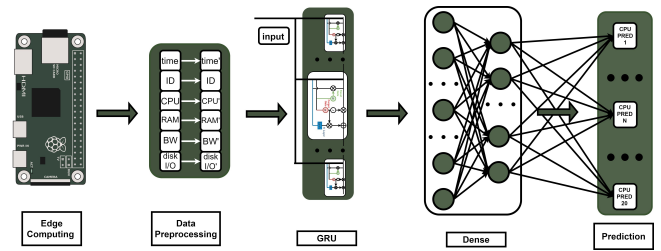

Fig. 4 – Integrated RNN in the resource usage prediction.

hidden state and the cell state compose the output of the LSTM cell which will be inputted to the next time step.

4.4 Gated recurrent units

GRU and LSTM are similar as both of them manage to prevent the vanishing gradient problem by utilizing gate structures. What sets them apart is the fact that GRU combines the forget gate and input gate to form a single update gate. By reducing the number of gates involved, GRU is able to provide less complex structures and thus, be more computationally efficient when compared to LSTM. At the same time, GRU manages to perform equally well. GRU networks also facilitate the hidden state mechanism which connects one unit of the network to the next, thus allowing the manifestation of dynamic temporal behaviour in a similar manner. Each GRU unit is indicative of a specific time step that facilitates the transfer of important information through the time continuum. Furthermore, it contains two distinct gate structures. The first one is referred to as the reset gate while the second one is referred to as the update gate. They both bear sigmoid layers which provide smooth curves in the 0 to 1 zone, thus ensuring that the model will remain differentiable. By squishing the values between 0 and 1, the sigmoid activation also helps the network learn which data is important or not and then accordingly keep it or forget it. In order to contextualize the GRU paradigm in accordance to edge computing, we input vectorized representations with information such as the perspective timestamps, the resource utilization of CPU, RAM, bandwidth, and disk through the data preprocessing as illustrated in Fig. 4. The functionality of GRU networks is carried out in the form of the following steps. As explained before each GRU uses a reset gate and an update gate. Each of these gates has two weight matrices. The first one corresponds to the input while the second one corresponds to the hidden state. The reset gate of GRU is responsible for deciding how much of the past information shall be forgotten. Much like in the case of LSTMs, the first step is to multiply the input and the hidden state by their corresponding weights. The sum of the multiplication results is then passed through a sigmoid layer. The update gate is in charge of determining how much of the information gathered over the previous time steps needs to be passed along for future use. In this regard, its behaviour is quite similar to the one of the reset gate. The first step requires the multiplication of the input and

the hidden state by their perspective weights. The hidden state entails information derived from the previous $t - 1$ units. Then, the multiplication results are added together and passed through a sigmoid layer. The output of the up- date gate will be referred to as u. The next step is to create a candidate new hidden state. Similarly, to the reset and update gates, there are also two weight matrices involved. The first one corresponds to the input and the second one corresponds to the hidden state. The first step towards creating a candidate new hidden state is to multiply the input by its correspond- ing weights. The second step is to calculate the Hadamard product in an element-wise manner between the hidden state and the output of the reset gate. This process is es- sential for deciding how much of the information gathe- red during the previous time steps will be removed. The Hadamard product is then multiplied by the weights of the hidden state. The results of the two multiplications are then added together. The sum is passed through a tanh layer, which minimizes the effects of the vanishing gradient phenomenon. This is performed by distributing the gradients in a sufficient manner, within a zero-centred range. Thus, it enables the information to flow longer without vanishing. The product of the operations so far is the candidate new hidden state will be referred to as h'.

In order to get the updated hidden state, the first step required is to perform element-wise multiplication to the output of the update gate and the hidden state. The second one is to perform element-wise multiplication to the h' and the product of $1 - u$. The updated hidden state is the sum of the two multiplication products. The updated hidden state is then carried over to the next GRU unit, which corresponds to the next time step.

4.5 Evolutionary strategy

Evolutionary strategy [36] belongs to the category of evolutionary algorithms that are population-based meta-heuristic optimization approaches inspired by the principles of biological evolution. The formulation of evolution strategy is based on successive iterations of mutation and selection over a population of candidate solutions. The candidate solutions, also named individuals, are initialised in random positions in an n-dimensional space and move toward positions that minimize an objective function. These dimensions are the numerical GRU-RNN hyper-paramaters that should be optimized.
For the needs of hyper-tuning GRU-RNN, we used its numerical hyper-parameters as the search space of the evolution strategy and the mean squared error of the candidate RNN as the fitness function. In each iteration, a number of RNNs are trained and evaluated with the mean squared error, whereas the most accurate of them are mutated to the next iteration. The mutation is a stochastic process based on a normal distribution that introduces variations in the best fit individuals of each iteration. In the beginning, the exploration for different candidate so-

lutions is intensive, making stronger mutations towards new areas of the search space. In each iteration the exploration decreases and the exploitation of the best fit individuals increase using a self-adaptation control variable. This means that the mutation introduces strong variations in the first iterations and the variations decay as the evolution progresses in order to converge to a close to optimal RNN architecture.

Hyper-tuning deep learning models with an evolution strategy in contrast with other evolution algorithms, like genetic algorithms, has the advantage of not recombining different neural network topologies that may have significant discrepancies in their phenotypes. This happens because the crossover of the genetic algorithm has the difficulty that the parents may have different architectures that cannot be unified in their offspring. A typical example is if the one parent is a 2 layered LSTM-RNN followed with 6 dense layers and the second parent is a 2 layered GRU-RNN followed with 4 dense layers. Thus, the phenotypes of LSTM and GRU cannot be smoothly recombined. On the other hand, evolution strategy is based only in the selection and the mutation which smoothly lead the evolution process. Specifically, the mutation operations introduce variations into the survived candidates providing the opportunity to test neighbour solutions that may lead to an improved fitness value.

4.6 Bayesian optimization

Bayesian optimization [37] is widely used to estimate hyper-parameters in machine learning and deep learning models. It was an obvious option for the searching process in the categorical dimensional space, in order to find the close to optimal nominal hyper-parameters of the RNN. Bayesian optimization iteratively requests new observations of the search space with an acquisition function and estimates the objective function with a surrogate function. The increase of the Bayesian optimization observations gives a higher probability for the global optimum location. Nonetheless, we should take into consideration that the number of observations are finite and computationally expensive so the smart search process should select points that maximize the probability to find a new optimal following an exploration vs exploitation trade-off.

The surrogate function approximates the objective one and is updated every time the objective function is evaluated in the new candidate points. The acquisition function decides where to sample next in the iterative process of Bayesian optimization, finding the points that maximize the expected improvement. The expected improvement is a function of two components. The first estimates the regions that the surrogate function has optimal points and the second estimates the regions with high prediction uncertainty that have not explored efficiently yet.
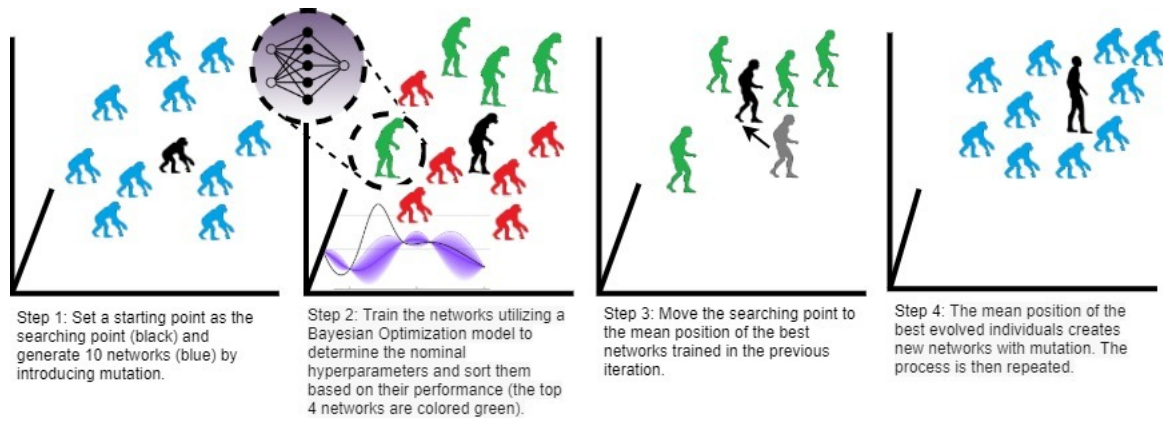
Fig. 5 – The four main steps of the HBES process for close to optimal RNN optimization.

4.7 Hybrid evolution strategy with Bayesian optimization

The hyper-parameter optimization for a composite neural network is a prominent challenge as it includes the important architectural decisions for a close to optimal topology. The HBES constitutes an innovative, holistic and unified approach for hyper-tuning by merging the evolution strategy and Bayesian optimization methodologies. The evolution strategy is responsible for evolving a population of candidate deep learning models based on their numerical hyper-parameters and each individual candidate solution estimates its nominal hyper-parameters with the Bayesian optimization as it is described in Algorithm 1. The numerical hyper-parameters are the number of recurrent layers and feedforward layers, the number of neurons for each layer, the lookback, epochs, the batch size, and the percentage of dropout and learning rate. The nominal hyper-parameters are the type of neural layers, the activation functions and the optimizers. The gained knowledge of the nominal hyper-parameters is universal through the population and updated by all the individuals over the generations. The ultimate goal of HBES is through the Bayesian evolution process to converge to a close to optimal solution and to train deep learning mod- els that can predict timely and accurately the resource utilization of the next time steps. The four main HBES steps for one iteration of the individuals evolution are illustrated in Fig. 5.

5. EXPERIMENTAL EVALUATION

To evaluate the IPFT methodology, we make two types of experiments. First, we experimentally evaluate and compare the applicability of RNN with the HBES against state-of-the-art methods using a real dataset. This dataset is constructed by monitoring Raspberry Pi's in an edge computing infrastructure. Next, we leverage the resource utilization prediction model in order to develop an IPFT mechanism that takes intelligent replication and migration decisions in a sufficient time before the process faults occur. The sufficient time in the context of intelligent

---

Algorithm 1 Hybrid Bayesian and evolution strategy

Step 1: Initialization of evolution strategy
Set the starting search point of the algorithm. Usually a_1=[0.5, 0.5, ..., 0.5] since we have already scaled our hyper-parameter options down to [0,1]

Step 2: for i = 1, 2, ..., n_{pop}:
i) add some random noise to the search point
ii) Scale back from [0,1] to the hyper-parameter search space to create the ordinal hyper-parameter values for the network to be trained
iii) Bayesian optimization with GP
  1) Apply a Gaussian process prior on f
  2) Observe f at n_0 points according to an initial experimental design
  3) Initialize $n = n_0$
  4) Repeat while $n \leq N$
  a) Update the posterior probability distribution on f using all available data
  b) Let x_n be a maximizer of the acquisition function over x.
  c) Observe $y_n = f(x_n, x_i(t+1), v_i(t+1))$
  d) $n \leftarrow (n+1)$

Step 3: Sort the results and its corresponding hyper-parameters

Step 4: Calculate the new search point by averaging the points of the top_n networks

Step 5: Go to Step 2 until desired number of iterations is completed

---

replication and migration involves the service deployment time of the processing nodes and the scheduling of the new tasks on that node. The experimental evaluation of IPFT took place in a seven day edge computing simulation using the CloudSim Plus framework [38]

5.1 Experimental evaluation in resource usage prediction

The edge infrastructure we used includes Raspberry Pi3 as processing nodes with a 64-bit quad-core ARM Cortex-A53 at 1.4GHz, loaded with Raspbian operating system which is a version of Debian Linux. The dataset constructed by a monitoring tool implemented in Python 3 using the libraries psutil [39] and GPUtil [40]. We monitored the real-time usage of CPU, RAM, disk and bandwidth in one second time interval. The deployed application was a natural language processing text classification. The use case was to make the text classification on an edge computing environment, locally, close to the text owners and not in cloud computing infrastructures for privacy issues. The reason for this choice is that the text owners did not agree for their texts to be transferred and processed in remote servers. In order to control the application remotely and take the resource us- age datasets we used the SSH protocol, but we did not have the privileges to access the processed texts.

5.1.1 Model implementation and frameworks for comparison

The HBES model and the RNN multi-output regression model are implemented in Python 3 using the frameworks NumPy, pandas, statistics, Scikit-learn, SciPy, Scikit-Optimize, TensorFlow 2 and Keras. The environment we used for the training and the evaluation of the model was the Jupyter notebook of the Google Colaboratory. The experiments' source code is available for any kind of reproduction and re-examination in our GitHub repository [41]. In this experimental setup we used the HBES with GRU as RNN and compared the results with a time series baseline approach, the machine learning meta-model for resource usage prediction AUCROP, the uto-sklearn, the XGBoost, our previous LSTM with genetic algorithm model (GA-LSTM) [9] and the Keras-Tuner.

5.1.2 Evaluation results and discussion

Our initial time-series analysis provided results which indicate positive correlations for lags in a value range from 1 to 22. This finding confirms the strong self-similarity property that the sequences of resource usage metrics have. Afterwards, using the ARIMA forecasting model we evaluated the resource metrics predictions. For instance, the CPU RMSE was 18.474. After comparing the results of the statistical models with the ones of the machine learning and deep learning approaches we found that the latter had an improvement that surpasses the 20% RMSE
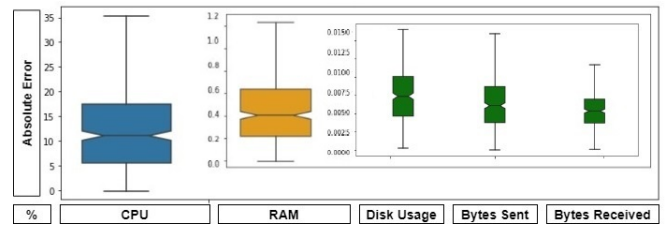

Fig. 6 – GRU-RNN with HBES prediction errors of resource usage metrics.

in most cases. Because of that, we decided to focus our research on the machine learning and deep learning models. Table 1 summarizes the experimental results. The first two columns provide the aggregated RMSE and MAE including all testing values of the devices, and the resource metrics. For RMSE, which gives an extra penalty to predictions with significant errors, we can see that HBES- GRU had the best performance. In the column entitled MAE, we see that the two best models are the auto-sklearn and HBES-GRU. Their prediction errors are very close and they have a significantly better performance compared to other models. CPU-1 and RAM-1 columns represent the RMSE and MAE for the processing edge node which had the least accurate predictions in the infrastructure. In addition, Fig. 6 illustrates the 25th and 75th percentile, the median, the min and the max of the error value metrics. These metrics include CPU, RAM, disk usage, and bandwidth in terms of the bytes sent and received. Regarding the disk usage and the bandwidth the prediction errors were insignificant. This is not only due to the ability of the HBES-GRU to provide accurate predictions, but due to the small fluctuation in these two resource metrics as well. The fluctuation in CPU is much greater than in RAM and HBES-GRU captures in a better way the various changes when compared to the other models. XGBoost has better performance than HBES-GRU in RAM. This may be justified by the ensemble structure that XGBoost has. XGBoost can build specific decision trees for the residuals of RAM and target on its slow change behaviour.

Lastly, we see the inference times of the models which are required in order to provide a single prediction or a batch with one hundred predictions. All time measurements are in seconds. All the inference times, except from auto-sklearn, are within a range from 11 to 60 msec. These inference times indicate that resource usage prediction is a rather fast process which can be incorporated in time sensitive applications. In this research we did not compare the training times because we wanted to make an exhaustive smart search in the hypothesis space and see the limits of accuracy that the different models can achieve. It is worth noting that we have made experiments using a wide range of time-frames. However, the measurements in Table 1 are produced using a 10 minute time-frame. We chose to illustrate this time-frame because it is close to the actual time which is required for the deployment time of a node. From the results we see that even if GRUs are simpler in their structure compared to LSTMs, due to their

Table 1 – Comparison of single-output & multi-output prediction methods of resource usage metrics.

Method	RMSE	MAE	CPU-1 (%)		RAM-1 (%)		Infer. Time	
			RMSE	MAE	RMSE	MAE	Single	Batch
HBES-GRU	<u>0.0641</u>	0.0276	<u>15.918</u>	<u>12.815</u>	1.694	0.580	0.033	0.038
GA-LSTM	0.0674	0.0338	16.099	12.838	1.746	0.917	0.020	0.024
Keras-Tuner	0.0785	0.0377	16.291	13.290	2.631	0.818	0.042	0.042
AUCROP	0.0814	0.0414	17.235	14.009	2.480	1.482	<u>0.004</u>	0.011
XGBoost	0.1139	0.0599	16.457	13.569	<u>1.515</u>	<u>0.472</u>	0.060	<u>0.010</u>
Auto-sklearn	0.1055	<u>0.0243</u>	52.659	17.856	1.546	0.526	0.263	0.572

lack of a dedicated output gate, they had slightly better performance.

Furthermore, the conducted experiments enabled us to reach the following conclusions:

- In most metrics the deep learning models (HBES-GRU, GA-LSTM, Keras-Tuner) have better performance than machine learning models (AUCROP, XG-Boost, auto-sklearn).

- The evolutionary algorithms for hyper-tuning (HBES-GRU and GA-LSTM) have better performance compared with the simple Bayesian optimization (Keras-Tuner).

- One can witness a significant improvement when using the hybrid Bayesian and evolution strategy approach instead of simple genetic algorithms.

5.1.3 Convergence of hybrid Bayesian evolution strategy

The convergence and the location of the global optimum are two of the most important topics in the domain of evolutionary algorithms. The convergence means that as the population evolves, the individuals go closer to the optimal solution shrinking their divergence. However, we cannot be sure if the convergence points in the genotype space constitute a global or local minimum. For this reason, the HBES algorithm in the beginning of the evolution process expresses a strong variance in the mutation which decays over iterations. Concurrently, we keep the best genotype found over all the iterations. The convergence of HBES is illustrated in Fig. 7. We observe that in the beginning the average population error per iteration fluctuated strongly. In some iterations it is trapped in local minima as an example between the iterations six to eleven. In some other iterations it lays in plateau regions, such as between iterations twenty to twenty five. Yet, using the mutation the individuals eventually escape from the plateau regions and the local minima and move towards close to optimal regions.

These close to optimal regions in the genotype space are decoded to the close to optimal GRU-RNN architectures in the phenotype space. These GRU-RNN architectures provide the most accurate resource usage predictions for CPU, RAM, disk, and bandwidth usage in an edge computing infrastructure.
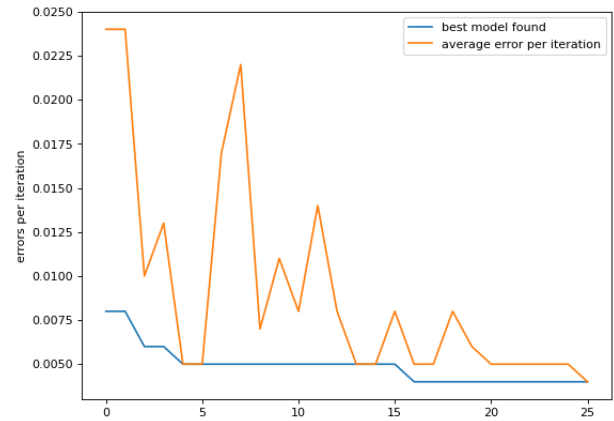

Fig. 7 – The convergence of HBES for close to optimal RNN.

5.2 Experimental evaluation in proactive fault tolerance

The promising experimental results of the HBES with RNN in regards to resource usage prediction, motivated us to continue the experiments in order to evaluate its applicability as a proactive fault tolerance mechanism. Specifically, we used the HBES to make a smart search in candidate two-channel deep learning models and applied the thresholding method as described in Section 4.

5.2.1 Experimental simulation

The composite deep learning model was integrated in an edge simulation of CloudSim Plus. We simulated an edge infrastructure that consists of a set of nodes, five available to us by default, and another 15 that can be activated for intelligent replication when needed. We simulated the local and global resource monitoring process, measuring CPU, RAM and bandwidth values, and saving those values every 60 seconds (time step). The task offloader of the infrastructure was receiving incoming traffic and was assigning each task on a node, based on the following scheduling algorithms: RoundRobin, MinMin, and MaxMin.

The local and global resource usage metrics are being monitored and then fed to the IPFT mechanisms of each processing node. During every single time step we use the monitoring data in order to formulate the appropriate data representations, featuring the past time-series measurements of a single node, as well as the state of the in-

frastructure as a whole. The input is then fed to the composite deep learning model, enabling it to make predictions of resource usage for every node in a time horizon of 10 minutes. In this experimental setup we made the assumption that the preparation time for the infrastructure to assure its availability and robustness to faults is 10 minutes.

The simulation lasted for seven days and the tasks were generated by a mixture of Gaussian probability distributions that simulate a realistic application workload behaviour [42]. The processing nodes simulated the processing capabilities of Raspberry Pi's. We defined a process fault if the time execution of a task lasts more than one second. The selection of one second is a reasonably acceptable latency for several data analytic applications [43]. Trying different latency times for the process faults, we noticed that the IPFT performance was better than the reactive approach. In the reactive fault tolerance approach, a node replication is triggered in case of a fault is taking place. In Table 2, as we will thoroughly discuss in the next section, we compare the IPFT mechanism to the reactive fault tolerance approach.

5.2.2 Fault tolerance evaluation metrics

In order to evaluate the performance of the IPFT mechanisms, we used a set of fault tolerance evaluation metrics [44]. Mean Time To Failure (MTTF) is defined as the expected time for a failure to occur given that the system functions properly. MTTF is an evaluation metric which corresponds to the overall inability of the edge infrastructure to operate properly and thus, it is calculated by taking into consideration the number of faults regardless of the actual processing node that failed. Mean Time To Repair (MTTR) is defined as the expected time required to repair the system after a failure occurs. For the MTTF the higher values are better and for MTTR the lower values are better. These evaluation metrics are calculated in terms of seconds.

Two additional fault tolerance evaluation metrics are reliability and maintainability. Reliability refers to the ability of an edge infrastructure to run continuously without any failure. Maintainability refers to how easily a failed system can be repaired. Both reliability and maintainability are numbers with no units and higher values mean better performance.

5.2.3 Evaluation results and discussion

The experimental results are summarized in Table 2. We compared the IPFT mechanism to the Reactive Fault Tolerance (RFT) approach. The RFT approach performs node replications after a fault occurs. Regarding the task offloading algorithm we used the Round Robin (RR) [45], the MinMin and MaxMin [46]. The experimental results show the superiority of IPFT compared to the RFT in all evaluation metrics. In addition we see that the outcomes are significantly affected from the task of loading mechanism.

This happens because the task offloading algorithms also integrate a workload balancing methodology with different criteria as we will discuss in the following paragraphs.

The number of generated tasks in all experimental setups was close to 1,500,000 with some parts of the day having an intensive task generation (i.e., 11:00 -13:00), while some other parts of the day a small number of tasks (i.e., 02:00 - 04:00). We simulated this task generation behaviour because it is close to the activity of many user applications during a day. In this way, we observed that the infrastructure made intelligent replications during the parts of the day with an increased task generation. Respectively, the infrastructure turned off the edge nodes when the IPFT mechanism predicted an under-utilization of the processing nodes. Sometimes there were some sudden spikes or drops in task generation and resource utilization, but the infrastructure using the IPFT mechanism could dynamically and timely adapt.

In Table 2, we can see that in RR, MaxMin and MinMin the MTTF in IPFT has been increased compared to the RFT. This means that leveraging the resource usage predictions, faults occur more sparsely and rarely. We can see from the MTTR metric that in the event of a fault, the infrastructure will recover very quickly, scheduling the new tasks in processing nodes with low resource utilization. The reliability metric shows that by using the IPFT, the edge infrastructure can provide the expected results up to 93% of the simulation length, even during the stressing time periods of the simulated days. The significant improvement noticed for the maintainability metric, declares that even if a fault occurs, the IPFT will increase the robustness of the edge infrastructure. In other words, the IPFT will take timely the right measures by triggering node replication and task migration, in order to reduce the likelihood of subsequent fault occurrence. A fault is recorded taking into consideration all the edge nodes that are currently active. This means that the MTTF value of 13.309 seconds in IPFT MaxMin includes the faults of different edge nodes. In addition to that, some generated tasks had a large number of million instructions that would have provoked a fault because of the CPU unavailability in the processing nodes. In this case, we wanted to know how these tasks affect the MTTF and MTTR. From the analysis of the results we saw that the variance of the task size is the reason that we see that the three different task offloading mechanisms have different performance. In particular, the MaxMin algorithm gives higher priority in big tasks, thus we see a significantly better MTTF metric.

During the simulation we examined the IPFT decisions and how the edge environment operates. The simulation confirmed that the infrastructure takes advantage of the timely decision to trigger proactive actions, such as intelligent node replication and task migration before the amount of tasks overwhelms the processing nodes. This can be particularly important for the infrastructure provider as it can save costs and energy, by shutting down

Table 2 – Evaluation of reactive and intelligent proactive fault tolerance methods

	MTTF	MTTR	Reliab	Maint
RFT RR	2.864	19.657	0.741	0.048
IPFT RR	9.506	3.343	0.904	0.230
RFT MinMin	8.733	36.169	0.897	0.026
IPFT MinMin	8.919	5.656	0.899	0.150
RFT MaxMin	3.721	24.239	0.788	0.039
IPFT MaxMin	13.309	7.425	0.930	0.118

nodes when they are no longer needed. Additionally, the provider can achieve a smoother flow of on-time completed tasks, avoiding crashes and minimizing QoS deterioration.

In regards to the actual cost of implementing the proposed IPFT paradigm, the consumption of computational resources was 3.2% greater when compared to the reactive approach. Furthermore, the burden imposed on the network infrastructure was about 10 bytes per second, due to the information flow that derived from the need of the prediction model to have access to the ongoing resource usage. Finally, the incorporation of the prediction mechanism used around 125MB of RAM and increased CPU consumption by around 47% for an average of 2250 ms on an Intel Xeon E312xx CPU. When contemplating the substantial benefits provided by the IPFT approach, we believe that the overall implementation cost is justified and quite reasonable.

6. CONCLUSION

In this paper, we proposed a proactive fault tolerance mechanism in an edge computing infrastructure based on the resource usage predictions. In the beginning we discussed and experimentally evaluated the use of RNN for resource usage modelling. We developed a composite deep learning model that leverages in two channels the resource usage metrics of the local processing nodes and the infrastructure as a whole. We also designed a hyper-tuning algorithm that combines an evolution strategy with Bayesian optimization and surpasses commonly-used hyper-tuners like Keras-Tuner and other state-of-the-art machine learning models. Last but not least, we presented how a proactive fault tolerance mechanism can leverage the resource usage predictions trig- gering node replication and task migration.

Our experiments and results corroborated the efficiency of our proactive fault tolerance methodology, the applicability of RNN and the two-channel architecture for the resource usage prediction. The limitation of our work is that we have not yet holistically worked with other types of faults like network faults, physical faults and service expiry faults. Our future work is to make data analysis and find the execution patterns in the edge resources that are related with these types of faults. In addition, we want to see how different threshold values affect the performance of the IPFT. Specifically, we plan to further investigate the resource usage metric threshold for triggering the node replication and task migration and the time intervals for monitoring the time series metrics.

REFERENCES

[1] Theodoros Theodoropoulos, Antonios Makris, Abderrahmane Boudi, Tarik Taleb, Uwe Herzog, Luis Rosa, Luis Cordeiro, Konstantinos Tserpes, Elena Spatafora, Alessandro Romussi, Enrico Zschau, Manos Kamarianakis, Antonis Protopsaltis, George Papagiannakis, and Patrizio Dazzi. "Cloud-based XR Services: A Survey on Relevant Challenges and Enabling Technologies". en. In: *Journal of Networking and Network Applications* 2.1 (Feb. 2022). Publisher: Institute of Electronics and Computer, pp. 1–22. ISSN: 2689-7997. DOI: 10 . 33969 / J – NaNA.2022.020101. URL: `https://iecscience. org / jpapers / 100 # abstract` (visited on 07/09/2022).

[2] Kevin Boos, David Chu, and Eduardo Cuervo. "Demo: FlashBack: Immersive Virtual Reality on Mobile Devices via Rendering Memoization". In: *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services Companion*. MobiSys '16 Companion. New York, NY, USA: Association for Computing Machinery, June 2016, p. 94. ISBN: 978-1-4503-4416-6. DOI: 10 . 1145 / 2938559 . 2938583. URL: `https : // doi.org/10.1145/2938559.2938583` (visited on 07/09/2022).

[3] Claudia Campolo, Gianmarco Lia, Marica Amadeo, Giuseppe Ruggeri, Antonio Iera, and Antonella Molinaro. "Towards Named AI Networking: Unveiling the Potential of NDN for Edge AI". en. In: *Ad-Hoc, Mobile, and Wireless Networks*. Ed. by Luigi Alfredo Grieco, Gennaro Boggia, Giuseppe Piro, Yaser Jarar- weh, and Claudia Campolo. Lecture Notes in Com- puter Science. Cham: Springer International Pub- lishing, 2020, pp. 16–22. ISBN: 978-3-030-61746-2. DOI: 10.1007/978–3–030–61746–2_2.

[4] Aroosa Hameed, John Violos, Nina Santi, Aris Leivadeas, and Nathalie Mitton. "A Machine Learning Regression Approach for Throughput Esti- mation in an IoT Environment". In: *2021 IEEE International Conferences on Internet of Things (iThings) and IEEE Green Computing Communica- tions (GreenCom) and IEEE Cyber, Physical Social Computing (CPSCom) and IEEE*

Smart Data (Smart-Data) and IEEE Congress on Cybermatics (Cyber- matics). Dec. 2021, pp. 29–36. DOI: 10 . 1109 /iThings − GreenCom − CPSCom − SmartData − Cybermatics53846.2021.00020.

[5] Antonio Petrosino, Giancarlo Sciddurlo, Giovanni Grieco, Awais Aziz Shah, Giuseppe Piro, Luigi Alfredo Grieco, and Gennaro Boggia. "Dynamic Management of Forwarding Rules in a T-SDN Architecture with Energy and Bandwidth Constraints". en. In: *Ad-Hoc, Mobile, and Wireless Networks*. Ed. by Luigi Alfredo Grieco, Gennaro Boggia, Giuseppe Piro, Yaser Jararweh, and Claudia Campolo. Lecture Notes in Computer Science. Cham: Springer Inter- national Publishing, 2020, pp. 3–15. ISBN: 978-3- 030-61746-2. DOI: 10.1007/978−3−030−61746−2_1.

[6] Farhan Nisar and Bilal Ahmed. "Resource Utilization in Data Center by Applying ARIMA Approach". en. In: *Intelligent Technologies and Applications*. Ed. by Imran Sarwar Bajwa, Tatjana Sibalija, and Dayang Norhayati Abang Jawawi. Communications in Computer and Information Science. Singapore: Springer, 2020. ISBN: 9789811552328. DOI: 10 . 1007/978−981−15−5232−8_64.

[7] B. Shiva Prakash, K. V. Sanjeev, Ramesh Prakash, and K. Chandrasekaran. "A Survey on Recurrent Neural Network Architectures for Sequential Learning". en. In: *Soft Computing for Problem Solving*. Ed. by Jagdish Chand Bansal, Kedar Nath Das, Atulya Nagar, Kusum Deep, and Akshay Kumar Ojha. Advances in Intelligent Systems and Computing. Singapore: Springer, 2019, pp. 57–66. ISBN: 9789811315954. DOI: 10 . 1007 / 978 − 981 − 13 −1595−4_5.

[8] Guizhu Shen, Qingping Tan, Haoyu Zhang, Ping Zeng, and Jianjun Xu. "Deep Learning with Gated Recurrent Unit Networks for Financial Sequence Predictions". en. In: *Procedia Computer Science*. Recent Advancement in Information and Communication Technology: 131 (Jan. 2018), pp. 895–903. ISSN: 1877-0509. DOI: 10 . 1016 / j . procs . 2018 . 04.298. URL: https://www.sciencedirect.com/ science / article / pii / S1877050918306781 (visited on 02/21/2021).

[9] John Violos, Evangelos Psomakelis, Dimitrios Danopoulos, Stylianos Tsanakas, and Theodora Varvarigou. "Using LSTM Neural Networks as Resource Utilization Predictors: The Case of Training Deep Learning Models on the Edge". en. In: *Economics of Grids, Clouds, Systems, and Services*. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2020, pp. 67–74. ISBN: 978-3-030-63058-4. DOI: 10.1007/978−3−030−63058−4_6.

[10] Tong Yu and Hong Zhu. "Hyper-Parameter Optimization: A Review of Algorithms and Applications". In: *arXiv:2003.05689 [cs, stat]* (Mar. 2020). arXiv: 2003.05689. URL: http://arxiv.org/abs/2003.05689 (visited on 02/21/2021).

[11] Seong Yeol An, Yoon Seok Cha, Eun Jin Jeon, Gwi Yeong Gwon, Byeong Chun Shin, and Byeong Rae Cha. "A Pre-Study on the Open Source Prometheus Monitoring System". kor. In: *Smart Media Journal* 10.2 (2021). Publisher: THE KOREAN INSTITUTE OF SMART MEDIA, pp. 110–118. ISSN: 2287-1322. DOI: 10 . 30693 / SMJ . 2021 . 10 . 2 . 110. URL: https : / / www . koreascience . or . kr / article / JAKO202130053181362 . view ? orgId = anpor&hide=breadcrumb, journalinfo (visited on 01/25/2022).

[12] Rong Chen, Youyang Yao, Peng Wang, Kaiyuan Zhang, Zhaoguo Wang, Haibing Guan, Binyu Zang, and Haibo Chen. "Replication-Based Fault-Tolerance for Large-Scale Graph Processing". In: *IEEE Transactions on Parallel and Distributed Systems* 29.7 (July 2018). Conference Name: IEEE Transactions on Parallel and Distributed Systems, pp. 1621–1635. ISSN: 1558-2183. DOI: 10.1109/TPDS.2017.2703904.

[13] *Transient fault aware application partitioning computational offloading algorithm in microservices based mobile cloudlet networks | SpringerLink*. URL: https : / / link . springer . com / article / 10 . 1007 / s00607 − 019 − 00733 − 4 (visited on 01/25/2022).

[14] Pekka Karhula, Jan Janak, and Henning Schulzrinne. "Checkpointing and Migration of IoT Edge Functions". In: *Proceedings of the 2nd International Workshop on Edge Systems, Analytics and Networking*. EdgeSys '19. New York, NY, USA: Association for Computing Machinery, Mar. 2019, pp. 60–65. ISBN: 978-1-4503-6275-7. DOI: 10 . 1145 / 3301418 . 3313947. URL: https : //doi.org/10.1145/3301418.3313947 (visited on 01/25/2022).

[15] Yuli Tian, Jeff Tian, and Ning Li. "Cloud reliability and efficiency improvement via failure risk based proactive actions". en. In: *Journal of Systems and Software* 163 (May 2020), p. 110524. ISSN: 0164-1212. DOI: 10 . 1016 / j . jss . 2020 . 110524. URL: https : // www . sciencedirect . com / science / article / pii / S0164121220300078 (visited on 01/25/2022).

[16] Alexander Power and Gerald Kotonya. "A Microservices Architecture for Reactive and Proactive Fault Tolerance in IoT Systems". In: *2018 IEEE 19th International Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM)*. June 2018, pp. 588–599. DOI: 10.1109/WoWMoM.2018.8449789.

[17] Theodoros Theodoropoulos, Antonios Makris, John Violos, and Konstantinos Tserpes. "An automated pipeline for advanced fault tolerance in edge computing infrastructures". In: *Proceedings of the 2nd Workshop on Flexible Resource and Application Management on the Edge*. Minneapolis MN USA: ACM, July 2022.

[18] Firdose Saeik et al. "Task offloading in Edge and Cloud Computing: A survey on mathematical, artificial intelligence and control theory solutions". In: *Computer Networks* 195 (2021), p. 108177. ISSN: 1389-1286. DOI: `https : / / doi . org / 10 . 1016 / j . comnet . 2021 . 108177`. URL: `https : / / www . sciencedirect . com / science / article / pii / S1389128621002322`.

[19] Dimitrios Dechouniotis, Nikolaos Athanasopoulos, Aris Leivadeas, Nathalie Mitton, Raphael Jungers, and Symeon Papavassiliou. "Edge Computing Resource Allocation for Dynamic Networks: The DRUID-NET Vision and Perspective". In: *Sensors* 20.8 (2020). ISSN: 1424-8220. DOI: `10 . 3390 / s20082191`. URL: `https:// www.mdpi.com/1424-8220/20/8/2191`.

[20] Mohammad Masdari and Afsane Khoshnevis. "A survey and classification of the workload forecasting methods in cloud computing". en. In: *Cluster Computing* 23.4 (Dec. 2020), pp. 2399–2424. ISSN: 1573-7543. DOI: `10.1007/ s10586-019-03010-3`. URL: `https:// doi.org/10.1007/s10586-019-03010-3` (visited on 02/23/2021).

[21] Theodoros Theodoropoulos, Angelos-Christos Maroudis, John Violos, and Konstantinos Tserpes. "An Encoder-Decoder Deep Learning Approach for Multistep Service Traffic Prediction". In: *2021 IEEE Seventh International Conference on Big Data Computing Service and Applications (BigDataService)*. Aug. 2021, pp. 33–40. DOI: `10.1109/BigDataService52369.2021.00010`.

[22] Soma Prathibha. "Investigating the Performance of Machine Learning Algorithms for Improving Fault Tolerance for Large Scale Workflow Applications in Cloud Computing". In: *2019 International Conference on Computational Intelligence and Knowledge Economy (ICCIKE)*. Dec. 2019, pp. 187–190. DOI: `10.1109/ICCIKE47802.2019.9004379`.

[23] Tianqi Chen and Carlos Guestrin. "XGBoost: A Scalable Tree Boosting System". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Aug. 2016). arXiv: 1603.02754, pp. 785–794. DOI: `10 . 1145 / 2939672.2939785`. URL: `http://arxiv.org/abs/ 1603.02754` (visited on 05/28/2020).

[24] Didrik Nielsen. "Tree Boosting With XGBoost - Why Does XGBoost Win "Every" Machine Learning Competition?" eng. In: (2016). Accepted: 2017-03-13T07:58:50Z Publisher: NTNU. URL: `https :`

[25] John Violos, Evangelos Psomakelis, Konstantinos Tserpes, Fotis Aisopos, and Theodora Varvarigou. "Leveraging User Mobility and Mobile App Services Behavior for Optimal Edge Resource Utilization". In: *Proceedings of the International Conference on Omni-Layer Intelligent Systems*. COINS '19. New York, NY, USA: Association for Computing Ma- chinery, May 2019, pp. 7–12. ISBN: 978-1-4503- 6640-3. DOI: `10 . 1145 / 3312614 . 3312620`. URL: `https:// doi.org/10.1145/3312614.3312620` (visited on 02/20/2021).

[26] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Tobias Springenberg, Manuel Blum, and Frank Hutter. "Auto-sklearn: Efficient and Robust Automated Machine Learning". en. In: *Automated Machine Learning: Methods, Systems, Challenges*. The Springer Series on Challenges in Machine Learning. Cham: Springer International Publishing, 2019. ISBN: 978-3-030-05318-5. DOI: `10 . 1007 / 978 - 3 - 030 - 05318 - 5 _ 6`. URL: `https : / / doi . org / 10 . 1007 / 978 - 3 - 030 - 05318 - 5 _ 6` (visited on 02/20/2021).

[27] Elie Bursztein. *Cutting Edge TensorFlow - Keras Tuner: hypertuning for humans*. en. Conference Name: Google IO. May 2019. URL: `https : / / www . elie . net / talk / cutting - edge - tensorflow - keras - tuner - hypertuning - for - humans` (visited on 02/20/2021).

[28] Priti Kumari and Parmeet Kaur. "A survey of fault tolerance in cloud computing". en. In: *Journal of King Saud University - Computer and Information Sciences* 33.10 (Dec. 2021), pp. 1159–1176. ISSN: 1319-1578. DOI: `10 . 1016 / j . jksuci . 2018 . 09 . 21`. URL: `https : / / www . sciencedirect . com / science / article / pii / S1319157818306438` (visited on 01/26/2022).

[29] Ashkan Yousefpour, Genya Ishigaki, and Jason P. Jue. "Fog Computing: Towards Minimizing Delay in the Internet of Things". In: *2017 IEEE International Conference on Edge Computing (EDGE)*. 2017, pp. 17–24. DOI: `10.1109/IEEE.EDGE.2017.12`.

[30] Weiwei Lin, James Z. Wang, Chen Liang, and Deyu Qi. "A Threshold-based Dynamic Resource Allocation Scheme for Cloud Computing". en. In: *Procedia Engineering*. PEEA 2011 23 (Jan. 2011), pp. 695–703. ISSN: 1877-7058. DOI: `10 . 1016 / j . proeng . 2011 . 11 . 2568`. URL: `https : / / www . sciencedirect . com / science / article / pii / S1877705811054117` (visited on 01/30/2022).

[31] Marios Avgeris, Dimitrios Spatharakis, Dimitrios Dechouniotis, Aris Leivadeas, Vasileios Karyotis, and Symeon Papavassiliou. "ENERDGE: Distributed Energy-Aware Resource Allocation at the Edge". In: *Sensors* 22.2 (2022). ISSN:

// ntnuopen . ntnu . no / ntnu - xmlui / handle / 11250/2433761 (visited on 02/24/2021).

1424-8220. DOI: `10 . 3390 / s22020660`. URL: `https://www.mdpi.com/1424-8220/22/2/660`.

[32] *Automatically scaling pods with the horizontal pod autoscaler - Working with pods | Nodes | OpenShift Container Platform 4.7*. URL: `https : / / docs . openshift . com / container – platform / 4 . 7 / nodes / pods / nodes – pods – autoscaling . html` (visited on 01/30/2022).

[33] Fahd Al-Haidari, Mohammed Sqalli, and Khaled Salah. "Impact of CPU Utilization Thresholds and Scaling Size on Autoscaling Cloud Resources". In: vol. 2. Dec. 2013, pp. 256–261. DOI: `10 . 1109 / CloudCom.2013.142`.

[34] Jyostna Devi Bodapati, Nagur Shareef Shaik, Veeranjaneyulu Naralasetti, and Nirupama Bhat Mundukur. "Joint training of two-channel deep neural network for brain tumor classification". en. In: *Signal, Image and Video Processing* 15.4 (June 2021), pp. 753–760. ISSN: 1863-1711. DOI: `10 . 1007 / s11760 – 020 – 01793 – 2`. URL: `https : / / doi . org / 10 . 1007 / s11760 – 020 – 01793 – 2` (visited on 01/26/2022).

[35] Hui Wang, Jiang Lu, Lucy Nwosu, and Ishaq Unwala. "Two-channel convolutional neural network for facial expression recognition using facial parts". In: *International Journal of Big Data Intelligence* 6.3-4 (Jan. 2019). Publisher: Inderscience Publishers, pp. 259–268. ISSN: 2053-1389. DOI: `10 . 1504 / IJBDI . 2019 . 100897`. URL: `https : / / www . inderscienceonline . com / doi / abs / 10 . 1504 / IJBDI.2019.100897` (visited on 01/27/2022).

[36] Nikolaus Hansen, Dirk Arnold, and Anne Auger. *Evolution Strategies*. Jan. 2015. DOI: `10 . 1007 / 978 – 3 – 662 – 43505 – 2_44`.

[37] Peter I. Frazier. "A Tutorial on Bayesian Optimization". In: *arXiv:1807.02811 [cs, math, stat]* (July 2018). arXiv: 1807.02811. URL: `http : / / arxiv . org / abs / 1807 . 02811` (visited on 02/24/2021).

[38] Manoel C. Silva Filho, Raysa L. Oliveira, Claudio C. Monteiro, Pedro R. M. Inácio, and Mário M. Freire. "CloudSim Plus: A cloud computing simulation framework pursuing software engineering principles for improved modularity, extensibility and correctness". In: *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. May 2017, pp. 400–406. DOI: `10 . 23919 / INM . 2017.7987304`.

[39] Giampaolo Rodola. *giampaolo/psutil, https://github.com/giampaolo/psutil*. June 2020. URL: `https://github.com/giampaolo/psutil` (visited on 06/05/2020).

[40] Anders Krogh Mortensen. *anderskm/gputil, https://github.com/anderskm/gputil*. June 2020. URL: `https : / / github . com / anderskm / gputil` (visited on 06/05/2020).

[41] STsanakas. *STsanakas/Hybrid-Bayesian-Evolution-Strategy-GRU-RNN*. Feb. 2021. URL: `https : / / github . com / STsanakas / Hybrid – Bayesian – Evolution – Strategy – GRU – RNN` (visited on 02/24/2021).

[42] Tran Ngoc Minh, Lex Wolters, and Dick Epema. "A Realistic Integrated Model of Parallel System Workloads". In: *2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*. May 2010, pp. 464–473. DOI: `10 . 1109 / CCGRID . 2010.32`.

[43] Stephen Cohen and William Money. "Data Systems Fault Coping for Real-time Big Data Analytics Required Architectural Crucibles". In: Jan. 2017. DOI: `10.24251/HICSS.2017.121`.

[44] Salma M. A. Ataallah, Salwa M. Nassar, and Elsayed E. Hemayed. "Fault tolerance in cloud computing - survey". In: *2015 11th International Computer Engineering Conference (ICENCO)*. Dec. 2015, pp. 241–245. DOI: `10.1109/ICENCO.2015.7416355`.

[45] Fahd Alhaidari and Taghreed Zayed Balharith. "Enhanced Round-Robin Algorithm in the Cloud Computing Environment for Optimal Task Scheduling". en. In: *Computers* 10.5 (May 2021). Number: 5 Publisher: Multidisciplinary Digital Publishing Institute, p. 63. DOI: `10 . 3390 / computers10050063`. URL: `https://www.mdpi.com/2073–431X/10/5/63` (visited on 01/09/2022).

[46] Majid Derakhshan and Zohreh Bateni. "Optimization of tasks in cloud computing based on MAX-MIN, MIN-MIN and priority". In: *2018 4th International Conference on Web Research (ICWR)*. Apr. 2018, pp. 45–50. DOI: `10 . 1109 / ICWR . 2018 . 8387236`.

AUTHORS

Theodoros Theodoropoulos received his Electrical and Computer Engineering Integrated Master Degree in 2022, graduating from National Technical University of Athens. Currently he is Ph.d. student and researcher in the Department of Informatics and Telematics at Harokopio University of Athens. He has a total of nine publications on top-tier conferences and journals. He is involved in several Eu-funded research projects such as Charity, Teaching and Accordion. His research interests include Deep Learning, Graph Neural Networks, Deep Reinforcement Learning, Cloud and Edge Computing.
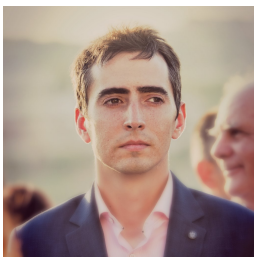
John Violos received the Diploma Degree in Electrical and Computer Engineering and the Dr. Engineering Degree from National Technical University of Athens (NTUA) in 2014 and in 2018. He was a member of the European Commission's Digital Single Market working group on the code of conduct for switching and porting data between cloud service providers. He has been involved in eight EU co-funded research projects. He ha more than thirty five publications and has served as guest editor, technical program committee member and reviewer in several conferences and research journals. His research interests include Deep Learning, Machine Learning, Edge Computing. He received the best paper award in IEEE iThings '21.

Stylianos Tsanakas received his Electrical and Computer Engineering Integrated Master Degree in June 2021, graduating from National Technical University of Athens. He currently works as a researcher for the Institute of Communication and Computer Systems (ICCS). He has a total of seven publications on top-tier conferences and journals. His professional interests include Deep Learning, Machine Learning, Edge Computing, Data Science.

Aris Leivadeas is currently an Associate Professor with the Dept. of Software and Information Technology Engineering at the Ecole de technologie Supérieure (ETS), Montreal, Canada. From 2015 to 2018 he was a postdoc in the Dept. of Systems and Computer Engineering, at Carleton University, Ottawa Canada. In parallel, Aris worked as an intern at Ericsson and then at Cisco in Ottawa, Canada. He received his diploma in Electrical and Computer Engineering from the University of Patras in 2008, the M.Sc. degree in Engineering from King's College London in 2009, and the Ph.D degree in Electrical and Computer Engineering from the National Technical University of Athens in 2015. His research interests include Edge Computing, IoT, and network automation and management. He received the best paper award in ACM ICPE'18 and IEEE iThings '21 and the best presentation award in IEEE HPSR'20.

Konstantinos Tserpes received the Ph.D. degree in the area of distributed systems from the School of Electrical and Computer Engineering, National Technical University of Athens, in 2008. He is currently an Assistant Professor with the Department of Informatics and Telematics, Harokopio University of Athens. He has been involved in several EU and National funded projects leading research for solving issues related to scalability, interoperability, fault tolerance, and extensibility in application domains, such as multimedia, e-governance, post-production, finance, and e-health. His research interests include distributed systems, software and service engineering, big data analytics, and social systems. He is a member of the Editorial Board of Future Generation Computer Systems.

Theodora Varvarigou is a professor at the School of Electrical and Computer Engineering of the National Technical University of Athens (NTUA). She received the B. Tech degree in Electrical Engineering from NTUA in 1988, the MS degrees in Electrical Engineering (1989) and in Computer Science (1991) from Stanford University, California. She received her Ph.D. degree from Stanford University as well in 1991. She has worked as a researcher at AT&T Bell Labs, USA and as an Assistant Professor at the Technical University of Crete, Chania, Greece. Prof. Varvarigou has great experience in cutting edge technologies, such as Cloud computing, multimedia content processing, semantic web, social networking technologies etc. She has published more than 200 papers in leading international journals, conferences and books. She has participated and co-ordinated numerous EC projects. From 2008-2012, she held the chair of the postgraduate program "Engineering Economic Systems" of NTUA.