A Zero-Shot Learning Approach for Task Allocation Optimization in Cyber-Physical Systems

Eliseu Pereira, João Reis, Rosaldo Rossetti, *Member, IEEE* and Gil Gonçalves, *Senior Member, IEEE*

***Abstract*—The design and reorganization of Cyber-Physical Systems (CPSs) faces challenges due to the growing number of interconnected devices. To effectively handle disruptions and improve performance, rapid CPS design and development is crucial. The Task Resources Estimator and Allocation Optimizer (TREAO) addresses these challenges, by simulating and optimizing the tasks assignment to the CPS machines, recommending suitable software layouts for the CPS characteristics. It employs Zero-Shot Learning (ZSL) to predict task requirements in heterogeneous devices, enabling the characterization of software pipeline execution in distributed systems. The Genetic Algorithm (GA) component then optimizes the task assignment across available machines. Through experiments, the tool is evaluated for task characterization, CPS modeling and optimization performance. TREAO, when compared with similar tools, allows the simulation of more resource usage metrics (CPU, RAM, processing time and network delay) and increases flexibility in heterogeneous CPSs by predicting the task execution behavior and optimizing the task assignment.**

***Index Terms*—Task Allocation, Zero-Shot Learning, Genetic Algorithms, Machine Learning**

I. Introduction

THE design of distributed systems, mainly Cyber-Physical Systems (CPS), has become crucial to obtain reliable, flexible, and robust digital environments [1]. The increasing number of interconnected devices in sectors like industry, agriculture, or transportation, raised the efforts associated with designing those systems. The design and specification of those systems requires several human resources and consumes large amounts of time, which in sectors such as the industry delays the stock production or the introduction of new products. The scenarios that can cause the system redesign or reconfiguration are almost infinite. Some might be due to 1) the components' failure and the introduction of new devices that require existing system reorganization to continue the process smoothly; or 2) the redefinition of the current process due to new system requirements. The operators and developers have limited time to mitigate those disruptions, so they use support tools to simulate the system behavior and obtain design recommendations.

Some tools help simulating large systems' behavior, finding the optimal assignment of software tasks, dimensioning the devices' capabilities, or searching for a faster network layout. Regarding those objectives, they must have two components: 1) the system simulator and 2) the optimization routine. The CPSs design, using simulation and optimization techniques, helps validating the CPSs before their deployment, identifying restrictions (network bottlenecks, machines in overload) and mitigating those situations, by suggesting new layouts or machine specifications. The system simulator enables the experimentation of different system setups producing output metrics that characterize the current experiment. However, most simulators only support the combinatorial testing of different configurations using entities (devices, tasks) with homogeneous characteristics [2]. The reason behind that limitation is the behavior modeling of each entity, which is a complex process that requires understanding the entity characteristics implemented using mathematical rules or object-oriented programming. The optimization routine explores an ample space of solutions and using the evaluation of the solution from the simulator, selects the optimal ones. Most tools do not implement the optimization component, contemplating only the system simulation, and those which do implement only use as input time behavior metrics discarding the resource usage.

The simulation and optimization of CPSs fits the task assignment problem, where a task pipeline representing the distributed software has each task associated with one machine. The task assignment simulation and optimization leads the designed CPSs to have more suitable hardware resources. Regarding the problem context, the Task Resources Estimator and Allocation Optimizer (TREAO) implements three components, 1) the task requirements estimator, 2) the task pipeline simulator, and 3) the optimization routine. The task requirements estimator predicts, using ZSL techniques, the task execution behavior in a machine without requiring task profiling or the manual behavior modeling. With the task requirements, the simulator converts the tasks pipeline into a graph representing the task sequence to extract some met-

Manuscript received Month xx, 2xxx; revised Month xx, xxxx; accepted Month x, xxxx. This work was partially supported by the HORIZON-CL4-2021-TWIN-TRANSITION-01 openZDM project, under Grant Agreement No. 101058673.

Eliseu Pereira, João Reis and Gil Gonçalves are with the SYSTEC-ARISE, Faculty of Engineering of the University of Porto, Rua Dr. Roberto Frias, 4200-465 Porto, Portugal (e-mail: eliseu@fe.up.pt, jpcreis@fe.up.pt and gil@fe.up.pt).

Rosaldo Rossetti is with the LIACC, Faculty of Engineering of the University of Porto, Rua Dr. Roberto Frias, 4200-465 Porto, Portugal (e-mail: rossetti@fe.up.pt).

rics that characterize the system behavior. The optimization component, using Genetic Algorithms (GAs), encodes into a chromosome the task-machine associations to explore different system layouts, producing the optimal solution. As result, TREAO main contributions are:

- Facilitation of the CPSs simulation, by not requiring the characterization of the task consumed resources (for executing in a particular machine) since the task requirements estimator predicts them.
- Optimization of the computational resources, avoiding over dimensioning hardware and reusing existent devices.

The reminder of this paper is organized as follows. Section II, presents the literature review. Section III provides the problem formulation. Section IV, focuses on the implementation of the task requirements estimator, task graph representation, and optimization algorithm. Section V presents the experiments and main results. Finally, Section VI presents the conclusions and future work.

II. Literature Review

The traditional solutions to design CPS base their suggestions on tools that emulate the system resources. Depending on the simulated variables (e.g., the processing and network delay or the memory utilization), the tools have the most distinguished applications. Some address the simulation of Cloud Systems, like the iFogSim [3] and the EdgeCloudSim [4], others enable the emulation of Fog Environments, like the FogTorchII [5], the EmuFog [6], the FogNetSim++ [7] and the YAFS [8], and others are directed for the simulation of CPS, like the SEED [9]. The suggestions are generally provided from optimization algorithms, based on metaheuristics methods, like Ant Colony Optimization, Simulated Annealing, or Genetic Algorithms (GAs) [10]–[12]. Those algorithms explore the space of configurations and designs, obtaining the evaluation metrics of each solution from the simulator.

The iFogSim [3] and EdgeCloudSim [4] simulators are extensions of the CloudSim simulator. The iFogSim [3] simulator provides quantitative measures for energy consumption, latency, and network delay. The tool allows the integration with external applications, with particular relevance for the usage of GAs [11] where the chromosome structure encodes the pair composed by the fog node and the associated service, corresponding each position in the chromosome to one service. The EdgeCloudSim [4] returns as output metrics the time behavior measures which fed the edge orchestrator module, deciding the node's workload and the link's placement in the network. The FogTorchII [5] supports most resource utilization metrics (disk, processor and memory usage). In this case, the user can define the infrastructure specifications and the application requirements (e.g., operating system, installed software) and then visualize the system performance throw memory utilization, disk usage, and quality of service assured. During the simulation process, the Monte Carlo simulation is used to validate different software application placements in the node's infrastructure. The main limitation is the hardness introduced when the user defines the application requirements. The FogNetSim++ [7] evolves from the OMNET++ [13] project; it supports various protocols (e.g., MQTT) and estimates time behavior metrics (processing and network delay). The simulator adopts a publisher-subscriber architecture, representing a more hierarchical architecture supporting heterogeneous fog nodes and a variable number of concurrent applications. The EmuFog [6] implements 1) the emulation of real applications and workloads and 2) the scalability of large topologies. Like other simulators, the EmuFog output estimates the time behavior metrics, actually the latency between nodes.

The Yet Another Fog Simulator (YAFS) [8] has as main features: 1) the simulated user mobility across the topology, allowing the fluctuation of the latency between nodes, according to the user distance to the gateway, 2) the dynamic failures of nodes in the network, and 3) the dynamic allocation of nodes. Worth highlighting is the capability to simulate the dynamic nodes' failure providing reliable topologies as output from the simulator. The assignment optimization's [14] primary goal is to increase the system availability and the quality of service. The Simulation EnvironmEnt Distributor (SEED) [9] uses a distributed architecture to decrease the simulation time, which partitions the CPS and associates each part with one machine.

Focusing on task assignment optimization to improve the system performance, several solutions adopt approaches based on metaheuristics algorithms. The Improved Genetic Algorithm [10] optimizes the energy consumption to particular quality of service requirements (time behavior metrics), using the fog node capabilities (processing, memory and network interfaces) and tasks computational requirements. The Dispersive Stable Task Scheduling (DATS) [15] optimizes the quality of experience using a computing resources competition algorithm. The node abstraction considers the heterogeneous types, and the metrics evaluated are the latency and the computation capacity. To optimize the resources, the system uses matching theory, namely many-to-one matching, which uses the node preferences to place the tasks according to that.

The described solutions are reliable in their outputs, helping designing systems. However, most simulators focus on time behavior metrics (e.g., processing or network delay), discarding resource utilization metrics and fault tolerance [16]. The typical architecture uses homogeneous nodes to set up the system simulation, limiting the system's flexibility in experimenting with different heterogeneous machines. The estimation of models that simulates the entities' behavior, like devices or tasks, is a complementary feature of the proposed tool that would enable the speed-up of the simulation setup due to avoiding the mathematical modeling of each entity. Regarding system design optimization, a few solutions implement that mechanism, used to obtain system layouts that optimize the quality of service or experience, or the task assignment.

III. Problem Formulation

The task assignment problem fits perfectly as a representation for the design and optimization of distributed CPSs. The problem has as its goal the optimization of the task assignment in the available machines of the CPS. So, the proposed solution must automatically find the optimal association between tasks

(t_i) and machines (m_i). Each task represents a software function to execute in a machine that is a CPS device with some computational power. The problem formulation and the solution definition use the notation described in Table I, describing each variable. The data structure used to organize the tasks is a directional graph ($G(T, E)$), where each vertex represents a task (t_i), and each edge (e_{t_i,t_j}) represents a dependency between tasks ($t_i \rightarrow t_j$). The graph's data structure represents the task sequence (e.g., $t_0 \rightarrow t_1 \rightarrow t_2$), mapping the existing pipeline workflow between tasks into a data structure. Each task (graph vertex) executes in an associated machine (m_i). The communication between machines (m_i, m_j) has delays (d_{m_i,m_j}), which affects the task pipeline performance if two consecutive tasks ($t_0 \rightarrow t_1$) perform in two different machines (t_0 in m_0 and t_1 in m_1); otherwise, the network delay between tasks in the same machine is zero.

TABLE I: Variables Description

Variable	Description	Domain
m_i	Machine i	Identifier (ID)
t_i	Task i	Identifier (ID)
e_{t_i,t_j}	Dependency edge from t_i to t_j	Identifier (ID)
cpu_{t_i,m_i}	CPU consumption distribution of t_i in m_i	$\mathcal{N}(\mu, \sigma^2)$
ram_{t_i,m_i}	RAM consumption distribution of t_i in m_i	$\mathcal{N}(\mu, \sigma^2)$
$time_{t_i,m_i}$	TIME consumption distribution of t_i in m_i	$\mathcal{N}(\mu, \sigma^2)$
d_{m_i,m_j}	Delay between m_i and m_j	Non-negative Float
$G(T, E)$	Graph representing the Workflow Pipeline	Graph

The decision of which machine executes each task depends on several factors, particularly the task computational costs, represented as requirements. The estimation of the task computational costs of its execution in a certain machine follows a normal distribution ($\mathcal{N}(\mu, \sigma^2)$), characterized by its mean (μ) and standard deviation (σ^2). So, for each task executing in a specific machine, there is a normal distribution that allows the estimation of the consumption of RAM (ram_{t_i,m_i}), CPU (cpu_{t_i,m_i}), and processing time ($time_{t_i m_i}$). The RAM is measured in megabytes (MB); the CPU is measured in percentage usage; and the total execution time of one task is measured in seconds. Using a probabilistic distribution adds the stochastic factor to the modeling of the environment, enabling a trustful representation of real CPSs. Those variables are the basis to model the functions to quantify the pipeline consumed resources and actual execution status, detailed in Section IV-B. Then the goal is to minimize $Z = f(C)$, where $C = \{(t_0, m_0), ..., (t_k, m_k)\}$. The summation of the pipeline characterization functions results in the fitness function to minimize (reducing the consumed resources), detailed in Section IV-C. The fitness function considers the entire pipeline costs which increases the complexity of the task assignment problem, leading to the usage of metaheuristics optimization techniques. The optimization algorithm considers that all tasks can be assigned to every machine, without existing hardware dependencies.

IV. Methodologies

The process of task assignment estimation in a distributed heterogeneous CPS passes through a set of steps (presented in Figure 1) to obtain, as output, the optimal assignment of all tasks. The Task Resources Estimator and Allocation Optimizer (TREAO)[1] contains two main components: 1) the simulator capable of predicting the CPS resources' consumption given one possible assignment of tasks (Figure 1a, 1b and 1c), and 2) the optimizer responsible for evaluating different task assignments to minimize the simulator estimated consumption costs (Figure 1d).

The simulator to predict the consumed resources by the tasks pipeline uses two main components, 1) the Hyper-Model (HM), which estimates the task requirements distribution ($\mathcal{N}(\mu, \sigma^2)$) in a particular machine using as input the task static metrics (profiling/machine independent) and the machine specifications and 2) the graph model representation that transforms the task pipeline into a graph data structure enabling the extraction of resource usage metrics. The optimization strategy uses Genetic Algorithms (GAs), an evolutionary computation method capable of obtaining optimal solutions in a large environment.

A. Hyper-Model Task Requirements Estimation

The task requirements ($ram_{t_i m_i}$, $cpu_{t_i m_i}$, $time_{t_i m_i}$) estimation typically demands the task execution in each machine type to model each probability distribution. The execution of each task in every machine implies several costs, particularly human efforts to access each machine, execute every task and collect the profiling results. Additionally, each task should be executed several times and left executing for a significant amount of time. Thus, estimating task requirements without involving the profiling process efforts is advantageous and accelerates the data-driven simulation setup. The Hyper-Model (HM) [17] is a Zero-Shot Learning algorithm considered in the literature as a model of models, where the main goal is to predict the parameters of new models based on existing ones. For the TREAO, the main goal of the HM is to predict the distribution parameters (mean and standard deviation) that characterize the task consumption requirements for new tasks and machines. The analysis of each task uses a machine-independent profiling technique, which provides code analytic metrics, like the cyclomatic complexity or the Halstead metrics. The static code analysis provides quantitative indicators of how complex and resource-consuming one task is without requiring task execution.

The cyclomatic complexity corresponds to the total number of decision blocks in the task source code. It's computation uses abstract syntax trees to represent the task source code using a tree data structure. They represent the existent decision blocks as nodes (e.g., if-conditions, for loops, etc.), then the cyclomatic complexity computation algorithm counts the number of instructions accounted by it to calculate the metric itself. The Halstead metrics provide an alternative source of information about the state of the task source code. Those

[1] https://github.com/DIGI2-FEUP/TREAO

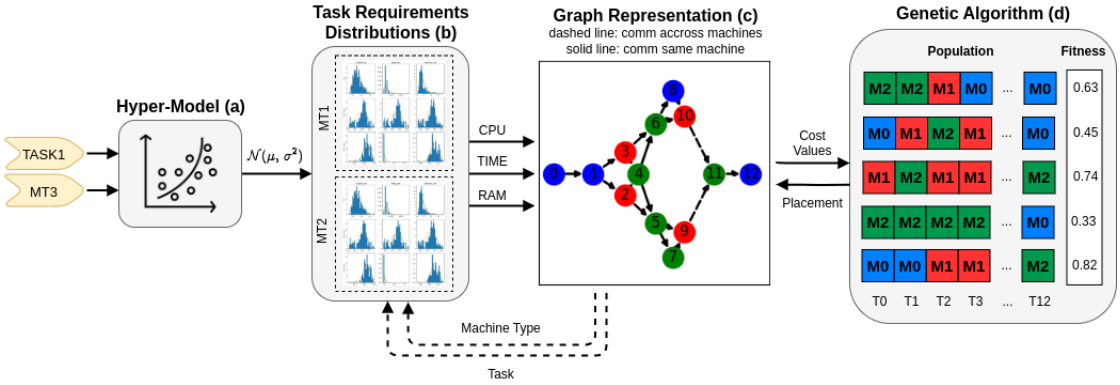


Fig. 1: TREAO Data Architecture.

metrics contemplate the operators (logical, rational, or arithmetic) and their operands (raw values); both entities provide information about the distinct types of operators (n_1) and operands (n_2) and the total instances of operators (N_1) and operands (N_2). The sum of the distinct types of operators and operands provides the variability of the task source code, i.e., the program vocabulary ($n = n_1 + n_2$). In parallel, the sum of the total number of instances of operators and operands results in the program length ($N = N_1 + N_2$). Both the program vocabulary and length are arguments of the Halstead volume providing as output the program volume reflecting the trade-off between the vocabulary and the length of the program ($volume = N \log_2 n$). Besides the static code analysis metrics, the function calls provide information about the total number of external methods executed, including built-in functions. The function calls describe the runtime behavior of the task independently of the hosting machine, which complements the static metrics.

The HM algorithm uses as input the described metrics to predict as output the values of mean and standard deviation that model the task consumption requirements probability distribution. Regression methods fit perfectly as a prediction model due to the continuous values of the output targets. The used tasks for the use case scenario are six sorting algorithms (bubble, quick, heap, selection, insertion, and merge sort) executed in two different machines, which generate 12 probability distributions to use as target variables (mean and standard deviation). The input features were the offline task characterization metrics (13 variables: cyclomatic complexity, maintainability index, logical lines of code, source lines of code, difficulty ($\frac{n_1}{2} \cdot \frac{N_2}{n_2}$), effort (product between difficulty and volume), program length, program vocabulary, volume, number of delivered bugs, calculated program length, number of function calls, and time required to program) and the machine specification (3 variables: total RAM, CPU frequency, and CPU delay). Combined, the task characterization and the machine specification metrics generate a dataset of 16 input features, which is of high complexity regarding the available data points (12 data samples). As a way to perform feature selection, the analysis of variance (ANOVA) comparing the feature distributions and the examination of the Linear Regression coefficients selecting the ones with higher value enable the extraction of the five most relevant and distinct features reducing the feature space. The five resulting features were: 1) the cyclomatic complexity, 2) the effort and 3) the number of function calls as task characterization metrics and 4) the RAM, and 5) CPU rate considering the machine specifications. Training and evaluating the regression model (HM) requires partitioning the dataset, using the strategy leave-one-out cross-validation. Since six different tasks execute in two machines, the total number of samples to train and test is 12.

The regression model correlates the input features with the target variables, enabling the prediction of the consumption requirements of new tasks. Each task consumption metric (RAM, CPU, and processing time) uses a different instance of the regression model. In summary, each consumption metric has a regression model associated, which uses as input the selected features to predict the mean and standard deviation of its distribution. A large pool of regression methods (e.g. Elastic-Net, Linear, Lasso, or Ridge Regression) was tested, using as performance metrics the coefficient of determination (R-squared) and the mean square error (MSE). Based on the results presented in Section V, the selected technique was the Ridge Regression which has more accurate predictions and higher performance on the evaluation metrics. The Ridge Regression hyperparameters (alpha) were tuned, obtaining the best results with 0.1 as alpha.

B. Graph Cost Functions

The proposed graph approach models the pipeline defined by the user and enables the extraction of cost functions, quantifying the efforts to execute each task pipeline. The primary cost function is the longest path between the start and last tasks of the pipeline. The longest path is an excellent metric because it indicates the worst execution time of the entire pipeline. Usually, each pipeline has many parallel routes between the start vertex to the end one, and the path with the higher total of edge weights marks the execution time of the pipeline. According to graph theory, the longest path between two vertices is the path that has the highest network cost (total of edge weights). However, for the task assignment problem, it is essential to consider also the processing time cost of each vertex (task) to have a more accurate value of the pipeline processing time. Regarding the longest path estimation, the

algorithm searches for all possible paths (P) between the start vertex (t_0) and the end vertex (t_N), for each path calculates the pipeline processing time cost ($p_{\max}$), which is composed of the tasks processing time ($time_{t_i m_i}$) and the delay between machines ($d_{m_i,m_{i+1}}$). Then, from all the possible routes, the algorithm selects the one with the maximum cost (p_{max}), as follows the first part of Equation 1.

$$p_{\max} = \arg\max_{p \in P}\left(\sum_{i}^{|p|} time_{t_i,m_i} + \sum_{i}^{|p|-1} d_{m_i,m_{i+1}}\right)$$
$$dmc = \sum_{j}^{AM}\sum_{k \neq j}^{AM} |mc_j - mc_k| \quad (1)$$
$$mc_j = \frac{\sum_i^T cpu_{t_i,m_j}}{m\_cpu_{m_j}} + \frac{\sum_i^T ram_{t_i,m_j}}{m\_ram_{m_j}} + \frac{\sum_i^T time_{t_i,m_j}}{m\_time_{m_j}}$$

Additionally, ensuring the appropriate assignment of tasks across the machines is fundamental to securing device operation. For that, it is imperative to estimate each machine's cost, given by the second part of Equation 1, which depends on the machine specifications and the tasks associated with each machine. The user is responsible for defining the machine's capabilities in terms of RAM ($m\_ram_{m_j}$), CPU ($m\_cpu_{m_j}$), and processing time ($m\_time_{m_j}$). So, the machine costs (mc_j) are the quotient between the total cost of the tasks (executing in that particular machine) and their machine capabilities. Additionally, the machine cost balance (dmc) measures the difference between all the available machines (AM). This way, if the pipeline has the tasks uniformly placed across the devices, the result of the function should be close to zero.

The compression of several tasks in organized clusters of machines avoids delays in the communications. A cluster consists of a group of linked tasks mapped in the same device. Regarding that, the ideal number of clusters is one per machine, which means every machine has at least one task associated and avoids separate clusters/groups of tasks associated with the same device. The metric that measures that behavior is the cluster difference (cd) that computes the absolute value of the difference between one and the actual number of clusters per machine.

Furthermore, other cost functions allow global comprehension of the CPS actual state. Between them are the mean RAM usage (μ_{ram}) and the mean CPU usage (μ_{cpu}) by each machine, which enables the user to select from a solution that consumes less RAM or CPU. Plus, the task assignment problem is essential to guarantee minimal network traffic and minimize the network infrastructure requirements. So, the cost function evaluates the network state using the delay between machines (d_{m_i,m_j}) if there is an edge (e_{t_i,t_j}) in the graph between different devices. The collection of the network delays converges into a mean value (μ_d) that is directly proportional to the network traffic. Figure 1c reflects the network delay, represented as dashed lines between tasks executing in different machines.

C. Optimization Algorithm

Optimization algorithms based on metaheuristics allow for exploiting an ample space of solutions with minimal computation efforts. GAs are an excellent option for the problem of different task assignments in the available machines due to the combinatorial characteristics of the problem. The adoption of simple GAs is due to them being straightforward to integrate with the task assignment problem. However, since TREAO implementation is modular, the GA can be easily replaced by other optimization algorithm such as multi-island genetic algorithms, as described in Section VI. As mentioned before, the task allocation/assignment problem consists of matching each software module in a machine. For TREAO, a simulation is characterized by a particular pipeline as a graph data structure, where each task is a vertex, and the dependency between tasks is an edge. The GA interprets the entire graph as a chromosome (Table II) where each gene position (array index) corresponds to a task and its content to the machine associated. So, the chromosome length will be variable according to the number of tasks to be allocated. That approach allows the combination of different pairs of tasks (t_i) and machines (m_j) to obtain the optimal solution. Figure 1d presents the optimization process, showing examples of tasks ($t_{0:12}$) associated to machines ($m_{0:2}$), where each association is an individual of the population with the respective fitness value.

TABLE II: Chromosome Gene Structure

Task	t_0	t_1	t_2	t_3	t_4	...	t_N
Machine	m_0	m_2	m_1	m_0	m_1	...	m_0

The GA implements a sequence of five steps: 1) creation of an initial population, 2) calculation of the fitness value of each individual, 3) selection of the best individuals in the population, 4) from the selected portion (parents), select a few ones to crossover and generate a new part of the population (children), and 5) pick another few parents to mutate and append to the rest of the population. Remark that some steps are performed in a loop due to the generations required to converge to an optimal solution. So, in the setup stage, the method creates the original population, calculates each fitness, and selects the stronger ones to pass over to the next generation. After that stage, the algorithm performs in a cycle, where, from the previous generation, selects a few individuals to crossover (children) and mutate. The strongest individuals from that pool (parents, children, and mutations) jump to the next generation.

The crossover step uses a uniform operation, randomly picking half of the genes from each parent and joining them into a new individual. That operation allows the inheritance of the strongest individual genes for the new individuals of the current generation. The mutation of the individuals uses two different mutation operations, the swap mutation and the random resetting. The random resetting mutation selects randomly one gene to mutate and maps its content to another machine; that operator increases the variability in the new solutions. The swap mutation picks two different genes and

swaps their content; that operation allows the modification of the actual assignment of machines without unbalancing the costs between devices, which reflects in the fitness value.

The fitness function (f), in Equation 2, reflects the assignment cost of a given task graph in particular machines due to the algorithm's goal to minimize the fitness function output. The fitness function returns the addition of the cost functions: 1) the longest path ($p_{\max}$), 2) the cost difference between machines (dmc), 3) the mean RAM (μ_{ram}) and CPU (μ_{cpu}) usage, 4) the mean delay in the network connections (μ_d), and 5) the difference between the number of clusters (cd).

$$f = p_{\max} + dmc + cd + \mu_{cpu} + \mu_{ram} + \mu_d \qquad 2$$

Each metric has its order of magnitude, which induce bias into the fitness function, giving more relevance to the metric with a higher order of magnitude. Due to different orders of magnitude, each metric passes through a normalization process to obtain an equilibrium between the metrics' weights. The normalization process divides each metric by its order of magnitude, e.g., for $\mu_{ram} = 19.8$, the normalized value is $\mu_{ram} = 0.198$. Also, the user must have preferences for solutions, e.g., that consume more RAM or have more balanced costs between machines. So, each metric multiplies with a coefficient representing the user preferences. The GA hyperparameters were tuned including the size of the initial population, the size of a generation, the number of select individuals (parents), and the number of mutations.

V. Experiments & Results

The system evaluation addresses several components: 1) task requirements estimations, 2) task graph modeling, 3) GA performance, and 4) a comparison with similar tools. Both task distribution requirements and their predictions must illustrate the actual resources consumption per task according to the host machine's specifications. The graph that models the task pipeline must reflect the actual state of the system, and the metrics comparison between different setups allow the validation of their precision. The evaluation of the optimization strategy verifies if the algorithm output corresponds to the optimal solution. Additionally, the optimization algorithm passes through a workload validation, evaluating how the algorithm performance evolves with the increasing complexity of the tasks graph. Those experiments used two machines, 1) a laptop with 16GB of RAM and an i7 with 12 cores 2.60GHz processor (*UBUNTU_PC*), and 2) a raspberry pi 3 model B (*RASP_PI*).

A. Task Characterization Evaluation

Each task requirement distribution comes from previous executions of the task (task profiling) or from HM estimations. Regarding the different procedures to obtain the task requirements, Figure 2a compares the task profiling values using the physical equipment (green bars) and the HM estimations (blue bars) using three different metrics as columns and two machine types as rows. Each bar corresponds to the mean value of the distribution, and the standard deviation appears as the bar error. Each task corresponds to a sorting algorithm, which facilitates its theoretical characterization to validate the practical results.

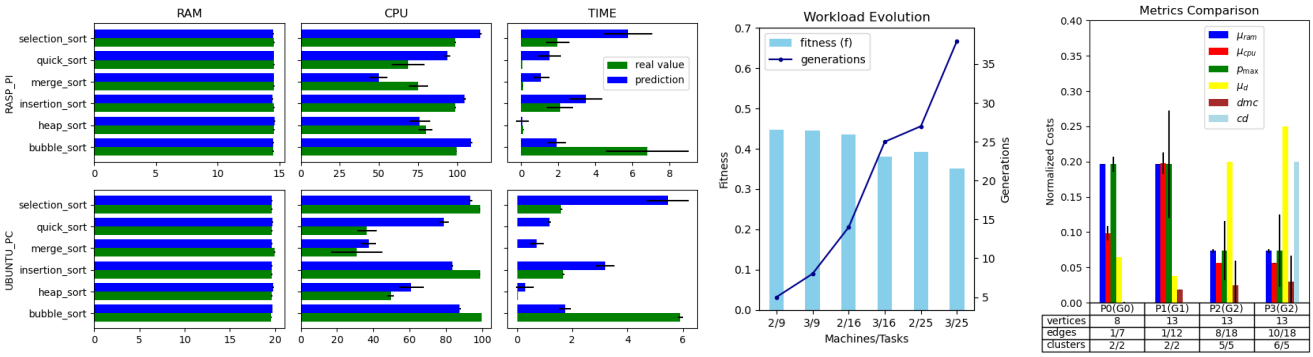
The profiling values (green bars) must reflect the theoretical computational cost of the task, in particular, the time and memory complexity. Focusing on those indicators, tasks characterized by complexity functions with higher gradients theoretically perform worst than tasks with lower gradient complexity functions. The profiling values show that relation, e.g., by comparing the time complexity function of the heap sort ($n \log n$) with the bubble sort (n^2); in theory, the computational cost should be higher for the bubble sort than the heap sort. The profiling values correspond to that relation, where, for the *UBUNTU_PC* machine type, the bubble sort consumes 99.6% of CPU usage and 5.9 seconds of processing time, and the heap sort retains 50% of CPU usage and 0.03 seconds of processing time. Concerning the RAM usage, due to the same values of memory complexity (constant) of the algorithms, the experimental consumptions are pretty similar, approximately 14.5 MB and 19.6 MB for each machine type, respectively.

The HM learning process, as mentioned before, uses five tasks for training and one task for testing, repeating for each task to have one prediction for each one. The selection process of the regression model to adopt as a HM, evaluates different techniques (Linear Regression, Lasso Regression, Ridge Regression, and Elastic-Net), choosing the one with better results (R-squared and MSE). The selected algorithm was the Ridge Regression, which has the more accurate predictions, with lower values of MSE, i.e., 0.006, 178.42, and 3.52 for the RAM, CPU, and processing time models, respectively. The comparison between predictions and profiling/actual values in Figure 2a demonstrates the accurate estimations for the RAM and CPU consumption metrics, with slightly worst results for the processing time. Due to the generalization nature of those models, for some extreme values (e.g., quick sort algorithm on the UBUNTU_PC machine type) the predictions are not as accurate as for the rest. Usually, there should be a relationship between CPU usage and processing time; in some algorithms (e.g., bubble sort), there is no relation between them due to hidden factors in the implementation.

B. Graph Modeling Validation

The graph representation validation aims to prove that the simulated metrics reflect the system's state described in the input parameters. Its validation uses different pipelines of tasks to compare the outputs of the simulation component with the expected results, using the profiling values and the HM predictions as task requirements. Figure 2c reflects the simulator response to three different setups of pipelines (G0, G1, and G2), where the configuration G2 has two different assignments validated (P2 and P3). Each bar reflects the output of the cost functions (described in Equation 2) using as task requirements the real profiling data. The error in each bar indicates the difference between the usage of real profiling data and consumption predictions. The table below Figure 2c characterizes each pipeline task, where 1) the number of vertices indicates the number of tasks in the pipeline, 2) the edges row registers the task connections between two different

This article has been accepted for publication in IEEE Transactions on Industrial Cyber-Physical Systems. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TICPS.2024.3392151



(a) HM Predictions Comparison (b) GA Workload Evolution (c) Graph Cost Estimations

Fig. 2: Results obtained in the different components.

machines versus the total number of task connections, and 3) the clusters row indicates the number of available machines in contrast with the total number of clusters.

As for the analysis of component outputs, Figure 2c reflects a correlation between the characterization of the tasks pipeline and the cost function output values. Regarding the CPU and RAM usage, the comparison between pipeline P1, with 13 tasks and two machines available, and P2, with the same number of tasks but five machines available, indicate a decrease in both costs of CPU and RAM usage, which is expected due to the increasing number of devices to execute the same number of tasks. The network traffic and cluster differential cost functions are reliable indicators of the actual task assignment and system communications. Concerning that, the network traffic has a direct correlation with the number of connections between different machines, e.g., the increasing traffic costs between pipelines P2 and P3, where the number of external connections goes from 8 to 10. The cluster differential cost function reflects the number of groups of linked tasks mapped to the same machine; the results indicate that for the first three pipelines, the cost is zero due to the presence of one cluster per machine. When the number of groups increases from more than one per machine, the cost function follows that trend increasing its output value.

C. Optimization Validation & Workload

The optimization algorithm evaluation validates the method convergence and the time required to converge, i.e. how the performance (e.g., number of generations) fluctuates according to the total number of combinations, which depends on the available machines and the number of graph' vertices (tasks). Figure 2b compares the fitness (light blue bars), with the number of generations (dark blue line) required to obtain that fitness value. The complexity of the initial setup (number of available machines and the total number of tasks), on the x-axis, in Figure 2b, increases along the axis; that trend reflects on the required number of generations to obtain a target value of fitness. For a constant value of fitness, the number of generations required to converge increases according to the growing number of available machines and tasks.

In terms of performance, TREAO presents a time complexity of $O(gpc)$, where g is the number of generations, p the population size and c the size of the chromosome (number of tasks). Since, the number of generation and the population size can be fixed values, the time complexity presents a linear behavior, that increases with the number of tasks to optimize.

D. Comparison with State of the Art

Comparing TREAO with similar tools indicates a reliable alternative for the CPS design, enabling simulation and optimization of resources. The performed comparison, in Table III, is qualitative, mainly due to the difference in the simulated and optimized metrics and resources. Besides TREAO, the simulator that fills the most similar gap is the YAFS; however, it focus on simulating homogeneous devices and in optimizing the CPS infrastructure, suggesting new specifications for the existing machines. The simulation of heterogeneous types of equipment is an advantage for TREAO, when compared to other solutions. Additionally, the output metrics provide a vision of resource utilization, which is complementary to the given metrics from the other simulators. The tests indicate an accurate simulation and optimization of the resources.

TABLE III: Tools Comparison

Tool	Metrics	Optimization
EdgeCloudSim [4]	$time$, d	Infrastructure
FogTorchII [5]	ram, d	Infrastructure
iFogSim [3]	cpu, ram, $time$	Infrastructure
EmuFog [6]	d	Infrastructure
FogNetSim++ [7]	$time$, d	-
YAFS [8], [14]	$time$, d	Infrastructure
SEED [9]	$time$, d	-
TREAO	cpu, ram, $time$, d	Task Assigment

VI. Conclusions & Future Work

TREAO consists of a valuable tool to support the CPS design, particularly regarding the optimization of the task assignment, without requiring manual modeling of the task behavior due to the usage of the Zero-Shot Learning algorithm. The TREAO architecture consists of three components, the task requirements estimator, the task graph simulator, and the optimization algorithm. The task requirements estimator uses the task characterization metrics (code analysis) and the

machine specifications to predict the execution consumption requirements of a task in a particular machine. The task graph simulator characterizes the task and machine resources enabling the quantification (using metrics) of the input pipeline with a particular machines' assignment. Those metrics reflect the state of the network communications, the resources consumed in each device, the distribution of tasks between the machines, and the system's performance. The optimization component uses GAs to explore different combinations of assignments in such a way that finds the assignment with an optimal cost value. The algorithm converges to the optimal solution according to the outputs received from the simulation system. The results validate both the simulation output metrics and optimization solutions, proving the advantages of that tool.

Concerning future work, the main objective is to integrate that tool with a platform for the development and management of CPS. The tool integration will enable a performance optimization option in the platform to obtain improved results with the same hardware resources, using the simulation component to explore the solution before deploying it. Additionally, an essential point of improvement will pass through experiment other optimization algorithms, such as the multi-island genetic algorithm, in order to accelerate the convergence time.

REFERENCES

[1] P. Hu, S. Dhelim, H. Ning, and T. Qiu, "Survey on fog computing: architecture, key technologies, applications and open issues," vol. 98, DOI https://doi.org/10.1016/j.jnca.2017.09.002, pp. 27–42, 2017.

[2] S. Svorobej, P. Takako Endo, M. Bendechache, C. Filelis-Papadopoulos, K. M. Giannoutakis, G. A. Gravvanis, D. Tzovaras, J. Byrne, and T. Lynn, "Simulating fog and edge computing scenarios: An overview and research challenges," *Future Internet*, vol. 11, DOI 10.3390/fi11030055, no. 3, 2019.

[3] R. Mahmud, S. Pallewatta, M. Goudarzi, and R. Buyya, "ifogsim2: An extended ifogsim simulator for mobility, clustering, and microservice management in edge and fog computing environments," *Journal of Systems and Software*, vol. 190, DOI https://doi.org/10.1016/j.jss.2022.111351, p. 111351, 2022.

[4] R. Freymann, J. Shi, J.-J. Chen, and K.-H. Chen, "Renovation of edgecloudsim: An efficient discrete-event approach," in *2021 Sixth International Conference on Fog and Mobile Edge Computing (FMEC)*, DOI 10.1109/FMEC54266.2021.9732572, pp. 1–8, 2021.

[5] A. Brogi and S. Forti, "Qos-aware deployment of iot applications through the fog," *IEEE Internet of Things Journal*, vol. 4, DOI 10.1109/JIOT.2017.2701408, no. 5, pp. 1185–1192, Oct. 2017.

[6] R. Mayer, L. Graser, H. Gupta, E. Saurez, and U. Ramachandran, "Emufog: Extensible and scalable emulation of large-scale fog computing infrastructures," in *2017 IEEE Fog World Congress (FWC)*, DOI 10.1109/FWC.2017.8368525, pp. 1–6, Oct. 2017.

[7] T. Qayyum, A. W. Malik, M. A. Khan Khattak, O. Khalid, and S. U. Khan, "Fognetsim++: A toolkit for modeling and simulation of distributed fog environment," *IEEE Access*, vol. 6, DOI 10.1109/ACCESS.2018.2877696, pp. 63 570–63 583, 2018.

[8] I. Lera, C. Guerrero, and C. Juiz, "Yafs: A simulator for iot scenarios in fog computing," *IEEE Access*, vol. 7, DOI 10.1109/ACCESS.2019.2927895, pp. 91 745–91 758, 2019.

[9] P. Garraghan, D. McKee, X. Ouyang, D. Webster, and J. Xu, "Seed: A scalable approach for cyber-physical system simulation," *IEEE Transactions on Services Computing*, vol. 9, DOI 10.1109/TSC.2015.2491287, no. 2, pp. 199–212, Mar. 2016.

[11] A. J. Page, T. M. Keane, and T. J. Naughton, "Multi-heuristic dynamic task allocation using genetic algorithms in a heterogeneous distributed system," *Journal of Parallel and Distributed Computing*, vol. 70, DOI https://doi.org/10.1016/j.jpdc.2010.03.011, no. 7, pp. 758 – 766, 2010.

[10] X. Li, Y. Liu, H. Ji, H. Zhang, and V. C. M. Leung, "Optimizing resources allocation for fog computing-based internet of things networks," *IEEE Access*, vol. 7, DOI 10.1109/ACCESS.2019.2917557, pp. 64 907–64 922, 2019.

[12] O. Skarlat, M. Nardelli, S. Schulte, M. Borkowski, and P. Leitner, "Optimized iot service placement in the fog," *Service Oriented Computing and Applications*, vol. 11, DOI 10.1007/s11761-017-0219-8, no. 4, pp. 427–443, Dec. 2017.

[13] A. Varga, *A Practical Introduction to the OMNeT++ Simulation Framework*, pp. 3–51. Cham: Springer International Publishing, 2019. [Online]. Available: https://doi.org/10.1007/978-3-030-12842-5_1

[14] I. Lera, C. Guerrero, and C. Juiz, "Availability-aware service placement policy in fog computing based on graph partitions," *IEEE Internet of Things Journal*, vol. 6, DOI 10.1109/JIOT.2018.2889511, no. 2, pp. 3641–3651, Apr. 2019.

[15] Z. Liu, X. Yang, Y. Yang, K. Wang, and G. Mao, "Dats: Dispersive stable task scheduling in heterogeneous fog networks," *IEEE Internet of Things Journal*, vol. 6, DOI 10.1109/JIOT.2018.2884720, no. 2, pp. 3423–3436, Apr. 2019.

[16] M. Ashouri, F. Lorig, P. Davidsson, and R. Spalazzese, "Edge computing simulators for iot system design: An analysis of qualities and metrics," *Future Internet*, vol. 11, DOI 10.3390/fi11110235, p. 235, 11 2019.

[17] J. Pollak and N. Link, "From models to hyper-models of physical objects and industrial processes," in *2016 12th IEEE International Symposium on Electronics and Telecommunications (ISETC)*, pp. 317–320, 2016.

Eliseu Pereira is a researcher at the Research Center for Systems & Technologies (SYSTEC-ARISE) hosted in the Faculty of Engineering, University of Porto (FEUP). He received an M.Sc. in Electrical and Computers Engineering degree from FEUP in 2018 and is a Ph.D. student in Informatics Engineering. He is currently involved in several projects related to Smart Manufacturing, applying concepts of reconfiguration and data analytics to CPSs.

João Reis is currently Data Science Lead at DEUS company, a role that he has been gladly taking for the past year and a half. Before that, he was mostly an academic with close to 10 years of research at University of Porto, specifically at DIGI2 Lab at SYSTEC-ARISE. Within this period, he has participated in close to 20 National and European projects focusing on Smart Manufacturing, specifically applying machine learning and smart sensing solutions in industry.

Rosaldo J. F. Rossetti has a PhD degree in computer science from INF-UFRGS. He is FEUP's COO with ARMIS.Lab@FEUP, focused on applying multi-agent systems to address issues in artificial transportation systems. He served as a member of the Board of Governors of the IEEE ITS Society and of the Steering Committee of the IEEE Smart Cities Initiative. He is the Co-Chair of IEEE ITS Society's Artificial Transportation Systems.

Gil Gonçalves has a PhD in Electrical and Computer Engineering, focused on the analysis, specification and implementation of complex systems with adaptive capabilities. Gil is an Assistant Professor at the Faculty of Engineering of the University of Porto (FEUP) and Coordinator of the DIGITAL and InTelligent Industry laboratory (DIGI2). Gil has been involved in over 35 National and European RTD projects to promote digital transformation.