Noname manuscript No.
(will be inserted by the editor)

Intelligent Tasks Allocation at the Edge based on Machine Learning and Bio-Inspired Algorithms

Madalena Soula · Anna Karanika · Kostas Kolomvatsos · Christos Anagnostopoulos · George Stamoulis

Received: date / Accepted: date

Abstract Current advances in the Internet of Things (IoT) and Cloud involve the presence of an additional layer between them acting as mediator for data transfer and processing in close distance to end users. This mediator is the Edge Computing (EC) infrastructure. In EC, we can identify an ecosystem of heterogeneous nodes capable of interacting with IoT devices, collecting and locally processing the data they report. The ultimate goal is to eliminate the latency we face when relying on Cloud to perform the desired processing activities. In EC, any processing is performed over a number of geo-distributed datasets formulated by the collected data that exhibit specific statistical characteristics. Processing can have the form of tasks requested by end users or applications. It becomes obvious that in the EC ecosystem, we have to carefully decide the EC nodes that will host and execute any requested task. In this paper, we extend our previous research efforts on the conclusion of efficient task allocations into the available EC nodes. We go a step forward and propose a batch processing model executed over multiple tasks and study two allocation models: a scheme based on an unsupervised machine learning technique and a bio-inspired optimization algorithm. Our models enhance the autonomous behavior of entities performing the envisioned task allocations. We provide the analytical description of the problem, our solution and the advances

Madalena Soula
Department of Electrical and Computer Engineering, University of Thessaly, email: msoula@e-ce.uth.gr

Anna Karanika
Department of Electrical and Computer Engineering, University of Thessaly, email: ankaranika@e-ce.uth.gr

Kostas Kolomvatsos
Department of Informatics and Telecommunications, University of Thessaly, email: kostasks@uth.gr

Christos Anagnostopoulos
School of Computing Science, University of Glasgow email: christos.anagnostopoulos@glasgow.ac.uk

George Stamoulis
Department of Electrical and Computer Engineering, University of Thessaly, email: georges@e-ce.uth.gr

over the state of the art. We present and evaluate the proposed algorithms and compare them with other efforts in the domain. The pros and cons of our models are revealed through the provided extensive experimental evaluation adopting real and synthetic data.

Keywords Edge Computing · Edge Mesh · Internet of Things · Tasks Scheduling · Tasks Allocation · Machine Learning · Swarm Intelligence

1 Introduction

The new era of the *Internet of Things* (IoT) incorporates numerous devices capable of interacting with their environment and performing simple processing activities to serve end users. IoT devices can record ambient data and transfer them in an 'upwards mode' to the Edge, the Fog and the Cloud infrastructures. These concrete 'layers' can be the host of multiple processing activities (i.e., tasks) based on which the available data can be processed towards the provision of knowledge and analytics. In general, the processing at Cloud can be realized in a batch processing manner over humongous volumes of data. However, the transfer of data to Cloud and the increased waiting time for getting the final response (i.e., the outcome of the desired processing activities) negatively affects the latency in the provision of the final results [4]. This is one of the reasons made the research community to focus on data processing at the edge infrastructure through the emergence of *Edge Computing* (EC). EC mainly focuses on streams processing in a number of EC nodes to limit the latency and succeed the immediate delivery of analytics to any requestor. Compared to Cloud, EC nodes are characterized by lower computational resources and heterogeneity.

EC nodes are geo-distributed and become the host of data reported by IoT devices. They act as mediators between the IoT infrastructure and Cloud. No matter the limited resources of EC nodes, they can store and process locally a (sub-)set of the collected data while the remaining are sent to Cloud. This means that at the edge of the network, we can detect an ecosystem of EC nodes placed at different locations hosting geo-distributed datasets. It becomes obvious that the discussed datasets can be the subject for the execution of various tasks (e.g., queries, simple processing tasks, etc) defined by end users or applications. When such a setting is the case, the most significant research question is to provide an efficient methodology for allocating the incoming tasks to specific EC nodes instead of broadcasting them to the entire ecosystem. A broadcasting action will 'flood' the network with useless messages as only a sub-set of EC nodes could be capable of responding to the requested tasks. Suppose there is a task asking the statistics of humidity measurements between 10 and 20 for a specific area. It is useless to broadcast this task to all the available EC nodes 'forcing' them to deliver their results. The first reason is that only EC nodes located in the desired area should respond to the task aligned with the desired spatial requirements. In addition, from the sub-set of EC nodes placed in the specific area, only nodes having in their datasets humidity values in the interval [10, 20] can efficiently respond while the remaining will spend resources to return an empty set. The allocation of the desired tasks only to EC nodes that 'match' to their requirements saves resources and time involving only those that can return an efficient response

based on the statistics of their datasets. Further, it is important to introduce the EC nodes' performance into the decision making process. In fact, resources are scarce as well as heterogeneous in EC nodes. Therefore, taking into consideration the performance of nodes can greatly improve the allocation.

In this paper, we envision an entity called *Task Controller* (TC) being responsible to receive tasks and apply a scheme for allocating them to the appropriate EC nodes. As an allocation, we define the optimal selection of an EC node for a distinct task determined by the task's characteristics and EC nodes' current performance. The TC can be present at Cloud or at a 'sink' node being also responsible to aggregate the results after the allocation of the task and formulate the final response for the requestor. TCs have insights on EC nodes status, i.e., the data they possess as well as historical values related to their performance. For limiting the congestion of the network, nodes can relay statistical synopses to the TC [36] to depictthe underlying datasets. TCs are the realization of an 'evolving system' that is capable of adapting its behaviour to the dynamically changing requirements of the incoming tasks and the performance of the underlying EC nodes. They should exhibit learning capabilities and self-organization behaviour that make them capable of being aligned with the needs of the requestors. We assume that multiple TCs are present at various locations having interactions with multiple EC nodes. This creates a very dynamic and unknown environment concerning the behaviour of all the involved 'actors'. Every TC decides upon an unknown and unpredictable ecosystem of EC nodes that are the owners of geo-distributed datasets fed by data reported through the interaction with IoT devices. We have to notice that TCs receive tasks for execution from multiple other entities imposing additional requirements for their effective functioning. The challenge is to provide to TCs the necessary intelligence to be able to efficiently allocate a batch of tasks to the available nodes instead of adopting a sequential approach, i.e., the allocation of an individual task at a time. The batch oriented approach is expected to assist in increasing the throughput of TCs giving them the ability to serve numerous tasks upon numerous EC nodes. Previous research has revealed that batch processing outperforms the 'immediate' allocation (the allocation of individual tasks when they arrive at TCs) when the number of tasks is high [50].

In this paper, we extend our previous research efforts in the domain and provide a novel mechanism that assists TCs in their decision making (i.e., the allocation process). Such a decision-making is dynamically influenced by the continuously updated datasets and EC nodes' performance. We consider that TCs process the incoming tasks in batches, i.e., *they allocate multiple tasks to multiple EC nodes at once*. We propose two allocation schemes: (i) the first is based on *Machine Leaning* (ML) and more specifically on a clustering process; (ii) the second adopts a bio-inspired model and more specifically a variant of the widely known *Particle Swarm Optimization* (PSO) algorithm. Both models aim at detecting the optimal allocations at a specific time instance. We rely on techniques that do not require any training to reduce the cost of 'preparing' them before they are adopted in the envisioned decision making activities. We also incorporate into our decision process the *Cost of Allocation* (CoA) as defined in our previous effort [26] to depict the cost for each allocation based on tasks' requirements and EC nodes current status. The CoA is also adopted to be an indicator for realizing EC nodes rankings creating a 'rewarding' mechanism for solving the discussed problem. However, our current work differs with [26] in the following aspects: (i) In [26], we focus on

'one-to-one' allocations meaning that we allocate a single task at each EC node. In this paper, the proposed models can manage a high number of tasks that could be greater than the number of the available EC nodes; (ii) In this paper, we adopt a ML and an optimization technique while in [26], we are based on the solution of the known assignment problem (i.e., a minimum cost flow problem). Finally, additional efforts in the past [33], [30], [34], [35] focus on the processing/allocation of a single task while our current work deals with the management of a batch of tasks.

Comparing our research with other similar efforts in the domain, we can discern that this paper aims to contribute in the respective research proposing an approach for tasks management at the EC unlike other studies which propose the transfer of tasks to the Cloud prior to processing them. Again, the adopted batch oriented management differentiates our models and adds novelty to the approach. In addition, for deciding allocations, we pay attention not only on tasks' data requirements compared to the available datasets but also on their priorities and the performance of EC nodes (concerning their ability to quickly return the final response). Our target is to handle the dynamics of the environment where TCs and EC nodes act being aligned with all the aforementioned parameters (i.e., tasks and nodes characteristics). Based on our approach, we avoid broadcasting activities related to the allocation of tasks that may flood the network with unnecessary messages and uselessly spend the resources of nodes while pointing the appropriate hosts for every processing activity.

The remaining paper is organized as follows. Section 2 discusses the related work while Section 3 presents preliminary information and the problem under consideration. Section 4 depicts the proposed solution and in Section 5, we provide our experimental evaluation results adopting real and synthetic datasets. Finally, in Section 6, we conclude our paper by presenting our future research plans in the domain.

2 Prior Work

2.1 Independent Edge

The development of the large-scale Internet of Things (IoT) has led to the generation of enormous amounts of data. There are studies which propose real-time processing by deploying edge servers taking into consideration the traffic and the wireless diversity which define IoT networks [66]. It should be highlighted that the users of mobile devices and applications are increasing by thousands in a daily basis leading to the need of infrastructures that will process data in the most efficient way without transferring them to Cloud. The respective research suggests Mobile Edge Computing (MEC) which aim to handle data streams at the mobile edge [57]. Practically, EC comes into scene to 'cover' the disadvantages of Cloud, especially to limit the increased latency in the delivery of results when requestors demand various processing activities. There are many scenarios where EC can facilitate the desired processing especially when no connection is feasible with the Cloud back end. EC offers a huge infrastructure which allows the execution of processing activities close to the source of data instead of delivering the information to data centers/Cloud [27]. The EC allows nodes to respond to service demands

locally, reducing the consumption of bandwidth and the latency of the network [65]. EC lies in the middle between Cloud and the IoT infrastructure where numerous devices interact with each other to communicate data. EC nodes, before data are transferred to Cloud, can be the host of a virtual or physical data separation process, i.e., they can be part of an ecosystem where distributed datasets are formulated through the communication with the IoT devices. IoT data can be formulated in any format (heterogeneity) and their amount becomes humongous as the reporting rate is high. Data should be stored in the appropriate format and their volumes should meet EC nodes' storage capabilities. Recall that EC nodes are diverse ranging from routers and gateways to small servers, thus, their computational capabilities differ. Evidently, tasks or queries are requests for information, usually issued by human users, including simple or even complex transactions aiming to either read or update the collected data [13]. Tasks can have any form that the application domain dictates. For instance, if we consider queries formulated by SQL commands, EC nodes should be capable of detecting the parts and constraints of these queries and, accordingly, of interacting with the local data storage mechanism (e.g., the RDBMS or any other model) to retrieve the corresponding results. These processing outcomes are communicated back to the requestor while the execution process can be centralized (e.g., in the same machine) or distributed [10]. A future challenge of EC is to provide nodes capable of understanding tasks requirements before starting their execution. In addition, EC nodes should be also capable of understanding their internal status and the status of the environment before starting any processing activity to be able to select the optimal line of actions (this is the so-called *stateful EC*).

2.2 Data Partitioning

In EC and Cloud, data partitioning can be the result of the physical allocation of various nodes connected with reporting devices. We can design our network in a way to have EC nodes hosting the data reported by specific IoT devices in close distance with them. For instance, suppose there is a smart city where nodes are placed at various locations to interact with users' smart devices (e.g., in a mall). Additionally, data partitioning can be adopted, on purpose, to increase the parallelism in the execution of tasks, thus, to facilitate manageability and control. This approach requires the design of efficient algorithms to separate data that will deliver the final outcomes in the minimum time. A survey on data partitioning algorithms can be found in [52]. The majority of the discussed efforts is coming from the database community while gaining increased attention in the community of the management of large scale data. There are two models for data partitioning, i.e., (i) 'batch-oriented' processing: data are, firstly, collected and, then, are separated into a number of partitions [52]; (ii) 'stream-oriented' processing: as data arrive at 'collection end points', an algorithm decides the partition where data will be finally stored [29]. In the former model, we have plenty of room to pre-process the data and eliminate any heterogeneity, missing values and so on and so forth. However, due to the huge volumes of data, the former model, naturally, requires increased computational resources. The latter model should take decisions on the fly, i.e., for data 'pre-processing' and conclude their placement to the appropriate partition. In any case, the adopted algorithm should be efficient enough to detect and resolve

the present heterogeneity, missing values and any other pre-processing activity in real time. It becomes obvious that data heterogeneity, at least concerning their format, imposes an additional burden when we want to execute analytics over the incoming streams [8]. The former model better 'matches' to a Cloud environment while the latter is aligned with the needs of EC and IoT.

2.3 ML for Large-Scale Data Management

In research activities adopted for the management of tasks, ML can play a significant role to address not only the increasing amount of data but also the distributed management of the available datasets. In [1], the authors present a comparative analysis of exploratory techniques for large scale data. A framework adopted for the analysis of sensory recordings targeting the minimization of the amount of data transferred in the network is discussed in [22]. Such an analysis aims at supporting analytics used for decision making in various application domains. Analytics refer to various tasks aiming to reveal knowledge and the hidden aspects of the distribution of data. Example applications of analytics are provided by the anomaly detection research domain; an overview of the relevant approaches is provided in [48]. A 'special' type of analytics are prognostics or predictive analytics [6]. Predictive analytics deal with the detection of the distribution of data and calculations around specific key performance indicators to reveal their future trends. Hence, decisions can be made not only based on current trends but also on future estimations of data. In [11], the authors report the effect that the quality of data has on the outcomes of processes targeting on predictive analytics and propose a method for improving it. The proposed method targets to the maintenance domain where we have to manage a set of devices and production equipments. In [6], the authors propose the SERENA system comprising of a number of services, which collectively provide predictive analytics enabling the application of predictive maintenance policies. Other example analytics models are Prism [9], CONTROL [20] and DBO [23] systems. All these systems aim at saving time in the provision of responses adopting sampling or progressive analytics. Usually, the discussed systems are built upon the MapReduce model (e.g., [12], [42]).

2.4 Task Allocation at the Edge

The task scheduling/allocation problem attracts the attention of scientists working in the EC domain. The domain is in its infancy bearing in mind that the allocation of tasks should take into consideration the 'special' conditions/requirements that should be met at the edge infrastructure (e.g., heterogeneity of devices/data/protocols, huge volumes of data reported by IoT devices, limited latency and support of real time applications) [28]. In [17], the authors investigate the task completion time minimization in multiuser mobile EC networks. Multiple users can offload tasks simultaneously, thus, an efficient management mechanism should be delivered. The authors adopt a partial offloading model, i.e., each user can partition its computation tasks into offloading computing and locally computing parts. In [41], to eliminate the long-term sum cost which includes the power consumption and the

task execution latency, the authors propose the adoption of channel conditions between end devices and the gateway, the computation task queue and the remaining computational resources of devices. The problem is modelled as a Markov Decision Process (MDP) and solved through the use of a reinforcement learning model. In [61], a device-to-device model is studied targeting to assist users to offload tasks to multiple nodes. Afterwards, results are retrieved over orthogonal pre-scheduled time slots. The subject of [63] is the modelling of the task completion delay in mobile EC environments and the analysis of the energy consumption of different equipments. A placement method is proposed to allocate each requested task to the available cloudlets and the public Cloud with the minimum total energy consumption. In [44], the authors study the elimination of the cloudlets for all the task requests to be allocated and provisioned. Requests scheduling is also the subject of [45]. The allocation problem is modelled as a variant of K-facility location problem. The authors propose an approximation scheme and an online algorithm to deal with how to schedule requests when cloudlets are placed. In [45], the authors consider the delay of transmitting tasks. Another effort in this domain studies the allocation of a number of edge servers based on an exact integer linear programming solution [60]. The aim is the minimization of the total access delay which is proportional to the distance between the base station and the edge server. The authors of [62] propose an approximation algorithm to solve the cloudlet placement problem while in [16], the authors try to jointly minimize the costs and the total end-to-end transmission delay based, again, in a linear programming model. In [39], the authors consider a heterogeneous mobile cloud computing (HMCC) approach that consists of remote Cloud servers, local cloudlets, task offloading mobile devices (TMDs) and non-task offloading MDs (NTMDs). TMDs have the capability of offloading tasks to Cloud servers or cloudlets. The authors adopt stochastic geometry and calculate the outage probability to support offloading decisions. Another model for offloading tasks from the network to the EC is discussed in [21]. The model is based on a sum cost delay scheme and deal with the optimal binary computational offloading decision. Reinforcement learning is adopted to conclude the final allocation. The authors of [18] model the offloading problem as a double auction game and analyze the Bayes-Nash Equilibrium (BNE). A Stackelberg game is adopted to model the interactions between nodes that desire to exchange tasks and conclude the optimal allocation of resources. The proposed model deals with tasks related to a blockchain scenario. Multiagent systems are also adopted for providing solutions in the management of tasks offloading [49]. The authors detect the role of the multiagent system in the mapping between three decision tables towards the optimization of the problem. The proposed model relies on tasks' priority, the load caused in the network and resources availability. In [40], the authors present a mechanism for the allocation of data collection tasks. The correlation between tasks and the available sensors is modelled, then, a double selecting strategy is adopted to conclude the best node and sensor network that fulfils the desired quality of data and collection time constraints for each task. In [2], the authors focus on a MEC scenario and propose a model for tasks offloading based on a minority game combined with a distributed learning algorithm. Every node adopts the proposed approach to declare if it is active or not and mobile devices rely on a stochastic learning automata scheme to distributively select the appropriate active peer. The authors of [3] present a non-cooperative game between users and conclude the Pure Nash Equilibrium (PNE) related to the opti-

mal data offloading. The performance of the proposed framework are demonstrated through modelling and simulation. Another work relying on the combination of game theory and machine learning (i.e., reinforcement learning) is presented in [53]. The aim is to model the distributed resource management in MEC for computation offloading. The proposed game theoretical scheme is dedicated to deal with the activation of energy-efficient distributed edge servers. The problem of the distributed activation is also studied in [54]. The authors solve it through the use of minority games guaranteeing energy-efficient activation together with the satisfaction of users as far as the quality-of-experience concerns. Deep learning is also adopted for the tasks offloading problem [32]. Advanced neural networks can undertake the responsibility of learning the characteristics of data that represent the optimal line of actions. However, the training dataset should efficiently expose any aspect of the allocation process upon multiple parameters in order to conclude the best possible model.

Recently, the problem of tasks allocation under uncertainty has attracted significant attention especially in the robotics domain. The proposed algorithms take into consideration the characteristics of robots as well as the special requirements of tasks under the restrictions of the application domain. Uncertainty can limit the performance as it can result in inefficient allocations. The 'allocator' cannot be certain about the hidden aspects of the matching between tasks and 'executors'. Usually, the proposed models incorporate uncertainty to account for the performance of robots localisation and navigation reliability [43], [59]. One type of uncertainty is the knowledge about the environment. Multiple efforts assume that the environment is fully observable; however, this is not always the case. Hence, the problem of robots navigation and tasks execution can be modelled by incorporating a probabilistic decision over the available actions and formulate a planning problem with the use of MDPs [38], [56]. For producing probabilistically-guaranteed behaviour policies, we can adopt verification-based methods where elements of an MDP are learnt from experience [37].

2.5 Positioning to the Respective Literature

As exposed above, there are multiple research activities dealing with the problem of tasks allocation/offloading. The research community has adopted many theories and technologies to solve the problem and deliver efficient models that will take the corresponding decisions upon a high number of parameters. Models dealing with the partial offloading of tasks incorporate an increased complexity in the decision making that is imposed by the splitting of each task into parts and aggregate the partial responses. Additionally, the aforementioned complexity is dictated by the need of the dynamic detection of data that are relevant to every part of the initial task. Increased complexity is also observed when advanced ML models are utilized in the decision making. This is more intense when supervised methods are the case, i.e., there is the need for the presence of the appropriate training dataset that covers all the aspects of the problem. Such models require time for the training process and a retraining phase when the decision strategy is altered. Game theoretic approaches may impose a high number of interactions between various entities present in the infrastructure, thus, they are characterized by an increased time to deliver the final outcome. Furthermore, the game based decision

making may mandate the knowledge of various parameters that could not be the usual case in real setups. Finally, only a few research efforts try to combine a high number of parameters in the decision making as the solution space is heavily expanded and, as natural, the the complexity of the approach is increased.

3 Preliminaries

3.1 The Envisioned Setting

For the formulation of our problem, we borrow the basic notation adopted in our previous work [26]. We consider a set of N EC nodes (see Figure 1 [26]), i.e., $\mathcal{E} = \{e_1, e_2, \ldots, e_N\}$ being 'connected' with a number of IoT devices. Every node becomes the host of data reported by the devices, thus, it becomes the owner of a local dataset upon which the desired processing activities are executed. IoT devices report multidimensional data, i.e., $\mathbf{x} = \langle x_1, x_2, \ldots, x_M \rangle$ where M is the number of dimensions. EC nodes should adopt the appropriate mechanisms to store data locally and execute tasks/queries demanding for analytics. Due to resources constraints, EC nodes cannot host all the reported data, thus, they select the vectors that will be the subject of local storage transferring the remaining data to Cloud. Without loss of generality, we consider that EC nodes store only 'fresh' data and adopt a sliding window approach. The window size depends on nodes' storage capabilities. The obsolete data (data reported in a time instance out of the adopted window) are evicted (sent, in an upwards mode, to Cloud) as new data vectors arrive.

Without loss of generality, from this point forward, when we refer in tasks, we consider tasks or queries asking for data processing. The local execution of tasks is performed by a dedicated component, i.e., the *Tasks Execution Component* (TEC). The performance of this component depends on the computational capabilities of each EC node. Recall that the nodes are 'characterized' by diversity in their capabilities, e.g., we can meet routers, gateways or small servers bounded all together to cooperatively execute tasks and exchange data. TECs adopt specific plans to execute the incoming tasks. For instance, if we focus on queries execution, TECs can rely on query execution plans as adopted in known database management systems. In that case, TECs should 'communicate' and interact with the underlying database management system to efficiently conclude the appropriate responses. If we focus on 'generic' tasks, TECs could adopt any known task scheduling and execution algorithm. Our model is not bounded to any specific algorithm. The study of the internal process of TECs is beyond the scope of this paper. Formally, each node e_j is 'characterized' by a specific processing speed $s_j, j = 1, 2, \ldots, N$ defined in the unity interval that depicts the rate at which data and tasks are processed. $s_j \to 1$ means that the jth TEC is characterized by the maximum possible speed (e.g., a node with increased computational capabilities).

We also consider that in front of each TEC, there is a first come first served queue where the incoming tasks are placed to wait for their execution. The size of the queue is depicted by the number of tasks that the queue can host. The size of the queue and the throughput of the corresponding TEC affect the current load of each EC node. The load is a significant parameter as it can depict the 'smooth' execution of future tasks. For instance, if a TEC is overloaded (i.e., the

number of the tasks waiting in the queue is close to the maximum queue size), the node will conclude the results with an increased latency (especially for tasks relying at the end of the queue). This can negatively affect the performance of the applications waiting for the corresponding responses (especially, if they are time sensitive applications). Without loss of generality, we consider that there is an upper bound for the number of tasks that can be placed in the discussed queues. In our analysis, we adopt the parameter l_j that depicts the jth TEC's load, i.e., the amount of tasks waiting for execution in the jth queue. As we adopt an upper bound for each queue, l_j can be defined in the unity interval, i.e., $l_j \in [0, 1]$. $l_j \to 1$ means that the jth TEC/queue is overloaded and cannot host additional tasks.

We focus on the discrete time $\mathbf{T}$ and consider that at each time instance $t \in \mathbf{T}$, a new task ϵ_t arrives at a TC. ϵ_t has specific characteristics and constraints. This information is 'exposed' through the metadata accompanied ϵ_t. For instance, if we focus on queries, constraints can be defined in the 'Where' clause. For generic tasks, constraints can be defined in the form of requirements for their execution, e.g., the software that should be adopted, the criticality and so on and so forth. In this paper, we consider that constraints refer in data over which the processing should be performed. For instance, we can have a task asking for analytics only for data present in a specific range (e.g., temperature in [10,30] and humidity in [80,100]), i.e., a 'type' of a range query. Easily, we can conclude the complexity of each task, i.e., c_{ϵ_t}, based on the complexity of the required calculations. For that, we can rely on the algorithmic theory or other techniques like the one presented in [33]. In addition, tasks' constraints can impose a deadline representing the upper bound of time for delivering the final results, i.e., τ_{ϵ_t}. Any application can define τ_{ϵ_t} according to its criticality. For instance, if we focus on the provision of real time services, τ_{ϵ_t} will be short ($\tau_{\epsilon_t} \to 0$).

Based on the above description, when focusing on tasks allocation delivered by TCs, we have to 'match' c_{ϵ_t} and τ_{ϵ_t} against l_j and s_j. TCs should take into consideration these four parameters, matching them in pairs, i.e., (c_{ϵ_t}, l_j) and (τ_{ϵ_t}, s_j) to realize the most efficient allocations. The term 'efficient' represents allocations that will 'secure' the smooth and immediate execution of tasks limiting the latency in the provision of results. Our mechanism supports the decision making of TCs and acts proactively trying to detect the most efficient allocations. Recall that TCs serve multiple users/applications, thus, they should manage numerous tasks. Our previous efforts [33], [30], [35] propose models for allocating a single task to an individual EC node, i.e., the final outcome is a pair in the form ¡task-EC node¿. In this paper, we depart from our previous models and propose a scheme for processing a batch of tasks. The proposed scheme builds over a window W defining the number of tasks that will be the subject of our allocation mechanism. We consider $f(\cdot)$, i.e., a function that gets W tasks and the information for the available EC nodes and results the final allocations. For every task $\epsilon_t^i, i = 1, 2, \ldots, W$, we deliver the EC node where it should be allocated, i.e., $f(\epsilon_t^i, \mathcal{E}) \to e_{selected}$ ($e_{selected}$ is the selected node for ϵ_t^i). There is a high number of combinations between W tasks and N nodes, thus, a 'brute-force' approach is not the optimal solution to get (near) real-time results. We propose the use of two models, i.e., an unsupervised ML model and a *Computational Intelligence* (CI) scheme. More specifically, we propose the use of a clustering process and a *Swarm Intelligence* (SI) approach over tasks and EC nodes characteristics. The final result is the mappings for each of the W tasks to one of the N available EC nodes.

3.2 Problem Definition

The envisioned allocation process is performed upon the ecosystem of EC nodes exhibiting specific characteristics that should be combined with the requirements of the incoming tasks. The aforementioned allocation function $f(\cdot)$ is, actually, repeatedly evaluated for each of the W tasks. Our problem is a simple optimization process upon $f(\cdot)$ that gets the requirements of the incoming tasks and try to find the optimal solution. The optimization problem can be formulated as follows:

$$\begin{aligned} maximize \quad & f\left(c_{\epsilon_t}, \tau_{\epsilon_t}\right) \\ \text{subject to} \, & \left\{l_j, s_j\right\}, \forall j \\ & \left\{\epsilon_t\right\}, t = 1, 2, \ldots, W \end{aligned} \tag{1}$$

The problem discussed in this paper can be categorized as an assignment problem. The assignment problem is the problem of mapping each of n tasks to one of the available EC nodes in the most efficient manner. The total cost equals the sum of the partial costs of every individual allocation. Our work focuses on allocation techniques which build on top of queries' characteristics, the state and the features of the system. The objective of such algorithms is the achievement of an optimal tasks allocation which leads to the minimization of execution and communication costs, load balancing among the nodes and, generally, to the efficient usage of the system resources. In our discussion, communication costs are considered negligible, thus, they are omitted in our analysis.

Every TC, having the information about the available EC nodes and the incoming tasks, adopts a simple rewarding mechanism to elaborate on the matching between pairs c_{ϵ_t} - l_j and τ_{ϵ_t} - s_j. More details can be found in [26]. The rewarding mechanism aims at 'supporting' the allocation of complex tasks to EC nodes characterized by a low load towards the minimization of the *initiation time*. As the initiation time, we define the time required to start the execution of a task. The rationale is simple; an EC node with a low load can quickly devote the required resources for the execution of tasks. Moreover, when τ_{ϵ_t} is short, we prefer to perform the allocation to a node exhibiting a high speed to conclude the execution as soon as possible. In any case, both characteristics should be 'combined' to support an efficient allocation. For each pair, we conclude the *Cost of Allocation* (CoA). The CoA consists of two parts; it is equals to the sum of partial CoAs (one for each pair of characteristics). Initially, the TC should calculate the *rounded ratio* ρ of c_{ϵ_t} compared to τ_{ϵ_t}, i.e., $\rho = \frac{c_{\epsilon_t}}{\tau_{\epsilon_t}}$. It is important to notice that before the conclusion of ρ, c_{ϵ_t} is delivered as the number of the required steps to execute a task as proposed in [33]. ρ represents the speed required by a task (in execution steps per time unit) to be concluded in accordance with τ_{ϵ_t}. Subsequently, s_j is compared to ρ to detect whether the EC node is fast enough to serve ϵ_t in the required time interval. Then, l_j is compared to pre-defined thresholds indicating the 'range' where it is realized. For instance, we can adopt a 'high' and a 'low' thresholds, thus, l_j can be 'placed' into a number of ratio intervals (i.e., below the lowest threshold, between thresholds, above the highest threshold). We aim to support three intervals/levels for l_j; low, medium, high. For instance, if the load is over 0.80, we consider a high CoA especially for tasks exhibiting a high complexity. Actually, we propose a fuzzy number to depict the cost for pairing l_j with c_{ϵ_t}. For each interval/level, we adopt a constant value for the corresponding

cost and for each combination of pairs c_{ϵ_t} - l_j and τ_{ϵ_t} - s_j, our mechanism defines the appropriate reward or penalty attributed to the CoA. After the conclusion of the CoA, we apply the envisioned models (i.e., the clustering scheme and the SI model). Hence, the problem is transformed to the problem of deciding the final allocations over the CoAs.

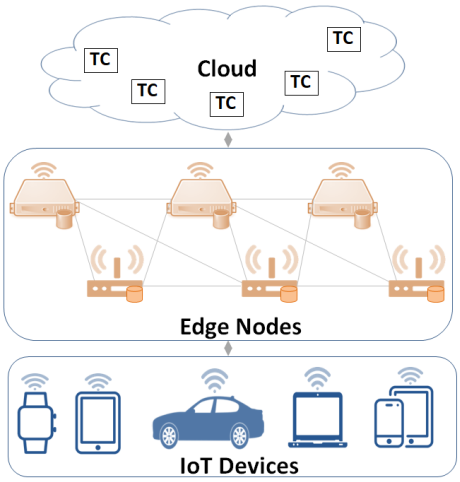


Fig. 1: The connection of task controllers and edge nodes.

4 Tasks Management based on ML and CI

4.1 Clustering of EC nodes & Tasks

We propose and apply a prioritization scheme of the incoming tasks, based on [58]. As TCs receive new tasks, they detect the available EC nodes, i.e., nodes with l_j below a pre-defined threshold. If no EC node meets the condition, the incoming tasks are placed in another queue at TCs. This additional queue is sampled to collect randomly some tasks and examine again the availability of EC nodes. Recall that EC nodes exhibit a very dynamic behaviour concerning the conclusion and the allocation of tasks, thus, their availability varies. In any case, our algorithm adopts a priority model to immediately serve tasks with a high priority. High priority tasks will be firstly allocated in the available EC nodes. The priority of a task ϵ_t is calculated through the adoption of the following equation: $p_{\epsilon_t} = \tau_{\epsilon_t} + \frac{i}{max(i)}$ where i is the index of the incoming task when it arrives at the TC and τ_{ϵ_t} is the deadline that the user/application have specified for the task. Therefore, the priority of a task is closely connected to the deadline. The priority is inverse to the p_{ϵ_t} value, i.e., $p_{\epsilon_t} \to 0$ represents the highest priority. Tasks are sorted in an ascending order of p_{ϵ_t} and the TC processes W of them. To avoid the 'starvation' problem (a task exhibits a low priority for a long time), we apply an ageing mechanism through the

second term in the equation. Based on this mechanism, after a pre-defined interval, we increase the priority of tasks waiting for a long time in the corresponding queue.

Clustering is a widely adopted unsupervised ML algorithm aiming at grouping objects exhibiting similar characteristics. The clustering approach is applied into our scenario as follows. We consider a number of 'virtual' clusters and their centroids represented by the tasks that the TC wants to allocate. *A virtual cluster is a set of EC nodes having the minimum 'distance' δ with the centroid, i.e., a task.* This means that every task 'attracts' the set of EC nodes that exhibit the maximum similarity becoming candidates to host and execute it. A 'free' EC node is the node that is not yet assigned to a cluster, thus, the EC node is not assigned as a candidate to execute a task. A 'candidate' is an EC node which is part of a cluster, thus, it is a candidate to be assigned the task representing the centroid of the specific cluster. δ is the difference between EC nodes characteristics' and tasks' requirements. Our algorithm defines a cluster centroid for every task and populates the corresponding cluster with free EC nodes, rendering them candidates. The selection of candidates is determined by a heuristic realized by the parameter R_j defined for every EC node as follows: $R_j = \frac{\delta_j}{s_j}$.

We consider that the k EC nodes with the minimum R_j are defined as candidates for a specific cluster. The EC node that will, finally, execute the task is selected among the candidate nodes. Cluster members are sorted in an ascending order based on the CoA and the node with the minimum execution cost is selected to 'host' the task being the centroid of the corresponding cluster. For realizing δ_j, we adopt an additional heuristic function, i.e., $\delta_j = \tau_{\epsilon_t} \frac{1-l_j-c_{\epsilon_t}}{s_j}$. Considering that l_j is the load of the node, $1-l_j$ is the degree to which the specific node can accept new tasks. Additionally, R_j is defined in a way that serves our purpose to assign a task to the appropriate node. We refer to the node which has the highest load while still being able to host it. This means that a node with virtually no free space for new tasks is preferred. At the same time, the speed of the appropriate node should be high. As $\delta_j \to 0$ and $s \to 1$, then $R \to 0$ which is desired, since it leads to optimized task allocation. In Algorithm 1, we present the proposed algorithm based on the aforementioned process.

4.2 Bio-inspired Tasks Allocation

Particle Swarm Optimization (PSO) [15] consists of a method which involves a population of candidate solutions, i.e., a swarm of particles. A candidate solution (particle) can be iteratively improved given a quality threshold. Particles search for the optimal solution adopting a 'distributed' coordination model based on specific parameters like their current position (solution) and velocity (update on their position) towards the best solution in the swarm. This way, particles are able to move in the search space influenced by their own local best position and the best position in the swarm. Any movement to a new solution is performed and affected through the adoption of a fitness function. Obviously, the discovery of a better position by a peer-particle will affect the searching activities of any other participant in the swarm representing the bio-inspired approach in searching the solution space.

In the current work, we adopt the *Simplified Swarm Optimization* (SSO) algorithm [64]. In this variant of the PSO algorithm, the concept of stochasticity

Algorithm 1 The clustering approach

Input: Set of W tasks, Set of N EC nodes $\mathcal{E}$
Output: Allocation $X(\epsilon_i, e_j), \forall i = 1, 2, \ldots, W$
$newTasks = receiveNewTasks()$ //new cycle starts
$tasks = countAllTasks()$
$free = countFreeENs()$ //EN: EC node
if $(free == 0)$ **then**
//if no available EC nodes exist$addToQueue(newTasks)$ //add new tasks to queue to be executed in the next cycle $return$
end if
$prioritizeAllTasks()$ //prioritize tasks based on time of arrival and task index
$clusters = min(tasks, free)$ //calculate number of tasks that can be allocated in this cycle
$defineNewTasksAsClusterCentroids()$ //make a centroid for each of the most prioritized tasks that can be allocated in this cycle
if (tasks $\geq$ free) **then**
//if clusters greater than or equal to free nodes$C = 1$ //allocate exactly one node to each task/centroid $remainder = 0$ **if** $(tasks > free)$ **then**
$addToQueue(extraTasks)$ //cannot accommodate all tasks
end if
else
//if clusters less than free nodes$C = free/tasks$ //allocate at least C nodes to each task
$remainder = free \bmod tasks$ //and the rest of the nodes to some tasks
end if
for all clusters i in priority **do**
//find candidate nodes for each cluster in priority**for** $c = 1, 2, \ldots, C$ **do** //at least C candidates
for all free ENs j **do**
$R_{ij} = distance_{ij}/speed_j$ //based on heuristic realized by R
end for
$EN = findENWithMinRForTask(i)$
$addToCluster(EN, i)$
end for
end for
for all clusters i in priority **do**
//allocate remaining free EC nodes as candidates to clusters**for all** free ENs j **do**
$R_{ij} = distance_{ij}/speed_j$
end for
$EN = findENWithMinRForTask(i)$
$addToCluster(EN, i)$
end for
for all clusters i **do**
for all ENs j in cluster i **do**
$cost_{ij} = costFunction(i, j)$ //find cost for each candidate node j's allocation to task i
end for
$X_i = findENWithMinCost(i)$ //choose minimum-cost solution for each task
end for

is introduced and incorporated in the selection of particles' next position. The aim is to apply randomness in their behaviour limiting the possibilities of having particles being trapped in local minima. PSO combines local search methods (through self experience) with global search methods (through neighboring experience), attempting to balance exploration and exploitation. PSO is a stochastic search algorithm, thus, is prone to the lack of global search ability at the end of a run. PSO sometimes suffers from premature convergence on problems with many local minima, thus, this property may cause a swarm to become trapped

in one of them and fail to explore more promising neighboring minima [55]. In every iteration, each candidate solution's fitness value is realized according to a function that is affected by the costs of execution and communication. This way, the SSO algorithm incorporates into its logic the constraints of the execution for each task. Such constraints aim at securing that the retrieved solution is aligned with the requirements of the problem. Particles personal best position is updated only when the current fitness value is better than the previous one while the global best position is also updated to be the optimal of the local best solutions in the swarm. The last step for every iteration is to calculate each particle's new position randomly selecting among its personal best, the global best, its current position and a random value. Decisions over random values impose the aforementioned stochasticity of the model. In our scenario, a particle is a potential allocation of W tasks to the available nodes. Every movement of a particle is represented by updates in tasks allocation. The cost of execution for the ith task to the jth node is calculated as follows: $ec_{ij} = \frac{c_i}{s_j}$ This equation, i.e., the calculation of ec_{ij}, depicts the ability of a node to quickly conclude a task affected by the complexity of the task as depicted by the steps required to deliver the final outcome. The speed s_j depicts the processing capacity of the jth node as requested by the SSO algorithm. In addition, we consider l_j as the memory capacity of a node as also required by the SSO algorithm. Without loss of generality, we assume the communication cost equal to zero (it depicts the communication between the TC and the available nodes). Finally, the tasks' processing requirement is defined through $\hat{d}_i$. The stopping criterion for SSO is the execution of the envisioned processing for a number of epochs. In Algorithm 2, we present our bio-inspired approach.

5 Experimental Evaluation

5.1 Simulation Setup & Performance Metrics

The development of the real world large-scale IoT applications leads to the generation of enormous amounts of data ready to be processed. Such a use case constitutes the smart-watch applications widely used during these times. A set of sensors embedded in a smart watch constantly collect data concerning biometrics as a result of human activities. On a daily basis, data are aggregated for each user and preliminary calculations performed on top of them taking place at the edge prior the transfer to Cloud. The proposed models distribute multiple tasks coming from multiple users to the appropriate EC nodes in order to be processed efficiently. Thus, the overall load can be distributed in a way that all nodes have uniform loads, by taking into consideration their current performance as well as the tasks' requirements.

We present the performance of the proposed allocation algorithms, i.e., the Clustering Approach and the Simplified Swarm Optimization Approach. We adopt a set of performance metrics and simulate a task streaming environment by setting the values of tasks and nodes' characteristics. Specifically, the load of nodes is drawn from two datasets; the Cooling Load feature of the Energy Efficiency dataset[1] and the Processor Utilization feature of the Optical Interconnection Net-

[1] https://archive.ics.uci.edu/ml/datasets/Energy+efficiency

Algorithm 2 The SSO approach

Input: Set of W tasks, Set of N EC nodes $\mathcal{E}$
Output: Allocation $X(\epsilon_i, e_j), \forall i = 1, 2, \ldots, W$
$initializeConstants(C_g, C_p, C_W, \lambda)$ //$C_g < C_p < C_w \in (0,1)$: constants that affect the possibility of updating a particle based on global best, personal best, or random value respectively, λ: coefficient defining the restraints' effect on the fitness function
$X = produceRandomCandidateSolutions()$ //start with random particles
$first = true$ //flag signifying the first iteration
while $!criterionIsMet()$ **do**
for all candidate solutions X_k **do**
//calculate fitness function for all particles$f(X_k) = \sum_{j=1}^{m}\sum_{i=1}^{n} ec_{ij}(X_{ki} == j?1 : 0)$ //consider execution costs for particle's allocation $F(X_k) = f(X_k) + \lambda\left(\sum_{t=1}^{m} \frac{\left(max(0, M_t - \sum_{i=1}^{n} m_i(X_{ki}==t?1:0)\right)^2}{\sum_{w=1}^{S}\left(max(0, M_t - \sum_{i=1}^{n} m_i(X_{wi}==t?1:0))\right)^2} + \sum_{t=1}^{m} \frac{\left(max(0, P_t - \sum_{i=1}^{n} p_i(X_{ki}==t?1:0)\right)^2}{\sum_{w=1}^{S}\left(max(0, P_t - \sum_{i=1}^{n} p_i(X_{wi}==t?1:0))\right)^2}\right)$ //consider whether the particle's allocation exceeds nodes' memory and processing capacities $best = 1$ **if** $(first || (F(X_k) > pBestFitness_k))$ **then** //check and save if particle's current solution is personal best
$pBest_k = X_k$
$pBestFitness_k = F(X_k)$
$first = false$
end if
if $(pBestFitness_k > pBestFitness_{best})$ **then**
//check and note if particle's personal best solution is global best$best = k$
end if
$gBest = pBest_{best}$
end for
for all candidate solutions X_k **do**
//update every particle's position per task**for all** tasks X_{ki} **do**
$\rho = random(0,1)$ //randomly pick $\rho \in (0,1)$
if $(\rho < C_g)$ **then**
$X_{ki} = gBest_i$ //update with globally best EC node for this task
else if $\rho < C_p$) **then**
$X_{ki} = pBest_{ki}$ //update with personally best EC node for this task
else if $(\rho \geq C_w)$ **then**
$X_{ki} = random(1, N)$ //update with random EC node
end if
end for
end for
end while

work dataset[2]. At this point, we highlight that the aforementioned values are used in the experiments after their normalization in the unity interval. The remaining features - nodes' speed, tasks' complexity and deadline - are initialized randomly being uniformly distributed in the same interval. Recall that s_j, c_{ϵ_t} and τ_{ϵ_t} can be also defined in the unity interval. When these parameters approach the unity means that we experience the highest possible value (the highest speed, the highest load and the highest possible deadline) while the opposite scenario holds true when they are approaching zero (we experience the lowest possible value). In addition, we compare our model with other mechanisms found in the relevant literature:

– the *Greedy Fast Processing* (GFP) model presented in [46]. The model selects the node offering the best processing time for each task. The model is also met in [24] named as the myopic best response selection algorithm and in [7] named

[2] https://archive.ics.uci.edu/ml/datasets/Optical+Interconnection+Network+

as the performance aware allocation scheme. We have to note that the model presented in [7] allocates tasks to the best idle nodes; if not any idle nodes are available, the model performs a random allocation (see the RTS model below);
- the *Random Task Scheduling* (RTS) model proposed in [7]. The model selects nodes without taking into consideration any contextual information. Tasks are randomly allocated in the available nodes.
- the model proposed in [5] called ETSI. ETSI is mainly focused to IoT environments and is based on a heuristic that delivers the final outcome upon the distance from the edge of the network and the number of neighbours to retrieve the ranking of each processing node. The node with the lowest ranking is selected for the final allocation.
- the *Demand-driven Proactive Tasks Management* (DPTM) model proposed in [25]. DPTM considers historical demand values for tasks in various regions. Based on them, it maintains a Kernel Density Estimator (KDE) that predicts future task demands in these areas. Local and external KDEs are considered to decide whether or not a task should be executed locally in the future.

We define the metric T which represents the time required for the allocation of a task ϵ_t to a node e_j. T realizations may vary depending on the adopted algorithm and the number of tasks or nodes involved in the allocation process. Our main target is to reduce the average T for each simulation scenario. As far as the chosen node's load is concerned, we adopt the metric Λ. This metric depicts the difference of the selected node's load with the lowest load among all nodes. The following equation holds true:

$$\Lambda = l_{selected} - l_{lowest} \tag{2}$$

When $\Lambda \to 0$ the selected node's load is low, thus, this node is the most appropriate to host the task since the waiting time before starting the execution is the least among all nodes. In contrast, when $\Lambda \to 1$, we conclude that the selected node is the worst to choose w.r.t. the waiting time for tasks' execution. Another performance metric adopted in our experiments in order to evaluate the selected node's speed is Σ. Σ is defined as the difference between the highest speed among all nodes with the speed of the selected node. Σ is calculated by the following equation:

$$\Sigma = s_{highest} - s_{selected} \tag{3}$$

When $\Sigma \to 0$ the speed of the selected node is high, approaching the highest possible. In that case, the selected node is the most appropriate to host the task since the task will be quickly processed and results will be returned as early as possible. On the other hand, the higher the Σ is, the lower the processing speed for the task becomes. The final adopted metric, Φ, is a linear combination of Λ and Σ specified as follows:

$$\Phi = \alpha * \Lambda + (1 - \alpha) * \Sigma, \alpha \in [0, 1] \tag{4}$$

where α constitutes a weighting factor, whose value is set depending on the metric - Λ or Σ - we want to focus on. More specifically, when $\alpha \to 0$, Σ gains full attention and dominates the performance of the system leaving no room for Λ to affect Eq.(4). When $\alpha \to 1$, Λ is the parameter which mostly affects Eq.(4). In case of $\alpha = 0.5$, Φ depends equally on Λ and Σ. When $\Phi \to 0$, the best performance

is achieved for Λ, Σ at the same time. It should be noted that the higher the Φ is, the lower the overall performance becomes. We also evaluate the performance of the clustering algorithm based on a widely adopted metrics, i.e., the Silhouette coefficient (SC) [14]. The SC refers to the validation of the consistency within the delivered clusters clusters and provides a representation of how well each object has been classified. The SC metric is defined as follows:

$$a_i = \frac{1}{|CL_i - 1|} \sum_{j \in CL_i, j \neq i} d(i,j) \tag{5}$$

$$b_i = \min_{k \neq i} \frac{1}{|CL_k|} \sum_{j \in CL_k} d(i,j) \tag{6}$$

$$SC_i = \frac{b_i - a_i}{max(a_i, b_i)} \tag{7}$$

where CL_i is the ith cluster, $d()$ is the distance function.

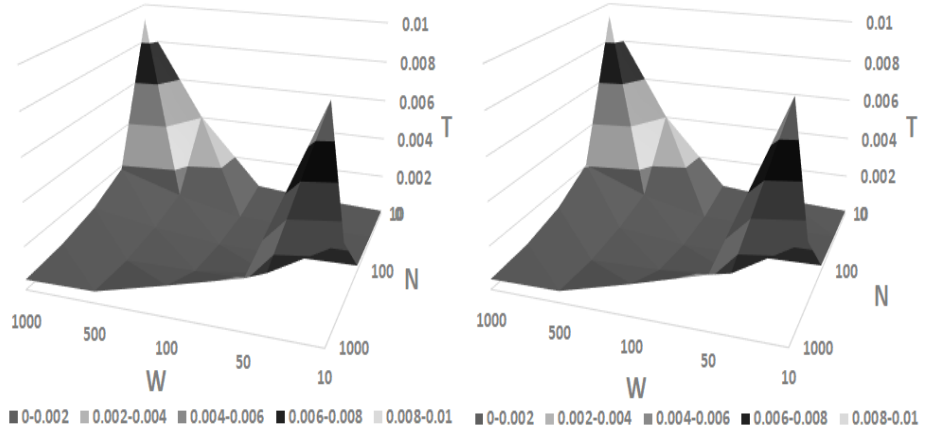
5.2 Performance Assessment

In this subsection, we study and analyse the metrics that we previously defined for the two models and for the two datasets used for the experiments respectively. The first model of which experiments are discussed is the Clustering Approach; Following that, we demonstrate results of the Bio-inspired Tasks Allocation Model and as a final step of our experiments we compare the aforementioned models.

5.2.1 The Clustering Approach

In this section, the results of our experiments using the Clustering Approach are demonstrated. The metrics as well as their graphical representation are analysed in order to provide the full understanding of the experiments. As Figure 2 shows, when $N \rightarrow W$ the T metric is minimized. In case of $N \rightarrow 1000$, our mechanism should process a high number of nodes that clearly affects the total required time. The same stands for $W \rightarrow 1000$, i.e., many tasks should be processed before a final decision is in place. In any case, our results exhibit an efficient decision mechanism as the maximum conclusion time is below 0.01 seconds no matter the adopted dataset. This means that we are able to have a high throughput in the allocation process increasing the performance of TCs and making them capable of efficiently supporting a tasks streaming scenario.

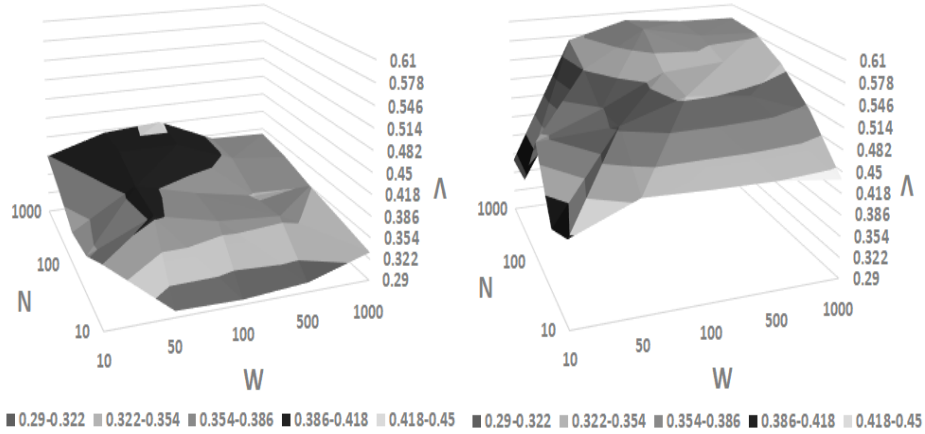
In Figure 3, we notice that the Λ tends to have small deviations. When adopting the Energy Efficiency dataset, Λ ranges in the interval $[0.29, 0.43]$. It appears that as long as $M > N$, which indicates that the EC nodes outnumber the tasks, every task selects the most appropriate candidate among the nodes which are assigned to its cluster. In contrast, in case the number of tasks N is high and the number of nodes M is low, Λ maximizes. Specifically, in each iteration, tasks with the highest complexities choose nodes with low loads (for their clusters), rendering those nodes unavailable until the tasks' completion. As a result, the remaining free nodes, whose load values are high, are repeatedly chosen by low-complexity tasks leading to the maximization of the average Λ. The same 'attitude' stands



(a) 1st dataset (b) 2nd dataset

Fig. 2: The conclusion time for the clustering approach

when the second dataset feeds our parameters. Performance outcomes are affected by the number of nodes leading to a slightly higher Λ realizations compared to the previous experimental scenario.

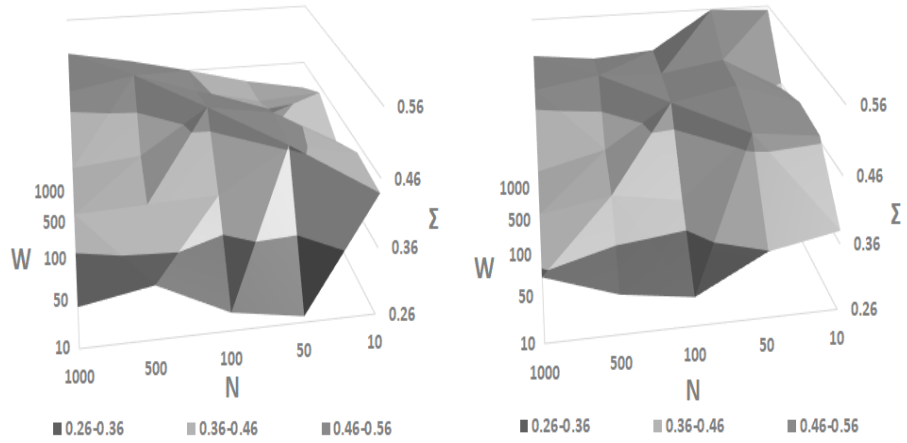


(a) 1st dataset (b) 2nd dataset

Fig. 3: The load of the selected node - Clustering approach

Figure 4 shows that for both datasets, Σ is low when the number of nodes is low. This is explained by the fact that, after the first iteration of the allocation process, the fastest nodes are mainly available to execute the next set of queries, leading to the reduction of Σ. An increased Σ is realized when multiple nodes are available and examined for selecting the best one for each task. In these cases, our

model results a slightly higher Σ compared to the experimental scenario involves a low N. Recall that are decision mechanisms tries to take into consideration all parameters when matching tasks with nodes, thus, fluctuations may be present in the realizations of each metric. This is because the proposed model tries to achieve the best results for all parameters at the same time.



(a) 1st dataset (b) 2nd dataset

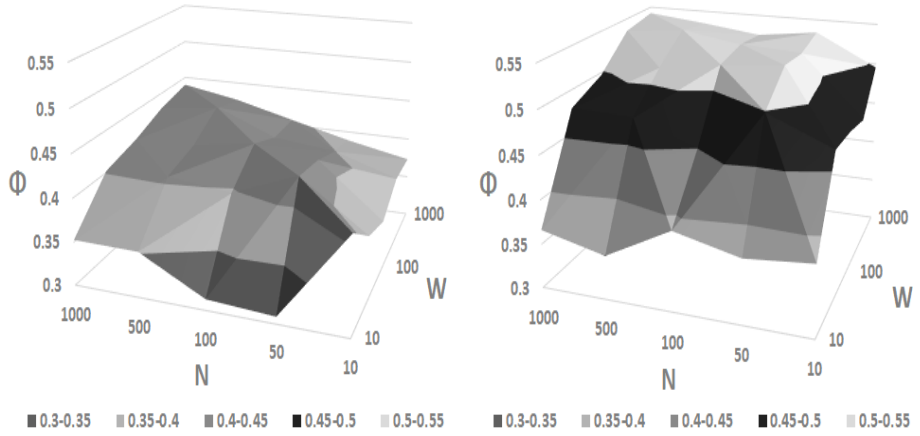
Fig. 4: The speed of the selected node - Clustering approach

Figure 5 demonstrates our results for the Φ metric. Recall that Φ depicts the performance of Λ and Σ at the same time. We observe that the trend of Φ is affected, as natural, by the trends of the two aforementioned parameters. The interval where Φ takes its values is $[0.30, 0.55]$ for both datasets. In general, N is the parameter that affects more the final outcome as in the previous experimental scenarios.

Concerning the performance of the clustering approach, we get a mean SC in $\{0.25, 0.29, 0.48, 0.71, 1.0\}$ and $\{0.23, 0.29, 0.47, 0.71, 1.0\}$ for the aforementioned datasets, respectively and for all the W realizations (i.e., $W \in \{10, 50, 100, 500, 1000\}$). These results are retrieved for $N \in \{10, 50, 100, 500, 1000\}$. We observe that the increased number of nodes and tasks positively affect the performance of the clustering model as it manages to place the available EC nodes to the appropriate cluster, i.e., the task-centroid. Recall that when the SC metric is near unity, it indicates that every object (i.e., EC node) is far away from the neighboring clusters (i.e., the other tasks). Our observations reach to the conclusion that the underlying dataset does not affect the outcome, thus, making the proposed model to be adopted in various application scenarios.

5.2.2 Bio-inspired Tasks Allocation

In this section, the results of the experiments adopting the Bio-inspired Tasks Allocation are depicted through the graphical representation of the results interpretation. In Figure 6, we observe that when N is low and, especially, when W is



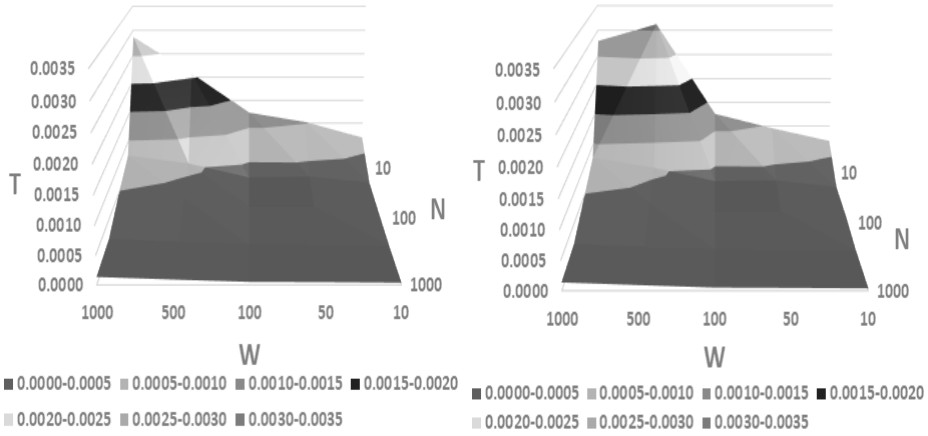
(a) 1st dataset (b) 2nd dataset

Fig. 5: Our performance evaluation for the Φ metric - Clustering approach

high, T is maximized. This is caused by the fact that for every node and every candidate solution a high set of constraints have to be calculated according to the SSO algorithm. However, T is less than 0.0030 seconds which, in turn, is lower than T realization when adopting the clustering approach. Our model is capable of supporting the immediate conclusion of allocations no matter the number of nodes. In this set of experiments, we adopt a number of 10 iterations for reaching a solution and the number of particles is 10 as well. This means, that the solution space is limited, however, the proposed scheme can easily and timely respond to the desired allocations.

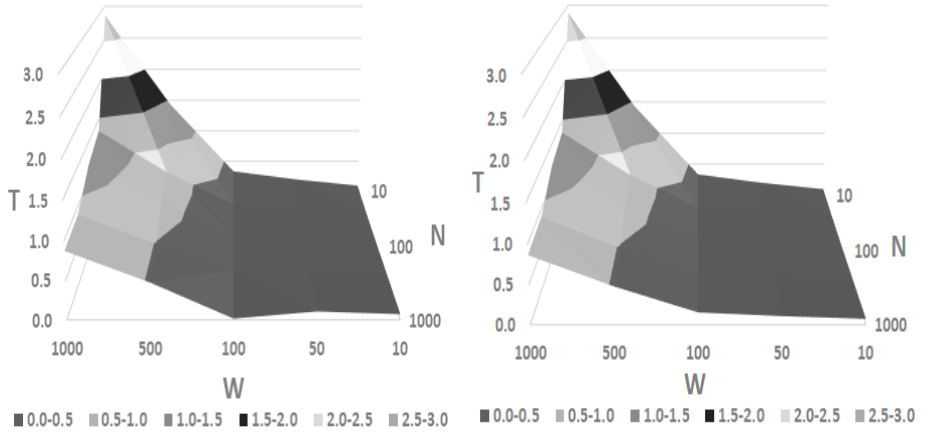
In Figure 7, we focus on an experimental scenario where the number of particles and the number of iterations are 1000. This increases the size of the solution space with a clear impact in the time required to conclude a decision. Now, T is higher than in the previous experimental scenario (i.e., adopting a limited particles and iterations number). This is natural as our model needs more time to examine all the potential solutions and conclude the final decision. Again, T is maximized when $N \rightarrow 10$ and $W \rightarrow 1000$.

In Figures 8 and 9, we present our results for Λ and different number of particles and iterations. When the number of particles and iterations are limited, out model results Λ below 0.40 and 0.60 for the 1st and 2nd datasets, respectively. The performance outcomes are affected by increased N and W; in these experimental scenarios Λ is maximized. This is natural, since as M increases the probability of finding the most appropriate node for every task decreases, because the probability distribution is uniform, and the candidate solutions are generated and changed randomly. In case we adopt an increased number of particles and iterations, our model results Λ below 0.42 and 0.70 for the 1st and the 2nd datasets, respectively. Now, we observe a difference in our results depending on the adopted dataset. For instance, when the 2nd dataset is feeding our parameters, an decreased N ($N \rightarrow 10$) leads to the highest Λ. In these cases, our model cannot find the optimal



(a) 1st dataset (b) 2nd dataset

Fig. 6: The conclusion time of the bio-inspired model - low number for particles and iterations

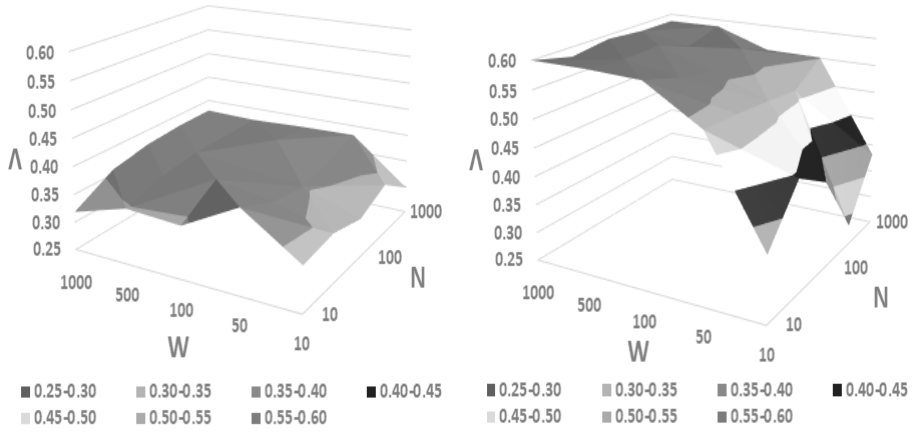


(a) 1st dataset (b) 2nd dataset

Fig. 7: The conclusion time of the bio-inspired model - high number for particles and iterations

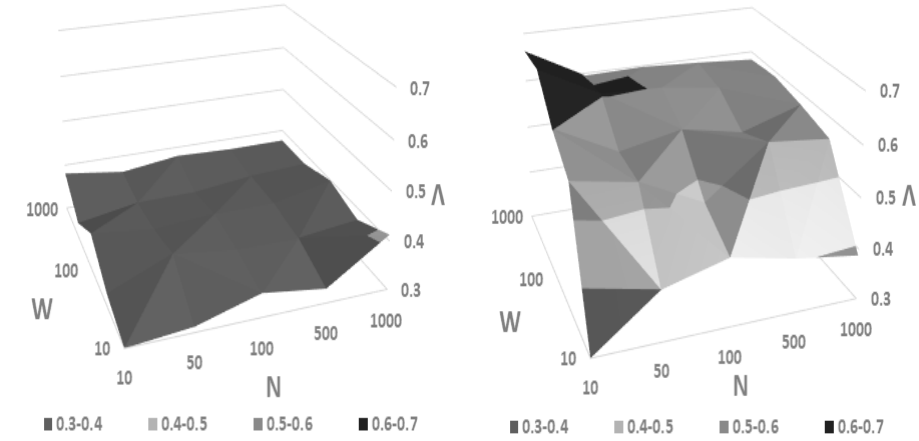
solution in the limited solution space. However, as the number of particles and the number of iterations increase, the Λ values, for the most part, decrease.

As far as the Σ is concerned, Figures 10 & 11 present our results. When we adopt a low number of particles and iterations, Σ is in the interval $[0.30, 0.51]$ for both datasets. In general, Σ is close to the maximum value for the majority of the experimental scenarios. An increment of the number of particles and iterations increases the maximum value of Σ as well. Now, the realizations of Σ are between 0.30 and 0.70 for both datasets. This means that the increased number of particles



(a) 1st dataset (b) 2nd dataset

Fig. 8: The load of the selected node for the bio-inspired model - low number for particles and iterations

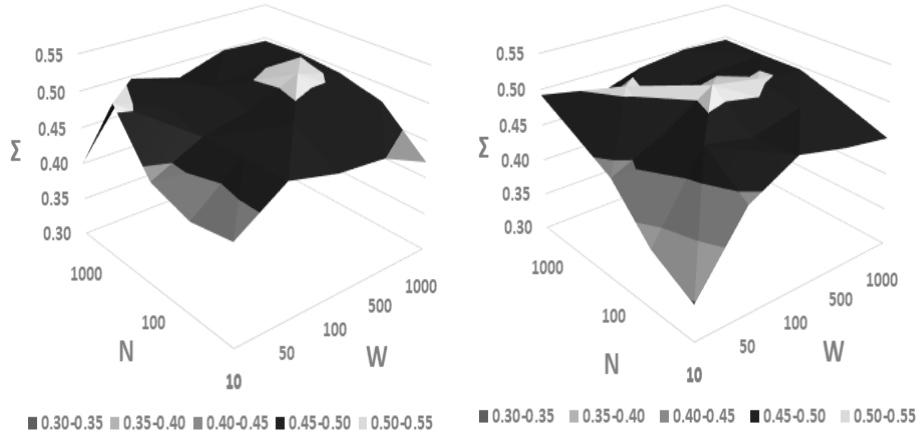


(a) 1st dataset (b) 2nd dataset

Fig. 9: The load of the selected node for the bio-inspired model - high number for particles and iterations

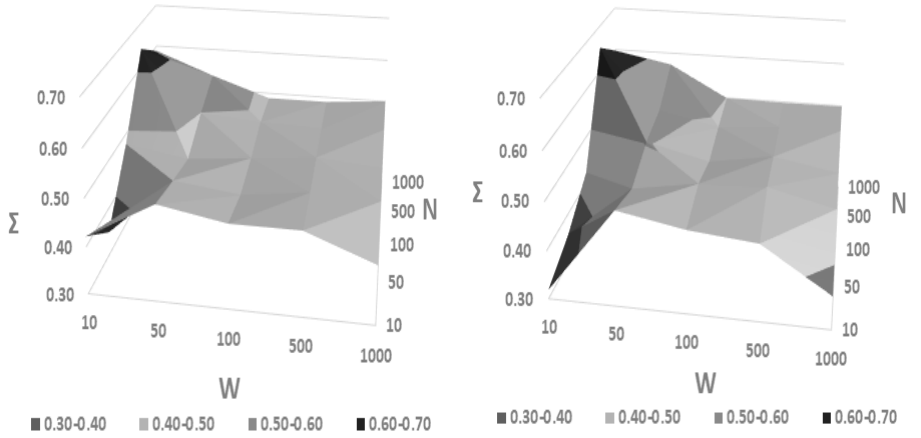
makes our model to conclude a higher speed difference with the optimal node. The best performance of the proposed scheme is achieved when a limited solution space is adopted (low N and W). When M is low, as explained previously for the Λ metric, the probability of generating the appropriate solution increases. On the other hand, when N is low, the tasks' complexity and deadline requirements don't exceed the EC nodes' load and speed limits, thus leading to better allocations and, by extension, low Σ values. What is more, as long as M is low and N is high, Λ's

values are increased due to the fact that the tasks' requirements exceed the nodes' capacities which leads to inaccurate allocations.



(a) 1st dataset (b) 2nd dataset

Fig. 10: The speed of the selected node for the bio-inspired model - low number for particles and iterations

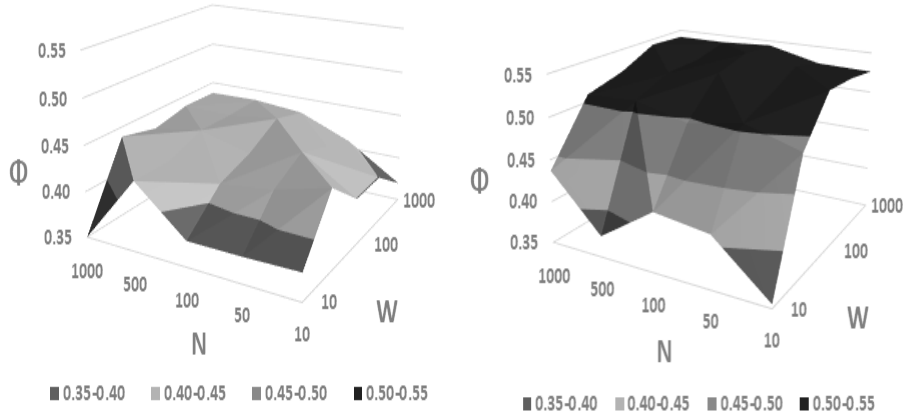


(a) 1st dataset (b) 2nd dataset

Fig. 11: The speed of the selected node for the bio-inspired model - high number for particles and iterations

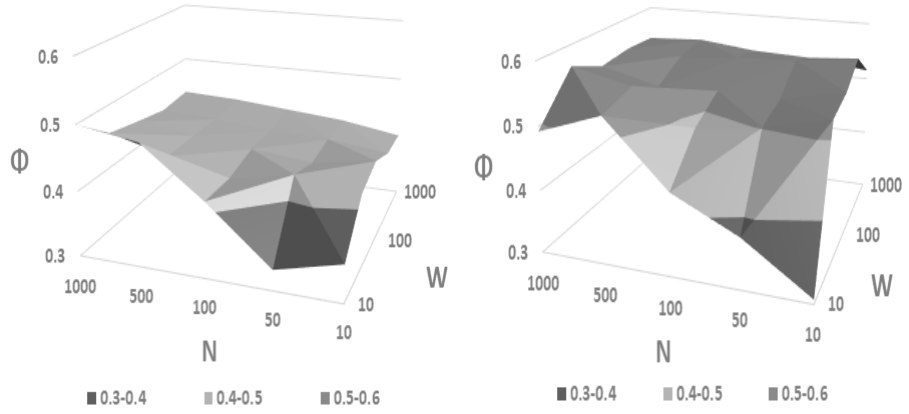
Figures 12 & 13 demonstrate the results of our experiments for the Φ metric, for both datasets. As natural, the metric 'inherits' its realization being affected

by Λ and Σ. Having Λ and Σ increased when the number of particles and iterations are low, Φ increases as well. In any case, Φ is below 0.55 and 0.60 (for the two experimental scenarios) which relies at the middle of the interval where Φ is realized.



(a) 1st dataset (b) 2nd dataset

Fig. 12: Our results for Φ and for the bio-inspired model - low number for nodes and iterations

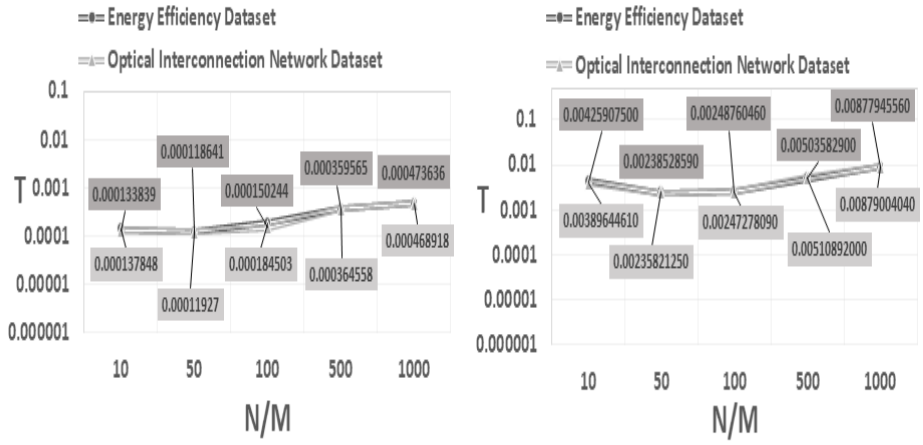


(a) 1st dataset (b) 2nd dataset

Fig. 13: Our results for Φ and for the bio-inspired model - high number for nodes and iterations

5.2.3 Comparative Assessment

In this section, we compare the results of the two proposed algorithms along with other models found in the literature. The performance concerning the time T that each algorithm needs to allocate a task is presented on Figures 14, 15 and 16. When the number of tasks N equals the number of nodes M, T, for both algorithms, remains mainly constant with the clustering approach exhibiting the best results among the algorithms when $N = 1000$. In the case where the number of nodes exceeds the number of tasks, the clustering approach shows the best results while the SSO behavior is the worst. For the opposite case, the SSO model provides better time results than the clustering approach.



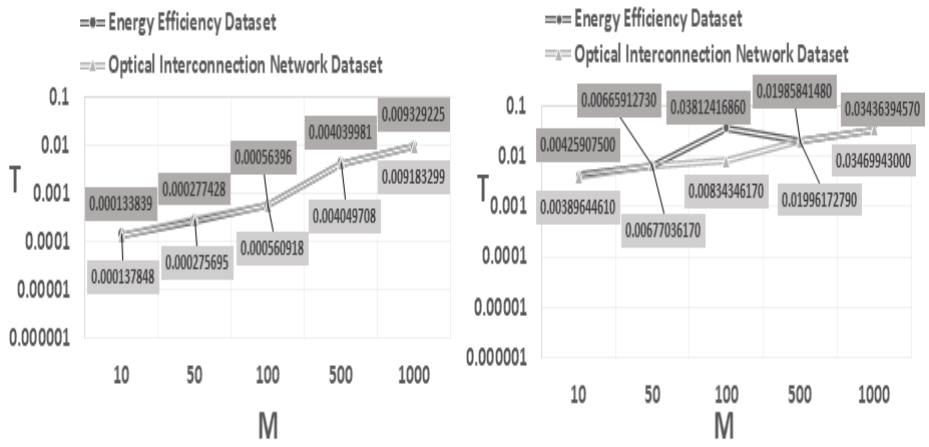
(a) Algorithm 1 (b) Algorithm 2

Fig. 14: Comparison of the proposed approaches (T metric - $N = M$)

In Figures 17, 18, 19, 20, 21 and 22 we present our performance assessment results for the Φ metric comparing the proposed algorithms for $\alpha \in \{0.20, 0.50, 0.70\}$. Specifically, in Figures 17 and 18, where the numbers of tasks and nodes are equal, Φ values follow a virtually steady course, since there are only small increases which may be considered negligible. Furthermore, in Figures 19 and 20, as the number of tasks increases with respect to the number of nodes, the observed Φ of the clustering approach is reduced, whereas the SSO model's respective values increase.

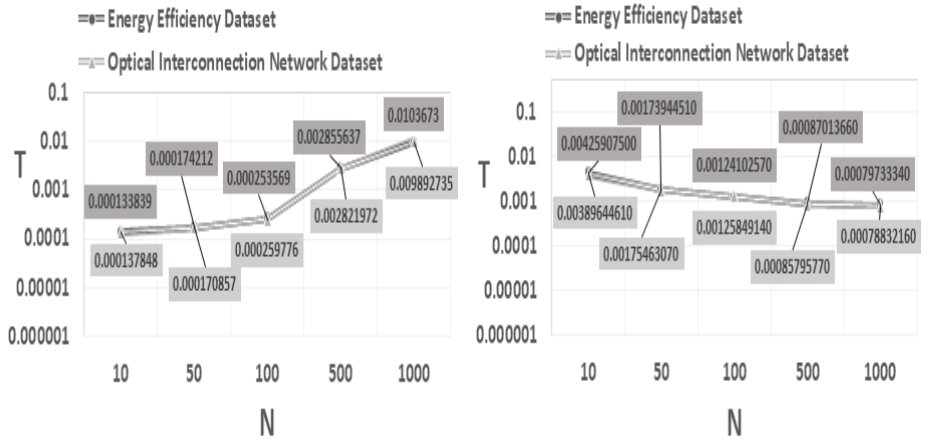
In Figure 21, we observe that the clustering approach results better than the SSO model when $N \to 10$ and the opposite stands when $N \to 1000$. α is not affecting the final outcome having a Φ around 0.50. In Figure 22, we provide results for the second dataset. We observe a similar performance as when the 1st dataset feeds our parameters. In both experimental scenarios we get $M = 10$.

We compare our models with the GFP, RTS and ETSI schemes found in the relevant literature. The GFP, for uniformly distributed parameters, results Λ in $[0.41, 0.48]$ for $N \in \{10, \ldots, 1000\}$. For the same setup, the RTS achieves Λ around 0.50. For the same experimental scenarios, ETSI achieves Λ in $[0.04, 0.23]$ for both



(a) Algorithm 1 (b) Algorithm 2

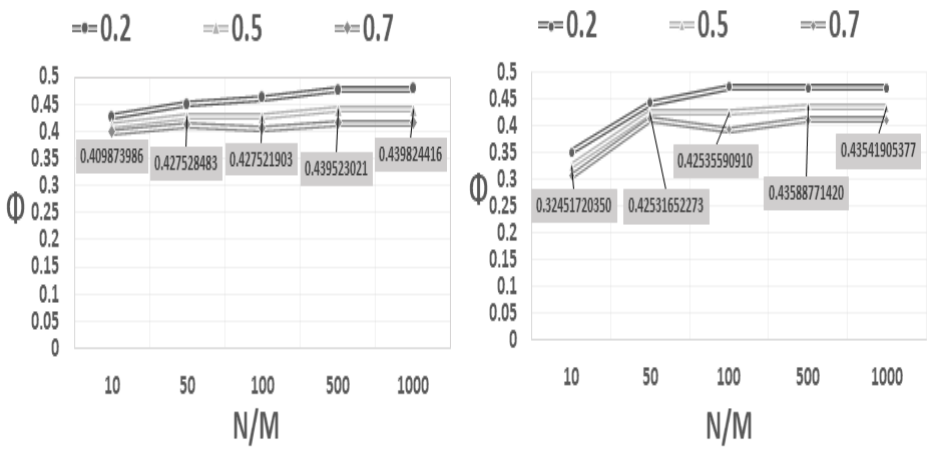
Fig. 15: Comparison of the proposed approaches (T metric - $N = 10$)



(a) Algorithm 1 (b) Algorithm 2

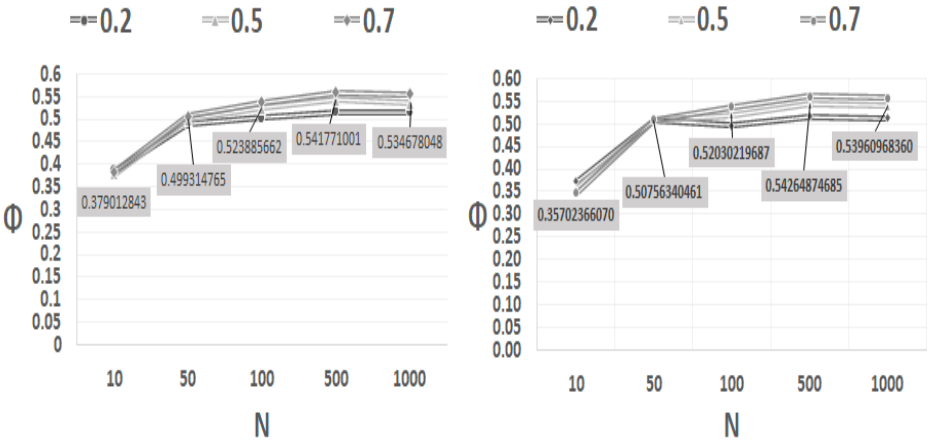
Fig. 16: Comparison of the proposed approaches (T metric - $M = 10$)

the experimental datasets. Additionally, ETSI achieves Σ in $[0.001, 0.44]$ for the aforementioned datasets. Recall that our clustering approach manages to retrieve Λ in $[0.29, 0.43]$ and $[0.37, 0.61]$ for the 1st and the 2nd datasets, respectively. Furthermore, the clustering process manages to achieve Σ in $[0.26, 0.56]$ for both the experimental datasets. Our SSO model results Λ in $[0.30, 0.40]$, $[0.25, 0.60]$ for the 1st dataset (low and high values for the number of particles and iterations). In the scenario where we adopt the 2nd dataset, our performance outcomes are Λ in $[0.30, 0.42]$, $[0.30, 0.70]$. The outcomes for Σ are in the following interval $[0.30, 0.68]$ for both datasets. We observe that our models outperform the GFP and the RTS managing to select nodes with low load, thus, to facilitate the fast initiation of tasks



(a) Algorithm 1 (b) Algorithm 2

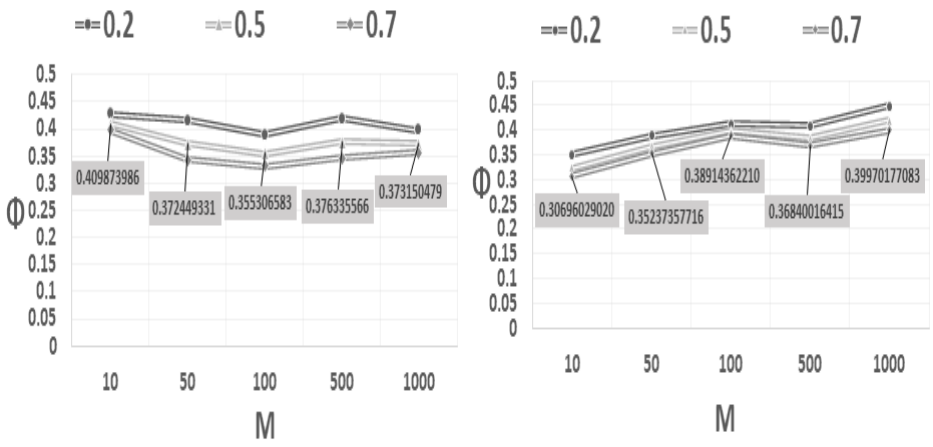
Fig. 17: Comparison of the proposed approaches (Φ metric - $N = M$ - 1st dataset)



(a) Algorithm 1 (b) Algorithm 2

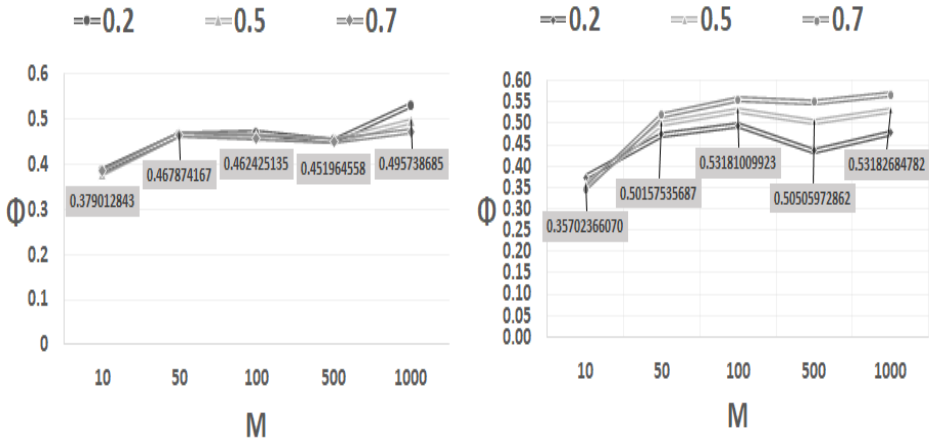
Fig. 18: Comparison of the proposed approaches (Φ metric - $N = M$ - 2nd dataset)

execution. When comparing our models with ETSI, we can detect the strength of our approach. We observe that ETSI outperforms the proposed models when we take into consideration only the metric Λ. This holds true due to the 'attitude' of ETSI to 'reward' the allocation of tasks in nodes with a low load, however, being 'blind' to the remaining decision making parameters. However, when it comes to compare ETSI with our models concerning the metric Σ, we observe the clear superiority of our approach. ETSI exhibits a very low Σ compared to our schemes. Evidently, we see that our clustering and SSO schemes target to take into consideration all the adopted parameters before the allocation takes place. Hence, we manage to achieve a low load and a high speed at the same time making our



(a) Algorithm 1 (b) Algorithm 2

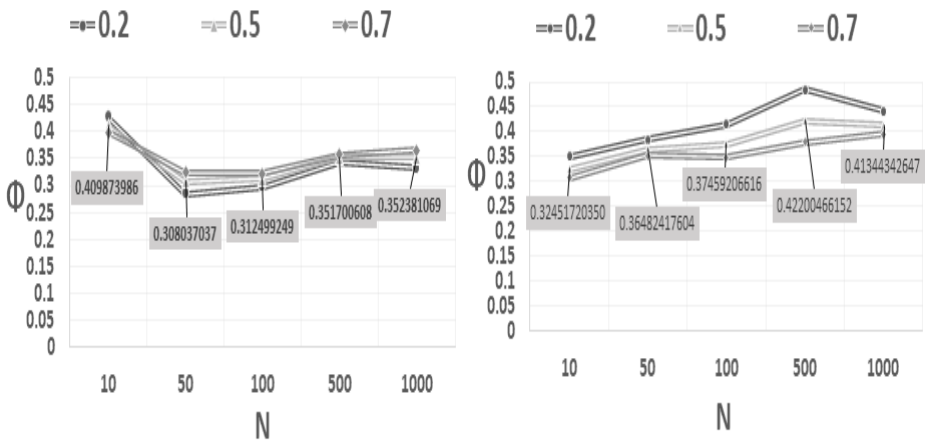
Fig. 19: Comparison of the proposed approaches (Φ metric - $N = 10$ - 1st dataset)



(a) Algorithm 1 (b) Algorithm 2

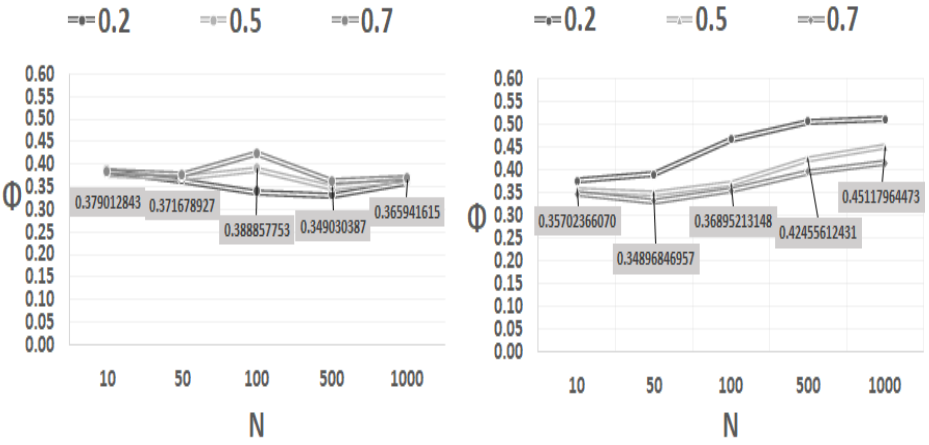
Fig. 20: Comparison of the proposed approaches (Φ metric - $N = 10$ - 2nd dataset)

approach resilient to the adoption of multiple parameters into the decision making model. In addition, comparing our models with DPTM, we reach the following conclusions. DPTM demonstrates low Φ values, which fall in $[0.01, 0.17]$. Our own models achieve Φ values in $[0.3, 0.6]$, which are higher than the DPTM. Nevertheless, the time it takes for our models to make decisions is $T \in [0.0001, 0.038]$, whereas the DPTM model achieves $T \in [0.5, 0.7]$. The proposed models' decision-making exhibits lower precision than the DPTM while still being acceptable. Our models achieve a Φ around 0.45 (the aggregated difference from the optimal node exhibiting the lowest load and the highest speed) which is not forbidden to decide an allocation for a task. This value indicates that our models manage to select



(a) Algorithm 1 (b) Algorithm 2

Fig. 21: Comparison of the proposed approaches (Φ metric - $M = 10$ - 1st dataset)



(a) Algorithm 1 (b) Algorithm 2

Fig. 22: Comparison of the proposed approaches (Φ metric $M = 10$ - 2nd dataset)

nodes with the load and speed being at acceptable levels. If $var\Phi$ was close to unity, then the performance could be the worst possible as the worst node (the node with the highest load and the lowest speed) is selected. However, if we also focus on the T metric, we can see a huge difference in the throughput of our models compared with the DPTM. In average, the DPTM manages to serve (allocate) around 1.67 tasks per time unit while the proposed models serve (allocate) around 52.36 tasks per time unit. This means that if we want to support streams submitting tasks at high rates and support real time applications, we can accept the trade off for serving a high number of tasks per time unit (thus, limiting the waiting time at TCs) in the burden of a (still acceptable as the outcomes for Φ are beyond the

unity) higher load and lower speed than the DPTM. The ability of the proposed models to conclude immediately the envisioned allocations while spending lower resources than the DPTM can be an alternative solution in a scenario where TCs select an allocation model depending on the surrounding contextual data.

6 Conclusions & Future Work

The management of the huge amounts of data generated in a daily basis constitutes a challenge to overcome in order to serve real time applications. Together with that novel models, especially defined in the Internet of Things, require intelligent and efficient models to process the continuously tasks execution requested by users or applications. In this paper, we propose models for tasks management that are based on Machine Learning and Computational Intelligence. We define the concept of a *Task Controller* (TC), a module which orchestrates tasks' allocation at a number of available nodes. TC is based on nodes' and tasks' characteristics aiming to reduce the time costs for the envisioned allocations and achieve a load balance among the available nodes. Our experimental results show that different algorithms would best answer the problem's requirements. We present performance insights for the time required to conclude an allocation as well as the optimality of each allocation in terms of selected nodes' characteristics.

Our future research plans involve the incorporation of more parameters into the decision-making process. One such parameter can be the data that a tasks demands to be executed for. Another parameter to be taken into consideration is the communication cost during the allocation of the tasks to EC nodes. We have to study a more complex decision process realized over multiple parameters not only related to the load and the speed of every node. Hence, we have to incorporate data similarity measures to the proposed approach to create a data-aware scheme.

References

1. Apiletti, D., et al., 'Frequent itemsets mining for Big Data: a comparative analysis', Big Data Research, 9, 2017, pp. 67-83.
2. Apostolopoulos, P., Tsiropoulou, E., Papavassiliou, S., 'Game-theoretic Learning-based QoS Satisfaction in Autonomous Mobile Edge Computing', Global Information Infrastructure and Networking Symposium (GIIS), 2018.
3. Apostolopoulos, P., Tsiropoulou, E., Papavassiliou, S., 'Risk-Aware Data Offloading in Multi-Server Multi-Access Edge Computing Environment', IEEE/ACM Transactions on Networking, vol. 28(3), 2020.
4. Bangui, H., et al., 'Moving to the Edge-Cloud-of-Things: Recent Advances and Future Research Directions', Electronics, 7, p. 309, 2018.
5. Baranidharan, B., Saravanan, K., 'ETSI: Efficient Task Allocation in Internet of Things', International Journal of Pure and Applied Mathematics, 117(22), 2017, pp. 229–233.
6. Bowden, D., et al., 'A Cloud-to-Edge Architecture for Predictive Analytics', Workshops of the EDBT/ICDT Conference, 2019.
7. Breitbach, M., Schafer, D., Edinger, J., Becker, C., 'Context-Aware Data and Task Placement in Edge Computing Environments', IEEE International Conference on Pervasive Computing and Communications (PerCom), pp. 1-10, 2019.
8. Chai, Z., et al., 'Towards Taming the Resource and Data Heterogeneity in Federated Learning', USENIX Conference on Operational Machine Learning, pp. 19–21, 2019.
9. Chandramouli, B., Goldstein, J., Quamar, A., 'Scalable Progressive Analytics on Big Data in the Cloud', in the VLDB Endowment, vol. 6(14), pp. 1726-1737, 2013.

10. Chatterjea, S., Havunga, P., 'A Taxonomy of Distributed Query Management Techniques for Wireless Sensor Networks', IJCS, 20(7), pp. 889-908, 2007.
11. Chen, Y., Zhu, F., Lee. J., 'Data quality evaluation and improvement for prognostic modeling using visual assessment based data partitioning method', Computers in Industry, 64(3), 2013, pp. 214-225.
12. Condie, T., et al., 'MapReduce online', In the 7th Conference on Networked Systems Design and Implementation, p. 21, 2010.
13. Cummins, R., et al., 'A Polya Urn Document Language Model for Improved Information Retrieval', ACM TIS, 9(4), pp. 1-34, 2010.
14. de Amorim, R. C., Hennig, C., 'Recovering the number of clusters in data sets with noise features using feature rescaling factors', Information Sciences. 324: 126–145, 2015.
15. Engelbrecht, A. P., 'Computational Intelligence An Introduction', Wiley, 2007.
16. Fan, Q., Ansari, N., 'Cost aware cloudlet placement for big data processing at the edge', in Proceedings of the IEEE International Confenference on Communications (ICC), 2017, pp. 1–6.
17. Fang, F., Xu, Y., Ding, Z., Shen, C., Peng, M., Karagiannidis, G., 'Optimal Task Assignment and Power Allocation for NOMA Mobile-Edge Computing Networks', http://arxiv.org/abs/1904.12389v1, 2019.
18. Guo, S., Dai, Y., Guo, S., Qiou, X., Qi, F., 'Blockchain Meets Edge Computing: Stackelberg Game and Double Auction Based Task Offloading for Mobile Blockchain', IEEE Transactions on Vehicular Technology, vol. 69(5), 2020.
19. Gupta, M., Banerjee, P. (1992). 'Demonstration of automatic data partitioning techniques for parallelizing compilers on multicomputers.' IEEE Transactions on Parallel and Distributed Systems, 3(2), pp. 179-193.
20. Hellerstein, J. M., Avnur, R., 'Informix under control: Online query Processing', Data Mining and Knowledge Discovery Journal, 2000.
21. Hossain, S. H., Nwakanma, C. I., Lee, J. M., Kim, D.-S., 'Edge computational task offloading scheme using reinforcement learning for IIoT scenario', ICT Express, 2020.
22. Huang, Z., Zhong, A., Li. G., 'On-Demand Processing for Remote Sensing Big Data Analysis', In IEEE ISPDPA, pp. 1241-1245, 2017.
23. Jermaine, C., et al., 'Scalable approximate query processing with the DBO engine', In SIGMOD, pp. 1-54, 2007.
24. Jošilo, S., Dán, G., 'Decentralized Algorithm for Randomized Task Allocation in Fog Computing Systems', IEEE/ACM Transactions on Networking, vol. 27(1), 2019, pp. 85-97.
25. Karanika, A., Oikonomou, P., Kolomvatsos, K., Loukopoulos, T., 'A Demand-driven, Proactive Tasks Management Model at the Edge', in 2020 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE), 2020, pp. 1-8.
26. Karanika, A., Soula, M., Anagnostopoulos, C., Kolomvatsos, K., Stamoulis, G., 'Optimized Analytics Query Allocation at the Edge of the Network', in 12th International Conference on Internet and Distributed Computing Systems, Naples, Italy, Oct. 10-12, pp. 181-190, 2019.
27. Khan, W., et al., 'Edge Computing: A Survey', FGCS, 97, 2019, pp. 219-235.
28. Kolomvatsos, K., 'Proactive Tasks Management for Pervasive Computing Applications', Journal of Network and Computer Applications (JNCA), Elsevier, vol. 176, 2021.
29. Kolomvatsos, K., 'A Distributed, Proactive Intelligent Scheme for Securing Quality in Large Scale Data Processing', Springer Computing, vol. 101(11), 2018, pp. 1687–1710.
30. Kolomvatsos, K., 'An Intelligent Scheme for Assigning Queries', Applied Intelligence, pp. 2730-2745, 2018.
31. Kolomvatsos, K., Anagnostopoulos, C., 'Multi-criteria Optimal Task Allocation at the Edge', FGCS, 93, 2019, pp. 358-372.
32. Kolomvatsos, K., Anagnostopoulos, C., 'A Deep Learning Model for Demand-driven, Proactive Tasks Management in Pervasive Computing', IoT, MDPI, 1(2), 240-258, 2020.
33. Kolomvatsos, K., Anagnostopoulos, C., 'An Edge-Centric Ensemble Scheme for Queries Assignment', in 8th CIMA Workshop, 2018.
34. Kolomvatsos, K., Anagnostopoulos, C., 'Reinforcement Machine Learning for Predictive Analytics in Smart Cities', Informatics, 4(16), pp. 16, 2017.
35. Kolomvatsos, K., Hadjiefthymiades, S., 'Learning the Engagement of Query Processors for Intelligent Analytics', Applied Intelligence, 46, 2017, pp. 96-112.
36. Kolomvatsos, K., Anagnostopoulos, C., Koziri, M., Loukopoulos, T., 'Proactive Time-Optimized Data Synopsis Management at the Edge', IEEE Transactions on Knowledge and Data Engineering (IEEE TKDE), 1-1, 2020, 10.1109/TKDE.2020.3021377.

37. Lacerda, B., Parker, D., Hawes, N., 'Optimal policy generation for partially satisfiable co-safe LTL specifications', in IJCAI, pp. 1587-1593, 2015.
38. Lahijanian, M., Kwiatkowska, M., 'Specification revision for Markov decision processes with optimal trade-off', in CDC, pp. 7411-7418, 2016.
39. Lee, H. S., Lee, J. W., 'Task Offloading in Heterogeneous Mobile Cloud Computing: Modeling, Analysis, and Cloudlet Deployment', IEEE Access, vol. 6, 2018, pp. 14908-14925.
40. Li, X., Zhu, L., Chu, X., Fu, H., 'Edge Computing-Enabled Wireless Sensor Networks for Multiple Data Collection Tasks in Smart Agriculture', Journal of Sensors, art. id 4398061, 2020.
41. Liu, X., Qin, Z., Gao, Y., 'Resource Allocation for Edge Computing in IoTNetworks via Reinforcement Learning', http://arxiv.org/abs/1903.01856v1, pp. 1-6, 2019.
42. Logothetis, D., Yocum, K., 'Ad-hoc Data Processing in the Cloud', VLDB Endowment, 1(2), pp. 1472-1475, 2008.
43. Ma, H., Kumar, T., Koenig, S., 'Multi-agent path finding with delay probabilities', in AAAI, 2017.
44. Ma, L., Wu, J., Chen, L., 'OTA: Delay bounded optimal cloudlet deployment and user association in WMANs', in Proceedings of the IEEE/ACM International Symposium on Cluster Cloud Grid Computing (CCGRID), 2017, pp. 196–203.
45. Meng, J., Shi, W., Tan, H., Li, X., 'Cloudlet placement and minimum-delay routing in cloudlet computing', in Proceedings of the IEEE International Conference on Big Data Computing and Communication (BIGCOM), 2017, pp. 297–304.
46. Mijumbi, R., Serrat, J., Gorricho, J., Bouten, N., De Turck, F., Davy, S., 'Design and evaluation of algorithms for mapping and scheduling of virtual network functions', in Proceedings of the 1st IEEE Conference on Network Softwarization, 2015, pp. 1-9.
47. Munkres, J., 'Algorithms for the assignment and transportation problems', JSIAM, 5(1), 1957, pp. 32-38.
48. Murphree, J., 'Machine learning anomaly detection in large systems', IEEE AUTOTESTCON, 2016, pp. 1-9.
49. Mutlag, A. A., et al., 'MAFC: Multi-Agent Fog Computing Model for Healthcare Critical Tasks Management', Sensors, 20(7), 2020.
50. Muthucumaru, M., et al., 'Dynamic Mapping of a Class of Independent Tasks onto Heterogeneous Computing Systems', Journal of Parallel and Distributed Computing, vol. 59(2), 1999, pp. 107–131.
51. Nguyen, G., Dlugolinsky, S., Bobák, M., Tran, V., García, Á. L., Heredia, I., ... Hluchý, L. 'Machine Learning and Deep Learning frameworks and libraries for large-scale data mining: a survey.' Artificial Intelligence Review, 52(1), 2019, pp. 77-124.
52. Phansalkar, S., Ahirrao, S., 'Survey of Data Partitioning Algorithms for Big Data Stores', in Proceedings of the 4th IEEE International Conference on Parallel, Distributed and Grid Computing, pp. 163-168, 2016.
53. Ranadheera, S., Maghsudi, S., Hossain, E., 'Mobile Edge Computation Offloading Using Game Theory and Reinforcement Learning', arXiv:1711.09012, 2017.
54. Ranadheera, S., Maghsudi, S., Hossain, E., 'Computation Offloading and Activation of Mobile Edge Computing Servers: A Minority Game', IEEE Wireless Communications Letters, vol. 7(5), 2018.
55. Salehizadeh, S. M. A., Yadmellat, P., Menhaj, M. B., 'Local Optima Avoidable Particle Swarm Optimization', in proceedings of the IEEE Swarm Intelligence Symposium, pp. 16-21, 2009.
56. Smith, S., Tmova, S., Belta, C., Rus, D., 'Optimal path planning for surveillance with temporal-logic constraints', IJRR, vol. 30(14), pp. 1695-1708, 2011.
57. Sun, X., Ansari, N. (2016). 'EdgeIoT: Mobile edge computing for the Internet of Things.' IEEE Communications Magazine, 54(12), pp. 22-29.
58. H. Meireles Valadares, Clustering task assignment: an algorithm for time critical task assignment problems, UNIVERSIDAD POLITÉCNICA DE MADRID, 2017.
59. Wagner, G., Choset, H., 'Path planning for multiple agents under uncertainty', in ICAPS, 2017.
60. Wang, S., Zhao, Y., Xu, J., Yuan, J., Hsu, C.-H., 'Edge server placement in mobile edge computing', Journal of Parallel Distributed Computing, vol. 127, pp. 160–168, 2018.
61. Xing, H., Liu, L., Xu, J., Nallanathan, A., 'Joint Task Assignment and Resource Allocation for D2D-Enabled Mobile-Edge Computing', http://arxiv.org/abs/1902.10017v1, pp. 4193-4207, 2019.

62. Xu, Z., Liang, W., Xu, W., Jia, M., Guo, S., 'Efficient algorithms for capacitated cloudlet placements', IEEE Transactions on Parallel and Distributed Systems, vol. 27(10), pp. 2866–2880, 2016.
63. Yang, S., Li, F., Shen, M., Chen, X., Fu, X., Wang, Y., 'Cloudlet Placement and Task Allocation inMobile Edge Computing', IEEE Internet of Things Journal, vol. 6(3), 2019, pp. 5853-5863.
64. Yeh, W. C., Lai, C. M., Huang, Y. C., Cheng, T. W., Huang, H. P., Jiang, Y., 'Simplified Swarm Optimization for Task Assignment Problem in distributed computing system', in 2017 13th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD), IEEE, 2017, pp. 773-776.
65. Yu, W., et al., 'A Survey on the Edge Computing for the Internet of Things', IEEE Access, 2017, pp. 6900-6919.
66. Zhao, Z., Min, G., Gao, W., Wu, Y., Duan, H., Ni, Q. (2018). 'Deploying edge computing nodes for large-scale IoT: A diversity aware approach.' IEEE Internet of Things Journal, 5(5), pp. 3606-3614.