A Demand-driven, Proactive Tasks Management Model at the Edge

Anna Karanika
Dept. of Informatics and Telecommunications
University of Thessaly
Papasiopoulou 2-4, Lamia 35100 Greece
ankaranika@uth.gr

Panagiotis Oikonomou
Dept. of Informatics and Telecommunications
University of Thessaly
Papasiopoulou 2-4, Lamia 35100 Greece
paikonom@uth.gr

Kostas Kolomvatsos
Dept. of Informatics and Telecommunications
University of Athens
Panepistimiopolis, Ilisia 15784 Greece
kostasks@di.uoa.gr

Thanasis Loukopoulos
Dept. of Computer Science and Biomedical Informatics
University of Thessaly
Papasiopoulou 2-4, Lamia 35100 Greece
luke@dib.uth.gr

***Abstract*—Tasks management is a very interesting research topic for various application domains. Tasks may have the form of analytics or any other processing activities over the available data. One of the main concerns is to efficiently allocate and execute tasks to produce meaningful results that will facilitate any decision making. The advent of the Internet of Things (IoT) and Edge Computing (EC) defines new requirements for tasks management. Such requirements are related to the dynamic environment where IoT devices and EC nodes act and process the collected data. The statistics of data and the status of IoT/EC nodes are continuously updated. In this paper, we propose a demand- and uncertainty-driven tasks management scheme with the target to allocate the computational burden to the appropriate places. As the proper place, we consider the local execution of a task in an EC node or its offloading to a peer node. We provide the description of the problem and give details for its solution. The proposed mechanism models the demand for each task and efficiently selects the place where it will be executed. We adopt statistical learning and fuzzy logic to support the appropriate decision when tasks' execution is requested by EC nodes. Our experimental evaluation involves extensive simulations for a set of parameters defined in our model. We provide numerical results and reveal that the proposed scheme is capable of deciding on the fly while concluding the most efficient allocation.**

***Index Terms*—Edge computing, Tasks management, Probabilistic model, Kernel Density Estimator**

I. Introduction

Tasks offloading is a key research topic in Edge Computing (EC) and Internet of Things (IoT) if we consider the dynamic environment where nodes act and their heterogeneity. Efficient solutions should be provided that will allocate tasks to the available processing nodes to conclude the desired analytics. This becomes more important if we consider that any processing at the EC will limit the latency that end users enjoy. Hence, time sensitive applications can be easily served increasing the performance and the quality of services provided to end users. In addition, tasks processing at the edge can reduce the network traffic [30], driving data analytics towards geo-distributed processing, known as edge analytics [23], [27], [36], [41].

It becomes obvious that an efficient tasks management scheme can facilitate the delivery of analytics at the edge and the respective decision making. Tasks offloading was first applied to Mobile Cloud Computing (MCC) [11], [24], i.e., we can offload the computing tasks of mobile terminals to traditional cloud data centers where centralized computing and storage are the main features. However, in such cases there are some obstacles related to the experienced delay especially when communications are realized over a Wide Area Network (WAN). Additionally, the intense variability of the contextual information around the status of EC nodes, tasks and the collected data impose strict requirements for the effective conclusion of tasks allocation. For instance, heterogeneity heavily affects the decision making. These requirements should be met by any model trying to manage task at the edge of the network. In the relevant literature (see next Section), task allocation and scheduling originates in the management of a group of nodes. The allocation is, then, adopted to determine the assignment of each task to a node while scheduling mainly aims to the sequence of the execution for each task. The challenges are to **(C1)** maximize the performance and **(C2)** minimize the energy consumption, thus, maximizing the lifetime of the network. Multiple research efforts deal with centralized approaches, thus, the allocation and scheduling models suffer from the drawbacks reported in the literature for Cloud computing [14].

In this paper, we focus on the investigation of tasks offloading methodologies adopted to decide when a task should be executed locally or be offloaded to other peer nodes. We adopt a distributed approach, i.e., every EC node autonomously decides for tasks allocation reported to it. Users' mobility is taken into consideration to, eventually, support the decision making. Mobility affects the demand for the execution of tasks imposing spatio-temporal requirements in our model. Users' mobility also increases the complexity when trying to find out

978-1-7281-6932-3/20/$31.00 ©2020 IEEE

if a task will be kept locally and adds uncertainty in nodes' behaviour. Demand and contextual information is exchanged between nodes to support our mechanism. The contextual information is related to the status of nodes as well as the data present in them. Our strategic decision is to keep and execute locally tasks exhibiting a high demand and offload the remaining to nodes where also a high demand is observed for each of them. The intuition behind this is two fold: First, nodes save resources through the re-use of tasks execution framework and load balancing is conserved; Secondly, the latency experienced by users is minimized as highly demanded tasks are initiated to be executed immediately. Another source of complexity for the discussed problem is the need for a scalable approach. The number of users and tasks may be extensively high, thus, it would be difficult to apply any optimization model that requires increased time to produce outcomes. This, in combination with the need for real time responses, makes us to focus on a 'fast' technique that covers the uncertainty present into our problem, i.e., Fuzzy Logic (FL), together with incremental statistical learning that limits the required processing time. The following list depicts our contributions:

- We propose a georeferenced task management scheme where computation offloading is decided based on data present at every node and tasks demand.
- We adopt a *Fuzzy Logic Controller* (FLC) to indicate when a task should be offloaded or not, thus, we manage the uncertainty related to the discussed decision making.
- We provide an extensive experimental evaluation that reveals the pros and cons of the proposed approach. Our evaluation is performed for a set of metrics adopting real and synthetic traces.

Results indicate that our model is capable of supporting real time applications while exhibiting an increased performance for a large set of experimental scenarios.

The remaining paper is organized as follows. Section II reports on the related work and presents important research efforts in the field. In Section III, we discuss preliminary information and describe our problem while in Section IV, we present the proposed mechanism. Section V is devoted to the description of our experimental evaluation adopting a set of performance metrics. Finally, in Section VI, we conclude our paper giving our future research plans.

II. Related Work

EC could involve numerous nodes capable of interacting with IoT devices and execute a set of tasks. Tasks are 'dictated' by applications or directly by end users and can have the form of a processing activity, a query over the collected data and so on and so forth. EC extends the Cloud infrastructure offering numerous processing nodes close to end users, thus, limiting the latency they enjoy [18]. This advantage leads to the trend of offloading tasks to the edge infrastructure. It becomes natural to have recent studies dealing with computation offloading, i.e., tasks partitioning, tasks allocation, resource management and distributed execution [18]. In general, tasks offloading could be performed in two modes, i.e., full offloading and partial offloading [33]. In the former model, tasks should be executed as a whole no matter the location. For instance, we could adopt a model that delivers the appropriate place to offload the desired tasks based on various characteristics (tasks and nodes) [25]. The latter mode builds on the parallel execution of a set of sub-tasks possibly offloaded in different places. Apart from the distributed nodes, we should not forget that every EC node serves a set of users/IoT devices/applications. To increase EC nodes' performance we could incorporate into their decision making mechanisms models for joint tasks allocation, i.e., the allocation of tasks requested by different users/devices/applications [7]. Joint allocations is also the subject of [41] where the target is to minimize a tradeoff between the task execution time and mobile energy consumption. Resource sharing is considered as early as in [30], in which a greedy task dissemination algorithm was developed to minimize the completion time. In [15], a polynomial-time task assignment scheme is proposed for allocating tasks with inter-dependency towards achieving guaranteed latency-energy trade-offs. Machine Learning (ML) can play a significant role through the powerful models that can be adopted on the needs of EC nodes, tasks and their environment. For instance, in [7], the authors propose a strategy based on a Q-Learning algorithm. Another option is to study the tasks allocation problem as an optimization problem [10]. For instance, in [32], data transfer and computation in terms of monetary and time costs, with task deadlines guaranteed are studied. The problem becomes a maximization problem over the two coupled phases: data transfer and computation. However, its increased (it is an NP-hard problem) complexity mandates the use of a set of assumptions before we proceed with the final solution.

Various other models have been proposed for supporting the efficient tasks allocation. In [9], a dynamic, decentralized resource-allocation strategy based on evolutionary game theory is presented. The matching theory is adopted in [12], i.e., the model does not take into consideration the central Cloud in the Mobile Edge Computing (MEC) platform considering the autonomous nature of edge nodes. One coalition-game-based cooperative method to optimize the problem of task offloading is the subject of [37] while in [13], the authors present game-based strategies for the discussed problem to achieve the Nash equilibrium among mobile users. In [29], the authors discuss a model for computation offloading under a scenario of multi-user and multi-mobile edge servers that considers the performance of intelligent devices and server resources. The task scheduling part of the model is based on an auction algorithm by considering the time requirement of the computing tasks and the performance of the mobile edge server. In [35], the authors propose a device-to-device (D2D)- enabled multi-helper MEC system, in which a local user offloads its tasks to multiple helpers for cooperative computation. The model tries to minimize the latency by optimizing the local user's task assignment jointly with the time and rate for task offloading and results downloading, as well as the computation frequency for task execution.

The dynamicity of the environment also imposes requirements in the tasks offloading problem. For instance, to reduce energy while guaranteeing delay constraints for mobile applications, the authors of [38] propose an access control management architecture for a 5G heterogeneous network. Two algorithms are considered, i.e., an optimal static algorithm based on dynamic programming to get the exact solution and a two-stage online algorithm to adaptively obtain the current optimal solution in real time. In the majority of the relevant research efforts, the rarely changing network is treated as a stable environment, thus, 'static' algorithms are adopted e.g., integer linear programming approach [6], dynamic programming [34] and so on and so forth. Enhancements can increase the performance being adapted to the variations of the environment. For instance, a dynamic programming solution with randomization is presented in [26]. Hence, we are able to get an approximate solution and early formulate a time constrained offloading policy.

In [22], the authors propose the cooperation of Cloud computing and MEC in IoT. The offloading problem is solved through a branch and bound algorithm, a Mixed Integer Linear Programming (MILP) scheme and an Iterative Heuristic MEC Resource Allocation (IHRA) algorithm to make the offloading decision dynamically. The authors of [16] consider an estimator for predicting the total processing duration of each task on each candidate node using linear regression. Another estimator is proposed in [20]. It is responsible to predict the round trip time (RTT) based on the network's characteristics. Then, the estimated RTT is 'bounded' with other parameters (e.g., energy consumption) to decide when and where to offload tasks.

Tasks offloading is studied with the presence of Software Defined Networking (SDN) and virtualized resources [21]. The optimality of the decision is related to the local or remote task computation, the selection of the appropriate node and the selection of the appropriate path for the offloading. In [19], the authors study the flexible compute-intensive task offloading to a local cloud trying to optimize energy consumption, operation speed, and cost. In [1], a model based on the Optimal Stopping Theory is adopted to deliver the appropriate time to offload data and tasks to an edge server. The challenge is to to determine the best offloading strategy that minimizes the expected total delay. Finally, in [5], the authors consider unmanned vehicles (i.e., Unmanned Aerial Vehicles - UAVs) and propose a framework enabling optimal offloading decisions as a function of network and computation load parameters and current state. The optimization is formulated as an optimal stopping time problem over a Markov process.

III. Preliminaries & Problem Formulation

We consider an EC scenario where a set of N nodes, $\mathcal{N} = \{n_1, n_2, \ldots, n_N\}$, are available. EC nodes are 'connected' with a number of IoT devices being responsible to collect and store data while performing the requested tasks. Nodes are also connected with the Cloud to transfer data for 'long-term' processing. They act between the IoT and Cloud infrastructures being capable of hosting and executing various tasks for limiting the latency that end users enjoy. Locally, at every node, a dataset is formulated over the data reported by IoT devices. These data become the subject of various processing activities that may requested by end users or applications. Due to the limited computational resources of EC nodes (compared to the Cloud infrastructure), only a part of the collected data should be locally stored. The remaining data are transferred to the Cloud for further processing. We have to notice that the study of the methodology adopted to select the data that will be locally stored is beyond the scope of this paper.

Furthermore, EC nodes are able to execute a set of tasks. Tasks may have the form of queries over the available data, the calculation of statistical metrics or the execution of various processing activities like the delivery of a local ML model. In any case, one can define before hand the categories of tasks that can be executed in every EC node. The execution of tasks aims at generating knowledge locally, thus, to make EC nodes capable of efficiently reacting in users' requests. Without loss of generality, we consider that nodes may support the same number of tasks, i.e., E. At a time instance t, a node may have to execute a subset of the predefined tasks. Tasks are placed in a queue and wait for their execution. The throughput of each node depends on its computational capabilities and affects the size of the queue. In this effort, we consider that the corresponding queue can host as many tasks as we want without any limit. We have to notice that a task may be requested by multiple users/applications, thus, every node should execute it repeatedly in no consecutive time instances. We also consider that tasks exhibit specific characteristics like their complexity (the steps and resources required to be spent for their execution), priority (depending on the critically of the application asking the execution of the task) or their sub-tasks (if the task is separated based on an appropriate algorithm). These characteristics may be adopted to build more complex tasks management mechanisms applying a strategical decision making. For instance, high priority tasks may be executed first, however, securing that starvation effects are eliminated. We have to notice that the study of these research issues is beyond the scope of this work.

Tasks demand is realized by the number of users/devices asking for their execution. Requests are delivered to EC nodes through specific interfaces and affect the demand. For simplicity, we consider that the demand for a task is realized by the number of users asking for its execution in a specific node at a specific time instance (i.e., at t). In addition, users (IoT devices) may move while requesting for services/applications, thus, their mobility affects tasks demand in 'neighbour' EC nodes. Every node maintains a vector, i.e., the *Tasks Demand Vector* (TDV), $TDV = \{e_1^t, e_2^t, \ldots, e_M^t\}$ which depicts the demand for each task at time instance t. EC nodes at specific epochs, i.e., $t = 1, 2, \ldots$, update the corresponding TDV as users are moving in an area. EC nodes should decide which tasks will be locally executed or offloaded to another peer node. The aim is to keep the execution locally for popular

tasks eliminating the time required for their conclusion. No popular tasks can be offloaded to other nodes. In addition, when executing popular tasks, EC nodes may adopt techniques like incremental models or caching to facilitate their execution. By offloading non popular tasks, nodes may save resources as they are not benefited from re-using previous outcomes. For instance, consider the case of a task waiting for execution in the queue. At t, the hosting EC node updates the TDV and sees that there is limited demand for this task. Hence, it may decide to offload the task to another node that may have increased demand for it (incremental models and caching may be adopted to deliver the final result) paying the communication cost (for sending the task and getting the response) and the time for waiting the final outcome. In this paper, we focus on the study on how we can decide which tasks will be kept locally to be executed based on the TDVs. The important is that our model takes into consideration not only the demand for tasks locally at an EC node but also in its peers concluding a complete georeferenced tasks management scheme. The study on the decision on where (peer selection) we can offload a task is available in [17]. For this decision, the data present at peers, the load, etc can be adopted to secure the efficient execution of the task.

IV. Georeferenced Tasks Demand Management

A. Tasks Demand Indicator

EC nodes, at predefined intervals, exchange their TDVs to spread the demand information for the available tasks. The incoming TDVs could exhibit different information, e.g., the ith task may be requested at n_j but not at n_k. In our work, we are interested in the 'locality' of the demand, i.e., the demand in neighbour nodes. The rationale is that mobile users will not be capable of performing distant hand overs in consecutive time instances. Hence, EC nodes keep the TDVs only for peers being in a distance ρ. The limitation of the approach (for local demand modelling) is that nodes should monitor a high number of users/devices to be able to calculate the local demand for the available tasks. This could lead to a complicated processing and increase the complexity of nodes' 'reasoning' when the number of users is very high. However, a solution to alleviate the complexity of the required processing could be the grouping of users in the range of each node as proposed in [2].

Let TDVs be $\{TDV_1^t, TDV_2^t, \ldots, TDV_D^t\}$. (the index of each vector does not correspond to the index of a peer node but to the order of the received vectors) with $D \leq N$. Actually, nodes apply a sliding window approach and maintain the last W historical TDVs, i.e., $\left[\left\{TDV_1^1, TDV_2^2, \ldots, TDV_D^3\right\}\right]$, $\left[\left\{TDV_1^2, TDV_2^2, \ldots, TDV_D^2\right\}\right]$, $\ldots, \left[\left\{TDV_1^{Ws}, TDV_2^W, \ldots, TDV_D^W\right\}\right]$. This way, they can have a view on the 'trends' of tasks demand at each peer. Our aim is to define a decision making mechanism that at each epoch t will result the tasks that will be kept locally while the remaining will be offloaded in peer nodes. For this, we define a function $f()$ being responsible to deliver a ranked list of the tasks waiting for execution in the corresponding queue. $f()$ gets as input the local and the incoming TDVs and delivers the ranked $\{e_i\}$.

Let us focus on a specific EC node n_j. n_j applies a monitoring scheme for updating the local TDV and receives TDVs from its peers. It tries to estimate the demand for each task locally and in peers. When receiving the incoming TDVs, n_j detects the common tasks for which it should estimate the demand. n_j concludes the *Local Demand Indicator* (LDI) and the *Group Demand Indicator* (GDI). LDI and GDI are metrics that will affect the final ranking as delivered by $f()$. Formally, LDI and GDI are defined as the number of users requesting a specific task, i.e., $LDI_i = e_i^{n_j}$ & $GDI_i = e_i^{n_k}, \forall k, k \neq j$. We adopt a simple scheme and 'connect' LDI and GDI with the probability of having the demand for a task over a pre-defined threshold T. The LDI is based on the local observations while the GDI is concluded over D nodes.

As mobility patterns are unknown and we are not aware of the distribution of the demand in each node, n_j applies the widely known *Kernel Density Estimator* (KDE) [31] to derive the demand distribution. Our intention, when applying a statistical learning process (i.e., the KDE), is to detect the hidden statistics of the mobility patterns as exposed by the demand for each task. The KDE can be adopted to deliver the *probability density function* (pdf) of the unknown distribution of the demand for a task $\mathbf{d} \in TDV_i$. The statistical learning process is applied on top of $\mathbf{d}_1, \mathbf{d}_2, \ldots, \mathbf{d}_W$ which represent the historical demand values. We adopt an incremental estimation of $g(\mathbf{d}_i)$ to derive the hidden statistics of the unknown distribution. The estimation of the expected value $\mathbb{E}[\mathbf{d}] = \int_{\mathbb{R}} \mathbf{d}g(\mathbf{d})\mathbf{d}\mathbf{d}$ gives an insight of current demand observations. $\mathbb{E}[\mathbf{d}]$ is estimated through the adoption of an approximation applied over a set of measurements and, also, over $g(\mathbf{d})$'s approximation [4]. If we consider $\mathbf{d}[1], \mathbf{d}[2], \ldots, \mathbf{d}[W]$, as the demand values, our problem is to estimate $g(\mathbf{d})$ on-the-fly. This requires the estimation of $\mathbb{E}[\mathbf{d}]$ in limited time. We define the cumulative KDE through the following equation: $\hat{g}(\mathbf{d}; k) = \frac{1}{k \cdot h} \sum_{m=1}^{k} K\left(\left|\frac{\mathbf{d} - \mathbf{d}[k-m]}{h}\right|\right)$. $h > 0$ represents the bandwidth of the adopted Kernel function $K(\cdot)$ that is is symmetric and integrates to unity. For $K(\cdot)$, we adopt the Gaussian kernel, $(K(z) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}z^2\right))$. Additionally, $\hat{g}(\mathbf{d}; k)$ is incrementally estimated through its previous estimation $\hat{g}(\mathbf{d}; k-1)$ and the current distance $\mathbf{d}[k]$. More specifically, if $x[k] = |\frac{\mathbf{d} - \mathbf{d}[k]}{h}|$, we take through calculations that,

$$\begin{aligned} \hat{g}(\mathbf{d}; k) &= \frac{1}{kh}\Big(\sum_{m=1}^{k-1} K(x[k-m]) + K(x[t])\Big) \\ &= \frac{k-1}{kh}\hat{g}(\mathbf{d}; k-1) + \frac{1}{kh}K(x[k]) \end{aligned} \quad (1)$$

When $\mathbf{d}[k]$ is received, we can easily estimate $\hat{g}(\mathbf{d}; k)$ by adopting the proposed incremental approach. The proposed approach estimates $\hat{g}(\mathbf{d}; k)$ through $\hat{g}(\mathbf{d}; k-1)$ plus the Kernel function applied on the difference $|\mathbf{d} - \mathbf{d}[k]|$. By adopting the described incremental KDE estimation of $\hat{g}(\mathbf{d}; k)$, we approximate $\mathbb{E}[\mathbf{d}; k]$ by $\mathbb{E}[\mathbf{d}; k-1]$. Integrating in both sides of

the Eq(1), we obtain $\mathbb{E}[\mathbf{d};k] = \frac{k-1}{k}\mathbb{E}[\mathbf{d};k-1] + \frac{1}{k}\mathbf{d}[k]$. LDI is concluded over the local observations for each task, i.e., LDI_{e_i}. Based on the KDE, we get a single probability, i.e., $LDI_{e_i} = P(\mathbf{d}_{e_i} > T)$ where $\mathbf{d}_{e_i}$ depicts the random variable depicting the demand for task e_i.

GDI is calculated for all peers, i.e., we consider D GDI realizations (D realizations of the above described statistical learning process), i.e., $p_i = GDI_{e_i}^{n_j} = P(\mathbf{d}_{e_i}^{n_j} > T)$. In this case, we face the problem of combining multiple probabilities reported by different sources. For combining them, we adopt the geometric mean [8], i.e., a geometric opinion pool; this metric does not increase by orders of magnitude affected by a single data point and performs better with small samples. The geometric mean combines our values with a product instead of a sum and sees each data point as a scaling factor. Actually, this is the strong assumption behind the use of the geometric mean, i.e., data can be interpreted as scaling factors. The following equation stands true:

$$p_i = \frac{\prod_{k=1}^{D} p_k^{w_k}}{\prod_{k=1}^{D} p_k^{w_k} + \prod_{k=1}^{D}(1-p_k)^{w_k}} \quad (2)$$

where w_k are weights usually taken as $w_k = \frac{1}{D}$. In our case, weights are affected by p_k, i.e., $w_k = \frac{p_k}{\sum p_k}$. This means that the node exhibiting the highest probability of exceeding affects more the corresponding weight. This strategy makes us to be more 'sensitive' in scenarios where there is an increased probability of facing a high demand for a task. As explained, in these cases, we try to host popular tasks to specific nodes (at nodes where the increased demand is observed), thus, to gain benefits from their 'repetitive' execution. We have to notice that the adoption of the specific strategy for weights definition is not contradictory with the use of the geometric mean as we target to smoothly 'aggregate' all the available probabilities considering them as scaling factors. In general, the presence of a few extremely low or high values has no considerable effect on the geometric mean, thus, we can easily incorporate into the envisioned processing the desired focus on the probabilities that exhibit a high weight without resulting extreme outcomes.

B. Our Uncertainty Driven Decision Making Model

For each task, as described above, we calculate two probabilities of having the demand over the threshold, i.e., the LDI_{e_i} and the GDI_{e_i}. Both of them depict EC nodes' local knowledge about the estimation of the demand of a task. As it is difficult to be aware and define specific thresholds for both metrics to support efficient decision making and aiming at the management of the ambient uncertainty, we adopt an FLC to deliver a value over which the final decision is made. In FL systems, the objects of discourse are associated with information which is, or is allowed to be, incomplete, partially true or partially possible. FL deals with incomplete information and provides knowledge representation models (i.e., Fuzzy Set Theory) through which an entity can automatically take decisions. FL principles express human expert knowledge and enable the automated interpretation of the results. The local execution of a task or the offloading is based on our FLC that is a non-linear mapping between l inputs $u_i \in Ui, i = 1, \ldots, l$ and m outputs $y_i \in Y_i, i = 1, \ldots, m$. In this paper, we adopt two inputs, i.e., LDI_{e_i}, GDI_{e_i} and a single output, i.e., the Task Offloading Indicator (TOI) TOI_{e_j}. The knowledge base of our FLC consists of a set of rules defined in the following form: R_j: IF u_{1j} is A_{1j} AND/OR u_{2j} is A_{2j} AND/OR ... AND/OR u_{lj} is A_{lj} THEN y_{1j} is B_{1j} AND ... AND y_{mj} is B_{mj}, where R_j is the jth fuzzy rule, $u_{ij}(i = 1, \ldots, l)$ are the inputs of the jth rule, $y_{kj}(k = 1, \ldots, m)$ are the outputs and A_{ij}, B_{kj} are membership functions usually associated by linguistic terms. Without loss of generality, we assume that inputs and output are in the unity interval. When $LDI_{e_i} \to 1$ means that there is an increased demand for the task e_i while $LDI_{e_i} \to 0$ depicts the case where a limited number of users express interest for the specific task. $GDI_{e_i} \to 1$ depicts an increased demand for the ith task as exposed by peer nodes (the opposite stands for $GDI_{e_i} \to 0$). Concerning the output TOI_{e_i}, a value close to zero depicts a 'keep locally' decision in contrast when a value close to unity is met. For inputs and the output, we consider three linguistic values: Low, Medium, High. A Low value represents that the fuzzy variable takes values close to the lower limit while a High value depicts the case where the variable takes values close to the upper level. In addition, we consider triangular membership functions as they are widely adopted in the literature. The proposed FLC receives values for the two inputs, it fuzzifies them and, accordingly, proceeds with the inference process. The inference process involves a set of fuzzy rules that result the best possible value for the output TOI_{e_j}. These rules are defined by experts and incorporate a human view on the described decision process. In Table I, we present the adopted FL rule base.

TABLE I
FUZZY LOGIC RULE BASE

No	LDI_{e_i}	GDI_{e_i}	TOI_{e_i}
1	Low	Low or Medium	Low
2	Low	High	High
3	Medium	Low	Low
4	Medium	Medium or High	Medium
5	High	Low or Medium	Low
6	High	High	Medium

The final step is the de-fuzzification process to derive the final TOI_{e_j}. n_j produces a sorted list of TOI_{e_j}s and the 'last-k' tasks are offloaded to peer nodes as described in [17].

V. EXPERIMENTAL EVALUATION

A. Performance Indicators & Setup

We report on the performance of our model concerning its ability of making correct decisions when deciding the execution of an incoming task. We also focus on the time requirements to perform the final allocation in order to reveal if our model is capable of deciding in real time, thus, supporting time critical applications. We evaluate our scheme through

an extensive set of simulations involving a high number of tasks. We consider that such tasks are 'generated' in various nodes into our network. To simulate users' activity, we use the dataset provide by [39]. This dataset describes real-world QoS evaluation results from 142 users on 4500 Web services over 64 different time slices.

The performance of the proposed mechanism is evaluated by a set of metrics. We adopt metrics in the following axes: (i) the number of correct decisions Δ. To measure the number of correct decisions, we assume a loss function $\lambda(C, R_s)$ that measures the cost of executing a task s locally and the cost of offloading a task to another peer node. The binary variable C is equal to unity when the task is executed locally otherwise is equal to zero. Three different metrics are adopted as elements to set the final value of R_s. The initiation time (IT_s), i.e., task's execution starting time, the response time (RT_s), i.e., the time spent to deliver the result of task t from a node to a specific user and the demand indicator d_s. If s is executed locally then $\lambda(C = 1, IT_s)$ =0 and $\lambda(C = 1, R_s)$ otherwise $\lambda(C = 0, IT_s)$ = MGT and $\lambda(C = 0, R_s)$ =RST. MGT and RST are the migration time and the response time, respectively. IT_s and RT_s are selected to be uniformly distributed in the interval (0,1]. Regarding d_s, we adopt the exponential function to produce the corresponding values as Eq(3) dictates. $\lambda(C, d_s)$ results values close to zero when the demand indicator of s is high. On the other hand, when the demand of s is low, $\lambda(C, d_s)$ approaches unity.

$$\lambda(C, d_s) = e^{-d_s} \tag{3}$$

$$Cost_t(C) = \sum_{r \in R}^{|R|} (C, r), R \in \{IT_s, RT_s, d_s\} \tag{4}$$

When a user requests a task s from a specific node, the cumulative cost is calculated for both $C = 1$ and $C = 0$ as imposed by Eq(4).

A decision is considered as correct when one of the equations Eq(5), Eq(6) holds true. For instance, Eq(6) indicates that a decision is correct when the overall cost of executing a task locally is higher than offloading a task to another peer node and at the same time the FLC decides that the node should offload the task.

$$Cost_s(C = 1) < cost_s(C = 0) \&\& \ FLC \rightarrow local\ execution \tag{5}$$

$$Cost_s(C = 1) > cost_s(C = 0) \&\& \ FLC \rightarrow offloading\ action \tag{6}$$

(ii) the average time τ required to take a decision. τ is measured for every task as the time spent (CPU time) by the system deciding if a task should be offloaded or not. For this reason, τ is calculated as the sum of (a) the time spent to estimate KDE and (b) the time spent for FLC to produce the final TOI. We perform a set of experiments for different N, W, E and T. The number of EC nodes is set to $N \in \{50, 100, 500, 1000\}$. We adopt $W \in \{10\%, 50\%, 100\%\}$, i.e., different sliding window sizes to measure the effect on Δ and τ. The total number of tasks requested by the users is set to $E \in \{5000, 10000, 50000, 100000\}$. The probability of having the demand for a task over a pre-defined threshold is set to $T \in \{0.5, 0.7\}$. In total, we conduct 100 iterations for each experiment and report our results for the aforementioned metrics. Experiments where conducted on a Linux server with two 6-core Intel Xeon E5-2630 CPUs running at 2.3GHz.

B. Performance Assessment

Initially, we evaluate of our model in terms of Δ. The number of tasks increases from a baseline value equal to 5000 up to 2x, 10x and 20x. Having as a base the realistic dataset presented in [39], we extend it and create additional datasets to experiment with different number of tasks and nodes. In Fig. 1, we plot the average number of correct decisions Δ for various combinations of and W. In this set of experiments, we assume that our model offloads the top $k = 10\%$ of the incoming tasks. As the number of tasks increases, the same stands for Δ as well. When we focus on an extreme value for the number of tasks, i.e., 100000, the number of correct decisions is close to 90% in all the experimental scenarios. This is significant for our model as it is capable of correctly allocating the requested tasks, thus, it increases the performance of nodes. It is worth mentioning that a small performance degradation is observed as the sliding window size increases. This stems from the fact that the solidity of the data located in a node is affected by the time. The demand of a task may significantly vary in different time units.

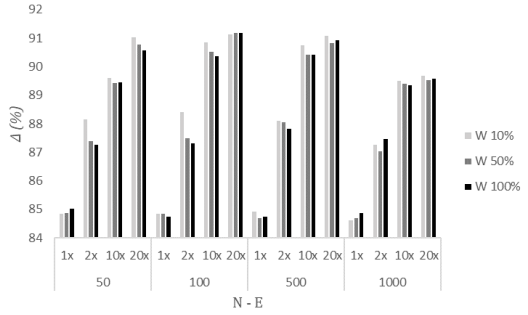

Fig. 1. Percentage of correct decisions; $k = 10\%$

Fig. 2 & Fig. 3 plot the average number of correct decisions for different values of the threshold T. It should be noted for $T = 0.7$, the average Δ ranges from 90% to 99% which is very close to the maximum possible outcome. As demonstrated in both figures, our model produces a low number of correct decisions as the number of nodes increases. This performance degradation is explained due to the high number of peer nodes available to be selected by our model. Comparing the results for the T values, we can observe that when $T = 0.7$, we have a considerable higher number of correct decisions up to 20%. The reason behind this is that a low number of tasks

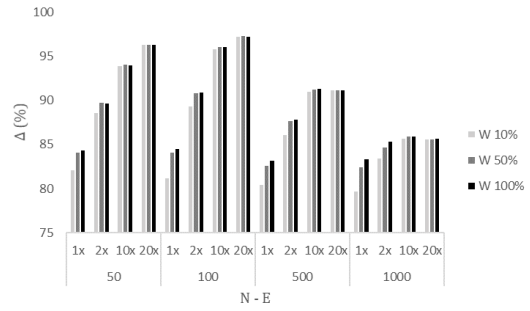

Fig. 2. Percentage of correct decisions; $T = 0.5$

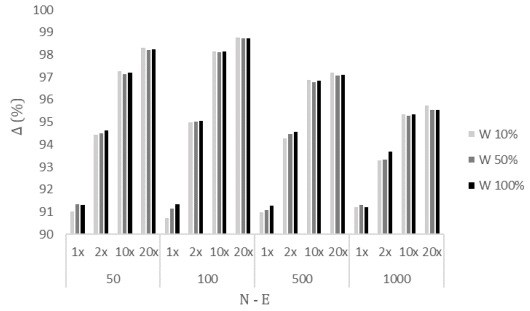

Fig. 3. Percentage of correct decisions; $T = 0.7$

are offloaded being kept for execution locally. Comparing the results from above described figures, we observe that a threshold close to $T = 0.7$ archives the best performance instead of using a threshold close to $T = 0.5$. Additionally, it is preferred to offload the top $k = 10\%$ tasks than to keep them locally. This is reasonable since a small threshold indicates that the majority of tasks should be offloaded.

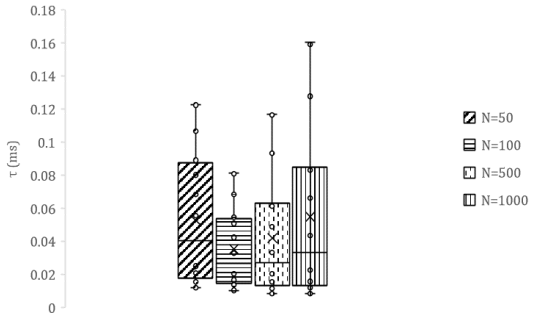

Fig. 4. Execution time

In Fig. 4, we present our evaluation results related to τ. Recall that τ depicts the time required by our model to deliver the final decision. We observe that our scheme is efficient managing to derive the final decision in short time, i.e., below 0.18 ms in the worst case. This leads to a throughput of the management of (approx.) 5500 tasks per second. If we focus on the average case, the throughput increases. The reason is that the mean required time is around 0.05 ms. The mean time is similar for scenarios where N increases form 50 to 1000. However, we observe a significant difference in the statistical deviation of the realized time values. In any case, the proposed approach is characterized by scalability as the increased number of tasks does not add a high amount of time to conclude the desired processing. Hence, the proposed model can be adopted to respond in real time to the requests for the execution of tasks at the edge infrastructure.

We compare the performance of our model with the scheme presented in [3] where the authors propose a task scheduling algorithm (ETSI) that is based on a heuristic. This heuristic delivers the final outcome based on the remaining energy, the distance from the edge of the network and the number of neighbours calculating the rank of each node. The node with the lowest ranking is selected for the final allocation. Fig. 5 presents our comparison results for Δ. We observe that our model clearly outperforms ETSI no matter the experimental scenario we adopt. ETSI manages to result a limited number of correct decisions related to the offloading of tasks. The highest realization of Δ is 45% (approximately) with the mean and median be around 25%. The lowest value for Δ in our evaluation scenarios is around 80%.

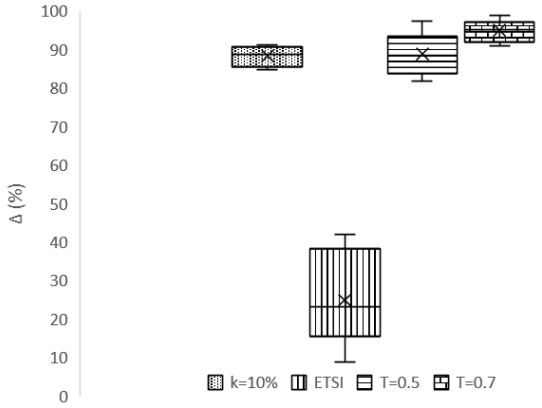

Fig. 5. Comparative assessment

VI. Conclusions & Future Work

In this paper, we tackle the problem of task allocation/offloading in the presence of a heterogeneous IoT environment. IoT devices are connected with EC reporting the collected data for further processing. Over these data hosted at the EC nodes, multiple users can request the execution of tasks. The demand for each task may be altered over time affected not only by the desires of end users but also by their mobility. For this reason, we propose a georeferenced task management scheme that decides when a task should be offloaded or not. This scheme incorporates a FLC that delivers the realization of an indicator over which the final decision is made. The discussed indicator shows the efficiency of the allocation either decided to be locally or offloaded to peer nodes. Through simulations and using a real dataset, it is concluded that our scheme can achieve a high amount of

correct decisions (up to 98%) while it is efficient to derive the final solution in short term. In the first places of our future agenda is to conduct our experiments in a real world IoT environment and to apply Optimal Stopping Theory to identify the appropriate time to offload a task. Furthermore, we will enhance our model with a scheduling component in order to minimize tasks execution time, maximize throughput and satisfy QoS constraints defined by end users.

ACKNOWLEDGMENT

This research received funding from the European's Union Horizon 2020 research and innovation programme under the grant agreement No. 745829.

REFERENCES

[1] Alghamdi, I., Anagnostopoulos, C., Pezaros, D., 'Time-Optimized Task Offloading Decision Making in Mobile Edge Computing', IEEE Wireless Days, 2019.

[2] Anagnostopoulos, C., Hadjiefthymiades, S., Kolomvatsos, K., 'Time Optimized User Grouping in Location Based Services', Elsevier Computer Networks (COMNET), vol. 81, pp. 220-244, 2015.

[3] 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.

[4] Bishop, C. M., 'Pattern Recognition and Machine Learning', Springer, 2006.

[5] Callegaro, D., Levorato, M., 'Optimal Computation Offloading in Edge-Assisted UAV Systems', in Proceedings of the IEEE GLOBECOM, 2018.

[6] Cuervo, E., Balasubramanian, A., Cho, D.-K., Wolman, A., Saroiu, S., Chandra, R. and Bahl, P., 'Maui: Making Smartphones Last Longer with Code Offload', in Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services, 2010, pp. 49–62.

[7] Dab, B., Aitsaadi, N., Langar, R., 'Q-Learning Algorithm for Joint Computation Offloading and Resource Allocation in Edge Cloud', in Proceedings of the IFIP/IEEE Symposium on Integrated Network and Service Management, 2019.

[8] Dietrich, F., List, C., 'Probabilistic opinion pooling', The Oxford Handbook of Probability and Philosophy, 2014, 10.1093/oxfordhb/9780199607617.013.37.

[9] Dong, C., Wen, W., 'Joint Optimization for Task Offloading in Edge Computing: An Evolutionary Game Approach', Sensors, 19, 2019.

[10] Du, W., Lei, T., He, Q., Liu, W¿, Lei, Q., Zhao, H., Wang, W., 'Service Capacity Enhanced Task Offloading and Resource Allocation in Multi-Server Edge Computing Environment', in Proceedings of the IEEE International Conference on Web Services, 2019.

[11] Fernando, N., Loke, S. W., Rahayu, W., 'Mobile cloud computing: A survey', Future Generation Computer Systems, 2013, 29, 84–106.

[12] Gu, B., Chen. Y., Liao, H., Zhou, Z., Zhang, D., 'A Distributed and Context-Aware Task Assignment Mechanism for Collaborative Mobile Edge Computing', Sensors, 2018, 18–2423.

[13] Jošilo, S., Dan, G., 'Selfish Decentralized Computation Offloading for Mobile Cloud Computing in Dense Wireless Networks', IEEE Transactions on Mobile Computing, 2019, 18, 207–220.

[14] Islam, M. M., Morshed, S., Goswami, P.,'A survey on its limitations and potential solutions', International Journal of Computer Science Issues (IJCSI), 10(4), 2013.

[15] Kao, Y., Krishnamachari, B., Ra, M., Bai, F., 'Hermes: Latency optimal task assignment for resource-constrained mobile Computing', IEEE Transactions on Mobile Computing, 16(11), pp. 3056–3069, 2017.

[16] Kim, K., Lynskey, J., Kang, S., Hong, C., 'Prediction Based Sub-Task Offloading in Mobile Edge Computing', in Proceedings of the International Conference on Information Networking, 2019.

[17] Kolomvatsos, K., Anagnostopoulos, C., 'Multi-criteria Optimal Task Allocation at the Edge', Elsevier Future Generation Computer Systems, 93, 2019, pp. 358-372.

[18] Lin, L., et al., 'Computation Offloading towards Edge Computing', Proceedings of the IEEE, 107(8), 2019.

[19] Lin, Pankaj, S., Wang, D., 'Task Offloading and Resource Allocation for Edge-of-Things Computing on Smart Healthcare Systems', Computers and Electrical Engineering, 72, 2018, pp. 348-360.

[20] Messaoudi, F., Ksentini, A., Bertin, P., 'On Using Edge Computing for Computation Offloading in Mobile Network', In Proceedings of the IEEE Global Communications Conference, 2017.

[21] Misra, S., Saha, N., 'Detour: Dynamic Task Offloading in Software-Defined Fog for IoT applications', IEEE Journal on Selected Areas in Communications, PP(99), 2019.

[22] Ning, Z., Dong, P., Kong, X., Xia, F., 'A Cooperative Partial Computation Offloading Scheme for Mobile Edge Computing Enabled Internet of Things', IEEE Internet of Things Journal, 6(3), 2019.

[23] Q. Pu, G. Ananthanarayanan, P. Bodik, S. Kandula, A. Akella, P. Bahl, I. Stoica, "Low latency geo-distributed data analytics", Proc. ACM Conf. Special Interest Group Data Commun., pp. 421-434, 2015.

[24] Sanaei, Z., Abolfazli, S., Gani, A., Buyya, R., 'Heterogeneity in mobile cloud computing: taxonomy and open challenges', IEEE Communication Surveys Tutorial, 2014, 16, 369–392.

[25] Sardellitti, S., Scutari, G., Barbarossa, S., 'Joint optimization of radio and computational resources for multicell mobile edge computing', IEEE Transactions on Signal and Information Processing over Networks, 1(2), pp. 89–103, 2015.

[26] Shahzad, H., Szymanski, T. H., 'A Dynamic Programming Offloading Algorithm Using Biased Randomization', in Proceedings of the 9th IEEE International Conference on Cloud Computing, 2016, pp. 960–965.

[27] M. Satyanarayanan et al., "Edge analytics in the Internet of Things", IEEE Pervasive Comput., 14(2), pp. 24-31, 2015.

[28] W. Shi, J. Cao, Q. Zhang, Y. Li, L. Xu, "Edge computing: Vision and challenges", IEEE Internet Things J., 3(5), pp. 637-646, 2016.

[29] Sheng, J., Hu, J., Teng, X., Wang, B., Pan, X., 'Computation Offloading Strategy in Mobile Edge Computing', Information, 10, 191, 2019.

[30] Shi, C., Lakafosis, V., Ammar, M., Zegura, E., 'Serendipity: Enabling remote computing among intermittently connected mobile devices', in Proceedings of the ACM International Symposium on Mobile Ad Hoc Networking and Computing, 2012.

[31] Wand, M. P., Jones, M. C., '*Kernel Smoothing*', Chapman and Hall, 1995.

[32] Wang, L., Jiao, L., Kliazovich, D., Bouvry, P., 'Reconciling Task Assignment and Scheduling in Mobile Edge Clouds', in Proceedinsg of the IEEE 24th International Conference on Network Protocols, 2016.

[33] Wang, Y., Sheng, M., Wang, X., Wang, L., Li, J., 'Mobile edge computing: partial computation offloading using dynamic voltage scaling', IEEE Transactions on Communications, 64(10), pp. 4268–4282, 2016.

[34] Wu, H. and Wolter, K., 'Software Aging in Mobile Devices: Partial Computation Offloading as a Solution', in 2015 IEEE International Symposium of Software Reliability Engineering Workshops, pp. 125–131, 2015.

[35] Xing, H., Liu, L., Xu, J., Nallanathan, A., 'Joint Task Assignment and Resource Allocation for D2D-Enabled Mobile-Edge Computing', IEEE Transactions on Communications, 67(6), pp. 4193–4207, 2019.

[36] S. Yi, Z. Hao, Z. Qin, Q. Li, "Fog computing: Platform and applications", Proc. 3rd IEEE Workshop Hot Topics Web Syst. Technol., pp. 73-78, 2015.

[37] Zhang, T., 'Data Offloading in Mobile Edge Computing: A Coalition and Pricing Based Approach', IEEE Access, 2018, 6, 2760–2767.

[38] Zhang, Z., Wu, J., Chen, L., Kiang, G., Lam, S., 'Computation Result Reusing for Mobile Edge Computing', The Computer Journal, 62(10), 2019, pp. 1450–1462.

[39] Zibin Zheng, Yilei Zhang, and Michael R. Lyu, "Investigating QoS of Real- World Web Services", IEEE Transactions on Services Computing, 7(1), pp.32-39, 2014.

[40] Zhou, W., Fang, W., Li, Y., Yuan, B., Li, Y., Wang, T., 'Markov Approximation for Task Offloading and Computation Scaling in Mobile Edge Computing', Mobile Information Systems, 2019.

[41] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, J. Zhang, "Edge intelligence: Paving the last mile of artificial intelligence with edge computing", Proc. IEEE, 107(8), 2019.