Jordanian Journal of Computers and Information Technology (JJCIT), Vol. 09, No. 02, June 2023.

AGENT BASED APPROACH FOR TASK OFFLOADING IN EDGE COMPUTING

Hossein Morshedlou[1] and Reza Vafa Shoar[2]

(Received: 7-Jan.-2023, Revised: 24-Mar.-2023 and 9-Apr.-2023, Accepted: 10-Apr.-2023)

ABSTRACT

Due to limited resource capacity in the edge network and a high volume of tasks offloaded to edge servers, edge resources may be unable to provide the required capacity for serving all tasks. As a result, some tasks should be moved to the cloud, which may cause additional delays. This may lead to dissatisfaction among users of the transferred tasks. In this paper, a new agent-based approach to decision-making is presented about which tasks should be transferred to the cloud and which ones should be served locally. This approach tries to pair tasks with resources, such that a paired resource is the most preferred resource by the user or task among all available resources. We demonstrate that reaching a Nash Equilibrium point can satisfy the aforementioned condition. A game-theoretic analysis is included to demonstrate that the presented approach increases the average utility of the users and their level of satisfaction.

KEYWORDS

Edge computing, Task offloading, Nash equilibrium, Agent, User satisfaction.

1. INTRODUCTION

Offloading is a solution that enhances the capabilities of mobile systems by migrating computations to more resourceful and powerful nearby devices, such as edge nodes, fog nodes and cloudlet or base stations. Nearest nodes, like edge nodes, are the first choice for offloading purposes, because offloading to edge nodes alleviates congestion in cellular networks. Edge node resource capacity may not always allow for the handling of computation-intensive tasks and large data storage from mobile devices. Therefore, such tasks must be transferred to the cloud. However, especially in peak times, due to many offloaded tasks, edge resources may not be able to provide the needed capacity for all tasks. Therefore, inevitably, some tasks should be transferred to the cloud, which may result in additional delays. This may lead to dissatisfaction among users of the transferred tasks. This paper proposes a decentralized approach for task offloading without a central decision-maker component. The proposed approach incorporates user satisfaction level in decision-making about which tasks should be transferred to the cloud and which ones should be served on edge nodes locally. This approach tries to pair tasks with resources, such that a paired resource is the most preferred one by the user or task among all available resources. We show that reaching an NE point can satisfy the mentioned condition. Additionally, a game-theoretic analysis is given to demonstrate how the proposed approach raises the total satisfaction level of users and the acceptability of results by rational users. To the best of the authors' knowledge, this is the first work in the literature that addresses task-offloading strategy optimization based on user satisfaction level. The satisfaction of different users depends on different criteria. Each user understands which decision will maximize his/her utility and satisfy him/her. The user ranks available resources based on his/her criteria. Improving user satisfaction, our approach can handle offloading processes in a decentralized manner with heterogeneous users. The rest of this paper is organized as follows: There is a brief review of related works in Section 2. Section 3 describes the task offloading problem. Section 4 models the problem as a pairing process and explains the mapping of the pairing process to a pairing game. The results of the conducted experiments are reported in Section 5. Section 6 concludes the paper.

2. RELATED WORKS

Task and computation offloading is a significant feature for task performance and resource use optimization in edge computing, which has become one of the preferred methods to increase the

1. H. Morshedlou is with Department of Computer Engineering and Information Technology, Shahrood University of Technology, Shahrood, Iran. Email: morshedlou@shahroodut.ac.ir
2. R.V. Shoar is is with Department of Computer Engineering, AmirKabir University of Technology, Tehran, Iran. Email: vafashoar@aut.ac.ir

performance of user tasks for smart devices. Computation offloading has been extensively studied in order to shorten the execution time of mobile devices and energy saving [1]-[2]. [3] investigated the problem of multi-user computation offloading in a single-channel wireless environment, where each user must decide whether or not to offload. [4] looked into an energy-harvesting mobile edge computing system. It suggests a Lyapunov-based dynamic computation offloading technique that jointly determines the offloading decision and the transmit power for computation offloading. [5] suggested offloading computational tasks to the mobile edge. The collective pool of resources for mobile devices is referred to by the authors as mist computing or cloudlets. They don't go into detail about how resource management might be carried out. Nevertheless, they place a strong emphasis on a hierarchical offloading of tasks from mobile nodes to mist, from mist to fog and finally from fog to cloud, all depending on application requirements. A Stackelberg game is used to model the interaction between cloud nodes and edge servers with the goal of increasing the benefits of cloud services [6]. Their analysis yields a unique Nash point. Two primary issues are addressed by their suggested strategy. First, a cloud server must decide whether to accept or deny a request to offload data to an edge server when it is made. The second issue is what to offer the edge node in the way of an incentive. [7] studied the issue of workload assignment and VM placement for mobile edge-computing applications. The paper develops a mathematical model to reduce the amount of hardware required by virtual machines (VMs) to support specified workloads in a multi-application scenario while satisfying the latency needs of various applications. According to the results, the users' request load, the hardware capacity of MEC servers and the latency requirements of applications all have a substantial impact on the amount of hardware used overall. MEC server utilization can be improved by utilizing remote VM deployment and workload aggregation. A novel approach is put out in [8] to address the issues of energy consumption and VM placement. The suggested approach can cut down on both energy usage and placement time. It proposes an OEMBA algorithm which integrates idle servers and reduces energy consumption. Using an improved Long Short-Term Memory model, virtual machine placement is then accelerated and latency is reduced based on historical data. According to the reported findings, the improved learning model can reduce placement latency and save energy. [9] suggested a method that allows for communication channel selection and virtual machine placement. It takes advantage of user movement prediction to manage the system's dynamic nature. According to anticipated user-movement, the prediction is used for dynamic VM placement and to identify the best communication path. The authors assert that their method reduces work offloading delays by 10% to 66% while maintaining a constant level of energy consumption by user equipment. In [10], the distributed task-offloading optimization issue is examined. First, a fresh optimization scheme is suggested that seeks to maximize the anticipated offloading rate of multiple agents by enhancing their offloading limits. Then, the issue is formulated in terms of game theory, which ultimately results in the development of a distributed best-response (DBR) iterative optimization system. Because of the frequently limited resources supplied by edge servers, not all IoV users' requirements can be met at once. A task offloading strategy built on fuzzy neural network (FNN) and game theory was developed in [11] to address the aforementioned issue. Following the load balancing of each RSU, game theory is used to determine the best task-offloading strategy for the users. Following this, the edge server acts as an agent to allocate computing resources for the offloaded tasks by the Q-learning algorithm. A task-offloading scheduling approach that combines multi-agent reinforcement learning and meta-learning was suggested by [12]. A first-order approximation technique is suggested to effectively train the policy network. Experiment results show that the MRL-based strategy has an outstanding overall performance and can be quickly applied in a variety of environments with good stability and generalization. [13] suggested a multi-agent deep reinforcement learning-based priority-driven joint task-offloading method where each edge server's decision-making is influenced by both its own state and shared global information. The work continues to create a worldwide state-sharing model that greatly lowers transmission overhead between peripheral servers. The performance analysis done on a dataset of mobile device movement trajectories shows that the suggested method can shorten job-completion times and increase the efficiency of edge computing resources. In [14], an approach, called NFSP, was proposed for adopting the architecture to offload decision-making. NFSP explicitly tackles the non-stationarity issue with the built-in self-play mechanism and uses a mixed strategy consisting of deep RL and the past average strategy, which is approximated by supervised deep learning. The conducted simulation experiments show that the proposed method outperforms the raw Independent RL (IRL) approaches. IRL approaches emerge as promising solutions for scenarios in which, due to

privacy and security concerns, mobile devices may be unwilling to expose their local information.

3. TASK OFFLOADING PROBLEM

Assume that there are limited resources at the edge network that offer computing or other services to users. For reasons of resource limitation, all tasks cannot be provided locally at the edge and some of them should be transferred to the cloud. One simple approach is to consider the deadlines of the tasks when making a decision about transferring some of them to the cloud. However, involving users or their agents in decision-making is a better approach for reaching an acceptable solution for all users and decreasing their dissatisfaction.

3.1 The Proposed Approach

There are two different types of agents in the approach presented in this paper. After initiation, agents interact with each other in the EOM module. First, we introduce these modules and entities in detail and then, we present the proposed approach and related analyses. Figure 1 depicts the components of the edge environment in the proposed approach. "DEVICES" are user devices that offload tasks to the edge. The edge layer contains some nodes or resources that provide the computing needs for user tasks. As illustrated in this figure, there are two sets of agents. eAgents interact as the brokers of edge nodes, with tAgents as the brokers of user devices. In our approach, EOM is only used to facilitate interactions among tAgents and eAgents; it plays no key role in decision-making and can be replaced by a decentralized platform for communications and interactions. Details of interactions are described in the following sections in details. In the cloud layer, there are data centers that, when the resources of the edge layer are not enough to provide all offloaded tasks, some tasks will be sent to these centers in the cloud.

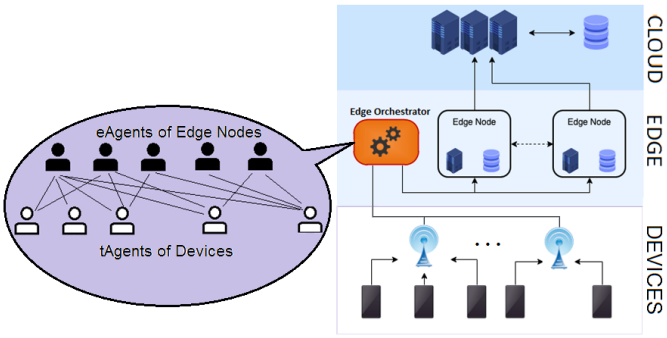

Figure 1. Components of edge environment in the proposed approach.

Edge Orchestrator Module (EOM): The edge orchestrator module makes decisions in the system. To determine how and where to handle incoming client requests, it consults the data gathered from the other modules.

tAgent: A tAgent is a broker or representative of a user task that tries to find a proper resource in edge or cloud for running its task. EOM initiates a tAgent when a new task is received in iteration t.

eAgent: An eAgent is a broker of EOM. According to the current number of tAgents and available resources in the system during iteration t, EOM generates eAgents. Each eAgent gets information about current tasks from all or some tAgents and gives them information about available capacity. Then, the pairing process of eAgents and tAgents in iteration t is initiated in the EOM. The pairing process is described in detail in the following section. We model the pairing process by using two games and show that the result of pairing is an NE point of them. Reaching an NE point means that the obtained result by our approach satisfies every rational agent or broker (resource broker (eAgent) or user broker (tAgent)).

4. PAIRING PROCESS

As said before, there are two sets of brokers, called task brokers (tAgents) and edge brokers (eAgents). By receiving an offloaded task, a new tAgent is initiated by the EOM and assigned to the task. tAgent

tries to maximize its user's utility. At the other side, there are eAgents which are policy-aware brokers or agents of edge resources. A tAgent offers its requirements (which include specific task needs) to all eAgents and waits to receive their proposals. When an eAgent receives offloading requests from tAgents, it queries the EOM for information, such as current workload and network congestion, among other things. Based on the information received from the EOM and the eAgent's resource management policy, it calculates the task-completion quality (TCQ) and announces the calculated value to the tAgent. Algorithm 1 shows how to calculate the TCQ for each tAgent. Notice that even when tAgent1 has a closer deadline than that of tAgent2, an eAgent may calculate a lower TCQ for tAgent1 than for tAgent2. However, when paired, it chooses tAgent1 due to its closer deadline. It announces a lower TCQ to tAgent1 to convince tAgent1 that it is better to choose another eAgent. A tAgent prefers to reach agreement with an eAgent the TCQ of which has the highest value among available eAgents. Between two tAgents, an eAgent chooses a tAgent with a closer task deadline. It's assumed that each eAgent contracts with up to one tAgent and *vice versa*. Since the capacity of each edge resource is different, the EOM may initiate multiple eAgents for each edge resource. Therefore, an edge resource may accept multiple tasks in iteration t. Each tAgent has an ordered list of eAgents based on their proposed TCQ. Similarly, each eAgent has an ordered list of tAgents based on their deadline. While the ordered lists of tAgents may be different, eAgents have similar ordered lists. These ordered lists are referred to as preferred lists.

Calculating TCQ: We use a meta-heuristic optimization algorithm for the 0-1 Knapsack problem to obtain an acceptable solution and then, we use the solution to calculate TCQ values. The meta-heuristic approach, that is used here, is simulated annealing. Note that finding an optimal solution for 0/1 Knapsack is possible, but for a large number of objects, time complexity makes it hard to use non-heuristic algorithms for finding optimal solutions. A task is considered as object and resource capacity is considered as Knapsack capacity. The details of the Simulated Annealing implementation are described as shown below:

Representation: A 0/1 representation is used, where 1 indicates that the task is chosen for running on the resource (object is placed in the Knapsack), while a value of 0 means that the task is not chosen (the object is left behind).

State Generation: State generation in our SA approach is done by randomly choosing a bit from the current state and flipping it to create the next state. This procedure is shown in Figure 2 for four objects.

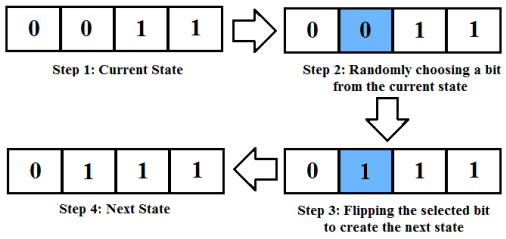

Figure 2. Simulated annealing state generation.

Pseudocode: The pseudocode of Figure 3 illustrates the heuristic that Simulated Annealing uses to search the solution space for an optimal solution for 0/1 Knapsack. The best_state contains the best state found up to the t-th step. In line 6, the function "value" calculates the value of the input state based on the objects in the Knapsack. Making the transition from the current_state to new_state is specified by the value(next_state) and value(current_state) (see line 7). States with a greater value are better than those with a smaller value. Even if value(next_state) is less than value(current_state), the probability of the transition must be positive. This feature prevents the method from becoming stuck at a local minimum that is worse than the global one (see lines 13 and 14). Transition is also affected by a global time-varying parameter Tt known as temperature. When Tt tends to zero (see line 18), the probability acceptance_function must also tend to zero (see line 13). For sufficiently small values of Tt, the system will then increasingly favor moves that go "downhill" and avoid those that go "uphill." With Tt = 0, the procedure reduces to the greedy algorithm, which makes only the downhill transitions. The system reaches equilibrium at Tt when we have no more state changes (see line 17).

Termination Criteria: When the temperature hits the system's lowest allowable temperature, the algorithm stops, signalling that the system is frozen. In our implementation, the temperature starts at 1000 and gradually drops to 0.0001 before freezing. To adjust the temperature, multiply the system temperature by = 0.9999.

After finding an acceptable solution, the EOM maps the solution to an array and first sorts them based on 0/1 and then sorts the selected and unselected objects separately based on task deadlines. After the sorting phase, objects are assigned TCQ values. The first selected object with the closet deadline has the maximum TCQ. Figure 4 illustrates the TCQ assignment process. Notice that the selection of tasks is done independently of their deadlines.

```
1   Set current_state = initial_state  and  best_state = initial_state;
2   t = 1;
3   do {
4           do {
5                   Produce next_state;
6                   delta = value (next_state)  -  value(current_state);
7                   if(delta > 0 && solution is acceptable) {
8                           current_state = next_state;
9                           if ( value(current_state) > value(best_state) )
10                                  best_state = current_state;
11                  }
12                  else {
13                          acceptance_function = exp(-delta/Tt);
14                          if(acceptance_function > random[0,1))
15                                  current_state = next_state;
16                  }
17          } While (system equilbrium a Tt);
18          Tt+1 = Tt*alpha;
19  } While (system has been frozen);
20  return best_state;
```

Figure 3. Pseudocode of algorithm 1 for SA implementation.

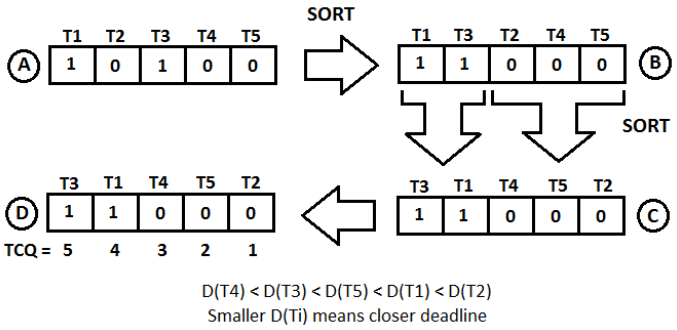

Figure 4. TCQ assignment process.

Definition 1: Preferred series of agents are loop-less if and only if there is no wrap-around order of agents $b_1, b_2, ..., b_k$ (k is even and $k > 2$) where each agent b_i likes b_{i+1} more than b_{i-1} (if $i = k$ then replace i+1 with 1). Keep in mind that in $b_1, b_2, ..., b_k$ for each i, b_{i-1} and b_{i+1} are both either tAgents or eAgents (have similar type) and are different from b_i.

Lemma1. The generated preferred series are loop-less if eAgents sort their preferred series based on the deadline of the tAgents, independent of the tAgents' preferred series.

Proof. There is a wrap-around order of agents, $b_1, b_2, ..., b_k$, such that each agent b_i prefers b_{i+1} to b_{i-1}, if the preferred series are not loop-less. Assume b_i to be an eAgent. Because eAgents organize their

preferred series according to deadlines of tAgents, they will obtain similar preferred series. This means that if eAgent b_i prefers tAgent b_{i-1} to tAgent b_{i+1}, all other eAgents do so. Because the order is wrap-around and preferring is transitive, after first loop over the order we conclude that eAgent b_i prefers tAgent b_{i+1} to tAgent b_{i-1}. This outcome conflicts with the definition of the order. Consequently, it is impossible to have such a chain of brokers to exist and the preferred series are loop-less.

Definition 2: A set of tAgents and eAgents is Pair-able if we can identify two agents (one tAgent and one eAgent) that favor each other above all other agents of the other type already present in the set at each cycle of an iterative process. The set for the following cycles or iterations no longer includes these two agents. After the last iteration, the set is either empty or contains agents of similar type.

Theorem 1. Set of tAgents and eAgents is Pair-able.

Proof. Begin with arbitrary tAgent such as b_1 from agent set and form a sequence $b_1, b_2, ...$ in which b_{i+1} is the most preferred agent by agent b_i among existing agents of the set which have the opposite type. Because there are only a certain number of agents, the sequence has a loop or cycle. Using lemma 1, the acquired preferred series are loop-less and hence the loop or cycle length is 2. Two agents that favor each other the most are present in this loop. A similar process will be carried out with fewer agents at the following iteration by removing these two agents from the group of brokers. This process is repeated until the final iteration.

Theorem 1 proves that sets of agents are pairable. The pairing approach in Figure 5 illustrates an algorithm for this pairing. Because each agent acts as a user's or edge resource agent, it should try to maximize its utility. We now want to be certain that the pairing specified in Theorem 1 satisfies both rational users and edge resources. For this purpose, in the following, we provide a description of this pairing from a game-theoretic perspective. Two games, an eGame and a tGame, are used to describe the situation. Let $r_i(j)$ represent the position of eAgent/tAgent j's (rank) in preferred serie of tAgent/eAgent i. The player i's strategy is indicated by s_i and S is the collection of all available strategy profiles for players.

4.1 tGame

tGame players are tAgents and eAgents which are part of the environment. Because agents of both types are rational, they select the strategy with maximum utility. The set of actions in tGame represent the tAgents' strategy space. Every action is the same as selecting a particular eAgent. For every strategy profile $s \in S$, utility of tAgent i is $u_i(s) = n - r_i(j) + 1$ if and only if $s_i = j$ (selection of eAgent j) and there is not another tAgent k $(k \neq i)$ such that $s_k = j$ and $r_j(k) < r_j(i)$, otherwise $u_i(s) = 0$.

4.2 eGame

Similar to tGame's definition, but this time, eAgents are the game players.

4.3 Game Theoretic Analysis

Theorem 2. A Pure Nash Equilibrium point of tGame and eGame is the result of Algorithm 1.

Proof. Assume that the current strategy profile of players in tGame and eGame is an equivalent point to the outcome of algorithm 1. All the eAgents have same preferred list and hence they have similar ranking of tAgents. Let uB_i denote a tAgent which is in i-th place of this ranking and iB_i is the paired eAgent to uB_i in the pairing procedure of algorithm 1.

In tGame, if tB_i deviates unilaterally and selects eB_k $(k \neq i)$, then for the case $r_{tB_i}(eB_k) > r_{tB_i}(eB_i)$, this deviation is not profitable, but if $r_{tB_i}(eB_k) < r_{tB_i}(eB_i)$, then eB_k has paired with a tB_k such that tB_k has a closer deadline than tB_i and every eAgent prefers tB_k to tB_i. So, eB_k refuses tB_i and

according to the definition of tGame $u_{tB_i}(s_{tB_i} = eB_k) = 0$. Therefore, for the case of $r_{tB_i}(eB_k) < r_{tB_i}(eB_i)$ deviation is not profitable as well. This means that the outcome of Algorithm 1 is a pure Nash Equilibrium of tGame.

For eGame, All eAgents prefer tAgent tB_i to tB_{i+1}. This means deviation of eB_1 is not profitable. Deviation of eB_2 can be profitable, if tB_1 prefers eB_2 to eB_1. But this is not true, because based on algorithm 1, tB_1 and eB_1 are two brokers that prefer each other the most. So, deviation of eB_2 is not profitable as well. It can be concluded in a similar way that for the rest of eAgents deviation is not profitable and the outcome of algorithm 1 will be a pure Nash Equilibrium of eGame as well.

Theorem 3. Both eGame and tGame have unique a Pure Nash Equilibrium (PNE) point.

Proof. Assume the PNE of eGame and tGame is not unique and there is another Nash Equilibrium $\underline{NE}$. Let $\overline{NE}$ denote the PNE which is equivalent to the outcome of algorithm 1 and definitions of tB_i and eB_i are like to Theorem 2. $s_B(N)$ denotes the strategy of broker B in a strategy profile N of game.

For tGame, if $s_{tB_1}(\underline{NE}) \neq eB_1$, then deviation of tB_1 to eB_1 will be profitable and $\underline{NE}$ is not a Nash. For the case of $s_{tB_1}(\underline{NE}) = eB_1$, we have $s_{tB_1}(\underline{NE}) = s_{tB_1}(\overline{NE})$. It can be shown that for all tAgents, the condition of $s_{tB_i}(\underline{NE}) = s_{tB_i}(\overline{NE})$ is required for $\underline{NE}$ to be a Nash point. So, $\overline{NE}$ is unique. Proof for eGame can be done in a similar way.

1- Put eAgents and tAgents in two different lists, called eList and tList.

2- Sort tList based on deadline of tAgents such that the first tAgent in the list is the tAgent with closest deadline.

3- While (min(eList.size, tList.size) > 0) {
3-1- Pick an eAgent in eList such that it is most preferred by the first tAgent in tList above all other agents currently in eList.
3-2- Pair the two eAgent and tAgent of step (3-1) and eliminate them from eList and tList.
}

Figure 5. Pairing algorithm of eAgents and tAgents.

Based on Theorems 2 and 3, the result of algorithm 1 corresponds to the unique pure NE point of eGame and tGame. Following the best response strategy, when the game has a unique pure NE point, this leads to convergence at that unique pure NE point [15]. This means that algorithm 1 produces a unique pure NE that satisfies rational tAgents and eAgents.

5. EXPERIMENTS AND RESULTS

In this section, we present the results of the conducted experiments. First, we introduce two criteria that are used for evaluation of the proposed approach. These criteria are User Satisfaction Level (USL) and Edge Utility Level (EUL).

5.1 User Satisfaction Level

We define User Satisfaction Level (USL) based on the average utility of users, as shown in Equation (1).

$$USL(t) = \sum_{i \in Users} u_i(r_i(t)) / number\_of\_users \quad (1)$$

At iteration t, the utility function u(r(t)) measures the utility that a user attaches to the allocated resource $r(t)$. Note that the service provider is not aware of the utility functions of a user.

If user i or its tAgent prefers resource A to resource B, then $u_i(A(t)) > u_i(B(t))$. For example, we can define the utility function as Equation 2, where $VMC_j(CPU)$ is CPU capacity of VM_j. $MinNeed_i(CPU)$ and $MaxNeed_i(CPU)$ are the minimum and maximum requirements of task i for the CPU, respectively. The symbols $VMC_j(Memory)$, $MinNeed_i(Memory)$ and $MaxNeed_i(Memory)$ have the same meaning but for memory. w_{cpu} and w_{memory} are weights that illustrate importance of CPU and Memory capacity for user task. β_{CPU} and β_{Memory} show the importance of wasting resources for users.

$$u_i(VM_j(t)) = w_{cpu} \times \begin{bmatrix} VMC_j(CPU) - MinNeed_i(CPU) \\ -\beta_{CPU} \times \max(0, VMC_j(CPU) - MaxNeed_i(CPU)) \end{bmatrix} + w_{memory} \times \begin{bmatrix} VMC_j(Memory) - MinNeed_i(Memory) \\ -\beta_{Memory} \times \max(0, VMC_j(Memory) - MaxNeed_i(Memory)) \end{bmatrix} \quad (2)$$

5.2 Edge Utility Level

The utility of edge is defined based on the used capacity and rate of successfully completed tasks for all received tasks in iteration t.

$$EUL(t) = \frac{used\_resources\_capacity(t)}{available\_resource\_capacity(t)} \times \frac{number\_of\_successfully\_completed\_tasks(t)}{number\_of\_all\_received\_tasks(t)} \quad (3)$$

5.3 Experiments

This sub-section illustrates the efficiency of the proposed approach using USL and EUL. In the conducted experiments, the resource needs of each category of tasks are expressed based on their minimum and maximum requirements. Here, the CPU needs of a task are illustrated using the [C, D] range. C is the minimum CPU need and D is the maximum CPU capacity consumed by the task. If there are two resources with different CPU capacities greater than C, both are capable of running and completing the task before its deadline. However, the task broker prefers to be paired with a resource that has a higher CPU capacity. Memory needs are also illustrated using ranges. For example, a task with a low memory requirement and the availability of extra memory results in faster running. Task brokers score the resources based on their available CPU and memory capacities.

There are three types of virtual machines (VMs) and their capacity details are listed in Table 1. The <cpu, memory> vector represents the available capacity of a VM and the user task requirement. The vectors are unitized for simplicity. Table 2 illustrates CPU and memory requirements for different task types. According to the table information, any task of any type can be executed on any type of VM, though the results may differ in terms of execution speed. For example, a task of type 1 prefers VM of type 3 to the other types, while a task of type 2 prefers VM of type 2 to VM of type 3. This is due to the fact that a task of type 2 cannot consume more CPU and memory capacities of VM type 3. Its maximum requirements are <3, 5>, which can be met by a weaker VM such as a VM of type 2.

Table 1. CPU and memory capacity of VMs.

Resource Types	VM Type 1	VM Type 2	VM Type 3
CPU Capacity	4	5	6
Memory Capacity	4	5	6

Table 2. CPU and memory needs of different task types.

Task Requirements	Task Type 1	Task Type 2	Task Type 3	Task Type 4	Task Type 5
CPU Need Range	[1~2]	[1~3]	[2~4]	[3~5]	[4~6]
Memory Need Range	[4~6]	[3~5]	[2~4]	[1~3]	[1~2]

5.3.1 Workloads of the Experiments

Following a task-type-based distribution, each user device generates a workload with a specific task type. The times at which tasks are generated are defined by a Poisson process. Each task type has a certain expected value. The time interval between two tasks of a device is generated at random by an exponential distribution with this expected value. The number of edge servers and user devices is 10 and 200, respectively. The Poisson parameter (λ) for all task types is 1. Each edge server's CPU and memory capacity are assumed to be 100. Therefore, an edge server can host 25 VMs of type 1, 20 VMs of type 2 or 16 VMs of type 3. An edge server usually hosts multiple VMs of different types. To evaluate the proposed approach against the existing works in the literature, we selected two rival approaches to compare, called RW1 and RW2 in this paper. RW1 and RW2 are described in [14] and [8], respectively. For details on these approaches, please see the related work section. There are 200 user devices in the experiments that generate workloads. Workload1 is generated by 40 user devices for each task type. Workloads 2 and 3 are generated by (80, 60, 40, 20, 0) and (0, 20, 40, 60, 80) users, respectively. The tuples show the number of users per each task type according to (TaskType1, TaskType2, TaskType3, TaskType4, TaskType5) tuple.

Figure 6 shows the utility of a user with a specific task type on each type of VM. The utility function of a user is defined using (2), where $\beta_{CPU} = 1.5$, $\beta_{Memory} = 0.8$ and $w_{CPU} = w_{Memory} = 0.5$. Here, $\beta_{CPU} > \beta_{Memory}$ means that preventing CPU wastage is more important than preventing memory wastage. As illustrated by this figure, tasks of type 1 or 5 prefer VMs of type 3 to the other types. In addition, VM of type 2 is preferred by tasks of type 4 or 2 and tasks of type 3 prefer VM of type 1. These utility values will be used to calculate the USL of users in the following. Figure 7 illustrates the average of USL over 1000 consecutive iterations by the proposed approach, RW1 and RW2, during workload 1. The workload is generated according to the approach described at the beginning of this section. USL in iteration t is calculated using (1). The higher values of USL mean a higher average utility of the users and, as a result, a higher satisfaction level. Figures 8 and 9 show similar diagrams for workload2 and workload3, respectively. The proposed approach leads to higher USL in comparison to rival approaches. The proposed approach achieves higher USL, because it takes user preferences into account during the pairing process, whereas competing approaches do not. Since a user ranks the available resources according to the amount of utility it gets from each of them, incorporating the user's preference during the pairing process will increase the level of USL. During workload 2 (Figure 8), tasks have less CPU requirement than Workloads 1 and 3. Because $\beta_{CPU} > \beta_{Memory}$, the difference in USL levels in different approaches during workload 2 is smaller than in workloads 1 and 3. This means that the proposed approach has a better performance when there are high CPU requirements in workloads.

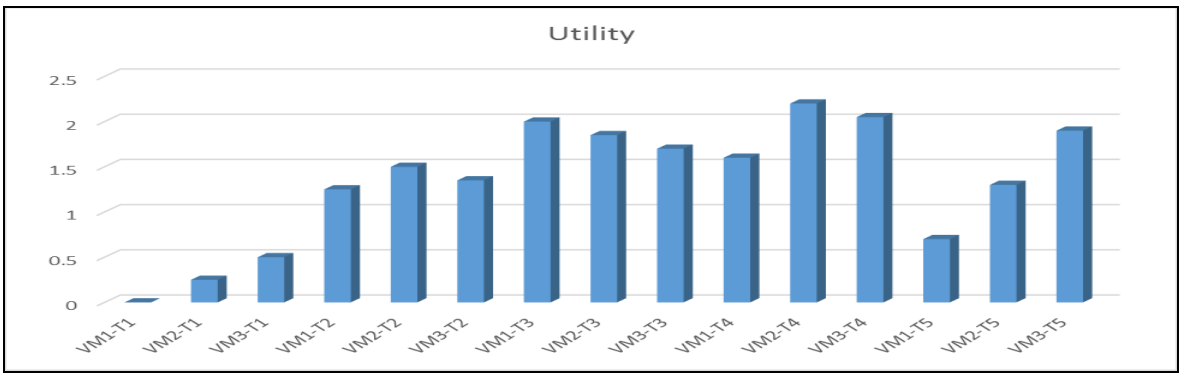

Figure 6. Utility of different VMs for each task type.

According to the diagrams in Figure 10, it is clear that when the majority of users have high CPU needs (e.g. Workload3), CPU wastage is very low. Furthermore, when the majority of users have high memory requirements (e.g. Workload2), CPU wastage is high. This is due to the fact that there is a positive correlation between the processing power and the amount of available memory at each virtual

machine (see Table 1). In other words, VMs with more processing power have more available memory. Since $\beta_{CPU} > \beta_{Memory}$ in (2), we have a smaller USL during workload 2 compared to both workload 1 and workload 3 (see Figures 7, 8 and 9). During workload 2, tasks of type 1, which waste more CPU compared to the other types, constitute 40% of all the tasks. To be more specific, the majority of the tasks in workload 2 are of types 1 and 2, which resulted in more CPU wastage than in workloads 1 and 3. According to the results, the proposed approach always performs better than similar works in terms of USL. In addition to this advantage, the proposed approach can perform better under different workloads in terms of CPU usage rate.

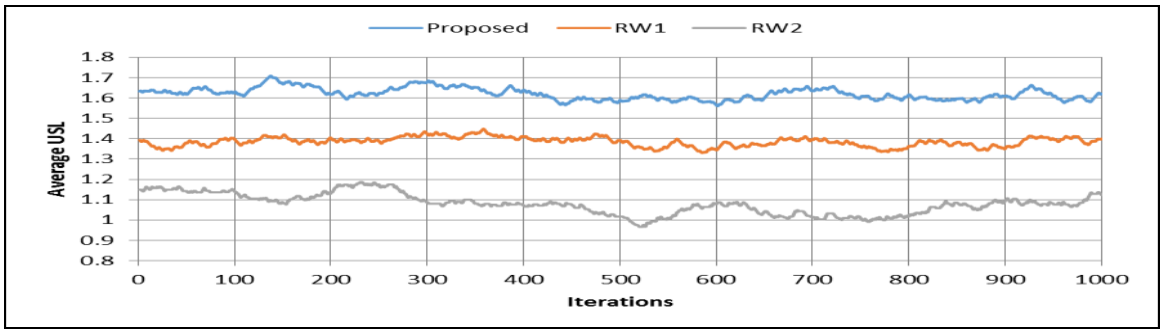

Figure 7. Average USL of users during workload 1.

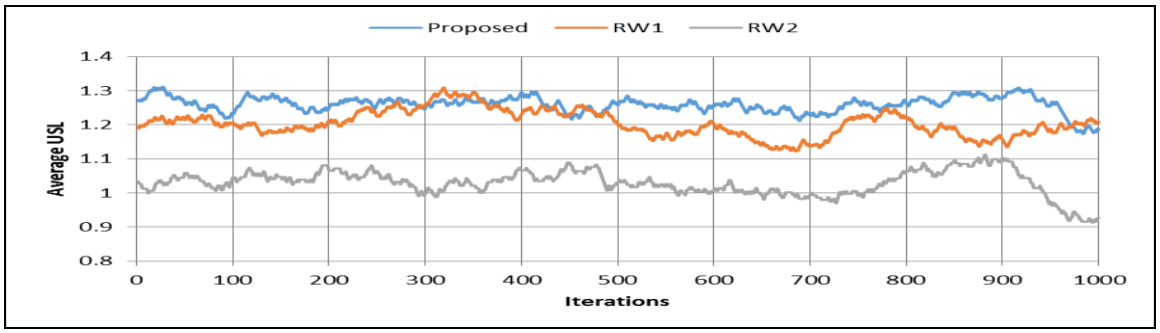

Figure 8. Average USL of users during workload 2.

Now, we compare the results of the proposed approach with rival methods using EUL measure. We use EUL as a criterion to measure resource capacity usage and task-completion rates. Here, each iteration is a time interval in which task-completion rate and average resource usage are used to compute EUL of that iteration using (3). Task-completion rate is reported by dividing the number of successfully completed tasks by the total number of all received tasks in the mentioned time interval. As it can be seen in Figure 11, the proposed approach has achieved better results in most iterations. In the proposed method, the allocation of extra, unutilized capacity to tasks is avoided. For example, when a task of type 4 prefers a VM of type 2 to a VM of type 3, the proposed method instantiates a VM of type 2. Therefore, the capacity of a physical server machine can be allocated to more virtual machines and more tasks have a chance to be completed successfully. This is why the proposed approach performs better compared to the other approaches.

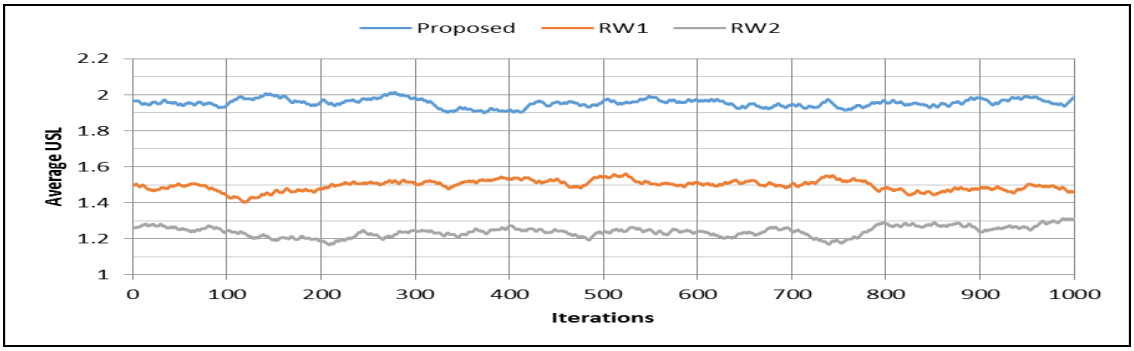

Figure 9. Average USL of users during workload 3.

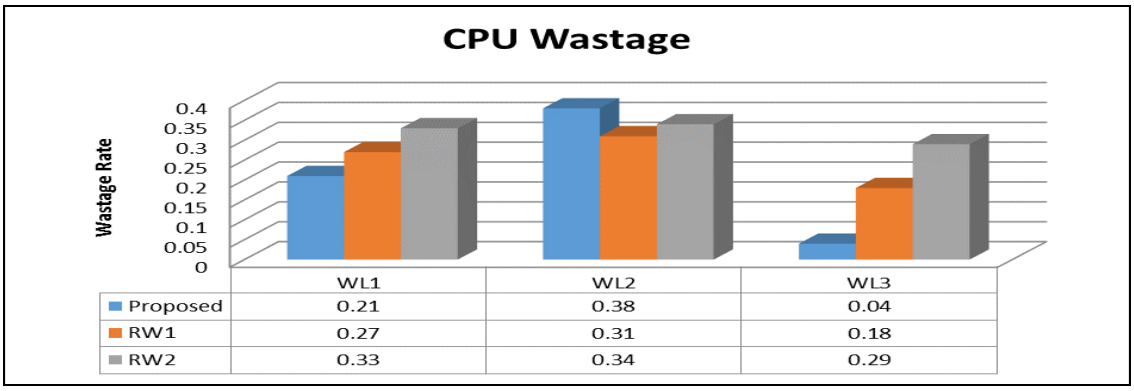

Figure 10. CPU wastage during workloads 1, 2 and 3.

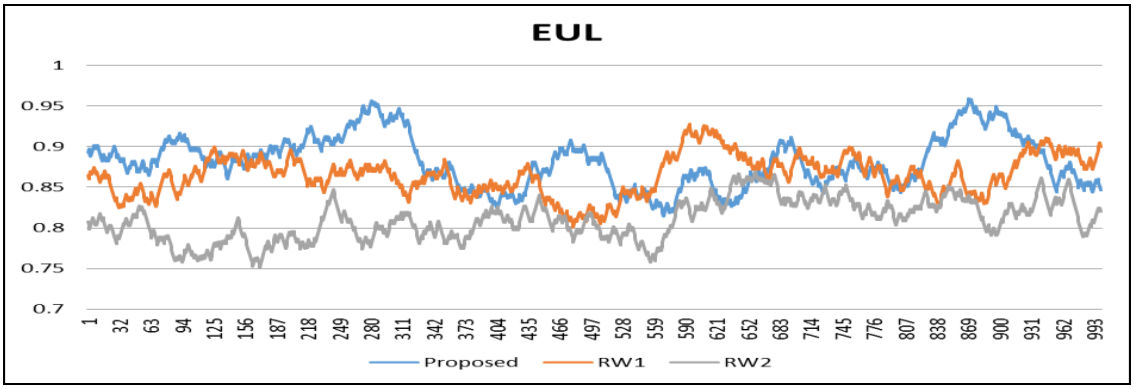

Figure 11. EUL measure using different approaches.

6. CONCLUSIONS

Due to the limited capacity, edge nodes may not be able to provide the total capacity necessary for delivering all user tasks offloaded to the edge. Therefore, it remains a challenge to reduce user-task handling delays when there are insufficient resources. One solution to avoid further delays is offloading some tasks to cloud data centers. But, finding an efficient approach for decision-making about which tasks should be handled by edge servers and which ones by the cloud is not easy. In addition to the complexity of interactions and relations in edge environments, some nodes or devices may be unwilling to expose their local information due to privacy and security concerns. This makes centralized control and decision-making very difficult and inefficient. Distributed and decentralized approaches, like agent-based approaches, emerge as promising solutions for such scenarios. In this paper, an agent-based approach for task offloading in edge computing environments is proposed. The proposed approach incorporates user satisfaction into decentralized decision-making. To the best of the authors' knowledge, this is the first work in the literature that addresses task-offloading strategy optimization based on user satisfaction level. In our work, user satisfaction level is defined based on the average utility that users attach to the CPU and the memory capacity of the allocated edge resources. But, the utility functions of different users may depend on different criteria. Each user knows better what will make him/her satisfied. As a future work, the definition of utility function can be extended to include complex criteria, such as energy efficiency. For example, dynamic frequency scaling (also known as CPU throttling) is a power management technique in today's computer architectures, whereby the frequency of a microprocessor can be automatically adjusted "on the fly" depending on the actual needs, to conserve power and reduce the amount of heat generated by the chip. In our approach, each user device, according to its condition, e.g. battery level, can attach a utility to the local running of a task (high frequency and energy consumption) or offloading to edge (low frequency and energy consumption). Regardless of what improves a user's satisfaction level, our approach can be extended to handle offloading processes in a fully decentralized way with users with heterogeneous satisfaction criteria. Another contribution of our work is representing the task-offloading problem in the form of two decentralized games. We proved that the proposed pairing process reaches the Nash equilibrium point of the games. It is also demonstrated that achieving Nash equilibrium is the best possible pairing for both tasks and VMs. A game-theoretic analysis is also

provided to prove that the presented approach increases the average utility of users and their satisfaction level. We also conducted some experiments to evaluate the proposed approach. The results illustrate that the proposed approach significantly improves the average utility of users and the EUL measure.

REFERENCES

[1] C. Feng et al., "Computation Offloading in Mobile Edge Computing Networks: A Survey," J. of Network and Computer Applications, vol. 202, no. 103366, p. 103366, Jun. 2022.
[2] Y.-Y. Huang and P.-C. Wang, "Computation Offloading and User-clustering Game in Multi-channel Cellular Networks for Mobile Edge Computing," Sensors (Basel), vol. 23, no. 3, p. 1155, Jan. 2023.
[3] X. Chen, "Decentralized Computation Offloading Game for Mobile Cloud Computing," IEEE Transactions on Parallel and Distributed Systems, vol. 26, no. 4, pp. 974–983, Apr. 2015.
[4] Y. Mao et al., "Dynamic Computation Offloading for Mobile-edge Computing with Energy Harvesting Devices," IEEE J. on Selected Areas in Communications, vol. 34, no. 12, pp. 3590–3605, Dec. 2016.
[5] X. Chen, L. Jiao, W. Li and X. Fu, "Efficient Multi-user Computation Offloading for Mobile-edge Cloud Computing," IEEE/ACM Transactions on Networking, vol. 24, no. 5, pp. 2795–2808, Oct. 2016.
[6] Y. Liu et al., "Incentive Mechanism for Computation Offloading Using Edge Computing: A Stackelberg Game Approach," Computer Networks, vol. 129, pp. 399–409, Dec. 2017.
[7] W. Wang, Y. Zhao, M. Tornatore, A. Gupta, J. Zhang and B. Mukherjee, "Virtual Machine Placement and Workload Assignment for Mobile Edge Computing," Proc. of the 2017 IEEE 6th Int. Conf. on Cloud Networking (CloudNet), DOI; 10.1109/CloudNet.2017.8071527, Prague, Czech Republic, 2017.
[8] C. Jian, L. Bao and M. Zhang, "A High-efficiency Learning Model for Virtual Machine Placement in Mobile Edge Computing," Cluster Computing, vol. 25, no. 5, pp. 3051–3066, Oct. 2022.
[9] J. Plachy, Z. Becvar and E. C. Strinati, "Dynamic Resource Allocation Exploiting Mobility Prediction in Mobile Edge Computing," Proc. of the 2016 IEEE 27th Annual Int. Symp. on Personal, Indoor and Mobile Radio Communications (PIMRC), DOI: 10.1109/PIMRC.2016.7794955, Valencia, Spain, 2016.
[10] J. Zhou et al., "Distributed Task Offloading Optimization with Queueing Dynamics in Multiagent Mobile-edge Computing Networks," IEEE Internet of Things J., vol. 8, no. 15, pp. 12311–12328, 2021.
[11] X. Xu et al., "Game Theory for Distributed IoV Task Offloading with Fuzzy Neural Network in Edge Computing," IEEE Transactions on Fuzzy Systems, vol. 30, no. 11, pp. 4593–4604, Nov. 2022.
[12] W. Ding et al., "A Multi-agent Meta-based Task Offloading Strategy for Mobile Edge Computing," IEEE Trans. on Cognitive and Developmental Systems, DOI: 10.1109/TCDS.2023.3246107, 2023.
[13] J. Yang, Q. Yuan, S. Chen, H. He, X. Jiang and X. Tan, "Cooperative Task Offloading for Mobile Edge Computing Based on Multi-agent Deep Reinforcement Learning," IEEE Transactions on Network and Service Management, DOI: 10.1109/TNSM.2023.3240415, 2023.
[14] J. Hou et al., "GP-NFSP: Decentralized Task Offloading for Mobile Edge Computing with Independent Reinforcement Learning," Future Generation Computer Systems, vol. 141, pp. 205–217, Apr. 2023.
[15] J. Lei and U. V. Shanbhag, "Stochastic Nash Equilibrium Problems: Models, Analysis and Algorithms," IEEE Control Systems, vol. 42, no. 4, pp. 103–124, Aug. 2022.

ملخص البحث:

نظـراً لمحدوديـة المـوارد فـي شـبكة الحافّـة وإحالـة عـدد كبيـر مـن المهـامّ الـى خـوادم الحافّـة، فـإنّ مـوارد الحافّـة قـد لا يكـون بمقْـدورها أن تـوفرّ السّـعة المطلوبـة لتنفيـذ جميـع المهـامّ. ونتيجـةً لـذلك، تتعـيّن إحالـة بعـض المهـامّ الـى السّـحابة، الأمـر الـذي مـن شـأنه أن يتسـبّب فـي تـأخيراتٍ إضـافية. وهـذا بـدوره قـد يقـود الـى زيـادة عـدم الرّضـا لـدى المسـتخدمين الّـذين تـم نقْـل مهـامّ تتعلّـق بهـم الـى السّـحابة. فـي هـذه الورقـة، يجـري اقتـراح نظـام جديـد مـن أجـل اتخـاذ القـرار حـول أيّ المهـامّ يـتمّ نقلهـا الـى السّـحابة وأيهـا يـتمّ تنفيـذها محلّيـاً. ويحـاول النّظـام المقتـرح تكـوين أزواجٍ مـن المهـامّ والمـوارد، بحيـث تكـون هـذه الأزواج مـن المهـامّ والمـوارد هـي أكثـر المـوارد تفضـيلاً بالنّسـبة للمسـتخدِمين مـن بـين جميـع المـوارد المتاحـة. وقـد بيّنّـا أنّ الوصـول الـى نقطـة "نـاش" للتّـوازن يُمكـن أن يلبّـي هـذا الشّـرط. وبـيّن تحليـلٌ باسـتخدام نظريـة اللُّعبـة أنّ النّظـام المقتـرح يزيـد مـن معدّل الاستفادة لدى المستخدِمين ويرفع من مستوى الرّضا لديهم.