Time-optimized Sequential Decision Making for Service Management in Smart City Environments

Saleh ALFAHAD [a,1], Christos ANAGNOSTOPOULOS [a] and Kostas KOLOMVATSOS [b]

[a] *School of Computing Science, University of Glasgow, UK*

[b] *Department of Informatics & Telecommunications, University of Thessaly, Greece*

ORCiD ID: Saleh ALFahad https://orcid.org/0000-0003-0101-4458, Christos Anagnostopoulos https://orcid.org/0000-0003-1517-6757, Kostas Kolomvatsos https://orcid.org/0000-0002-9442-3340

Abstract. Edge Computing is a new computing paradigm that aims to enhance the Quality of Service (QoS) of applications running close to end users. However, edge nodes can only host a subset of all the available services and collected data due to their limited storage and processing capacity. As a result, the management of edge nodes faces multiple challenges. One significant challenge is the management of the services present at the edge nodes especially when the demand for them may change over time. The execution of services is requested by incoming tasks, however, services may be absent on an edge node, which is not so rare in real edge environments, e.g., in a Smart Cities setting. Therefore, edge nodes should deal with the timely and wisely decision on whether to perform a service replication (*pull-action*) or tasks offloading (*push-action*) to peer nodes when the requested services are not locally present. In this paper, we address this decision-making challenge by introducing an intelligent mechanism formulated upon the principles of Optimal Stopping Theory and applying our time-optimized scheme in different scenarios of services management. A performance evaluation that includes two different models and a comparative assessment that includes one model are provided found in the respective literature to expose the behavior and the advantages of our approach which is the optimal stopping theory (OST). Our methodology (OST) showcases the achieved optimized decisions given specific objective functions over services demand as demonstrated by our experimental results.

Keywords. Service Management, Sequential Decision Making, Task Offloading, Edge computing, Smart Cities, Optimal Stopping Theory.

1. Introduction

As a result of the fast growth of the Internet of Things (IoT), smart cities are replete with linked sensors and computing devices used for information collection and process-

[1]Corresponding Author: Saleh ALFahad, Knowledge and Data Engineering Systems, School of Computing Science, University of Glasgow, *2426473A@student.gla.ac.uk*

ing [1]. Various collaborative services (e.g., smart medical assessment, smart university, elderly support) emerge as a result of the right processing and utilization of the huge data created by the Internet of Things [2]. On the basis of integrated services, it is likely that the smart city will be able to increase the Quality of Experience (QoE) of people while reducing the use of resources. Furthermore, smart cities are endowed with a robust capacity to observe, analyze, and adapt to their residents [3]. Nevertheless, because of the rapid growth of data and the limited processing capacity of endpoints, it is challenging to develop joint services that are typically time and latency-sensitive when depending purely on IoT devices. To increase the QoE in smart cities, it is essential to use additional computational resources [4]. As a result, Edge Computing (EC), which completely leverages computational resources at the network edge and can meet real-time demands, has been proposed as an effective paradigm to resolving the issue of inadequate computing resources in smart cities [5].

In order to offer ubiquitous and pervasive services, the present expansion of IoT and EC makes them possible to have a large number of devices and processing nodes located close to end-users [6]. In recent years, we have also seen the introduction of several mobile devices and applications, such as drones and Vehicular Networks (VN), each with its own unique set of capabilities. The development of this technology has made it possible for computing and sensing devices to launch applications such as Augmented Reality (AR), predictive analytics activities at the edge, intelligent vehicle control, traffic management, and interactive applications [7]. These types of services are given in response to the need for processing data being gathered by IoT devices and, then, being sent to EC nodes [8]. A range of IoT applications may offer to end users with more precise and detailed network services [9]. In this situation, an increasing number of equipment and sensors are being networked through IoT technology, and this equipment and sensors will create vast amounts of data that need additional processing, therefore delivering service providers and end users with intelligence. In the traditional Cloud computing setup, all information must be transferred to centralized servers, and after processing, the outcomes must be sent directly to the requestors, e.g., sensors or other devices. This procedure places significant stress on the network, particularly in terms of bandwidth and resource requirements for data transmission. Furthermore, when data size increases, bandwidth utilization will decline.

EC has been suggested to address the aforementioned difficulties [10], [11]. The fundamental goal is to increase the processing capabilities at the network edge; specifically, computing [12], bandwidth [13], and storage resources [14] are relocated nearer to IoT systems in order to minimize core data traffic and response delay and to support resource-intensive IoT applications. In comparison to Cloud [15] [16], EC nodes have limited storage and computing capabilities. Nonetheless, they are able to host a number of services that make it easier to carry out a variety of processing tasks [17]. These kinds of activities are often presented in the form of assignments that 'request' the processing of data (e.g., predictive analytics, explanatory-driven models). The fact that IoT and EC may work together to facilitate the implementation of collaborative activities in which nodes can collaboratively complete the tasks that have been requested is an intriguing development [18]. For instance, nodes share services or data and they even offload processing duties onto peers in certain circumstances. Because of their reduced capability for storage and computing, EC nodes are only able to host a (sub-)set of the total accessible services and the data that has been obtained. The collected data can be reported by stream monitoring e.g., the evolution of a certain phenomenon [19], [20], [21] and streams generated by autonomous nodes (e.g., unmanned vehicles) [22]. Upon these streams, we can support the extraction of knowledge, the detection of events or any other processing [6].

Services are essential in order to perform the processing responsibilities that have been delegated by applications or users in the form of tasks. As a result, the demand for services is always adjusting to accommodate the dynamic requests of various activities/tasks. This indicates that a task can be asking for a service that may or may not be available locally. Then, nodes have two options:

- Option 1: They can perform a service replication (*pull-action*) from their edge neighbors or the Cloud, but only if the service is suitable for their computing capabilities;
- Option 2: They can delegate the responsibility (*push-action*) to the peers/Cloud that presently host the service(s).

The aforementioned options deal with deciding where to transfer service in order to keep the Quality of Service (QoS) at a high level. Because of the nature of the EC environment and the non-static behavior of end users, it is obviously difficult to provide an ideal migration/replication plan that can be implemented. Moreover, due to the situations that make the local authority of the user complex, offloading tasks involves sending them to peers or the Cloud for processing. For instance, the node that is assigned the duty of processing as the receptor of a task could have a high load, and as a result, it might not be able to guarantee an effective completion within the permitted time interval that may be specified by the requester. A strategy for the migration of services was presented in [23]; its purpose is to satisfy the service latency requirements of EC while keeping migration costs and trip times to a minimum. Furthermore, in [24], the authors recommended using a reinforcement learning-based model to solve the service migration issue.

In this paper, we focus on the pull-action leaving the initiative to EC nodes to enhance their autonomous nature and suggest that services' replication may be optimized and controlled in the EC environment by adopting the principles of Optimal Stopping Theory (OST) [25]. In this case, EC nodes must decide locally when it is the *best* time to replicate the service in order to maximize its ability to carry out the requested tasks. We have to mention that the decision for services replication is made by every EC node *independently* being fully aligned with the needs of the incoming tasks and the status of the node. The remaining paper is organized as follows. In Section 2, we provide a summary of the prior work as well as the rationale and our contribution, while Section 3 delves into the specifics of the proposed OST-based decision-making model. The outcomes of our performance and comparative assessment are presented in Section 4, and Section 5 draws the conclusion of this work and indicates potential avenues for further research.

2. Related Work & Contribution

2.1. Related Work

The majority of the approaches for task offloading at the edge depends on the selection of whether or not the task should be processed locally or should be offloaded to peers or Cloud. The time in execution latency [26] and the amount of energy used should both be reduced as much as possible. These are the two primary goals. The work presented in [27] provides a framework for offloading computation workloads from a user device to a server hosted in EC. Our approach is different from that due to the fact that our time-optimized decision primarily considers the actual benefit of payoff which indicates the current users' requests for specific service. In [27], the User Equipment (UE) choice on whether to offload computing tasks with the highest CPU availability for a certain application is driven by the radio network information service (RNIS), according to the current value of round trip time (RTT) between the UE and the edge node. In our method, the offloading choice ideally relies on the number of users' requests for specific service as well as the cost of delay. Therefore, the decision of offloading is made when the dedicated time of the required service by users is finished which indicates the required service is not worth being hosted. As a result, this will optimize the edge users' capacity.

The work in [28] presents a method for optimizing the decision-making process that an Unmanned Aerial Vehicle (UAV) uses to choose whether or not to do the compute task locally or to offload it to an edge server. The decision is made based on a series of interactions that take place between the UAV and IoT systems. During these interactions, the UAV receives feedback on the state of the network and EC server, which enables an estimation of the remaining amount of time needed to complete the task. As a result, the UAV uses this information to solve an optimization problem with the purpose of decreasing a weighted sum of delay and energy costs as much as possible. In our work, EC chooses the ideal moment that has a high payoff to perform a service replication (pull-action) from their edge neighbors or the cloud. The approach described in [29] analyses the present state of the node's resources, which are modeled as a Markov Decision Process utilizing Q-learning for training, in order to identify which portion of the application ought to be offloaded and then moves on with an offloading decision. The primary objective is to reduce the amount of time that the offloaded applications are delayed while taking into account the mobile fog that is located nearby, the mobile fog that is located adjacent, or the Cloud as suitable offloading locations. In our work, we have examined the scenario in which the edge node only performs a service replication (pull-action) from its edge neighbors or the cloud since the edge node is static and makes this decision in a short delay as close to the real's delay scenario.

The work in [30] focuses on reducing the processing delay of all activities and the energy usage of the edge device by optimizing the task allocation decision and the central processing unit (CPU) frequency of the edge device. However, this study focuses on making decisions based on the popularity of the required service to complete the task. The authors of [31], [32] studied the offloading problem decision for a multi-edge user & multi-edge node. However, in our approach, we focus on making the decision of (pull or push -action) for a single user to edge node or peer since our approach OST has strong deals with low capacity better than the approach in [31], [32]. Using a time-division multiple access protocol in [33], edge users can conduct their individual activities locally or offload all or a portion of them to the Access Point (AP). Their objective is to decrease the AP's overall energy consumption relative to the user at the network's edge. Nevertheless, our recommended solution is subject to a choice depending on the service's popularity and our stopping time with the maximum payoff. The study presented in [34] addresses the issue of task offloading in ultra-dense networks with the goal of reducing latency as much as possible while preserving the battery life of user equipment. On the other hand, our method provides the optimum option in terms of (pull or push-action) the service (from or to) the edge's neighbors or peers.

The work in [35], [36] primarily formulate the decision-making of offloading based on a resource scheduling factor. However, in our approach, we develop the decision of (push or pull - action) w.r.t accumulation of the newly received tasks and cost incurring. A state-action-payoff-state-action (RL-SARSA) method is introduced in [37] to tackle the resource management issue at the edge server and make the best offloading option for reducing system cost. Moreover, the work in [38] aims to make the offload decision based on minimizing the cost. However, our work emphasizes making the decision with the greatest payoff, considering the expense cost each time that we do not make the decision of (push-action). In the study presented in [39], [40], the issue of offloading computation for many users in a wireless environment with uncertainty is investigated. Nevertheless, the goal of our work is to provide the single-edge user with the best possible time in which to make a decision (push or pull - action). In [41] [42], the authors examine the challenge of partial offloading scheduling and resource allocation for edge computing systems with various independent activities. The objective is to reduce the weighted total of processing latency and energy usage while satisfying the tasks' transmission power limitations. Nevertheless, our approach enables the edge user to choose the optimal (pull-action) solution. Since the service has previously been hosted, the wait for future tasks will be reduced. The work in [43] [44] investigate energy-efficient task offloading in edge computing. The goal is to reduce the amount of energy required for task offloading. However, our method focuses on making the choice at the optimal moment to host the most popular tasks, hence optimizing the edge's capacity resource utilization.

Migration of data and services may be used to fill 'gaps' in the local knowledge base of a processing node. In [45], the interested reader may obtain a survey of related initiatives. When migrating services, it will be efficient to meet processing demands locally; nonetheless, a list of factors must be considered. To minimize overburdening the network, service migration may be accomplished in phases; service components should be "hooked up" to the hosting infrastructure swiftly, easily, and automatically. One of the technologies used for these reasons is Machine Learning (ML). ML may serve as the foundation for producing models that predict the movement of users and provide a proactive procedure for services migration [46]. Reinforcement learning may improve agent behavior while determining the optimal action to take [47], [48]. The difficulty in depending on supervised machine learning technology is identifying the appropriate representative training data set. As a result of the raised degree of uncertainty in the corresponding decision-making, it is challenging to gather data that includes all alternative node statuses. The service migration model may alternatively be expressed as an online queue stability issue [49]. The issue may be addressed using either a Lyapunov optimization or a multi-objective optimization scheme [50], [51], and the Pareto optimum solution can be determined. In the past, efforts have mostly focused on minimizing delay within the restrictions of energy usage in EC settings. However, the bulk of research efforts that aim at a user-centric service migration model is 'affected' by the extra time required to address the optimization issue, except if a list of assumptions is made or approximate solutions can be accepted. Using Markov chains is also a possible solution to settle the issue. Using a modified policy-iteration algorithm, the authors of [52] provide a solution to a finite-state Markov decision process. A technique based on Thompson sampling is suggested in [53] to facilitate dynamic service migration choices. The authors of [54] describe a service allocation module for networked automobiles powered by Cloud-based services. The objective is to increase the amount of autonomy of automobiles capable of picking nearby automobiles to share adopted services. The authors of [26] suggest a proactive system for determining the effective administration of services and tasks that are present/reported at EC nodes. The suggested model monitors the demand for existing services and rationalizes their management, i.e., their local presence/invocation when the demand is modified by the desired processing operations. To proactively achieve the strategic objectives of the envisioned model, a statistical inference process is initiated for each incoming task.

2.2. Rationale & Contribution

Figure 1 illustrates in a simplified way the problem of deciding whether to perform a service replication (pull-action) from their edge neighbors or the cloud or offload (push-action) it to the cloud service to make the necessary computation to complete the required task. Let us assume that the starting time of recording the number of requests for a task is $t = 0$ and the end time period, hereinafter referred to as deadline is $T > 0$, where a decision should be made. Figure 1 shows two decisions against time. Specifically, in Figure 1 (a), we observe the scenario of making the pull-action decision at an early time t^* (and before the deadline T) since we are expecting receiving more tasks requesting a specific service. This denotes that there is a relatively high demand for this service, thus, it is advisable to decide on pulling the service locally to the node. In Figure 1 (b), we observe the scenario in which we delayed the decision making in light of increasing our confidence that there is no high demand of the service. Once the deadline elapses and we have not decided on a pull-action, which indicates that the service request rate was relatively low, then, the node decides on offloading the service requests (push-action) either to its peers or the Cloud. In other words, this implies that there was not a large volume of tasks requesting the specific service, thus, offloading these tasks is necessary. Assume the case where we made the decision to download/pull the service earlier enough, and then after a while, we realized that the service was requested only by a small number of users. We would have consumed additional cost of resources without benefiting from the download/pull service. As a result, it would have been more efficient to delay our decision until the end of the dedicated time and then perform the action of offloading as it is shown in Figure 1 (b). On the contrary, if we had decided to send/offload the request to the Cloud at an early time, then, we would not have high confidence whether this request was highly requested on the node, thus, this decision would not be efficient. The reason is that this service would have been highly requested in the future, thus, the push-action would not be appropriate. However, if after a while we realize that the service demand has significantly increased, then an early push-action results in overloading the network with offloading tasks to other peers or the Cloud unnecessarily. Thus, a proper process is to sophisticatedly delay any decision and perform service replication (pull-action) from node's neighbors or the cloud to the node at a specific time t^* as shown in Figure 1 (a). Hence, the problem has been identified as finding that time t^* to ensure that our resources are utilized in the best way trading off between task offloading and service replication. A fundamental solution has been proposed, which ensures the appropriate decision is taken at the time t^* taking into account the delay costs.

Contribution: Our approach is adaptable to applications that use offloading decision-making algorithms in EC environments. Our major technical contribution is:

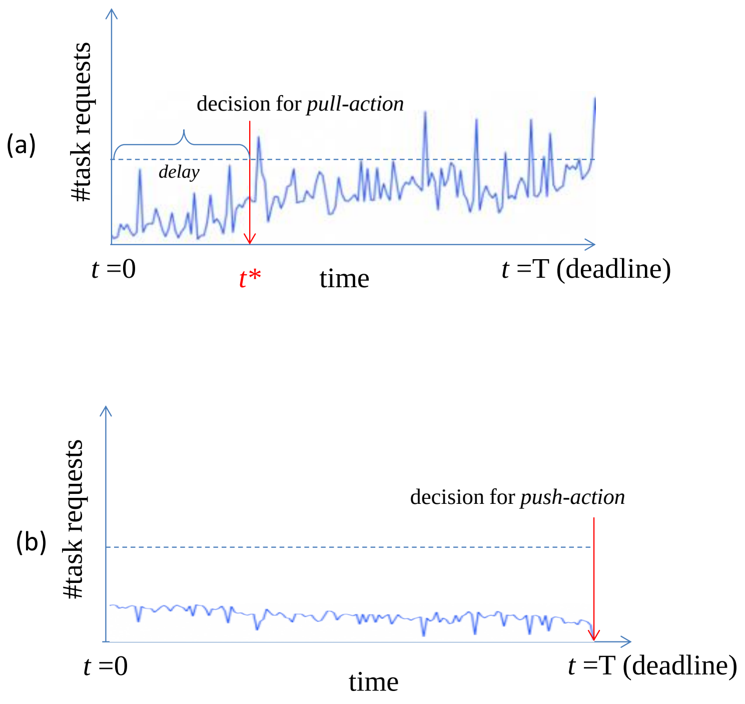

Figure 1. Observing the task requests rate at each time instance t to decide on (a) a high popularity or (b) low popularity of the service and act intelligently based on the history of the task requests.

- We provide a time-optimized intelligent method that enhances the likelihood of making the decision of service replication (pull-action) of the service with the greatest payoff. This is reflected by the cumulative sum of the number of task requests to the node under consideration taking into account the incurred cost.
- We provide theoretical analysis of the optimality of our model based on the principles of the Optimal Stopping Theory.
- We report on a detailed comparative evaluation of our OST-based method against the Ideal Decision ruLe (IDL), which produces the actual maximum payoff, and a heuristic Deterministic Rule (DR).
- A comparative assessment is provided comparing our results with the related work in [55], which is based on the Secretary Optimal Stopping Time problem.

In addition, Figure 2 highlights the state-space of the decision-making. Specifically, Figure 2 describes the mechanism of the seeking the best time for making the decision: either service replication (pull-action) or task offloading (push-action). Mainly, the decision is affected by the number of the received tasks at each time instance t for a specific service.

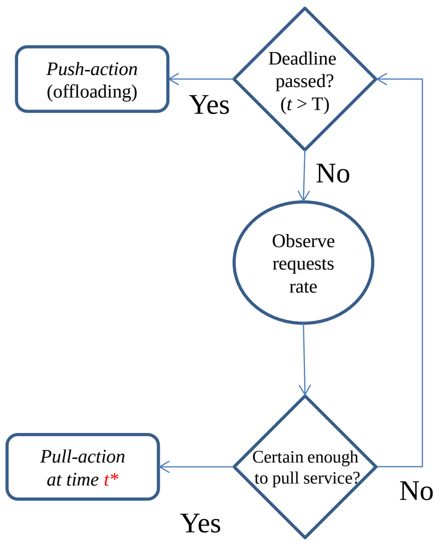
Push-action (offloading)
Yes
Deadline passed? (t > T)
No
Observe requests rate
*Pull-action at time t**
Yes
Certain enough to pull service?
No

Figure 2. Sequential decision-making diagram on either task offloading (push-action) or service replication (pull-action) at the best time $t^* < T$ by the edge node.

The rationale has as follows. The node receives a number of incoming task requests Y_t at every time instance $t < T$. Then, the node has two options:

- If the number of the received requests is high enough, therefore the decision will be to pull the service at a time $t \leq t^* < T$.
- Otherwise, the node reconsiders its decision at the next time $t + 1$, where new service requests are coming.

If the deadline T elapses, then the decision will be to push the task to a peer or the Cloud for further processing. This is the first study that we are aware of that sequential decision making, which considers the decision of pull/push-action as an OST problem in EC environments. The overall goal is to increase the possibility of making the decision of service replication (pull-action) with the greatest payoff. Moreover, since the nodes are located among the servers that have been distributed at the edge of the network, our approach will be applied to EC applications such as Virtual Networks (VN) [56], UAV [7], and data mining applications [6].

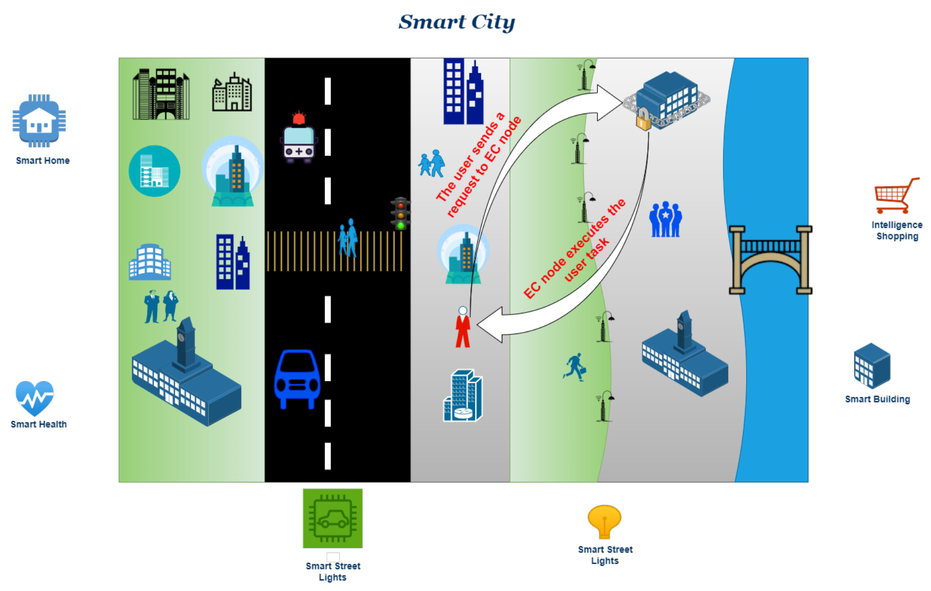

Figure 3. Example of the two actions: pull and push in a Smart City environment.

3. System Model & Problem Formulation

In Section 3.1, we begin by providing an overview of the system model, and then in Section 3.2 we discuss the problem definition. In Section 3.3, the Optimal stopping rule problem is introduced. Finally, in Section 3.4, we will give full details about cost-based sequential decision-making.

3.1. System Model

As seen in Figure 3, a smart city is increasingly establishing vast IoT networks with widely distributed IoT devices that provide vast amounts of services. In light of the vast size and extensively spread nature of IoT networks, EC has become a strong and efficient paradigm for providing processing capabilities to IoT devices at the network's edge [57]. In EC, IoT services may be executed on EC, which offers low delay and reduces the bandwidth load. Nevertheless, it remains difficult to enhance the entire EC execution performance, for instance, resource utilization, EC energy usage, and load balance. However, we believe that the environment of the network is made up of a global cloud data center and an EC system [58][59]. The EC system is constructed of M base stations as defined in table 1, and since each one is supplied with some EC servers, they collectively constitute an EC node. In accordance with it as well, we will refer to the set of EC nodes as $\mathscr{M} = [1,2,...,M]$. Our general assumption is that multiple service providers would deploy their own applications on each EC node. Each edge device has the ability to delegate tasks to an EC node that is equipped with appropriate computing services. In addition, edge devices have the capability of offloading tasks to a global cloud, presuming that the global cloud has all of the required services and a decent amount of computing resources[60]. Edge devices make the decision of task service replication (pull-action) in accordance with the maximum payoff value. The time is segmented into frames denoted by the notation $t = 1, 2, \ldots, T$. A decision of service replication (pull-action) is determined by the edge device at some moment in time t^*, depending on local information (e.g., task information like the number of services that have been received). We rely on the assumption that the location of the edge device and the network environment remain the same throughout each time instance. This allows us to guarantee that the task of replication (pull-action) decision will be effective at the same time instance t.

3.2. Problem Definition

Within the scope of the task model, we rely on the assumption that the formation of the user's tasks follows a Poisson distribution with Y_t : the number of the task's requests at time instance t with the rate λ [61]. Due to the fact that the Edge device is constant, the number of EC nodes that the edge device is capable of sensing and receiving tasks during each time instance t. As a result, the cumulation of the number of requests up to t is applied to every task that are received by the edge device. The decision of service replication (pull-action) is subjected to θ which is a number of tasks that our OST model is based on it to take the action of making the decision. The cumulative distribution function of the poison distribution can be expressed as in (1).

$$P(Y <= y; \lambda) = \sum_{k=0}^{y} \frac{e^{-\lambda} \cdot \lambda^k}{k!} \tag{1}$$

Let us define the cumulation of the number of the requests R_t, $0 \leq t \leq T$ as:

$$R_t = \sum_{k=0}^{t} Y_k. \tag{2}$$

The objective is to get as near as possible to θ. In the case that $R_t > \theta$, the observation is immediately terminated and the service offloading scenario is activated, since we have acquired much more usable information than requested. Clearly, the second scenario is not unwanted, but it results in a 'penalty' that we ought to have prevented before $R_t > \theta$. This penalty is measured by the formula $R_t > \theta$. However, we may include a penalty for the increased latency caused by receiving one more task before stopping. Each time we do not stop, hence not triggering the service replication (pull-action) scenario, we incur a cost greater than zero. This cumulative cost up to t requires the procedure to decide whether to stop or continue with another observation by evaluating the trade-off between extending the observation for potentially more usable data and the extra latency required to activate the service offloading scenario. This price may reflect the urgency of a monitoring application that demands a service offloading option in near real-time. Alternatively, if the application requires very precise service offloading option outcomes, a (limited) delay must be permitted.

Based on the above interpretation, we provide the following payoff function $H_t(R_t)$, which involves (i) the task threshold θ, (ii) the delay cost per monitoring $C \geq 0$.

$$H_t(R_t) = \begin{cases} R_t - C \cdot t & \text{if } R_t \leq \theta \\ 0 & \text{if } R_t > \theta \end{cases} \tag{3}$$

The rationale behind this payoff is that we desire the rate of service demands per time t taking out the delay cost to be significantly increasing. This expresses the likelihood that the pull-action is stochastically a better decision than the push-action. This is because we increase our confidence that in the (near) future we are expecting more requests for a service in a node. Therefore, we aim at maximizing this payoff and, hence, the corresponding likelihood. Our goal is to attain θ by maximizing the number of the received tasks. If R_t is greater than θ, then the return is zero since the mechanism should have triggered the service offloading scenario. Therefore, any additional delay was of no benefit.

It is mandatory to have a base result to be compared with our fundamental approach (OST). Therefore, we define the IDL, which measures the accumulative sum of the received tasks from users to the node. However, each time instance t the expected cost $C \cdot t$ must be in the account. As a result, the instance time t that has the maximum payoff that results from IDL, is the best time t^* instance for making the decision (pull or continue). Moreover, we can express the IDL best time instance t^* as (4):

$$t^* = \{0 \leq t < T : \max_t \sum_{k=0}^{t} Y_k - C \cdot k, \text{ s.t. } \sum_{k=0}^{t} Y_k \leq \theta\} \tag{4}$$

We suggest the deterministic DR. DR is mainly based on calculating the number of received tasks Y after passing the amount of time t. In this rule, the decision of offloading is made if R_t of the received task at time instance t, which is determined by $\alpha \cdot \frac{\theta}{C}$, is less than theta θ. Also, It is important to pay attention to the cost per each time instance $(C \cdot t)$. Then, the decision of service replication (pull-action) is expressed in DR as in (5). The pseudo-code is provided in algorithm 2.

$$t^* = \min\{0 \leq t < T : \sum_{k=0}^{t} Y_k > \theta \text{ or } t > \alpha \frac{\theta}{C}\}. \tag{5}$$

Moreover, our theory has been compared to the work in [55]. Their approach is based on choosing the stopping time using SECPRO model. The stopping time decision in [55] was based on choosing the highest value of the received requests per time instance:

$$Y^* = \max_t (Y_t), \tag{6}$$

by edge node during the period from $t \in \{0, \frac{T}{e}\}$. Furthermore, taking into account calculating the cumulative sum of the number of the received tasks (R_t) and the cost C. Then, the node will resume receiving tasks during the period of $\{\frac{T}{e}, T\}$ where the first value $Y_t > Y^*$ is the preferred stopping time of this theory. If the first value is detected, it stops immediately. This time is considered the stopping time for this theory. Also, we would like to emphasize that during the period $\{\frac{T}{e}, T\}$ the cumulative sum of the received tasks (R_t) is calculated beside the value of the cost as shown in (7). The pseudo-code is provided in algorithm 3.

$$t^* = \min\{\frac{T}{e} \leq t \leq T : \text{ such that } \sum_{k=0}^{t} Y_k \leq \theta \text{ and } Y_t > Y^*\} \tag{7}$$

As a result, we found surprising and noteworthy results that demonstrate the superiority of our theory OST. This is what we will explain later in Section 3.3.

3.3. Sequential Decision Making based on Optimal Stopping Theory

3.3.1. Optimal stopping Rule Problem

The concept of optimal stopping [62] addresses the issue of selecting a time instance to conduct a specific action. The measure is taken to minimize an anticipated loss (or maximize an expected payoff). A stopping rule problem is characterized by: (i) a set of stochastic variables, whose joint distribution is presumed to be known, and (ii) a set of loss functions $(S_t(y_1,\dots,y_t))_{1\leq t}$ or payoff functions $(H_t(y_1,\dots,y_t))_{1\leq t}$ that rely exclusively on the observed values $y_1,\dots,y_t$ of related random variables. An optimal stopping rule problem is expressed as follows: We are monitoring the series of $(Y_t)_{1\leq t}$, and at each time instance t, we decide whether to stop monitoring or proceed. If we cease monitoring at instance t, we incur loss S_t or gain payoff H_t. We would like to pick a stopping rule or time that reduces our estimated loss or, equivalently, optimizes our expected payoff.

Definition 1: The optimal stopping time t^* minimises the incurred loss $E[S_{t^*}] = \inf_{0\leq t\leq T} E[S_t]$. In the case of payoff, t^* maximises the expected payoff, i.e.,

$$E[H_{t^*}] = \sup_{0\leq t\leq T} E[H_t]. \tag{8}$$

Note that T may be ∞. The information accessible up to time t is a series F_t of values of the random variables $Y_t,\dots,Y_t$, which is also known as filtration.

Definition 2: The 1-stage look-ahead (1-sla) stopping rule (time) is defined as:

$$t^* = \inf\{t \geq 0 : H_t \geq E[H_{t+1}|F_t]\}. \tag{9}$$

This means that t^* requires stopping at the first time instance t where the payoff H_t for stopping at t is as large as the anticipated payoff of proceeding to the next time instance $t+1$ and then stopping.

Definition 3: Let B_t represent the occurrence $H_t \geq E[H_{t+1}|F_t]$. A stopping rule problem is monotonous if $B_0 \subset B_1 \subset \dots$ nearly certainly. The set B_t is the set for which the 1-sla rule requires pausing at time instance t. The condition $B_t \subset B_{t+1}$ indicates that if the 1-sla rule requires pausing at time t, it will likewise require stopping at time $t+1$ regardless of the value of y_{t+1}. In a same manner, $B_t \subset B_{t+1} \subset B_{t+2} \subset \dots$ states that if the 1-sla rule requires pausing at time t, it will also require stopping at all future times, regardless of the future observations.

Theorem 1: The 1-sla rule provides the best solution for monotonous stopping rule issues.

Proof. See [63] □

The edge device is required to complete making the decision of service replication (pull-action) of the service of the number of tasks y_t that it has received. Because of this, the primary goal is to increase the payoff attached to the decision of service replication (pull-action). The edge node is actively monitoring a continuous sequence of number of tasks that have been received. These tasks are being accumulated with those that have been received in the past w.r.t. performance criteria, which is denoted by the current delay cost C per time t. At each received new tasks, the node should decide whether to make the decision of service replication (pull-action) or not. The challenge arises from the fact that the node wants to establish a service replication (pull-action) policy that will increase the chances of making the decision of service replication (pull-action) with a high payoff. As a result, the node needs to go for the optimal decision time that maximizes the payoff, which is globally rated as being the same as or near to the actual (ideal) payoff.

3.4. Cost-based Sequential Decision Making

Problem 1: Given the tolerance threshold θ and delay cost $C > 0$, determine the optimal stopping time t^* such that $sup_{1 \leq t \leq \infty} E[H_t]$ is attained. The greatest expected return is then $E[H_t*]$.

The **Problem 1** is resolved by establishing a model based on the OST. Consider that the starting time $t = 0$ and the deadline is T, with $t < T$. Our OST model aims to stop receiving more tasks when the cumulative sum of the received task R_t is near to θ, taking into account the cumulative delay cost up to that point t. We provide a 1-sla stopping rule based on the optimal stopping principle stated in **Theorem 1**, whereby we stop at the first time t such that:

$$H_t(R_t) \geq E[H_{t+1}(R_{t+1})|F_t], \text{ with the event } \{R_t \leq \theta\} \in F_t. \tag{10}$$

That is, any further receiving tasks at time $t+1$ would not contribute to maximizing the payoff. When the difference $E[H_{t+1}(R_{t+1})|F_t] - H_t(R_t)$ is constant non-increasing with R_t, the 1-sla rule is optimal.

Theorem 2: Given a series of payoff random variables $R_1, \ldots, R_t$, the optimal stopping rule t^* for **Problem 1** is provided in (11):

$$t^* = \inf\{t \geq 0 : \sum_{y=0}^{\theta - R_t} (R_t + y) - C(t+1)) \cdot P(Y = y) \leq R_t - C \cdot t\} \tag{11}$$

Proof. Given that $R_t \leq \theta$, the conditional expectation $E[H_{t+1}(R_{t+1})|R_t \leq \theta]$ is given by:

$$\begin{aligned} E[H_{t+1}(R_{t+1})|R_t \leq \theta] &= E_Y[H_{t+1}(R_t + Y)|R_t \leq \theta, R_t + Y \leq \theta]P(R_t + Y \leq \theta) \\ &+ E_Y[H_{t+1}(R_t + Y)|R_t \leq \theta, R_t + Y > \theta]P(R_t + Y > \theta) \\ &= E_Y[(R_t + Y) - c(t+1)|Y \leq -R_t]P(Y \leq \theta - R_t) \\ &= \sum_{y=0}^{\theta - R_t} (R_t + y) - C(t+1)) \cdot P(Y = y). \end{aligned}$$

Therefore, the mechanism terminates at the first time t, where R_t is such that $E[H_{t+1}(R_{t+1})|R_t \leq \theta] \leq R_t - C \cdot t$. □

The 1-sla rule is optimal, since the corresponding difference is constantly non-increasing with R_t when $R_t < \theta$. A high-level description of the provided Theorem 2 is that, it is always better to stop at the first time t rather than at $t+1$, because the current pay off at that t will be bigger than the pay off at the next time instance $t+1$, and any other time $t + \tau$ with $\tau > 1$. This is the principle of 1-sla rule, where our problem falls into this category. Moreover, the difference of the pay off at the optimal time t and $t+1$

Algorithm 1: Time-optimized Sequential Decision Making (OST).

```
Input: Tolerance threshold $\theta$; observation cost $C > 0$.
Output: Optimal stopping time $t^*$.
$top \leftarrow False$; $t \leftarrow 0$; $R_0 \leftarrow 0$;
repeat
    observe number of requests $Y_t$;
    $R_t \leftarrow R_{t-1} + Y_t$ /* update the cumulative sum of requests */ ;
    if criterion in (11) is satisfied then
        Stop $\leftarrow$ True;
        $t^* \leftarrow t$ /* activate service replication (pull-action) and start-off a new task's service*/;
    else
        $t \leftarrow t + 1$ /* continue with the next received task*/ ;
    end
until $t \leq T$
    if $(Stop = False)$
        The (push-action)
    end
```

is always positive. However, that difference tends to decrease with the time t. This indicates that at the very first time t where the pay off is bigger than the future expected pay off, then it is always beneficial to stop at that t and not at any other future time, since the difference of the consecutive pay off values is always decreasing. Therefore, the 1-sla rule in **Theorem 1** is optimum, OST with fixed θ and taking into account the delay cost C may ensure that the anticipated cumulative sum of the received tasks based on the stopping criteria is as near as possible to θ, however, no other stopping rule can make such a claim. Based on the cost C, OST is adaptable for handling and controlling the latency and freshness of the service replication (pull-action) and offloading decision's variables. OST is concerned with the delay cost of monitoring and assumes that all received tasks are current for the application that uses the service replication (pull-action) and offloading decision. In this case, the OST is forced to terminate receiving new tasks to prevent waiting for an extended period of time, particularly when λ is very small. Therefore a large number of number of tasks are necessary to sum up to θ. The pseudo-code is provided in Algorithm 1.

4. Experimental Evaluation

4.1. Experimental Setup

In our experiments, we experiment with four different models: (i) The proposed OST model. (ii) The heuristic DR model, which depends on the heuristic factor α. In DR, the decision of pull-action is taken if the accumulation R_t at time instance t, determined by $\alpha \cdot (\theta/C)$, is greater than θ provided in (5). Otherwise, if the deadline elapses, the decision of offloading is made. (iii) The SECPRO approach introduced in [55]. SECPRO measures the current payoff when the decision has been made using the OST criterion

Algorithm 2: Heuristic decision-making model (DR).

```
Input: Tolerance threshold $\theta$; observation cost $C > 0$; heuristic factor $\alpha \in [0,1]$
Output: decided stopping time $t^*$. $Stop \leftarrow False$; $t \leftarrow 0$; $R_0 \leftarrow 0$;
repeat
    observe number of requests $Y_t$ ;
    $R_t \leftarrow R_{t-1} + Y_t$ /* update the cumulative sum of requests */ ;
    if criterion in (5) is satisfied then
        $Stop \leftarrow True$;
        $t^* \leftarrow t$ /* activate the service replication (pull-action) decision and start-off a new task's service*/;
    else
        $t \leftarrow t + 1$ /* continue with the next received task*/ ;
    end
until $t \leq T$
    if $(Stop = False)$
        The (push-action)
    end
```

in (7). (iv) The IDL is the real expected payoff (H) gained provided in (4). This is the ground truth used to compare all the methods.

In the performance evaluation, we investigate the performance of the models: OST, DR, and IDL. Also, we set the values of for each of λ, α and θ. We assigned $\lambda \in \{3,5,10\}$ and $\theta \in \{50,100,200\}$. Moreover, we performed the experiments into three parts which are divided as the first experiment with λ and θ set to 3 and 50, respectively. Then, the second experiment is set up with $\lambda = 5$ and $\theta = 100$. Finally, $\lambda = 10$ and $\theta = 200$ for the third experiment. Also, the cost C was given a range of values and set these values for the three experiments. In fact, $C = (0.01, 0.05, 0.1, 0.3, 0.5, 0.9) \cdot \lambda$ are the cost values. In addition, The α was a range of numbers applied for all experiments in $(0.01, 0.05, 0.07, 0.09, 0.1, 0.3, 0.45, 0.6, 0.9)$ as shown in Table 2. Moreover, in order to get a more precise result, we did each experiment 1000 times and then we calculate the mean values for all the results.

In our comparative assessment, we performed the experiment over all the models, i.e., OST, DR, IDL, and SECPRO. We set the values of λ , α and θ. we assigned $\lambda \in \{3,30\}$, $\alpha = 0.1$ and $\theta \in \{50,100,150,200\}$ $\theta \in \{600,1200,1800,2400\}$. Moreover, we performed the experiment in two stages. The fourth experiment, which is the first stage, was set up with $\lambda = 3$, $\alpha = 0.1$, and $\theta \in \{50,100,150,200\}$. Furthermore, the second stage, which is the fifth experiment, was set up with $\lambda = 30$, $\alpha = 0.1$, and $\theta \in \{600,1200,1800,2400\}$ as shown in Table 2. We run the experiments 1000 times and took the average values for all the obtained results.

4.2. Performance Evaluation

4.2.1. Experiment 1

The settings for the Experiment 1 are: $\lambda = 3$ and $\theta = 50$ (low service demand rate). Here, we performed the experiment with the defined λ and θ. Our experiment is mainly affected by the cost C and α. The experiment measures the payoff and the time that

Algorithm 3: Secretary-based OST Model (SECPRO).

```
Input: Tolerance threshold $\theta$; observation cost $C > 0$.
Output: Secretary problem's optimal stopping time $t^*$.
/* the algorithm observes task requests in the duration $[0, \frac{T}{\exp}]$ */;
$Stop \leftarrow False$; $t \leftarrow 0$; $R_0 \leftarrow 0$;$Y^* \leftarrow 0$;
for $t = (0 : \frac{T}{e})$
    observe number of requests $Y_t$ ;
    if $Y^* < y_t$ then
        $Y^* = y_t$
    end
    $R_t \leftarrow R_{t-1} + Y_t$ /* update the cumulative sum of requests */;
end
/* then the algorithm observes requests in the interval $t \in \{\frac{T}{\exp}, T\}$ */;
for $t = (\frac{T}{e} + 1 : T)$
    observe number of requests $Y_t$ ;
    if $R^t > \Theta$ then
        Push-action
      end
      $R_t \leftarrow R_{t-1} + Y_t$ /* update the cumulative sum of requests */;
    if criterion in (7) is satisfied then
      $Stop \leftarrow True$;
      $t^* \leftarrow t$ /* activate the service replication (pull-action) decision and start off a new task's service*/;
    else
      $t \leftarrow t + 1$ /* continue with the next received task*/ ;
    end
until $t \leq T$
    if $(Stop = False)$
        The (push-action)
    end
```

NOTATIONS	Definition
$\mathscr{M}$	Set of EC nodes.
T	Deadline.
t	Time instance.
t^*	Optimal stopping time.
Y_t	Number of tasks received at time t.
λ	Arrival rate of tasks Y.
θ	Tolerance threshold.
C	Cost of the decision delay.
R_t	Accumulation of number of tasks Y_t.
α	Heuristic factor in DR model.
H_t	Payoff at time instance t.

Table 1. Notations of parameters

Notation	Parameter	Value/Range
T	Deadline	100.
λ	Arrival rate of tasks	$\{3, 10, 30\}$.
θ	Tolerance threshold	$\{50, 100, 150, 200, 600, 1200, 1800, 2400\}$.
C	Delay cost	$\{0.1, 0.3, 0.5, 0.6, 0.9\}$.
α	Heuristic factor	$\{0.01, 0.05, 0.07, 0.09, 0.1, 0.3, 0.5, 0.6, 0.9\}$.

Table 2. Experimental Parameter Setting.

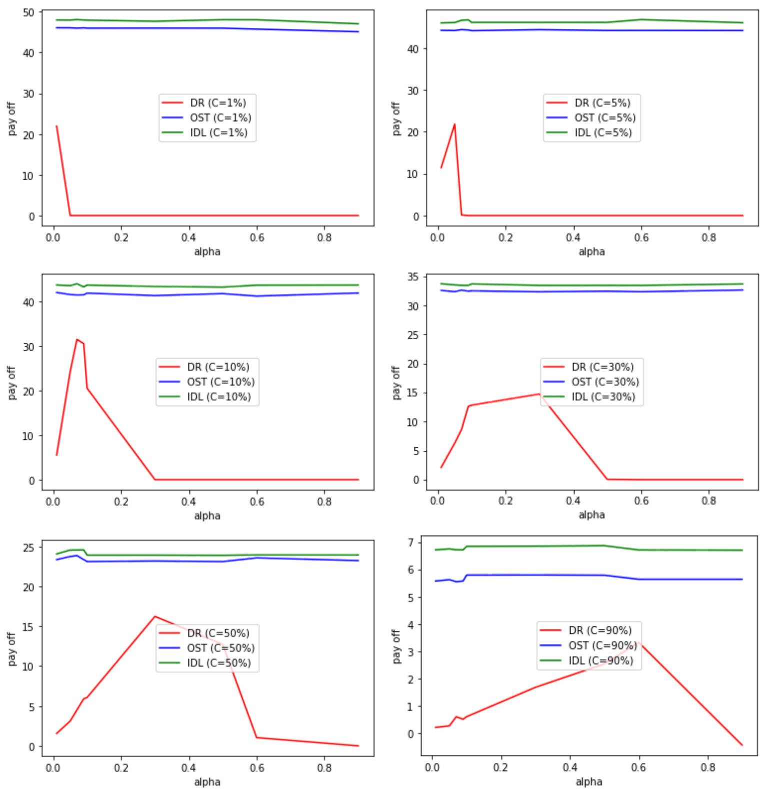

Figure 4. (Experiment 1) The expected payoff vs delay cost with low service demand rate $\lambda = 3$ and tolerance $\theta = 50$ for OST, DR and IDL models.

decision has been made. Moreover, the experiment is performed over three models. The first model is IDL which is being chosen to be the baseline to be compared with the other two models. The second model is our fundamental approach which is OST. Finally, the third model is DR. The experiment test three different models which are the real Figure 4 shows the different effects of the cost all over the three models and obviously shows the result of payoff and how our approach is as close as IDL compared with DR.

4.2.2. Experiment 2

The settings for the Experiment 2 are: $\lambda = 10$ and $\theta = 200$ (medium service demand rate). Here, we performed the experiment with the defined λ and θ. Our experiment is

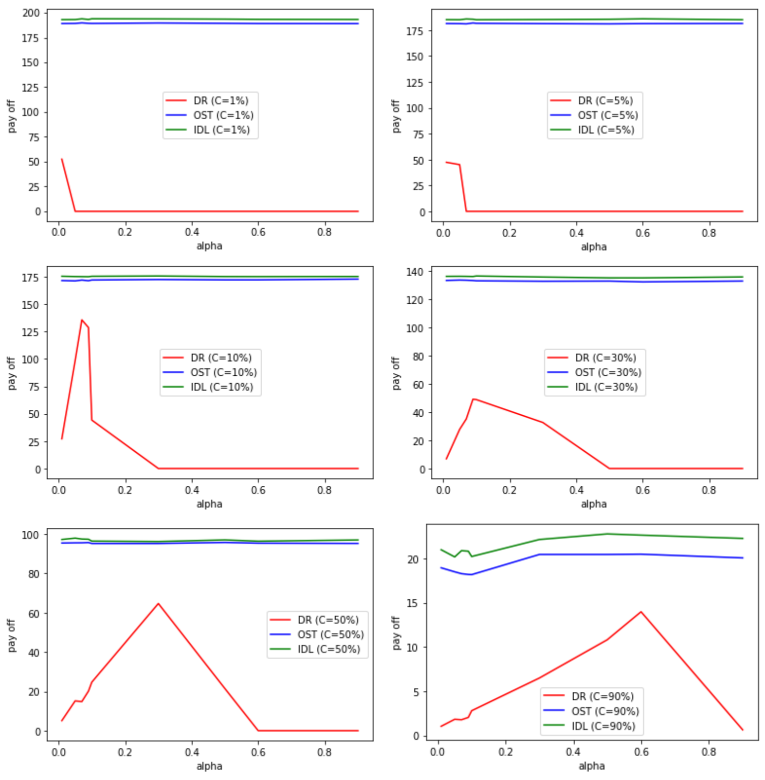

Figure 5. (Experiment 2) The expected payoff vs delay cost with medium service demand rate $\lambda = 10$ and tolerance $\theta = 200$ for OST, DR and IDL models.

mainly affected by the cost C and α. The experiment measures the payoff and the time that decision has been made. Moreover, the experiment is performed over three models. The first model is the IDL model which is being chosen to be the baseline to be compared with the other two models. The second model is our fundamental approach which is OST. Finally, the third model is the DR model. Figure 5 shows the different effects of the cost all over the three models and obviously shows the result of payoff and how our approach is as close as IDL compared with DR.

4.2.3. Experiment 3

The settings for the Experiment 3 are: $\lambda = 30$ and $\theta = 600$ (high service demand rate). We performed the experiment with the defined λ and θ. Our experiment is mainly affected by the cost C and α. The experiment measures the payoff and the time that decision has been made. Moreover, the experiment is performed over three models. The first model is IDL which is being chosen to be the baseline to be compared with the other two models. The second model is our fundamental approach which is OST. Finally, the third model is DR. Figure 6 shows the different effects of the cost all over the three models and obviously shows the result of payoff and how our approach is as close as IDL compared with DR.

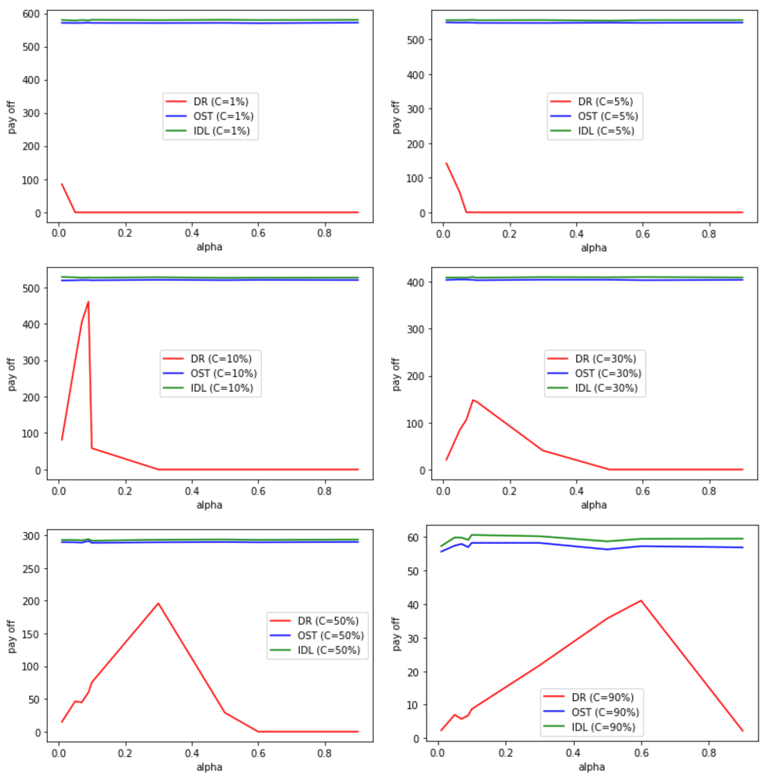

Figure 6. (Experiment 3) The expected payoff vs delay cost with high service demand rate $\lambda = 30$ and tolerance $\theta = 600$ for OST, DR and IDL models.

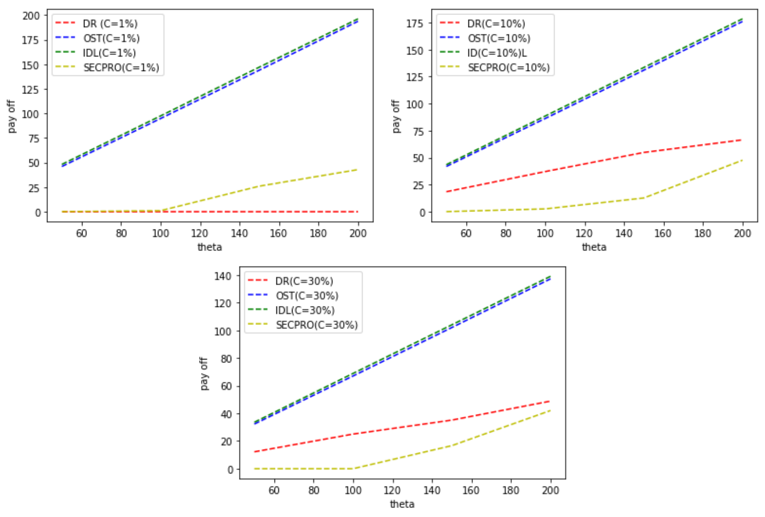

Figure 7. Expected payoff vs tolerance θ for diverse delay cot values with low service demand rate $\lambda = 3$ and tolerance $\theta \in \{50, 100, 150, 200\}$ for DR, OST, IDL and SECPRO models.

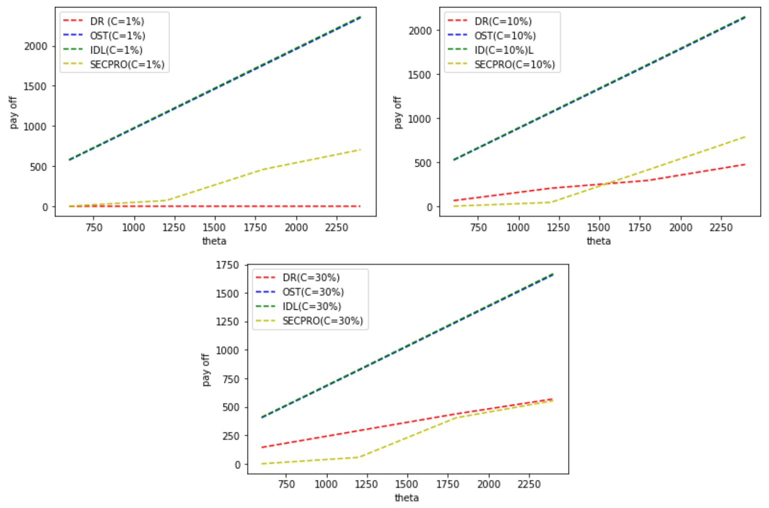

Figure 8. Expected payoff vs tolerance θ for diverse delay cost values with high service demand rate $\lambda = 30$ and tolerance $\theta \in \{600, 1200, 1800, 2400\}$ for DR, OST, IDL and SECPRO models.

4.3. Comparative Assessment

The settings for the comparative assessment experiment 1 are: $\lambda = 3$, $\theta \in \{50, 100, 150, 200\}$ and $C = (1\%, 10\%, 30\%) \cdot \lambda$ (low service demand rate). We performed the experiment with the defined λ and $\alpha = 0.1$. Our experiment is mainly affected by diffident values of θ and C. Furthermore, the experiment measures the payoff V^* and the time t^* that decision has been made. Moreover, the experiment was performed over four models. The first model is IDL which is being chosen to be the baseline. It used to be compared with the other three models. The second model is our fundamental approach which is OST. Then, the third model is DR. Finally, the fourth model is SECPRO. The experiment tested four different models. Figure 7 shows the different effects of θ over all four models. Moreover, Figure 7 obviously shows the result of payoff and how our approach is as close as IDL compared with DR and SECPRO.

The settings for the comparative assessment experiment 2 are: $\lambda = 30$, $\theta \in \{600, 1200, 1800, 2400\}$ and $C = (1\%, 10\%, 30\%) \cdot \lambda$ (high service demand rate): Here, we performed the experiment with the defined λ and $\alpha = 0.1$. Our experiment is mainly affected by diffident values of θ and C. Furthermore, the experiment measures the payoff V^* and the time t^* that decision has been made. Moreover, the experiment was performed over four models. The first model is IDL which is being chosen to be the baseline. It used to be compared with the other three models. The second model is our fundamental approach which is OST. Then, the third model is DR. Finally, the fourth model is SECPRO. The experiment tested four different models. Figure8 shows the different effects of θ over all four models. Moreover, Figure 8 obviously shows the result of payoff and how our approach is as close as IDL compared with DR and SECPRO.

In order to test the effectiveness of our theory (OST) in the first three experiments, where its results were compared with the IDL and another rule is DR. They were all tested on different parameters, which start from normal requests to medium requests and end with dense requests. Moreover, the results of our approach (OST) are amazing, which

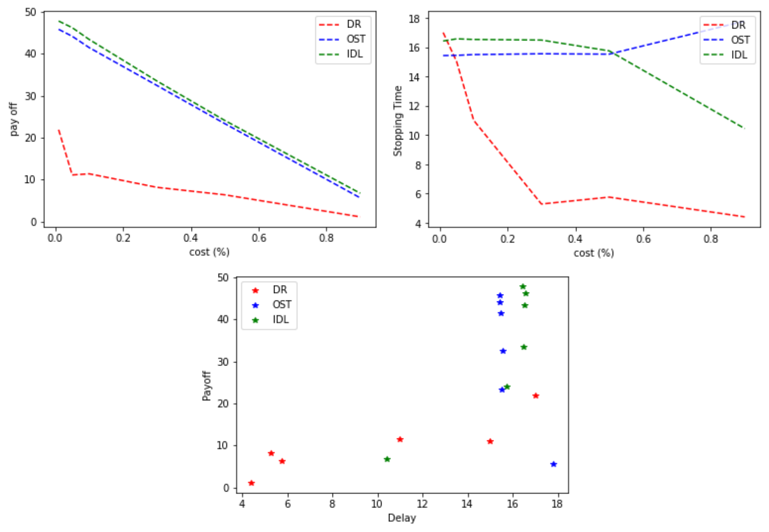

Figure 9. Expected payoff vs cost and delay for OST, IDL (low service demand rate $\lambda = 3$ and tolerance $\theta = 50$).

prove the effectiveness of the theory. As shown in Figures 9,10 and 11, it obvious that Figure 9 represents $\lambda = 3$ and $\theta = 50$, as well as Figure 10 shows $\lambda = 10$ and $\theta = 200$. Also, Figure 11 shows $\lambda = 30$ and $\theta = 600$. In fact, each Figure contains three graphs, which represent the cost versus payoff, the cost versus stopping time, and finally delay versus payoff.

In Figure 9, we see the results of our theory (OST), which are very close to IDL, and this is illustrated in the graphic that presents cost versus payoff. However, after increasing the cost to more than (0.5) cost in the figure that shows the cost versus the Stopping time, we find there is a small difference, and despite that, our approach obtains a delay time less than IDL in all results where the cost is (0.5) or less, and this is what makes it unique.

In Figure 10, the receiving of requests were increased by making $\lambda = 10$, where we notice in the graph that shown the cost versus the payoff is an almost perfect match between our theory (OST) and IDL. Also, the delay time difference decreased after more than (0.5) the cost, however, we obtained a perfect stopping time between our theory (OST) and IDL as shown in the graph that illustrates the cost versus the stopping time as well as the graph showing the delay time versus the payoff.

Figure 11 shows the robustness and effectiveness of our theory (OST). It is shown clearly OST's ability to deal with heavy demands where $\lambda = 30$. The results were amazing even after increasing the cost by more than (0.5) and this is obvious in the graph that shows the cost versus the stopping time and the graph that shows the delay time against the payoff. As the delay in our theory (OST) is very close to the delay for IDL even after (0.5) cost. We also note in the graph that presents cost versus payoff, a great match between our theory (OST) and IDL.

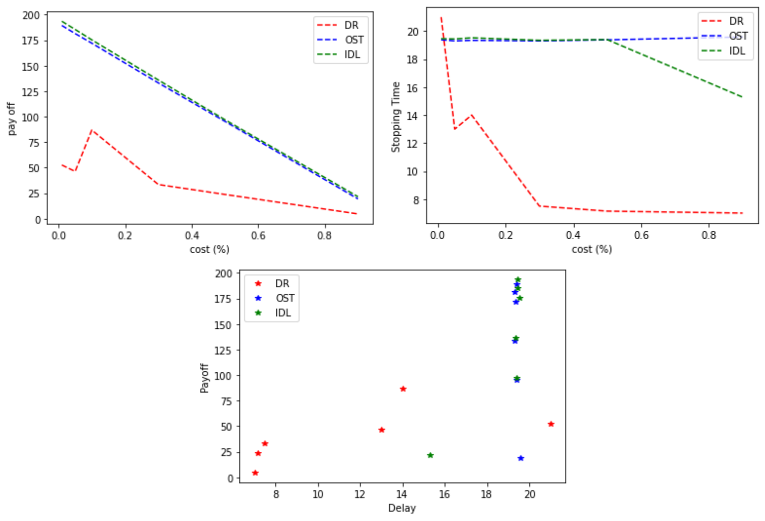

Figure 10. Expected payoff vs cost and delay for OST, IDL (medium service demand rate $\lambda = 10$ and tolerance $\theta = 200$).

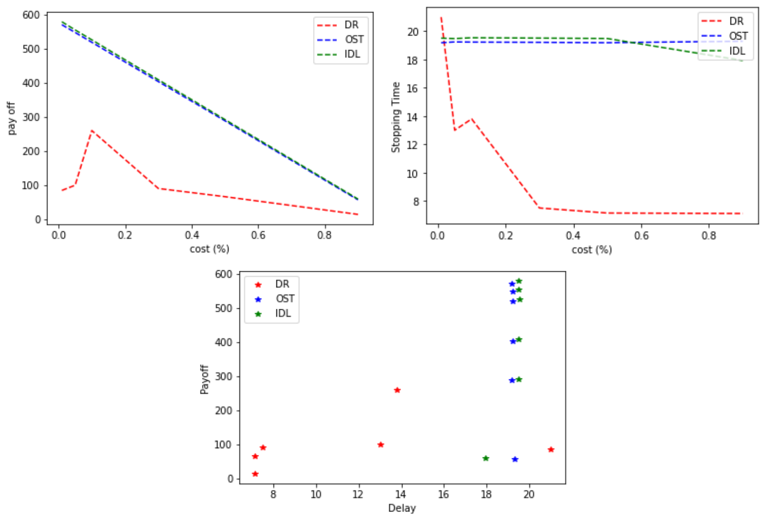

Figure 11. Expected payoff vs cost and delay for OST, IDL (high service demand rate $\lambda = 30$ and tolerance $\theta = 600$).

Moreover, we tested the effectiveness of our theory (OST) in the fourth & fifth experiments, where its results were compared with the three methodologies which are IDL, DR, and SECPRO. They were all tested on two different parameters, which are low & high requests. Moreover, the results of our approach (OST) are brilliant, which proves the effectiveness of our theory OST. As shown in Figures 12 and 13, it is obvious that Figure 12 represents $\lambda = 3$, $\theta \in \{50, 100, 150, 200\}$ and $C = (1\%, 10\%, 30\%) \cdot \lambda$, as well as Figure 13 shows $\lambda = 30$, $\theta \in \{600, 1200, 1800, 2400\}$ and $C = (1\%, 10\%, 30\%) \cdot \lambda$.

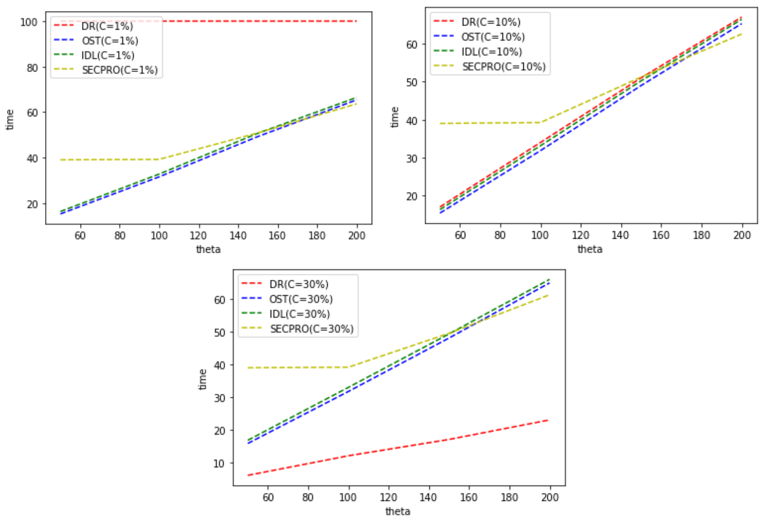

Figure 12. Expected delay (time to decision) vs tolerance θ for diverse cost values (low service demand rate $\lambda = 3$ and tolerance $\theta \in \{50, 100, 150, 200\}$.

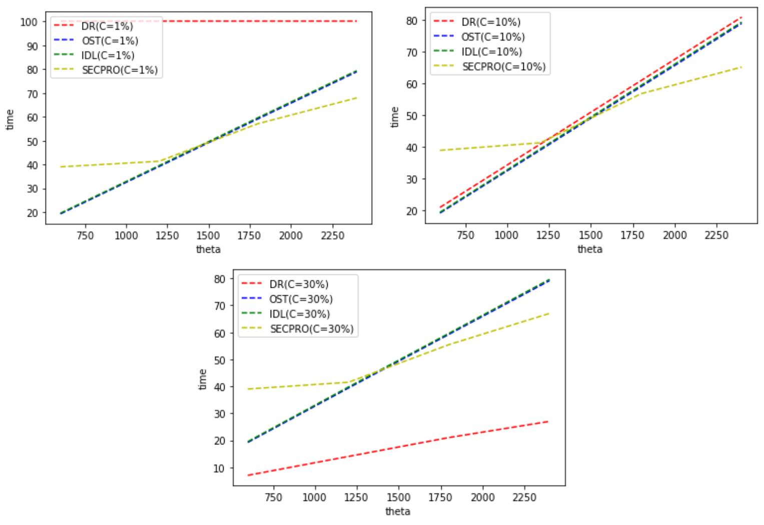

Figure 13. Expected delay (time to decision) vs tolerance θ for diverse cost values (high service demand rate $\lambda = 30$ and tolerance $\theta \in \{600, 1200, 1800, 2400\}$

.

In fact, each figure contains three graphs, which represent the time of the decision being made t^* versus payoff.

5. Conclusions

We provide a superior decision-making approach for users in EC that is based on the Optimal Stopping Theory. When using and adopting the OST in time-optimized decision-making for a service replication (pull-action) in EC environments, we offer a comprehensive review of the process. Based on the results of our experimental assessments, the OST model performs more effectively than the other approaches, which are suitable for use in Edge nodes, and do not demand for a significant amount of resources. Moreover, the highest payoff is obtained by our models, which outperform alternative baseline solutions. In further work, our aim is to analyze how the characteristics of mobile applications and data timeliness affect our OST models and build new ones that take these limitations into account.

ACKNOWLEDGMENT

The authors would like to thank the Saudi Prime Minister of Defense (Crown Prince Mohammad Bin Salman), the Royal Saudi Air Force, Senior Engineer Mohammad Aleissa, Saudi Arabia, and the Saudi Arabian Cultural Bureau in the UK for their support and encouragement.

References

[1] Mayuko Sato, Yoshikazu Fukuyama, Tatsuya Iizaka, and Tetsuro Matsui. Total optimization of energy networks in a smart city by multi-swarm differential evolutionary particle swarm optimization. *IEEE Transactions on Sustainable Energy*, 10(4):2186–2200, 2018.

[2] Mingyang Sun, Yi Wang, Goran Strbac, and Chongqing Kang. Probabilistic peak load estimation in smart cities using smart meter data. *IEEE Transactions on Industrial electronics*, 66(2):1608–1618, 2018.

[3] Ching-Chun Liao, Ting-Sheng Chen, and An-Yeu Wu. Real-time multi-user detection engine design for iot applications via modified sparsity adaptive matching pursuit. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 66(8):2987–3000, 2019.

[4] Qi Chen, Wei Wang, Fangyu Wu, Suparna De, Ruili Wang, Bailing Zhang, and Xin Huang. A survey on an emerging area: Deep learning for smart city data. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 3(5):392–410, 2019.

[5] Xiaolong Xu, Qihe Huang, Xiaochun Yin, Mahdi Abbasi, Mohammad Reza Khosravi, and Lianyong Qi. Intelligent offloading for collaborative smart city services in edge computing. *IEEE Internet of Things Journal*, 7(9):7919–7927, 2020.

[6] Kostas Kolomvatsos, Christos Anagnostopoulos, Maria Koziri, and Thanasis Loukopoulos. Proactive & time-optimized data synopsis management at the edge. *IEEE Transactions on Knowledge and Data Engineering*, 2020.

[7] Ibrahim Alghamdi, Christos Anagnostopoulos, and Dimitrios P Pezaros. On the optimality of task offloading in mobile edge computing environments. In *2019 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6. IEEE, 2019.

[8] Zhi Zhou, Xu Chen, En Li, Liekang Zeng, Ke Luo, and Junshan Zhang. Edge intelligence: Paving the last mile of artificial intelligence with edge computing. *Proceedings of the IEEE*, 107(8):1738–1762, 2019.

[9] Olakunle Elijah, Tharek Abdul Rahman, Igbafe Orikumhi, Chee Yen Leow, and MHD Nour Hindia. An overview of internet of things (iot) and data analytics in agriculture: Benefits and challenges. *IEEE Internet of things Journal*, 5(5):3758–3773, 2018.

[10] Navjeet Kaur, Ayush Mittal, Ashok Kumar, and Rajesh Kumar. Healthcare monitoring through fog computing: A survey. *ECS Transactions*, 107(1):7689, 2022.

[11] Zhe Yu, Yanmin Gong, Shimin Gong, and Yuanxiong Guo. Joint task offloading and resource allocation in uav-enabled mobile edge computing. *IEEE Internet of Things Journal*, 7(4):3147–3159, 2020.

[12] Jianyu Wang, Jianli Pan, Flavio Esposito, Prasad Calyam, Zhicheng Yang, and Prasant Mohapatra. Edge cloud offloading algorithms: Issues, methods, and perspectives. *ACM Computing Surveys (CSUR)*, 52(1):1–23, 2019.

[13] Jun Zhang, Yichuan Wang, Shuyang Li, and Shuaiyi Shi. An architecture for iot-enabled smart transportation security system: a geospatial approach. *IEEE Internet of Things Journal*, 8(8):6205–6213, 2020.

[14] Saurabh Singh, In-Ho Ra, Weizhi Meng, Maninder Kaur, and Gi Hwan Cho. Sh-blockcc: A secure and efficient internet of things smart home architecture based on cloud computing and blockchain technology. *International Journal of Distributed Sensor Networks*, 15(4):1550147719844159, 2019.

[15] Malong Ke, Zhen Gao, Yongpeng Wu, Xiqi Gao, and Kai-Kit Wong. Massive access in cell-free massive mimo-based internet of things: Cloud computing and edge computing paradigms. *IEEE Journal on Selected Areas in Communications*, 39(3):756–772, 2020.

[16] Dmitrii Chemodanov, Flavio Esposito, Andrei Sukhov, Prasad Calyam, Huy Trinh, and Zakariya Oraibi. Agra: Ai-augmented geographic routing approach for iot-based incident-supporting applications. *Future Generation Computer Systems*, 92:1051–1065, 2019.

[17] Qinglin Qi and Fei Tao. A smart manufacturing service system based on edge computing, fog computing, and cloud computing. *IEEE Access*, 7:86769–86777, 2019.

[18] Najmul Hassan, Saira Gillani, Ejaz Ahmed, Ibrar Yaqoob, and Muhammad Imran. The role of edge computing in internet of things. *IEEE communications magazine*, 56(11):110–115, 2018.

[19] Christos Anagnostopoulos, Stathes Hadjiefthymiades, and Kostas Kolomvatsos. Accurate, dynamic, and distributed localization of phenomena for mobile sensor networks. *ACM Transactions on Sensor Networks (TOSN)*, 12(2):1–59, 2016.

[20] Kostas Kolomvatsos, Christos Anagnostopoulos, and Stathes Hadjiefthymiades. Data fusion and type-2 fuzzy inference in contextual data stream monitoring. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 47(8):1839–1853, 2016.

[21] Kostas Kolomvatsos, Christos Anagnostopoulos, and Stathes Hadjiefthymiades. An efficient environmental monitoring system adopting data fusion, prediction, & fuzzy logic. In *2015 6th International Conference on Information, Intelligence, Systems and Applications (IISA)*, pages 1–6. IEEE, 2015.

[22] Kostas Kolomvatsos, Christos Anagnostopoulos, and Stathes Hadjiefthymiades. Distributed localized contextual event reasoning under uncertainty. *IEEE Internet of Things Journal*, 4(1):183–191, 2016.

[23] Quan Yuan, Jinglin Li, Haibo Zhou, Tao Lin, Guiyang Luo, and Xuemin Shen. A joint service migration and mobility optimization approach for vehicular edge computing. *IEEE Transactions on Vehicular Technology*, 69(8):9041–9052, 2020.

[24] Zhipeng Gao, Qidong Jiao, Kaile Xiao, Qian Wang, Zijia Mo, and Yang Yang. Deep reinforcement learning based service migration strategy for edge computing. In *2019 IEEE international conference on service-oriented system engineering (SOSE)*, pages 116–1165. IEEE, 2019.

[25] Qi Gu, Yiheng Jian, Gongpu Wang, Rongfei Fan, Hai Jiang, and Zhangdui Zhong. Mobile edge computing via wireless power transfer over multiple fading blocks: An optimal stopping approach. *IEEE Transactions on Vehicular Technology*, 69(9):10348–10361, 2020.

[26] Kostas Kolomvatsos and Christos Anagnostopoulos. A proactive statistical model supporting services and tasks management in pervasive applications. *IEEE Transactions on Network and Service Management*, 2022.

[27] Farouk Messaoudi, Adlen Ksentini, and Philippe Bertin. On using edge computing for computation offloading in mobile network. In *GLOBECOM 2017-2017 IEEE Global Communications Conference*, pages 1–7. IEEE, 2017.

[28] Davide Callegaro and Marco Levorato. Optimal edge computing for infrastructure-assisted uav systems. *IEEE Transactions on Vehicular Technology*, 70(2):1782–1792, 2021.

[29] Md Golam Rabiul Alam, Mohammad Mehedi Hassan, Md ZIa Uddin, Ahmad Almogren, and Giancarlo Fortino. Autonomic computation offloading in mobile edge for iot applications. *Future Generation Computer Systems*, 90:149–157, 2019.

[30] Thinh Quang Dinh, Jianhua Tang, Quang Duy La, and Tony QS Quek. Offloading in mobile edge computing: Task allocation and computational frequency scaling. *IEEE Transactions on Communications*, 65(8):3571–3584, 2017.

[31] Thinh Quang Dinh, Quang Duy La, Tony QS Quek, and Hyundong Shin. Learning for computation

offloading in mobile edge computing. *IEEE Transactions on Communications*, 66(12):6353–6367, 2018.

[32] Li Kuang, Tao Gong, Shuyin OuYang, Honghao Gao, and Shuiguang Deng. Offloading decision methods for multiple users with structured tasks in edge computing for smart cities. *Future Generation Computer Systems*, 105:717–729, 2020.

[33] Feng Wang, Jie Xu, Xin Wang, and Shuguang Cui. Joint offloading and computing optimization in wireless powered mobile-edge computing systems. *IEEE Transactions on Wireless Communications*, 17(3):1784–1797, 2017.

[34] Min Chen and Yixue Hao. Task offloading for mobile edge computing in software defined ultra-dense network. *IEEE Journal on Selected Areas in Communications*, 36(3):587–597, 2018.

[35] Qi Qi, Jingyu Wang, Zhanyu Ma, Haifeng Sun, Yufei Cao, Lingxin Zhang, and Jianxin Liao. Knowledge-driven service offloading decision for vehicular edge computing: A deep reinforcement learning approach. *IEEE Transactions on Vehicular Technology*, 68(5):4192–4203, 2019.

[36] Xiaoge Huang, Ke Xu, Chenbin Lai, Qianbin Chen, and Jie Zhang. Energy-efficient offloading decision-making for mobile edge computing in vehicular networks. *EURASIP Journal on Wireless Communications and Networking*, 2020(1):1–16, 2020.

[37] Taha Alfakih, Mohammad Mehedi Hassan, Abdu Gumaei, Claudio Savaglio, and Giancarlo Fortino. Task offloading and resource allocation for mobile edge computing by deep reinforcement learning based on sarsa. *IEEE Access*, 8:54074–54084, 2020.

[38] Ming Tang and Vincent WS Wong. Deep reinforcement learning for task offloading in mobile edge computing systems. *IEEE Transactions on Mobile Computing*, 2020.

[39] Ling Tang and Shibo He. Multi-user computation offloading in mobile edge computing: A behavioral perspective. *IEEE Network*, 32(1):48–53, 2018.

[40] Wei Chang, Yang Xiao, Wenjing Lou, and Guochu Shou. Offloading decision in edge computing for continuous applications under uncertainty. *IEEE Transactions on Wireless Communications*, 19(9):6196–6209, 2020.

[41] Zhufang Kuang, Linfeng Li, Jie Gao, Lian Zhao, and Anfeng Liu. Partial offloading scheduling and power allocation for mobile edge computing systems. *IEEE Internet of Things Journal*, 6(4):6774–6785, 2019.

[42] Yanting Wang, Min Sheng, Xijun Wang, Liang Wang, and Jiandong Li. Mobile-edge computing: Partial computation offloading using dynamic voltage scaling. *IEEE Transactions on Communications*, 64(10):4268–4282, 2016.

[43] Ying Chen, Ning Zhang, Yongchao Zhang, Xin Chen, Wen Wu, and Xuemin Shen. Energy efficient dynamic offloading in mobile edge computing for internet of things. *IEEE Transactions on Cloud Computing*, 9(3):1050–1060, 2019.

[44] Yijin Pan, Ming Chen, Zhaohui Yang, Nuo Huang, and Mohammad Shikh-Bahaei. Energy-efficient noma-based mobile edge computing offloading. *IEEE Communications Letters*, 23(2):310–313, 2018.

[45] Shangguang Wang, Jinliang Xu, Ning Zhang, and Yujiong Liu. A survey on service migration in mobile edge computing. *IEEE Access*, 6:23511–23528, 2018.

[46] Paolo Bellavista, Alessandro Zanni, and Michele Solimando. A migration-enhanced edge computing support for mobile devices in hostile environments. In *2017 13th International Wireless Communications and Mobile Computing Conference (IWCMC)*, pages 957–962. IEEE, 2017.

[47] Md Golam Rabiul Alam, Yan Kyaw Tun, and Choong Seon Hong. Multi-agent and reinforcement learning based code offloading in mobile fog. In *2016 International Conference on Information Networking (ICOIN)*, pages 285–290. IEEE, 2016.

[48] Shangguang Wang, Yan Guo, Ning Zhang, Peng Yang, Ao Zhou, and Xuemin Shen. Delay-aware microservice coordination in mobile edge computing: A reinforcement learning approach. *IEEE Transactions on Mobile Computing*, 20(3):939–951, 2019.

[49] Tao Ouyang, Zhi Zhou, and Xu Chen. Follow me at the edge: Mobility-aware dynamic service placement for mobile edge computing. *IEEE Journal on Selected Areas in Communications*, 36(10):2333–2345, 2018.

[50] Yuxuan Sun, Sheng Zhou, and Jie Xu. Emm: Energy-aware mobility management for mobile edge computing in ultra dense networks. *IEEE Journal on Selected Areas in Communications*, 35(11):2637–2646, 2017.

[51] Jinliang Xu, Xiao Ma, Ao Zhou, Qiang Duan, and Shangguang Wang. Path selection for seamless service migration in vehicular edge computing. *IEEE Internet of Things Journal*, 7(9):9040–9049, 2020.

[52] Shiqiang Wang, Rahul Urgaonkar, Murtaza Zafer, Ting He, Kevin Chan, and Kin K Leung. Dynamic

service migration in mobile edge computing based on markov decision process. *IEEE/ACM Transactions on Networking*, 27(3):1272–1288, 2019.

[53] Tao Ouyang, Rui Li, Xu Chen, Zhi Zhou, and Xin Tang. Adaptive user-managed service placement for mobile edge computing: An online learning approach. In *IEEE INFOCOM 2019-IEEE conference on computer communications*, pages 1468–1476. IEEE, 2019.

[54] Yuxuan Sun, Xueying Guo, Sheng Zhou, Zhiyuan Jiang, Xin Liu, and Zhisheng Niu. Learning-based task offloading for vehicular cloud computing systems. In *2018 IEEE International Conference on Communications (ICC)*, pages 1–7. IEEE, 2018.

[55] Ibrahim Alghamdi, Christos Anagnostopoulos, and Dimitrios P Pezaros. Data quality-aware task offloading in mobile edge computing: an optimal stopping theory approach. *Future Generation Computer Systems*, 117:462–479, 2021.

[56] Haibo Zhang, Zixin Wang, and Kaijian Liu. V2x offloading and resource allocation in sdn-assisted mec-based vehicular networks. *China Communications*, 17(5):266–283, 2020.

[57] Xiaolong Xu, Xihua Liu, Zhanyang Xu, Fei Dai, Xuyun Zhang, and Lianyong Qi. Trust-oriented iot service placement for smart cities in edge computing. *IEEE Internet of Things Journal*, 7(5):4084–4091, 2019.

[58] Shuyu Chen, Haopeng Chen, Jinteng Ruan, and Ziming Wang. Context-aware online offloading strategy with mobility prediction for mobile edge computing. In *2021 International Conference on Computer Communications and Networks (ICCCN)*, pages 1–9. IEEE, 2021.

[59] Zezu Liang, Yuan Liu, Tat-Ming Lok, and Kaibin Huang. Service migration for multi-cell mobile edge computing. In *GLOBECOM 2020-2020 IEEE Global Communications Conference*, pages 1–6. IEEE, 2020.

[60] Amir Erfan Eshratifar and Massoud Pedram. Energy and performance efficient computation offloading for deep neural networks in a mobile cloud computing environment. In *Proceedings of the 2018 on Great Lakes Symposium on VLSI*, pages 111–116, 2018.

[61] Fei Sun, Fen Hou, Nan Cheng, Miao Wang, Haibo Zhou, Lin Gui, and Xuemin Shen. Cooperative task scheduling for computation offloading in vehicular cloud. *IEEE Transactions on Vehicular Technology*, 67(11):11049–11061, 2018.

[62] Christos Anagnostopoulos and Kostas Kolomvatsos. A delay-resilient and quality-aware mechanism over incomplete contextual data streams. *Information Sciences*, 355:90–109, 2016.

[63] Goran Peskir and Albert Shiryaev. *Optimal stopping and free-boundary problems*. Springer, 2006.