User preference–based QoS-aware service function placement in IoT-Edge cloud

Briytone Mutichiro and Younghan Kim

Abstract
In the Internet of Things-Edge cloud, service provision presents a challenge to operators to satisfy user service-level agreements while meeting service-specific quality-of-service requirements. This is because of inherent limitations in the Internet of Things-Edge in terms of resource infrastructure as well as the complexity of user requirements in terms of resource management in a heterogeneous environment like edge. An efficient solution to this problem is service orchestration and placement of service functions to meet user-specific requirements. This work aims to satisfy user quality of service through optimizing the user response time and cost by factoring in the workload variation on the edge infrastructure. We formulate the service function placement at the edge problem. We employ user service request patterns in terms of user preference and service selection probability to model service placement. Our framework proposal relies on mixed-integer linear programming and heuristic solutions. The main objective is to realize a reduced user response time at minimal overall cost while satisfying the user service requirements. For this, several parameters, and factors such as capacity, latency, workload, and cost constraints, are considered. The proposed solutions are evaluated based on different metrics and the obtained results show the gap between the heuristic user preference placement algorithm and the optimal solution to be minimal.

Keywords
IoT-Edge, user preference, QoS optimization, similarity index, service function placement

Date received: 10 December 2020; accepted: 30 April 2021

Handling Editor: Yanjiao Chen

Introduction
Edge computing is an architectural development of computing with access to a shared pool of configurable, ubiquitous, and decentralized computing resources and services in close proximity to users. The Internet of Things-Edge (IoT-Edge) cloud is a distributed hierarchical network with partitioned layers of IoT/sensor devices, intermediate layer of processing servers, and a more centrally located core cloud for data processing and storage. To support the IoT and critical applications, edge computing needs to ensure reliability, availability, efficiency, high performance, self-healing, and cost-effective deployment. In IoT-Edge cloud computing, service provision presents a challenge to operators to meet the service-specific quality-of-service (QoS) requirements and satisfy service-level agreements (SLAs) with the users. Challenges such as limited resources, network congestion, large network latencies, and large amounts of generated data traffic contribute to lowering the user QoS, which can be mitigated by the optimal placement of service functions (SFs) at the edge network. Different virtual SF components are

Corresponding author:
Younghan Kim, School of Electronic Engineering, Soongsil University, 369 Sangdo-Ro, Dongjak-Gu, Seoul 06978, South Korea. Email: younghak@ssu.ac.kr
aggregated together to realize a service or a service function chain (SFC). IoT application components can also be implemented as SFs within the IoT-Edge environment with the structured graphs also forming SFCs. The relative placement of SFs affects the end-to-end latency leading to increased network delays, and in extreme cases may lead to a traffic loop as packets for a network service request traverse the same virtual machines (VMs) repeatedly consuming more bandwidth. While the problem of SF placement in core data centers (DCs) has been widely researched, a gap still exists in the IoT-Edge computing domain. This work addresses the service function placement at the edge (SFPE) problem, whereby less consideration has been made toward the user QoS with most solutions focused on service provider’s deployment perspective.

A key metric for IoT-Edge performance evaluation is latency. While most studies’ approach is optimizing resource allocation strategies or through proactive dynamic resource provisioning, our work takes a user-centric approach with user response time and cost as the main parameters under investigation. The IoT is heterogeneous and highly dynamic with multiple events which need edge systems to be able to dynamically adjust to network parameters (data traffic) and be able to deploy more resources on-demand for a timely response. A user-centric approach allows to anticipate user interests in different services which vary and may have different characteristics. Identifying a usage pattern with the objective of optimizing the QoS ought to inform how allocation decisions influence overall system performance when subsets of SFs are shared by multiple services. Typical characteristics and some representative use cases facilitated by the IoT-Edge include the following: low latency communication for real-time IoT applications (mobile gaming, intelligent transportation); geographical distribution for localized data processing; and effective bandwidth management by offloading to the edge cloud. Some issues to consider include, among others, the host compute capabilities, total latency, inter-host latencies, as well as the SF response times. Other issues include energy consumption and efficiency of IoT-Edge systems. In Agarwal et al., a placement heuristic is evaluated, with latency the main optimization goal. The study in Fei et al. focuses on proactive dynamic resource provisioning for overloaded SFs. While the approach in Fei et al. is similar to ours in terms of splitting the SF requests into categories based on their latency tolerance, the difference is that we employ user request patterns and their similarity to select and schedule a set of candidate requests for priority placement with QoS consideration factored. The studies in Yala et al. and Cziva et al. also optimize access latency for uRLLC traffic through a genetic algorithm and Tabu search metaheuristics, but these studies did not consider chaining constraints and user response time.

The complexity of user requirements creates a challenge in terms of resource management in a heterogeneous infrastructure like the IoT-Edge cloud, where each SF selects the best matching resource from the available resource pool. Most placement strategies assume a static cluster of resources, and this becomes a bottleneck in terms of resource management and service heterogeneity and regardless of how optimal the initial service task arrangement is, it will degrade as the resource utilization and workload change over time. Some services require both shareable and non-shareable which means that multiple demands compete for limited edge resources impacting the workload capacity. Our approach to address this involves splitting the workload into priority classes, which facilitates grouping requests by SF characteristics, resource availability, and user patterns.

Prediction and pre-allocation of resources also depend on users’ behavior and future probability of using the resources. Estimating resources beforehand creates efficiency and fairness in resource management. In Zhou et al., the selection of an SF for each service request is performed with the goal of optimizing the overall QoS criteria of the composite service. QoS correlation between services may have significant impact on the overall QoS values of composite services. These correlations occur when some QoS attributes of a service are not only dependent on the service itself but correlated to some other service. We present an approach to the SFPE problem with the objectives of optimizing user response time, resource workload, and cost for enhanced user QoS. An analysis of the resource workload for service requests is necessary in providing an optimal QoS guarantee for the user satisfaction. This work addresses SF placement in the IoT-Edge cloud environment and our approach models the multiple user requests in terms of user preference and service selection probability to determine placement. We also adopt a network hyper-exponential queuing approach to model the allocation of resources based on demand for specific service types and formulate a heuristic solution for optimized placement of different service requests. By utilizing the user patterns, we develop a framework that allows to prioritize placement and ensure reliable service provisioning. Our approach targets a lower cost of resources and user latency to improve the user QoS. Minimizing the total cost releases additional resources that can potentially be used to allocated additional SFs and improves the acceptance rate of requests for service chains. Specifically, this article makes the following contributions:

1. We define and formulate the SFPE problem and prove that this mixed-integer linear programming
(MILP) optimization problem is an NP-hard problem.

2. We model user preference and service request patterns; we adopt the cosine similarity index value to highlight similarities in user requests and determine an optimal policy for categorizing them.

3. We design a heuristic user preference placement algorithm (HUPPA) and compare our approach with Greedy, Baseline, and Optimal algorithms. The experimental results show that our approach is superior to Greedy on resource cost minimization.

The rest of this study is organized as follows. We formulate and model the SFPE optimization problem in section “Problem formulation.” Section “HUPPA algorithm design” presents our meta-heuristic HUPPA algorithm. Section “Simulation and numerical experiments” gives results evaluation, followed by a discussion. Finally, section “Conclusion” concludes the article.

### Problem formulation

We consider an IoT-Edge computing cloud consisting of a set of \( n \) edge nodes, \( N = \{v_1, v_2, \ldots, v_n\} \), and a set of \( f \) SFs, \( F = \{\phi_1, \phi_2, \ldots, \phi_f\} \). Service requests are generated from multiple users \( U = \{u_1, u_2, \ldots, u_m\} \), with these \( m \) users localized around an edge cloud. In practice, the resource allocation is related to multiple resource types, such as CPU, memory, and storage; however, we simplify the formulation with only the computing resource requirement that is readily extensible in the general case. Each SFC is constituted by a finite number of SFs in sequence, instantiated on hosts, and consequently, we define \( c_v \) as the CPU-bounded resource capacity of computing node \( v \) with \( r_v \) as the SF processing capacity of function \( \phi \). A summary of the sets, parameters, and variables defined and used in the study is presented in Table 1.

#### User request–similarity model

In practice, multiple users have varying degrees of interest for various services. This influx of request demands can overwhelm the provider infrastructure and affect the user QoS, thus the system is best served if it is prepared to handle such workloads. This problem can be likened to the resource matching problem, where the user and service request requirements are estimated a-priori and a match is made based mainly on the user preference data. Multiple-attribute decision-making methods are necessary to comparison and to set up the preference or similar relations among QoS attributes of user requests and available resources. Our model introduces preference modeling and considers it an indispensable step in the decision-making process. The request probability for different services correspondingly varies and can be characterized in terms of user preference. Given a set of SFs \( F = \{\phi_1, \phi_2, \ldots, \phi_f\} \), we introduce a probability distribution of requests for SFs from users denoted as \( h = [h_1, h_2, \ldots, h_f] \), where \( h_i = P(\phi_i), i \in [1, f] \), is the probability that the \( i \)-th SF is requested. Furthermore,
we define user preference as the conditional probability distribution of a user’s request for every SF \( f \) given that the user sends a request.\(^\text{19}\) This is denoted as \( q_f \), for the \( f \)th user, where \( q_{f,i} = P(q_{f}|u_i)∀i ∈ [1,m], i ∈ [1,f] \) is the conditional probability that the \( f \)th user will request the \( i \)th SF. User preference reflects the personal interest of each individual user in a service. To highlight the request distribution and active user level, a probability \( ω_l = P(u_l) \), is introduced that denotes the probability that a request is sent by user \( u_l \) and \( ∑_{i=1}^{m} ω_l = 1 \), giving the overall probability of a request as equation (1)

\[
q_{m,f} = ∑_{i=1}^{f} ∑_{l=1}^{m} ω_l q_{f,i} ∀i ∈ [1,m], i ∈ [1,f]
\]

(1)

To efficiently classify the request, we adopt a similarity computation approach for active users that are active around a given edge. A similarity measure or similarity function is a real-valued function that quantifies the similarity between two objects. We assume that there is no prior history data on user requests made for services hosted on these specific edge nodes made by users. The cosine similarity index method\(^\text{19}\) involves feature vectors which represent requests from users \( U \) for SFs \( F \). In our model, we assume multiple users make multiple requests for a specific service composed of various SFs. Figure 1 shows an example of this scenario whereby multiple user request queues with correlated SFs show in dashed circles. The arcs between SFs show the correlation in similarity. Afterwards, a candidate set is created based on similarity of these functions. A repository is created that is updated when a new service request is added into the repository or when an existing service is updated or removed from the repository. Based on a QoS criteria a new subset is implemented where similar requests are then aggregated, ranked, and finally placed on appropriate nodes.

One motivation for this is to reduce the cost on resources required to instantiate the constituent VMs hosting them by allocating collective resources to requests with similar demands, subsequently reducing the request processing time. The user requests follow a uniform distribution\(^\text{15}\) whereby requests are generated and a realistic representation of our scenario. The mathematical representation of the method is shown in equation (2), where the left hand side is the degree of proximity with a range of \([0,1]\) between users’ \( u \) and \( w \) requests. Afterwards, the requests are aggregated based on QoS requirements that will later form placement policy. To ensure objectivity, we conduct a similarity weighting\(^\text{19}\) to reduce matching time and improve matching efficiency. We use the enhanced Top-K algorithms in Wang et al.\(^\text{20}\) to rank the users based on the Pearson coefficient similarities and select the Top-K most similar users for making QoS value predictions. The Pearson correlation coefficient (PCC) mirrors the level of direct relationship between two factors, that is, the degree to which the factors are connected, and extends from +1 to –1.\(^\text{21}\) A connection of +1 implies that there is a great positive direct relationship between factors or in different words two clients have fundamentally the same tastes, while a negative connection shows that the clients have divergent tastes. Pearson’s connection coefficients are utilized to determine whether the level of the relationship is dynamic. Different from traditional Top-K algorithms, we adopt the approach in Wang et al.\(^\text{20}\) where users with PCC similarities less than or equal to 0 are excluded, that is, removing the negative similarity. This creates a top subset of similar users’ functions, where \( K \) represents the subset of similar user requests. The modified user request top set can be found with the following equation

\[
Φ_{l,w}[\text{sim}(q_{f,i}^l,q_{f,i}^w)] = \frac{2}{m(m-1)} \sum_{i=1}^{m} q_{f,i}^l q_{f,i}^w
\]

(2)

\[
Z_K = \{u_i | i = 1:K, Φ_{l,w}[|] > 0, l \neq w\}
\]

(3)

**Service latency**

Service operators aim to derive suitable placements for SFs on physical nodes or VMs can be launched subject to the demand. One objective of SFPE is to find an appropriate placement for \( f \) SFs on \( n \) computing nodes. We introduce a binary variable \( y_{ij} \) with \( i ∈ [1,f] \) and \( j ∈ [1,n] \), where \( y_{ij} \) is equal to 1 if SF \( \phi_i \) is placed on node \( v_j \), and 0 otherwise. It is also possible to
instantiate multiple request chains on the same node provided it can meet the needed requirements. We introduce another variable $g_j$ iff more than one function has been deployed at the same node

$$g_{v_j[i],v_d} = \begin{cases} 1, & \sum_{j=1}^{f} y_{j}>0 \\ 0, & \sum_{j=1}^{f} y_{j} = 0 \end{cases} \quad \text{(4)}$$

We explore the notion of multiple requests arriving at the edge in the Poisson distribution for packets with a certain frequency which we refer to as service request arrival rate $\lambda_i$. Our work adopts the hyper-exponential network queue $M/H_k/1$ (different to the $K$ subset of top similarities whereby in the hyper-exponential distribution, $k$ represents the SF processing phases) with a $M/G/1$ distribution model.\(^{22,23}\) The service time at node $v_j$ is exponentially distributed with average service rate of $\mu_j$. The total arrival rate $A'_u$ is derived as in equation (5)

$$A'_u = \sum_{i=1}^{f} \lambda_i \cdot q_{req} \cdot g_j \quad \forall j \in [1,n], l \in [1,m] \quad \text{(5)}$$

In our model, we assume that different users make a single request for a specific service at a given time and these similar requests are then aggregated together and instantiated. The motivation for this is to reduce the cost on resources required to instantiate the VMs hosting them by allocating collective resources to requests with similar demands. Before we proceed, we would like to stress the significance of transmission time as a dominant factor in large networks\(^{9,18}\) but in our work we do not consider it. This is because the SFs are deployed at the same physical location.

**Propagation latency.** The propagation delay involves the flow of packets that travel to the edge cloud which in turn experience a certain delay value determined by the physical path and the respective link congestion that occurs between the user source device and destination node where the SF instantiated. We assume the nodes are all located in a cluster and therefore we do not take into consideration the latency between edge nodes in our model, as the bandwidth consumption rate is negligible. Furthermore, we also assume a uniform propagation delay value $d_{i,j}$ for all communication from $i$th user to $j$th node as in Park et al.\(^{24}\) where $pl(\phi_i)$ is the packet length of SF $\varphi$

$$t_{prg} = \sum_{i=1}^{f} x_{\phi,\varphi} \cdot pl(\phi_i) \cdot d_{i,j}, \forall \phi \in F \quad \text{(6)}$$

**Processing latency.** Processing delay generally includes a queuing delay that is determined by the number of user service requests in addition to the traffic. In the first processing phase we consider the edge network an open network and assume the total incoming request arrival rate $A''_u$ which is the sum of the arrival rate from outside and the arrivals from other SFs in the request. The second processing phase involves the processing of requests at the edge nodes. The utilization metric $\rho$ shows the utility factor of the system; if $\rho < 1$, is satisfied then no requests will be dropped. On the contrary, if the utility condition is violated, then the packets will be retransmitted from source to destination as feedback. We determine the steady-state performance of the system in terms of processing delay and the expected total sojourn time as the total time a request spends in the network, that is, the sum of the expected sojourn times at each node $v_j$

$$t_{proc} = \frac{F - 1}{2A''_u} + E[S] + \frac{A''_uE[S^2]}{2(1 - A''_uE[S])} \quad \text{(7)}$$

where \(E[S] = \sum_{i=1}^{f} (d_{i,1}/\mu_1)\) and \(E[S^2] = \sum_{i=1}^{f} (2d_{i,1}/\mu_1)\) are the first and second moments of service time distribution, respectively. Based on the aforementioned discussion, the average service latency is calculated as follows

$$t_d = t_{prg} + t_{proc} \quad \text{(8)}$$

**Workload**

In most existing cluster management systems, for example, Alibaba, two main categories of workloads are defined: batch jobs and long running services.\(^{25}\) Batch jobs are characterized as delay tolerant with a limited lifetime, while the latter requires high availability and low latency, for example, web-based applications. Zhong and Buyya\(^{12}\) investigated four representative workloads: (a) a stable workload pattern represents situations where the workload density does not have any obvious fluctuations; (b) a growing workload pattern may emulate a scenario where an application suddenly becomes popular and attracts a considerable number of visits, holding a high accelerating rate; (c) a cycle/bursting workload could simulate online shopping websites where the network traffic experiences periodical changes and reaches a peak during certain time ranges; and (d) an on-and-off workload pattern depicts applications periodically having a short active stage, such as map-reduce jobs executed on a daily or weekly basis.

Our work focuses on a growing pattern that emulates a scenario where an application suddenly becomes popular and attracts a large visit rate. The expected
number of visits (requests) to a node hosting an SF is a ratio which shows the frequency of instantiation of different functions. Its value supplements the user preference model and is given as \( \eta_i = \lambda_i / \lambda'_i, \forall i \in \{1, f\} \). The total workload of a node \( W_j \) can be consequently calculated as the sum of the resource consumption of all its allocated SFs multiplied by the request rate for each SF\(^{26}\)

\[
W_j = \sum_{i=1}^{f} g^*_i r_i \eta_j \forall j \in \{1..n\} \quad (9)
\]

**Cost**

In our previous work\(^{27}\) we treat cost as a function of the associated VM workload and it was observed that when SF workload pressure is large among VMs, the execution time will be affected. We look at cost from a SF processing capacity perspective and consider inter-host traffic as a higher rate of inter-host traffic might lead to congestion given constraints on the bandwidth demand. Inspired by Raumer et al.\(^{28}\) and Ramaswamy et al.\(^{29}\) we formulate the total costs \( \pi_{\text{total}} \), which are constituted by different components: the deployment cost for an SF on a node is denoted by \( \pi_n \), the SF processing cost \( \alpha_n^f \) and SF link cost \( \beta_{n,j} \). These model the heterogeneity of request flows in networks and consider varying sizes of network services. In Raumer et al.\(^{28}\) it was observed that for bigger packets, the size did not influence the cost per packet for a consolidated scenario and following this logic, we do not consider packet size as we define cost. The service rate for different SFs is determined by the multiple operations on packets processed, for example, compression and/or decompression, therefore cost is derived as below

\[
\pi_{\text{total}} = \sum_{n \in N} \sum_{f \in F} W_j (\alpha_n^f + \pi_n) + \sum_{i=1}^{f} x_{\phi,p} \beta_{1-1,i} \quad (10)
\]

**Optimization analysis**

System delay is a critical metric to ensure QoS is maintained and in our work, the objective is to minimize the user response time and cost while optimizing the workload. The SFPE problem can thus be formulated into two parts: to realize the joint objectives of minimizing the response time, we solve the sub-modular optimization problems P1 and P2

\[
P1: \min t_{sl} \quad (11)
\]

\[
s.t. \sum_{i=1}^{f} y_{ij} r_i \leq c_j, \forall j \in \{1..n\} \quad (12)
\]

\[
\sum_{i \in F} \sum_{j \in N} y_{ij} = 1 \quad (13)
\]

\[
\sum_{i \neq a} \sum_{j \in N} y_{ij} + y_{aj} \leq 1 \quad (14)
\]

\[
P2: \min \pi_{\text{total}} \quad (15)
\]

\[
s.t. \forall j \in \{1..n\}, \forall \phi \in F : \mu_j = \sum_{i=1}^{f} \lambda_{\phi,x_{\phi,p}} \quad (16)
\]

\[
x_{\phi,p} \in \{0, 1\}, \forall \phi \in F \quad (17)
\]

Constraint in equation (12) ensures that the SF processing capacity does not exceed the available capacity. Constraint in equation (13) ensures the SF instance has to be instantiated for any active request. Anti-affinity (cases whereby the other correlated SFs can be instantiated separately but still preserve their functionality in a chain) is preserved in equation (14) for SFs that are dissimilar in characteristics and QoS requirements but form part of the same chain. Equation (16) ensures that no requests will be dropped. Congestion control is guaranteed in equation (17).

**HUPPA algorithm design**

To conduct multi-objective optimization, often huge computational effort is required.\(^{30}\) In most of the cases, multi-objective optimization problem is designed to meet a particular scenario reducing their adaptability and makes them less scalable. Besides, solutions to a multi-objective optimization problem have a large time complexity, therefore placing such an approach is at a disadvantage in a real-time IoT-Edge environment characterized by resource constrained nodes. It can be easily seen that P1 and P2 are NP-hard and an analytical solution will be computationally expensive. Therefore, we design a priority-driven algorithm HUPPA that places SF requests with lowest latency priority and to find a near-optimal cost-effective placement solution for the user. An SF repository is created and is updated when a new service request is added into the repository or when an existing service is updated or removed from the repository (Figure 2). We use a set \textit{Activlist} to save the active compute nodes, that is, nodes in service and a set \textit{Vaclist} to save the spare compute nodes with residual capacity. A set \textit{Servlist} holds the initial set of SFs that are to be instantiated. Another set \textit{Holdlist} holds the remaining SFs that are yet to be placed after instantiation. HUPPA first checks the availability of resources (lines 2 and 3) after which the \textit{Servlist} which contains the already categorized requests as results from Algorithm 1. This is facilitated by the visit rate. Lines 6 and 7 then select the first request and sorts its component SFs based on
decreasing latency values. The results are then placed into the Holdlist SF where similar instances are aggregated (line 8). The active compute nodes are then sorted from those with least to highest processing capacity. The reason for this is that we aim to place those SFs with least resource-demand to proportional nodes to prevent oversubscription of resources, as the scenario we are investigating is for a growing pattern that emulates a scenario where an application suddenly becomes popular and attracts a large visit rate, holding a high accelerating rate workload. To ensure system stability and maintain a high admission rate (line 11) affirms constraint in equation (17). At this point, the algorithm checks that the end-to-end latency requirement for multiple SF requests is not compromised (line 12) and the node capacity of active nodes can host another instance (line 13). For placement of multiple SFs, the algorithm chooses the node with maximum processing capacity to facilitate re-usability of the node for future placements (line 15). If the minimum QoS requirements cannot be guaranteed, then no placement can occur, and the lack of a feasible solution forces the system to terminate and loop back to line 2. This process continues in a loop fashion until all SFs are successfully placed (line 16), otherwise the placement is complete, and a new service is selected, until all SFs have been successfully placed.

Simulations and numerical experiments

In this section, we conduct simulations to evaluate our approach. We choose response time and executive cost to examine the efficiency of HUPPA algorithm. We investigate six commonly requested different SFs: NAT, FW, TM, WOC, VOC, and IDS. Each SF represents a different profile in terms of requests arrival distribution, delay tolerance and processing capacity.

We model a scenario similar to Cziva et al.,10 Mouradian et al.,16 and Mutichiro et al.27 with an edge cloud of 3–10 nodes. The number of requests range from 30–1000, with all instances featuring SFCs of varying lengths from 3 to 6 SFs/IoT applications generated uniformly. The packet size of the data transmitted in the chains is selected randomly in the range of 64–1518 bytes. The number of users is set at between 10 and 100. We assume the arrival time of requests follows a Poisson distribution with an arrival rate ranging from 500 to 800 requests/second. The service rate for SF instances varies between 600 and 1200 units. Optimal solutions for selected instances were obtained using general linear integer programming implemented in CPLEX solver. The performance of the proposed
We investigate the parameters of delay, workload and cost and observe how they meet the user QoS demand requirements. We briefly describe the sequence of the Greedy algorithm for comparison:

**Greedy approach**

For each service request, this algorithm will sort a list of candidate nodes and first place an SF into one of the best nodes in terms of available resources. If the first one is fails due to constraint violation, the next one will be selected. This loop is continued until placement of all the SFs into the physical nodes occurs.

**Baseline approach**

The algorithm performs similar to the approach used in hybrid clouds. It considers first SFs that have a high priority level and then selects a node to place an instance of the SF with adequate processing resources. The reason we selected this approach is to compare our own approach with an algorithm that seeks to benefit as much as possible from the IoT-Edge infrastructure.

In Figure 3, we compare our heuristic with the Optimal and Baseline approach in a typical scenario over a period of 1 hour. Optimal adopts the MILP model and as expected provides the lowest response time. It can then be observed that for the Greedy, the delay increases exponentially from 600 requests/hour. Greedy algorithm does not categorize the requests according to QoS requirements and tries to allocate them to instances-consolidate them on the same node thereby increasing the response time for $\lambda_i > 600$. HUPPA consistently yields substantially lower delay values and is closer to the optimum this is because it prioritizes similar requests with lower processing times that do not dominate the total latency. Baseline algorithm works on a best-fit approach where it classifies the requests into priority levels thereafter selecting the best available node for placement. This has the advantage of lowering the response times for incoming requests especially with strict latency demands. With the increase in user requests, the performance of Baseline deteriorates but still outperforms the Greedy approach. We can also observe this from the average delay values in Figure 4, that for the first set of instances, Optimal approach tends to give better assignments while HUPPA algorithm tends to find better placement. Another evaluation point is the effectiveness of our approach in terms of delay incurred in terms of user numbers and the generated traffic over multiple iterations. It can be inferred that as the arrival rate increased, the quality of obtained solutions was significantly worse, with Greedy performing worse this is after selecting the nodes with the highest availability, but subsequently increasing the processing time overhead.

We also observed that the huge impact on the performance in average delay is significantly lower than for Greedy and the solution values are comparable for the other three algorithms with the exception of Baseline. In detail, when $\lambda_i \leq 550$, there is no significant difference in delay values, this is due to the number of
considered resource objects and that processing queues can handle the incoming requests. Baseline over multiple iterations consumes a lot of resources as it is defined by priority constraints leading to an increase in the queuing delay and overall response time. It is also observed that up to 575 requests/second, there is no significant statistical difference in the delay values, but the values for Greedy deviate greatly from the standard mean, while HUPPA deviates slightly as the traffic increases over time.

Figure 5 shows the processing delay values for four algorithms, in which Greedy has a higher delay proportional to number of users. However, HUPPA exhibits a lower response time as the user numbers increase. This is due to the steady-state policy nature of the $M/H_k/1$ system whereby the idle time is minimized, and the busy cycle of the compute node is maximized with load balancing considerations, thus the response time is kept low as the user numbers increase.

Workload–delay tradeoff

We evaluate workload for a growing workload scenario as shown in Figure 6. For Greedy, most requests remain in pending status with poor QoS performance as it does not take into consideration how overloading the compute nodes can exponentially increase the overall queuing and processing delays. This leads to higher response time, as resource consumption from all active instances keeps accumulating simultaneously and there is under-utilization of resources. However, HUPPA supports an on-demand workload scaling strategy which together with the selection strategy allows it to avoid over-utilization of resources ensuring load balancing to optimize the workload. This can be explained with the increase in generated traffic, whereby the resources are stretched, and some SFs may need to be duplicated to meet demand as more SF instances are provisioned. Therefore, being able to instantiate multiple similar requests for the same SF instances allows HUPPA to better utilize resources increasing the CPU utilization efficiency and lowering the response time. Another observation is Baseline utilizes a higher number of resources as it instantiates a higher number of SFs and seeks to place them on only the best nodes. This limits its efficiency as the incoming traffic’s granular characteristics are not taken into account during placement resulting in increased delays. We observe from Figure 7 that with variance in workload processed by the nodes against incoming user requests. HUPPA is able to balance the workload relative to the processing capacities of the different nodes where the SFs are instantiated along the objectives reducing response time and cost. Baseline and Greedy only consider resource usage as an evaluation measure, as both algorithms try to exhaust the resources of each node to minimize resource idleness. While this is good for cost minimization, it increases delay overhead leading to increased response times.

Cost–delay tradeoff

To analyze the cost for transmitting, receiving, and processing packets, we set $\alpha_p = 0.12$/request and
Deploying a maximal number of SFs on each node keeps costs low but leads to a high utilization at some nodes as is shown in Figure 8, with $\lambda_i = 550$ for all four algorithms. For $\lambda_i < 600$, HUPPA maintains a steady increase in cost compared to Greedy and almost approaches Optimal. The Greedy approach results in bigger user response time as the request rate increases and the SFs are allocated resources on a first-come first-serve basis leading to higher processing and queuing delays, which result in increased costs. In Table 3, a comparison of the key evaluation attributes shows that our approach achieves lower values and records an improvement both in response time and the cost incurred. This is attributed to the selection of specific latency sensitive services and the allocation of compute resources with higher processing capacity.

### Table 2. Experimental values.

| Parameter | No. of requests | No. of users | No. of SFs | $\lambda$ (req/s) | $\mu$ (req/s) | $\alpha_i$ ($/req$) | $\beta_{i,j}$ ($/req$) | $\pi_f$ |
|-----------|----------------|--------------|------------|-------------------|---------------|---------------------|----------------------|--------|
| Value     | 30–1000        | 10–100       | 6          | 500–1000          | 600–1200      | 0.12                | 0.48                 | 0.014  |

### Table 3. Performance summary.

| QoS attribute | With similarity | Without similarity | Improvement |
|---------------|-----------------|--------------------|-------------|
| Response time | 0.027 s         | 0.146 s            | ~18%        |
| Cost          | $237.386$       | $407.046$          | ~58%        |

$\beta_{i,j} = 0.48$/request (Table 2). Deploying a maximal number of SFs on each node keeps costs low but leads to a high utilization at some nodes as is shown in Figure 8, with $\lambda_i = 550$ for all four algorithms. For $\lambda_i < 600$, HUPPA maintains a steady increase in cost compared to Greedy and almost approaches Optimal. The Greedy approach results in bigger user response time as the request rate increases and the SFs are allocated resources on a first-come first-serve basis leading to higher processing and queuing delays, which result in increased costs. In Table 3, a comparison of the key evaluation attributes shows that our approach achieves lower values and records an improvement both in response time and the cost incurred. This is attributed to the selection of specific latency sensitive services and the allocation of compute resources with higher processing capacity.

### State-of-the-art

The estimation of resources beforehand creates efficiency and fairness in resource management in the edge environment. Considering the rapid growth of 5G and enablers such as IoT and cloud computing technology, service providers have realized that service QoS and user satisfaction are fast becoming benchmarks in ensuring quality service provision. The current state-of-the-art has adopted users’ behavior as a tool to predict and pre-allocate resources as well as determine future probability of using the resources.\(^{14}\) Wang et al.\(^{20}\) have addressed the challenge of QoS prediction for service recommendation in edge computing by modeling user mobility and data volatility to adapt to mobile edge computing environments. They calculate user or edge server similarity depending on users’ changing locations. However, they also identify shortcomings in their approach specifically, a decreased prediction accuracy when the edge server distribution is sparse. Li et al.\(^{17}\) also proposed a method of resource prediction based on QoS values. Their approach uses QoS attribute values like response time, availability, and price that are requested by users to calculate the similarity between the user requirements and edge resources. They adopt similarity matching algorithm and regression-Markov chain prediction method. However, their work does not go beyond just matching resources as other areas like resource scheduling and allocation are not considered. Our work builds upon the state-of-the-art by combining similarity matching with the resource allocation aspect. We propose and implement a model that adopts service request patterns to realize a user preference and to highlight similarities in user requests. Using different QoS attributes, we are able to further filter and rank similar requests after which we implement an optimal policy.
for allocating resources to instances that are matched to those top requests. The benefit here is three-fold: our heuristic approach achieves a lower response time and due to its scalable nature, it ensures a better cost-delay tradeoff. It also prevents resource wasting and overprovisioning while minimizing QoS degradation. Also, another area we address is the characteristic of the incoming traffic, whereby we model for a growing workload with increasing intensity overtime. This is important as it imitates a typical edge environment that may need to scale periodically on-demand but still maintain a high resource utilization numbers while meeting the QoS requirements.

**Conclusion**

In this work, we investigated the SF placement in edge computing problem in a scenario with rapidly increasing workload within a short period in a resource-constrained environment. The method consists of a comparison of batch user requests in a similarity-based manner and categorizing them according to QoS requirements. The placement then aims to optimize user response time and cost. For this, we proposed a framework that utilizes the user patterns to derive similarity in user requests and subjected to policy engine rules, it then determines and prioritizes which requests to instantiate at the IoT-Edge cloud and how many replicas are needed to meet the user service requirements. The key parameters we considered were latency and cost, whereby consideration was given to the dynamic and bursty nature of IoT data traffic as well as consideration for the heterogeneous edge environment. We propose two algorithms: the first is similarity matching heuristic that creates a set of similar requests and ranks them; the second is heuristic that ensures placement occurs and user QoS requirements are not violated, while achieving the objective goal. The results of our experimental evaluation have further confirmed that a use case based on our similarity framework can not only mitigate delay without data loss but also improve the user QoS for the IoT-Edge cloud. Results show that our approach offers better performance with near-optimal results compared to current state-of-the-art. Future directions include enhancing our framework to consider different forms of dynamic workloads with a focus on user mobility.

**Declaration of conflicting interests**

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

**Funding**

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: This work was supported by the Institute for Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT, No. 2020-0-00946, Development of Fast and Automatic Service Recovery and Transition Software in Hybrid Cloud Environment) and the ITRC (Information Technology Research Center) support program (IITP-2021-2017-0-01633) supervised by the IITP.

**ORCID iD**

Briytone Mutichiro https://orcid.org/0000-0003-2796-5083

**References**

1. Porambage P, Okwuibe J, Liyanage M, et al. Survey on multi-access edge computing for internet of things realization. *IEEE Commun Surv Tut* 2018; 20(4): 2961–2991.
2. Brogi A, Forti S, Guerrero C, et al. How to place your apps in the fog: state of the art and open challenges. *Softw Pract Exp* 2020; 50(5): 719–740.
3. Laghrissi A and Taleb T. A survey on the placement of virtual resources and virtual network functions. *IEEE Commun Surv Tut* 2018; 21(2): 1409–1434.
4. Agarwal S, Malandrino F, Chiasserini CF, et al. Joint VNF placement and CPU allocation in 5G. In: *Proceedings of the IEEE INFOCOM 2018 —IEEE conference on computer communications*, Honolulu, HI, 16–19 April 2018, pp.1943–1951. New York: IEEE.
5. Xi Fei, Liu F, Xu H, et al. Adaptive VNF scaling and flow routing with proactive demand prediction. In: *Proceedings of the IEEE INFOCOM 2018 —IEEE conference on computer communications*, Honolulu, HI, 16–19 April 2018, pp.486–494. New York: IEEE.
6. Premsankar G, Di Francesco M and Taleb T. Edge computing for the internet of things: a case study. *IEEE Intern Things J* 2018; 5(2): 1275–1284.
7. Sarkar S, Chatterjee S and Misra S. Assessment of the suitability of fog computing in the context of internet of things. *IEEE Trans Cloud Comput* 2015; 6(1): 46–59.
8. Agbehadji IE, Frimpong SO, Millham RC, et al. Intelligent energy optimization for advanced IoT analytics edge computing on wireless sensor networks. *Int J Distrib Sens Netw* 2020; 16(7): 0908772.
9. Yala L, Frangoudis PA and Ksentini A. Latency and availability driven VNF placement in a MEC-NFV environment. In: *Proceedings of the 2018 IEEE global communications conference (GLOBECOM)*, Abu Dhabi, United Arab Emirates, 9–13 December 2018, pp.1–7. New York: IEEE.
10. Cziva R, Anagnostopoulos C and Pezaros DP. Dynamic, latency-optimal VNF placement at the network edge. In: *Proceedings of the IEEE INFOCOM 2018 —IEEE conference on computer communications*, Honolulu, HI, 16–19 April 2018, pp.693–701. New York: IEEE.
11. Guo S, Hu X, Dong G, et al. Mobile edge computing resource allocation: a joint Stackelberg game and matching strategy. *Int J Distrib Sens Netw* 2019; 15(7): 986–1556.

12. Zhong Z and Buyya R. A cost-efficient container orchestration strategy in Kubernetes-based cloud computing infrastructures with heterogeneous resources. *ACM Trans Intern Technol* 2020; 20(2): 1–24.

13. He T, Khamfroush H, Wang S, et al. It’s hard to share: joint service placement and request scheduling in edge clouds with sharable and non-sharable resources. In: *Proceedings of the 2018 IEEE 38th international conference on distributed computing systems (ICDCS)*, Vienna, 2–6 July 2018, pp.365–375. New York: IEEE.

14. Aazam M and Huh EN. Fog computing micro datacenter based dynamic resource estimation and pricing model for IoT. In: *Proceedings of the 2015 IEEE 29th international conference on advanced information networking and applications*, Gwangju, South Korea, 24–27 March 2015, pp.687–694. New York: IEEE.

15. Zhou A, Wang S, Wan S, et al. LMM: latency-aware micro-service mashup in mobile edge computing environment. *Neur Comput Appl* 2020; 32: 15411–15425.

16. Mouradian C, Kianpisheh S, Abu-Lebdeh M, et al. Application component placement in NFV-based hybrid cloud/fog systems with mobile fog nodes. *IEEE J Select Area Commun* 2019; 37(5): 1130–1143.

17. Li G, Song J, Wu J, et al. Method of resource estimation based on QoS in edge computing. *Wirel Commun Mob Comput* 2018; 2018: 7308913.

18. Jiang Y, Ma M, Bennis M, et al. User preference learning-based edge caching for fog radio access network. *IEEE Trans Commun* 2018; 67(2): 1268–1283.

19. Chen B and Yang C. Caching policy for cache-enabled D2D communications by learning user preference. *IEEE Trans Commun* 2018; 66(12): 6586–6601.

20. Wang S, Zhao Y, Huang L, et al. QoS prediction for service recommendations in mobile edge computing. *J Parall Distrib Comput* 2019; 127: 134–144.

21. Dharaneshwaram N, Nithya S, Srinivasan A, et al. Calculating the user-item similarity using Pearson’s and cosine correlation. In: *Proceedings of the 2017 international conference on trends in electronics and informatics (ICEI)*, Tirunelveli, India, 2017, pp.1000–1004. https://www.researchgate.net/publication/317349295_Calculating_the_User-item_Similarity_using_Pearson%27s_and_Cosine_Correlation_Senthilkumar_M

22. Wang KH and Yen KL. Optimal control of an M/Hk/1 queuing system with a removable server. *Math Meth Oper Res* 2003; 57(2): 255–262.

23. Bolch G, Greiner S, De Meer H, et al. *Queuing networks and Markov chains: modeling and performance evaluation with computer science applications*. Hoboken, NJ: John Wiley & Sons, 2006.

24. Park Y, Yang H, Dinh T, et al. Design and implementation of a container-based virtual client architecture for interactive digital signage systems. *Int J Distrib Sens Netw* 2017; 13(7): 7717864.

25. Rodriguez MA and Buyya R. Containers orchestration with cost-efficient autoscaling in cloud computing environments, 2018, https://arxiv.org/abs/1812.00300

26. Guerrero C, Lera I and Juiz C. A lightweight decentralized service placement policy for performance optimization in fog computing. *J Amb Intell Hum Comput* 2019; 10: 2435–2452.

27. Mutchiro B, Yang H and Kim Y. Usage aware VNF placement for improved QoS in edge computing. In: *Proceedings of the 2019 international conference on information and communication technology convergence (ICTC)*, Jeju Island, South Korea, 16–18 October 2019, pp.808–812. New York: IEEE.

28. Raumer D, Bauer S, Emmerich P, et al. Performance implications for intra-node placement of network function chains. In: *Proceedings of the 2017 IEEE 6th international conference on cloud networking (CloudNet)*, pp.1–6. New York: IEEE, https://www.net.in.tum.de/fileadmin/bibtex/publications/papers/Raumer_Cloudnet2017.pdf

29. Ramaswamy R, Weng N and Wolf T. Characterizing network processing delay. In: *Proceedings of the 2004 IEEE global telecommunications conference (GLOBECOM ’04)*, Dallas, TX, 2004, vol. 3, pp.1629–1634, http://www.ecs.umass.edu/~kdeb/papers/globecom2004.pdf

30. Helbig M, Deb K and Engelbrecht A. Key challenges and future directions of dynamic multi-objective optimisation. In: *Proceedings of the IEEE congress on evolutionary computation (CEC ’16)*, 2016, pp.1256–1261. New York: IEEE, https://www.egr.msu.edu/~kdeb/papers/c2016016.pdf

31. Savi M, Tornatore M and Verticale G. Impact of processing-resource sharing on the placement of chained virtual network functions. *IEEE Trans Cloud Comput*. Epub ahead of print 2 May 2019. DOI: 10.1109/TCC.2019.2914387.

32. Ben Jemaa F, Pujolle G and Pariente M. QoS-aware VNF placement optimization in edge-central carrier cloud architecture. In: *Proceedings of the 2016 IEEE global communications conference (GLOBECOM)*, Washington, DC, 4–8 December 2016, pp.1–7. New York: IEEE.