A Novel Framework for Mobile Edge Computing By Optimizing Task Offloading

Naouri Abdenacer, Hangxing Wu, Nouri Nabil Abdelkader, Sahraoui Dhelim, Memeber, IEEE, and Huansheng Ning, Senior Member, IEEE

***Abstract*—With the emergence of mobile computing offloading paradigms such as Mobile Edge Computing (MEC), many IoT applications can take advantage of the computing powers of end devices to perform local tasks without the need to rely on a centralized server. Computation offloading is becoming a promising technique that helps to prolong the device's battery life and reduces the computing tasks' execution time. Many previous works have discussed task offloading to the cloud. However, these schemes do not differentiate between types of application tasks. It is not reasonable to offload all application tasks into the cloud. Some application tasks with low computing and high communication cost are more suitable to be executed on the end devices. On the other hand, most resources on the end devices are idle and can be used to process tasks with low computing and high communication cost. In this paper, a three-layer task offloading framework named DCC is proposed, which consists of the device layer, cloudlet layer and cloud layer. In DCC, the tasks with high computing requirement are offloaded to the cloudlet layer and cloud layer. Whereas tasks with low computing and high communication cost are executed on the device layer, hence DCC avoids transmitting large amount of data to the cloud, and can effectively reduce the processing delay. We have introduced a greedy task graph partition offloading algorithm, where the tasks scheduling process is assisted according to the device computing capabilities following a greedy optimization approach to minimize the tasks communication cost. To show the effectiveness of the proposed framework, We have implemented a facial recognition system as use case scenario. Furthermore, experiment and simulation results show that DCC can achieve high performance when compared to state-of-the-art computational offloading techniques.**

***Index Terms*—Computation offloading, Cloud computing, Cloudlet computing, Dynamic mobile cloudlet, Cluster formation, Communication tasks.**

I. Introduction

WITH the tremendous growth of the Internet of Things (IoT), the number of objects connected to the IoT network is in the scale of billions. Most of mega cities around the world such as Beijing and New York have recently been equipped with thousands of smart objects, including cameras, sensors and actuators. These objects sense the environment and react to a real-time situation by gathering data from different sources, which requires sending a huge amount of data continuously. Analyzing such tremendous generated data, such as the video streams from smart cameras, or augmented reality, and facial recognition data in IoT applications requires a high computing resource that can be empowered by the Cloud Computing [1]. Cloud Computing can be described as a remote data center consisting of a collection of super computing nodes that share resources with each other forming intensive resource computing associated with smart management software such as a software-defined network (SDN). The devices that are unable to complete computing tasks locally, offload their computing task to the cloud. However, due to the large gap between the cloud and end devices, the network could suffers from connectivity delay which is not suitable for latency sensitive real-time applications, in addition to the back traffic weight that could overload the network. In order to reduce the network delays and the massive resulting traffic through the network, Edge Computing (EC) [2] was suggested as a solution to solve those issues where it enables computing at the edge of the network. However, shifting the computation from the cloud to the edge requires intelligent supervision. Furthermore, researchers suggested Mobile Edge Computing (MEC) in [3], which is considered to be a variant form of EC adapted to mobile networks.

Some MEC architectures and offloading strategies have been investigated in [4–8] for remote task computing, which deals separately with different minimizing or maximizing objectives, such as minimizing the energy consumption or the execution delay or maximizing the offloaded tasks ratio or the system profit within computing devices or the fog nodes [7, 8]. In [9] authors target the execution time within IoT devices, and introduced a fully polynomial-time approximation scheme to reduce the application tasks execution delay. Also, authors in [10, 11] have proposed femtocloud system which provides a dynamic, self-configuring, and multi-device mobile cloud out of a cluster of mobile devices. Moreover, authors in [11] identify an optimal scheduling decision for a mobile application comprising of dependent tasks, such that the communication and execution cost is minimized subject to an application deadline. However, most of these works did not consider the application task dependency and the communication burden, the IoT devices at the edge do not differentiate applications tasks and offloaded the entire applications tasks to the cloud using traditional offloading strategies, which cause a large volume of data to be transmitted, and subsequently network congestion.

In fact, offloading tasks to the cloud can be reasonable when the edge or fog computing nodes cannot fulfill the application needs. The high computing requests demands may

Naouri Abdenacer, Hangxing Wu, Sahraoui Dhelim and Huansheng Ning are with the University of Science and Technology Beijing, Beijing 100083, China

Naouri Abdenacer, Hangxing Wu, Sahraoui Dhelim and Huansheng Ning are also with Beijing Engineering Research Center for Cyberspace Data Analysis and Applications, Beijing, China

Nouri Nabil Abdelkader is with the University of Djelfa, Algeria

Corresponding author: Huansheng Ning (ninghuansheng@ustb.edu.cn).

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2021.3064225, IEEE Internet of Things Journal

exceed the capability of the nodes because of their insufficient resources. While offloading tasks with low computing and high communication cost to the cloud is not reasonable, due to the unstable devices connection with the cloud and the long distance between devices and cloud. It is better to process these tasks locally within the edge and nearby devices to reduce the communication traffic and execution time. Furthermore, there are still some problems that have not been addressed in the previous works. Firstly, the resources on the edge may not fulfill all user's requests due to the enormous offloading tasks. Secondly, the mobility of mobile devices still stands as a barrier to achieve efficient offloading. Thirdly, choosing the suitable task execution location remains a challenge. In order to solve all these problems mentioned above, we proposed a three-tier MEC architecture as shown in Fig. 1, which consists of device layer, cloudlet layer, and cloud layer (DCC).

Based on the observation that a large number of computing resources in the local devices are idle in practice, a large number of high communication tasks can be handled locally if these idle computing resources can be utilized. As a result, the processing delay of these tasks can be reduced, and we avoid sending large amounts of data to the cloud. Therefore, a device layer in DCC is defined by forming dynamic mobile cloudlet devices in the same area. The advantages of DCC are as follows. Firstly, high communication tasks in DCC can be handled locally within the nearby computing nodes, and high computing tasks can be sent to the cloud. Secondly, a mobile device jointly form a cloudlet, which will make its link to the cloud more reliable. Thirdly, some tasks will be processed locally which will ease the pressure on the cloud. Finally, a suitable task execution location can be chosen by adopting an optimal scheme in DCC.

Our contributions can be summarized as follows:

1) Proposed a novel a three-layer task offloading framework named DCC, which consists of the device layer, cloudlet layer and cloud layer.
2) Proposed a greedy task graph partition offloading algorithm, where the tasks scheduling process is assisted according to the device computing capabilities following a greedy optimization approach to minimize the tasks communication cost.
3) Implemented a facial recognition system based on the proposed framework as use case scenario.

The remainder of this paper is organized as follows. Section II discusses previous related works. Section III describes the application model used for computing and offloading tasks with the proposed architecture and outlines the proposed resource allocation and task partition algorithms. Section IV analyses the partition algorithm's performance and the experiment results. Finally, we conclude the paper in Section V.

II. Related Work

Recently, offloading computing shown a wide application in several domains in IoT, with different architectures and different policies, where application tasks executed remotely on other devices due to insufficient device resources, regarding the application execution time and devices energy. Different works have been explored this area and different

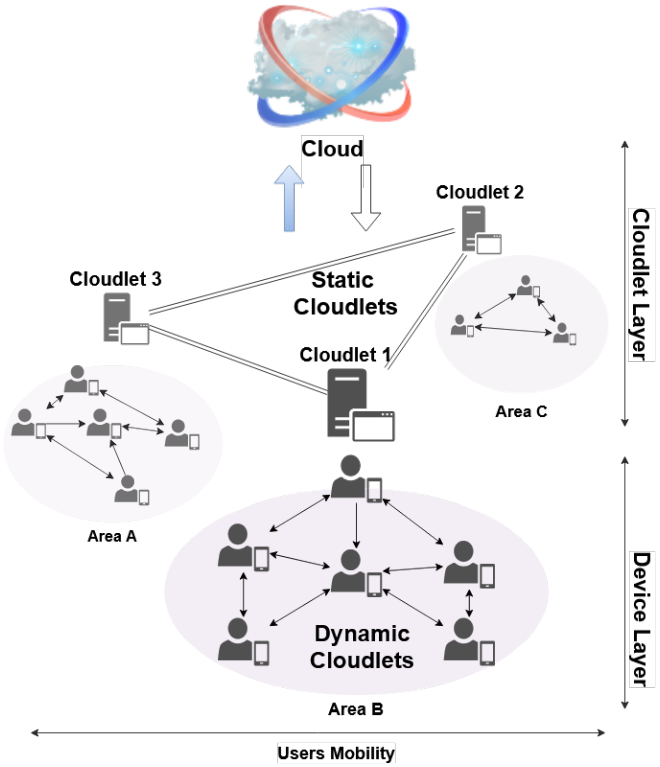


Fig. 1: DCC Computing Architecture

static and dynamic offloading frameworks and architectures were proposed. In [12, 13] authors introduced CloneCloud and MAUI frameworks that aim to improve battery life and device performance by offloading application components to cloud servers. Both targeting one server for offloading. In CloneCloud, tasks executed in a cloned image of the system of the device, it combines a static program analysis with a profiling program to select the offloaded components. In MAUI tasks executed based on methods annotation and static program analysis. However, a single server may not have enough communication and computing resources. In these cases, and in other situations, where there are significant latency limitations, frameworks with concurrent offloading on multiple servers have been suggested for task distribution among a cluster of servers with specific processing and communication capabilities. In [14] a framework named ThinkAir has been proposed to fix the drawbacks of two previous mentioned frameworks. In particular, ThinkAir use being used with new ways for resource management and simultaneous task execution. It focuses on the cloud's elasticity and scalability and improves the capacity of mobile cloud computing by using several virtual machine (VM) images for parallel process execution.

Concerning the offloading process, authors in [15–17] proposed several strategies. Most concentrate on independent task scheduling and concentrate on single metrics, such as energy or execution application time. In [16] the authors studied the task scheduling problem to reduce energy consumption, by assuming that the cloud and the mobile devices can not run simultaneously. While exploiting parallelism between the cloud and the mobile devices can quickly improve the

application execution time. For instance, a heuristics [18] and genetic algorithms [19] introduced to reduce the application tasks execution time respecting task execution order. Authors in [11] target the device's energy where propose a task allocation strategy for heterogeneous devices and mobile cloud. Furthermore, offloading computing of dependent tasks have been discussed in [9, 11]. In [9], the authors target the reduction of execution time by introducing a fully polynomial-time approximation scheme to reduce the overall latency while offloading dependent tasks to multiple devices. In [20], authors suggested deterministic and probabilistic delay constrained task partitioning algorithms to optimize the total of remote computational cost and mobile energy usage of all application tasks with delay constraints in polynomial time, where tasks are structured in form of a tree for sake of simplicity. However, both [21] and [20] assumed that the devices have infinite capacity. In contrast, our proposed task dependency offloading model regards the device constraint resource where it can serve more than one user and perform more than one task according to the device's ability.

III. System Architecture Model and Problem Formulation

Cloud computing offers easy accessibility and cooperation, but cloud centralization could expose it to a bottleneck problem due to the massive network traffic generated by the user's offloading requests. Also, adding a delay to the network, due to the large gap between the users and the cloud resources. For example, a self-driving car, it is critical to have the shortest possible time from collecting data through sensors to making a decision and then acting on it. Hence enabling computing at the network edge by placing computing resources within the edge network can improve the time response system. Although the advantages brought by the edge, the resource limitation remains challenging. For that, we introduced a DCC architecture, where consists of different layers collaborate with each other to mitigate the network burden as much as possible and increase the system performance.

Fig. 1 presents our proposed architecture where it consists of 3 tiers: a device layer, cloudlet layer, and cloud layer(DCC). The device layer consists of a set of heterogeneous devices, the cloudlet layer consists of a set of computing resources provided close to the user in the form of servers, and the last tier represents the cloud servers. Usually, edge devices play a limited role in sending data and information to and receiving processed information from the cloud. In our case, we explore the idle edge device resource for executing tasks to reduce the traffic of the network backbone and the response time. This requires a smart offloading task policy to explore the edge devices resource efficiently. Some offloading computing policies have been proposed regarding the energy and device computation cost. However, these policies assign the offloading tasks directly to the fog or to the cloud without considering the communication cost. This motivates us to investigate on offloading policy to reduce the communication and computation cost in an efficient manner within our proposed DCC architecture.

Allowing communication between devices can alleviate the communication burden at the edge network, where tasks are executed locally within devices instead of sent to cloudlet or cloud servers. Let N is the number of computing nodes and M is the number of computing tasks to be assigned. Our objective is to provide an optimal collaboration strategy regarding the communication cost, focusing on delegating the M computational correlated tasks to the appropriate computing nodes. In addition, the device location on the network can affect the task execution process due to their mobility, hence a management process required to avoid task execution failure.

A. An Overview of DCC

The DCC computing system provides a computing resource at different levels, where users can use it. When a user initiates its offloading process, the application tasks distinguished into different types, where each task will offload to the corresponding computing node according to its feature and the node ability. Application remote tasks can be classified into two main types, tasks with high computing and low communication (computation tasks) which are suitable to be offloaded to the cloud, and tasks with low computing and high communication (communication tasks) that are preferable to be executed at the nearby computing devices whether in device layer or edge layer. Fig. 2 illustrates different cases for the offloading process within the DCC system. As we can see in Case 1: User A initiates the offloading process where is starting by offloading communication tasks within its nearby computing nodes in the corresponding cluster. The communication tasks that cannot be handled inter-cluster, it is offloaded to neighboring clusters through the edge node as shown in Case 2. In Case 3: Computation tasks offloaded toward the edge nodes due to their high computation resource needs. Case 4: reflect on the tasks that cannot be performed in the current edge node due to the edge node resource limitation which is preferable to offloaded towards the neighboring edge nodes.

B. Dynamic Cloudlet Formation

The creation of mobile cloudlets in form of clusters helps to reduce energy consumption during the task offloading process, especially when considering scalability and robustness [22]. As a result, when the device's arrangement is suitably adjusted, the network life duration period is prolonged (without the need to replace the nodes' batteries). Centralized and distributed cluster formation approaches have been proposed in the literature [23]. Each one has advantages and disadvantages, where relying on the centralized approach provides an optimal bound over the distributed schemes, while the distributed approach shown is resistant to network topology changes but consumes significant edge devices energy which is critical. In contrast, the centralized approach is not scalable but it consumes low energy. However, we relied on the centralized approach where two important issues related to the clustering process for task offloading should be addressed. The first one: which are the suitable cluster nodes for current task execution according to task features (i.e. Soft and Hard deadline)?. The second one: where should tasks be executed for optimal execution "INTER/INTRA Cluster"?. Note that users can perform similar tasks giving the same outcome. Therefore, allowing caching among clusters can improve the application response time.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2021.3064225, IEEE Internet of Things Journal

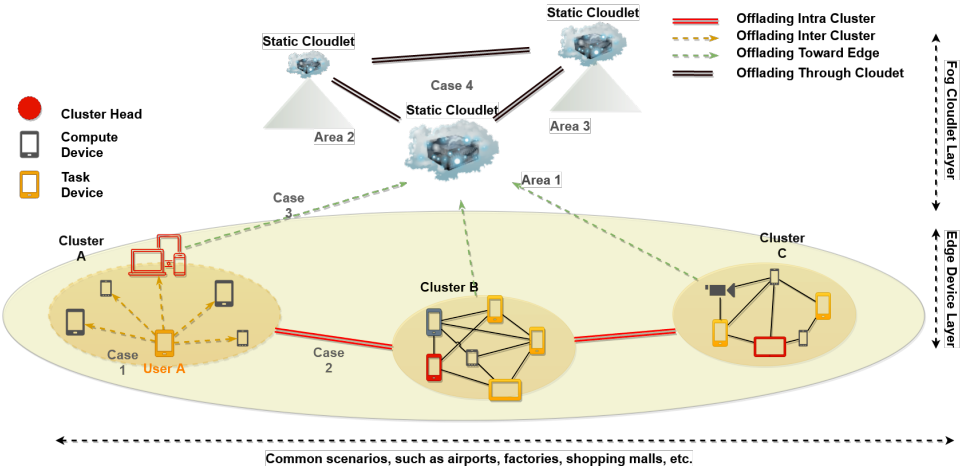


Fig. 2: Different Offloading Cases Through The DCC

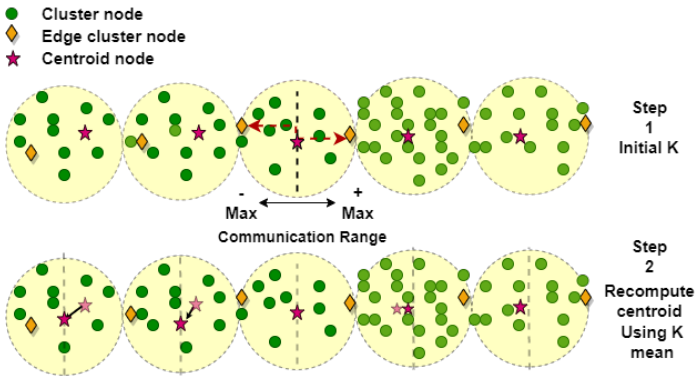


Fig. 3: Dynamic Cloudlet Formation Using K-mean Clustering Algorithm

The cloudlet formation is the process of finding the most adequate clustering method of intermediate servers. Cloudlet formation has been explored in previous works in different area with different networks. K-means algorithm introduced in [24] which is one of the common algorithms used for the clustering process. This algorithm partitions data set into K clusters using the mean euclidean distance. We adjusted the K-mean algorithm to fit our task offloading computing scenario where the cluster head (CH) responsible for task offloading outside the cluster (i.e. offload tasks toward the edge server), and cluster members (CM) receive and send their tasks among each other and toward the CH. In order to prolong the cluster lifetime and to shield the network from the results drawbacks due to the CH mobility, each node in the cluster can act as a CH when the current CH is out of service or under a certain threshold. In addition, the K-mean clustering algorithm strongly depends on the initial centroids, where inappropriate centroids distribution can leads to low performance. Hence, In order to ensure an effective cluster formation, we determine the initial K cluster centroids based on nodes locations, the formulation is presented as follows:

(A) Define initial centroids: Fig. 3

1) First, we define an initial centroid $Center_j$ for all network nodes N, where X_i refers to the location of nodes in the network.

$$Center_j = \frac{\sum_{i=1}^{N} X_i}{N} \tag{1}$$

2) Second, Select next centroids according to the initial centroid $Center_j$ edges (-max, max) in such way that the euclidean distance maximized.
3) After determining the initial K centroids, the eligible nodes will join to the corresponding cluster with the nearest centroid $Center_j$ using the following function:

$$\sum_{j=1}^{k} \sum_{i=1}^{n} \left\| x_i^{(j)} - Center_j \right\|^2 \tag{2}$$

To this end, the K numbers of the cluster centers are designated in such a way that the limitation of the basic algorithm such as the inappropriate distribution of the nodes have been covered.

In order to select the appropriate computing nodes that will serves as CHs, the corresponding cloudlet server calculate the energy residual and computing capacity of each cluster S, $S = [S_1, S_2, \ldots\ldots, S_k]$

1) Calculate the average node computing energy of each cluster.

$$e_{\text{avg}}\left(S_i(j)\right) = \frac{\sum_{j=1}^{|S_i(j)|} e(j)_{\text{r}}}{|S_i(j)|} \quad \forall j \in 1, 2 \ldots, |S_i| \tag{3}$$

where $e(j)_{\mathbf{r}}$ and $e_{\text{avg}}\left(S_i(j)\right)$ are the residual energy of the j_{th} member node and the average cluster energy of S_i ,respectively.

2) Calculate the average node computing capacity of each cluster.

$$c_{\text{avg}}\left(S_i(j)\right) = \frac{\sum_{\text{j=1}}^{|S_i(j)|} c(j)_{\text{r}}}{|S_i(j)|} \quad \forall j \in 1, 2 \ldots, |S_i| \tag{4}$$

3) After determining the computation and energy averages, the potential clusters head nodes are expressed as fol-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2021.3064225, IEEE Internet of Things Journal

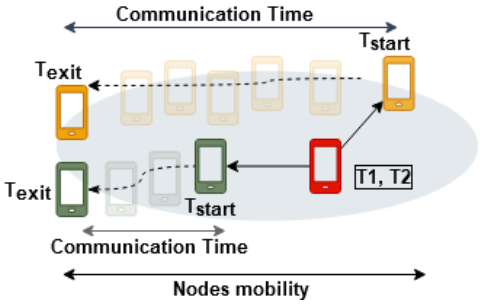


Fig. 4: User Mobility Model

lows:

$$CH\left(c_j, e_j\right) = \begin{cases} \text{Max}(j) & \text{if } c_j > c_{avg} \text{ and } e_j > e_{avg} \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

C. User Mobility Offloading Model

Note that the offloading decision is affected by nodes mobility. For this reason, we assigned to each user a random way-point mobility (RWP) model as defined in [25], which is widely used to describe nodes movement. In this model the mobility trajectories of node i represented by $\text{TR}_i = (X_i X_{i-1}, t_i, v_i)$. $|X_i - X_{i-1}|$ reference to the crossed distance during t_i with the variant speed v_i. Two mobile nodes can communicate if their on the same range $\left|X_i(t) - X_j(t)\right| < R$, $X_i(t) = \{X_1(t), X_2(t) \ldots\ldots X_n(t)\}$ represents the nodes location at instant t and R represents the transmission range as shown in Fig. 4.

To realize a successful and efficient offloading, the communication time is regarded as the main factor to complete the task operation. Since the relation between computing nodes is not stable, we introduce communication time $\text{CT}_\text{j}(\text{t})$ which indicate the link duration time between node j and node i within cluster S_i, represent by:

$$CT_j(t) = \{t : t_{\text{finish}} - t_{\text{start}}\} \tag{6}$$

$$\begin{aligned} t_{\text{start}} &= \inf_{0 \le t \le \tau} \left\{t : \left\|X_i(t) - X_j(t)\right\| \le R\right\} \\ t_{\text{finish}} &= \inf_{0 \le t \le \tau} \left\{t : \forall t' \ge t \cap t' \le \tau, \left\|X_i\left(t'\right) - X_j\left(t'\right)\right\| > R\right\} \end{aligned} \tag{7}$$

D. System Model Description

We defined a global and local middleware works as an SDN controller to fulfill the users offloading requests at the edge and monitoring the network and mobile devices state. The global middleware (GMDW) work on monitoring and redirecting the traffic through all network between the static cloudlets nodes (SC), while the local middleware (LMDW) focuses on managing the local explored area. Fig. 5 shows the architecture model where all mobile devices can subscribe to profit from the SC resources. In this architecture, users notify their states, including location, storage capacity, energy level and CPU to the corresponding SC. GMDW can enhance the network throughput by offloading data to the less overloaded servers across the network according to the user's density variation in different areas. LMDW can control and track users due to their mobility.

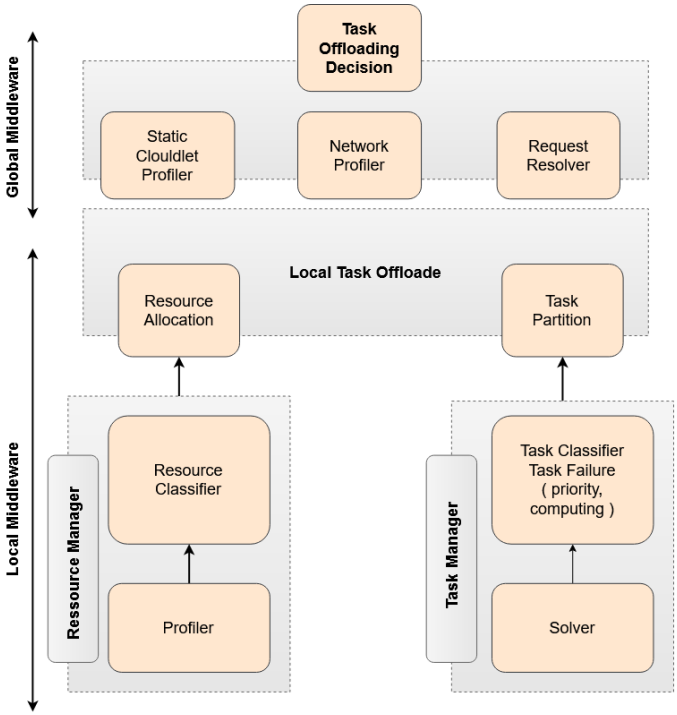


Fig. 5: DCC Computing Model For IoT Devices

Fig. 5 displays global and local middleware (GMDW, LMDW) components, each one associated with different components. First, the GMDW consists of Static Cloudlet Profiler (SCP) includes information concerning the status of supervised SC nodes (server load, location, processing power, etc.) and Static Cloudlet Network Profiler (SCNP) to monitor the network traffic over the available various SCs in the network. While Static Cloudlet Request Resolver (SCRR) is in charge of solving the offloaded heavy tasks. In the end, Task Offloading Decision (TOD) is responsible for task offloading decisions among the SCs based on the gathered information (SCs, network states, and the offloaded requests).

Second, the LMDW consists of task and resource management components, in which all collected information is concerning the edge devices states and the offloaded tasks. Resource Manager involves a Resource Classifier (RC) and a Profiler, the RC works on classifying the available resource on the network using the Profiler metadata. While the Profiler is responsible for monitoring the local network and device states. In addition, the Task Manager includes a Task Classifier and Solver. Where Task Classifier ranges the tasks according to their features and constraints. The Solver takes responsibility for sending and receiving the offloading request towards nodes according to the scheduling decision. Based on the resource and tasks information provided by the previously mentioned components a Local Task Offloaded (LTO) offload the computing tasks to the suitable devices using a partition task and resource allocation algorithms, which will be discussed later in the next sections taking into account the task deadline and their dependencies also the device's connectivity.

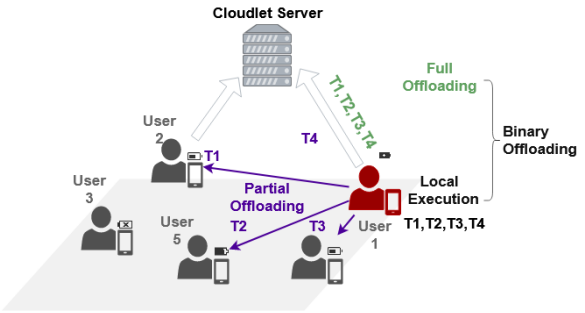


Fig. 6: Different Offloading Computing Strategies

E. Optimal Decision of Task Offloading

Although not all application tasks are suitable for remote execution, binary and partial offloading strategies have been introduced in [26–28] as shown in Fig. 6. Binary Offloading requires a task to be executed as a whole either locally at the device or remotely on the MEC server. Partial Offloading allows a task to be partitioned into two parts, some executed locally and some execute remotely. In practice, binary offloading is easier to implement and suitable for simple tasks that are not partitionable, while partial offloading is favorable for complex tasks that include multiple methods such as augmented reality and face recognition applications.

1) Task Specification

We represent a task flow by an acyclic directed graph G_t=(V_M, E), where V_M=$\{T_1, T_2, T_3, \ldots, T_M\}$ represents the set of computing tasks and E represent the communication link between tasks. A task T_i depends on task T_{i-1} if there is a edge between them. We assume that each computing node can serve more than one task on time based on its available resources, and can offload its computing tasks by a single hop. Each task T_i of V_M associated with following variables (w_i, i_i, o_i, τ_i, e_i), where w_i represent the computation workload, (i_i, o_i) represent the input and output data size, τ_i represent the task execution deadline time and e_i represent the energy. Those parameters related to the application nature such as real-time application e.g "recognition face and augmented reality". Where this type of application requires a hard deadline (i.e. tasks execute respecting to its τ_i) comparing to soft deadline application such networking apps where tasks can accept latency with relatively small τ_i. The task parameters can be forecast or estimated based on the device profiler as mentioned in [21, 29], which facilitates the execution process and assess the communication cost resulting during the tasks interactions.

2) Task Classification

According to [30], the application tasks can be could be classified according to the execution mode, tasks that can be run locally and tasks that must be executed remotely on the cloud, and those that can be executed on both modes. Hence, we distinguished the application tasks into three categories (Local, Shared, and Heavy) tasks. Local tasks refer to un-offloadable tasks denote by T_u (tasks that use local device components such "camera, GPS, user interface"). Shared Tasks denote by T_s refers to communication tasks that require low computation and high communication cost and can executed on both modes, either remotely or locally based on the device's resources. While heavy tasks denote by T_h refers to computation tasks that require a high computation and low communication cost and can be execute remotely at the edge within cloudlet nodes or the cloud. It was pointed out in [31], that shared tasks constitute the majority of network traffic and therefore, an optimum offloading regarding T_s can enhance the network performance.

3) Task offloading Computing Model

In order to ensure an efficient cooperative offloading within the DCC system, we evaluate the energy consumption and task execution time for local and remote task execution.

1) **Local Computing** the execution time and energy consumption for local computation could be expressed as follows:

$$T(i,j) = \omega(\mathrm{i,j})/f_j \quad \text{And} \quad E(i,j) = T(\mathrm{i,j}) * P_j \quad (8)$$

Where f_j is the device CPU frequency and P_j represent the computation energy of device j for task i. $\omega(\mathrm{i,j})$: represent task i workload on device j.

2) **Remote Computing** tasks can be executed remotely on the edge server or remotely at neighbor devices. For that, we discuss the communication overhead in terms of execution time and energy consumption for both cases, respectively:
For a task i_{th} executed by device j, the execution time for both cases (at the nearby mobile device or at the edge node), includes input data uploading, cloudlet execution, and result downloading.

$$T_(i) = \frac{\ln_i}{U} + \frac{\omega_i}{f} + \frac{out_i}{D} \quad (9)$$

For energy cost:

$$e_(i) = \frac{\ln_i}{U} \cdot p_s + \frac{\omega_i}{f} \cdot p_j + \frac{out_i}{D} \cdot p_r \quad (10)$$

p_s and p_r are the transmitting energy for uploading and downloading data.

4) Cooperation Task Execution Strategy

In this subsection, we described the functioning of our proposed offloading algorithm where it demonstrates the offloading phases. When a resource constraint occurs in a device, the designed nodes offload their computing task to their neighbor nodes within the current mobile dynamic cloudlet or to other computing nodes via CHs according to the central scheduler decision. Tasks can be executed parallel or sequentially based on their correlation as shown in Fig. 7, where nodes I and T indicate initiation and termination of the application, respectively, and the intermediate nodes indicate either tasks that can execute remotely or not. For instance, the tasks in Fig. 7 assigned to nearby computing nodes according to certain execution paths, such as $P_1(T3, T7, T9)$ assigned to N_1, $P_2(T4, T6, T8)$ assigned to N_2 and $(T2, T5)$ assigned to N_3. Based on the execution flow, we partition our task graph between I and T in terms of communication usage according to the heaviest execution paths, where it considered a near-optimal execution strategy and we denoted the communication

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2021.3064225, IEEE Internet of Things Journal

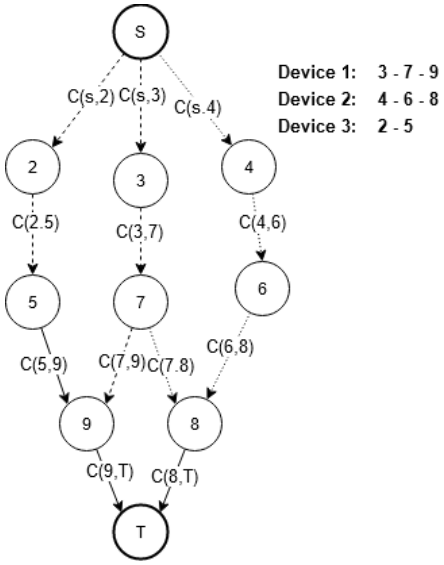


Fig. 7: Application tasks follow topology including communication and computation tasks, vertex weight refers to task workload and the edge weight refers to the ratio of the output data to the input data size

cost by $CC_{i,j}$ where:

$$CC_{i,j} = \frac{in_{i,j}}{B_{upload}} + \frac{out_{i,j}}{B_{download}} \tag{11}$$

Besides that, we represented the task offloading decision on two major cases as illustrated in Fig. 8.

Case 1: N_1 offload tasks to N_2 and N_3. N_2 execute the offloaded task by N_1 and send output results to N_3 to presume its task execution. Distributed tasks among the cluster nodes should be done in an efficient manner to reduce the communication overhead, energy consumption, and delay. Hence, we proposed a greedy task partition algorithm to seek near-optimal allocation to achieve efficient offloading according to available computing resources.

Case 2: N_1 offload tasks to N_c and waiting for the result back to resume its execution, where it is not practical to rely on one edge server on dense networks due to the enormous resulting requests from edge devices. When the edge server is unable to handle the assigned tasks, it will offload part of to its nearby edge servers where smart selection methods are required. which is out of the scope of the current work and can be considered as future work.

Since the application may include dependent and independent tasks, we focused on the dependent communication tasks (shared tasks) that add burden to the network, due to their high interaction cost. We minimized interaction between nodes by mapping tasks with high interactions on the same computing node as illustrated in Fig. 7. This, reduces the communication cost, task execution time, and extend the network lifetime. Consequently, we defined the application overall execution time T_{app} within the computing cluster as follows:

$$\text{Time}_{app} = \text{Time}_i^n + \text{ Time }_i^e \tag{12}$$

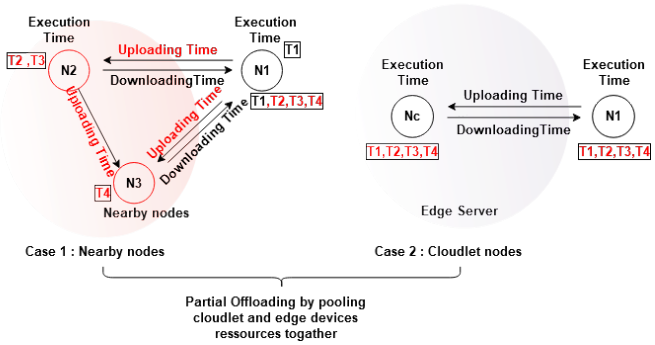


Fig. 8: Tasks Dependency Cases

Where $Time_i^n$ indicate the tasks execution time at nearby devices and $time_i^e$ indicate the execution time at the edge nodes.

$$\text{Time}^n_{i,N_d \in Q_j} = \sum_{i=1}^{|Q_j|} \sum_{j=1}^{M} C\left(\delta_{i,j}^{u,d}\right) + UD\left(\delta_{i,j}^{u,d}\right) + W\left(\delta_{i,j}^{u,d}\right) \tag{13}$$

N_d denotes the selected node to perform the assigned task $\delta_{i,j}^{u,d}$ of node N_u, Where $C\left(\delta_{i,j}^{u,d}\right)$ represent computing time of task j at node i, $UD\left(\delta_{i,j}^{u,d}\right)$ represent data upload and download time and $W\left(\delta_{i,j}^{u,d}\right)$ denote waiting data receiving time.

More formally, our proposed task scheduling allocation algorithm can formally be defined as the succeeding integer linear programming problem (ILP) 0-1 where decision variable $d_{i,j}$ addressed properly. $d_{i,j} = 1$ indicates that the j_{th} task is assigned to the remote node i.

$$\text{minimize} \quad \sum_{i=1}^{|Q|} \sum_{j=1}^{|M|} CC_{ij} \cdot d_{i,j} \tag{14}$$

$$\text{subject to} \quad \sum_{j=1}^{|M|} d_{i,j} T_{i,j} \leq CT_i(\tau), i = 1, 2 \ldots, |M| \tag{15}$$

$$\sum_{i=1}^{|Q|} d_{i,j} c\left(\delta_i^u\right) \leq c_i(\tau), i = 1, 2 \ldots, |Q| \tag{16}$$

$$d_{i,j} = 0, 1, i = 1, \ldots |Q|; j = 1, \ldots |M| \tag{17}$$

5) Centralized Task Scheduling Algorithm Description

Note that the response time of nearby computing nodes to deliver the output task result could vary depending on the computing capacity, connection time, and network condition. For that, we defined an electing strategy to evaluate the computing nodes' eligibility within each mobile cloudlet.

1) **Qualified Nodes Election Process** Node mobility could be a barrier during the offloading process leading to a task failure. Therefore, we set low task computation and communication time barriers to deal with task execution time respecting to task deadline within the cluster S_i nodes. Moreover, we maintained appropriate energy e_j to support the task completion time. Note that the

Algorithm 1: Greedy Task Graph Partition Algorithm

Input: Cluster node N_s, Originator N_i, Task graph G_t=< V_M, E >, V_M = Application Tasks, E = Dependency links

Output: Mapped task among our DCC Edge Tasks (i.e. Cloudlet Server), Unoffloadable Tasks (i.e. locally), Shared Tasks (i.e. Neighbor Devices)

```
initialization
Create a queue PrimaryTasks.
PrimaryTasks := S successors in AGD.
S := Represent DCC Asc nodes in cluster S_K according their resources.
for each Node i of S_k do
    if CT_i(t) > C(δ_i^{u,d}) and ET_i(t) > D(δ_i^{u,d}) then
        if Node i have enough Energy then
            Add Node i to Eligible Node List LENi.
for each Node j of LEN_k do
    while NoFinsih do
        E_i := Max(Path) if CurrentTask! = TermineTask and CurrentTask nonExplored then
            if C_j > C(δ_i) then
                Assign Task_i To Node_j
                Task_i is Explored
            else
                i=k Last Explored Node
                Break
        else
            if CurrentTask == TermineTask Or CurrentTask isExplored then
                Remove Ti From PrimaryTasks Queue.
                UPDATE(PrimaryTasks).
                CurrentTask:= PrimaryTasks1.
                if C_j > C(δ_i) then
                    Assign Task_i To Node_j.
        if PrimaryTasks isEmpty then
            Finish := True
```

communication time (CT) that can be obtained using a prediction model [29] should be reasonable to meet the task needs. While exchange time (ET) defined to ensure the successful data transfer between the originator and the neighbor nodes. Hence, the selection process must satisfy the following condition:

$CT_j > C\left(\delta_{i,j}^{u,d}\right), ET_j > D\left(\delta_{i,j}^{u,d}\right)$ and $e_j > E$

2) **Task Graph Partition**

After determining the eligible nodes sets Q within the mobile cloudlets, we introduce a greedy task graph offloading partition algorithm for acyclic task graph G_t. The algorithm takes a G_t as input, which includes task nodes represented as vertices and communication link between nodes represented as edges. Note that the communication cost between tasks at the same node is negligible.

- **Tasks Classification:**
  First, we distinguished tasks into different sets of unoffloadable tasks (Local), computation tasks (heavy), and communication tasks(shared). Unoffloadable tasks execute on the device due to its special features while the remaining tasks execute remotely. Apart from this, unoffloadable and computation tasks, each one can be regarded as one vertex where their communication and computation costs aggregated separately. Let G_{ts} represent the resulting graph after merging the initial graph.
- **Classification of Eligible Nodes:**
  Once the eligible nodes have been identified, they are sorted from the highest to the lowest weight according to their computational weight.
- **Edge Cut Partition:**
  This step aims to assign the G_{ts} resulting graph tasks within the selected eligible nodes, such as communication cost minimized.

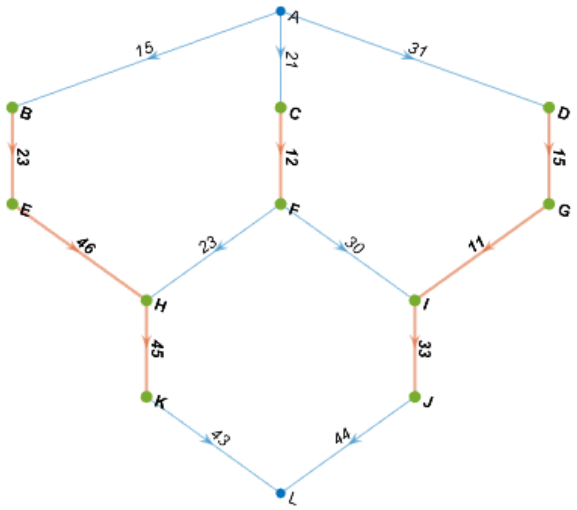


Fig. 9: Task Flow Partition Result

IV. Experimental and Simulations Evaluation

To evaluate our proposed partitioning algorithm, we conducted our experiment scenario over a large number of simulations, where we assign to each task node a computation weight and a communication weight for each edge. We have implemented the GTGP algorithm using Matlab that can serve as proof of its feasibility and utility. The algorithm assigns the target tasks to the mobile devices with the highest relative computing capacity, respecting devices connection, and energy constraints. For instance, Fig. 9 shows the partition result in which the start task and the terminate task methods are assumed unoffloadable tasks. Tasks D, G, I, J assigned to device 1, tasks C, F assigned to device 2, and tasks B, E, H, K assigned to device 3. Mobile devices 1 and 3 receive the output resulting data from the device 2 to presume the execution of their tasks with the following edge costs (E-H and E-I).

A. Performance of GTPG Algorithm

We compared the overall communication cost of mobile devices with random and uniform offloading strategies to our proposed GTGP algorithm under different numbers of tasks i.e. “different application size”. As shown in the Fig. 10. Compared to the other two algorithms, the GTGP algorithm shown the lowest communication cost. Because GTGP works on exploiting the device computing resources completely and

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2021.3064225, IEEE Internet of Things Journal

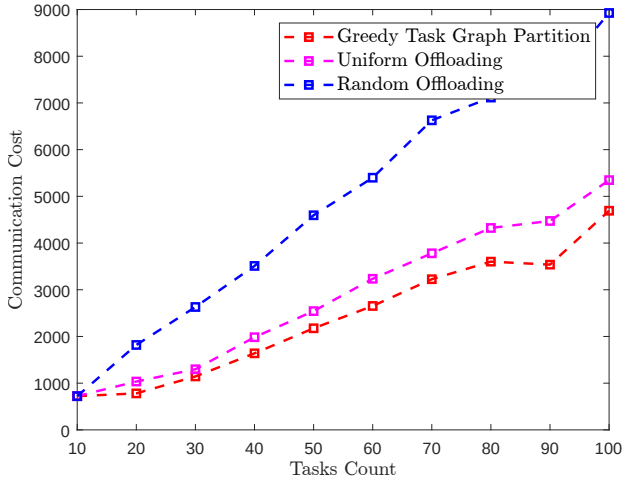


Fig. 10: Communication Cost During The Offloading Process

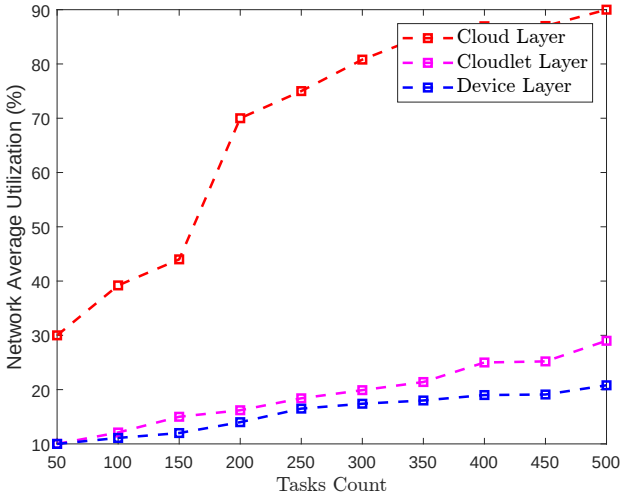


Fig. 12: Average Network Utilization

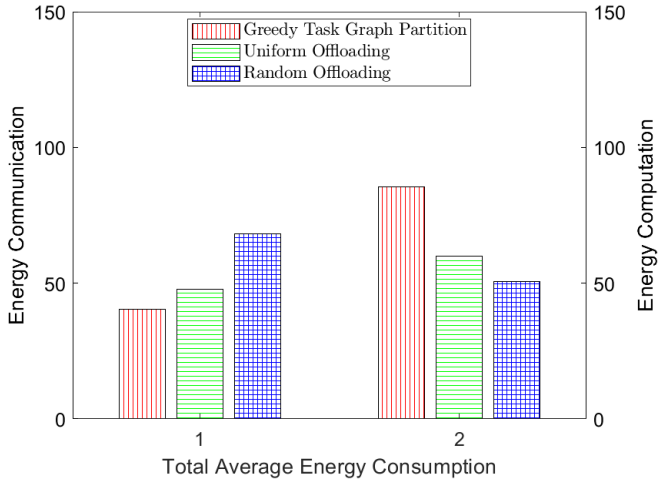


Fig. 11: Energy Average consumption during the offloading process

executes the correlated tasks with the high interaction cost in the same device. While random offloading algorithm runs separately in different devices, which may add more delay due to the resulting transmission. We observed that the uniform offloading strategy with the small-scale application can give similar performance to our algorithm. This relates to the task graph bifurcation where the probability to flow the same execution path is high at the start-up phase, while as the application size scales the communication cost increases and diverges from each other.

1) Computation and Communication Energy consumption

In Fig. 11, we compared the performance of our proposed algorithm during the offloading process to the random and uniform algorithms overall different shared tasks i.e (tasks with low workload and high interaction) in terms of the following-Axes. The first axe represents, energy consumption resulting due to the data transmission i.e. (task interactions), and the second axe represents, energy consumption for task execution. Random and uniform offloading strategies usually work on assigning tasks separately to the different available computing nodes which can be practical for independent tasks while increasing the communication costs for dependent tasks. Our GTGP algorithm shows low energy consumption considering the data transmission overall the two prior mentioned strategies while showing high energy consumption for the computing process. We note that the gain energy brought by decreasing the communication cost can affect inversely the energy devices due to the task computing burden on the device. However, the energy consumption resulting due to the the computing can be acceptable in case the communication cost is very high.

B. Performance of DCC Architecture

The application tasks could be executed in a cooperative manner where multiple heterogeneous devices can share their resources within the proposed computing system DCC. In this subsection, we show the advantages of DCC layers separately and completely on the function of the network usage, task failed ratio, and task completion delay.

1) Network Usage

We executed our application with DCC layers as shown in Fig. 12 where the application consists of high computation and communication tasks, high computation tasks should be offloaded to the cloudlet or cloud nodes which is clear due to the surrounded device resource limitation, and the high communication tasks should be offloaded to the nearby devices, according to the scheduler decision. Fig. 12 shows the offloading process of communication tasks where it shows that relying only on the cloud consumes more bandwidth comparing to the cloudlet and devices while offloading tasks to the nearby nodes showing the lowest consumed bandwidth.

2) Task Failure Ratio

In our experiment, we adopted a clustering schema to form a dynamic mobile cloudlet where each mobile within the cloudlet able to receive and send tasks. When a mobile node intends to offload its computing tasks, it starts by offloading tasks to eligible nodes within the same cloudlet and then offloading outside the mobile cloudlet according to the task scheduler. Note that the offloading outside the mobile cloudlet is affected through the cloudlet server to avoid extra routing

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2021.3064225, IEEE Internet of Things Journal

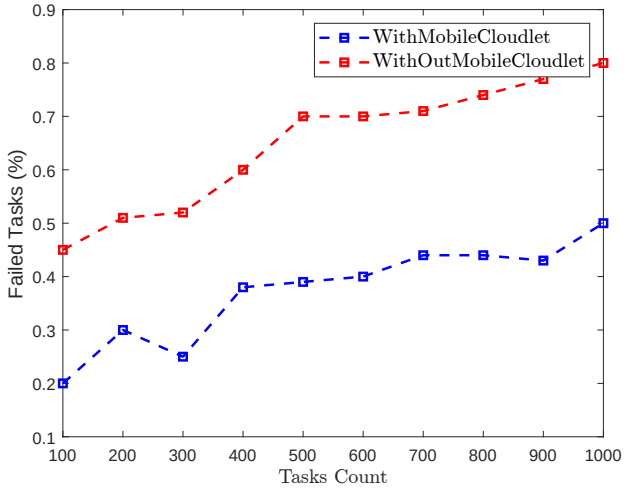


Fig. 13: Tasks Failed Due To The Link Instability

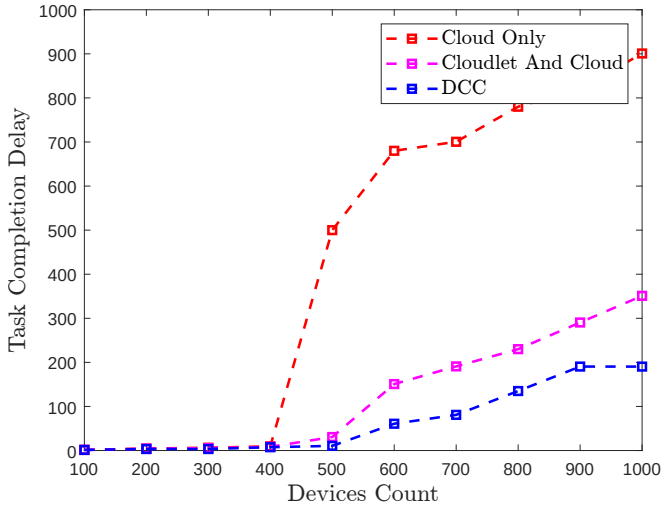


Fig. 14: Average Task Completion Delay

cost, which is considered a challenge in a dynamic network. Fig. 13 shows the task failure ratio overall offloaded tasks, wherewith 100 tasks, the task failure ratio is 0.2 with mobile cloudlet, and 0.45 without mobile cloudlet. We observe that offloading tasks failed ratio via mobile cloudlet is the lowest for various application scales and this relies on the reliability of the connection given by the mobile cloudlet to the cloudlet server.

3) Task Completion Delay

Fig. 14 shown the average delay of our proposed scheduling algorithm overall different architectures (Cloud-Only, CloudAndCloudlet, DCC). In the beginning of the offloading process, the average completion delay is almost identical for the different architectures, then it starts increases significantly, wherewith the Cloud-Only the delay increases to 9.5 seconds with 400 devices and to 900s with 1000 devices. And for the CloudAndCloudlet the delay changes from 9.5s with 400 devices to 400.3s with 1000 devices. Hence, this difference relies on the computing nodes' location. Although offloading computing with CloudAndCloudlet shown acceptable results comparing to the cloud but it's still limited in dense networks. However, when the cloud and cloudlet nodes unable to fulfill the users offloading requests, integrating the devices layer within the cloud and cloudlet layers showed efficient results, where the delay achieve (2s,5s) with (400,500) devices respectively and 400s with 1000 devices.

C. Experiment Evaluation

To explore the feasibility of our experiment, we have built a software-defined as proof of concept for our DCC architecture (SDDCC). We chose a facial recognition system as a scenario for our experiment due to its intensive computing tasks. It consists of smart mobile phones and a cloudlet server. Smart mobile phones act as an IP mobile camera. The cloudlet server works for automatic face recognition or verification of an individual from a digital image or a video frame from a live streaming source. It can be used primarily in crowded areas such as airports, stadiums, and army areas. Using a face recognition application could be a good demonstration to show the experiment results due to its task diversity, where consists of different computation and communication tasks such as capture frames task, face extraction task, face detection task, extraction face features, matching face, etc.

1) Experiment Setup

We defined a central scheduler within our specified SDN to determine optimum tasks scheduling, using the input information provided by the network and edge devices. Based on the node resources, eligible computing nodes within the formed mobile clusters are selected to share their resources which subject to prior mention constraints in the previous sections. In addition, note that the central scheduler is familiar with the entire local network where includes device profiles. Furthermore, in order to evaluate our proposed GTGP algorithm performance, we analyzed a face-recognition application using application profiler in [32] to select sub computing task, including dependent and independent tasks. Fig. 15 represents the task recognition face call graph that describes the application process. In the experiment scenario, we implement a compute-intensive recognition face mobile application using Opencv Library and NDN using JNI function and determine its corresponding tasks according to [32] where the application includes 19 tasks. Moreover, we build central middleware using JAVA work on monitoring the hull network and device profiling. We run the computes tasks in the proposed framework, which contains infrastructure-based static cloudlets and dynamic mobile cloudlets.

2) Experiment Configuration

The application and scenario experiments are based on an environment depicted in table I. We have fixed the numbers of mobile devices between 10 and 30 devices. These devices have a processing frequency of 300MHz-600MHz, respectively and the targeted application includes 19 tasks. A server node acts as a cloudlet node located one hop away from the mobile devices with a computing capacity of 0.5GHz-3GHz and a quad-core CPU.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2021.3064225, IEEE Internet of Things Journal

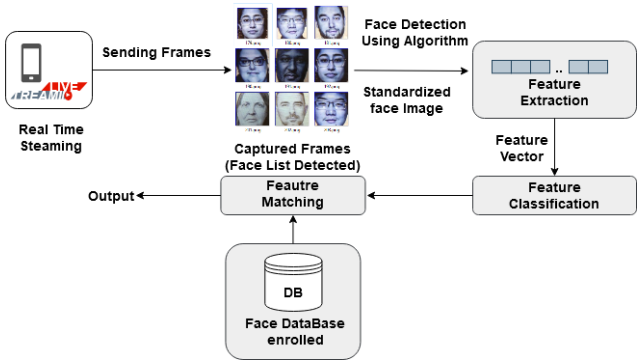


Fig. 15: Application Call Graph

Experiment Parameter Settings	
Parameter	Value
SMD	30
Edge Node	1
Edge Server Capacity	0.5GHz - 3GHz
Edge Devices Capacity	300MHz - 600MHz
Energy Device Status	50% - 100%
Network Type	Random
Task Number	19
Bandwidth	1Mbs-100 Mbs
Video size	250 MB

TABLE I: Experiment Parameter Setting

3) Application Response Time

To get an accurate estimation of the application response time, we run the face recognition application tasks within various architectures, locally at the device or remotely with CloudAndCloudlet and the DCC. It can be seen clearly that in Fig. 16 the response execution time with CloudAndCloudlet and the DCC is significantly reduced compared to the local device execution. It takes more than 15 minutes to process and detect 200 frames of streaming on mobile devices, while it takes less than 1 minute with the other remote offloading strategies. We observe that at the beginning, the remote offloading strategies have identical response time and then it starts increasing with video size scaling, We note that the offloading over DCC showed the lowest response time compared to CloudAndCloudlet and Cloudlet only, confirming that the offloading through DCC can improve the system performance.

4) Energy Consumption Within Different Offloading Strategies

Fig. 17 shows the results of the average communication energy consumed within the different computing clusters using random, uniform offloading strategies, and the GTGP offloading algorithm. In order to measure the communication energy, we select random devices within clusters 1 and 2 which have insufficient energy. Each cluster consisting of 10 devices at most. The results showed that the energy usage of the GTPG offloading algorithm is the lowest compared to the two other strategies. This relies on the task assignment strategy during the offloading process. By using random/uniform offloading strategies, tasks can be assigned equitably/inequitably to the

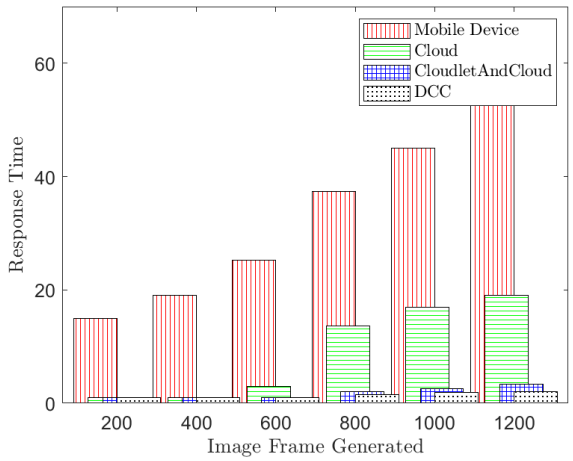


Fig. 16: Response Time Within Different Architecture

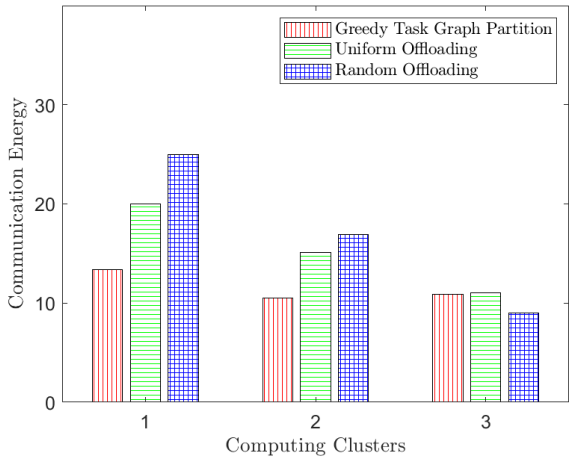


Fig. 17: Cluster Energy Usage

surrounded computing nodes which consume more resources. By our GTGP algorithm, the assignment process among computing clusters is affected based on devices' capacities and their locations which can reduce energy consumption. We note that the energy consumed by cluster 1 is higher than the other two clusters. Since the randomly selected nodes tend to offload their computing tasks within the same cluster node than to the other nodes in other clusters.

V. Conclusion

In this paper, we have evaluated the task and resource allocation problem of multiple tasks for a single application within the DCC environment taking into account the task dependencies and user mobility, and we have introduced a greedy task graph partition GTGP offloading algorithm, where the tasks scheduling process is assisted according to the device computing capabilities with following a greedy optimization approach to minimize the tasks communication cost. Over trace-driven and randomized simulations, the results show that our GTGP algorithm outperforms effectively over random and uniform offloading strategies. In addition, we build a

framework works as an SDN based on managing the offloading process in a centralized manner. Moreover, we implement a compute-intensive mobile application to operate in DCC architecture, which mostly includes infrastructure-based cloudlets, mobile cloudlets, and cloud. The experiment results also show that the performance of our proposed mechanism is excellent.

References

[1] P. Srivastava and R. Khan, "A Review Paper on Cloud Computing," *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 8, no. 6, p. 17, jun 2018.

[2] H. Bangui, S. Rakrak, S. Raghay, and B. Buhnova, "Moving to the edge-cloud-of-things: Recent advances and future research directions," p. 17, jun 2018.

[3] Y. He, J. Ren, G. Yu, S. Member, Y. Cai, and S. Member, "D2D Communications Meet Mobile Edge Computing for Enhanced Computation Capacity in Cellular Networks," *IEEE Transactions on Wireless Communications*, vol. 18, no. 3, pp. 1750–1763, 2019.

[4] D. Kovachev and R. Klamma, "Framework for Computation Offloading in Mobile Cloud Computing," *International Journal of Interactive Multimedia and Artificial Intelligence*, vol. 1, no. 7, p. 6, 2012.

[5] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A Survey on Mobile Edge Computing: The Communication Perspective," *IEEE Communications Surveys and Tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017.

[6] F. Messaoudi, A. Ksentini, and P. Bertin, "On Using Edge Computing for Computation Offloading in Mobile Network," in *2017 IEEE Global Communications Conference, GLOBECOM 2017 - Proceedings*, vol. 2018-Janua. edge2020: Institute of Electrical and Electronics Engineers Inc., jul 2017, pp. 1–7.

[7] Q. Tang, L. Chang, K. Yang, K. Wang, J. Wang, and P. K. Sharma, "Task number maximization offloading strategy seamlessly adapted to UAV scenario," *Computer Communications*, vol. 151, pp. 19–30, feb 2020.

[8] H. Yuan, J. Bi, M. Zhou, J. Zhang, and W. Zhang, "Profit-Maximized Task Offloading with Simulated-annealing-based Migrating Birds Optimization in Hybrid Cloud-Edge Systems," in *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE, oct 2020, pp. 1218–1223. [Online]. Available: https://ieeexplore.ieee.org/document/9283467/

[9] Y.-h. Kao, S. Member, and B. Krishnamachari, "Hermes : Latency Optimal Task Assignment for Resource-constrained Mobile Computing," no. December 2018, 2017.

[10] "Femto Clouds: Leveraging Mobile Devices to Provide Cloud Service at the Edge - IEEE Conference Publication."

[11] S. Sundar and B. Liang, "Communication Augmented Latest Possible Scheduling for cloud computing with delay constraint and task dependency," *Proceedings - IEEE INFOCOM*, vol. 2016-Septe, pp. 1009–1014, 2016.

[12] B. G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "CloneCloud: Elastic execution between mobile device and cloud," in *EuroSys'11 - Proceedings of the EuroSys 2011 Conference*, 2011, pp. 301–314.

[13] X. Wei, S. Wang, A. Zhou, J. Xu, S. Su, S. Kumar, and F. Yang, "MVR: An Architecture for Computation Offloading in Mobile Edge Computing," *Proceedings - 2017 IEEE 1st International Conference on Edge Computing, EDGE 2017*, pp. 232–235, 2017.

[14] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, "ThinkAir: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading," in *Proceedings - IEEE INFOCOM*, 2012, pp. 945–953.

[15] C. Wang and Z. Li, "Parametric analysis for adaptive computation offloading," in *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, vol. 1. Association for Computing Machinery, 2004, pp. 119–130.

[16] O. Castro-Orgaz, W. H. Hager, O. Castro-Orgaz, and W. H. Hager, "Computation of," *Shallow Water Hydraulics*, pp. 183–200, 2019.

[17] Y. Inag, M. Demirci, and S. Ozemir, "Implementation of an SDN Based IoT Network Model for Efficient Transmission of Sensor Data," *UBMK 2019 - Proceedings, 4th International Conference on Computer Science and Engineering*, pp. 682–687, 2019.

[18] M. Jia, J. Cao, and L. Yang, "Heuristic offloading of concurrent tasks for computation-intensive applications in mobile cloud computing," in *2014 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2014, pp. 352–357.

[19] M. A. H. Abdel-Jabbar, I. Kacem, and S. Martin, "Unrelated parallel machines with precedence constraints: Application to cloud computing," in *2014 IEEE 3rd International Conference on Cloud Networking, CloudNet 2014*. Institute of Electrical and Electronics Engineers Inc., nov 2014, pp. 438–442.

[20] Y. H. Kao and B. Krishnamachari, "Optimizing mobile computational offloading with delay constraints," in *2014 IEEE Global Communications Conference, GLOBECOM 2014*. Institute of Electrical and Electronics Engineers Inc., feb 2014, pp. 2289–2294.

[21] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "Maui: making smartphones last longer with code offload," in *Proceedings of the 8th international conference on Mobile systems, applications, and services*, 2010, pp. 49–62.

[22] D. Mazza, D. Tarchi, and G. E. Corazza, "A cluster based computation offloading technique for mobile cloud computing in smart cities," in *2016 IEEE International Conference on Communications, ICC 2016*. IEEE, jul 2016.

[23] L. Xiang, B. Li, and B. Li, "Coalition formation towards energy-efficient collaborative mobile computing," *Proceedings - International Conference on Computer Communications and Networks, ICCCN*, vol. 2015-Octob, 2015.

[24] N. Shi, X. Liu, and Y. Guan, "Research on k-means clustering algorithm: An improved k-means clustering algorithm," in *3rd International Symposium on Intelligent Information Technology and Security Informatics, IITSI 2010*, 2010, pp. 63–67.

[25] S. Deng, L. Huang, J. Taheri, and A. Y. Zomaya, "Computation Offloading for Service Workflow in Mobile Cloud Computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 12, pp. 3317–3329, dec 2015.

[26] Y. Lan, X. Wang, C. Wang, D. Wang, and Q. Li, "Collaborative computation offloading and resource allocation in cache-aided hierarchical edge-cloud systems," *Electronics*, vol. 8, no. 12, p. 1430, 2019.

[27] Y. He, J. Ren, G. Yu, and Y. Cai, "D2D communications meet mobile edge computing for enhanced computation capacity in cellular networks," *IEEE Transactions on Wireless Communications*, vol. 18, no. 3, pp. 1750–1763, 2019.

[28] H. Wu, W. Knottenbelt, K. Wolter, and Y. Sun, "An optimal offloading partitioning algorithm in mobile cloud computing," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9826 LNCS, pp. 311–328, 2016.

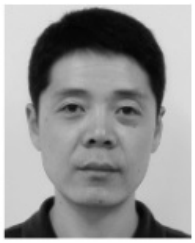
[29] L. Luo and B. E. John, "Predicting task execution time on handheld devices using the keystroke-level model," in *Conference on Human Factors in Computing Systems - Proceedings*, 2005, pp. 1605–1608.

[30] A. Khanna, A. Kero, and D. Kumar, "Mobile cloud computing architecture for computation offloading," *Proceedings on 2016 2nd International Conference on Next Generation Computing Technologies, NGCT 2016*, no. October, pp. 639–643, 2017.

[31] ——, "Mobile Cloud Computing Architecture for Computation Offloading," *2016 2nd International Conference on Next Generation Computing Technologies (NGCT)*, no. October, pp. 639–643, 2016.

[32] Y. Zhang, H. Liu, L. Jiao, and X. Fu, "To offload or not to offload: An efficient code partition algorithm for mobile cloud computing," *2012 1st IEEE International Conference on Cloud Networking, CLOUDNET 2012 - Proceedings*, pp. 80–86, 2012.

Naouri AbdeNacer received his B.S. degree in computer science from the University of Djelfa Algeria, in 2011, and the M.Sc. degree in networking and distributed systems from the University of Laghouat Algeria, Laghouat, Algeria, in 2016. He is currently pursuing the Ph.D. degree with the University of Science and Technology Beijing China, Beijing, China. His current research interests include Cloud computing,Smart communication,machine learning,Internet of vehicles and Internet of Things.



HANGXING WU received his B.S. degree in automation control from Chang'an University, China in 2001, and his Ph. D. degree in control science and technology from Northwestern Polytechnical University in 2008. He is now an associate professor in School of Computer and Communication Engineering, University of Science and Technology, Beijing, China. His current research interests include flow control and congestion control, high speed networks, data center networks and Mobile Edge Computation.

Nouri Nabil Abdelkader Received his engineer's degree in computer sciences from the university of Laghouat, Algeria, in 2003 and his magister degree in networking and distributed systems from the university of Bejaia, Algeria, in 2007. His current research interests include Wireless Networking Design, Internet of Things, Performance Evaluation, Fog Computing, Optimization.

Sahraoui Dhelim Received his B.S. in Computer Science from the University of Djelfa, Algeria, in 2012 and his Master degree in Networking and Distributed Systems from the University of Laghouat, Algeria, in 2014. And PhD in Computer Science and Technology from University of Science and Technology Beijing, China, in 2020. His current research interests include Social Computing, Personality Computing, User Modeling, Interest Mining, Recommendation Systems and Intelligent Transportation Systems. Interest Mining,

Huansheng Ning Received his B.S. degree from Anhui University in 1996 and his Ph.D. degree from Beihang University in 2001. Now, he is a professor and vice director of the School of Computer and Communication Engineering, University of Science and Technology Beijing, China. His current research focuses on the Internet of Things and general cyberspace. He is the founder and chair of the Cyberspace and Cybermatics International Science and Technology Cooperation Base. He has presided many research projects including Natural Science Foundation of China, National High Technology Research and Development Program of China (863 Project). He has published more than 150 journal/conference papers, and authored 5 books. He serves as area editor for IEEE Internet of Things Journal (2020-2022), and editor role for some other journals. He is a visiting chair professor of Ulster University, UK.