SLA-aware Approach for IoT Workflow Activities Placement based on Collaboration between Cloud and Edge

Awatif Alqahtani
School of Computing, Newcastle University
Newcastle, UK
awa.alqahtani1@gmail.com

Devki Nandan Jha
School of Computing, Newcastle University
Newcastle, UK
i.dnjha@gmail.com

Pankesh Patel
Pandit Deendayal Petroleum University
Gandhinagar, INDIA
dr.pankesh.patel@gmail.com

Ellis Solaiman
School of Computing, Newcastle University
Newcastle, UK
ellis.solaiman@ncl.ac.uk

Rajiv Ranjan
School of Computing, Newcastle University
Newcastle, UK
raj.ranjan@ncl.ac.uk

ABSTRACT

In the Internet of Things (IoT) era, various nodes generate vast quantities of records, and data processing solutions consist of a number of activities/tasks that can be executed at the Edge of the network or on the Cloud. Their management at the Edge of the network may limit the time required to complete responses and return the final result/analytic to end users or applications. Also IoT nodes can perform a restricted amount of functionality over the contextual information gathered, owing to their restricted computational and resource capacities. Whether tasks are assigned to an Edge or a Cloud is based on a number of factors such as: tasks' constraints, the load of nodes, and energy capacity. We propose a greedy heuristic algorithm to allocate tasks between the available resources while minimizing the execution time. The allocation algorithm considers factors such as the deadline associated with each task, location, and budget constraint. We evaluate the proposed work using iFogSim considering two use case studies. The performance analysis shows that the proposed algorithm has minimized cost, execution time, control loop delay, networking, and Cloud energy consumption compared to the Cloud-only approach.

KEYWORDS

IoT, Allocation Algorithm, Placement Algorithm, SLA

1 INTRODUCTION

Cyber-Physical Systems (CPSs) are integrated systems that aim to bring physical entities together with the all-present computing and networking systems [2]. An important concept in close association with CPS is the Internet of Things (IoT), which promotes the use of emerging technologies and architectures for large-scale applications to define and virtualize physical objects [2]. The IoT is a revolutionary technology that provides a completely linked "intelligent" universe, accelerating the 4th industrial revolution in which millions of things are linked with each other on the globe [4]. These items share information and services to define, track, and handle the physical environment, enabling different applications e.g. smart cities, agriculture, and health care applications to transform our lifestyle and enhance both quality of life and human civilization. However, the large-scale realization of IoT services is restricted by the constraints of IoT devices (integrated with everyday objects such as cars, utility parts, and sensors) and their limitations in computing resources, memory capability, power, and bandwidth.

Many of these problems could be solved by using Cloud-Assisted Things or Cloud of Things (CoT) technology as it provides large-scale and on-demand computing resources for managing, storing, processing, and sharing IoT information and services [4] [5]. However, with the increasing number of applications, and their time-sensitive nature, Cloud-based solutions are not enough and mostly suffer from latency due to the centralized nature of Cloud data centers, which are mostly located at a far distance from the data sources. Therefore, utilizing Edge resources while benefiting from the Cloud whenever it is required, is essential for time-sensitive applications, and for overcoming the problems associated with centralized control. However, there are a number of challenges to consider such as maximizing the utilization of Edge resources while considering the limitations of their computation capabilities. Also there is a possibility that some of the tasks can be time-sensitive and it is therefore crucial to be allocated and executed immediately. Executing forthcoming tasks requires proper task allocation and scheduling which satisfies the requirement of all the tasks while maintaining the SLA (service level agreement).

There are certain challenges that need to be considered while making the allocation and scheduling decisions. The main challenges are given below.

(1) *The uncertainty of the data-in rate*: The IoT captures data from the physical environment, which is dynamic in nature. For example, in camera-based applications that capture people passing by a certain point, the number of pictures taken varies for many reasons (e.g., rush hours vs peak hours, weekdays vs weekends, or if there are certain events nearby). Therefore, it is essential to rerun the algorithm to reschedule the activities whenever the data-in rate exceeds the predefined data-in rate.
(2) *The conflict between objectives*: To minimize the cost, it is better to deploy the activities on Edge resources; however, in some cases, deploying activities on Edge resources affects the throughput of an application and it will suffer from resource bottlenecks due to the limited processing capabilities.

1st Workshop on Cyber-Physical Social Systems (CPSS2019), October 22, 2019, Bilbao, Spain.

Copyright © 2019 for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR-WS.org/Vol-2530/paper5.pdf

To address these challenges while making the allocation, we propose a layered-based algorithm that identifies and minimizes the global bottleneck, i.e., minimizing the processing time and the cost as well as maximizing the utilization of resource computation at the Edge layer. The main contributions of this work are summarized as follows:

- We consider a task placement approach based on cooperation between the Edge and Cloud that supports service decentralization, leveraging context-aware information such as location and available computing and storage capacities of Edge resources. The placement approach considers deadline constraints associated with each task as a way to emphasize utilizing Edge resources and a budget constraint at application level. This maximizes the utilization of Edge resources and minimizes latency, energy consumption, and cost.
- We conduct a performance analysis with various case studies using iFogSim[1] to reveal the effectiveness of the proposed approach in terms of maximizing the utilization of Fog devices while reducing latency, energy consumption, and network load.

2 SLA- AND CONTEXT-AWARE APPROACH FOR IOT ACTIVITY PLACEMENT ACROSS CLOUD AND EDGE

In this approach, we consider IoT applications, which mostly consist of a set of activities, some of which requires high bandwidth and low computation. They can be performed on one of the resources at the Edge of the network, while if an activity requires more computation, then it can be offloaded to the Cloud. In the case where there is more than one activity, selecting which one to deploy at the Edge or the Cloud level can be based on different criteria (e.g., cost, distance, location, processing time). Our aim is to provide a solution that distributes the workflow activities in such order that optimize both consumer requirements and utilizes computation capabilities at the Edge, meanwhile aiming to avoid any SLA violation.

2.1 Problem Definition and Modeling

In IoT applications, task/activity placement is the problem of allocating tasks/activities to a set of processors (resources) that are distributed across layers (Edge and Cloud). The input to the task placement controller is an activity graph and a processor graph, and the output is a placement plan that maps each activity to a suitable processor/resource. Whether the resources are located at the Edge or the Cloud is based on each task/activity's computation and communication requirements.

2.1.1 Multi-layer Computing Architecture for IoT. This section presents the proposed multi-layer architecture for IoT. It also describes the task placement information and the necessary concepts related to the proposed scheme. Figure 1 denotes the three main layers as described below:

- IoT devices: the layer consists of devices with sensing and actuating purposes. It also has internet connection capability for transferring the data to Edge or Cloud.
- Edge Computing layer: this includes the upper layer of the IoT devices which provides lightweight computation. Resources in this layer are distributed to cover different regions. In other words, resources can be clustered by the region which covers the IoT devices within that region to connect and transfer data.
- Cloud Layer: It is the centralized layer where heavy computation/storage processes can be delegated.

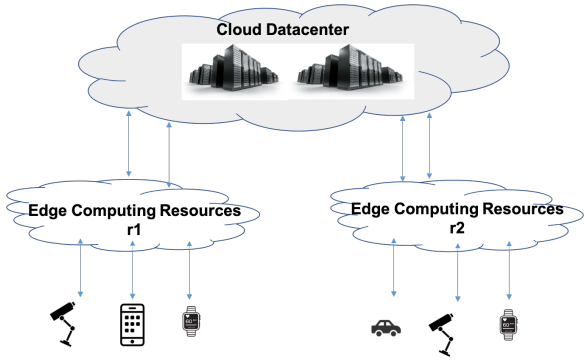

Figure 1: Tasks dependencies example for an IoT application

2.1.2 Task Graph: A task graph is represented by a directed acyclic graph (DAG), $TG = (T, E)$ where the set of nodes $T = \{t_1, t_2, t_3, ..., t_n\}$ represent n tasks. Between tasks, there is a set of edges belongs to E which represents the data dependency between nodes. For example, tasks t_1 and t_2 are connected by $e_{1,2}$. In another word, between any t_i and t_j, there is $e_{i,j}$ belongs to E. Thus, we can define Edges and Tasks as follow:

$$\forall(t_i \wedge t_j \in T),\ \exists\ e_{i,j} = (t_i, t_j)\ \in E$$

Consider T represents a task which is defined as $T = \{T_{id}$, ReqPCapcty, deadline, region, level$\}$. T_{id} represents the task Id, $ReqPCapcty$ represents requested processing capacity, $deadline$ represents the deadline constraint of a single task while $region$ reveals the region/location where this task is preferable to be deployed. Finally $level$ is used to denote how many hops this task is far from the starting point which in our case is sensing events.[2]

Each $e_{i,j}$ can be defined as $e_{i,j}$ = ($SrcID$, $DisID$, dataTransfereRate, TupleProcessingReq, TupleLentgth) where $SrcID$ represent the source task id (t_i node); $DisID$ represents the destination task id (t_j node); dataTransfereRate expresses the data transfer rate between t_i and t_j, TupleProcessingReq represent the processing requirement of coming tuples, and TupleLength represents the total length of the tuple.

Each task has one or more predecessors unless it is a start task as well as successors (one or more) unless if it is a finish task (see Figure 2 which depicts the start and finish tasks). Any task starts only after

[1]The iFogSim is an open source toolkit for modeling and simulating resource management approaches for IoT, Edge and Fog Computing https://github.com/Cloudslab/iFogSim

[2]Within the text, we used tasks, workflow activities and intermodules interchangeably for the same meaning.

all its predecessor tasks have completed, so its earliest start time is equal to the maximum finish time of all of its predecessors.

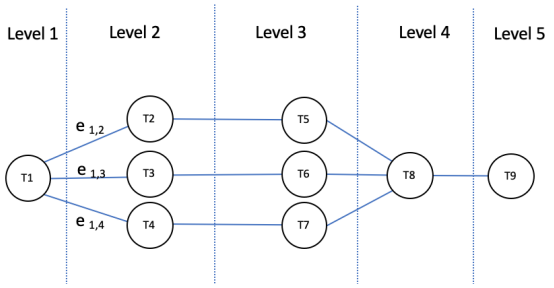

Figure 2: Tasks dependencies example for an IoT application

2.1.3 Processor Graph. Consider the presented topology in Figure 3. A processor graph is represented by a DAG, $PG = (P, D)$, where the set of nodes $P = p_1, p_2, p_3, ..., p_n$ represents n processors, a processor belongs to P can be a Cloud or an Edge resource. Between processors, there is a link distance d that connects them, for example, processors p_1 and p_2 are connected by $d_{1,2}$, so there is a set of links $d_{i,j}$ between any p_i and p_j belongs to P and $d_{i,j}$ belongs to D. We can define the distance links and processors as follows:

$$\forall p_i \wedge p_j \in P, \ \exists \ d_{i,j} = (p_i, p_j) \ \in D$$

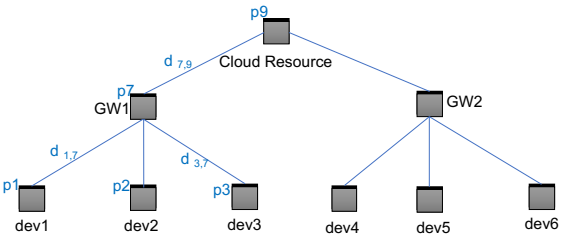

Figure 3: Tasks dependencies example for an IoT application

Each processor p_i can be defined as p_i =($pCapcty_i$, $upLnkLatency_i$, $pmLoad_i$). $pCapcty_i$ is considered to hold processing capacity of p_i, communication bandwidth is $upLnkLatency_i$ and $pmLoad_i$ is the current PM load.

2.1.4 Objectives. The objective is to propose a placement mechanism which aims to maximize the utilization of Edge resource as well as minimizing the cost and latency of an IoT application. In this work, the main information considered for task/workflow activity placement is as follows:

- Network Topology: available resources and their computation capabilities.
- Location of initiated requests or consumed services.
- Service Type: data storing and data filtering are services that require different computation/storage capabilities, therefore, approximate size of data (e.g. required Million Instructions Per Second (MIPS) required by an activity/task).
- Level of activity: number of hops that separate a task/activity from its starting point, in our case from the IoT devices that generate the data. It is essential to denote the dependency between tasks that helps to avoid assigning a task to the resource on the level lower than their predecessor, as well as to allow parallel processing for tasks that are within the same level.
- Quality of Service: knowing in advance the constraints on the offered services plays a role in selecting the type and layer of resource. In our work we considered minimizing the end-to-end response time by considering the deadline constraint for each involved task/activity.

To express our objective in maximizing the utilization of Edge resources, Each Task t_i can be deployed on R_{Cloud} or on R_{Edge}, however, deploying t_i on a Cloud resource or Edge resource is a binary variable. If t_i is deployed on Cloud, then 1 is assigned to $T_{i_{R_{Cloud}}}$ while 0 to $T_{i_{R_{Edge}}}$ and vice versa.

Each task is processed on either Edge or Cloud resources, thus, we try to maximize assigning tasks to Edge resources whenever it is appropriate which we represent mathematically as :

$$Maximize \sum_{i=0}^{i=n} T_{i_{R_{Edge}}} \tag{1}$$

The processing time of a task$_i$ on a p$_j$ is calculated as given in equation 2.

$$T_{Exec}(t_i, p'_j) = \frac{\alpha_{p_j}^{t_i} f_{t_i}(z_i)}{p'_j} + upLnkLatency_{p_j} \tag{2}$$

Here, $f_{t_i}(z_i)$ represents computation requirement for task t_i and z_i represents data into task t_i and $\alpha_{p_j}^{t_i}$ represent number of current modules running concurrently with t_i on node p'_j.

To calculate the CPU requirement for upcoming data/tuples for task t_i is given in equation 3 for each edge has t_i as its distention(i.e DisID for edge e is t_i). Where DTR represents the data transfer rate of an edge e and TPR represents the required processing capacity for each transferred tuple by edge e.

$$\sum_{edgee=0}^{edgee=y} DTR \times TPR \ \ \forall \ \ edge \ e \ has \ \ \ DisID \ = \ t_i \tag{3}$$

The total cost of of running t_i over node p_j is sum of memory cost $memCost_{Ti}$, communication cost $commCost_{Ti}$, storage cost $storgCost_{Ti}$ and node cost $nodeCost_{Ti}$ as given in equation 4. Each of these costs are explained in equation 5 - 8 respectively.

$$Cost(T_i) = memCost_{Ti} + commCost_{Ti} + storgCost_{Ti} + nodeCost_{Ti} \tag{4}$$

$$memCost_{Ti} = \frac{f_{t_i}(z_i)}{p_i} memoryCostUnit \tag{5}$$

$$commCost_{Ti} = sizeDataIn * commCostUnit \tag{6}$$

$$storgCost_{Ti} = DataToStore * StorageCostUnit \tag{7}$$

$$nodeCost_{Ti} = T_{iExec} * NodeCost \tag{8}$$

2.1.5 Proposed Algorithm. In the following section, we present the proposed algorithm for placing intermoduls among available resources with considering the following objectives and constraints:

Offline Integer Programming Formulation. Here we aim to minimize the cost of deploying intermodules on available resources and and the end to end response time. For the offline version of the intermodules placement problem, integer programming formulations are derived. These formulations are used to devise limits on the suggested approach. The main objective aims at minimizing the execution time while considering other constraints such as cost/budget constant. Table 1 summarizes the notations used in our formulation. Our main decision variables, denoted as x_{ij} is defined as follows:

$$obj = \textbf{Minimize} \sum_{i=0}^{i=n} Time_{task_i} \tag{9}$$

subject to

$$x_{ij} \in 0, 1, \forall i = 0, 1, ..., n; \forall j = 0, 1, ..., m \tag{10a}$$

$$\sum_{i=0}^{i=n} Cost_{T_i} <= CostConstraint \tag{10b}$$

$$\sum_{i=0}^{i=n} T_{i_{R_{Edge}}} > \sum_{i=0}^{i=n} T_{i_{R_{Cloud}}} \tag{10c}$$

$$\sum_{i,level=l} t_{ij} <= 1 \tag{10d}$$

$$\sum_{k=0,i,j}^{k=predecessorListSize} t_{kj_{tier}} <= t_{ij_{tier}} \tag{10e}$$

Constraint 1 enforces the binary nature of x_{ij}. Constraints 2 ensures that the cost of the deployment plan will not exceed the cost/budget limit. Constraints 3 ensures that number of assigned intermodules to Edge resources is greater than the number of assigned intermodules to Cloud resources. Constraint 4 ensures that no two intermodules from the same level are assigned to the same resource. Constraint 5, for all predecessors of a t_i t_i that has required processing capacity more than t_i itself, constraint 5 ensures that no intermodule t_i is assigned to a resource that is located in a tier lower than any of its predecessors.

In Algorithm 1, the Tasks and resources graphs are inputs. Line 6,7 and 8 are defining the associated level of each intermodules which calculated based on number of hops between the intermodule and the source of captured interesting event. Line 10 to 12 are to calculate the corresponding execution time of deploying t_i on resource p_j and then sort all resources based on their execution time. Starting with the least execution time do the constraints checking starting from Line 14 till 28: it check if p_j resource has predecessors to check that no predecessor of an intermodule t_i, is assigned to a resource that allocated in a layer higher than the current checked p_j. If there is a resource p_j, that has one of t_i's predecessor deployed on OR there is no predecessor of t_i are deployed on a resource with layer above the current p_j, then do checking on the associated constraints with the interemodule t_i by calling checkConstraintsConsistencyFunction (Line 21). If there is no resource p_j that matches the requirements and considered constraints then assign t_i to Cloud resource. In checkConstraintsConsistencyFunction t checks that a resource p_j can sustain the intermodule within its deadline constraint, not exceeding the budget and it is not running other tasks that have the same level as the coming intermodule and it is within the same region. If searching all resources within the same region does not satisfy the constraints, then check the other regions, if there is none then return false. If false is returned, then the task is assigned to Cloud resource if the cost constraint is not violated.

Table 1: Notations for the Offline Integer formulations and symbols used in the algorithm

Notations	Meanings
t_i	task/activity/intermodules i
p_j	a resource with processor pj
e_{ij}	dependency edge between two tasks t_i and t_j
l_{ij}	Link between two resources Pi and Pj
bw_{ij}	bandwidth of link l_{ij}
z_{ij}	data size transferred over edge dependency e_{ij}
dis_{ij}	distance between Pi and Pj
d_{ij}	Link delay of link l_{ij} between Pi and Pj
$p_{i_{capacity}}$	Computation capacity of resources Pi
$ReqPCapcty$	Represents requested processing capacity
$level$	Reflects how many hops this task is far from the starting point
$e_{i,j}$	Edge between t_i and t_j task nodes
$SrcID$	Represent the source id
$DisID$	Represents the destination id
$dataTransfereRate$	Expresses data transfer rate between t_i and t_j
$TupleProcessingReq$	Represent processing requirement of the tuples
$TupleLentgth$	Represents the length of tuples
$pCapcty_i$	Holds processing capacity of p_i in MIPS
$upLnkLatency_i$	Up Link Latency of p_i
$pLoad_i$	Current CPU load of p_i
$TupleLentgth$	Represents the length of tuples
$T_{i_{R_{Edge}}}$	Task t_i is running on Edge resource
$T_{i_{R_{Cloud}}}$	Task t_i is running on Cloud resource
$TimeConstraint_{t_{ij}}$	Time constraint of runing t_i
$Time_{t_{ij}}$	Execution Time of running t_i on resource p_j
$t_{ij_{tier}}$	Reflects the tier of resource p_j that run t_i
$predecessorListSize$	predecessors of t_i with $ReqPCapcty$ less than t_i's
$SortedResorces$	Sort resources based on their execution time of task t_i in ascending order

Algorithm 1: SLA aware algorithm for application modules placement cross layers

```
1  Input: TG=(T,E) ; PG=(P,D)
2  region =-1
3  found= false
4  Output: a mapped applications modules to available resources
5  Objectives: Minimize Cost and Minimize Application latency
6  foreach t_i in TG do
7      define a deadline d constraint
8      assign a level value to t_i
9      foreach resource p_j in PG do
10         calculate Time_{t_ij}
11         add(SortedResorces, p_j)
12     end
13     foreach resource p_j in SortedResorces do
14         if t_i has a predecessor then
15             foreach t_k in predecessor list of t_i do
16                 if t_k is already assigned to a PG resource then
17                     p_l = the PG resource that t_k is assigned to
18                     if p_llevel is greater than p_jlevel then
19                         break
20                 else
21                     call checkConstraintsConsistencyFunction
                       if checkConstraintsConsistencyFunction
                       then
22                         assign t_i to p_j update list of assigned
                           modules of p_j Found= true
23                     else
24                         continue
25                     end
26                 end
27             end
28     end
29     if not found then
30         Calculate the Cost of executing t_i on Cloud as in Eq. 4
31         if TotalCost+=cost of executing t_i on a Cloud resource is
           less than budget then
32             update TotalCost
33             assign t_i to a cloud resource
34     else
35         Log The Cost exceeds the allowed Budget and break
36     end
37 end
```

2.2 Time Complexity Analysis

We solved this problem as a context-aware approach, therefore, if the intermodule does not have predecessors, then in best case scenario, first search attempt returns a resource that matches the requirement for each intermodule, thus the time complexity is $\theta(n)$, where n represents number of intermodules. However, in worst case scenario when finding a resource that matches intermodules' constraints and performing it for each intermodule t_i, this requires checking all m resources in the resource list. Therefore, the time

Algorithm 2: Checking Budget Constraints Consistency after mapping tasks to resources

```
1  checkConstraintsConsistencyFunction
2  if region==-1 then
3      if t_iregion == p_jregion then
4          if that p_j does not have tasks from the same level of t_i
           then
5              Calculate the Execution Time of t_i on p_j by
               applying Equation 2
6              if the requested CPU Less than available CPU then
7                  //Check if the resource can sustain the place
                   module check if the time is less than or equal
                   to the associated deadline with t_i then
8                      return true
9          else
10             change the region value from -1 to another region
               not the same as region of t_i return false
11         end
12 else
13     Go To line 4
14 end
```

complexity is $\theta(nm)$ where m represent number of IoT devices (e.g., mobile) in the available resources in PG. Cases where an intermodule t_i has predecessors requires more time to avoid assigning an intermodule to a resource in a layer lower than its predecessors's resource. Thus we perform a checking step that iterates all of an intermodule's predecessors. As a result, in best case scenario, when an intermodules has only one predecessor and finds a resource that matches intermodules' constraints from the first attempt, the time complexity is $\theta(n)\theta(1)$ and considering only the upper bound means time complexity equals to $\theta(n)$. In worst case scenario, when an intermodule t_i has k predecessors and finding a resource that matches intermodules' constraints leads to checking all available m resources. In this case, time complexity can be calculated as $\theta(nmK)$, so time complexity can be in worst case scenario: $\theta(nm)+\theta(nmK)$, however, since k, which represents number of predecessors of an intermodule, is less than total number of intermodules n, therefore the Time complexity is $\theta(nm)$.

3 EVALUATION

To evaluate our proposed algorithm we run it using the iFogSim simulator. It simulates IoT applications where it can enable application modules to be allocated and scheduled among Fog and Cloud resources. There is a number of simulators for IoT, however, we chose iFogSim due to the fact that it is built based on CloudSim, which is popular among researchers for testing various strategies/algorithms. Furthermore, iFogSim has been used by a considerable number of published works such as [7], [6] and [1].

3.1 Use Case Studies

For evaluation purpose, we consider the following use case scenario:

3.1.1 Remote Health Monitoring Service (RHMS) Case Study 1. Consider a Remote Health Monitoring Service (RHMS) where patients

(elderly people, people with long term treatments,...) can subscribe to be monitored on a daily basis. Data is collected and filtered and if there is an interested pattern or event that match a threshold value, then data can be analysed on a small scale if it is related to a specific patient, however, in cases that require comparing coming data with historical data, or in cases that same events come from different subscribers such as fever signs, then the scenario can be considered as a large scale data analysis task that needs high computational power. Most interesting analysis results are then stored.

In this use case, the following workflow activities; data collection, data filtering, small-scale real-time data analysis, large-scale real-time data analysis, and storing data are considered as tasks t_1, t_2, t_3, t_4, t_5. There are sensors, attached to patients, hand-wrist, camera video for some patients and mobile accelerometers to capture activity patterns. They are connected to a smart phone as a gateway which is then connected to WiFi gateway. The WiFi gateway is connected to an Internet Service Provider, which in turn is connected to a Cloud datacenter. Description of intermodule of this case study is depicted in Table 2.

Table 2: Description of intermodule edges of Case Study 1

Tuple Type	CPU Length[MIPS]	N/W Length
Sensor	2500	500
COLLECTED_DATA	2500	500
FILTERED_DATA	1000	1000
ANALYSED_RESULT1	5000	2000
ANALYSED_RESULT2	10000	2000
COMMANDS	28	100

3.1.2 Intelligent Surveillance Case Study 2. Intelligent Surveillance application comprises five main processing modules: Motion Detector, Object Detector, Object Tracker, PTZ Control, and User Interface. This case study is one of the case studies mentioned in [3]. We have used it because we are planning to compare our results with the Edge-ward module placement algorithm presented in [3]. For more details of the case study and the algorithm, readers are advised to refer to [3]. A description of the intermodule of this case study is depicted in Table 5.

3.2 Physical network

For the case study, we have considered a physical topology with different types of Fog devices. Table 3 and Table 4 present the configuration of the used topology. This configuration is the same for both case studies, except the number of Fog devices. Case study 1 consists of four areas and each area has four fog devices; Case study 2 consists of two areas and each area has four fog devices.

Table 3: Associated Latency of network links

Source	Destination	Latency [ms]
IoT device	Smart Phone	1
smart phone	WiFi Gatway	2
WiFi Gatway	ISP Gatway	2
ISP Gatway	Cloud datacenter	100

Table 4: Configuration Description of Fog Devices

Device Type	CPU [GHz]	RAM [GB]	Power [W]
Smart Phone	1.6	1	87.53(M) 82.44(I)
WiFi Gatway	3	4	107.339(M) 83.433(I)
ISP Gatway	3	4	107.339(M) 83.433(I)
Cloud VM	3	4	107.339(M) 83.433(I)

3.3 Performance Evaluation Results

3.3.1 Analysis of Case Study1. We applied the proposed algorithm for Case Study 1 and compared the performance result with placing the intermodules on Cloud. Execution Time, Energy Consumption, Network Usage and Cost are the metrics that are Captured simulating the application and applying the proposed approach to place its intermodules using iFogSim. The following subsections compare the results of applying the proposed approach with applying the Cloud only approach when considering configurations listed in 2.

Execution Time. The overall execution time for Case Study 1 recorded less time when applying the proposed approach for task placement against the Cloud only approach. Figure 4.

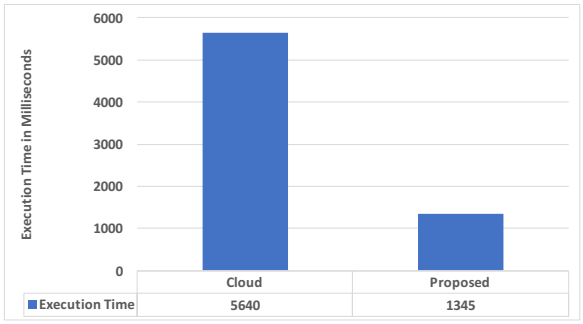

Figure 4: Time Execution of Case Study1

Networking Usage. As can be noted in Figure 5, there is not much difference in network usage, however, the proposed approach reflects slightly more network usage at the Edge and this is probably because of allocating most intermodules to the Edge resources.

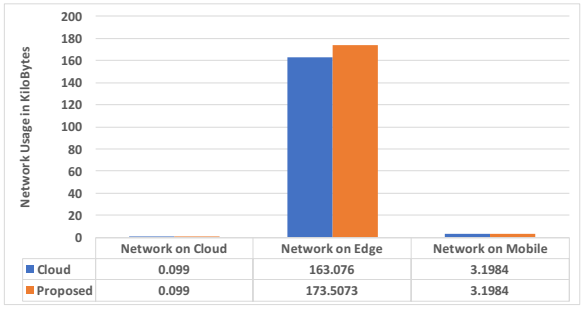

Figure 5: Networking Usage of Case Study1

Energy Consumption. In general, the Cloud-only approach as depicted in Figure 6 recorded a higher level of energy consumption on both Cloud and mobile layers compare with the proposed approach and on Edge layer, there is a slightly difference between both approaches which might have the same reason which is because the propose approach allocates more intermodules on Edge rather than Cloud.

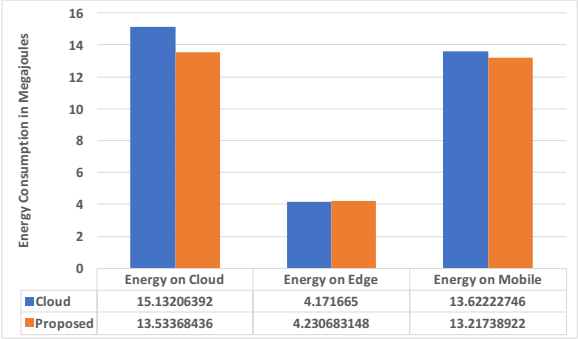

Figure 6: Energy Consumption of Case Study1

Cloud Cost. Figure 7 depicts the cost of implementing case study 1 when applying the Cloud-only approach, and our proposed approach. Cloud cost is higher with the cloud-only approach, while our approach is five times less costly than the Cloud-only approach because it only allocates "storing data" to the Cloud while the rest of deployed tasks are allocated to Edge resources.

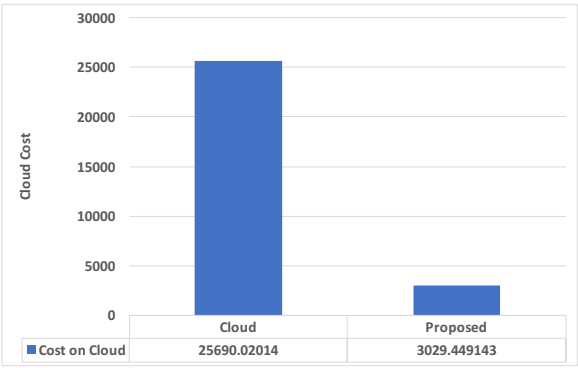

Figure 7: Cloud Cost of Case Study1

3.3.2 Analysis of Case Study2. We applied the proposed algorithm for case study 2 and compared the performance result with placing the intermodules on Cloud only as well as with the Edge-ward placement algorithm proposed in [3]. Execution Time, Control loop Latency, Energy Consumption, Network Usage and Cost are the metrics that have been Captured. The following sections describes the comparison results.

Execution Time. Execution time of all three approaches: Cloud-only, Edge-ward placement and our proposed placement approaches are reflected in Figure 8. Edge-ward placement has the highest level of time execution and the proposed approach has the least execution time.

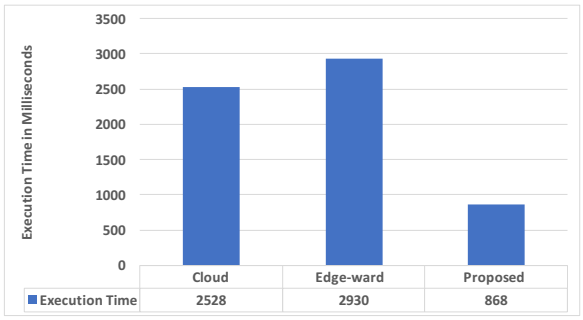

Figure 8: Time Execution of Case Study2

Networking Usage. Networking usage is described for the three tiers: IoT devices (mobiles), Edge resources (WiFi Gateways) and Cloud. Edge-ward placement reflects the least network usage among all proposed approaches for network usage on Edge and mobile tiers. The proposed approach has recorded no network usage on Cloud, but has recorded high level of network usage on the Edge tier. This seems to be because of placing intermodules on different Edge resources since it considered parallel processing for independent tasks as well as following a greedy approach might affect the overall optimality of resource allocation mechanism.

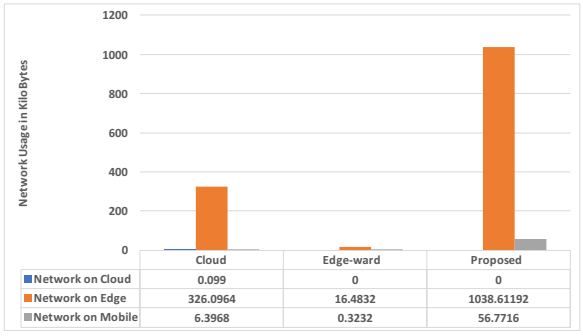

Figure 9: Networking Usage of Case Study2

Energy Consumption. Figure 10, shows the energy consumption for the three approaches; the Cloud-only approach demonstrates the highest value for energy consumption on the Cloud tier, however, on Edge and mobile/IoT devices all approaches reflect a similar level of energy consumption with a slightly low level for the proposed approach on mobile layer.

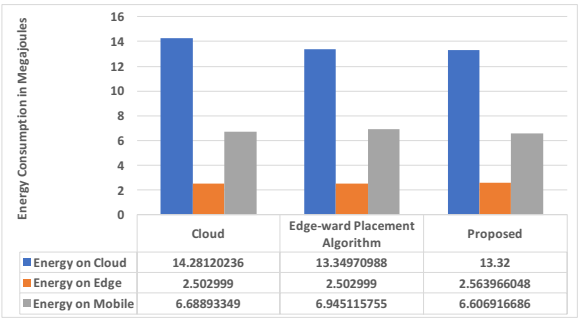

Figure 10: Energy Consumption of Case Study2

Cloud Cost. Figure 11 shows the Cloud cost. Since the proposed approach placed all tasks on Edge, therefore Cloud cost is none while Edge-ward placement approach reflect lower cost than Cloud approach.

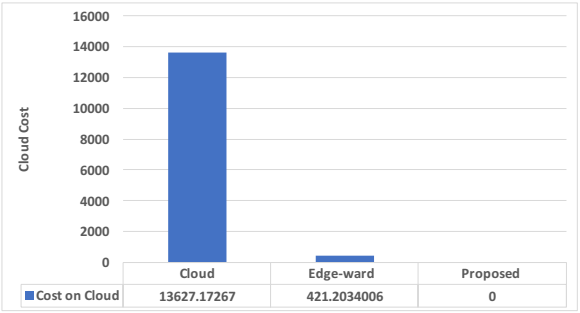

Figure 11: Cloud Cost of Case Study2

4 DISCUSSION

We have applied our heuristic algorithm to decentralize task placement in a cooperative way between Edge and Cloud resources. We have considered an RHMS as Case Study and compared it with the Cloud-only approach (an approach where tasks are placed only on the Cloud). The proposed approach demonstrates lower execution time, control loop, cost, network usage and energy consumption. Furthermore, we considered comparing our approach with a built-in use case in iFogSim. Thus we compared the results with the Edge-ward placement algorithm applied to Case Study 2. In general our proposed approach shows better results, as has been presented in the previous section.

5 RELATED WORK

Ref [7] offers a new, multi-layered, IoT-based fog computing architecture. In particular, it has developed a service placement mechanism that optimizes service decentralization in the fog landscape by using context-aware information like location, response time and service resource consumption. The approach is being used in an optimal way to increase the efficiency of IoT services in terms of response time, energy and cost reduction. However, this work considers tasks to be independent, and deadline constraints to be at application level only (i.e., there is no deadline constraints for each task involved). Ref [6] iFogStor seeks to take advantage of Fog nodes' heterogeneity and location to reduce the overall latency of storage and data retrieval in the fog. They have formulated the data placement problem as a GAP (Generalized Assignment Problem) and proposed two solutions: 1) an exact solution using integer programming and 2) a geographically based solution to decrease solving time. However, its focus is related to storing data at the Edge to ease data retrieving. In Ref [1] the authors propose an infrastructure and IoT application model as well as a placement approach taking into account the power consumption of a system and minimizing delay violations using a Discrete Particles Swarm Optimization (DPSO) algorithm. iFogSim simulator is used to evaluate the proposed approach. However, the authors of this approach consider the effect of their algorithm on energy consumption and delay only. Ref [4] suggests a smart decision-making system to assign tasks locally. The remaining tasks in the network or the Fog / Cloud are transferred to peer nodes. However, their approach only allows for task processing to be executed in sequential order (there is no parallel execution capability).

.

6 CONCLUSION AND FUTURE WORK

Due to the limited computational and resource capabilities of IoT nodes, tasks can be allocated to Edge or Cloud resources taking into account a number of possibilities such as: task constraints, node load and computing capability. We suggested a heuristic algorithm for allocating tasks among the available resources. The allocation algorithm takes into account factors such as related time limits for each task, location, and budget constraints. We utilized iFogSim to evaluate the proposed approach for two use-case studies. The performance analysis demonstrates that the suggested algorithm has minimized costs, execution time, control loop delay, networking and cloud power usage compared to cloud-only and edge-ward positioning methods. For Future work, we will carry an evaluation of the proposed approach on real systems.

REFERENCES

[1] T. Djemai, P. Stolf, T. Monteil, and J. Pierson. 2019. A Discrete Particle Swarm Optimization Approach for Energy-Efficient IoT Services Placement Over Fog Infrastructures. In *2019 18th International Symposium on Parallel and Distributed Computing (ISPDC)*. 32–40. https://doi.org/10.1109/ISPDC.2019.00020

[2] Ioan Dumitrache, Ioan Stefan Sacala, Mihnea Alexandru Moisescu, and Simona Iuliana Caramihai. 2017. A conceptual framework for modeling and design of Cyber-Physical Systems. *Studies in Informatics and Control* 26, 3 (2017), 325–334.

[3] Harshit Gupta, Amir Vahid Dastjerdi, Soumya K. Ghosh, and Rajkumar Buyya. 2017. iFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, Edge and Fog computing environments. *Software: Practice and Experience* 47, 9 (jun 2017), 1275–1296. https://doi.org/10.1002/spe.2509

[4] Kostas Kolomvatsos and Christos Anagnostopoulos. 2019. Multi-criteria optimal task allocation at the edge. *Future Generation Computer Systems* 93 (2019), 358 – 372. https://doi.org/10.1016/j.future.2018.10.051

[5] Kostas Kolomvatsos and Christos Anagnostopoulos. 2019. Multi-criteria optimal task allocation at the edge. *Future Generation Computer Systems* 93 (2019), 358 – 372. https://doi.org/10.1016/j.future.2018.10.051

[6] M. I. Naas, J. Boukhobza, P. Raipin Parvedy, and L. Lemarchand. 2018. An Extension to iFogSim to Enable the Design of Data Placement Strategies. In *2018 IEEE 2nd International Conference on Fog and Edge Computing (ICFEC)*. 1–8. https://doi.org/10.1109/CFEC.2018.8358724

[7] Minh Quang Tran, Duy Tai Nguyen, Van An Le, Hai Nguyen, and Tran Vu Pham. 2019. Task Placement on Fog Computing Made Efficient for IoT Application Provision. *Wireless Communications and Mobile Computing* 2019 (01 2019), 1–17. https://doi.org/10.1155/2019/6215454