An Approach to QoS-based Task Distribution in Edge Computing Networks for IoT Applications

Yaozhong Song, Stephen S. Yau, Ruozhou Yu, Xiang Zhang and Guoliang Xue
School of Computing, Informatics, and Decision Systems Engineering
Arizona State University
Tempe, Arizona, USA
{*ysong75, yau, ruozhouy, xzhan229 and xue*}*@asu.edu*

***Abstract*—Internet of Things (IoT) is emerging as part of the infrastructures for advancing a large variety of applications involving connection of many intelligent devices, leading to smart communities. Due to the severe limitation on the computing resources of IoT devices, it is common to offload tasks of various applications requiring substantial computing resources to computing systems with sufficient computing resources, such as servers, cloud systems, and/or data centers for processing. However, the offloading method suffers from the difficulties of high latency and network congestion in the IoT infrastructures. Recently edge computing has emerged to reduce the negative impacts of these difficulties. Yet, edge computing has its drawbacks, such as the limited computing resources of some edge computing devices and the unbalanced load among these devices. In order to effectively explore the potential of edge computing to support IoT applications, it is necessary to have efficient task management in edge computing networks. In this paper, an approach is presented to periodically distributing incoming tasks in the edge computing network so that the number of tasks, which can be processed in the edge computing network, is increased, and the quality-of-service (QoS) requirements of the tasks completed in the edge computing network are satisfied. Simulation results are presented to show the improvement of using this approach on the increase of the number of tasks to be completed in the edge computing network.**

***Keywords*-edge computing; task distribution; quality-of-service; IoT applications**

I. Introduction

With its capability of interconnecting a large number of various intelligent devices across wide geographical areas, Internet of Thins (IoT) has become part of the infrastructures for many advanced applications leading to smart cities and other connected communities. This trend has inspired the development of a large variety of applications with connected intelligent devices. However, despite the rapid progress of IoT-related technologies, a major bottleneck of IoT applications is the limited computing resources available in each IoT device, including CPU, storage, etc. The situation is further exacerbated by the proliferating IoT applications, many of which involve resource-intensive operations, including big data analysis and real-time processing applications.

A common way to overcome the small computing resources of IoT devices in various applications is to offload some tasks of IoT applications to computing systems with sufficient computing resources, such as servers, cloud systems, and/or data centers for processing [1]–[3]. Task offloading improves the performance of IoT applications and reduces the energy consumption in IoT devices [1], [2], [4]. However, using the offloading method often causes additional overhead by data transmission over wide area networks, which would increase the latency of IoT applications and network congestion, especially in those applications involving resource-intensive operations [5], [6].

The disadvantages of task offloading to remote computing systems motivate the recent development of edge computing [5], [7], [8], which uses network devices with larger computing resources, such as cloudlets, computing-enabled switches, and computing-enabled base stations at the network edge, to support IoT applications. Due to the proximity to IoT devices, edge computing networks can decrease the latency of IoT applications and make larger bandwidth available for interconnecting IoT devices. Due to these benefits, task offloading to edge computing networks has attracted much attention of researchers in both academia and industry [4], [6], [8]–[11].

Yet, task offloading to edge computing networks may cause unbalanced workload among edge computing devices, which may result in congestion in edge computing networks. There are two reasons for this problem: One is that the resources of some edge computing devices are too small for processing tasks with resource-intensive operations. The other is that the IoT applications tend to be more sophisticated and require more resource-intensive operations. Enabling resource sharing and task offloading among edge computing devices, have been used for addressing this problem [6], [11]–[16].

In this paper, we will present an effective approach to periodically distributing incoming tasks in the edge computing network so that the number of tasks can be processed in the edge computing network is increased. The task distribution generated by our approach is ensured to satisfy all the accommodated tasks' quality-of-service (QoS) specifications required in the IoT applications. Simulation results are presented to show the improvement of using our approach.

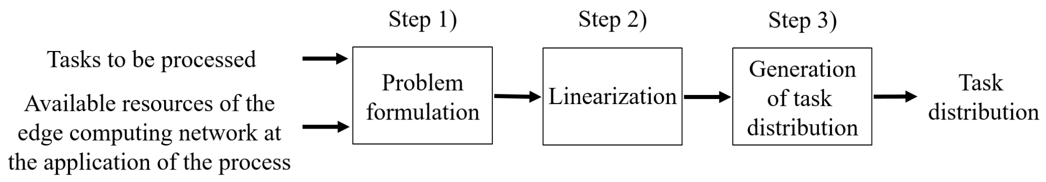
Step 1)
Step 2)
Step 3)
Tasks to be processed
Available resources of the edge computing network at the application of the process
Problem formulation
Linearization
Generation of task distribution
Task distribution

Figure 1: The task distribution process of our approach.

This paper is organized as follows. In Section II, we will discuss the current state of the art in this area. In Section III, we will present our overall approach. In Section IV, we will formulate the problem of task distribution in edge computing networks. In Section V, we will present how to linearize the formulation of the task distribution problem to reduce its computation complexity. In Section VI, we will show how to generate the task distribution to increase the number of tasks accommodated in the edge computing network. In Section VII, we will present our simulation results to show the improvement of using our approach. In Section VIII, we will conclude this paper and discuss the future work.

II. Current State of the Art

Since the concept of edge computing was first introduced [5], [8], various terms such as "cloudlet" [5], "mobile edge computing" [7], and "fog computing" [8], have been used to refer to edge computing. Task distribution in edge computing networks has been studied in [6], [11]–[14].

Zeng, Gu, Guo, Cheng, and Yu, [6], presented a joint optimization of task scheduling and image placement to minimize the average task completion time for fog computing supported software-defined embedded system. Gu, Zeng, Guo, Barnawi, and Xiang, [11], presented a joint optimization of user association, virtual machine deployment, and task distribution to minimize the cost of operation for fog computing supported medical cyber-physical system. In both papers, the authors did not consider the resource limitations of edge computing networks, or the impacts of network bandwidth sharing and the constraint of security on the performance of task distribution.

Oueis, Strinati, and Barbarossa, [12], presented an approach to load distribution for small cell cloud computing and a joint optimization of computational and radio resources to minimize the power consumption for dynamically formed small cell cloud systems. Sardellitti, Barbarossa, and Scutari, [13], presented a similar approach to minimize the energy consumption at the mobile terminal side. Sardellitti, Scutari, and Barbarossa, [14], presented a joint optimization of radio and computational resources for mobile edge computing. In all the three papers, the authors did not consider the impacts of network bandwidth sharing and the constraint of security on the performance of task distribution and load balancing.

III. Our Approach

In this section, we will present our overall approach to task distribution in edge computing networks. The following assumptions are made in our approach:

A-1: The tasks to be processed in the edge computing network do not have different priorities.

A-2: The tasks to be processed in the edge computing network do not have dependency relationship among them.

A-3: The operating systems of all the virtual machines (VMs) in the edge computing network are of the same type, such as Linux or Windows [5], so that the incoming tasks can be processed on any VM as long as the VM has sufficient resources.

It is noted that assumptions A-1 and A-3 are made for the sake of simplicity, and we plan to remove these two assumptions in our future work. Assumption A-2 is normally valid if the incoming tasks are from multiple IoT applications and/or various IoT devices.

As incoming tasks to the edge computing network are continuously sent from IoT devices, the task distribution process needs to be applied periodically. Let σ be the time interval between two consecutive applications of task distribution process. The input set of tasks to the next task distribution process consists of all the incoming tasks in the current time interval and those tasks which were not distributed in the last task distribution process.

Because incoming tasks are dynamically changing in terms of both arrival rate and specifications (workload, data size, deadline, etc.), σ should be dynamically adjusted to maintain acceptable performance of the task distribution process at each application. As the performance of the task distribution process is measured by the number of tasks being distributed in the edge computing network, σ should be adjusted based on the number of distributed tasks. If the number of distributed tasks falls below a preset threshold, for example, 30, σ should be reduced so that the task distribution process will be applied more frequently.

The task distribution process can be depicted in Figure 1, and summarized as follows:

Step 1) Formulate the task distribution problem in the edge computing network, including the given QoS requirements of tasks and resource constraints of the edge computing network. The details of this step are presented in Section IV.

Step 2) Linearize the formulation and derive a mixed-integer linear program for generating the task distribution in the edge computing network. The details of this step are presented in Section V.

Step 3) Generate the task distribution using our algorithm, which is based on relaxation, rounding, and validation. The details of this step are presented in Section VI.

IV. Task Distribution Problem Formulation

To formulate the task distribution problem in edge computing networks, we first describe the edge computing network and the incoming tasks. We will then generate the objective function with various constraints of edge computing networks and QoS requirements of tasks.

An edge computing network is built on the traditional network infrastructure, where devices such as cellular base stations, access points, routers, and switches, are associated with certain amount of computing and storage resources [6], [7], [11], [17], [18]. An edge computing network can be represented by a directed graph $G = (V, E)$, where $V = \{v_0, \ldots, v_i, \ldots, v_{m-1}\}$ is the set of edge nodes, and $E = \{e_0, \ldots, e_j, \ldots, e_{n-1}\}$ is the set of edge links. The set of incoming links of node v_i is denoted by $in(i)$, and the set of outgoing links of node v_i is denoted by $out(i)$. Each edge node runs a number of VMs, and each VM performs one task at a time. Let h_i be the number of available VMs on edge node v_i at the time of applying the task distribution process. Each VM is granted with computing rate r_i. Let s_i be the available storage of edge node v_i at the time of applying the task distribution process. Let p_i be the lower bound of the security level that each VM in v_i, has because of the security mechanisms incorporated in the VM, including the encryption for secure transmission of data to other nodes in the edge computing network [19], [20]. Let b_j be the available bandwidth of link e_j of the edge computing network at the time of applying the task distribution process.

Let $T = \{t_0, \ldots, t_k, \ldots, t_{\psi-1}\}$ be the set of tasks to be processed at the time of applying the task distribution process. A task is denoted by $t_k = (a_k, w_k, d_k, s'_k, p'_k, \delta_k)$, where a_k is the access node of t_k, w_k is the computation workload [1] of t_k, d_k is the data size of t_k, s'_k is the storage requirement of t_k, p'_k is the upper bound of the required security level of t_k, and δ_k is the completion deadline of t_k.

As all the incoming tasks are initially sent from IoT devices to their connected access nodes, we further denote b_{a_k} as the access bandwidth for t_k, which is the bandwidth between the IoT device that sends t_k and its access point a_k.

A. Constraints of Task Distribution

Before discussing these constraints, we need to identify the variables for task distribution due to the characteristics of the task distribution problem. Let $x_{k,i} = 1$ if t_k has been decided to be distributed to v_i, and $x_{k,i} = 0$ if t_k has been decided not to be distributed to v_i. Let $f_{k,j}, 0 \leq f_{k,j} \leq b_j$, be the required bandwidth on e_j for the flow of t_k to pass through e_j.

The constraints of the task distribution problem are given below:

C-1: Task assignment constraint:

Each task t_k can be assigned to at most one edge node, and must be executed and completed on the assigned edge node. That is,

$$\sum_{i=0}^{m-1} x_{k,i} \leq 1. \tag{1}$$

C-2: Node storage constraint:

Each edge node v_i, must have sufficient storage to store the data of all the tasks distributed to v_i. That is,

$$\sum_{k=0}^{\psi-1} x_{k,i} s'_k \leq s_i. \tag{2}$$

C-3: Security constraint:

The VMs on each node v_i must be sufficiently secure in order to satisfy the specified security requirement of t_k. That is,

$$x_{k,i} p'_k \leq p_i. \tag{3}$$

C-4: Node sharing constraint:

Each node v_i can only host at most h_i tasks due to its available VMs. That is,

$$\sum_{k=0}^{\psi-1} x_{k,i} \leq h_i. \tag{4}$$

C-5: Task completion time duration constraint:

This constraint is that each accommodated task t_k must be completed within the deadline specified by its IoT application.

The completion time duration of t_k on an edge node v_i consists of two components: the t_k's data transmission time duration between t_k's IoT device and its execution node v_i, and the execution time duration of t_k on node v_i.

The first component consists of the t_k's data transmission time duration between t_k's IoT device and its access node a_k, and the data transmission time duration between t_k's access node a_k and its execution node v_i. Recall that we use b_{a_k} to denote the bandwidth between t_k's IoT device and its access node a_k. Let $b_{v_i}^{a_k}$ be the bandwidth for task t_k to transmit its data from its access node a_k to its execution node v_i, then the bandwidth between the IoT device and the execution node v_i is $\min\{b_{a_k}, b_{v_i}^{a_k}\}$. The data transmission time duration $\tau_k^{tr}(v_i)$ between t_k's IoT device and the execution node is then given by $d_k / \min\{b_{a_k}, b_{v_i}^{a_k}\}$. The second component of the task completion time duration of t_k, is the execution time duration $\tau_k^{ex}(v_i)$ of t_k on v_i, which is given by w_k / r_i.

The constraint on the completion time duration τ_k for each task t_k is given as follows,

$$\begin{aligned}\tau_k &= \sum_{i=0}^{m-1} \tau_k(v_i)\cdot x_{k,i} \\ &= \sum_{i=0}^{m-1} \left[\tau_k^{tr}(v_i)+\tau_k^{ex}(v_i)\right]\cdot x_{k,i} \\ &= \sum_{i=0}^{m-1} \left[\frac{d_k}{\min\{b_{a_k}, b_{v_i}^{a_k}\}} + \frac{w_k}{r_i}\right]\cdot x_{k,i} \\ &\le \delta_k. \end{aligned} \tag{5}$$

C-6: Link bandwidth sharing constraints:

There are two constraints for link bandwidth sharing by multiple tasks in the edge computing network [21]. The first constraint C-6.1 concerns the difference between the total bandwidth of all the incoming and all the outgoing links used by the flow of task t_k of each node in the edge computing network. The second constraint C-6.2 concerns the link capacity constraint of each link in the edge computing network.

C-6.1: Link bandwidth conservation constraint:

For a node v_i, which is neither the access node, nor the execution node, of t_k, the sum of the required bandwidth on all the incoming links to v_i for the flow of t_k must be the same as the sum of the required bandwidth on all the outgoing links from v_i for the flow of t_k.

For a node v_i, which is the execution node of t_k, the sum of the required bandwidth for the flow of t_k on all the incoming links to v_i must be greater than the sum of the required bandwidth for the flow of t_k on all the outgoing links from v_i by $b_{v_i}^{a_k}$, which is the bandwidth for task t_k to transmit its data from its access node a_k to its execution node v_i.

Hence, the flow conservation constraint is expressed as follows,

$$\sum_{e_j\in in(i)} f_{k,j} - \sum_{e_j\in out(i)} f_{k,j} = b_{v_i}^{a_k}\cdot x_{k,i}. \tag{6}$$

C-6.2: Link bandwidth constraint:

For each link e_j, the sum of the required bandwidth for the flows of all the tasks to pass through e_j must be no greater than the capacity of link e_j. That is,

$$\sum_{k=0}^{\psi-1} f_{k,j} \le b_j. \tag{7}$$

B. The Task Distribution Program

Note that due to Constraint (1), some tasks may not be distributed to any edge node for execution. In practice, not all tasks can be accommodated for processing in the edge computing network because of the limited resources in edge computing network. The objective of the task distribution generated by our approach is to increase the number of accommodated tasks in the edge computing network. Hence, the task distribution problem can be formulated as follows:

$$\boxed{\begin{aligned} &\max \quad \sum_{k=0}^{\psi-1}\sum_{i=0}^{m-1} x_{k,i} \\ &\text{s.t.} \quad (1)\text{–}(7). \end{aligned}} \tag{8}$$

Note that (8) is a mixed-integer nonlinear program (MINLP), and is NP-hard because of the following reason: Consider a special case of the task distribution program, which assumes that all VMs in the edge computing network finish their assigned tasks instantly and satisfy the maximum security level, all nodes have the same storage available, and all links have infinite bandwidth, is equivalent to the decision version of Bin Packing problem, which is NP-hard [22]. Because this special case is NP-hard, (8) is NP-hard. Due to the NP-hard nature of (8), it is impossible to generate the optimal solution for (8) in polynomial time, unless P=NP [22].

In order to achieve the goal of our task distribution approach, we will use a mixed-integer linear program (MILP) by simplifying Constraint (5) and (6) in (8).

V. Linearization of the Task Distribution Program

The nonlinearity of (8) is due to Constraints (5) and (6), which are both nonlinear because they involve both $x_{k,i}$ and $b_{v_i}^{a_k}$. By assuming that the execution of all accommodated tasks will be completed exactly at their respective deadlines, $b_{v_i}^{a_k}$ can be replaced by a constant $\hat{b}_{v_i}^{a_k}$, which will be defined in (11). This assumption does not increase any limitations on the services provided by the edge computing network since each accommodated task still satisfies its completion time duration requirement. The details of the linearization are explained as follows:

Given an execution node v_i, Constraint (5) now becomes

$$\frac{d_k}{\min\{b_{a_k}, b_{v_i}^{a_k}\}} + \frac{w_k}{r_i} \le \delta_k. \tag{9}$$

Therefore, we can solve (9), and have the following inequality:

$$\min\{b_{a_k}, b_{v_i}^{a_k}\} \ge \hat{b}_{v_i}^{a_k}, \tag{10}$$

where

$$\hat{b}_{v_i}^{a_k} = \frac{d_k}{\delta_k - \frac{w_k}{r_i}}. \tag{11}$$

In other words, if $b_{a_k} < \hat{b}_{v_i}^{a_k}$ or $\hat{b}_{v_i}^{a_k} < 0$, then node v_i cannot be assigned as t_k's execution node, because Constraint (5) is not satisfied. On the other hand, if $0 < \hat{b}_{v_i}^{a_k} \le b_{a_k}$, then Constraint (5) is equivalent to the following

$$b_{v_i}^{a_k} \ge \hat{b}_{v_i}^{a_k}. \tag{12}$$

Note that letting equality hold for (12), which means that all tasks are completed exactly on their respective

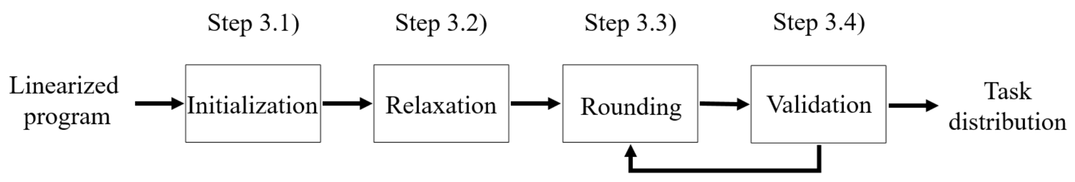

Figure 2: Step 3 of the task distribution process.

deadlines, has no impact on the optimal solution set, since bandwidth allocation may only reduce on each link to a lower bandwidth requirement of $\hat{b}_{v_i}^{a_k}$. Hence the variable $b_{v_i}^{a_k}$ is now resolved to a constant number $\hat{b}_{v_i}^{a_k}$ in Constraint (6).

To sum up, the linearization consists of two components: first, for each task t_k, remove all potential execution nodes $v_i \in V$ where $b_{a_k} < \hat{b}_{v_i}^{a_k}$ or $\hat{b}_{v_i}^{a_k} < 0$; second, replace Constraints (5) and (6) with the following:

$$\sum_{e_j \in in(i)} f_{k,j} - \sum_{e_j \in out(i)} f_{k,j} = \hat{b}_{v_i}^{a_k} \cdot x_{k,i} \tag{13}$$

for $\forall v_i \in V \setminus \{a_k\}$

The resultant program is now an MILP:

$$\begin{aligned} \max \quad & \sum_{k=0}^{\psi-1} \sum_{i=0}^{m-1} x_{k,i} \\ \text{s.t.} \quad & (1)\text{–}(4),\ (7),\ (13). \end{aligned} \tag{14}$$

VI. Generation of Task Distribution

We generate the task distribution by solving the MILP (14), which is the Step 3 of our overall task distribution process. It has four sub-steps, which can be depicted in Figure 2 and presented as follows.

Step 3.1) Initiate this sub-step by setting both the set of accommodated tasks, T_{Acc}, and the set of failed tasks, T_{Fail}, as empty sets.

Step 3.2) Relax the MILP (14) by replacing the binary variables $x_{k,i} \in \{0,1\}$ with new variables $\bar{x}_{k,i} \in [0,1]$, that is each $\bar{x}_{k,i}$ has a value between 0 and 1, inclusive. The resultant program with relaxed variables $\bar{x}_{k,i}$ is called a linear program relaxation (LPR) of the original program (14).

Step 3.3) Solve the LPR of (14) over T and T_{Acc}, and obtain the values of the relaxed variables $\bar{x}_{k,i}$. Find the variable $\bar{x}_{k,i}$ with the largest value, round off its value to 1, and all other variables $\bar{x}_{k,i'} (v_i' \neq v_i)$ for the same task t_k to 0. Then go to Step 3.4).

Step 3.4) Update T_{Acc} to $T_{Acc} \cup t_k$. Generate a program for the data routing in the edge computing network by replacing the objective function in (14) with a new objective function, which is to minimize the sum of the flows of all tasks in the edge computing network. That is,

$$\begin{aligned} \min \quad & \sum_{k=0}^{\psi-1} \sum_{j=0}^{n-1} f_{k,j} \\ \text{s.t.} \quad & (1)\text{–}(4),\ (7),\ (13). \end{aligned} \tag{15}$$

Validate the rounding in Step 3.3) by checking if (15) with the updated accommodated task set, has any feasible solution. If there is a feasible solution, the rounding of the values of $\bar{x}_{k,i}$ and $\bar{x}_{k,i'}$ in Step 3.3) is validated, and move t_k from T to T_{Acc}. In addition, put the tasks with all their variables $\bar{x}_{k,i}$ marked as failed the validation to T_{Fail}. Then go to Step 3.3) if T is not empty. On the other hand, if there

Algorithm 1: Relaxation, Rounding and Validation (RRV)

```
Input: The edge computing network $G = (V, E)$, set
       of tasks to be processed $T$
Output: Set of accommodated tasks $T_{Acc}$, set of failed
        tasks $T_{Fail}$, task distribution $\{x_{k,i}\}$, flows of
        tasks $\{f_{k,j}\}$
1  $T_{Acc} \leftarrow \emptyset$, $T_{Fail} \leftarrow \emptyset$;
2  while $T \neq \emptyset$ do
3      Solve LPR of (14) and obtain $\{\bar{x}_{k,i}\}$ and $\{f_{k,j}\}$;
4      $X \leftarrow \{\bar{x}_{k,i} \,|\, t_k \in T, v_i \in V\}$;
5      for each $\bar{x}_{k,i} \in X$ in descending order of value do
6          Assign $\bar{x}_{k,i} \leftarrow 1$, and $\bar{x}_{k,i'} \leftarrow 0$ for $\forall v_i' \neq v_i$;
7          Solve (15) for $T_{Acc} \cup t_k$ with fixed task
           assignment;
8          if (15) has a feasible solution then
9              Fix task assignment for $t_k$;
10             $T_{Acc} \leftarrow T_{Acc} \cup \{t_k\}$, $T \leftarrow T \setminus \{t_k\}$;
11             break;
12         else
13             Un-assign $\bar{x}_{k,i}$ and $\bar{x}_{k,i'}$ values in Line 6;
14             $X \leftarrow X \setminus \{x_{k,i}\}$;
15         end
16     end
17     for each $t_k \in T$ where $x_{k,i} \notin X$ for $\forall v_i \in V$ do
18         $T_{Fail} \leftarrow T_{Fail} \cup \{t_k\}$, $T \leftarrow T \setminus \{t_k\}$;
19     end
20 end
21 return $T_{Acc}, T_{Fail}, \{x_{k,i}\}, \{f_{k,j}\}$.
```

is no feasible solution for (15), the rounding of the values of $\bar{x}_{k,i}$ and $\bar{x}_{k,i'}$ in Step 3.3) will be undone, variable $\bar{x}_{k,i}$ will be marked as failed the validation. Then go back to Step 3.3) to round off the variable with the next largest value.

This section is implemented by Algorithm 1, which terminates in polynomial time. To see this, observe that in each outer iteration(Line 2–20), either a task is added to T_{Acc} at Line 10, or all the rest tasks in T fail the validation and be moved to T_{Fail} at Line 18. The size of T is reduced by at least 1 in each outer iteration. Hence the outer loop terminates in at most ψ iterations (ψ is the number of tasks initially in T). The inner loop (Line 5–16) iterates over all task distribution variables, which is at most of the size $O(\psi m)$, where m is the number of nodes in V. Since LPs in each iteration can be solved in polynomial time [23], the algorithm terminates in polynomial time as well.

VII. Evaluation

In this section, we will present the evaluation of our approach to task distribution in the edge computing network. The simulation results show the improvement of our approach on the number of tasks accommodated in the edge computing network, compared to the two baseline approaches: the local execution approach (Local) and random distribution approach (Random). The Local approach does not involve task distribution in the edge computing network, and each task is executed locally on its access node. The Random approach randomly selects an eligible node for each task as its execution node, where an eligible node for a task is a node that can satisfy all the QoS requirements of the task. We use these two baseline approaches, as there are no existing approaches to the same task distribution problem.

A. Simulation Environment

We used Waxman model [24] to generate network topologies for edge computing networks. The Waxman model defines the probability of the connection of any two nodes u and v in a network, based on four factors: the distance between u and v, the maximum distance between any two nodes in the network, and two user defined parameter α and β, where α and β are in the range of $(0, 1]$. Larger values of β result in networks with higher link densities, while small values of α increase the density of short links relative to that of longer ones. The formulation of Waxman model is :

$$p(u, v) = \beta \cdot \exp(-d(u, v)/(L \cdot \alpha)), \tag{16}$$

where $p(u, v)$ is the probability of the connection of nodes u and v, and $d(u, v)$ is the distance between u and v.

In our simulations, we changed the value of α and β to generate different network topologies for edge computing networks. In the first two simulations, we set the number of edge nodes to 15, and the values of both α and β to 0.8, which generated a network with 80 links. Our third simulation shows that no matter what values of α and β are selected, our approach always outperforms the two baseline approaches. In the third simulation, we set the number of edge nodes to 15, and changed the values of both α and β from 0.1 to 1, with an increment of 0.1, which generated networks with 28, 28, 36, 38, 38, 54, 66, 80, 96, and 116 links respectively.

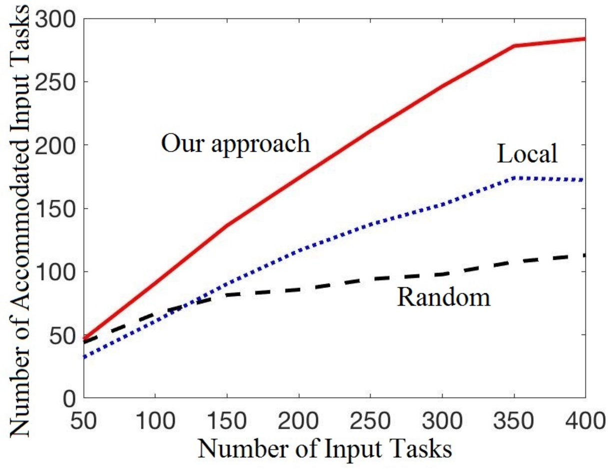

Figure 3: Comparison of the average number of the accommodated input tasks of our approach with Local and Random approaches on varying number of input tasks.

We ran each of the following three simulations 20 times and obtained the average of the number of accommodated tasks. We selected the value of 20 because the results of the average of 20 instances of the simulations are stable enough to make comparison with all the three approaches. This is shown in Figures 3 to 5 that all the curves in these figures are highly smooth.

We compared all the three approaches using the number of accommodated input tasks, which is a direct reflection of the performance of different approaches.

B. Simulation with Varying Number of Tasks

In this simulation, we changed the number of input tasks from 50 to 400, with an increment of 50, to examine the effect of the number of input tasks. Figure 3 shows the simulation results. It shows that on the average, our approach accommodates 54% more input task than the number of input tasks than the Local approach and 100% more input tasks than the Random approach. We have also found that when the number of tasks increases, our approach performs even better than the baseline approaches. The reason is that our approach can balance the workload among edge nodes and utilize resources in the edge computing network more effectively than the baseline approaches. The overall performance of the Random approach is worse than the Local approach because the Random approach suffers from the overhead of data transmission in the edge computing network.

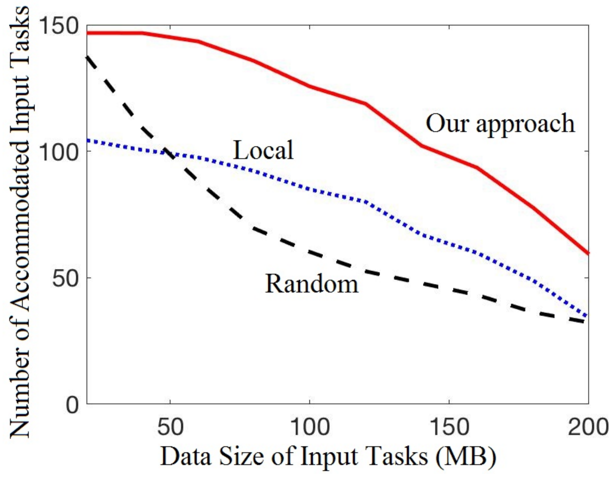

Figure 4: Comparison of the average number of the accommodated input tasks of our approach with Local and Random approaches on varying data size of input tasks.

C. Simulation with Varying Data Size of Tasks

In this simulation, we set the number of input tasks to 150, and changed the data size of input tasks from 20 MB to 200 MB, with an increment of 20 MB to examine the effect of the data size of input tasks. Figure 4 shows the simulation results. It shows that on the average, our approach accommodates 52% more input tasks than that of the Local approach and 86% more input tasks than that of the Random approach. It is noted that as the data size increases, our approach performs even better, and the Random approach performs even worse. This is because as the data size increases, although the overhead of data transmission increases, our approach achieves effective load balancing to reduce the overhead, which is accomplished by Step 3.2) – 3.4) in our approach.

D. Simulation with Varying Connectivity of the Edge Computing Network

In this simulation, we set the number of input task to 200, data size of each input task to 50 MB, and changed α and β as described in Subsection A, to examine the effect of the connectivity of the edge computing network. Figure 5 shows the simulation results. It shows that on the average, our approach accommodates 34% more input tasks than that of the Local approach and 205% more input tasks than that of the Random approach. As the connectivity of the edge network increases, the performance of our approach also improves because a higher connectivity leads to better resources utilization and more effective load balancing in the edge computing network. As the Local approach does not involve input task distribution, the change of network connectivity does not affect its performance.

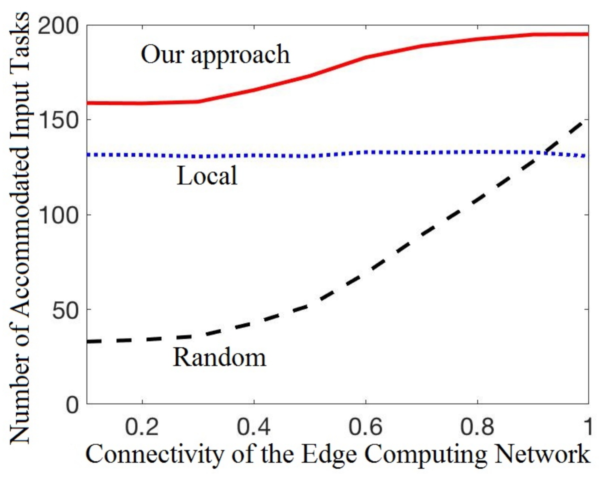

Figure 5: Comparison of the average number of the accommodated input tasks of our approach with Local and Random approaches on varying connectivity of the edge computing network.

E. Summary of the Experiments

We have found that in all the three simulations, our approach outperforms both the baseline approaches, in terms of the number of accommodated input tasks. We have also found that our approach has better performance when the number of input tasks is large, data size of input tasks is large, or the connectivity of the edge network is high.

VIII. Conclusion and Future Work

In this paper, we have presented an approach to input task distribution to increase the number of input tasks that can be processed in the edge computing network for IoT applications satisfying all the QoS requirements of the input tasks. The improvement of our approach on accommodating more input tasks in the edge computing network over the local execution approach and the random distribution approach is shown by simulation.

To make our approach more useful, we plan to extend our approach to include the input tasks with different priorities, and VMs with different types of operating systems in the edge computing network. That is, the assumptions A-1 and A-3 in our approach will be removed.

References

[1] K. Kumar, J. Liu, Y. Lu, and B. K. Bhargava, “A survey of computation offloading for mobile systems,” *Mobile Networks and Applications*, vol. 18, no. 1, pp. 129–140, 2013.

[2] M. R. Rahimi, J. Ren, C. H. Liu, A. V. Vasilakos, and N. Venkatasubramanian, “Mobile cloud computing: A survey, state of art and future directions,” *Mobile Networks and Applications*, vol. 19, no. 2, pp. 133–143, 2014.

[3] K. Kumar and Y. Lu, "Cloud computing for mobile users: Can offloading computation save energy?" *IEEE Computer*, vol. 43, no. 4, pp. 51–56, 2010.

[4] W. Hu, Y. Gao, K. Ha, J. Wang, B. Amos, Z. Chen, P. Pillai, and M. Satyanarayanan, "Quantifying the impact of edge computing on mobile applications," in *Proc. ACM SIGOPS Asia-Pacific Workshop on Systems*, 2016, pp. 5:1–5:8.

[5] M. Satyanarayanan, P. Bahl, R. Cáceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *IEEE Pervasive Computing*, vol. 8, no. 4, pp. 14–23, 2009.

[6] D. Zeng, L. Gu, S. Guo, Z. Cheng, and S. Yu, "Joint optimization of task scheduling and image placement in fog computing supported software-defined embedded system," *IEEE Trans. Computers*, vol. 65, no. 12, pp. 3702–3712, 2016.

[7] M. T. Beck, M. Werner, S. Feld, and S. Schimper, "Mobile edge computing: A taxonomy," in *Proc. Intl. Conf. on Advances in Future Internet*. Citeseer, 2014.

[8] F. Bonomi, R. A. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proc. Workshop on Mobile cloud computing*, 2012, pp. 13–16.

[9] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Trans. Networking*, vol. 24, no. 5, pp. 2795–2808, 2016.

[10] Y. Mao, J. Zhang, and K. B. Letaief, "Dynamic computation offloading for mobile-edge computing with energy harvesting devices," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 12, pp. 3590–3605, 2016.

[11] L. Gu, D. Zeng, S. Guo, A. Barnawi, and Y. Xiang, "Cost efficient resource management in fog computing supported medical cyber-physical system," *IEEE Trans. Emerging Topics Computing*, vol. 5, no. 1, pp. 108–119, 2017.

[12] J. Oueis, E. C. Strinati, and S. Barbarossa, "The fog balancing: Load distribution for small cell cloud computing," in *Proc. IEEE Vehicular Technology Conf.*, 2015, pp. 1–6.

[13] S. Sardellitti, S. Barbarossa, and G. Scutari, "Distributed mobile cloud computing: Joint optimization of radio and computational resources," in *Proc. IEEE GLOBECOM Workshops*, 2014, pp. 1505–1510.

[14] S. Sardellitti, G. Scutari, and S. Barbarossa, "Joint optimization of radio and computational resources for multicell mobile-edge computing," *IEEE Trans. Signal and Information Processing over Networks*, vol. 1, no. 2, pp. 89–103, 2015.

[15] A. S. Thyagaturu, Y. Dashti, and M. Reisslein, "Sdn-based smart gateways (sm-gws) for multi-operator small cell network management," *IEEE Trans. Network and Service Management*, vol. 13, no. 4, pp. 740–753, 2016.

[16] A. S. Thyagaturu, A. Mercian, M. P. McGarry, M. Reisslein, and W. Kellerer, "Software defined optical networks (sdons): A comprehensive survey," *IEEE Communications Surveys and Tutorials*, vol. 18, no. 4, pp. 2738–2786, 2016.

[17] S. S. Yau and H. G. An, "Adaptive resource allocation for service-based systems," *Intl. Journal of Software and Informatics*, vol. 3, no. 4, pp. 483–499, 2009.

[18] S. S. Yau and D. Huang, "Distributed monitoring and adaptation of multiple qos in service-based systems," in *in Proc. IEEE Intl. Workshop on Software Cybernetics, in conjunction with Intl. Computer Software and Applications Conf. (COMPSAC)*, 2011, pp. 31–36.

[19] S. S. Yau, Y. Yin, and H. G. An, "An adaptive approach to optimizing tradeoff between service performance and security in service-based systems," *Intl. Journal on Web Service Research*, vol. 8, no. 2, pp. 74–91, 2011.

[20] S. S. Yau and H. G. An, "Protection of users' data confidentiality in cloud computing," in *Proc. Asia-Pacific Symposium on Internetware*, 2010, pp. 32–37.

[21] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms (3. ed.)*. MIT Press, 2009.

[22] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[23] Y. Ye, "An o (n 3 l) potential reduction algorithm for linear programming," *Mathematical programming*, vol. 50, no. 1, pp. 239–258, 1991.

[24] B. M. Waxman, "Routing of multipoint connections," *IEEE Journal on Selected Areas in Communications*, vol. 6, no. 9, pp. 1617–1622, 1988.