An Efficient Module Deployment Algorithm in Edge Computing

Jang-Ping Sheu, Yi-Cian Pu, Jagadeesha RB, and Yeh-Cheng Chang
Department of Computer Science, National Tsing Hua University, ROC
sheujp@cs.nthu.edu.tw, bravery93@hotmail.com, jagadeesha.rb@gmail.com, jas1123kimo@gmail.com

***Abstract*—Edge computing refers to data storage and processing at the edge of a network instead of cloud data centers. In edge computing, user data can be preprocessed to reduce the network traffic and load of the cloud system. Module deployment scheme is vital in edge computing since it has a significant effect on the performance and the resource utilization of the edge computing system. Our objective is to maximize satisfied user requests under the constraints of limited communication bandwidth and storage capacity of devices where the modules are deployed. Since the optimization problem is NP-hard, we propose a heuristic algorithm based on relaxation and rounding techniques to solve this problem. The simulation results show that our algorithm satisfies more user requests than previous work.**

Keywords- Edge Computing; Module Deployment; Integer Nonlinear Programming; Bandwidth Guarantees

I. INTRODUCTION

The concept of edge computing has been introduced in recent years [1, 2]. Edge computing is a method of optimizing cloud computing systems by pushing computing power and storage capacity to the edge of a network, near the source of data. In edge computing, servers at edge network help to preprocess user data, which can reduce increasing amount of application traffic and enhance user privacy of applications involving private data. Furthermore, the performance of latency-sensitive applications is improved by edge computing because of short geographic distance to computing function.

In current cloud computing and edge computing networks, virtualization is one of the key components. Virtualization abstracts hardware that allows multiple workloads to share a common set of resources. More specifically, server virtualization allows multiple virtualized modules, which provide services to be collocated in a single physical server while maintaining full isolation from each other. For brevity, we call virtualized modules as modules in the rest of this paper. With virtualization technology, network operators can simply deploy or migrate the modules to achieve better resource utilization, load balance, and energy efficiency [3].

Previous work on edge computing covers a variety of aspects. Some works [4-6] are about edge computing architecture implementation, some of them such as [7, 8] focus on virtualization technologies used in edge computing infrastructure. There is also research like [9] that study module scheduling in edge computing environment. It is no doubt that there would arise a need for further research to meet the challenges evolving from cloud computing into edge computing. Currently, the computational and storage capacity in edge computing are not as high as the ones in cloud computing. With the explosive growth of the Internet of Things (IoTs) applications, the resource in cloud and edge computing such as storage capacity or network bandwidth will gradually become insufficient to support applications. Due to the reasons given above, there arises a need to schedule modules more carefully in edge computing. Our research is highly motivated by the above requirement.

In this paper, we investigate the module deployment problem in edge computing platform. Our objective is to maximize the number of satisfied user requests. To serve a set of user requests, the central server needs to deploy all required modules to the devices of edge computing platform and the modules on devices would collaborate to complete the user requests. Since there are traffic flows between modules, we also consider the bandwidth capacity constraint of each device on the platform. We formulate the problem as an integer nonlinear programming (INLP) problem. However, the optimal solution of this problem is intractable. We devise an algorithm with the use of linear programming (LP) relaxation and rounding scheme to solve this problem. We evaluate the performance of our algorithm by simulations and the simulation results show that our algorithm has better performance than alternative approaches.

The rest of this paper is organized as follows. In Section II, we discuss the related work. In Section III, we define our problem and present the module deployment algorithm. In Section IV, we evaluate the performance of our algorithm with simulations. Finally, Section V concludes this paper.

II. RELATED WORK

There is research in earlier years presenting similar concept about edge computing such as cloudlet [10] and fog computing [11]. Recently, there are studies about edge computing architecture implementation [4-6]. Although the above research present different paradigms involving edge network, the main goal of all the research is the same and is to optimize cloud computing systems by performing data processing at the edge of the network. In addition to the studies about edge network paradigm, some research focuses on virtualization technologies used in edge computing infrastructure [7, 8]. Currently, there are two kinds of popular virtualization: the traditional virtual machine (VM) such as Xen and KVM and emerging container technology like LXC and Docker. For VMs, they run not just a full copy of an operating system, but a virtual copy of all the hardware that the operating system needs to run; therefore, VMs would take up a lot of system resources. As for container, it is a lightweight virtualization technology and has advantages that it requires fewer resources and can be set up in a short time compared to VMs.

The authors in [9] study module deployment algorithm (MDA) in fog computing platform. They implement the system with a server as the controller and some Raspberry Pi IoT devices to collect sensing data. In addition, the authors formulate module deployment problem as an INLP problem and propose a heuristic algorithm to solve it in polynomial time. The problem aims at maximizing satisfied request. However, different from our work, the problem does not consider bandwidth constraint, which is an important factor of the performance in edge computing systems.

There are two popular bandwidth provisioning models applied in current research. The so-called pipe model [12] guarantees bandwidth between pairs of VMs. The traffic demand is expressed as a matrix of bandwidth requirements between each pair of VMs. The hose model is a less complex model [13]. In hose model, traffic demand is only expressed as aggregate incoming traffic to one endpoint (ingress traffic) or outgoing traffic from one endpoint (egress traffic). When the traffic flow is predictable and stable, the pipe model can precisely capture the application traffic needs. However, it is usually difficult for network operators to measure and predict the actual traffic matrix [14]. Also, the implementation of pipe model provisioning has increasing high complexity when the number of endpoint increases. The hose model has advantages of ease of specification and flexibility. In hose model, the data to and from a given hose endpoint can be distributed arbitrarily over other endpoints. Thus, we adopt hose model in the paper to specify the bandwidth requirement of each module. In our considered problem, each module would have ingress and egress bandwidth requirement.

III. Algorithms

3.1 Problem Formulation

We consider our problem in edge computing. There is a powerful server, namely controller, and many heterogeneous devices on the edge computing platform. The platform can receive requests from users and then the controller would analyze the requests and deploy specific modules on devices to fulfill the user jobs. Since the device resources such as storage and network bandwidth are limited, the controller needs to deploy modules on suitable devices to make use of system resources efficiently and try to satisfy as many user requests as possible. The platform can be used as a micro data center at edge network and serve user requests or perform data preprocessing to alleviate the loading of cloud networks.

Let R denote the set of all user requests. Each request $r \in R$ requires $M_r \subseteq M$ modules, that is, only the modules in M_r can collaborate to satisfy the request *r*. Let M denote the set of all modules. Let D denote the set of all devices and each device has storage capacity C_d which indicates the number of modules that can be loaded in a device *d*. This is because, we assume each module has the same size for simplicity. Note that, each module *m* is deployed at most on one device *d*. The modules in M_r would communicate with each other to satisfy user request *r*, so we consider the module deployment problem with network bandwidth constraint. We denote the aggregation ingress traffic of each module *m* as $\alpha_{in}(m)$ and the aggregation egress traffic as $\alpha_{out}(m)$ in terms of data bits per unit time. Note that, a module *m* can be used by more than one request. In this paper, both $\alpha_{in}(m)$ and $\alpha_{out}(m)$ stand for amount of maximum ingress and egress traffic volume. For device bandwidth, each device *d* has bandwidth capacity $\beta_{in}(d)$ and $\beta_{out}(d)$ denoting ingress bandwidth capacity and egress bandwidth capacity in terms of data bits per unit time, respectively. The controller will deploy modules on devices for each request in the edge computing platform. Moreover, the modules in M_r have to be deployed on user-specified devices $D_r \subseteq D$.

With the above definitions, the module deployment problem can be formulated as an INLP problem as follows.

$$max \sum_{r\in R} \prod_{m\in M_r} y_{r,m} \tag{1a}$$

$$st: \; y_{r,m} = \sum_{d\in D_r} x_{m,d}, \; \forall r \in R, \forall m \in M_r \tag{1b}$$

$$\sum_{d\in D} x_{m,d} \le 1, \; \forall m \in M \tag{1c}$$

$$\sum_{m\in M} x_{m,d} \le C_d, \; \forall d \in D \tag{1d}$$

$$\sum_{m\in M} \alpha_{in}(m)\, x_{m,d} \le \beta_{in}(d), \; \forall d \in D \tag{1e}$$

$$\sum_{m\in M} \alpha_{out}(m)\, x_{m,d} \le \beta_{out}(d), \; \forall d \in D \tag{1f}$$

$$x_{m,d}, y_{r,m} \in \{0, 1\}, \; \forall m \in M, \forall d \in D, \forall r \in R \tag{1g}$$

The objective function (1a) is to maximize the satisfied requests, where binary decision variable $y_{r,m}$ indicating if module *m* in M_r has been deployed on any device in D_r. In constraint (1b), $x_{m,d}$ is a binary decision variable indicating if module *m* is deployed on device *d*. Constraint (1c) means that each module *m* is deployed at most on one device *d*. Constraint (1d) guarantees that the assigned number of modules cannot exceed the device storage capacity. Constraint (1e) and (1f) ensure that the aggregated module traffic on a device *d* should not exceed its bandwidth capacity for both ingress and egress directions. Constraint (1g) indicates that $x_{m,d}$ and $y_{r,m}$ are binary decision variables, which makes the formulation problem as an integer programming problem. Since the binary terms $y_{r,m}$ belonging to the same request are multiplied by each other, the objective function (1a) is a nonlinear form.

The above problem is a variation of the NP-hard 0/1 multiple knapsack problem [15]. We have limited knapsack (devices) capacity, and want to pack as many items (modules) as possible, thus, getting more profits (satisfied requests). Since our problem is NP-hard, it costs high computational effort to find the optimal solution. Hence, we develop a Bandwidth guaranteed Module Deployment Algorithm (BMDA) in polynomial time to get a feasible solution.

3.2 LP Relaxation

The primary idea of our method to solve the INLP problem is based on LP relaxation and rounding method. We relax the INLP formulation to an LP formulation first and solve the relaxed formulation with problem input by an LP solver. Then, we get fractional solution given by solver and try to round the solution with our proposed algorithm.

Due to the nonlinear nature of our objective function as stated in 1(a), the request *r* is satisfied if all binary terms $y_{r,m}$ belonging to the request *r* equal to one. In short, a request *r* is satisfied when all its modules in M_r are deployed on the specific devices D_r. We use a trick to transform the objective function (1) by replacing product operator with summation and obtain a new form in (2). The objective function (2) with constraints (1b) - (1g) is equivalent to the original problem formulation; we get the same result when all decision variables $x_{m,d}$ are determined by the same value. Instead of multiplying all $y_{r,m}$ belonging to the same request *r*, we add them together and divide the summation with $|M_r|$, which is the number of modules in request *r*. Then, we add a floor function and get the objective function (2). When all $y_{r,m}$ belonging to the request *r* are equal to one, then $\lfloor \sum_{m\in M_r} y_{r,m}/|M_r| \rfloor$ has a value 1, otherwise 0.

$$max \sum_{r\in R} \lfloor \sum_{m\in M_r} y_{r,m}/|M_r| \rfloor \tag{2}$$

However, the formulation in (2) is nonlinear too because of the floor function. Thus, we relax the formulation by removing floor function. With the relaxation, we get an objective function in linear form. At last, we relax binary decision variables $x_{m,d}$ by allowing them to take any real value in [0, 1] and get a continuous linear programming that can be solved in polynomial time.

3.3 Algorithm BMDA

The pseudo-code of our proposed algorithm is described in Algorithm BMDA. There are three procedures in BMDA. First, we solve the LP problem by an LP solver and get the fractional solution $x' = \{x'_{m,d} | m \in M$ and $d \in D\}$ (line 3). Second, we try to round the fractional decision variables $x'_{m,d}$ with designed rounding method (line 5). Last, we update the problem input data to a new one by removing the deployed modules and satisfied requests (line 7). The algorithm BMDA iteratively carries out the three procedures to get a deployment solution until no feasible fractional solution exists (line 12) or the rounding procedure cannot round any fractional decision variables $x'_{m,d}$ (line 9). There are two conditions to determine the process of the algorithm BMDA. First, we check that if the LP solver outputs a fractional solution (line 4) after calling the LP solver to solve the relaxed problem. If there is a feasible fractional solution given by the LP solver, the algorithm BMDA executes the next procedure, the rounding method (line 5). If there is no feasible fractional solution for rounding, the algorithm BMDA should terminate (line 12). Second, we check that if at least one decision variable $x_{m,d}$ has been updated, i.e. deploying at least one module (line 6) after calling the rounding method. If at least one decision variable $x_{m,d}$ has been updated in current iteration, the algorithm BMDA executes the next procedure to update problem input (line 7). If no decision variable $x_{m,d}$ has been updated, i.e. deploying no any module, the algorithm BMDA should terminate (line 9).

Algorithm BMDA

Input: R; M; D; M_r; D_r; C_d; $\alpha_{in}(m)$; $\alpha_{out}(m)$; $\beta_{in}(d)$; $\beta_{out}(d)$; P_d; P_in_d; P_out_d;
Output: The module deployment solution $x = \{x_{m,d} | m \in M$ and $d \in D$; $x_{m,d}=1\}$
1. *final*←**false**;
2. **while** !*final* **do**
3. Call an LP solver to solve the relaxed problem;
4. **if** a feasible fractional solution $x' = \{x'_{m,d} | m \in M$ and $d \in D\}$ exists **then**
   Call Rounding method to round the fractional solution x';
5. **if** at least one decision variable $x_{m,d} \in x$ has been
6. updated in rounding method **then**
7. Update input data by removing deployed modules and satisfied requests;
8. **else**
9. *final*←**true;**
10. **end if**
11. **else**
12. *final*←**true**;
13. **end if**
14. **end while**

3.3.1 Rounding Method

After solving the relaxed formulation with problem input by an LP solver, we get the fractional solution $x' = \{x'_{m,d} | m \in M$ and $d \in D\}$. Next, we need to round the fractional solution x' to get the binary solution $x = \{x_{m,d} | m \in M$ and $d \in D\}$. We propose a greedy rounding method based on a priority index called request satisfaction value S_r. We determine the value S_r by calculating the amount of a request *r* that has been satisfied with the fractional solution x' given by the LP solver.

$$S_r = \sum_{m \in M_r} y_{r,m} / |M_r|, \forall r \in R \qquad (3)$$

For example, assume a request r_1 requires modules m_1, m_2, and m_3 i.e. $|M_{r_1}|$=3. Assume we obtain the solution to the relaxed problem given by the LP solver, x'_{m_1,d_1} =1.0, x'_{m_2,d_2} =0.4, x'_{m_2,d_3} =0.6, and x'_{m_3,d_4} =0.8. Then, we have y_{r_1,m_1}=1.0, y_{r_1,m_2}=1.0 and y_{r_1,m_3}=0.8. Thus, we can calculate the request satisfaction value S_{r_1}=0.93 ((1 + 1 + 0.8) / 3). The rounding method assigns a higher priority to a request of higher S_r value that corresponds to our objective of maximizing the number of satisfied requests. Note that since the value of decision variable $x'_{m,d}$ is fractional, the value of $y_{r,m}$ is also fractional. The fractional value $y_{r,m}$ stands for the amount of a module *m*, where $m \in M_r$, has been deployed on the edge computing platform.

The pseudo-code for our proposed method to round fractional solution is described as follows. In line 1, we set a Boolean variable *updated* to false. The variable is an indicator showing that if at least one fractional decision variable has been rounded, i.e. deploying at least one module in each iteration of algorithm BMDA. We will use the indicator to terminate the process of the rounding procedure. In line 2, we sort fractional decision variables based on $x'_{m,d}$ value in decreasing order. Note that a module *m* may be deployed on more than one device. We prefer to round the largest one. From line 3 to line 5, the rounding procedure calculates request satisfaction value S_r for each unsatisfied request *r*. Then, the procedure sorts the $r \in R$ based on S_r in decreasing order. If more than one request has the same S_r, we use $|M_r|$ to break the tie (in increasing order) because it is easier to satisfy a request with smaller size $|M_r|$ (line 6).

Next, the procedure is to round the fractional solution x' and there are two conditions in the rounding process (line 8). First, when there exists an S_r equal to one, the rounding procedure has to handle the user request *r*. This is because the S_r equals to one means that the procedure has big chance to deploy all required modules to satisfy the request *r*. Therefore, we should try to handle a request of S_r=1. Second, when no any module is deployed in current iteration of rounding procedure, i.e. *updated* is false, the rounding procedure should keep trying to deploy modules on devices. We set the second condition in rounding procedure to guarantee at least one module to be deployed when algorithm BMDA calls rounding method every time. The rounding procedure would continue to handle requests until S_r is less than one and *updated* is true (line 8). Our rounding method is designed to gradually obtain the deployment decision $x_{m,d}$, $\forall m \in M$ and $\forall d \in D$ for all modules. After choosing a request *r*, for each decision variable $x'_{m,d}$ in fractional solution set x' we check if the condition in line 11 is satisfied. In line 11, the P_d means the number of modules deployed on device *d*. The P_in_d and P_out_d mean accumulated ingress and egress module traffic on device *d*, respectively. If the above conditions are met, we deploy the module *m* on the device *d*, set *updated* to true (line 12), and break out the loop to deploy next module (lines 13).

Rounding Method

Input: $x' = \{x'_{m,d} | m \in M$ and $d \in D$; $x'_{m,d} > 0\}$;
Output: The module deployment solution $x = \{x_{m,d} | m \in M$ and $d \in D$; $x_{m,d}=1\}$;
1. Initially, set *updated*←**false**;
2. Sort the $x'_{m,d} \in x'$ based on $x'_{m,d}$ value in decreasing order;
3. **for** each $r \in R$ **do**
4. Calculate the request satisfaction value S_r for each request *r*;

5. **end for**
6. Sort the $r \in R$ based on S_r in decreasing order and uses $|M_r|$ to break the tie (in increasing order);
7. **for** each $r \in R$ **do**
8. **if** S_r == 1 **or** *updated* == **false do**
9. **for** each $m \in M_r$ **do**
10. **for** each $x'_{m,d} \in x'$ **do**
11. **if** there exists matched *m* of $x'_{m,d}$ **and** $P_d + \lceil x'_{m,d} \rceil \leq C_d$ **and** $P\_in_d + \alpha_{in}(m)\lceil x'_{m,d} \rceil \leq \beta_{in}(d)$ **and** $P\_out_d + \alpha_{out}(m)\lceil x'_{m,d} \rceil \leq \beta_{out}(d)$ **then**
12. deploy module *m* on device *d* and set *updated* to **true**;
13. **break**; //procedure breaks out the loop and goes to line 9
14. **end if**
15. **end for**
16. **end for**
17. **else**
18. **break**;
19. **end if**
20. **end for**

Note that the LP solver tends to satisfy a request with small size of $|M_r|$. The algorithm BMDA is designed to satisfy first a request with high S_r and small size of $|M_r|$. The BMDA would update problem input by removing satisfied requests and deployed modules thus reducing the size of $|M_r|$ of requests. Therefore, although a request *r* is not fully satisfied in current iteration of algorithm BMDA, the unsatisfied request *r* will have higher priority to be processed in the next iteration due to reduced size of $|M_r|$.

We illustrate the concept of algorithm BMDA by taking an example as follows. Assume there are three user requests r_1={m_1, m_2}, r_2={m_3}, r_3={m_4, m_5} and two devices d_1 and d_2. The storage capacity of C_{d_1}= 2 and C_{d_2}= 1. For explaining the rounding method easily, we assume bandwidth capacity of all devices is large enough to hold module traffic in the example. In addition, all the requests allow the controller to deploy required modules on all devices. In the first iteration of algorithm BMDA, the LP solver solves the relaxed problem and we get fractional solution: x'_{m_1,d_1}=1.0, x'_{m_3,d_1}=0.7, x'_{m_3,d_2}=0.3, x'_{m_4,d_1}=0.3, and x'_{m_4,d_2}=0.7. Thus, we have y_{r_1,m_1}=1.0, y_{r_1,m_2}=0, y_{r_2,m_3}=1.0, y_{r_3,m_4}=1.0 and y_{r_3,m_5}=0. Then, we calculate request satisfaction value for each request and get S_{r_1}=0.5, S_{r_2}=1.0, and S_{r_3}=0.5. We sort all the requests based on S_r and have the processing priority: $r_2 > r_1 = r_3$. The rounding procedure handles request r_2={m_3} and then rounds x'_{m_3,d_1}=0.7 to one successfully. Therefore, module m_3 is deployed on device d_1 and request r_2 is satisfied. Since S_{r_1} is less than one and *updated* has been set to true after deploying m_3, the rounding procedure terminates.

Algorithm BMDA updates input data into: r_1={m_1, m_2}, r_3={m_4, m_5}, C_{d_1}= 1, and C_{d_2}= 1. In the second iteration, the LP solver outputs: x'_{m_1,d_1}= 0.7, x'_{m_1,d_2}= 0.3, x'_{m_4,d_1}= 0.3, and x'_{m_4,d_2}= 0.7. Thus, we have y_{r_1,m_1}= 1.0, y_{r_1,m_2}=0, y_{r_3,m_4}= 1.0 and y_{r_3,m_5}= 0, S_{r_1}= 0.5, and S_{r_3}= 0.5. The rounding procedure selects r_1 and rounds x'_{m_1,d_1}= 0.7 to one successfully. Therefore, module m_1 is deployed on device d_1. Because S_{r_3} is less than one and *updated* has been set to true after deploying m_1, the rounding procedure terminates. Algorithm BMDA updates input data into: r_1={m_2}, r_3={m_4, m_5}, and C_{d_2}= 1. In the third iteration, LP solver outputs: x'_{m_2,d_2}=1 and we have S_{r_1}=1, and S_{r_3}=0. The module m_2 is deployed on device d_2 and the request r_1 is satisfied. Because there is no storage capacity for all devices, the algorithm BMDA terminates. In this example, two requests r_1 and r_2 are satisfied by our proposed algorithm BMDA.

3.3.2 Time Complexity Analysis

We use a popular optimization solver, CPLEX [16], to be the LP solver. When CPLEX solves an LP problem, the running time of the algorithm in CPLEX is proportional to the number of constraints [17]. Thus, the time complexity to solve the relaxed LP problem is $O(|M||D|)$. In the rounding method, it first takes $O(|M||D|\lg|M||D|)$ to sort the fractional solution (line 2). Then, it takes $O(|R|\lg|R|)$ to sort the requests R (line 6). The time complexity of rounding method in lines 7, 9 and 10 is $O(|R||M|^2|D|)$. Algorithm BMDA iteratively executes LP solver and rounding the fractional solution. Since the rounding procedure must round at least one fractional variable in each iteration, there are at most $|M|$ iterations in BMDA algorithm. The time complexity of BMDA is $O(|R||M|^3|D| + |M||R|\lg|R| + |M|^2|D|\lg(|M||D|)$.

IV. PERFORMANCE EVALUATION

In this section, we evaluate the performance of algorithm BMDA by simulations. We compare algorithm BMDA with a Greedy algorithm, the algorithm MDA [9] and the optimal solution. The Greedy algorithm uses $|M_r|$ and the total required module traffic of a request to determine the handling priority of each request *r*. The Greedy algorithm sorts the requests based on $|M_r|$ in increasing order and use the total required module traffic of a request to break the tie (in increasing order). When we compare BMDA with MDA, we assume a special case that the bandwidth capacity of all devices is infinity because MDA does not consider the network bandwidth constraint. We use C++ programming language to implement all algorithms in a personal computer.

4.1 Simulation Environment

In our simulations, the number of required modules $|M_r|$ of each request *r* varies between 1 ~ 5 randomly. We determine the required modules M_r of each request *r* by selecting from all *M* modules randomly. Also, we determine specified devices D_r of each request *r* by selecting from all $|D|$ devices randomly. For module traffic in the simulations, we generate the traffic load based on [18]. The authors in [18] show that traffic characteristic and arrival pattern in data centers properly fit with the log-normal distribution. Therefore, we use log-normal distribution to generate module traffic in our platform. The log-normal distribution probability density function (PDF) is shown in (4).

$$f(v; \mu, \sigma) = \frac{1}{v\sigma\sqrt{2\pi}} e^{-\frac{(\ln v - \mu)^2}{2\sigma^2}}, v > 0 \quad (4)$$

In our simulations, the mean μ is set to 0 while the standard deviation σ is set to 1. For each device, the value of bandwidth capacity varies between 8 ~ 12 with increments of two randomly. As to module capacity of each device, the value of module capacity varies between 1 ~ 4 (as an integer) randomly. When we conduct simulations with small-scale problem input, we set module capacity as 1 ~ 3 randomly. In all the following experiments, we run the tests 100 times in each simulation and show 95% confidence interval of all tests.

4.2 Simulation Results

At first, we conduct the simulation with small-scale inputs. We set the number of modules $|M|$=10 and the number of devices $|D|$=5. We vary the number of total requests $|R|$ from 20 ~ 30 with increments of 2 to determine the satisfied requests.

We compare the performance of our algorithm BMDA with Greedy algorithm and the optimal solution given by CPLEX solver. From the simulation result shown in Fig. 4.1, it is obvious that algorithm BMDA has better performance than Greedy algorithm. Furthermore, our algorithm BMDA can satisfy about 87% of the number of satisfied requests compared to optimal solution, while the Greedy algorithm can satisfy about 68% of the optimal solution.

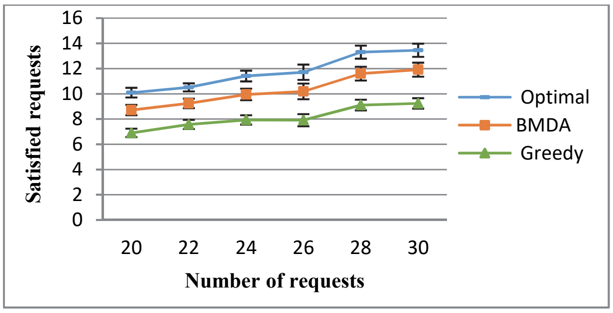

Figure 4.1 Satisfied requests vs. different number of requests with small-scale problem.

In the next simulation, we also compare our algorithm BMDA with algorithm MDA [9]. Figure 4.2 shows the simulation results and it reveals that our algorithm BMDA has better performance than MDA and Greedy algorithm. We can observe that BMDA can satisfy about 87% of the number of satisfied requests by optimal solutions while the MDA and Greedy algorithm can satisfy about 69% of the optimal solutions. It is noticeable that the MDA and the Greedy algorithm have nearly the same performance. This is because the MDA is also a greedy heuristic algorithm. The MDA sorts requests based on $|M_r|$ in increasing order while use $|D_r|$ to break the tie. Both the MDA and Greedy algorithm first determine handling priority of requests based on $|M_r|$ leading to almost the same performance.

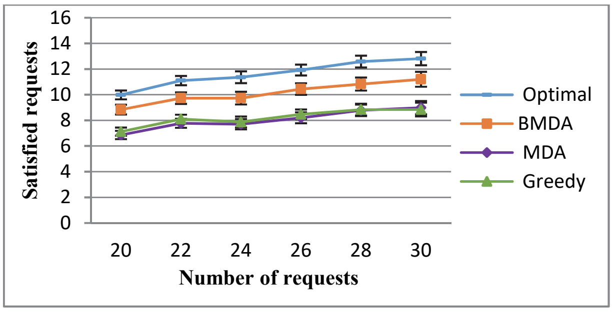

Figure 4.2 Satisfied requests vs. different number of requests with small-scale problem and bandwidth capacity is unlimited.

In the following, we conduct the simulations with large-scale problem input without the optimal solution due to the time complexity of the optimal solution. We set the number of modules $|M|$ = 50 and the number of devices $|D|$ = 20. We vary the number of total requests $|R|$ from 50 to 300 with increments of 50. Figure 4.3 shows the performance of our algorithm and Greedy algorithm under the limited bandwidth capacity. Figure 4.4 shows the performance of our algorithm, MDA, and Greedy algorithm under the unlimited bandwidth capacity. Since the 95% confidence interval is small, it is not evident in the figures. From Fig. 4.3 and Fig. 4.4, we can observe that our algorithm BMDA has significant performance improvement compared to MDA and Greedy algorithm. Besides, both MDA and Greedy algorithms satisfy about 60% of the number of satisfied requests compared to BMDA.

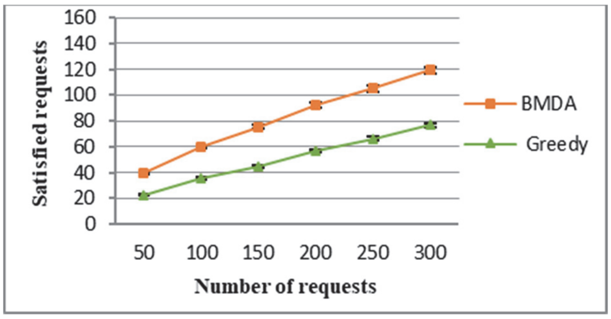

Figure 4.3 Satisfied requests vs. different number of requests with large-scale problem.

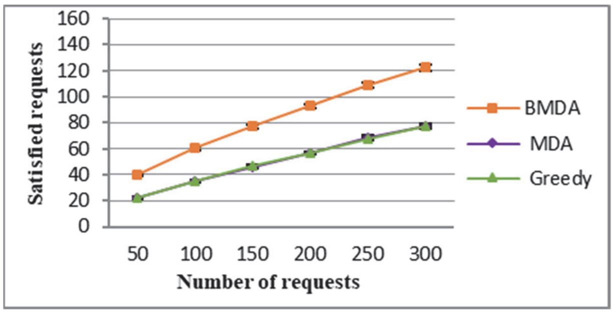

Figure 4.4 Satisfied requests vs. different number of requests with large-scale problem and bandwidth capacity is unlimited.

In Fig. 4.5, we evaluate the performance of algorithm BMDA by varying the average size of specified devices D_r of each request *r*. For the simulations, we set the number of requests $|R|$=200, the number of modules $|M|$=50 and the number of devices $|D|$=20. We determine the average size of D_r by setting a probability when choosing each device. We use the normal distribution PDF to assign the probability to choose a device. Therefore, a user can select each device by different probabilities when determining specified devices D_r. In the simulation, we increase the mean μ of the normal distribution PDF from 0.1 to 1.0 by 0.1 at regular intervals while the standard deviation σ is set to 0.1. Thus, average size of D_r would be μ*$|D|$, i.e. 2~20 with increment of 2 at regular intervals in the simulations. In Fig. 4.5, we can see that the size of D_r has a significant influence on the number of satisfied requests. This is because each module would be located on at most one device in the module deployment problem. If a user request *r* specifies larger size of D_r to deploy its modules in M_r, it is more likely the modules in M_r deployed on devices in D_r. From the simulation results, we can observe that our algorithm has higher performance than Greedy algorithm for various size of D_r. The performance of Greedy algorithm is about 60% of the algorithm BMDA. However, when users allow controller to deploy required modules on all devices, i.e. $|D_r|$ = 20, the number of satisfied requests by BMDA and Greedy algorithm get closer. This is because a module can be deployed on any device *d* and the module can be used by all the requests.

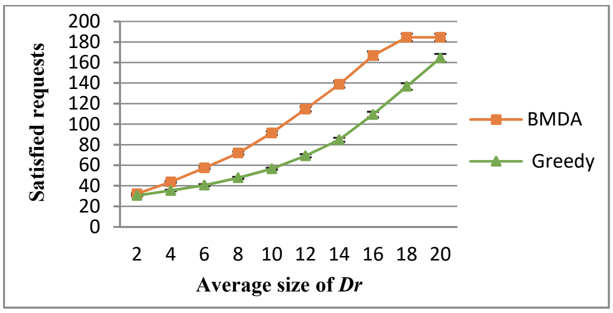

Figure 4.5 Satisfied requests vs. average size of D_r.

In Fig 4.6, we evaluate the performance of algorithm BMDA by varying the average size of specified devices D_r of each request r and also compare BMDA with Greedy and MDA. The simulation result is similar to Fig 4.5. The performance of our algorithm is higher than the benchmark algorithms. The performance gap between BMDA and the benchmark algorithms is large when the average size of D_r is between 10 and 18. The performance of MDA and Greedy algorithms is about 61% of our algorithm BMDA.

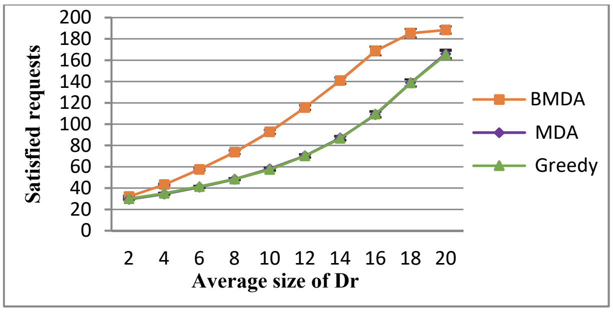

Figure 4.6 Satisfied requests vs. average size of D_r and bandwidth capacity is unlimited.

In the last simulations, we evaluate the performance of algorithm BMDA under different network size. We vary the number of devices $|D|$ from 5~30 with increments of 5 and set the number of requests $|R|$ as six times of $|D|$. The number of modules $|M|$ is set to 20. In Fig. 4.7 and Fig. 4.8, we can observe that algorithm BMDA has better performance than algorithm MDA and Greedy algorithms, under various network sizes. We can find that the MDA and Greedy algorithms satisfy about 68% of the number of satisfied requests compared to algorithm BMDA.

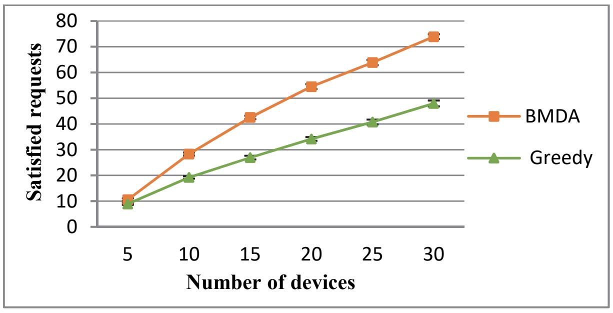

Figure 4.7 Satisfied requests vs. different size of network.

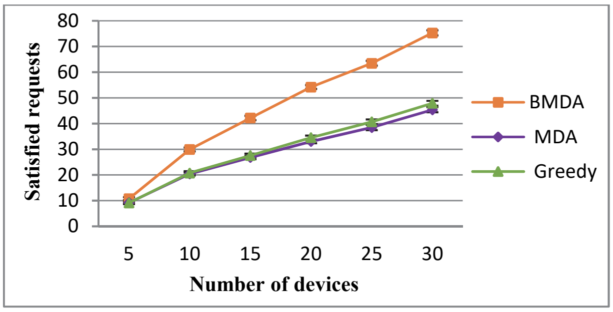

Figure 4.8 Satisfied requests vs. different size of network and bandwidth capacity is unlimited.

V. CONCLUSION

In this paper, we investigate the problem of module deployment in the edge computing with limited physical resources such as device storage and communication bandwidth. We formulate the problem as an integer nonlinear programming problem and the objective is to maximize the satisfied user requests. To solve the problem efficiently, we design a polynomial time algorithm BMDA based on LP relaxation and rounding technique. The numerical results show that our algorithm can satisfy more requests than Greedy and previous work. In addition, the result also reveals that our proposed algorithm BMDA has near-optimal performance with polynomial time complexity.

REFERENCES

[1] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, “Edge Computing: Vision and Challenges,” *IEEE Internet of Things Journal*, Vol. 3, No. 5, pp. 637-646, Oct. 2016.

[2] O. Salman, I. Elhajj, A. Kayssi, and A. Chehab, “Edge Computing Enabling the Internet of Things,” in *Proceedings of IEEE 2nd World Forum on Internet of Things* , pp. 603–608, Milan, Italy, Dec. 2015.

[3] M. F. Bari, R. Boutaba, R. Esteves, L. Z. Granville, M. Podlesny, M. G. Rabbani, Q. Zhang, and M. F. Zhani, “Data Center Network Virtualization: A Survey,” *IEEE Communications Surveys & Tutorials*, Vol. 15, No. 2, pp. 909-928, Sep. 2012.

[4] A. Ahmed and E. Ahmed, “A Survey on Mobile Edge Computing,” in *Proceedings of IEEE 10th International Conference on Intelligent Systems and Control*, pp. 1-8, Coimbatore, India, Jan. 2016.

[5] R. Cziva and D. P. Pezaros, “Container Network Functions: Bringing NFV to the Network Edge,” *IEEE Communications Magazine*, Vol. 55, No. 6, pp. 24-31, Jun. 2017.

[6] K. Kaur, T. Dhand, N. Kumar, and S. Zeadally, “Container-as-a-Service at the Edge: Trade-off between Energy Efficiency and Service Availability at Fog Nano Data Centers,” *IEEE Wireless Communications*, Vol. 24, No. 3, pp. 48-56, Jun. 2017.

[7] R. Morabito, “Virtualization on Internet of Things Edge Devices with Container Technologies: A Performance Evaluation,” *IEEE Access*, Vol. 5, pp. 8835-8850, May. 2017.

[8] F. Ramalho and A. Neto, “Virtualization at the Network Edge: A Performance Comparison,” in *Proceedings of* WoWMoM, pp. 1-6, Coimbra, Portugal, Jun. 2016.

[9] H.-J. Hong, P.-H. Tsai, and C.-H. Hsu, “Dynamic Module Deployment in a Fog Computing Platform,” in *Proceedings of IEEE 18th Asia-Pacific Network Operations and Management Symposium*, pp. 1-6, Kanazawa, Japan, Oct. 2016.

[10] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, “The Case for VM-Based Cloudlets in Mobile Computing,” *IEEE Pervasive Computing*, Vol. 8, No. 4, pp. 14-23, Oct. 2009.

[11] S. Yi, C. Li, and Q. Li, “A Survey of Fog Computing: Concepts, Applications and Issues,” in *Proceedings of the ACM 2015 Workshop on Mobile Big Data*, pp. 37-42, Hangzhou, China, Jun. 2015.

[12] C. Guo, G. Lu, H. J. Wang, S. Yang, C. Kong, P. Sun, W. Wu, and Y. Zhang, “Secondnet: A Data Center Network Virtualization Architecture with Bandwidth Guarantees,” in *Proceedings of the ACM 6th International Conference*, No. 15, Philadelphia, Pennsylvania, Nov. 2010.

[13] N. G. Duffield, P. Goyal, A. Greenberg, P. Mishra, K. K. Ramakrishnan, and J. E. van der Merive, “A Flexible Model for Resource Management in Virtual Private Networks,” in *Proceedings of the ACM SIGCOMM*, pp. 95-108, Massachusetts, USA, Aug. 1999.

[14] T. Benson, A. Akella, and D. A. Maltz, “Network Traffic Characteristics of Data Centers in the Wild,” in *Proceedings of Conference on Internet Measurement*, pp. 267-280, Melbourne, Australia, Nov. 2010.

[15] M. G. Lagoudakis, “The 0–1 Knapsack Problem An Introductory Survey,” The Center for Advanced Computer Studies, University of Southwestern Louisiana, 1996

[16] CPLEX. Available: https://www-01.ibm.com/software/commerce/optimization/cplex-optimizer

[17] D. A. Spielman and S.-H. Teng, “Smoothed Analysis of Algorithms: Why the Simplex Algorithm Usually Takes Polynomial Time,” *Journal of the ACM*, Vol. 51, No. 3, pp. 385-463, May. 2004.

[18] T. Benson, A. Anand, A. Akella, and M. Zhang, “Understanding Data Center Traffic Characteristics,” *ACM SIGCOMM Computer Communication Review*, Vol. 40, No. 1, pp. 92–99, Jan. 2010.