2016 IEEE 24th International Conference on Network Protocols (ICNP)
Workshop on Hot Topics in Practical Networked Systems (HotPNS 2016)

Reconciling Task Assignment and Scheduling in Mobile Edge Clouds

Lin Wang*, Lei Jiao§, Dzmitry Kliazovich‡, Pascal Bouvry‡
*Technische Universität Darmstadt, Germany
§Nokia Bell Labs, Ireland
‡SnT, University of Luxembourg, Luxembourg

***Abstract*—The prosperous growth of the Internet-of-Things industry attracts numerous interests in employing edge clouds (a.k.a. cloudlets) to enhance the performance of mobile services and applications. Most existing research has been focused on offloading computational tasks from mobile devices to a single cloudlet or a central location, yet overlooked the issue of jointly coordinating the offloaded tasks in a system of multiple cloudlets. In this paper, we fill this gap by investigating the assignment and the scheduling of mobile computational tasks over multiple cloudlets, while optimizing the overall cost efficiency by leveraging the heterogeneity of cloudlets. We model both data transfer and computation in terms of monetary and time costs, with task deadlines guaranteed. We formulate the problem as a mixed integer program and prove its NP-hardness. By introducing admission control for the cloudlet provider to shape the system workload, we transform our problem into maximizing the task admission rate over the two coupled phases: data transfer and computation. We propose an efficient two-phase scheduling algorithm, and demonstrate that, compared with the conventional approach of always selecting the closest cloudlet, our approach achieves significantly higher admission rate with up to 20% reduction in the average cost of all offloaded tasks.**

I. Introduction

The last decade has witnessed a tremendous proliferation of mobile devices. While providing more sophisticated functionalities, the mobile applications are becoming more and more resource-hungry. Today, mobile devices are often drained of computing and storage capacities, leading to reduced battery lifetime and deteriorated user experience. To alleviate this situation, several cloud-based solutions (e.g., [1], [2], [3]) have been proposed to offload some of the computation-intensive tasks from mobile devices to a distant cloud. Although it bypasses the resource limitation at mobile devices, the execution of the mobile applications, especially those that are more interactive, can be highly interrupted due to the high latency between the mobile device and the remote cloud [4].

The ideas of *edge computing* and *fog computing* delegate data-related operations including storage and analytics from mobile devices to surrogates at the network edge, instead of a distant cloud, to enhance the capabilities of mobile devices. By placing a cluster of servers (a.k.a. cloudlets) close to mobile devices, task offloading can be achieved without introducing intolerable delay as in the cloud-based solution. Given the explosive adoption of mobile devices and users' heavy dependence on mobile applications in the era of Internet-of-Everything, cloudlets would be widely implemented and foreseen to serve as an indispensable infrastructure for the future Internet. A basic system model for mobile edge computing with multiple cloudlets is illustrated in Figure 1.

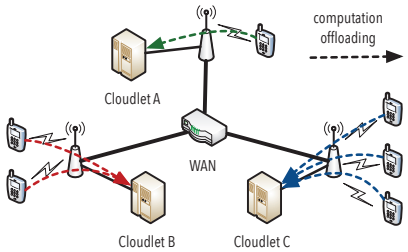


Fig. 1. System model for mobile edge computing with multiple cloudlets.

To achieve the lowest latency for every offloaded task, the current practice always allocates mobile tasks to the closest cloudlet, which, from the viewpoint of the cloudlet provider that provides and operates multiple cloudlets, can be far from optimal in terms of the overall system-wise cost efficiency. For instance, let's consider load distribution and cost optimization: i) some cloudlets may suffer from overloading when the nearby user population becomes dense at some time of the day, while other cloudlets may be idle in most of the time, leading to unbalanced resource utilization [5]; ii) the unit operational cost of each cloudlet can be different due to the heterogeneous energy prices or hardware specifications [6] so that the total operational cost of all cloudlets may be suboptimal if the closest cloudlet is always chosen. We argue that it is unnecessary to always use the closest cloudlet, as long as a certain service quality is guaranteed. The decisions on assigning and scheduling offloaded mobile tasks over a network of cloudlets become fundamental towards realizing the optimal cost efficiency in the edge cloud operations.

In this paper, we investigate the problem of optimizing the cost efficiency in mobile edge clouds through task assignment and scheduling, by taking advantage of the heterogeneity of cloudlets in terms of capability and cost. To ensure the quality of experience for mobile applications, we associate hard deadlines to offloaded tasks. Upon the arrival of an offloaded task, we determine to which cloudlet this task will be assigned and in which execution sequence the tasks that have been assigned to the same cloudlet will follow. If a task is assigned to a cloudlet other than the closest one, a bulk of data (e.g., a virtual machine containing the snapshot of the current running application) has to be firstly transferred to the target

978-1-5090-3281-5/16/$31.00 ©2016 IEEE

cloudlet, incurring an inevitable delay and network overhead. The task will eventually be executed in the assigned cloudlet with its associated deadline respected. Despite few very recent attempts on multi-cloudlet coordination [5], [6], our model is more comprehensive and realistic by taking into account both data transfer and task execution, with respect to both time and monetary aspects. To the best of our knowledge, this has not been covered in the existing literature.

We first represent the problem by a formal model and then formulate it as a mixed integer program. Unsurprisingly, the problem can be proved to be NP-hard. Moreover, we show by analysis that regardless of optimality, even deciding whether the problem has a feasible solution is already NP-complete. Following this result, we observe the necessity of introducing admission control[1] to our problem, based on which the cloudlet provider would be able to shape system workload and pursue the optimal resource utilization or cost reduction [7]. The problem is transformed into maximizing the admission rate by task scheduling while maintaining the best cost efficiency by assigning tasks to "cheaper" cloudlets. While the optimal task assignment can be achieved more straightforwardly, our target will be the problem of scheduling to achieve the maximized admission rate, which is non-trivial.

The remainder of this paper is organized as follows. Section II provides the system model, the problem formulation and its complexity analysis. Section III presents our algorithm design. Section IV validates the performance of our proposed algorithm by simulations. Section V summarizes related work and Section VI concludes the paper.

II. Preliminaries

In this section, we present the system model, formulate the problem, and carry out complexity analysis for the problem.

A. System Model

We consider a mobile edge cloud system which consists of a set of m cloudlets denoted by $\mathbf{S} = \{S_1, S_2, ..., S_m\}$. Each of the cloudlet $S_i \in \mathbf{S}$ is equipped with a certain number of servers and its maximum computing capacity is captured by C_i. All the cloudlets are connected by a Wide Area Network (WAN) as depicted in Figure 1. We assume no bottleneck in the network core, i.e., the only bottleneck exists in the connection between each cloudlet and the core. For simplicity we abstract the network as one non-blocking switch with *heterogeneous* transmission rates on ports and thus, the downlinks and uplinks are the only sources of contention. Note that the heterogeneity on network bandwidth brings novelty, as well as new challenges, to the problem. The download and upload network bandwidth of each cloudlet S_i is upper-bounded by B_i^{in} and B_i^{out}, respectively.

Our goal is to design an optimizer that takes task requests as input and determines the combination of choices for the following factors: the cloudlet each task is assigned to and the order in which tasks are scheduled for both cross-cloudlet data transfer and computation at each cloudlet. We call this problem ASCO – Assigning and Scheduling for Cost Optimization.

[1]Admission control is used to block tasks out if there is not enough resource for those tasks in order to avoid performance compromise. If blocked, a task can remain at the mobile device or be offloaded to a remote cloud.

B. Task Characterization

We are given a set of n tasks $\mathbf{J} = \{J_1, J_2, ..., J_n\}$ that are offloaded from mobile devices and need to be executed in the aforementioned set of cloudlets. Each task is associated with a piece of data that has already been stored in the closest cloudlet. The parameters for describing each task $J_j \in \mathbf{J}$ are given by a five-tuple $\langle a_j, b_j, l_j, d_j, S_{d,j}\rangle$, where a_j, b_j are the arriving time and deadline, respectively, l_j is the computation workload, d_j is the volume of input data, and $S_{d,j}$ is the cloudlet that currently stores the associated data. Each task has to be completed before its deadline for guaranteed user experience. Note that in order to achieve resource efficiency and to reduce operational cost, it is not necessary to have a task to be executed in the closest cloudlet where the data for the task resides. As a result, if a task is assigned to a cloudlet that fortunately stores its associated data, it will be scheduled and executed directly; otherwise, the data for the task has to be first transferred to the assigned cloudlet and then, the execution of the task is carried out.

The optimizer will first need to make decisions on which cloudlet to assign each task to. We denote by $x_{i,j} \in \{0, 1\}$ the decision variable indicating whether task J_j is assigned to cloudlet S_i, where

$$x_{i,j} = \begin{cases} 1 & \text{if } J_j \text{ is assigned to } S_i, \\ 0 & \text{otherwise.} \end{cases} \quad (1)$$

We also denote by $y_j \in \{0, 1\}$ the indicator for whether the assigned cloudlet S_i for task J_j is the same as the one $S_{d,j}$ that stores the data for this task, i.e., $y_j = 1$ if $S_i = S_{d,j}$; $y_j = 0$ otherwise.

C. Total Cost

We denote by H the combined cost for executing a set of tasks $\mathbf{J}$. The cost H_i we consider here for each cloudlet S_i consists in two parts: energy cost E_i and network cost N_i. We assume that the energy cost is linearly related to the computation workload, which is a fair approximation in reality, and the energy price is given by P_i (i.e., energy cost per unit of computation) at cloudlet S_i. The electricity cost varies in different cloudlets as in different geographical locations the electricity generation cost can be different [8]. The network cost N_i is proportional to the volume of data being transferred across cloudlets which covers both upload and download traffic. To stay generic we assume that the prices for sending and receiving unit of data at the same cloudlet are not identical and both prices can also be heterogeneous among all cloudlets. Denote by Q_i^{in} and Q_i^{out} the download and upload network price at cloudlet S_i, respectively.

For a given task J_j being processed in cloudlet S_i, the energy cost can be expressed simply by $P_i \cdot l_j$, while the network cost is captured by $y_j \cdot d_j \cdot (Q_{d,j}^{out} + Q_i^{in})$. Denoting by $\mathbf{J}_i$ the set of tasks that are assigned to cloudlet S_i, i.e.,

$\mathbf{J}_i = \{J_j \mid J_j \in \mathbf{J} \wedge x_{i,j} = 1\}$, the total cost H_i at cloudlet S_i is given by

$$H_i = \sum_{J_j \in \mathbf{J}_i} \left(P_i \cdot l_j + y_j \cdot d_j \cdot (Q_{d,j}^{out} + Q_i^{in})\right). \quad (2)$$

Note that both the computation and network costs will be obtained in the monetary form so we omit the scaling factor. The total cost H of the system is given by the sum of the costs at all cloudlets in the system, i.e., $H = \sum_{S_i \in \mathbf{S}} H_i$, which we aim to minimize.

D. Problem Formulation and Complexity

The goal of the ASCO problem is to assign tasks to proper cloudlets such that the total cost H is minimized, while the deadlines of all the tasks can be respected. We denote by t_0 and t_1 the earliest arriving time and the latest deadline of the tasks in set $\mathbf{J}_i$, respectively, i.e., $t_i^a = \min\{a_j \mid J_j \in \mathbf{J}_i\}$ and $t_i^b = \max\{b_j \mid J_j \in \mathbf{J}_i\}$. Given an arbitrary non-empty subset $\widehat{\mathbf{J}}_i$ of $\mathbf{J}_i$, we denote by $\widehat{t_i^a}$ and $\widehat{t_i^b}$ the corresponding earliest arriving time and the latest deadline for the tasks in $\widehat{\mathbf{J}}_i$. The minimum total computation time for all the tasks in $\widehat{\mathbf{J}}_i$ is given by

$$\sum_{J_j \in \widehat{\mathbf{J}}_i} \frac{l_j}{C_i}, \quad (3)$$

while the minimum total communication time[2] taken by transferring the data for the tasks in $\widehat{\mathbf{J}}_i$ can be represented by

$$\sum_{J_j \in \widehat{\mathbf{J}}_i} y_j \cdot \frac{d_j}{\min(B_{d,j}^{out}, B_i^{in})}. \quad (4)$$

Putting everything together, the optimization problem ASCO can be further formulated as the following mixed integer program.

$$(P_1) \quad \min H$$

subject to

$$(3) + (4) \leq \widehat{t_i^b} - \widehat{t_i^a} \qquad \forall \widehat{\mathbf{J}}_i \subseteq \mathbf{J}_i$$

$$\sum_{S_i \in \mathbf{S}} x_{i,j} = 1 \qquad \forall J_j \in \mathbf{J}$$

$$x_{i,j} \in \{0,1\} \qquad \forall S_i \in \mathbf{S}, \forall J_i \in \mathbf{J}$$

The first inequality ensures that all the tasks can be completed before their deadlines at the assigned cloudlet. The second constraint forces that every task is assigned to one and only one cloudlet, while the last constraint is the binary constraint for the decision variable $x_{i,j}$.

Non-surprisingly, the ASCO problem can be proven to be NP-hard. Moreover, we observe that even deciding whether there exists a feasible solution to the ASCO problem, regardless of the total cost, is already NP-complete.

Theorem 1: Deciding whether there is a feasible solution for ASCO is NP-complete.

[2]For the sake of tractability, this minimum time is computed based on the assumption that cloudlets can only serve one data transfer at a time.

Proof: The goal is deciding whether there is an assignment of tasks to cloudlets, together with a schedule of the tasks at all cloudlets, such that the deadlines of all the tasks are respected in the ASCO problem, regardless of the total cost. We call this problem F-ASCO. The proof can be conducted by a polynomial time reduction from the classical Minimum Makespan Scheduling (MMS) problem, the decision version of which is NP-complete even if there are only two identical machines [9]. We start from an MMS instance where we are given a set of identical machines indexed by the set $M = \{1, ..., m\}$ and a set of jobs indexed by the set $J = \{1, ..., n\}$ to be assigned to the machines. Each job contains a certain workload w_j ($j \in [1, n]$) to be processed. The goal is to assign and process the jobs on the machines and the objective is to minimize the makespan, i.e., the maximum completion time of the machines. We denote by OPT_0 the minimum makespan that can be achieved.

From the above MMS instance we now construct an instance for the F-ASCO problem. We assume each machine in M represents a cloudlet S_i and we have in total n cloudlets given by the set $\mathbf{S}$. Each job j in J represents a task $J_j \in \mathbf{J}$ where $\mathbf{J}$ denotes the set of tasks for the F-ASCO instance. For all the tasks in $\mathbf{J}$, we assume they arrive at the same time and have the same deadline as OPT_0. We also assume that the network bandwidths at every cloudlet are infinite and the time used for data transmission is thus negligible. The question we need to answer in the F-ASCO problem instance then becomes to correctly decide whether there exists a schedule for the tasks to cloudlets such that all the tasks can be completed within OPT_0. It is easy to confirm that our answer to the F-ASCO instance is YES if, and only if, we solve the MMS instance and find its optimal solution. This completes the proof. ■

III. The Algorithm

Since we cannot efficiently have any guarantee on the existence of feasible solutions, we introduce *admission control* to block some task requests when there is no enough computation or network resource to serve them. This is a typical approach used for cloudlet operators to shape the system workload and make optimal decisions on resource utilizing [7]. The problem is then relaxed and the optimal cost can be achieved at the risk that there might be tasks that cannot be accommodated by any cloudlet. In the following, we will show how to achieve the best cost efficiency by cloudlet selection for tasks and how to improve admission rate by carrying out a well-designed scheduling algorithm for joint cross-cloudlet data transfer and computation.

A. Cloudlet Selection

The incentive to transfer the data for a task from one cloudlet to another is to reduce the total cost for processing the task. As a result, the following two necessary (yet not sufficient) conditions have to be met: i) Data transfer is possible subject to time limit; ii) The total cost for processing the task is reduced as a result of the data transfer.

Assume there is a task $J_j \in \mathbf{J}$ and the data for this task is stored at cloudlet $S_{d,j}$. According to the above two conditions, a candidate cloudlet $S_c \in \mathbf{S} \backslash S_{d,j}$ that can host task J_j has to satisfy the following inequalities.

$$\frac{l_j}{C_c} + \frac{d_j}{\min(B_{d,j}^{out}, B_c^{in})} \leq b_j - a_j \tag{5}$$

$$l_j \cdot P_c + d_j \cdot (Q_{d,j}^{out} + Q_c^{in}) < l_j \cdot P_i \tag{6}$$

Based on the above two conditions, for each task J_j we carry out a screening process, which aims at removing the cloudlets that cannot host the task. We denote by $\mathbf{S}_j$ the set of the valid candidate cloudlets for task J_j. If $\mathbf{S}_j = \emptyset$, the task will be by default assigned to cloudlet $S_{d,j}$; otherwise we choose the cloudlet from all the candidate cloudlets $\mathbf{S}_j$ that gives the minimized cost for executing the task (including the cost for data transfer). We denote the chosen cloudlet by $S_{p,j}$. The assignment of tasks will be completed when we finish repeating the above process for every task.

B. Two-phase Task Scheduling

Once we have decided the cloudlet that each task will be preferably assigned, the problem becomes how to schedule the data transfer and computation for the tasks at each cloudlet. Having in mind that there might be tasks that are not admitted into the cloudlet system, our implicit objective for the scheduling would be to accommodate as many tasks as possible in order to maintain higher user satisfaction.

We first describe a similar problem that has been widely studied in traditional scheduling literature: We are given three sets of parallel machines, denoted by $\mathbf{A}, \mathbf{B}$ and $\mathbf{C}$, respectively. Note that the machines in each set can be heterogeneous with non-uniform processing capabilities. We are also given a set of jobs, each of which consists of three operations that have to be carried out on the three sets of machines stage by stage, respectively. The objective of the problem is to minimize the total completion time of all the jobs. If the operations of each job can be flexibly assigned to any of the machines in the corresponding class, the problem is called *Flexible Flow Shop Scheduling with Parallel Machines* and it has been shown to be strongly NP-hard even when the machines are uniform [10].

Our problem inherits the same problem structure but differs in the following two aspects: (1) The machine for carrying out the operation of each task in every stage is fixed. (2) The machines for carrying out the first two operations of each task are coupled, i.e., they will be occupied at the same time. The problem under the two constraints is still NP-hard. The proof can be straightforwardly conducted by a reduction from the traditional flow shop scheduling problem, which is known to be NP-hard with at least two machines (three machine sets in our case). Our objective, however, instead of minimizing the total task completion time, is to maximize the number of tasks that could be completed before given deadlines. We call this problem *Flow Shop Scheduling with Coupled Resources (FSS-CR)*. The term *coupled resources* implies that the scheduling decisions for the first two operations of each task have to be done simultaneously.

C. FSS-CR

The high complexity of the problem implies a vast searching space and thus, we seek for efficient heuristics that could generate comparably good results within very short time. Under the objective of accommodating as more tasks as possible, our algorithm aims at reducing the time wasted at every stage due to resource contention.

While sharing a common deadline, the decision making process for scheduling each task can be divided into two independent phases: `dtrans` (data transfer, the first two stages) and `comp` (computation, the third stage). We notice that once we fix the scheduling for the tasks in phase `comp`, the deadlines for phase `dtrans` of tasks can be accordingly determined. This is achievable as the set of tasks that will be processed at each cloudlet is already known after the cloudlet selection process described before. The problem is then decomposed into two sub-problems, i.e., scheduling for `comp` and scheduling for `dtrans`.

1) Scheduling for Computation: As in phase `comp` tasks will be processed independently at each cloudlet, we focus on an arbitrary cloudlet $S_i \in \mathbf{S}$. The main idea behind this is to ensure that all the tasks at this cloudlet can be completed before their deadlines and every task starts its computation phase as late as possible to make time for the data transfer phase as resource contention is more severe in data transfer due to resource coupling. To this end, we define an auxiliary problem called *Reverse-Task Scheduling (RTS)*. Given a set of tasks with deadline constraints to be processed on a single machine, the goal of the RTS problem is to decide the order of tasks to be processed such that the average starting time of all the tasks is maximized.

The RTS problem can be transformed into a single-machine task scheduling problem by treating task deadlines as arriving times and reversing the task execution from the end to the beginning. As a result, the problem becomes that given a set of tasks that arrive at a single machine, we design a schedule for the tasks such that the average completion time is minimized. We observe that the preemptive scheduling policy Shortest Remaining Time First (SRTF) gives the minimal average completion time compared to non-preemptive policies First Come First Serve (FCFS) and Earliest Deadline First (EDF). Consequently, the total time saved in the `comp` phase would be maximized.

2) Scheduling for Data Transfer: The starting time determined for each task in phase `comp` will provide a strict deadline for each task in phase `dtrans`. Given this deadline, the problem of scheduling for data transfer becomes a network flow scheduling problem: each task $J_j \in \mathbf{J}$ represents a flow F_j with size of d_j from cloudlet $S_{d,j}$ to cloudlet $S_{p,j}$, with arriving time a_j and deadline b_j. Our goal is to schedule the flow transmissions (i.e., sending or receiving) on a semi-clos network while guaranteeing flow deadlines. Our solution to this problem is based on two iterative processes where the first process is admission control based on pruning, while the second one is scheduling. Before presenting the algorithm, we first carry out a flow size normalization process.

Flow Size Normalization. Normalization is necessary for computing the most intensive time interval, as the maximal transmission rate for each flow can be different due to the fact that network nodes (i.e., cloudlets) we assumed here can have different capacities. The normalization process is straightforward, i.e., assuming the capacity of each network port as one. To this end, for each flow we divide its size by the maximal transmission rate it can achieve when being transmitted on the network. More formally, the normalized size of flow F_j is given by

$$|F_j| = \frac{d_j}{\min(B^{out}_{d,j}, B^{in}_{p,j})} \tag{7}$$

where $B^{out}_{d,j}$ and $B^{in}_{p,j}$ are the egress bandwidth of cloudlet $S_{d,j}$ and the ingress bandwidth of cloudlet $S_{p,j}$, respectively.

Admission Control. The admission control process is designed to prune the flow set by removing the flows that will not be able to be completed, such that the feasibility of the transmissions of the residual flows can be guaranteed. Denote by $\mathbf{F}_i$ the set of flows that will be routed through cloudlet S_i. We first provide the following definition.

Definition 1: The intensity of a cloudlet S_i in a given time interval $I = [a, b]$ is defined as the average normalized amount of data to be transmitted by S_i in this interval, i.e.,

$$\delta(S_i, I) = \frac{\sum_{[a_j,b_j]\subseteq[a,b]\wedge F_j\in\mathbf{F}_i} |F_j|}{a \sim b} \tag{8}$$

where $a \sim b$ denotes the total time in which cloudlet S_i is free.

It is intuitive that $\delta(S_i, I)$ has to satisfy $\delta(S_i, I) \leq 1$, meaning that the maximal intensity is constrained by the normalized capacity of each cloudlet; otherwise, there will be flows that cannot meet their deadlines. Our design for admission control is based on the inequality $\max\{\delta(S_i, I)\} \leq 1$.

Definition 2: For a given cloudlet S_i, a time interval $I = [a, b]$ is defined as a critical interval if it maximizes $\delta(S_i, I)$.

Definition 3: The most critical time interval $I^* = [a^*, b^*]$ is defined as the time interval that maximizes $\delta(S^*_i, I^*)$ among all cloudlets. Cloudlet S^*_i is the corresponding most critical cloudlet and flow set $\mathbf{F}^*_i = \{F_j \in \mathbf{F}_i \wedge [a_j, b_j] \subseteq I\}$ is the corresponding critical flow set.

Based on the above definitions, the pruning process works iteratively as follows: in each iteration we search for the most critical time interval I^*. Once this interval has been found, we check if feasibility can be satisfied in this interval, i.e., whether or not $\delta(S^*_i, I^*) \leq 1$. If not, we remove the flow $F_j \in \mathbf{F}^*_i$ which has the maximized $d_j/(a_j \sim b_j)$, meaning it contributes the most to the intensity of interval I^*. By removing this flow, the intensity of this interval will be reduced. We repeat the above process until $\delta(S^*_i, I^*) \leq 1$ is satisfied.

Most Critical First with EDF. We now discuss how to decide the schedule for the rest flows. We design an algorithm MCF-EDF (Most Critical First with EDF) for flow scheduling, which is listed in Algorithm 1. The algorithm is conducted on an iterative process: in each iteration, MCF-EDF first finds the most critical interval I^* and its corresponding most critical cloudlet S^*_i. The flows that fall into interval I^* is denoted by set $\mathbf{F}^*_i = \{F_j \mid F_j \in \mathbf{F}_i \wedge [a_j, b_j] \subseteq [a, b]\}$. After that, we schedule the flows in the interval I^* using the EDF policy, from which the spanning time $[a'_j, b'_j]$ of each flow $F_j \in \mathbf{F}^*_i$ will be determined. Finally, we update the available time intervals on all the cloudlets that have been affected, i.e., cloudlets that contain flows from set $\mathbf{F}^*_i$ that have just been scheduled in this iteration. The algorithm terminates when all the flows have been scheduled.

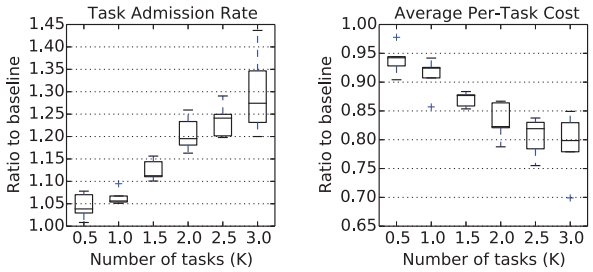
Algorithm 1 MCF-EDF

while $\exists S_i \in \mathbf{S}, \mathbf{F}_i \neq \emptyset$ **do**
  $(S^*_i, I^*) \leftarrow \text{argmax}_{(S_i,[a,b])} \{\delta(S_i, [a, b])\}$
  $\mathbf{F}^*_i = \{F_j \mid F_j \in \mathbf{F}_i \wedge [a_j, b_j] \subseteq [a, b]\}$
  for $F_j \in \mathbf{F}^*_i$ **do**
    $a'_j = \sum_{F_k\in\mathbf{F}^*_i \wedge b_k<b_j} |F_k|$
    $b'_j = \sum_{F_k\in\mathbf{F}^*_i \wedge b_k<b_j} |F_k| + |F_j|$
  end for
  for $F_j \in \mathbf{F}_i$ **do**
    $\mathbf{F_i} \leftarrow \mathbf{F_i} \backslash F_j$ for $F_j \in \mathbf{S_i}$
    Mark $[a'_j, b'_j]$ as unavailable on cloudlet S_i if $F_j \in \mathbf{F}_i$
  end for
end while

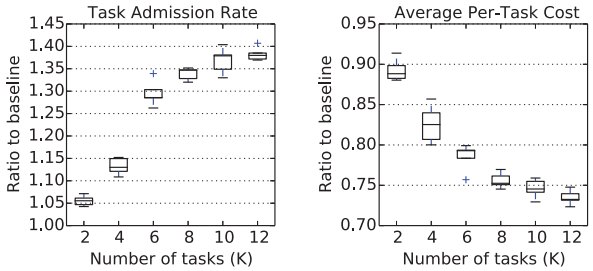
IV. Performance Evaluation

We developed a discrete-time multi-cloudlet simulator in Python, with the proposed assignment and scheduling algorithms implemented. The simulator exposes interfaces for various parameters in our model for both the cloudlet and the task. Serving as a primitive evaluation, the simulations use values for the parameters that were generated following random distributions. For cloudlets, the processing capacity, as well as the upload and download network bandwidths, follows uniform distributions, while the prices for computation and communication follow normal distributions. For tasks, the arrival time and deadline follow a uniform distribution while the volume of input data as well as the computation workload follow normal distributions.

We compare the proposed algorithm with a baseline approach where tasks are assigned to the closest cloudlets and the scheduling of tasks is done following an FCFS manner complemented with EDF for tasks that arrive simultaneously. This baseline is considered to be the *de facto* approach used in current cloudlet systems. We focus on two aspects of interest: admission rate and average per-task cost. The experimental results are shown in Fig 2. All the values are normalized by the ones we obtained from the baseline approach and the values are averaged over five independent runnings. We tested with both a small scale (with 20 cloudlets) and a large scale (with 100 cloudlets) scenarios. Experiments in both scenarios confirm that (i) the admission rate is significantly improved (up to 30%) in our approach and (ii) the proposed assignment and scheduling algorithm together can help reduce largely the average per-task cost (up to 20%). This is mainly due to the fact that the assignment and scheduling algorithm expand the solution space by enabling data transfer between cloudlet pairs



(a) Small scale with 20 cloudlets



(b) Large scale with 100 cloudlets

Fig. 2. Comparisons on task admission rate and average per-task cost in the cloudlet system in different scales.

and by introducing elaborate scheduling mechanisms for joint data transfer and task execution.

V. Related Work

The concept of cloudlet was introduced by Satyanarayanan et al. [4] where they discussed the possibility of exploiting virtual machine (VM) technology to rapidly instantiate customized services on a nearly by cloudlet for mobile users. The VM-based idea was then extended to multiple mobile task offloading proposals including ThinkAir [3], Virtual Smartphone [11], and CloneCloud [1]. The generality of offloading was further improved by allowing runtime dynamic decision making [12] and support multi-threaded and user-interactive applications [13], [14]. Most recently, Chen et al. [15] explored efficient wireless access coordination among multiple mobile device users. Tong et al. [16] studied the tradeoff between energy efficiency and responsiveness of mobile applications by traffic scheduling for single cloudlets. Jia et al. [5] investigated the problem of balancing the load of multiple cloudlets to optimize mobile application performance. A novel hierarchical architecture design with multiple cloudlets for mobile edge clouds was also proposed in [6].

In contrast, we study the impact of both task assignment and scheduling on the overall operational cost of multi-cloudlet based mobile edge clouds. We aim to optimize per-task cost and to ensure quality of experience by enforcing hard deadlines for offloaded tasks through joint task assignment and scheduling in multi-cloudlet environments. This consequently leads to the best operation of cloudlets in terms of both load distribution and cost optimization for mobile edge cloud operators (i.e., cloudlet operators) as we claimed at the beginning of this paper.

VI. Conclusion

This paper studied the problem of cost reduction in mobile edge clouds through assignment and scheduling for mobile offloaded tasks. We provided the formulation of the problem as well as thorough analysis on its computational complexity. By introducing admission control, we simplified the problem and proposed efficient algorithm for scheduling tasks for maximized admission rate. The task scheduling consists of two coupled phases namely data transfer and computation, thus the proposed algorithm first decouples the two phases and then, it generates efficient scheduling for each of the phases. Simulations in both small and large scales confirmed the hypothesis that considerable reduction on average per-task cost can be achieved by reconciling task assignment and scheduling in mobile edge clouds. It is also of interest to explore the possibility of partitioning tasks and assigning each partition to one of the cloudlets. We leave it as future work.

Acknowledgment

This research was partially supported by the German Research Foundation (DFG) within the Collaborative Research Center (CRC) 1053 – MAKI and by the National Science Foundation of China (NSFC) Key Program 61520106005.

References

[1] B. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: elastic execution between mobile device and cloud," in *EuroSys*, 2011.
[2] M. V. Barbera, S. Kosta, A. Mei, and J. Stefa, "To offload or not to offload? the bandwidth and energy costs of mobile cloud computing," in *INFOCOM*, 2013.
[3] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, "Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading," in *INFOCOM*, 2012.
[4] 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.
[5] M. Jia, W. Liang, Z. Xu, and M. Huang, "Cloudlet load balancing in wireless metropolitan area networks," in *INFOCOM*, 2016.
[6] L. Tong, Y. Li, and W. Gao, "A hierarchical edge cloud architecture for mobile computing," in *INFOCOM*, 2016.
[7] L. Zheng, C. Joe-Wong, C. Tan, M. Chiang, and X. Wang, "How to bid the cloud," in *SIGCOMM*, 2015, pp. 71–84.
[8] H. Xu, C. Feng, and B. Li, "Temperature aware workload managementin geo-distributed data centers," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 6, pp. 1743–1753, 2015.
[9] R. G. Michael and S. J. David, "Computers and intractability: a guide to the theory of np-completeness," *WH Free. Co.*, 1979.
[10] G. J. Kyparisis and C. Koulamas, "Flexible flow shop scheduling with uniform parallel machines," *European Journal of Operational Research*, vol. 168, no. 3, pp. 985–997, 2006.
[11] E. Y. Chen and M. Itoh, "Virtual smartphone over IP," in *WOWMOM*, 2010.
[12] E. Cuervo, A. Balasubramanian, D. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "MAUI: making smartphones last longer with code offload," in *MobiSys*, 2010.
[13] M. S. Gordon, D. A. Jamshidi, S. A. Mahlke, Z. M. Mao, and X. Chen, "COMET: code offload by migrating execution transparently," in *OSDI*, 2012.
[14] M. Ra, A. Sheth, L. B. Mummert, P. Pillai, D. Wetherall, and R. Govindan, "Odessa: enabling interactive perception applications on mobile devices," in *MobiSys*, 2011.
[15] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Transactions on Networking*, 2016.
[16] L. Tong and W. Gao, "Application-aware traffic scheduling for workload offloading in mobile clouds," in *INFOCOM*, 2016.