Journal of Network and Computer Applications 167 (2020) 102735

Contents lists available at ScienceDirect

Journal of Network and Computer Applications

journal homepage: www.elsevier.com/locate/jnca

Data-intensive application scheduling on Mobile Edge Cloud Computing[☆]

Mohammad Alkhalaileh[*], Rodrigo N. Calheiros, Quang Vinh Nguyen, Bahman Javadi

School of Computer, Data and Mathematical Sciences, Western Sydney University, Australia

ARTICLE INFO

Keywords:
Mobile Edge Cloud
Data-intensive offloading
Resource allocation
Mobile energy
Application scheduling

ABSTRACT

Mobile cloud computing helps to overcome the challenges of mobile computing by allowing mobile devices to migrate computation-intensive and data-intensive tasks to high-performance and scalable computation resources. However, emerging data-intensive applications pose challenges for mobile cloud computing platforms because of high latency, cost and data location issues. To address the challenges of data-intensive applications on mobile cloud platforms, we propose an application offloading optimisation model that schedules application tasks on an integrated computation environment named Mobile Edge Cloud Computing. The optimisation model is formulated as a mixed integer linear programming model, which considers both monetary cost and device energy as optimisation objectives. Moreover, the allocation process considers parameters related to data size and location, data communication costs, context information and network status. To evaluate the performance of the proposed offloading algorithm, we conducted real experiments on the implemented system with a variety of scenarios, such as different deadline and multi-user parameters. Our results demonstrate the ability of the proposed algorithm to generate an optimised resource allocation plan in response to dramatic fluctuations in application data size and network bandwidth. The proposed technique reduced the execution cost of data-intensive applications by an average of 46% and 76% in comparison with particle swarm optimisation (PSO) and full execution on a mobile device only, respectively. In addition, our new technique reduced mobile energy consumption by 35% and 84%, compared to PSO and full execution on a mobile device only, respectively.

1. Introduction

The use of mobile devices, such as smartphones and tablets, has increased markedly, due to advancements in functionality supported by features such as high connectivity, faster central processing unit (CPU), large memory and sophisticated sensors. In 2019, 46% of Internet traffic originated from mobile connected devices, with an estimated 30.6 exabytes generated monthly (C. V. N. I. Cisco, 2014). Although mobile devices are now equipped with high-performance computation resources, they still face significant challenges in meeting the requirements of computation-intensive and data-intensive mobile applications.

Mobile Cloud Computing (MCC) has been widely accepted as a computation architecture that overcomes the shortcomings of mobile computing. MCC involves the integration of mobile devices and public cloud resources to provide computation and storage services for mobile applications in a scalable manner, while providing a high quality of service (QoS) (Samimi et al., 2006). MCC aims to augment mobile devices by improving and optimising the devices' computing capabilities by performing computationally intensive tasks using cloud-based resources (Zhou et al., 2015a), and involves migrating resource-intensive computations from smartphone devices to the cloud via wireless communication technologies, a process referred to as the computation offloading concept. However, MCC has two significant limitations: 1) the long propagation distance from mobile device to a remote cloud centre, which results in excessively long latency for mobile applications, and 2) insufficient computation and storage at network edges to enable ubiquitous mobile computing. An alternative computing model that handles the latency and resource utilisation issues is Mobile Edge Computing (MEC) (Mach and Becvar, 2017).

Mobile Edge Computing is an emerging paradigm designed to meet the ever-increasing computation demands of mobile applications (Mao et al., 2016). Offloading some computing tasks to the edge improves energy efficiency due to the huge data transfer saving, particularly with mobile network usage (Shi et al., 2016). Most studies of offloading opti-

☆ Fully documented templates are available in the elsarticle package on CTAN.
* Corresponding author.
E-mail addresses: mohammad.nour@westernsydney.edu.au (M. Alkhalaileh), r.calheiros@westernsydney.edu.au (R.N. Calheiros), q.nguyen@westernsydney.edu.au (Q.V. Nguyen), b.javadi@westernsydney.edu.au (B. Javadi).

https://doi.org/10.1016/j.jnca.2020.102735
Received 10 September 2019; Received in revised form 13 May 2020; Accepted 26 May 2020
Available online 24 June 2020

misation in MEC focus on finding an optimal application tasks distribution strategy to edge resources that considers minimising data transfer latency with respect to the capacity of edge nodes (Enzai and Tang, 2016; Jararweh et al., 2017; Mach and Becvar, 2017; Goudarzi et al., 2017; Patel et al., 2014; Abbas et al., 2017; Chen and Hao, 2018). The main feature of MEC is to push mobile computing, network control and storage from resource-limited mobile devices to the network edges to enable computation-intensive and latency-critical applications (Mao et al., 2017). With MEC, a mobile application can track real-time information such as behaviours, location and environment, which reduces the exchange of sensitive information between the mobile device and cloud resources, while being more energy efficient. However, edge resources are limited in computation capacity, scalability and high-energy sensitivity to perform long-term computation. One solution is integrating edge and cloud resources to take advantage of the capabilities of cloud resources and the availability of access resources at the edge layer. This joint computation architecture is referred to as Mobile Edge Cloud Computing (MECC) (Ren et al., 2019).

In this paper, we propose multi-objective data-intensive mobile application scheduling and allocation optimisation in an MECC computation environment. The contributions of this work include:

- Multi-objective optimisation of device energy and monetary cost in the MECC environment under the constraints of available mobile device energy and task deadline;
- Formulation of joint task offloading and resource allocation optimisation for multi-user mobile applications using mixed integer linear programming (MILP), considering both monetary cost and device energy as optimisation objectives; and
- A multi-perspective analysis of data-intensive application optimisation based on deadline and multi-user constraints.

This paper is structured as follows. In Section 2, we review related work on task scheduling and allocation optimisation using MCC, MEC and MECC. Section 3 presents an overview of the proposed system architecture. System modelling and problem formulation are presented in Section 4, while the optimisation technique and the proposed offloading algorithm are explained in Section 5. The model's performance evaluation and experimental results are discussed in Section 6, and Section 7 provides conclusions and future work.

2. Related work

In this section, we review the existing work on task scheduling and allocation optimisation using MCC, MEC and MECC from different perspectives and based on different optimisation objectives, including, energy, cost, latency and multi-user implementation. Mobile device input/output processing and network communications are energy-hungry components (Abolfazli et al., 2014), and offloading heavy tasks to the high-performance computation can substantially reduce energy consumption. Many techniques have been proposed for overcoming the challenge of energy-intensive applications, such as device augmentation (Abolfazli et al., 2014), virtual machines (VM) cloning (Chun et al., 2011), computation migration (Cuervo et al., 2010) and code decomposition and component reusability (March et al., 2011). Despite their capacity to fulfil the requirements of some applications, like peer-to-peer gaming and low-scale image processing, these techniques encounter major challenges in supporting large-scale and complex applications with unpredictable number of users, exchanged data size, network bandwidth and corresponding data communication costs.

Mobile/wireless network performance makes a significant contribution to mobile application responsiveness and processing time (Armbrust et al., 2010). Hung et al. (2012) argued that a high percentage of mobile users would prefer to run mobile applications locally due to network performance implications. The decision to run an application locally or remotely is complicated and requires steady monitoring of network conditions and application profiling (Giurgiu et al., 2009). Many researchers have focused on resolving the issues of instability of network bandwidth and data communication quality. Satyanarayanan et al. (2009) proposed an architecture to improve application responsiveness and availability by relying on high-performance cloudlets, which have good connectivity with cloud servers. However, one limitation of cloudlets is their limited convergence on the mobile network for service provisioning, which does not support a high number of mobile users sharing available resources (Chen et al., 2015). Chen et al. (2015) adopted MEC architecture and applied game theory to find optimised application scheduling based on latency and energy constraints. Jararweh et al. (2017) addressed the same optimisation problem using MILP in an edge-only resource environment. Cardellini et al. (2016) designed a game-theoretic approach to computation offloading in MECC and used the waiting time in computation nodes to separate the execution of multi-user applications. These researchers proposed efficient techniques for adopting cloudlet/edge computing to enhance mobile application responsiveness, reduce latency and optimise device energy. However, they did not attempt to integrate between MCC and MEC, and in particular did not propose practical and adequate resource allocation and task scheduling techniques in the context of data-intensive mobile applications. Moreover, the techniques they discussed do not related to offloading decision optimisation behaviour based on data-aware parameters, such as application data size and location.

Processing large data files on mobile devices has a direct effect on device energy, whereas transferring large data files over mobile networks can increase the device idle time, as well as the total computation cost and time. Processing on close edge nodes is more efficient for reducing data transfer latency and capturing more real-time context information (Patel et al., 2014). Multi-user edge computing can improve MEC by providing efficient management and fair distribution of edge resources. Zhao et al. (2015) designed a threshold-based policy to improve the QoS of MEC, involving the combination of local edge resources with public cloud resources, which takes advantage of the low latency of a local cloud and abundant computational resources of public clouds simultaneously. Enzai et al. (Enzai and Tang, 2016) developed a heuristic algorithm for multi-site computation offloading in MECC. The work considered multiple objective optimisations of time, cost and energy. The idea was to transform the multi-weight optimisation to single-weight using a heuristic approach. Vu et al. (2019) proposed a joint task offloading and resource allocation optimisation problem, aiming to minimise the energy consumption for mobile devices under the fog nodes, resource constraints and task delay requirements. In the context of data-intensive mobile applications, some issues related to MECC are not clearly resolved. Firstly, transferring huge amounts of data via unstable mobile networks will lead to an unpredictable increase in data transfer latency. Secondly, edge resources have limited capability for processing application tasks with high data input. Finally, transferring massive data over cellular networks and processing in public clouds incurs significant costs for application users.

Data-intensive applications, such as customised data analytic services, natural language processing and face recognition, are resource-intensive applications that require machines with powerful CPU and huge memory space to load dynamic application code and data. Wang et al. (2015) argued that data-intensive application computations on mobile computing are always costly in terms of computation time, mobile energy and resource cost. Nan et al. (2011) studied the challenges of data-intensive-aware mobile applications. The study highlighted the significance of increasing data size on monetary cost and QoS improvement. Zhou et al., 2015 proposed a three-tier MCC middleware that empowers programmers with computation alternatives, based on the application cost model and the offloading decision-maker. Even though the proposed model includes the task data size in generating an optimised execution application task plan, the data size is relatively small and cannot reflect the scenario of data-intensive application task scheduling on MCC. Abbas et al. (2017) proposed an offloading optimisation technique-based execution path to reduce execution latency for data-intensive mobile applications in MEC, and Terefe et al. (2016) proposed a data-intensive energy-efficient optimisation model on a hybrid-cloud environment. The results of both studies show how adopting a heterogeneous resource model can minimise energy consumption. The literature includes efforts to overcome the challenges of data-aware mobile application offloading; however, the proposed models do not consider the contribution of data size variation in association with other optimisation parameters in application offloading decisions. In addition, these models largely neglect the integration of cloud and edge computing to resolve data-intensive application offloading optimisation and fail to address certain other issues. These include the efficient adoption of MEC to overcome latency-sensitive application when transferring and processing large data files and offloading optimisation in response to new scenarios, such as deadline and multi-user models. In this paper, we propose an optimisation model for offloading data-intensive mobile applications in MECC. The model adopts a MILP technique to construct an allocation strategy, reducing energy consumption and total monetary cost on the mobile user device under the constraints of task-level deadline and available device energy.

3. System architecture

In this section, we discuss the adopted MECC architecture. A description of the MECC environment is provided, followed by the main optimisation engine components. Fig. 1 illustrates the proposed MECC resource model and optimisation framework. The MECC environment leverages three resource layers, namely, public cloud, edge and mobile devices. The public cloud is a powerful and scalable resource that allows for convenient processing and storage capabilities for computation-intensive and data-intensive tasks. At the edge layer, a user can access nearby computation and storage units with low latency time via mobile cellular networks (3G and 4G) or WiFi. In this research, we assume a low to medium computation power of edge machines in comparison to those in the public cloud. In addition, at the edge layer, we assume the availability of local data storage units that are capable of storing large application files. At the application level, we assume the ability to execute tasks locally on users' mobile devices under the constraint of available device energy.

The optimisation engine is responsible for handling all activities to construct the offloading/allocation plan for application tasks including profiling, cost and queuing estimation, and decision making. The optimisation process itself is not resource intensive. Both cloud and edge servers can be used to eliminate the energy overhead to perform the optimisation process on the user device. However, the edge is selected as it has lower latency for decision making than the cloud. The optimisation engine consists of six components categorised into three service levels: context data collection (application profiler and resource handler), optimisation variables estimation (cost and queue estimator) and offloading decision optimisation (QoS optimiser and decision-maker). These components are described as follows.

- *Application Profiler*: The framework offers three types of profiling, namely, energy profiling for energy usage, network profiling to monitor mobile network information, and application execution profiling to record heuristic data about application execution with awareness of contextual information about network and bandwidth. The application profiler keeps a record of application execution at task level. Heuristic data include type of application; context information including, device energy, network interface and bandwidth; and optimisation result, including execution time, cost, energy, and task allocation plan. The profiling data is stored in an accessible database to be shared with estimation components (cost and queuing estimators).
- *Resource Handler*: The resource handler works on collecting data regarding the current status of user device energy and edge resources. Collected data includes device energy level, in-use mobile network and current network bandwidth, number of available cores on mobile and edge nodes, and the connectivity status between the mobile device and edge nodes. The resource handler fetches resource status prior to the optimisation process to be recorded by

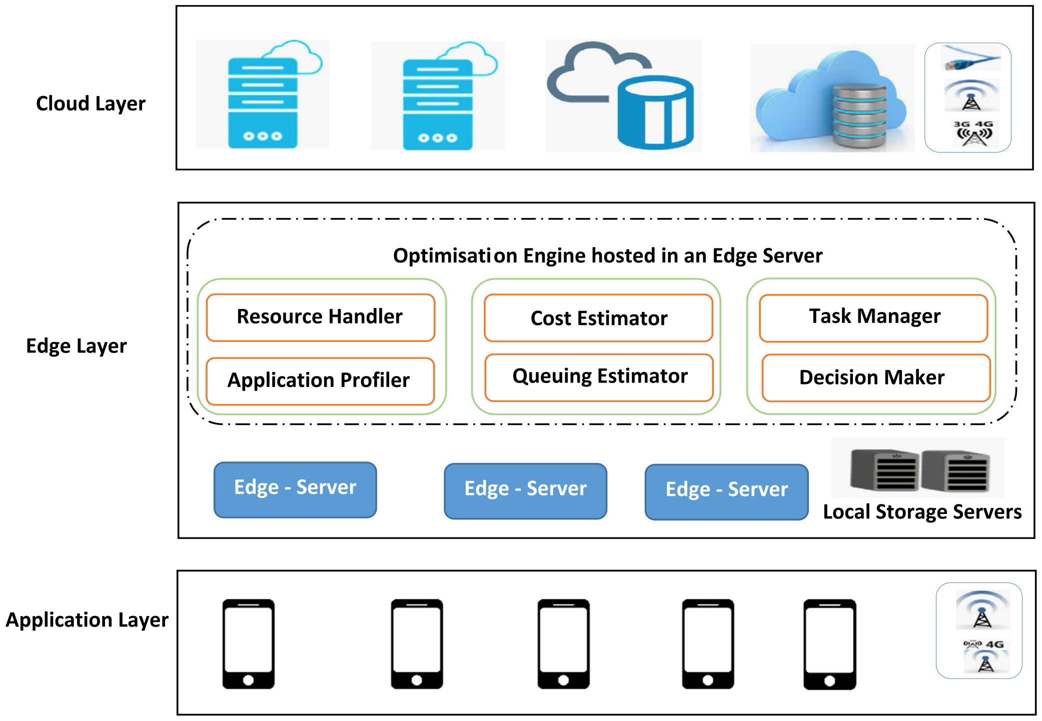


Fig. 1. Mobile Edge Cloud Computing (MECC) framework.

the application profiler. In addition, the resource handler is responsible for communicating with computation nodes and storage units to run the application according to the optimisation plan. In other words, the resource handler performs all communication aspects with external entities for computation, storage and profiling.

- *Queuing Estimator*: We assume a limited number of cloud servers and edge nodes. Thus, a queuing system is adopted to support the application offloading decision and manage task offloading to these resources with consideration of server capability and task waiting time. The work represents computation servers at cloud and edge layers as G/G/1 queues (Marchal, 1976). The queue estimator computes the task waiting time in each processing queue, based on the current distribution of application tasks and the capability of the computation server (service time).
- *Cost Estimator*: The cost estimator handles the calculation of optimisation parameters, that is, execution time, monetary cost and expected energy consumption. The cost estimator communicates with the application profiler to get updated energy profiling parameters for task processing, data transfer and device waiting for external task processing.
- *Task Manager*: The task manager coordinates the execution of application tasks based on the offloading plan generated by the decision-maker. It sends task executions through the resource handler, and updates the application profiler with execution results.
- *Decision-Maker*: The decision-maker is responsible for generating execution plans (paths) and then selecting the optimised one, based on the optimisation constraints of available device energy and task deadline. The decision-maker runs the optimisation process based on resource data from the resource handler and estimation values from cost and queuing estimators.

4. Application model

This section describes the modelling of mobile application execution in an MECC system as a three-tier computing system of mobile devices (application user), edge resources and cloud server. Table 1 describes the mathematical notations used in the problem formulation.

4.1. Task model

We assume the representation of a data-intensive mobile application A as a Bag of Tasks (BoT), in which tasks are independent in computation and data sharing. The BoT application model allows a variety of data locations (to test the bandwidth impact on data transfer) and a variety of data sizes. In BoT, each task can have its own computation complexity, which allows better linkage between data-intensive and computation-intensive application features. BoT is a common application model for large-scale systems, such as computational biology, parameter sweeps, fractal calculations and data mining (Terzopoulos and Karatza, 2016). An application A is modelled as:

$$A = \{t_1, t_2, \dots, t_n\} \tag{1}$$

where n is number of tasks. A task t_i is modelled as:

$$t_i = \{L_i, s_i, I_i, \partial_i\} \tag{2}$$

We include data size and location parameters in task modelling to serve the objective of building a data-aware optimisation model for scheduling mobile application tasks in an MECC environment. To avoid the complexity of distributed file storage, we assume that each task is associated with a single file storage system and the file storage is consistently available for data collection and is capable of storing all task data files. The available storage in the edge layer is temporary and only for input data and temporary data needed for tasks. This assumption about data storage at the edge layer is illustrated in the proposed system architecture, Fig. 1. Data files are stored in temporary centralised local storage servers at the edge layer to handle the complexity of large data file access by computation resources, which allows the execution of data-intensive applications such as data analytics, online monitoring and social sensing. In addition, there is long-term storage in the cloud layer, which will be used to store the data required for all applications.

4.2. System model

The system assumes three computation environments: public cloud, M edge nodes and N mobile devices. A mobile device P_m is modelled as: $P_m = \{\beta, \beta_{cost}, d_m, e_m, w_m\}$. A mobile device is connected to an edge and the public cloud via WiFi or cellular networks. A remote computation node on edge or public cloud P_r is modelled as: $P_r = \{\beta, \beta_{cost}, p, w_r\}$. We assume that, in a given time slot, a mobile user can request application execution; some tasks will be executed locally (on the mobile user device) and others remotely (on an edge node or the cloud server). A joint offloading technique is adopted to schedule application tasks in the MECC computation environment. In addition, in this work, we assume that cloud and edge resources are available and pre-allocated (static provisioning) by our platform, (dynamic provisioning is an interesting aspect for future research). Thus, rather than adding cloud resources to process extra tasks arriving in the system, we adopt a queuing system in which tasks wait for resources. This model to minimises the monetary cost of cloud resources.

Next, we describe the cost estimation models involved in formulating our response to the mobile application scheduling and allocation problem. To find the application execution plan, three value estimations need to be calculated, namely, task execution time, total energy and total monetary cost.

Table 1
Problem modelling notation.

Symbol	Definition
t_i	Application task i
L_i	Task input file location, either local or remote
s_i	Task input size
I_i	The number of task execution instructions (MIPS)
∂_i	Task deadline
β	Available network bandwidth
β_{cost}	The monetary cost of data communication using a mobile network interface
w_i	Processing power for mobile devices or remote computation nodes
d_m	Mobile device storage (MB)
e_m	Mobile device available energy (J)
p	Cost of processing in a remote execution server ($/hour)
ω_i	task t_i data size sensitivity factor
l	the network latency

4.3. Task execution time model

The task execution time for task t_i is the sum of task processing time D_i^P in the target computation environment P_r or P_m, data communication time D_i^C and task average waiting time D_i^W for remote execution. However, the task processing time D_i^P depends on the number of task instructions I_i to run the task and the target computation unit power w_{target}. Moreover, we consider the task data size in calculating the task processing time. To do so, we introduce a data sensitivity factor ω_i, which is calculated as the ratio of change in processing time to the change in data size. To get an accurate measurement of the sensitivity factor, we analysed the correlation between the task input data size and the change in processing time. We conducted several experiments to reach a stable measurement with minimum variance on data sensitivity values.

$$D_i = D_i^P + D_i^C + D_i^W \tag{3}$$

$$D_i^P = \frac{I_i}{w_{target}} + (s_i \cdot \omega_i) \tag{4}$$

$$D_i^C = \frac{s_i}{\beta} + l \tag{5}$$

To compute the task waiting time for task t_i, a G/G/1 queuing model is adopted. The G/G/1 queue represents the queue length in a system with a single server where inter-arrival times have a general (arbitrary) distribution and service times have a (different) general distribution. We assume edge and public cloud computation resources as G/G/1 queues, in which the task arrival rate λ for processing follows a general distribution and the service time μ to process incoming tasks similarly follows a general distribution. Marchal (1976) proposed an approximation for G/G/1 as follows:

$$L_q \approx \frac{\rho^2(1+C_s^2)(C_a^2+\rho^2 C_s^2)}{2(1-\rho)(1+\rho^2 C_s^2)} \tag{6}$$

$$C_s^2 = \frac{\sigma_s^2}{(1/\mu)^2} \tag{7}$$

$$C_a^2 = \frac{\sigma_a^2}{(1/\lambda)^2} \tag{8}$$

where L_q is the queue length, ρ is the server utilisation, $\rho = \frac{\lambda}{\mu s}$, $s = 1$ is the number of servers, C_s^2 is the coefficient square of the service time variance, C_a^2 is the coefficient square of the inter-arrival time variance, σ_s^2 is the variance of the service time and σ_a^2 is the variance of inter-arrival time. Using Little's rule, we can compute the queue waiting time (D_i^W) as:

$$D_i^W = \frac{L_q}{\lambda} \tag{9}$$

4.4. Mobile device energy model

The energy consumed by mobile device E_i to execute t_i is estimated by calculating the total processing energy E_i^P consumed by the mobile device, the waiting energy E_i^W and data transfer energy E_i^C.

$$E_i = E_i^P + E_i^C + E_i^W \tag{10}$$

$$E_i{}^P = D_i{}^P \cdot \epsilon_i{}^P \tag{11}$$

$$E_i^C = D_i{}^C \cdot \epsilon^C \tag{12}$$

$$E_i^W = D_i{}^W \cdot \epsilon^W \tag{13}$$

Where $\epsilon_i{}^P$, ϵ^W, ϵ^C are the estimated energy consumption per second in the mobile device for task t_i, remote execution waiting (in seconds) and data communication MB/s, respectively.

4.5. Monetary cost model

The monetary cost represents the amount of money required to run a task t_i in a target computation environment P_r. This includes two parts. The first part is the task processing cost C_i^P in a remote server at edge nodes or the public cloud. The processing cost is measured using the processing time length D_i and the cost per hour for the host server p_i. The second part is the data communication cost C_i^C, which is measured by the amount of data s_i to be transferred with bandwidth cost β_{cost}.

$$C_i = C_i^P + C_i^C \tag{14}$$

$$C_i{}^P = D_i{}^P \cdot p_i \tag{15}$$

$$C_i^C = s_i \cdot \beta_{cost} \tag{16}$$

5. The proposed offloading algorithm

The system objective is to find the optimal solution in which the total energy consumption of the mobile device and the total monetary cost are minimised under the constraints of available device energy and task execution deadline. The optimisation algorithm is implemented in the decision-maker, which is responsible for collecting the information needed to run the algorithm and produce the optimisation solution. The solution represents a tuple of each task t_i and the selected computation environment, either local execution on the mobile client device P_m or remote execution on external computation machine P_r (edge nodes or public cloud VM).

This work utilises MILP to address cost, energy and time optimisation for an MECC architecture. A MILP formulation essentially determines that all objective functions and constraint equations are linear. The linear nature of MILP facilitates the easy and rapid solution of any subproblems, while allowing for relatively complex formulations. Such an approach is usually faster and less computation intensive than non-linear. Additionally, MILP's linear ensures that any minimum obtained is a global minimum and not a local one (Vielma, 2015). Furthermore, the complexity time for MILP increases linearly as the problem space grows. On the other hand, the MILP algorithm has some limitations with respect to the nonlinearity effects and the high dimensionality of the given problem (Urbanucci, 2018). Based on our assessment, for the system and workload considered here, the proposed MILP technique is able to find the optimal solution. However, for more complex workload (e.g., applications with complex dependencies or variable data size), MILP might be able to generate only suboptimal solutions. This could be part of future work on non-linear methods involving quantitative comparisons.

We denote the binary offloading decision variable for task t_i by

$$x_i = (x_i^l, x_{i1}^f, \ldots, x_{iM}^f, x_i^c) \tag{17}$$

In which x_i^l, x_{ij}^f, x_i^c indicate that task t_i is processed locally at the mobile device, edge node ($1 \le j \le M$) or the cloud server, respectively. A solution x_i represents the binary encoding for task t_i allocation on system resources. Only one position is set to number 1, which indicates that the corresponding machine is allocated that task. Remaining positions are set to number 0. Based on solution x_i for task t_i, optimisation parameters, that is, execution time D_i, consumed energy E_i and monetary cost C_i are calculated as follows.

$$D_i = h_i^T x_i, E_i = e_i^T x_i, C_i = c_i^T x_i \tag{18}$$

where

- $h_i = (T_i^l, T_{i1}^f, \ldots, T_{iM}^f, T_i^c)$ (the calculated end-to-end execution time for task t_i on each computation unit). See Eq. (3);
- $e_i = (E_i^l, E_{i1}^f, \ldots, E_{iM}^f, E_i^c)$ (the calculated end-to-end energy for task t_i on each computation unit). See Eq. (10);

- $c_i = (C_i^l, C_{i1}^f, \dots, C_{iM}^f, C_i^c)$ (the calculated end-to-end monetary cost for task t_i on each computation unit). See Eq. (14).

The optimisation problem formulated as monetary cost (C) times energy (E) is based on the assumption that they contribute equally to the objective function:

- $E = e^T x$, where $e = (e_1, \dots, e_N)$ represents the matrix of energy values for task t_i on all computation resources and e^T is the energy matrix transposition;
- $C = c^T x$, where $c = (c_1, \dots, c_N)$ represents the matrix of cost values for task t_i on all computation resources and c^T is the cost matrix transposition;
- $x = (x_1, \dots, x_N)$ (the tuple represents the decision variable of each task t_i); and
- N is the number of tasks.

The optimisation function can be formulated as:

$$P_0 : \min(E \cdot C) \tag{19}$$

Subject to R_0

$$D_{t_i} < \partial_i, \forall t_i \in A$$

$$E < e$$

$$x_i^l, x_{ij}^f, x_i^c \in \{0, 1\}, \forall (i, j) \in NxM$$

where constraints of deadline $D_{t_i} < \partial_i$ and user mobile device energy $E < e$ should be satisfied.

$\boldsymbol{P_0}$ is an NP-hard optimisation problem, due to its mixed integer non-linear programming (Vielma, 2015). The resulting problem is a convex optimisation problem (Boyd and Vandenberghe, 2004). To prove the convexity of our problem, we relaxed the binary decision variables $x_i^l, x_{ij}^f, x_i^c, \forall (i, j) \in NxM$ into real numbers of range $[0, 1]$ (please refer to the work of Vu et al. (2019) for the mathematical proof). Thus, the optimisation problem can be formulated as:

$$\widetilde{P}_0 : \min(E \cdot C) \tag{20}$$

subject to R_0 and :

$$(C3) : x_i^l + \sum_{j=1}^{M} x_{ij}^f + x_i^c = 1$$

$$x_i^l, x_{ij}^f, x_i^c \in [0, 1], \forall (i, j) \in NxM$$

The Branch and Bound algorithm (BB) is a commonly used search optimisation algorithm for solving integer linear programming and MILP problems. The algorithm solves linear programming relaxations using restricted ranges of possible values of the integer variables. It attempts to generate a sequence of updated bounds on the optimal objective function value. Like dynamic programming, BB is an intelligently structured search of the space of all feasible solutions (Lawler and Wood, 1966). In BB, an optimisation problem is considered as a search tree, in which every tree node represents the transition to a subproblem after fixing a binary variable. Subproblems have the same objective function, bounds and linear constraints as the original problem, but without integer constraints.

Algorithm 1 provides a high-level abstraction of the behaviour of the BB optimisation method. Algorithm 1 takes two inputs: an application A and available computation resources set R. The algorithm aims to find the optimal offloading decision for minimising the cost.energy objective value. The optimisation objective *optVal* is initialised to infinity because we target a minimisation problem. Algorithm is a high-level description of how BB works, with the following steps:

1. The algorithm starts with an initial subproblem P_0.
2. Calculate the objective value *solObjValue* of the stack top solution *toCheckSol*. In Lines 11–12, the solution (decision) is evaluated by calling *callSolObjectiveValue* in Algorithm 2, and the optimum value is updated if a new minimum is found, Lines 16–18. A solution *toCheckSol* represents the binary encoding for application A tasks allocation on system resources R. Only one position is assigned to 1, which indicates the target machine for that task. Based on solution *toCheckSol*, optimisation parameters, that is, execution time D_i, consumed energy E_i and monetary cost C_i are calculated via *callSolObjectiveValue* in Algorithm 2.
3. Check to update the best solution *bestSol*.
4. Branch on *toCheckSol* to produce new solutions (nodes), Line 20. The branching step is taken heuristically, according to one of several rules. Each rule is based on the idea of splitting a problem by restricting one variable to be less than or equal to an integer $J = 1$, or greater than or equal to $J + 1$. On each branching step, two subproblems *toAddSubProblems* are constructed by changing on one decision variable x_i. Technically, a subproblem is a candidate offloading decision.
5. Each subproblem is checked for problem upper and lower bound constraints using the function *checkIntegerConstraints*. This is called bounding a solution, Line 22.
6. Loop until the solutions stack (*subP*) is empty, Line 10.
7. Algorithm 1 returns the optimised application scheduling plan s and the minimum problem value *optVal*, Line 29.

Algorithm 1 Find optimal application tasks schedule.

```
1: Inputs:
2: Application tasks $A = \{t_i, \dots, t_n\}$
3: Computation resources $R = \{r^l, (r_1^f, \dots, r_m^f),\ r_1^c\}$
4: Output:
5: optimised application tasks allocation plan $\boldsymbol{P_0}$
6: Initialise:
7: $optVal = \infty$
8: $bestSol = \{\}$
9: $subP = \{P_0\}$
10: while $Len(subP > 0)$ do
11:   $toChecksol = subP[0]$
12:   $solObjValue = callSolObjectiveValue(A, R, toChecksol)$
13:   if $solObjValue > optVal$ then
14:     $subP.removeAt(0)$
15:   else
16:     if $solObjValue < optVal$ then
17:       $bestSol = toChecksol$
18:       $optVal = solObjValue$
19:     else
20:       $toAddSubProblems = Branch(subP[0])$
21:       for $i = 1$ to $Len(toAddSubProblems)$ do
22:         if $checkIntegerConstraints(toAddSubProblems[i]) == True$ then
23:           $subP.insertAt(0, toAddSubProblems[i])$
24:         end if
25:       end for
26:     end if
27:   end if
28: end while
29: RETURN $s, optVal$
```

Algorithm 2 provides the steps to calculate the objective value for an offloading solution *sol*. The algorithm takes three inputs: proposed solution *sol*, the application A and the computation resources R. The resource handler is responsible for collecting data about the current status of the resources R. Prior to running the MILP solver, Line 19, the algorithm starts with building the problem. A MILP problem has two main components. The first is the list of constraints *Constraints*. Two constraints are included: the deadline constraint, Line 14, and the maximum available energy constraint, Line 18. A solution cost value is calculated at the cost estimator, which communicates with the queuing estimator to estimate task waiting time t_i at resource r_j. Moreover, the cost estimator collects data about mobile device energy status by communicating with the application profiler. To set the deadline constraint, the execution time for each task is calculated by calling the function *findTime* and using Eq. (3), Line 11. The energy constraint expresses the cumulative energy consumed, Lines 12 and 18, using Eq. (10). The monetary cost is calculated for each task, Line 13, using Eq. (14) and the cost value for each task is added to the list, Line 15. Line 15 shows the model objective function of energy E and cost C. The MILP solver function, Line 19, uses constraints and cost matrices to run the solver to find the offloading decision with minimum cost value. Finally, the offloading decision and cost value is returned to Algorithm 1.

Algorithm 2 callSolObjectiveValue.

```
1: Inputs:
2: Application tasks $A = \{t_i, \dots, t_n\}$
3: Computation resources $R = \{r^l, (r^f_1, \dots, r^f_m),\ r^c_1\}$
4: Problem solution - decision variable $X = \{x_i, \dots, x_n\}$
5: Output:
6: MILP problem parameters
7: Begin:
8: $Constraints = \{\}$
9: for $i = 1$ to $Len(A)$ do
10:     for $j = 1$ to $Len(R)$ do
11:         $T[i,j] = findTime(t_i, r_j)$
12:         $E[i,j] = findEnergy(t_i, r_j)$
13:         $C[i,j] = findCost(t_i, r_j)$
14:         $Constraints[i,j] = AddToConstraint(T[i,j], t_i.deadline)$
15:         $CostVal[i,j] = AddToObjective(E[i,j] \cdot C[i,j])$
16:     end for
17: end for
18: $Constraints[Len(A), Len(R)] = AddToConstraint(Sum(E), maxE)$
19: $X, ObjVal = SolveProblem(CostVal, Const)$
20: RETURN $X, ObjVal$
```

6. Performance evaluation

The performance evaluation for the proposed multi-objective offloading technique was conducted using two experiments. The first one was a real experiment to validate the model by comparing the results of the proposed execution model and offloading technique against real executions. For this experiment, we implemented a real MECC environment, as illustrated in Fig. 1, while in the second experiment we used synthetic application data to evaluate the proposed offloading technique.

6.1. Experimental setup

The research objective was to study the contribution of parameters, such as the mobile network interface, task input data size distribution and number of application users, to the offloading optimisation objectives (monetary cost and device consumed energy). Firstly, a mobile network interface determines the cellular data transfer technology that is available to the user device to send or receive data. Table 2 provides details about the bandwidth distribution of each mobile network interface and the average data transmission latency over these interfaces. Secondly, we initialise the application size with 30 tasks. This number of tasks provides the ability to verify a convenient range of data size, for the purposes of a data-intensive application, without complicating the application structure.

Table 2
Network interface bandwidth.

Network Type	Min. Bandwidth (MB/s)	Max. Bandwidth (MB/s)
3G	2	5
4G	8	12
WiFi	25	30
Latency	**Min. Latency (s)**	**Max. Latency (s)**
	0.85	6.5

Table 3
Task input data size distribution.

Data Distribution	Min. Size (MB)	Max. Size (MB)
Small	20	200
Medium	200	500
Large	500	2000
X-Large	2000	4000

Each task has the following properties:

- Computation requirement (task workload) D^P: we used the workload model proposed by Anglano and Canonico (2008). Application task deadline values are uniformly distributed in the interval $[X \pm 0.5X]$, where X is the granularity of the tasks. The model is used to determine the task deadline with $X = 1000$ granularity. After applying the model, deadline values are uniformly distributed between 500 and 1500, all in seconds
- The task data file locations (L) are distributed randomly between the edge storage server, the mobile device and other cloud storage (i.e. Amazon Simple Storage Service (AWS S3))
- Task data size (s) model: we assume four different scenarios for the application data model to create each task. Table 3 presents four task data distributions with minimum and maximum sizes. The data distributions are selected to align with the study of data size contribution to the offloading optimisation decision
- Task data sensitivity factor (ω_i): the factor has a value between zero and 1, reflecting the task execution response in the change of data size

We assume the edge computation cost is 40% of the cloud computation cost. In addition, we assume that CPU cores have the same clock speed and MIPS, and that the computation machine has enough memory at the time of processing. Cost for the edge nodes can be specified by the owner or resource provider. Table 4 provides details about the computation resources used in the experiment.

6.2. System profiling

We implemented a profiling process to compute an approximation of energy and bandwidth estimation parameters. Prior to a profiling iteration, an experiment configuration is obtained from data size distribution. For each configuration setup, a BoT application of 30 tasks is executed, and averages of processing, communication and waiting energy values are recorded for 30 executions. PowerTutor software is used for power estimation and provides energy consumption estimates within 5% of actual values (Yang, 2012).

Table 4
Experiment resources configuration.

Resource Name	#Cores	#Nodes	Computation Cost ($/hour)
Mobile Device	2	1	0
Edge Node	2	8	0.0742
Cloud Server	32	1	0.3712

The profiling process includes network bandwidth and latency profiling to examine bandwidth boundaries for data communication between computation and storage units at cloud and edge layers. Table 2 shows bandwidth profiling results. In addition, energy parameters are used to calculate the energy the device consumes for task processing, data transfer and device waiting for external task processing ${\epsilon_i}^P$, ϵ^C, ϵ^W, respectively. The profiling process considers the parameters of task data size, mobile network, mobile device processing capability and physical data file transfer between data storage and computation locations. We conducted the experiment for each combination of data size and bandwidth. Table 3 shows the data size distributions for data files used in the profiling experiments. Also, as part of system profiling, the data sensitivity factor ω_i for a task t_i was estimated. The sensitivity factor measures how task processing is sensitive to the change in task data size.

6.3. *Model validation*

In this section, we provide the validation for the proposed execution model using a real execution scenario. The validation objective was to evaluate optimiser stability and validity by comparing results obtained from the optimisation model with those collected from the real experiments. Results include execution time, consumed energy and monetary cost of computation and data communication.

We ran the evaluation experiments with 30 task applications, small input data size (see Table 3) and included three mobile network interfaces (3G, 4G and WiFi). Fig. 2 shows the evaluation results of the three model parameters based on the change in the mobile network interface. The experiments predict slight estimation errors for three model parameters, with 7%, 7% and 12% for execution time, consumed energy and monetary cost, respectively. On average, the estimation error per scenario is around 9%.

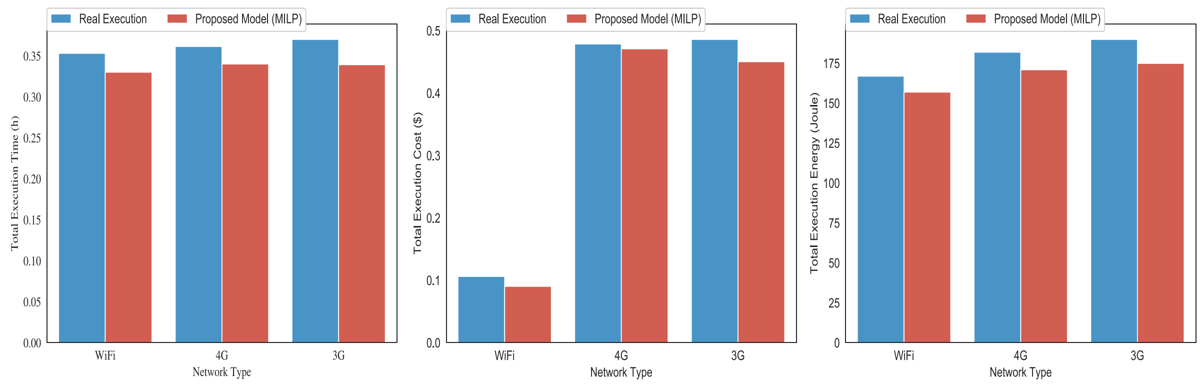
6.4. *System evaluation results*

After validation of the execution model, we used a similar setup to evaluate the proposed offloading technique under different working conditions. We began by comparing the proposed optimisation technique with PSO, mobile and random techniques. The PSO technique was employed in our previous work in data-intensive mobile application offloading (Alkhalaileh et al., 2019). The mobile technique involves executing all application tasks in the user device. The random technique is a baseline technique of much less complexity than the proposed techniques. We wanted to know how our proposed techniques perform against the baseline and measure the performance gap. The task deadline is a soft deadline and still applications can be completed with the random technique (with low QoS achievement). Experimental results provide insight on how the optimiser responds to a variety of parameters, such as data size, bandwidth, deadline and number of users, when constructing an optimised application offloading plan.

Figs. 3 and 4 compare the results of applying the four techniques for optimising model parameters with respect to the variation in task input size. With the 4G network interface, Fig. 3 shows that all techniques demonstrate a linear increase in execution time, monetary cost and mobile energy consumption as data size increases. The MILP optimiser produces the lowest scores of energy and cost; there is a large performance gap between it and the second-best optimiser, the PSO, even for large input files, proving the stability of the model. With the WiFi network, as shown in Fig. 4, the MILP technique minimised the execution cost regardless of the change in input data size. This can be inferred from execution time change behaviour, in which the MILP tried to work with maximum task deadlines to schedule tasks in resources with minimum cost, in this case, edge resources. Overall, results revealed that the proposed technique reduced the execution cost for data-intensive applications by an average of 46% and 76%, in comparison to PSO and full execution on a mobile device, respectively. In addition, the model provides energy reductions of 35% and 84%, respectively.

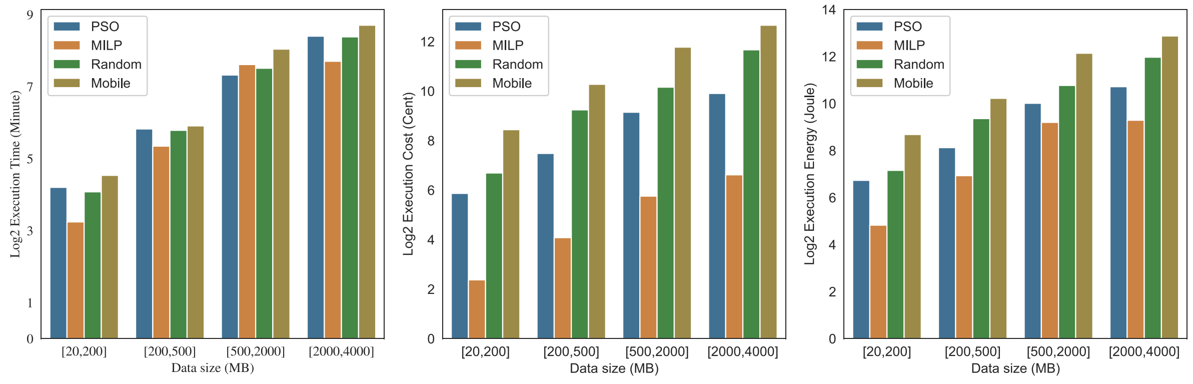
To study the contribution of the task deadline, we applied two scenarios, relaxed deadline and hard deadline. To construct the two deadline models, we applied the execution model proposed by Anglano and Canonico (2008). The model used to determine the task deadline is based on the level of granularity. We assume a relax deadline model will have a twice the granularity of a hard deadline model. Application task deadline values will be uniformly distributed in the interval $[X \pm 0.5X]$, where X is the granularity of the tasks. Fig. 5 compares the two scenarios with respect to the change in input data size. It can be concluded that with large input data the optimiser works toward scheduling tasks in high-performance computation machines in a public cloud, which explains the cost difference in the two scenarios. In addition, the consumed energy is increased, reflecting the greater of mobile waiting time for remote execution.

The number of users parameter was added to the experiment to study the impact of task waiting time on consumed energy and total monetary cost. A task waiting time is estimated based on the G/G/1 queuing model (Marchal, 1976). We assume that both cloud and edge servers behave as G/G/1 queuing systems with variation in task arrival rate and service time. The arrival rate represents the number of received tasks in a certain time slot and the service time measures the time required to process the incoming task. However, application tasks differ in computation requirements; thus, the queuing system needs to handle



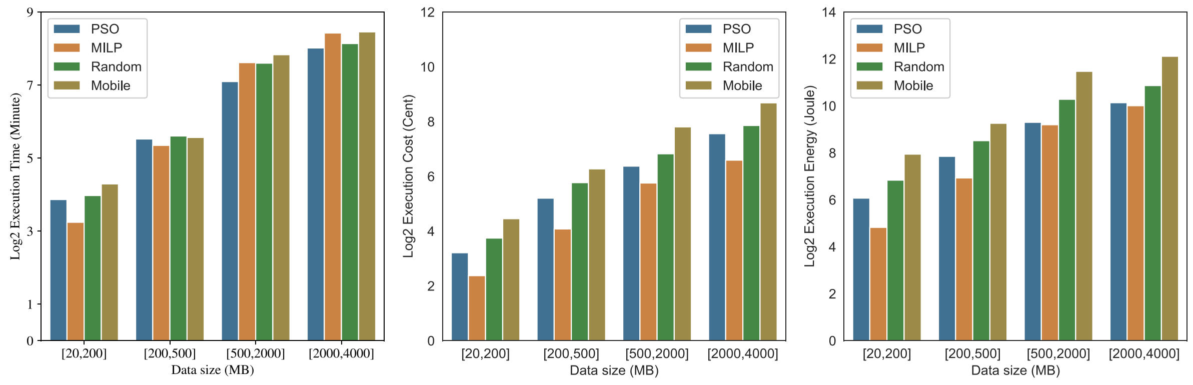
(a) Execution Time   (b) Monetary Cost   (c) Mobile Energy

Fig. 2. Evaluation of proposed optimiser with real scenarios for three different communication networks.



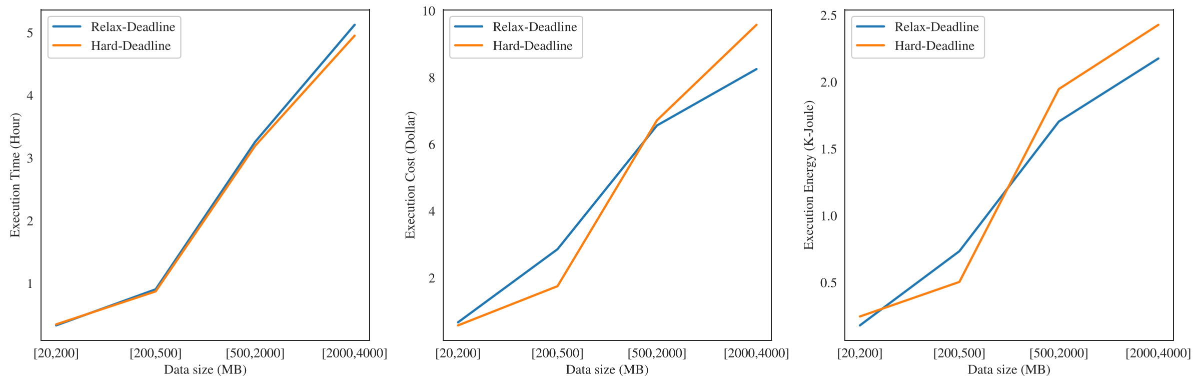
(a) Execution Time (b) Monetary Cost (c) Mobile Energy

Fig. 3. System performance measurements with 4G network interface for different data sizes.



(a) Execution Time (b) Monetary Cost (c) Mobile Energy

Fig. 4. System performance measurements with WiFi network interface for different data sizes.

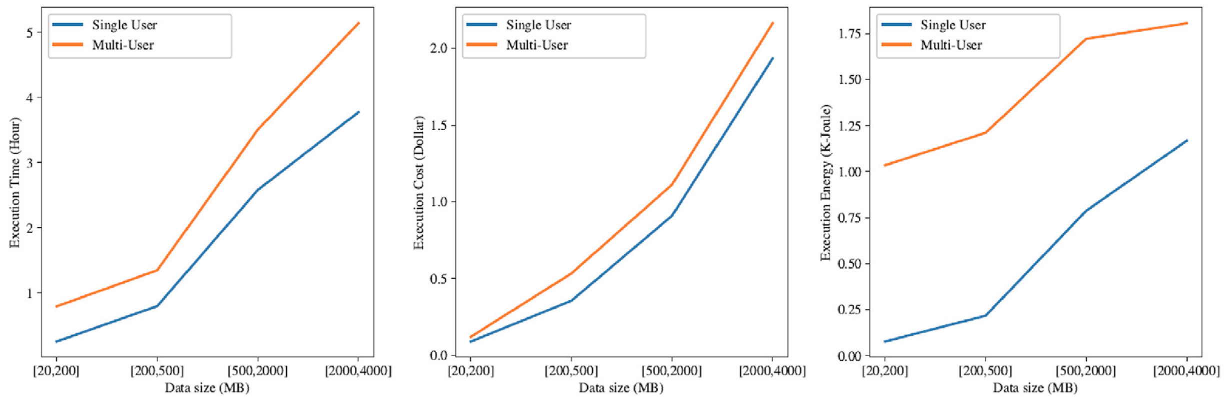


(a) Execution Time (b) Monetary Cost (c) Mobile Energy

Fig. 5. Comparing the impact of deadline sensitivity on optimisation parameters.

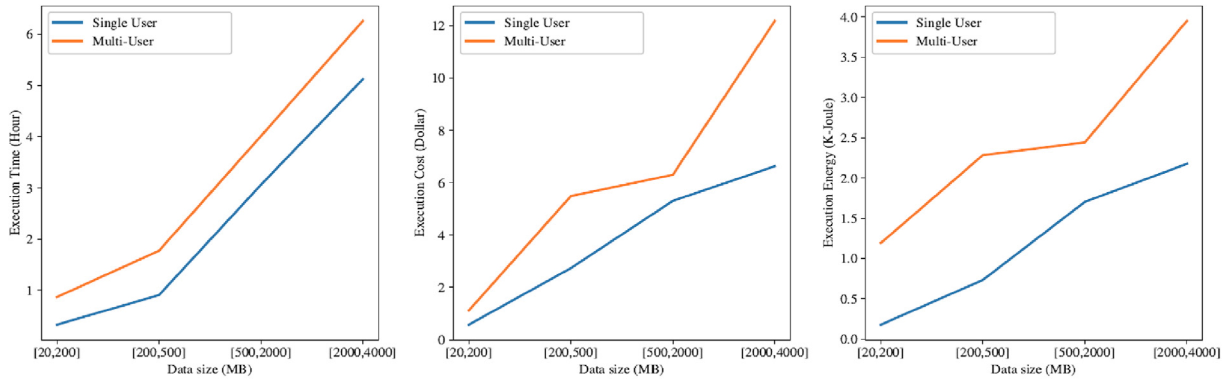
heterogeneity in service times. To overcome this issue, we performed an experiment to profile queuing results at cloud and edge servers and used the means of variable rates and service times to feed the optimisation model. We tested two scenarios, single-user and multi-user: single-user means only one user application is in execution, while in the multi-user scenario many of these applications are submitted for execution.

Figs. 6 and 7 show that the difference in execution time between multi-user and single-user scenarios is increased marginally with greater input data size, due to the increase in application waiting time. With the WiFi network, as shown in Fig. 6, the optimiser was able to



(a) Execution Time (b) Monetary Cost (c) Mobile Energy

Fig. 6. Comparing the impact of multi-user and single-user on the system performance: WiFi mobile network.



(a) Execution Time (b) Monetary Cost (c) Mobile Energy

Fig. 7. Comparing the impact of multi-user and single-user on 4G mobile network.

optimise the cost by allowing high waiting time and thus, high energy. On the other hand, with the high-cost network, such as 4G as shown in Fig. 7, the time difference between the two scenarios remained steady, allowing energy savings.

Fig. 8 compares the optimisation time of the proposed MILP-based technique and our previous PSO-based technique (Alkhalaileh et al., 2019). The figure confirms how PSO-based optimisation time increases exponentially as the number of tasks increases (application size). The MILP technique shows near-zero optimisation time variation with increase in application size, while PSO demonstrates a 45% time increase on average. For example, running applications of 100 and 150 tasks using the MILP technique will take 2.7 and 2.9 s, respectively, while running the same application using PSO will take 23 and 56 s, respectively. This result reveals that our proposed technique has a low time complexity and can be adapted for large-scale data-intensive applications.

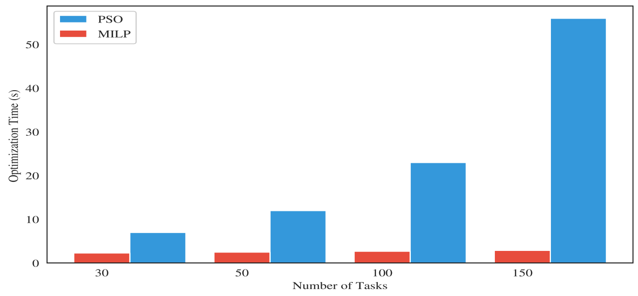


Fig. 8. Comparing the optimisation time between MILP and PSO.

7. Conclusions and future work

We propose a mobile application offloading algorithm to schedule data-intensive mobile applications in the MECC environment. The algorithm generates an application execution plan which accounts for application size, input data size, available network bandwidth and mobile device energy. Results show the MILP-based technique ensures that the execution cost and energy consumption change linearly with change in data size. For latency-sensitive application models, our results imply that data size is the major driver of both monetary cost and device energy, with the mobile network model making a smaller contribution. Moreover, the work measures the impact of the number of active application users on model parameters. Results demonstrate the model's ability to handle data-intensive applications by improving the efficiency of mobile edge computing. Moreover, the MILP-based model has significantly lower computation time than the PSO technique. Finally, the model's advantages mean it should be adopted for data-inventive mobile edge computing to save energy saving and reduce costs.

In future work, we plan to conduct experiments in data-intensive application offloading planning and optimisation for other application models, such as workflow and stream models. In addition, we plan to study other optimisation objectives, such as energy-only and cost-only.

CRediT authorship contribution statement

Mohammad Alkhalaileh: Conceptualization, Methodology, Software, Data curation, Writing - original draft. **Rodrigo N. Calheiros:** Writing - review & editing. **Quang Vinh Nguyen:** Software. **Bahman Javadi:** Supervision, Writing - review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

Abbas, N., Zhang, Y., Taherkordi, A., Skeie, T., 2017. Mobile edge computing: a survey. IEEE Internet Things J. 5 (1), 450–465.

Abolfazli, S., Sanaei, Z., Gani, A., Xia, F., Yang, L.T., 2014. Rich mobile applications: genesis, taxonomy, and open issues. J. Netw. Comput. Appl. 40, 345–362.

Alkhalaileh, M., Calheiros, R.N., Nguyen, Q.V., Javadi, B., 2019. Dynamic resource allocation in hybrid mobile cloud computing for data-intensive applications. In: International Conference on Green, Pervasive, and Cloud Computing. Springer, pp. 176–191.

Anglano, C., Canonico, M., 2008. Scheduling algorithms for multiple bag-of-task applications on desktop grids: a knowledge-free approach. In: Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on. IEEE, pp. 1–8.

Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., et al., 2010. A view of cloud computing. Commun. ACM 53 (4), 50–58.

Boyd, S., Vandenberghe, L., 2004. Convex Optimization. Cambridge university press.

C. V. N. I. Cisco, 2014. Global Mobile Data Traffic Forecast Update, pp. 2013–2018 white paper.

Cardellini, V., Person, V.D.N., Di Valerio, V., Facchinei, F., Grassi, V., Presti, F.L., Piccialli, V., 2016. A game-theoretic approach to computation offloading in mobile cloud computing. Math. Program. 157 (2), 421–449.

Chen, M., Hao, Y., 2018. Task offloading for mobile edge computing in software defined ultra-dense network. IEEE J. Sel. Area. Commun. 36 (3), 587–597.

Chen, X., Jiao, L., Li, W., Fu, X., 2015. Efficient multi-user computation offloading for mobile-edge cloud computing. IEEE/ACM Trans. Netw. 24 (5), 2795–2808.

Chun, B.-G., Ihm, S., Maniatis, P., Naik, M., Patti, A., 2011. Clonecloud: elastic execution between mobile device and cloud. In: Proceedings of the Sixth Conference on Computer Systems. ACM, pp. 301–314.

Cuervo, E., Balasubramanian, A., Cho, D.-k., Wolman, A., Saroiu, S., Chandra, R., Bahl, P., 2010. Maui: making smartphones last longer with code offload. In: Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services. ACM, pp. 49–62.

Enzai, N.I.M., Tang, M., 2016. A heuristic algorithm for multi-site computation offloading in mobile cloud computing. Procedia Comput. Sci. 80, 1232–1241.

Giurgiu, I., Riva, O., Juric, D., Krivulev, I., Alonso, G., 2009. Calling the cloud: enabling mobile phones as interfaces to cloud applications. In: Proceedings of the 10th ACM/IFIP/USENIX International Conference on Middleware. Springer-Verlag New York, Inc., p. 5.

Goudarzi, M., Zamani, M., Haghighat, A.T., 2017. A fast hybrid multi-site computation offloading for mobile cloud computing. J. Netw. Comput. Appl. 80, 219–231.

Hung, S.-H., Shih, C.-S., Shieh, J.-P., Lee, C.-P., Huang, Y.-H., 2012. Executing mobile applications on the cloud: framework and issues. Comput. Math. Appl. 63 (2), 573–587.

Jararweh, Y., Al-Ayyoub, M., Al-Quraan, M., Loai, A.T., Benkhelifa, E., 2017. Delay-aware power optimization model for mobile edge computing systems. Personal Ubiquitous Comput. 21 (6), 1067–1077.

Lawler, E.L., Wood, D.E., 1966. Branch-and-bound methods: a survey. Oper. Res. 14 (4), 699–719.

Mach, P., Becvar, Z., 2017. Mobile edge computing: a survey on architecture and computation offloading. IEEE Commun. Surv. Tutor. 19 (3), 1628–1656.

Mao, Y., Zhang, J., Letaief, K.B., 2016. Dynamic computation offloading for mobile-edge computing with energy harvesting devices. IEEE J. Sel. Area. Commun. 34 (12), 3590–3605.

Mao, Y., You, C., Zhang, J., Huang, K., Letaief, K.B., 2017. A survey on mobile edge computing: the communication perspective. IEEE Commun. Surv. Tutor. 19 (4), 2322–2358.

March, V., Gu, Y., Leonardi, E., Goh, G., Kirchberg, M., Lee, B.S., 2011. cloud: towards a new paradigm of rich mobile applications. Procedia Comput. Sci. 5, 618–624.

Marchal, W.G., 1976. An approximate formula for waiting time in single server queues. AIIE Trans. 8 (4), 473–474.

Nan, X., He, Y., Guan, L., 2011. Optimal resource allocation for multimedia cloud based on queuing model. In: Multimedia Signal Processing (MMSP), 2011 IEEE 13th International Workshop on. IEEE, pp. 1–6.

Patel, M., Naughton, B., Chan, C., Sprecher, N., Abeta, S., Neal, A., et al., 2014. Mobile-edge Computing Introductory Technical White Paper, White Paper, Mobile-Edge Computing (MEC) Industry Initiative, pp. 1089–7801.

Ren, J., Yu, G., He, Y., Li, G.Y., 2019. Collaborative cloud and edge computing for latency minimization. IEEE Trans. Veh. Technol. 68 (5), 5031–5044.

Samimi, F.A., McKinley, P.K., Sadjadi, S.M., 2006. Mobile service clouds: a self-managing infrastructure for autonomic mobile computing services. In: IEEE International Workshop on Self-Managed Networks, Systems, and Services. Springer, pp. 130–141.

Satyanarayanan, M., Bahl, P., Caceres, R., Davies, N., 2009. The case for vm-based cloudlets in mobile computing. IEEE Pervasive Comput. 8 (4), 14–23.

Shi, W., Cao, J., Zhang, Q., Li, Y., Xu, L., 2016. Edge computing: vision and challenges. IEEE Internet Things J. 3 (5), 637–646.

Terefe, M.B., Lee, H., Heo, N., Fox, G.C., Oh, S., 2016. Energy-efficient multisite offloading policy using markov decision process for mobile cloud computing. Pervasive Mob. Comput. 27, 75–89.

Terzopoulos, G., Karatza, H.D., 2016. Power-aware bag-of-tasks scheduling on heterogeneous platforms. Cluster Comput. 19 (2), 615–631.

Urbanucci, L., 2018. Limits and potentials of mixed integer linear programming methods for optimization of polygeneration energy systems. Energy Procedia 148, 1199–1205.

Vielma, J.P., 2015. Mixed integer linear programming formulation techniques. SIAM Rev. 57 (1), 3–57.

Vu, T.T., Nguyen, D.N., Hoang, D.T., Dutkiewicz, E., 2019. Optimal Task Offloading and Resource Allocation for Fog Computing. arXiv:1906.03567.

Wang, Y., Chen, R., Wang, D.-C., 2015. A survey of mobile cloud computing applications: perspectives and challenges. Wireless Pers. Commun. 80 (4), 1607–1623.

Yang, Z., 2012. Powertutor-a Power Monitor for Android-Based Mobile Platforms. EECS, University of Michigan, p. 19. retrieved September 2.

Zhao, T., Zhou, S., Guo, X., Zhao, Y., Niu, Z., 2015. A cooperative scheduling scheme of local cloud and internet cloud for delay-aware mobile cloud computing. In: 2015 IEEE Globecom Workshops (GC Wkshps). IEEE, pp. 1–6.

Zhou, B., Dastjerdi, A.V., Calheiros, R.N., Srirama, S.N., Buyya, R., 2015a. A context sensitive offloading scheme for mobile cloud computing service. In: Cloud Computing (CLOUD), 2015 IEEE 8th International Conference on. IEEE, pp. 869–876.

Zhou, B., Dastjerdi, A.V., Calheiros, R., Srirama, S., Buyya, R., , 2015b. mcloud: a context-aware offloading framework for heterogeneous mobile cloud. IEEE Trans. Serv. Comput. 10 (5), 797–810.

Mohammad Alkhalaileh received his M.Sc. in Software Engineering from Central Queensland University, Australia, in 2012. He received B.Sc. degree in Computer Information System from AL al-Bayt University, Jordan, in 2008. He is currently pursuing his Ph.D. studies on Mobile Edge Cloud Computing with the School of Computer, Data and Mathematical Sciences at the Western Sydney University, Australia. His research interest includes mobile cloud and edge computing, healthcare informatics, internet of things and big data.

Dr. Rodrigo N. Calheiros is a Senior Lecturer in the School of Computer, Data and Mathematical Sciences at the Western Sydney University, Australia. He has been conducting research in the area of Cloud computing since 2008, and contributed to diverse aspects in the field including Multiclouds, energy-efficient cloud computing, and Edge computing. He is also one of the original designers and developers of CloudSim, a widely used simulator of Cloud environments. He co-authored more than 70 papers, which attracted together 12,000 Google Scholar citations. His research interests also include Big Data, Internet of Things, Internet Computing, and their application. He is a Fellow of the Higher Education Academy of UK, Senior Member of the IEEE and Senior Member of the ACM.

Dr Nguyen is a Senior Lecturer in visual analytics at the School of Computer, Data and Mathematical Sciences and The MARCS Institute for Brain, Behaviour and Development, Western Sydney University, Australia. His focus is to find effective visualisations to support the analysis of large and complex datasets, particularly genomic, flow cytometry and biomedical data, network and other application based data. His research expertise has been built up since his PhD study, his various experiences at Western Sydney University, University of Technology, Sydney, University of Texas at Dallas and various visiting positions internationally. For his academic career, he has authored and co-authored more than 100 refereed publications, including edited book, book chapters, journals and conference papers relating to this research field. He has received multiple research funding. He has been successful (co)supervised as well as is being supervising several research students. His research details are available: http://staff.scem.uws.edu.au/~vinh/ or http://orcid.org/0000-0002-0815-6224

Bahman Javadi is an Associate Professor in Networking and Cloud Computing in the ***School of Computer, Data and Mathematical Sciences at the Western Sydney University, Australia.*** Prior to this appointment, he was a Research Fellow at the University of Melbourne and a Postdoctoral Fellow at the INRIA Rhone-Alpes, France. He has published more than 100 papers in high quality journals and international conferences and received numerous Best Paper Awards at IEEE/ACM conferences for his research papers. He has presented many keynote and invited talks in several conferences and universities around the world. He served as a program committee of many international conferences and workshops. He has also guest edited many special issue journals. He is co-founder of the Failure Trace Archive, which serves as a public repository of failure traces and algorithms for distributed systems. His research interests include Cloud computing, Edge computing, performance evaluation of large-scale distributed computing systems, and reliability and fault tolerance. He is a Senior Member of ACM, Senior Member of IEEE and Senior Fellow of the Higher Education Academy of UK. His website is: http://staff.scem.uws.edu.au/~bjavadi/