Multimedia Systems
DOI 10.1007/s00530-017-0563-8

SPECIAL ISSUE PAPER

Tradeoff between execution speedup and reliability for compute-intensive code offloading in mobile device cloud

Sajeeb Saha[1] · Md. Ahsan Habib[1] · Tamal Adhikary[1] · Md. Abdur Razzaque[1] · Md. Mustafizur Rahman[1]

Abstract With the advent of different mobile computing technologies, mobile devices have opened up a plethora of computational infrastructure to provide improved performance for compute-intensive applications to the end users. Mobile Device Cloud (MDC) technology brings the code offloading mechanism from distant cloud to neighbor mobile devices. However, the major challenges of code offloading in MDC systems include maximization of computation speedup and reliability; unfortunately, these two performance parameters often oppose each other. In this paper, an optimization framework, namely TESAR, has been devised to tradeoff between application execution speedup and reliability while maintaining device energy within a predefined range. We also provide an algorithm for developing a dependency tree among the modules of an application so as to allow higher number of parallel executions, wherever and whenever it is possible. The emulation results of the proposed algorithm outperform the relevant state-of-the-art works in terms of application completion time, communication latency and rescheduling overhead.

✉ Md. Abdur Razzaque
razzaque@du.ac.bd

Sajeeb Saha
sajeeb.saha.bd@ieee.org

Md. Ahsan Habib
m.a.habib@ieee.org

Tamal Adhikary
tamal@cse.du.ac.bd

Md. Mustafizur Rahman
mustafiz@du.ac.bd

[1] Green Networking Research Group, Department of Computer Science and Engineering, University of Dhaka, Dhaka, Bangladesh

Keywords Mobile device cloud · Compute-intensive · Code offloading · Execution speedup · Reliability · MILP

1 Introduction

The emergence of portable and wearable devices has increased the usage of mobile applications dramatically from last few years. Recent statistics predicted that around 2.6 billion mobile devices, including 220 million wearable devices, are going to be sold by 2020 [1, 2]. These devices are coming with numerous in-built sensors (accelerometers, GPS, microphone, camera, etc.), that stimulate mobile traffics or applications to increase ten times within 2019, whereas cellular capacity only by a factor of 1.4 during the period. Meanwhile, the mobile application market is expected to exceed $100 billion worldwide by 2020 [1, 2]. The sensor based sophisticated applications including pattern or gesture recognition, health monitoring and diagnosis, mobile biometric, reality augmentation, video and image processing etc. require continuous sampling and high speed processing capabilities from today's smart devices [3]. However, these devices still show poor performance while running compute-intensive applications because of limited battery life, insufficient memory and processing resources.

In the literature, computational offloading is proposed to increase the performances by offloading some remotely executable parts of a compute-intensive application to the nearby cloud service providers [4–7]. However, executing codes at distant master clouds [6–11] suffers from higher latency, intermittent connectivity and even unavailability of remote clouds [12, 13]. Nevertheless, the emerging applications including military operations, disaster recovery require real-time responses. In such cases, resource-rich

Published online: 20 July 2017

nearby smaller instances of cloud, known as cloudlets, have been exploited in the literature [14, 16–19] to decrease the service latency. While virtual machine resources on a cloudlet provide better accessibility for executing user applications, they also suffer from resource limitations either due to execution demands of very large application or arrival of many execution requests from devices at pick hour [20]. In such circumstances, in addition to using cloudlet resources, underutilized resources of nearby mobile devices can be exploited opportunistically for the execution of codes from other users, in the form of Mobile Device Cloud (MDC) [12, 13, 21]. In the literature, this opportunistic computation of application codes on nearby mobile devices is also known as Mobile Edge Cloud [20, 22–24] or Virtual Mobile Cloud [7, 25]. Since executors in MDC are located closer to the initiators, the energy consumption, communication latency or response delay have been reduced significantly compared to cloudlets [12, 26, 27].

Large applications are partitioned into a set of modules to make them executable on MDC. A module is executed either on MDC or local device to minimize energy consumption, response time, data transmission time and the wastage of computing resources [28–30]. Almost at everywhere, we are surrounded with a plethora of mobile devices either in large scale stationary place (i.e. stadium, shopping center, restaurants or movie theater) or during travelling by bus/train and on air. Though the devices are mobile, collectively they become stationary in the context of onboard vehicle or the situated place. In such cases, collaboration among these stationary mobile devices may unfold improved computing opportunities for solving resource-hungry applications like augmented reality, face recognition, real time multimedia, visual text translation, voice synthesis etc. on smart devices [31]. However, to execute an application in MDC system, the major challenge is to select reliable devices while minimizing execution time. Nearby devices with high computing resources may often not provide services with good reliability, and vice-versa. Offloadable module selection, estimation of associativity time, data rate, energy consumption and signal strength between offloader and offloadee are other challenges that need to be addressed for the successful execution of an application.

In this paper, MDC platform has been used to design and develop a collaborative and distributed computing architecture to minimize the application execution time. The main philosophy of the work is to reduce the overall execution time of an application by scheduling its modules in a non-overlapping manner. We have aggregated the idle computing power of the nearby mobile (donor) devices to execute compute-intensive applications. Best donor devices are selected considering both execution time and reputation. A preliminary version of this work has been published in [32] that considers parallel execution of modules in the same dependency level of an application tree to minimize the total execution time. However, in this work, we develop TESAR system that trades off in between execution speedup and reliability of computation through optimal selection of donor devices for the application modules. The key contributions of this work are summarized as follows:

- Development of a novel computation framework with necessary system components that enables collaborative computational capability of nearby mobile devices to execute compute-intensive applications in MDC environment.
- Formulation of a Mixed Integer Linear Programming (MILP) objective function with necessary constraints to assign the computation loads for the application modules on nearby donor devices so that the performances are optimized.
- An algorithm to construct a dependency tree among the modules of an application to determine relative execution order.
- Extensive evaluation of computation performances of the proposed TESAR system and compared with the state-of-the-art works in an emulated environment.

The rest of the paper is organized as follows. Sect. 2 contains a study of the existing works on the issue of code offloading in MDC environment. We discuss our application model, MDC architecture and proposed tradeoff analysis in Sect. 3. In Sect. 4, we present experimental environments and results. Finally, we conclude the paper in Sect. 5 with future directions.

2 Related works

The immense increase of mobile devices and applications running on those, cloudlet based solutions can no longer satisfy the resource requirement let alone the remote master cloud based solutions. Offloading modules of an application to nearby mobile devices with idle computing resources is considered to be the future of mobile computing [25, 33, 34]. This concept was first introduced in Serendipity [26] that presented an algorithm to distribute tasks among nearby mobile devices so as to speed up the computing speed and to conserve energy. The authors in [27] developed an application to use the computational resources of nearby mobile devices in Cirrus clouds. The application used RollerNet and Haggle trace to emulate the Serendipity functionality, where fixed size of independent tasks are disseminated among the mobile devices. Although the authors considered the intermittent connectivity, computing

capacity of the devices and quantification of network signal strengths were untouched.

Mtibaa et al. designed an emulation testbed to evaluate the potentiality of Serendipity architecture in [12, 35]. The authors showed that, offloading tasks to nearby mobile devices can save up to 50% execution time and 26% energy compared to master cloud based execution. However, the authors did not consider heterogeneous computational capacities of the nearby devices. Furthermore, the system assumed all devices having the same energy level and number of offloadable tasks was fixed at 50%, which is not often practical.

The user mobility patterns and its opportunistic contact rates with nearby devices were taken into consideration for determining the appropriate devices for offloading by Wang et. al. in [21]. The authors developed a convex optimization technique to determine the amount of computation to be offloaded to other devices and results showed the efficiency of the algorithm in terms of higher computation success rate.

In [13], Mtibaa et. al. provided a generic computation offloading framework to heterogeneous devices including master cloud and cloudlets. The framework maximizes the computational gain with respect to response time, energy consumption and network lifetime. However, there were no consideration of execution dependency among the application modules, expected data rate and available energy level of the devices.

Fernando et. al. exploited stealing method [37] on a set of mobile devices for better load sharing among the worker nodes in [36]. It was first implemented using Bluetooth technology and later on Wi-Fi Direct technology [24]. The model works only for independent tasks and it does not consider the computing capacity of the devices, resulting in stealing jobs from weak workers frequently which in turn causes poor system throughput.

In [38], Habak et al. proposed a dynamic, self-configuring and multi-device mobile cloud, named FemtoCloud, consisting of a cluster of mobile devices coordinated by a cloudlet. The authors tried to maximize the overall workload of the participating devices in the cluster. However, it didn't consider signal strength (or data rate) offered by the devices and their residual energy levels.

Note that all of the above works emphasized on the benefits of MDC technology that distributes computing loads to other mobile devices. However, none of those considered the parallel execution of the application modules, while maintaining the inter-dependencies so as to speed-up the execution of the application. Furthermore, whether a target mobile device is reliable for code offloading or not has not been quantified for its selection. In this work, we develop code offloading framework for MDC system that makes a tradeoff in between execution speedup and reliable execution of codes. The performance is optimized through selecting devices that are reliable, offer higher computing capacities, signal strengths and energy levels.

3 Mobile device cloud architecture

The mobile device cloud (MDC) architecture is comprised of heterogeneous set of mobile devices including laptops, smart-phones, palmtops, tabs/pads, wearable devices like watch, glass, etc. The amount of computation, communication and storage resources of the devices greatly vary from each other. Typically, they have small scale computation and storage capacities that remain idle most of the time on a large number of devices. These idle computing resources can be accumulated together to run heavy weight applications that are not executable on their devices within delay deadline. In MDC, these idle resources act as small virtual machines (VMs) for executing codes of nearby users. The user can configure the VM of his/her device by limiting the amount of resources (CPU, storage, bandwidth usage, etc.) to serve applications of other users and to gain benefits. The amount of resources shared by a device is reconfigurable at any point of time and users have complete control of managing resources of their devices. What follows next, we present a framework for code offloading in MDC and an optimal selection method of nearby mobile VM resources for code offloading.

3.1 Compute-intensive code offloading framework

Limited resources of a mobile device restrict it to deliver the results of an application in time. Application codes requiring higher computing power, need a great number of CPU cycles, storage and thus its battery drains out very fast. However, if the application is partitioned into a number of modules that are offloadable to nearby mobile devices, then the execution load gets distributed and completion time of the application gets reduced. This work is based on such a collaborative computing architecture, as shown in Fig. 1. The mobile devices are connected to each other through Wi-Fi access points (APs).

The proposed architecture has three different tiers. A device that requires to offload codes to others for faster execution is known as donee device and it resides in tier one. The cloudlet and wireless communication infrastructure that facilitate collaboration are at tier two. At tier three, we have (donor) devices where the compute-intensive codes are offloaded for execution. A (donor) device communicates with the cloudlet at tier two and show its interest to run any compute-intensive applications. The cloudlet collects information of the (donor) devices, makes scheduling plan and offloads modules of the application in favor of the

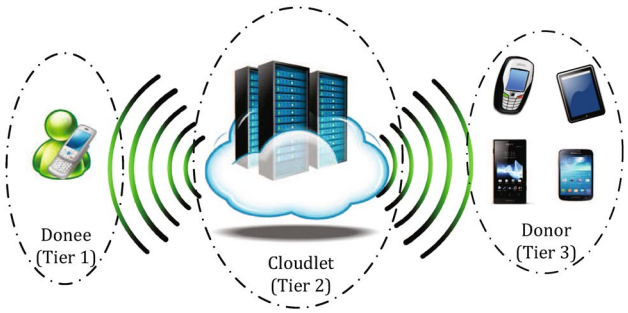


Fig. 1 Computation architecture in Mobile Device Cloud

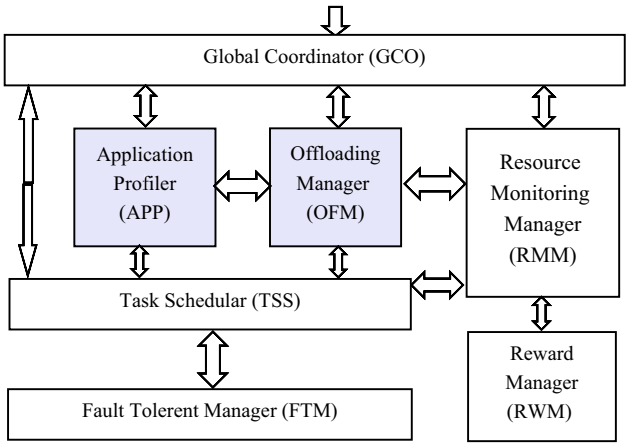


Fig. 2 Compute-intensive code offloading framework

donee devices. It also performs the mapping of application modules to the VMs of (donor) devices and offloads accordingly. It acts as a broker by partitioning the application into modules, coordinating the distribution of modules to remote devices and accumulating results obtained from donor devices. It's very energy hungry for a donee device to keep constant communication with all other devices and thus to decide the code distribution among nearby devices.

The functional components of the proposed compute-intensive code offloading framework are shown in Fig. 2. Detailed descriptions for each of the components are given below.

- *Global coordinator (GCO)* The GCO takes an application as input from mobile user and acts as an interface for other components to communicate with the user. It also controls and coordinates the functionality of other system components.
- *Application profiler (APP)* The APP takes the application from GCO and partitions it into modules which can be processed through parallel execution. Based on static analysis, it determines total number of task modules, instruction size of each module and inter-dependencies among these modules. For each module, it creates a tuple containing module number, instruction size, set of parent nodes of that module and relative execution dependencies. The relative dependencies determine modules that can be executed in parallel so as to minimize the overall execution time of the application. A rooted tree construction algorithm for determining the module dependencies is presented in Sect. 3.2.1.
- *Resource monitoring manager (RMM)* The RMM monitors and stores the device specific information that can be used during module allocation. Specifically, the RMM stores device ID, signal strength, clock cycle, available energy, reputation value, present workload and updates it periodically.
- *Offloading manager (OFM)* The OFM takes information from APP and RMM in order to decide which modules are offloadable and which devices can be selected for offloading. The OFM determines the optimal mapping of individual donor devices to different application modules. Moreover, the OFM ensures that, no device will be assigned two modules which can run simultaneously, i.e., having zero relative execution dependency. Detail operation of offloading decision making mechanism is presented in Sect. 3.2.
- *Task scheduler (TSS)* The TSS dispatches the selected modules to donor devices, and notifies failure(s) of execution, if any, to RMM so as to reassign a different donor device.
- *Fault tolerant manager (FTM)* The FTM handles the operation of the failed modules execution in a priority basis to continue the intended execution flow of the modules. It directly communicates with RMM and selects an available donor device to execute the failed module so that the execution of the other modules are not halted.
- *Reward manager (RWM)* The RWM is a global authority that increases or reduces reward of a mobile device after successful completion or failure of execution of a task respectively. The reputation is given as a means of reward that is increased/decreased based on the successful/failed execution of a task. Whenever a new device wants to get registered, the Resource Monitoring Manager communicates with the RWM to collect and store the reputation information. Detail procedure of calculating the reputation values is presented in Sect. 3.2.5.

The critical part of the proposed TESAR architecture is the identification of parent-child relationship between modules and mapping donor devices to offloadable modules. Although we provide a complete architecture for MDC system, this work, particularly, focuses on OFM that optimally assigns computation modules to donor devices. However, for parallel execution of offloadable modules, identification

of parent-child relationship among the modules is a prerequisite. For this reason, the APP executes Algorithm 1, described in Sect. 3.2.2, to construct a dependency tree between the modules and then the OFM determines the optimal assignment of computation modules to donor devices.

In this work, initial reputation of mobile devices has been chosen randomly. Moreover, a number of works have already been carried out to determine the associativity time of mobile devices [14, 15] and we have adopted [14], since it exploits historical trace of mobile devices and fairly captures the MDC environment.

3.2 Optimization problem formulation

In this subsection, we first define an application model for compute-intensive MDC system. Then, we determine the sets of parent and child modules of the application and time required for executing those locally. After that, we derive an objective function that will select remote mobile hosts in such a way that total execution time gets minimized and reliability of execution in respect of device reputation get maximized while satisfying a number of constraints.

Table 1 shows the list of notations used in this work.

Binary variable $y_{m,k} = \{1, 0\}$ is used to indicate that module $m \in \mathcal{M}$ is executed on device k. Since the application contains both local and remote executable tasks, the total number of modules therefore, can be obtained as,

$$\mathcal{M} = |\mathcal{R}| + |\mathcal{L}|. \tag{1}$$

3.2.1 Compute-intensive application model

Each application in MDC system can be considered as a directed graph G($\mathcal{M}$, e) with $\mathcal{M}$ processing modules where each module performs a specific operation of the application. The directed edges between modules form a rooted

Table 1 List of notations

$\mathcal{M}$	Set of modules in the application
$\mathcal{K}$	Set of available devices
Π_m	Set of parents of module m
L	Set of Leaf nodes
$\mathcal{R}$	Set of modules executed Remotely
$\mathcal{L}$	Set of modules executed Locally
$\mathcal{S}_m^o$	Instruction size of offloadable module $m \in \mathcal{R}$
$\mathcal{S}_m^u$	Instruction size of locally executable module $m \in \mathcal{L}$
$\mathcal{W}_m$	Output instruction size of module $m \in \mathcal{M}$
μ_k	Execution speed of device $k \in \mathcal{K}$
$\mathcal{T}_m^l(o)$	Local device execution time of module $m \in \mathcal{R}$
$\mathcal{T}_m^l(u)$	Local device execution time of module $m \in \mathcal{L}$
$\mathcal{T}_{m,k}^x$	Remote execution time of module $m \in \mathcal{R}$ in device $k \in \mathcal{K}$
$\mathcal{T}_{m,k}^t$	Transmission time of module $m \in \mathcal{R}$ from the cloudlet
$\mathcal{T}_{m,k}^r$	Output reception time of module $m \in \mathcal{R}$ in the cloudlet
$\mathcal{B}_k^u$	Uplink bandwidth between cloudlet and device $k \in \mathcal{K}$
$\mathcal{B}_k^d$	Downlink bandwidth between cloudlet and device $k \in \mathcal{K}$
ϵ_k^x	Energy consumption rate of device $k \in \mathcal{K}$ for module execution
ϵ_k^t	Energy consumption rate of device $k \in \mathcal{K}$ for input transmission
ϵ_k^r	Energy consumption rate of device $k \in \mathcal{K}$ for output reception
$\mathcal{E}_{m,k}^x$	Total execution energy of module $m \in \mathcal{R}$ in device $k \in \mathcal{K}$
$\mathcal{E}_{m,k}^t$	Total input transmission energy of module $m \in \mathcal{R}$ in device $k \in \mathcal{K}$
$\mathcal{E}_{m,k}^r$	Total output transmission energy of module $m \in \mathcal{R}$ in device $k \in \mathcal{K}$
Ψ	Energy threshold
$\mathcal{P}_k$	Signal strength of device $k \in \mathcal{K}$
γ	Tolerable signal strength threshold
λ_k	Associativity time of device $k \in \mathcal{K}$
E_k	Available energy of device $k \in \mathcal{K}$
Ω_k	Reputation of device $k \in \mathcal{K}$
$\phi_{n,m}$	Percentage of dependency of a child module $m \in \mathcal{M}$ on its parent $n \in \mathcal{M}$

tree to define dependencies between modules. This rooted tree determines execution flow of the application as shown in Fig. 3. Each edge consists of two weights—$\mathcal{S}_m^o$ and $\mathcal{W}_m$, the input and output instruction sizes, respectively, of the module $m \in \mathcal{M}$. A dependent child module can be executed only when output of the parent module is available. However, all parallel modules can be executed simultaneously based on availability of the donor devices.

The execution time of an application is inversely related with the number of dependency levels since the later decreases execution parallelism. The total execution time of the application is the maximum execution time of a subtree including communication delays. In this work, local and remote executions have been used interchangeably with donee and donor device executions, respectively.

3.2.2 Construction of rooted tree of modules

A module n is said to be the parent of another module m if, there exists a dependency between the execution of module m and n that is, module m requires the output from n at any particular instance of it's execution. A module may require outputs form more than one parent modules and the set of all parent modules of module m is given by Π_m. Again, a module has been considered as leaf, if it has no child dependent on it. The set of leaf modules is given by $\mathcal{L}$. Algorithm 1 summarizes the steps of determining Π_m and $\mathcal{L}$ from the dependency graph $G(\mathcal{M}, e)$. Here, each element of e is a directional edge from parent module to child module.

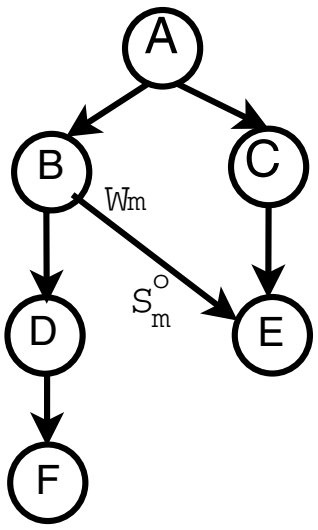


Fig. 3 Dependency tree of the application modules

Algorithm 1 Algorithm for constructing parent and leaf sets of modules

INPUT:
$G(\mathcal{M}, e)$: *A graph of modules* ($\mathcal{M}$) *of an application and edges* (e) *between them*
P : *Parent module from which the execution begins*
OUTPUT:
Π_m : *Set of parent modules for each module* $m \in \mathcal{M}$
$\mathcal{L}$: *Set of leaf modules*

1. *Set* $\Pi_m \leftarrow \Phi$, $\forall m \in \mathcal{M}$
2. *Set* $\mathcal{L} \leftarrow \Phi$
3. $Color_m \leftarrow White$, $\forall m \in \mathcal{M}$
4. *Queue* $Q \leftarrow \Phi$
5. $\Pi_P \leftarrow \Phi$
6. $Q.enqueue(P)$
7. **while** Q *is not empty* **do**
8. $n \leftarrow Q.dequeue()$
9. **for all** $m \in M | (n, m) \in e$ **do**
10. $\Pi_m \leftarrow \Pi_m \cup \{n\}$
11. **if** $Color_m = White$ **then**
12. $Q.enqueue(m)$
13. **end if**
14. **end for**
15. **for all** $m \in \mathcal{M}$ **do**
16. **if** $(n, m) \cap e = \Phi$ **then**
17. $\mathcal{L} \leftarrow \mathcal{L} \cup \{n\}$
18. **end if**
19. **end for**
20. $Color_n = Black$
21. **end while**

3.2.3 Time required for local execution

Time of execution for the whole application in the local device is the total time required for the completion of both local and remote executable modules. Here, time required for executing module $m \in \mathcal{R}$ is given by, $\mathcal{T}_m^l(o) = \frac{\mathcal{S}_m^o}{\mu_{k|k=0}}$ where, $\mathcal{S}_m^o$ represents the size of module $m \in \mathcal{R}$ and μ_k represents the execution speed of device $k \in \mathcal{K}$; $k = 0$ represents the local device and all the remote devices under the cloudlet takes a nonzero value of k. Similarly, time required for executing module $m \in \mathcal{L}$ is represented as, $\mathcal{T}_m^l(u) = \frac{\mathcal{S}_m^u}{\mu_{k|k=0}}$. Therefore, the total local execution time is obtained from the execution delay of all local ($\mathcal{L}$) and remote ($\mathcal{R}$) executable modules,

$$\mathcal{T}(l) = \sum_{m=1}^{|\mathcal{R}|} \mathcal{T}_m^l(o) + \sum_{m=1}^{|\mathcal{L}|} \mathcal{T}_m^l(u). \tag{2}$$

3.2.4 Time estimation for remote execution

To calculate the execution time of offloadable modules at remote hosts and unoffloadable modules at local device, we need to consider the time for transmitting the modules, execute the modules and collect the results back to the cloudlet. The time for transmitting module $m \in \mathcal{M}$ from the cloudlet to device $k \in \mathcal{K}$ can be expressed as,

$$\mathcal{T}_{m,k}^t = \left(\frac{\mathcal{S}_m^u}{\mathcal{B}_{k|k=0}^d} + \frac{\mathcal{S}_m^o}{\mathcal{B}_{k|k>0}^d} \right) \times y_{m,k}, \tag{3}$$

where, $\mathcal{B}_k^d$ represents the available bandwidth for data transmission from cloudlet to device $k \in \mathcal{K}$; $k = 0$ again represents the donee device which executes the unoffloadable

modules. Now, the time for execution of module $m \in \mathcal{M}$ in device $k \in \mathcal{K}$ of MDC can be represented as,

$$\mathcal{T}^x_{m,k} = \left(\frac{\mathcal{S}^u_m}{\mu_{k|k=0}} + \frac{\mathcal{S}^u_m}{\mu_{k|k>0}} \right) \times y_{m,k}. \tag{4}$$

After completion of execution of a module, all devices including the donee will transmit the result back to the cloudlet. The time required to send the execution result of module $m \in \mathcal{M}$ from device $k \in \mathcal{K}$ to cloudlet can be calculated as,

$$\mathcal{T}^r_{m,k} = \frac{\mathcal{W}_m}{\mathcal{B}^u_k} \times y_{m,k}, \tag{5}$$

where, $\mathcal{W}_m$ represents the size of results produced after the execution of module m. Therefore, the total execution time of module $m \in \mathcal{M}$ in device $k \in \mathcal{K}$ can be obtained as,

$$\mathcal{T}^c_{m,k} = \mathcal{T}^t_{m,k} + \mathcal{T}^x_{m,k} + \mathcal{T}^r_{m,k}. \tag{6}$$

Now, for calculating the completion time of a module, we have summed up the total time required to execute the module from the beginning of execution of the application. This is performed by adding the execution time of the module ($\mathcal{T}^c_{m,k}$) with the total completion time of it's parent ($\mathcal{T}_n(r)|n \in \Pi_m$). However, a child may not always depends completely on the result of it's parents. It may start independently and after completion of a certain percentage of execution it requires the results from it's parents. This dependency relation is represented by $\phi_{n,m}$. Therefore, the total completion time of module $m \in \mathcal{M}$ is given by,

$$\mathcal{T}_m(r) = max\,(\mathcal{T}^c_{n,\bar{k}}) + \mathcal{T}^c_{m,k}(1 - \phi_{n,m}); \forall n \in \Pi_m. \tag{7}$$

The total time required for execution completion of the application is therefore, the largest time required for completion among all the leaf modules. The total time required for execution of the complete application is represented as,

$$\mathcal{T}(r) = \max\,(\mathcal{T}_m(r)); \forall m \in L. \tag{8}$$

In order to evaluate the completion time of an application in MDC and compare among different scheduling arrangements, scheduling speedup factor needs to be calculated. The speedup factor of a scheduling can be represented as,

$$\mathcal{T}(f) = 1 - \frac{\mathcal{T}(r)}{\mathcal{T}(l)}. \tag{9}$$

3.2.5 Reputation value calculation

For successful execution of offloaded modules reliable donor selection is a prerequisite. If a low performing or unreliable donor is selected for execution of a module, it turns into an unsuccessful execution, increasing response time and hence deteriorates the overall performance of offloading mechanism. Any device may advertise itself as a first-rate donor with high computing capability whereas its successful execution rate might be very poor. The reputation parameter can be used to guard against such unqualified donors. The reputation of a device is calculated as,

$$\Omega_k = \delta \times \Omega_k + (1 - \delta) \times \frac{\sum_{m \in \mathcal{R}} c_{m,k}}{\sum_{m \in \mathcal{R}} y_{m,k}}, \tag{10}$$

where, δ is a relative weight parameter and takes value from the range [0,1]; $c_{m,k}$ is a binary variable having value 1 when, module $m \in \mathcal{R}$ is completed successfully in device $k \in \mathcal{K}$ and, 0 otherwise. Similarly, binary variable $y_{m,k}$ is set to 1, if $m \in \mathcal{R}$ is executed on device $k \in \mathcal{K}$, and 0 otherwise.

While scheduling the offloadable modules of an application, reputation of the mobile devices for task execution needs to be considered. Involving devices with higher reputation for execution of the modules of an application increases the execution reliability. Calculation of total reputation for scheduling all the modules of an application can be expressed as,

$$\Omega_t = \sum_{k \in \mathcal{K}} \Omega_k \times y_{m,k}. \tag{11}$$

As soon as a registered device comes in contact with a cloudlet, the RMM communicates with the RWM and loads reputation value (Ω_k) of that device. Whenever execution of a particular application ends, the RWM calculates reputation value of all the involved mobile devices. Since this work mainly focuses on making the offloading decision optimally, we do not discuss on calculation of reputation value further in this paper.

3.2.6 Energy required for remote execution

Since mobile devices suffer from the scarcity of energy, we need to calculate the total energy that will be required to offload a module to a donee device. To calculate the total energy consumed to offload a module, we need to consider the energy required for the transmission and execution of the module and collection of the result back. Energy required to transmit module $m \in \mathcal{M}$ from cloudlet to device $k \in \mathcal{K}$ is given by,

$$\mathcal{E}^t_{m,k} = \mathcal{T}^t_{m,k} \times \epsilon^t_k \times y_{m,k}, \tag{12}$$

where, ϵ^t_k represents the energy consumption rate for transmission by device $k \in \mathcal{K}$. Now, the energy required to execute module $m \in \mathcal{M}$ in device $k \in \mathcal{K}$ is expressed as,

$$\mathcal{E}^x_{m,k} = \mathcal{T}^x_{m,k} \times \epsilon^x_k \times y_{m,k}. \tag{13}$$

Similarly, energy required to transmit output of module $m \in \mathcal{M}$ from device $k \in \mathcal{K}$ to cloudlet is,

$$\mathcal{E}^r_{m,k} = \mathcal{T}^r_{m,k} \times \epsilon^r_k \times y_{m,k}, \tag{14}$$

where, ϵ^x_k and ϵ^r_k represents the energy consumption rate of device $k \in \mathcal{K}$ for execution and result transmission respectively.

3.2.7 Optimal selection of mobile server

Now to select the optimal set of remote mobile devices for the execution of offloadable modules, we need to choose those devices for which the execution delay of offloadable and unoffloadable modules gets minimized while total reputation of all the scheduling devices gets maximized. The objective function for the selection of remote mobile devices is formulated as,

$$\text{Maximize: } \{\mathcal{Z} = \alpha \times \mathcal{T}(f) + (1-\alpha) \times \Omega_t\}. \tag{15}$$

Here, weight factor α has been used to represent relative priority between application completion time and device reputation. The value of α can be determined by the requirement of the application. Time sensitive applications can set a higher of α while applications requiring high reliability can choose a lower percentage of α.

Constraints Each module should be executed in a single device at a time.

$$\sum_{k \in \mathcal{K}} y_{m,k} = 1; \forall m \in \mathcal{M} \tag{16}$$

The offloadable computation time of the modules through MDC should be less than local computation time of the whole application, i.e.,

$$\mathcal{T}(r) < \mathcal{T}(l). \tag{17}$$

Participating node signal strength should be greater than a certain minimum threshold, i.e.,

$$\mathcal{P}_k > \gamma; \forall k \in \mathcal{K}, \tag{18}$$

where, $\mathcal{P}_k$ represents the signal strength of device $k \in \mathcal{K}$ which is obtained through simulation process; and, γ represents the threshold value of signal strength that must be satisfied by a potential donor.

Participating device energy, after execution, should be greater than a certain minimum threshold, that is,

$$E_k > \mathcal{E}^t_{m,k} + \mathcal{E}^x_{m,k} + \mathcal{E}^r_{m,k} + \Psi; \forall k \in \mathcal{K}, \forall m \in \mathcal{M}, \tag{19}$$

where, Ψ represents the energy threshold that the donor devices must hold after the completion of the execution.

During the execution and transmission period, the participating devices will be available within the range of the cloudlet, i.e.,

$$\mathcal{T}_m(r) < \lambda_k; \forall m \in \mathcal{M}, \forall k \in \mathcal{K}, \tag{20}$$

where, λ_k represents the associativity time of device $k \in \mathcal{K}$ with the cloudlet.

All the unoffloadable modules ($m \in \mathcal{L}$) must have to be executed on the local device

$$y_{m,k} = \begin{cases} 1, \text{ if } k = 0 \\ 0, \text{ otherwise.} \end{cases} \tag{21}$$

Note that, the objective function of the proposed TESAR algorithm provided in equation (15) selects those mobile devices for which total execution time in remote mobile device cloud is minimum and have the highest previous reputation of execution. It is a multi-objective mixed integer linear programming (MILP) problem that has both combinatorial and continuous constraints. To solve the MILP problem, the NEOS optimization tool [39] has been used to find the impact of optimization function parameters and the optimal mapping between modules and donor devices for task allocation and scheduling in TESAR. Two Intel Xeon E5-2698@2.3 GHz CPU with 192GB RAM has been used to find the optimal scheduling for an application containing 12–15 modules and 60–80 mobile devices. Note that, with the increase of number of modules and available mobile devices, real-time solution of TESAR becomes intractable in a typical cloudlet and thus the problem can be grouped as NP-complete one [40]. However, the constraints (16–21) facilitate us to significantly reduce the input sets in TESAR environment and thus the optimal solution was found in polynomial time.

4 Performance evaluation

In this section, we discuss the emulation testbed that is used to implement the proposed module offloading algorithm TESAR and compare the obtained results with state-of-the-art works. We compare the performance of TESAR with the following algorithms:

Table 2 Device settings

Device	Model	OS Version	RAM	CPU
Laptop (cloudlet)	ASUS ZenBook UX303LN	Windows 10	8GB	Core i5-5200U 2.20GHz
Cell phone (donee)	Sony LT18i	Android 4.0.4	512 MB	1.4 GHz Scorpion
Tablet PC (donor)	Symphony T8Q	Android 4.2.1	1 GB	Quad-core 1.2 GHz Cortex A7
Cell phone (donor)	Walton Primo X2mini	Android 4.2.1	1 GB	Quad-core 1.5 GHz Cortex-A7

- *OMDC* In OMDC [13], the application modules are assigned to different available donors in a round robin scheduling order.
- *Honeybee* In Honebee [36], application modules are scheduled on different donor devices based on availability in a purely random fashion. If a poor donor was chosen for a module, the work stealing mechanism is applied to take out the module (from the poor donor) and is executed later on a computation rich donor.
- *Random* In this mechanism, the modules are assigned to different donor devices randomly without considering the device status.

4.1 Experimental testbed

To evaluate the performance of our proposed TESAR, an emulation testbed has been set up by implementing an Android application on a number of heterogeneous mobile devices. The cloudlet functionalities are implemented on a laptop through which all the mobile devices are connected. The mobile devices and the laptop communicate to each other via a Wi-Fi access point. Different parameters and their values used to carry out the emulation are summarized in Table 2.

We consider prime number calculation problem as an experimental prototype to represent a compute-intensive application. Generation of prime numbers with a large range requires a lot of computation. This particular problem can easily be subdivided into several modules that are passed through Algorithm 1 to construct the parent-child dependency tree. Then, we run the objective function on this set of modules for distributing the execution of modules on nearby donor devices.

In this experiment, the prime number problem produces primes between 1 and 300,000, where the complete range is divided into modules of different size. The number of modules varies from 4 to 12 according to the need of the experiment. Total number of available devices were 12 which is also varied for measuring different performance metrics. First module of the application is always executed on the donee device. The donee device can execute one or more modules while the rest of the modules are offloaded to be executed on the donor devices.

A device must contain a certain percentage of remaining battery power (ψ) for self-sustainability and a minimum of -80 dB signal strength to be a candidate donor. The ψ is a system defined parameter and its value can be tuned following the needs of the computing environment and without loss of generality, we have kept $\psi = 20\%$ in our experiments. The access point that has been used to connect the mobile devices with the cloudlet supports IEEE 802.11b/g/n and can achieve maximum 150 Mbps data rate through different

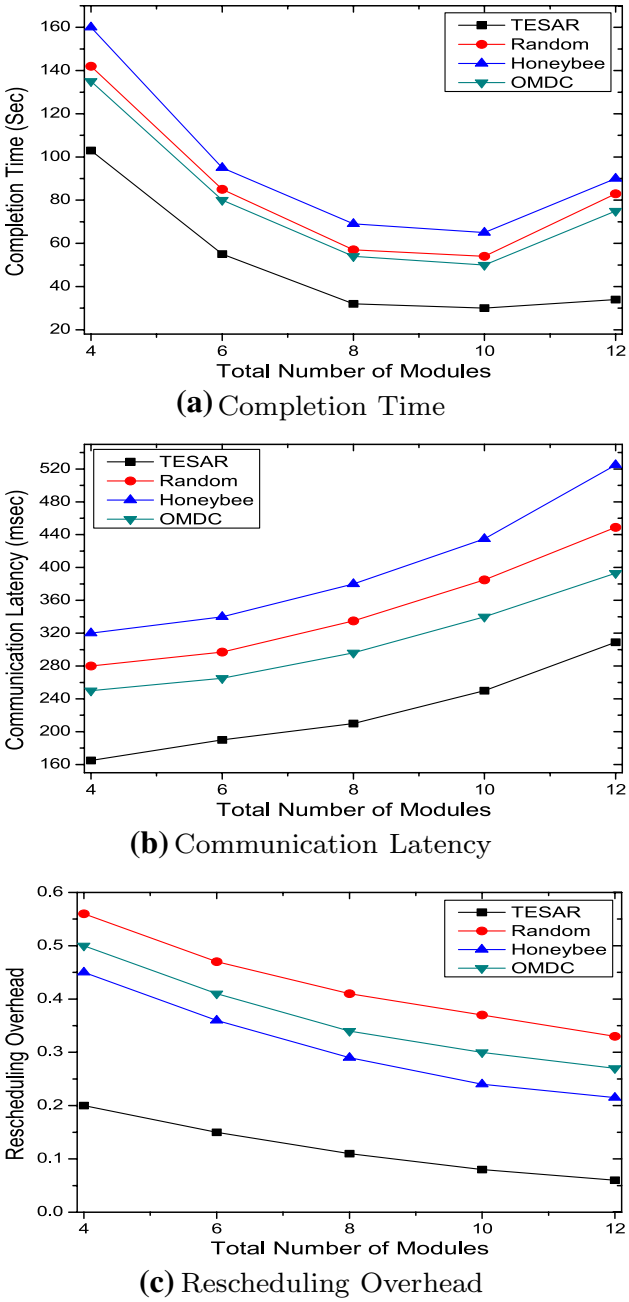
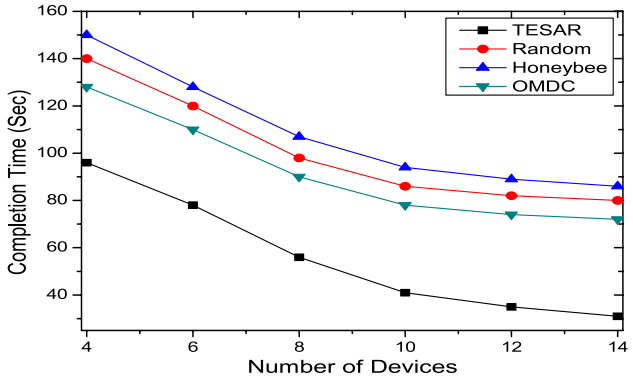


Fig. 4 Impacts of increasing number of modules in an application

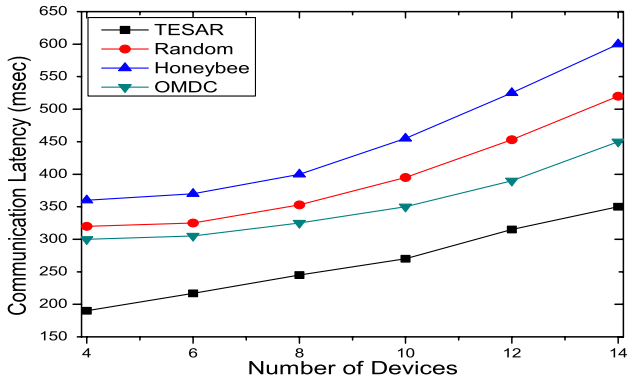
channels [41]. The value of α has been chosen to be 0.6 to give emphasis on the execution time. All the experiments have been conducted for 20 times and the obtained results are averaged. The local device takes 235 s on an average to execute the application without offloading.

4.2 Results and discussion

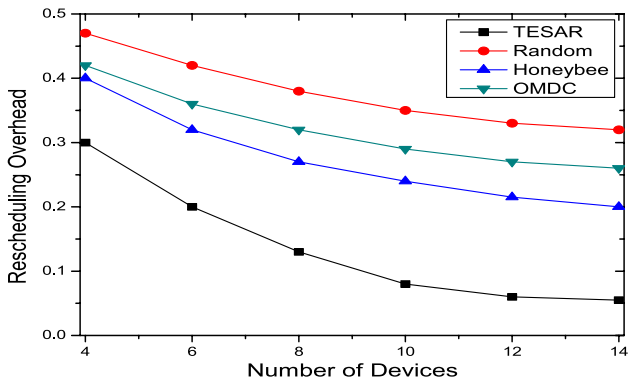
This subsection provides the experimental result and analysis of our proposed TESAR system with other benchmark



(a) Completion Time

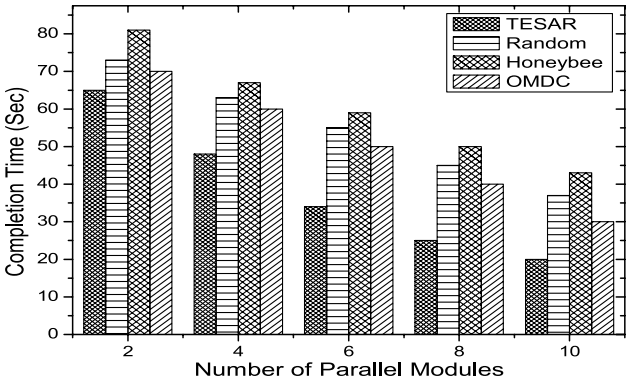


(b) Communication Latency

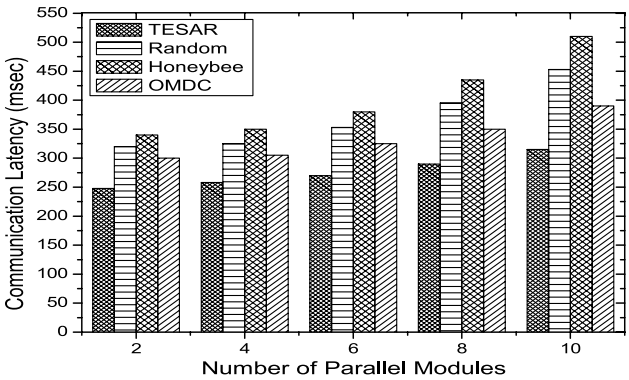


(c) Rescheduling Overhead

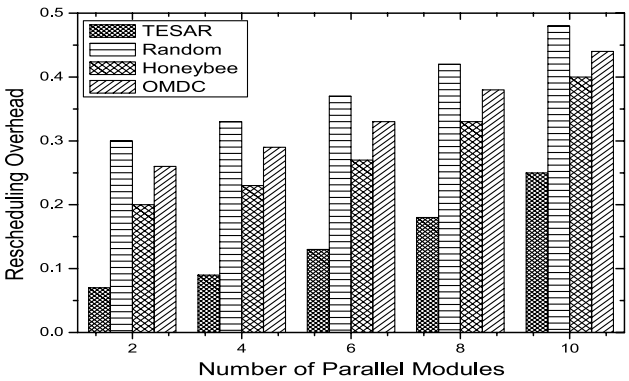
Fig. 5 Impacts of increasing number of donor devices



(a) Completion Time



(b) Communication Latency



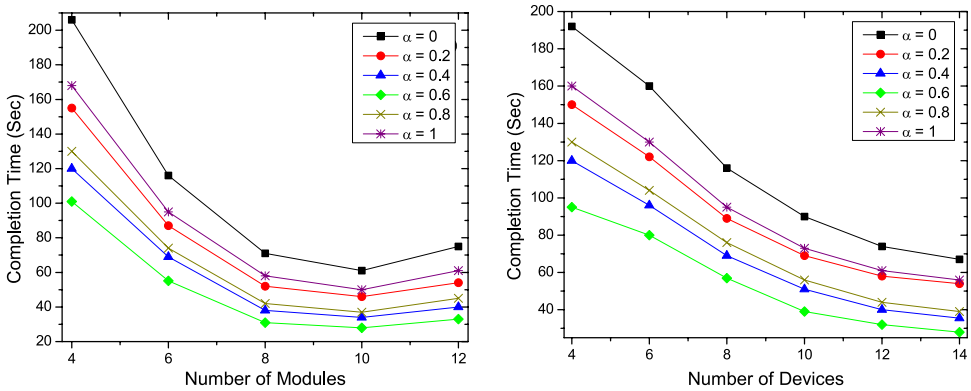
(c) Rescheduling Overhead

Fig. 6 Impacts of increasing number of parallel modules

solutions. In most of the cases, random allocation method fails to execute the allocated module as it doesn't consider the capability of the donor device and handles failed modules. The results depict that, the magnitude of transmission time is negligible compared to completion time. The results for random allocation are obtained from the successful completion of application executions only.

4.2.1 Impact of number of modules in an application

Figure 4 shows the impact of varying the total number of modules in an application on the performances of the studied systems. Figure 4a shows that, initially, the total completion time is decreased significantly with the increasing number of application modules in all the studied systems. Such



(a) Increasing number of modules **(b)** Increasing number of devices

Fig. 7 Impact of α value on the performance of TESAR system

behavior is theoretically expected as well since the scope of parallel execution is enhanced with the number of modules. However, after reaching at a certain level of partitioning (10 modules in the figure), the completion time starts increasing gradually with the number of modules. This is due to the fact that, as the number of modules increases, the assignment of modules to relatively poor donor devices also increases and communication latency among the interdependent modules is increased with the same rate. In case of communication latency (Fig. 4b), with the rise of the number of modules, transmission time and output reception time increases for all the approaches. Figure 4c shows the result of rescheduling overhead with the growth of number of modules. As the number of modules increases, the module size decreases and hence the rescheduling overhead is also decreased. However, our TESAR system outperforms all others with respect to completion time, communication latency and rescheduling overhead since it selects devices with higher signal strength, reliability and associativity period.

4.2.2 Impact of number of devices

With increasing number of devices, the opportunity of selecting more suitable candidates for code offloading is enhanced, resulting in better performances in completion time as well as rescheduling overhead of modules, as shown in Fig. 5a, c. However, with the increase in the number of devices, communication latency for code offloading is increased gradually (Fig. 5b). Nevertheless, since TESAR method selects devices with high reputation, it can avoid rescheduling of application modules to a large extent and hence it experiences better performance compared to state-of-the-art offloading algorithms.

4.2.3 Impact of number of parallel modules

Comparative study among the systems on varying number of parallel modules is illustrated in Fig. 6. Here, the number of total modules is fixed at 12, amongst which the number of parallel modules has been increased from 2 to 10. Fig. 6 shows that, the completion time of the application decreases gradually with the number of parallel modules. However, computation latency and module rescheduling overhead rise for higher number of parallel modules. This is caused by increased communication latency among the higher number of modules. Again, the likelihood to module rescheduling increases with growing number of parallel executable modules. However, the proposed TESAR system considers partial dependency and it selects optimal devices for code offloading and hence outperforms compared to other offloading algorithms under study.

4.2.4 Impact of α value on the performance of TESAR system

Figure 7 shows performances of the proposed TESAR system in terms of completion time of applications with respect to increasing number of devices and modules for different values of α. The graph reveals the fact that, the proposed TESAR system provides the worst completion time for $\alpha = 0$. The completion time is decreased with the gradual increase in α value. This is because, with the increase of α, the algorithm chooses devices having high computational speed and reasonable reliability. It exhibits the optimal behavior when α takes the value of 0.6. However, further increase of α value

starts increasing the completion time again. This is due to the fact that, much higher value of α forces the system to choose devices offering reduced reliability, causing a number of modules experience rescheduling and therefore, completion time of the application is increased.

5 Conclusion

In this paper, we have focused on strategies for code offloading to surrounding mobile devices instead of distant remote cloud. Offloading decision has been implemented to make a tradeoff between execution speedup and reliability for compute-intensive applications. The proposed TESAR system employs cloudlet infrastructure to coordinate apportion of application into modules and to distribute on different donor devices for faster execution. The system outperforms as the best donor devices are extracted from the set of candidate donors by considering offered computation speed, reliability, signal strength and available energy. Simultaneous execution of parallel modules with most suitable donor device achieves better result in terms of execution time, communication latency and rescheduling overhead compared to the state-of-the-art works for varying number of modules and devices. In future, this work can be extended further to develop an incentive mechanism for providing rewards to different donor devices according to their execution performances. Such kind of incentives might encourage a mobile device to share its resources and to act as a donor.

Acknowledgements Special thanks to the Information and Communication Technology Department of the Government of Bangladesh for student fellowship.

References

1. Conti, M., Mascitti, D., Passarella, A.: Offloading service provisioning on mobile devices in mobile cloud computing environments. In: European Conference on Parallel Processing, pp. 299–310. Springer, New York (2015)
2. http://www.statista.com/topics/1002/mobile-app-usage/. Mobile-app usage overview. Access Date 10 Feb 2017
3. Murray, D.G., Yoneki, E., Crowcroft, J., Hand, S.: The case for crowd computing. In: Proceedings of the Second ACM SIGCOMM Workshop On Networking, Systems, and Applications on Mobile Handhelds, pp. 39–44. ACM (2010)
4. Cuervo, E., Balasubramanian, A., Cho, D.-k., Wolman, A., Saroiu, S., Chandra, R., Bahl, P.: Maui: making smartphones last longer with code offload. In: Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services, pp. 49–62. ACM (2010)
5. Kosta, S., Aucinas, A., Hui, P., Mortier, R., Zhang, X., Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading. In: INFOCOM, 2012 Proceedings IEEE, pp. 945–953. IEEE (2012)
6. Huang, D., Wang, P., Niyato, D.: A dynamic offloading algorithm for mobile computing. Wirel. Commun. IEEE Trans. **11**(6), 1991–1995 (2012)
7. Yang, L., Cao, J., Yuan, Y., Li, T., Han, A., Chan, A.: A framework for partitioning and execution of data stream applications in mobile cloud computing. ACM SIGMETRICS Perform. Eval. Rev. **40**(4), 23–32 (2013)
8. Jia, M., Cao, J., Yang, L.: Heuristic offloading of concurrent tasks for computation-intensive applications in mobile cloud computing. In: Computer Communications Workshops (INFOCOM WKSHPS), 2014 IEEE Conference on IEEE, pp. 352–357 (2014)
9. Qian, H., Andresen, D.: Extending mobile device's battery life by offloading computation to cloud. In: Mobile Software Engineering and Systems (MOBILESoft), 2015 2nd ACM International Conference on, IEEE, pp. 150–151 (2015)
10. Cheng, Z., Li, P., Wang, J., Guo, S.: Just-in-time code offloading for wearable computing. Emerg. Topics Comput. IEEE Trans. **3**(1), 74–83 (2015)
11. Khoda, M.E., Razzaque, M.A., Almogren, A., Hassan, M.M., Alamri, A., Alelaiwi, A.: Efficient computation offloading decision in mobile cloud computing over 5g network. Mob. Netw. Appl. **21**(5), 777–792 (2016)
12. Mtibaa, A., Harras, K., Fahim, A., et al.: Towards computational offloading in mobile device clouds. In: Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference, vol. 1, pp. 331–338. IEEE (2013)
13. Mtibaa, A., Harras, K.A., Habak, K., Ammar, M., Zegura, E.W.: Towards mobile opportunistic computing. In: Cloud Computing (CLOUD), 2015 IEEE 8th International Conference on IEEE, pp. 1111–1114 (2015)
14. Li, J., Bu, K., Liu, X., Xiao, B.: ENDA: embracing network inconsistency for dynamic application offloading in mobile cloud computing. In: Proceedings of the Second ACM SIGCOMM Workshop on Mobile Cloud Computing, pp. 39–44. ACM (2013)
15. Kulkarni, V., Moro, A., Garbinato, B.: Mobidict: a mobility prediction system leveraging realtime location data streams. In: Proceedings of the 7th ACM SIGSPATIAL International Workshop on GeoStreaming, p. 8. ACM (2016)
16. Satyanarayanan, M., Bahl, P., Caceres, R., Davies, N.: The case for vm-based cloudlets in mobile computing. Pervasive Comput. IEEE **8**(4), 14–23 (2009)
17. Jararweh, Y., Tawalbeh, L., Ababneh, F., Dosari, F.: Resource efficient mobile computing using cloudlet infrastructure. In: Mobile Ad-hoc and Sensor Networks (MSN), 2013 IEEE Ninth International Conference on IEEE, pp. 373–377 (2013)
18. Zhang, Y., Niyato, D., Wang, P., Tham, C.-K.: Dynamic offloading algorithm in intermittently connected mobile cloudlet systems. In: IEEE International Conference on Communications (ICC), pp. 4190–4195. IEEE (2014)
19. Mahmud, M.R., Afrin, M., Razzaque, M.A., Hassan, M.M., Alelaiwi, A., Alrubaian, M.: Maximizing quality of experience through context-aware mobile application scheduling in cloudlet infrastructure. Softw. Pract. Exp. **46**(11), 1525–1545 (2016)
20. Chen, X., Jiao, L., Li, W., Fu, X.: Efficient multi-user computation offloading for mobile-edge cloud computing. IEEE/ACM Trans. Netw. **24**(5), 2795–2808 (2016)
21. Wang, C., Li, Y., Jin, D.: Mobility-assisted opportunistic computation offloading. Commun. Lett. IEEE **18**(10), 1779–1782 (2014)

22. Drolia, U., Martins, R., Tan, J., Chheda, A., Sanghavi, M., Gandhi, R., Narasimhan, P.: The case for mobile edge-clouds. In: Ubiquitous Intelligence and Computing, 2013 IEEE 10th International Conference on and 10th International Conference on Autonomic and Trusted Computing (UIC/ATC), , pp. 209–215. IEEE (2013)
23. Bhardwaj, K., Sreepathy, S., Gavrilovska, A., Schwan, K.: Ecc: edge cloud composites. In: Mobile Cloud Computing, Services, and Engineering (MobileCloud), 2014 2nd IEEE International Conference on IEEE, pp. 38–47 (2014)
24. Fernando, N., Loke, S.W., Rahayu, W.: Computing with nearby mobile devices: a work sharing algorithm for mobile edge-clouds. IEEE Trans. Cloud Comput. **99**, 1–1 (2016)
25. Huerta-Canepa, G., Lee, D.: A virtual cloud computing provider for mobile devices. In: Proceedings of the 1st ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond, p. 6. ACM (2010)
26. Shi, C., Lakafosis, V., Ammar, M.H., Zegura, E.W.: Serendipity: enabling remote computing among intermittently connected mobile devices. In: Proceedings of the Thirteenth ACM International Symposium on Mobile Ad Hoc Networking and Computing, pp. 145–154. ACM (2012)
27. Shi, C., Ammar, M.H., Zegura, E.W., Naik, M.: Computing in cirrus clouds: the challenge of intermittent connectivity. In: Proceedings of the first edition of the MCC workshop on mobile cloud computing, pp. 23–28. ACM (2012)
28. Barroso, L.A., Hölzle, U.: The case for energy-proportional computing. Computer **40**(12), 33–37 (2007)
29. Carroll, A., Heiser, G.: An analysis of power consumption in a smartphone. In: USENIX Annual Technical Conference, vol. 14. Boston (2010)
30. Rodríguez, J.M., Mateos, C., Zunino, A.: Are smartphones really useful for scientific computing? In: International Conference on Advances in New Technologies, Interactive Interfaces, and Communicability, pp. 38–47. Springer, New York (2011)
31. Ren, J., Zhang, Y., Zhang, K., Shen, X.: Exploiting mobile crowdsourcing for pervasive cloud services: challenges and solutions. IEEE Commun. Mag. **53**(3), 98–105 (2015)
32. Saha, S., Habib, M.A., Razzaque, M.A.: Compute intensive code offloading in mobile device cloud. In: Region 10 Conference (TENCON), 2016 IEEE, pp. 436–440. IEEE (2016)
33. Chun, B.-G., Maniatis, P.: Augmented smartphone applications through clone cloud execution. In: Proceedings of the 12th Conference On Hot Topics in Operating Systems, USENIX Association, pp. 8 (2009)
34. Marinelli, E.E.: Hyrax: Cloud Computing on Mobile Devices Using Mapreduce, Citeseer (2009)
35. Fahim, A., Mtibaa, A., Harras, K.A.: Making the case for computational offloading in mobile device clouds. In: Proceedings of the 19th Annual International Conference on Mobile Computing & Networking, pp. 203–205. ACM (2013)
36. Fernando, N., Loke, S.W., Rahayu, W.: Honeybee: a programming framework for mobile crowd computing. In: International Conference on Mobile and Ubiquitous Systems: Computing, Networking, and Services, pp. 224–236. Springer, New York (2012)
37. Fernando, N., Loke, S.W., Rahayu, J.W.: Mobile crowd computing with work stealing. In: NBiS, pp. 660–665 (2012)
38. Habak, K., Ammar, M., Harras, K.A., Zegura, E.: Femto clouds: Leveraging mobile devices to provide cloud service at the edge. In: 2015 IEEE 8th International Conference on Cloud Computing, pp. 9–16 . IEEE (2015)
39. http://www.neos-server.org/neos/N. optimization server. Access Date: 10 Feb 2017
40. Hassan, M.M., Song, B., Hossain, M.S., Alamri, A., Alnuem, M.A., Monowar, M.M., Hossain, M.A.: Efficient virtual machine resource management for media cloud computing. TIIS **8**(5), 1567–1587 (2014)
41. http://www.tp-link.com.bd/products/details/cat-15-TD-W8901N.htmlTP-Link TD-W8901N Wireless Router Specification. Access Date: 10 Feb 2017