Cluster Computing (2021) 24:1793–1824
https://doi.org/10.1007/s10586-020-03226-8

Energy and cost trade-off for computational tasks offloading in mobile multi-tenant clouds

Yashwant Singh Patel[1] · Manoj Reddy[1] · Rajiv Misra[1]

Received: 12 February 2020 / Revised: 12 December 2020 / Accepted: 18 December 2020 / Published online: 7 January 2021

Abstract
Mobile cloud computing augments smart-phones with computation capabilities by offloading computations to the cloud. Recent works only consider the energy savings of mobile devices while neglecting the cost incurred to the tasks which are offloaded. We might offload several tasks to minimize the total energy consumption of mobile devices; however, this could incur a huge monetary cost. Furthermore, these issues become more complex in considering the multi-tenant cloud, which is not addressed in literature adequately. Thus, to balance the trade-off between monetary cost and energy consumption of the mobile devices, we need to decide whether to offload the task to the cloud or run it locally. In this article, first, we have formulated a 'MinEMC' optimization problem to minimize both the energy as well as the monetary cost of the mobile devices. The 'MinEMC' problem is proven to be NP-hard. We formulate a special case with an equal amount of resource requirement by each task for which a polynomial-time solution is presented. Further various policies are proposed, the cloud can employ to solve the general case. Then we proposed an efficient heuristic named 'Off-Mat' based on distributed stable matching, the solution for which determines whether the tasks are to be offloaded or not under multi-constraints. We also analyze the complexity of this proposed heuristic algorithm. Finally, performance evaluation through simulation results demonstrates that the Off-Mat algorithm attains high-performance in computational tasks offloading and scale well as the number of tenants increases.

Keywords Computation offloading · Distributed algorithm · Stable matching · Mobile cloud computing · Multi-tenancy

1 Introduction

Nowadays, mobile devices (e.g., tablets, smartphones, and smartwatches) have become a necessary part of our daily life as the most valuable and handy communication tools. Mobile users accumulate rich experience of more sophisticated mobile applications such as gaming, web surfing, and navigation, that run on either the devices or distant servers using wireless networks [1, 2]. However, with these improvements, mobile devices still encounter many resource challenges (e.g., shorter battery lifetime, memory, storage, and processor performance) and communications (e.g., network bandwidth and mobility) [3]. Mobile devices typically have limited computational power and computing resources when compared to desktop devices. Thus, the essential, challenging issue for mobile devices is to improve battery life while running complex applications. To overcome the performance, energy and resource limitation issues, the best possible solution is to adopt cloud technology. As infrastructure-as-a-service (IaaS) model of cloud offers storage, computing, and networking resources to mobile clients (tenants) to deploy instances as virtual machines (VMs) on servers of a data center. Many mobile clients want to receive reliable services in a multi-tenant cloud, while the provider intends to maximize their revenue.

Recent work has recognized the mobile cloud computing as a promising computing infrastructure, where storage and processing of data occurring outside of the mobile device [1, 4]. Various cloud-integrated mobile frameworks e.g. CloneCloud [5], ThinkAir [6], cloudlet [7], DIET [8] are proven to be effective for scientific computing applications.

✉ Yashwant Singh Patel
yashwant.pcs17@iitp.ac.in

[1] Department of Computer Science and Engineering, Indian Institute of Technology Patna, Bihar 801106, India

In mobile cloud computing research, the offloading technique is gaining significant attention because it has augmented the computation potential of mobile devices by relocating computation to the cloud. To realize the computation offloading, the elastic mobile applications of mobile clients can be partitioned into the number of tasks. Such tasks are broadly divided into two groups: offloadable and non-offloadable group [33]. The non-offloadable group runs locally because of the inter-dependencies between tasks. On the other hand, the offloadable group contains independent tasks and can be selected to run on clouds. The significant advantage of the computation offloading technique is energy-saving and improved battery lifetime. Figure 1 illustrates the basic mobile cloud computing model, where mobile users, along with mobile devices, can be connected to servers in multiple ways. One straightforward way is to connect the mobile devices to the Internet via Wi-Fi access points. However, due to its range limitation, a more flexible approach is to use the cellular networks for long-distance connections. In cellular networks, the mobile users are connected to an LTE/3G/4G network via devices such as Base Transceiver Station (BTS) and Mobile Switching Center (MSC). Then, the data is transmitted to the Internet. Such type of connection provides much higher availability in comparison to Wi-Fi because of its high coverage. After the connection establishment, the mobile applications discover a suitable cloud service and send the computational tasks offloading request to cloud and subsequently cloud response to the request. If the tasks are offloaded to the cloud server, its computation results will be returned to the mobile devices [29]. Nevertheless, it persists a challenging task to minimize the offloading decision time and to achieve a mutually beneficial relationship between the tasks of multiple mobile clients and cloud servers. Thus, the effectiveness of computation is measured through four fundamental questions: *whether*, *what*, *when* and *where* to offload.

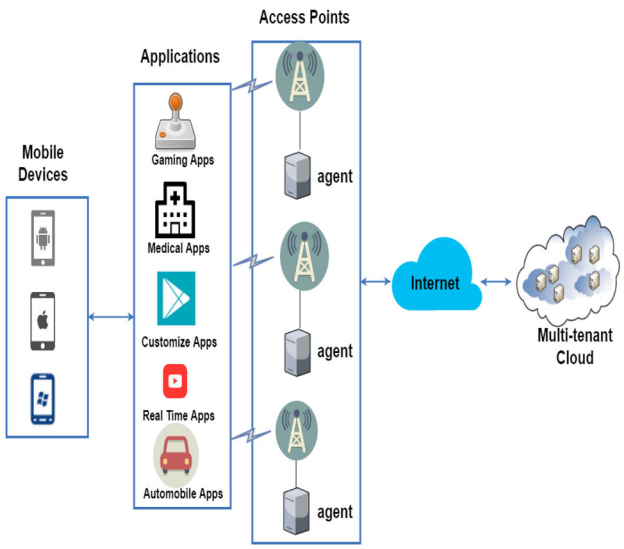

Fig. 1 Heterogeneous mobile cloud computing

Over the last years, different frameworks and schemes have been introducing in offloading techniques [5–8] and [10–14], such approaches either focused on battery lifetime by reducing the energy consumption or the execution time while offloading the compute-intensive tasks to a remote server. In most of the existing frameworks [5, 9], mobile devices directly send the computation offloading requests to the cloud server. Then the server sends back the offloading decision to the mobile devices. However, during the decision-making, mobile clients have to wait for offloading decisions from the cloud which may result in the serious waste of time without determining whether the task offloading is beneficial. Offloading of computational tasks also includes communication cost and monetary cost for accessing the cloud resources. If the total cost of task execution at a cloud server exceeds the maximum completion time limit or the budget of mobile devices, that may lead to the offloading failures. Also, most of the existing works on task offloading are centralized and consider a simple single-tenant environment, where the task of a mobile client is allocated to one cloud server. Thus, it is necessary to design an energy, monetary cost, and delay-aware distributed approach for a multi-tenant environment. Therefore, this work addresses the following significant issues related to the decision of computation offloading such as (i) How to design a framework to reduce the long waiting time during the decision making of computational task offloading in a multi-tenant environment? (ii) How to design a distributed computation offloading strategy under multi-constraints (i.e., completion time, budget, available resources), (iii) How to achieve the trade-off between energy consumption and monetary cost of offloading requests, and (iv) How to use game theory heuristics to maintain the dual preferences between the tasks of multiple mobile clients and servers.

To address the above issues, in this work, we present a novel distributed computation offloading framework named 'Off-Mat' to improve the offloading decision under multi-constraints and minimizing multi-tenant resource contention. Our framework's challenges are to design an offloading strategy to effectively minimize the monetary cost and energy consumption of offloading requests under multiple constraints and propose an efficient, stable matching mechanism to allocate the offloaded tasks to servers based on their preferences. Concretely, the idea of dual preferences enables the game players to express different policies based on the ranked preference list satisfying requirements. In contrast, the concept of stability is applied to address the conflicts of interest among players.

We present a stable matching based approach to solving the computation offloading problem. More specifically, this algorithm works into two phases: In the first phase, it achieves the offloading decision and improves the response time. The second phase performs a many-to-one distributed stable matching to achieve a trade-off of preferences between tasks and servers. The key contributions of this work are as follows:

- Formulated 'MinEMC' problem using a 0-1 integer linear programming (ILP) problem and theoretically proven to be NP-Hard.
- Then, a special case is provided by relaxing the conditions by assuming the same amount of resources are needed by each task, and each of the mobile devices has infinite energy capacity for which a polynomial time optimal solution is proposed using the bipartite weighted graph matching.
- To solve the general case, we propose a distributed heuristic algorithm named *Off-Mat* algorithm, which works in two phases: (i) Off: Offloading phase and (ii) Mat: Matching phase. We also analyze the overall complexity of the Off-Mat algorithm.
- Finally, we demonstrate our proposed approach by numerical results and algorithmic analysis. The experimental results validate the efficacy of the proposed Off-Mat algorithm.

The rest of this work is arranged in following way: Sect. 2 discusses a summary of the existing literature on computation offloading and stable matching. Section 3 presents the novel computational tasks offloading framework. The system model is discussed in Sect. 4. In Sect. 5, problem formulation is provided. In Sect. 6, we design algorithms to resolve the special and general case of the MinEMC problem. We then show the extensive simulations through real-world parameters in Sect. 7. Finally, we summarized the paper with future remarks in Sect. 8.

2 Related work

We have categorized the existing literature from two perspectives: The first part is focused on offloading schemes while the other part is on stable matching. Along with the related work, we have also highlighted their limitations to draw motivations for this work.

2.1 Computation offloading

To fully utilize cloud potentials, the majority of researches is focused on using distant cloud infrastructures with rich resources to augment the abundant capabilities of mobile devices. For example, the model of cloudlet [19, 20] was introduced to utilize nearby devices as cloud servers. It has attracted substantial research attention as it can deliver computing services with minimum energy consumption and delay. Several offloading schemes have been introduced in [11, 12, 21, 22] to decrease the energy-saving level of devices while meeting the constraint of completion time. Wang et al. [10] proposed an application offloading scheme for delay and energy trade-off by using the model of a bipartite graph. Barbera et al. [30] studied the energy cost, bandwidth, and data backup in the real-time system for computation offloading. Huang et al. [31] proposed a dynamic offloading scheme by using Lyapunov based optimization for energy saving while fulfilling the execution time of applications. To ensure the energy fairness at the device end, Song et al. [13] presented an energy-efficient application offloading framework for traffic-energy trade-off. For execution delay optimization, Xia et al. [14] presented a threshold-based model for heterogeneous networks. It offloads the application if the forecasted execution delay is smaller than the acceptable delay; otherwise, the application runs locally on mobile devices. In the work of Huang et al. [32], two approaches are presented: the first one is to manage the offloading users, and the other is the estimate the execution delay for offloading decisions. These approaches followed a traditional offloading technique, where the cloud makes the decision based on the information received from mobile devices and either focused on battery lifetime or the execution time while offloading the applications to the cloud.

To minimize the delay between mobile clients and cloud, the middleware or agent-based architecture is introduced in the literature. To solve the task allocation problem and reduce the overall consumption of energy, Nir et al. [34] proposed a task scheduler model on the centralized broker. In [35] Liu et al. presented a wireless resource scheduling based on back-off technique for mobile agent-based architecture to further enhance the QoS features of real-time streams. In comparison to traditional offloading frameworks, the agent-based framework is advantageous into two aspects: first, it can provide faster-offloading decisions on the neighboring agents itself in place of the remote cloud. Second, it collects the resource utilization details on a periodic basis from the cloud to accommodate the resource demand of mobile clients. Hence it minimizes the number of offloading requests and heavy computation workload directly to the cloud. Concerning energy saving in a multi-user offloading environment, Meskar et al. [36] designed a deadline constraint-based computation offloading approach. In [37, 38] Chen et al. provided a computation based multi-user offloading scheme to reduce the processing time and energy consumption but not consider the completion time. Liu et al. in [39] presented a multi-resource allocation strategy to

optimize the system throughput and service time latency. To optimize the resource allocation, in [40], Kuang et al. proposed a quick response framework. In this framework, the authors have considered the bandwidth constraint and completion time to optimize energy saving. Haber et al. [42] modeled the optimization problem for energy efficiency and computational cost of offloading task in a multi-tier edge based cloud architecture. They have designed an algorithm using branch & bound for finding the optimal solution. Further, they have proposed a low-complexity algorithm and an inflation-based approach for finding a polynomial-time solution. Fang et al. [43] designed scheduling schemes to enhance multi-tenant serving performance for real-time mobile offloading systems. They have implemented a system named ATOMS for computer vision algorithms and proposed a Plan-Scheduling algorithm to improve delay and mitigate resource contention. Ghobaei-Arani et al. [44] presented an organized literature survey of resource management techniques for fog computing. They have designed a classical taxonomy to identify cutting edge methods and also discussed the open issues. Lakhan et al. [45] provided a microservices based mobile cloud platform and designed the application partitioning based task assignment algorithm for robust execution of applications. Verma et al. [46] presented a robust architecture for multimedia applications using mobile cloud computing. Shakarami et al. [47] provided a systematic literature review on computation offloading based on the game theory techniques for mobile edge environment. They have designed a classification to identify state-of-the-art techniques and also discussed the open issues. Nagasundari et al. [48] proposed a service selection scheme for multi-user based computation offloading environment. Further, they have exploited hidden markov model and fuzzy KNN based mobility prediction via cloudlet servers to enhance the framework. De et al. [49] provided multi-level partial and full offloading approaches using cloudlet, public, and private cloud servers. They have further analyzed the delay and power consumption and compared them with the existing offloading methods. For multiplayer online gaming in the cloud, Ghobaei-Arani et al. [50] provided an autonomous resource provisioning architecture. They have designed an adaptive neuro-fuzzy inference system based prediction model to handle workload fluctuations and designed a fuzzy decision tree approach to determine the auto-scaling decisions. Derhab et al. [51] designed a mobile cloud offloading framework for the two-factor mutual authentication applications.They have introduced a decision-making scheme for offloading the authentication application along with its virtual smart card, using energy cost, mobile device's residual energy, and security. Nir et al. [54] proposed a centralized broker-node based architecture for task scheduling mobile cloud computing. The experimental results demonstrated that the offloading with optimization based technique results less energy consumption and monetary cost in comparison with the offloading without optimization using the centralized scheduler. To provide incentives for fog nodes and minimize the computational cost of mobile devices, Chen et al. [55] provided a cost-effective offloading approach for mobile-edge environment with the cooperation between the remote cloud and fog nodes while considering task dependency constraint. Hassan et al. [56] presented a reinforcement learning-based SARSA approach to solve the resource allocation issue in the edge and perform the optimal offloading decision to reduce computing time delay, system cost, and energy consumption. Hassan et al. [57] proposed a deep Q-learning based code offloading strategy in mobile edge for IoT applications. The proposed approach has significantly improved the computation offloading by minimizing the latency of service computing, execution time, and energy consumption. Enayet et al. [58] proposed a mobility-aware resource provisioning framework, named Mobi-Het to enable remote execution of big data tasks on the mobile cloud, which promises higher efficiency in timeliness and reliability. Islam et al. [59] developed an ant-colony based mobility and resource-aware VM migration model for the mobile cloud-based healthcare system in smart cities. Bedi et al. [60] proposed a multi-cloud storage technique for resource-constrained mobile devices to optimize mobile devices' resources and improve the performance of CPU usage, battery consumption, and data usage. Durga et al. [61] designed an efficient context-aware dynamic resource allocation that utilizes the client present context information to meet the performance requirements specified by user. Saleem et al. [62] proposed a dynamic bitrate adaptation strategy using stochastic optimization for maximizing the user's QoE. They have applied video assessment models and QoE feature metrics for evaluation. Elashri et al. [63] provided schemes for efficient offloading decision-making for soft and weakly hard (firm) real-time applications while ensuring the tasks schedulability. Milan et al. [64] designed a bacterial foraging optimization based task scheduling approach using for minimizing the idle time of VMs. However, the aforementioned literature's major limitation is the non-existence of stability inducing offloading under multi-constraints (i.e., completion time, budget, available resources) while minimizing both the monetary cost and energy consumption of mobile devices in the heterogeneous multi-tenant mobile cloud environment. The detailed side-by-side comparison between the proposed approach and the existing techniques discussed in Tables 1 and 2.

Table 1 Comprehensive review of existing computation offloading approaches

Year	Work	Utilized technique	Performance metrics	Evaluation tools	Advantages	Disadvantages
2020	[45]	Application partitioning	Setup time, energy consumption, response time	Java, REST API	Reducing setup time and response time	Lack of data privacy mechanism
2020	[48]	Fuzzy K nearest neighbour (KNN) and Hidden Markov Model	Computation cost	MATLAB	Cost effective service selection	Overhead have not been investigated
2020	[49]	Multilevel full and partial task offloading	Power and delay consumptions	MATLAB	High power and delay efficiency	Low scalability
2020	[51]	Two-factor mutual authentication scheme	Efficiency against different attacks		Ensure both secrecy and authentication properties	High computational complexity
2020	[56]	Reinforcement-learning-based SARSA method	Minimizing system cost in terms of energy consumption and computing time delay	–	Reduced system cost, Better decision making	Scalability issue
2020	[62]	Lyapunov optimization	Average video quality, average bitrate level, and switch frequency	NS-2	Maximize viewer QoE (quality of experience) in adaptive streaming	Heterogeneous wireless environment is not considered
2020	[63]	Dynamic speed scaling	Execution cost, energy efficiency	Amazon EC2 platform	Reduce the power costs, Maximize the power saving	Static power consumption is not considered
2020	[64]	Bacterial foraging optimization	Makespan, running time, imbalance degree, energy consumption	Cloudsim	Low makespan, Less energy consumption	Not evaluated on actual environments
2019	[57]	Deep Q-learning	Execution time, latency, energy consumption	MATLAB	Support parallelism, Better execution time and energy	Dynamic workloads are not used
2019	[42]	Branch-and-Bound, convex approximation method	Computational cost, energy consumption	–	Hierarchical edge-clouds	Problem of users mobility, load balancing
2019	[43]	Plan-Schedule approach	Resource contention, accuracy, delay	Micro Testbed, AWS Testbed	Multi-tenant serving performance is improved	Sub-second latency violations due to waiting time in queue
2019	[50]	ANFIS predictor and Fuzzy decision tree	Mean Square Error, Cost, Response Time	CloudSim	Accuracy, Reducing cost, Reducing response time, High correlation between ANFIS and experimental data	Energy efficiency, throughput, reliability are not considered
2019	[60]	Multi cloud storage approach	Battery consumption, CPU usage and data usage	Android Emulator	Improve resource consumption of mobile devices in both stationary and mobility modes	Privacy, reliability have not considered
2019	[61]	Queuing theory	Response time, energy consumption, throughput	CloudSim	Improve service time and quality of service	Overhead of the proposed approach has not been investigated, Multi-cloud environment is not considered

2.2 Stable matching

The concepts of stable matching have been widely adopted since 1962 when Gale and Shapley presented a deferred acceptance algorithm in their pioneering work for solving the college admission problem [25]. Such type of game theory is successfully applied in many areas. Such as Kim et al. [16] have used the Hospital/Residents based

Table 2 Comprehensive review of existing computation offloading approaches

Year	Work	Utilized technique	Performance metrics	Evaluation tools	Advantages	Disadvantages
2018	[40]	Dynamic programming	Energy saving, request response delay	Custom simulator	Improve response time and energy saving	User mobility, scalability, and heterogeneity have not been considered
2018	[54]	Centralized broker node architecture	Energy consumption, monetary cost	IBM ILOG CPLEX	Minimized the total energy consumption and cost	Task priority, network congestion, and execution redundancy are not considered
2018	[58]	Multi-layer architecture	Throughput, load standard deviation, delay	Custom testbed	Support heterogeneity and mobility, Reduce scheduling delay under high load	Not considering the task priority and admission control
2017	[55]	Greedy offloading policy, simulated annealing technique, brute force method	Response delay, User budget, Service cost	MATLAB	Minimize the application finish time	Mobility, and heterogeneity have not considered
2017	[59]	Ant colony optimization	Completion time, VM migration overhead, resource over-provisioning	Custom testbed	Reduces the average task-execution lifetime, VM migrations Increase the total number of task completed	Mobility aware task execution time and context-awareness requires improvement
2017	[10]	Heuristic-based	Energy consumption, execution delay	AMPL CPLEX, C++ simulator	Reduce energy consumption and execution delay	Not evaluated using a real-world scenario
2017	[36]	Gauss-Seidel method	Energy, number of iterations needed, Solution quality	–	High performance compared to a lower bound on total energy-performance	Communication overhead and user mobility patterns are not considered
2016	[38]	Game theory	System-wide computation overhead, Beneficial users,	–	Achieve superior computation offloading performance and scale well with the users	Scalability issue, Decision making capability requires improvement
2016	[39]	Markov decision process	Throughput, service latency	MATLAB	Better performance over a broad range of environments	User mobility, multicloudlet setup have not been considered
2015	[37]	Game theory	Computing cost, number of iterations, number of messages	–	Achieve efficient computation offloading performance and scale well with the system size	The user mobility patterns are not considered
2020	Proposed	Matching theory (weighted bipartite matching, and stable matching)	Monetary cost, energy consumption, response time, delay, budget, happiness	Custom simulator	Multi-tenancy, stability, reduces response time, and monetary cost, high energy-efficiency	Dynamic workloads are not applied, more QoS metrics are needed

stable matching to resolve the problem of cloud supported smart TV migration. $Many-to-one$ matching approach is similar to the college admission problem [18], where a student can be admitted to a university, on the other hand, a university can admit many students. Similar to this problem, in [17], adopted a college admission based game in which the small scale stations and macrocells (i.e., colleges) are seeking to enroll the users (i.e., students) with given preferences. The other real-world applications of the matching game are: assigning hostel rooms to students and matching medical interns to hospitals etc. Based on the notion of matching theory [23–25], the tasks of multiple mobile client applications and the cloud servers can be identified as the players of two sides in a $many-to-one$ matching game. In particular, one task of the mobile application may be offloaded to one server; on the other hand, one server can host many tasks of different mobile apps depending on its resource availability. However, the

classical theory of stable matching cannot be directly applied in computation offloading scenario as the tasks have different demands of CPU, memory, bandwidth, and storage, etc. and the servers have a different capacity constraint. This problem becomes more complicated because of the size and demand heterogeneity. To clarify such an ambiguous situation, we have developed new preference functions and propose a new distributed stability concept based on a deferred acceptance algorithm and proved its convergence as well as optimality results.

Thus, in the proposed work, the matching game approach is adopted to solve the stable matching issues between the workload i.e., tasks of mobile clients and the cloud servers. This work maintains the trade-off of preferences by creating individual preference set to model each player's interest and stability results as the solution rather than optimality.

3 Off-mat: framework for computational tasks offloading

This section presents the 'Off-Mat' framework with underlying assumptions, components, and their interaction.

3.1 Components of framework

The mobile multi-tenant cloud environment is depicted in Fig. 1. It composes the crowd of mobile devices, elastic mobile applications, computation-intensive tasks, access points (APs), agents, and cloud servers. The mobile clients operating mobile devices are geographically distributed into different regions and lie in the coverage of APs. Each mobile device has some tasks to be offloaded. The agents are active near to the APs and connected to the cloud via the high-speed wireline network. It handles the offloading requests sent by mobile devices. In a multi-tenant cloud, it provides shared computing resources to multiple mobile clients. Cloud have sufficient resources (i.e., compute, storage, and networking) to execute the requests in the form of VMs, but at a time cloud can support limited requests. Therefore, it requires the optimal decision-making method to filter unnecessary requests. The Off-Mat framework is represented in Fig. 2.

In this framework, we have designed middleware for mobile devices as well as for agents. The device middleware is composed of application partitioner, device profiler, offloading manager, and local execution manager. The application partitioner is responsible for partitioning the dependent and independent tasks. The device profiler gathers the information of device resources (i.e., energy, used resources, etc.), currently executing tasks, network resources, i.e., bandwidth. The offloading manager is used to filter the incompetent tasks at the device end. The execution manager follows the decision of the offloading manager. The middleware on the agent is composed of a resource monitor, which periodically collects the servers' resource information from the cloud. The task profiler detects the invalid requests based on resource constraints. The matching engine matches the offloadable tasks to cloud servers while the remote execution manager follows the decision of a matching engine. The middleware at the agent minimizes the request delay and determine the final execution of tasks to cloud servers. Figure 3 depicts the sequence diagram of the computation offloading process.

3.2 Phases of framework

The 2-phase computation offloading framework work in the following phases:

(i) Phase I- Off (Offloading): The offloading workflow starts with the device partitioner that partitioned the application into independent and dependent tasks and sent it to the device profiler. The device profiler gathers the meta-data of device and information such as: currently executing applications, type of device, tasks, network bandwidth, available energy and resources, and sends it to the offloading manager.

Based on the available resource information, the offloading manager decides whether the mobile device can be benefited by the task offloading or not? Here it checks: (i) whether the offloading energy is less than the local energy or not and (ii) whether the required bandwidth is lesser than the available bandwidth. If it satisfies the constraints, then it sends the request to the agent, otherwise, the device profiler kept on collecting the information. The agent periodically collects the cloud server information from the resource monitor. If it receives tasks, then it validates the constraints for maximum task completion time, monetary cost, and cloud resources availability. In case of failure, it filters the useless requests and sends an offloading failure message to the device local execution manager through its remote execution manager. To execute the above process, both mobile and agent perform the asynchronous procedures.

(ii) Phase II-Mat (Matching): If the tasks are offloadable, then it negotiates with the distant cloud for reserving resources for the given time periods and send the request to the matching engine. The matching engine creates the preference sets for mobile client tasks and servers and applies the distributed matching. Then it sends the outcomes to the remote execution manager. The remote execution manager is mainly responsible for sending the offloadable tasks to the cloud servers and finally receiving and returning the computation results to the device's local execution manager. Hence the agent is a critical component

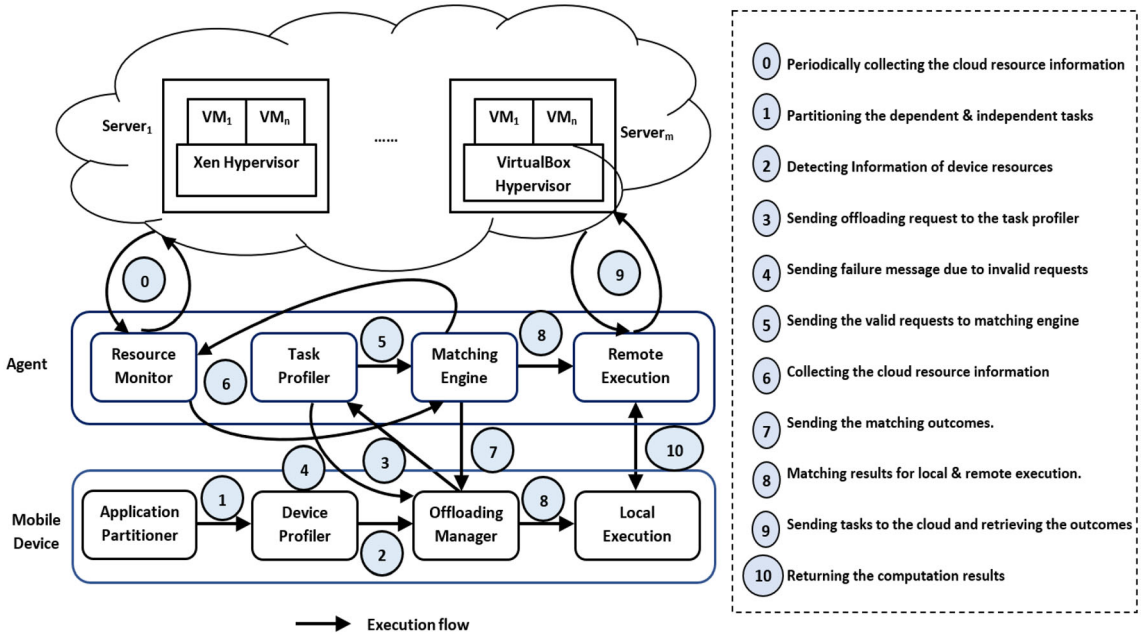

Fig. 2 The Off-Mat computation offloading architecture

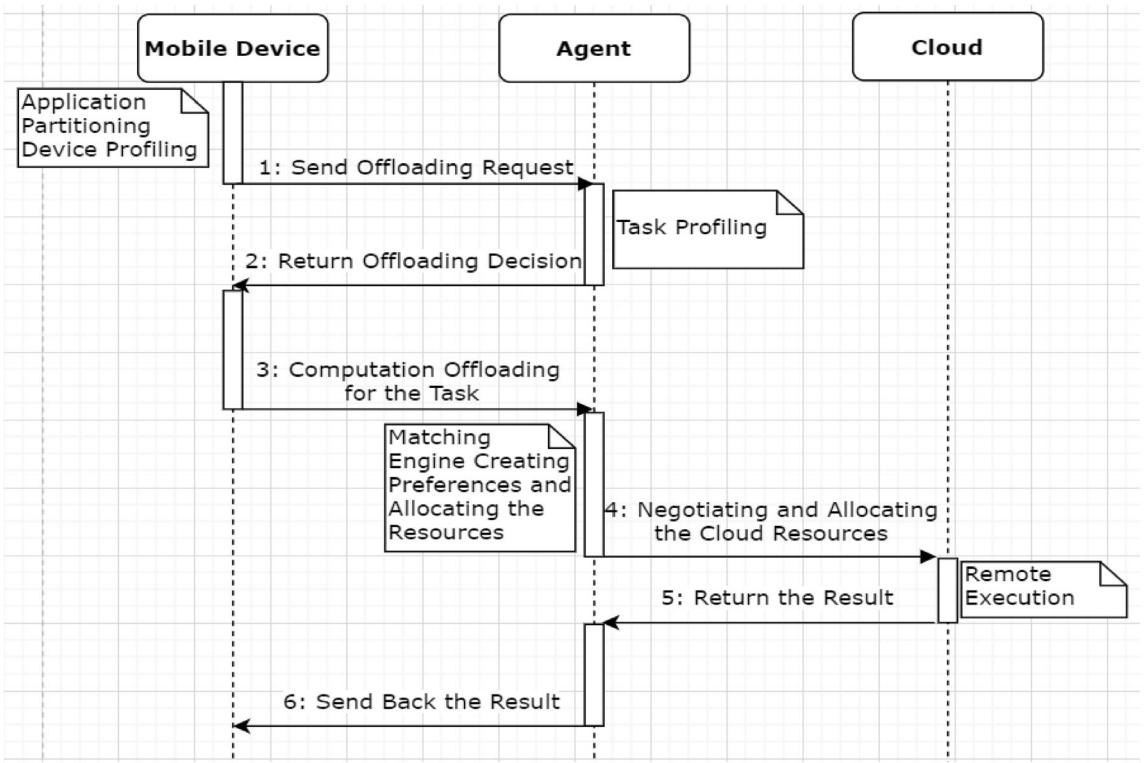

Fig. 3 Sequence diagram of computation offloading process

as it monitors the cloud servers information and filters the device requests based on different parameters. It applies the distributed algorithm to find out the stable matching between tasks of mobile clients and servers. Thus it performs the optimal resource allocation to minimize multi-tenant resource contention. The workflow of the filtering process is shown in Fig. 4. Next, we have analyzed and formulated various task models to be used for this framework.

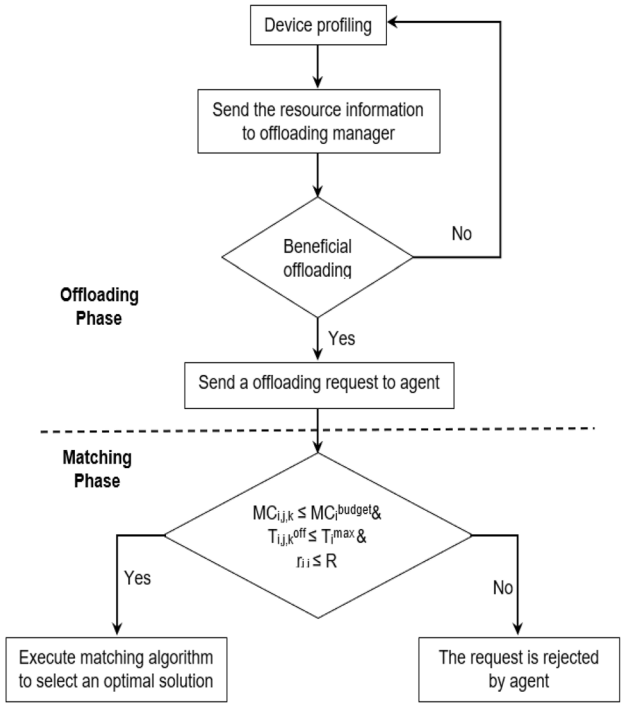

Fig. 4 Off-Mat workflow

4 System model

This section discusses the model of computation & monetary cost and illustrate the matching concepts. The key notations used in work are described in Table 3.

Let us assume a set of D mobile devices, denoted by $MD = \{D_1, D_2, ...\}$. For each mobile device, there are several computational tasks for offloading, denoted by $T = \{t_{i,1}, t_{i,2}..., t_{i,j}\}$, where $t_{i,j}$ refers to the jth task of mobile device i. $r_{i,j}$ denotes the resources (i.e, memory, CPU, storage, and bandwidth etc.) required by task j of device i. Each computational task $t_{i,j} \in \bigcup_{i=1}^{D} MD_i$ can be run locally or offloaded to the k^{th} server. We represent the set of cloud servers by $S = \{s_1, s_2, ...s_k\}$ and the total number of resource available at s_k by R_k. We consider the total available resource at the cloud servers as well as at the devices are enough the execute all the computational tasks of different applications. For each task $t_{i,j}$, if it runs locally, the energy consumption would be $e_{i,j}^{loc}$. We also assume E_i^{rem} is the energy left at the mobile end. Let us denote energy to offload task $t_{i,j}$ to server k as $e_{i,j,k}^{off}$. Now, as we all know that each server will have its own policy as to what kind of tasks to execute. Policies here mean that whether the cloud prefers the one which gives the highest revenue, or it could be like preferring the larger task. These policies are the ones that decide how much monetary cost is incurred during offloading of the computational task to the cloud. As we are using shared multi-tenant cloud resources while maintaining isolation between the tasks offloaded. We denote $MC_{i,j,k}$ as the cost incurred by task $t_{i,j}$ when offloaded to server k.

Table 3 Main notations with descriptions

Symbol	Description
$t_{i,j}$	jth task of mobile device i
MD	Set of mobile devices
$e_{i,j}^{loc}$	Energy to run the task $t_{i,j}$ locally
T_i	Set of tasks at mobile device i
S	Set of cloud servers
$P_{i,j}^{loc}$	Power used in local execution of task $t_{i,j}$
$C_{i,j}^{loc}$	CPU cycles for task $t_{i,j}$
$S_{i,j}^{loc}$	Execution speed of local device i
$C_{i,j,k}^{cloud}$	CPU cycles for task $t_{i,j}$ at cloud server s_k
$S_{i,j,k}^{cloud}$	Execution speed of cloud server s_k
$e_{i,j,k}^{off}$	Energy to offload task $t_{i,j}$ to server k
$e_{i,j,k}^{sent}$	Sending energy
$e_{i,j,k}^{rec}$	Receiving energy
$e_{i,j}^{idle}$	Idle energy consumption
$r_{i,j}$	Resource required by $t_{i,j}$
R_k	Total number of available resources at s_k
$B_{i,j,k}^{send}$	Data bits uploaded
$B_{i,j,k}^{rec}$	Data bits received
$z_{i,j,k}$	Variable to denote if task $t_{i,j}$ is offloaded to cloud server k
$MC_{i,j,k}$	Monetary cost to offload $t_{i,j}$ to server k
$MC_{i,j,k}^{trans}$	Data transfer cost for offloading the task $T_{i,j,k}$
$MC_{i,j,k}^{VM}$	Cost for running task $t_{i,j}$ on cloud VM
$MC_{i,j,k}^{trans}$	The data transfer cost
$T_{i,j}^{loc}$	Time to locally execute task $t_{i,j}$
$T_{i,j,k}^{off}$	Time to offload $t_{i,j}$ and get result back from server k
T_i^{rem}	Maximum allowed time to get tasks results in device i
E_i^{rem}	Residual energy at mobile device i
α	Trade-off preference parameter
β	Policy factor deciding cost per unit time for cloud servers
γ	Cost per megabyte (MB) in network
μ	Matching function

4.1 Computation and communication task models

Next, we have derived task models to compute energy consumption and execution time.

Energy consumption analysis:

(i) Local energy consumption: If the task is decided to run locally then the local energy consumption $e_{i,j}^{loc}$ can be

computed by the power used during local execution $P_{i,j}^{loc}$, number of CPU cycles $C_{i,j}^{loc}$ for task $T_{loc}^{i,j}$ and execution speed $S_{i,j}^{loc}$ of local device.

$$e_{i,j}^{loc} = \frac{C_{i,j}^{loc} \times P_{i,j}^{loc}}{S_{i,j}^{loc}} \tag{1}$$

(ii) Offloading energy consumption: The energy costs of offloading tasks to distance cloud can be calculated as the sum of transmission energy i.e., sending energy $e_{i,j,k}^{sent}$ and receiving energy $e_{i,j,k}^{rec}$ and $e_{i,j}^{idle}$ is the idle energy consumption waiting for the results from the cloud.

$$e_{i,j,k}^{off} = e_{i,j,k}^{sent} + e_{i,j}^{idle} + e_{i,j,k}^{rec} \tag{2}$$

$$e_{i,j,k}^{off} = P_{i,j,k}^{send} \times T_{i,j,k}^{send} + P_{i,j}^{idle} \times T_{i,j}^{idle} + P_{i,j,k}^{rec} \times T_{i,j,k}^{rec} \tag{3}$$

Execution time analysis:

(i) Local execution time: To run the task locally, the local execution time can be determined as the ratio of CPU cycles to execute task $t_{i,j}$ to the execution speed $S_{i,j}^{loc}$ of device.

$$T_{i,j}^{loc} = \frac{C_{i,j}^{loc}}{S_{i,j}^{loc}} \tag{4}$$

(ii) Remote execution time: If the task is granted to execute on the distant cloud, then the remote execution time can be determined by the CPU time consumption and transmission time i.e., sending and receiving time.

$$T_{i,j,k}^{off} = \frac{B_{i,j,k}^{send}}{r_{i,j,k}^{send}} + \frac{C_{i,j,k}^{cloud}}{S_{i,j,k}^{cloud}} + \frac{B_{i,j,k}^{rec}}{r_{i,j,k}^{rec}} \tag{5}$$

where $B_{i,j,k}^{send}$ and $B_{i,j,k}^{rec}$ represents the data bits uploaded and the data bits received respectively.

4.2 Monetary cost model

Monetary cost analysis: Each device has an offloading budget for mobile cloud services set by mobile clients. The monetary cost is the sum of data transferring cost and pubic cloud services cost.

(i) Cost of data transferring: The data transfer cost for offloading the task $T_{i,j,k}$ can be expressed as:

$$MC_{i,j,k}^{trans} = \gamma(B_{i,j,k}^{send} + B_{i,j,k}^{rec}) \tag{6}$$

where γ is the cost per megabyte (MB) in network i.e. 3G, wifi etc.

(ii) Public cloud services cost:

The cost of public cloud services depends on service type and its usage. For running task $t_{i,j}$ on cloud VM, It can be expressed as follows:

$$MC_{i,j,k}^{VM} = \beta \frac{C_{i,j,k}^{cloud}}{S_{i,j,k}^{cloud}} \tag{7}$$

where β denotes the cost per time unit of using the cloud instance which is the factor depending on the policy of the cloud.

Hence the total monetary cost can be expressed as:

$$MC_{i,j,k}^{trans} + MC_{i,j,k}^{VM} \tag{8}$$

$$= \gamma(B_{i,j,k}^{send} + B_{i,j,k}^{rec}) + \beta \frac{C_{i,j,k}^{cloud}}{S_{i,j,k}^{cloud}} \tag{9}$$

4.3 Matching concepts and preference functions

The matching of tasks to servers can be considered as an outcome of a many-to-one matching game. Where multiple tasks can be allocated to one server based on preference functions and matching constraints. In this section, we have defined the preliminaries to explain the concepts of matching theory.

Definition 1 Given the set of tasks T and the set of servers S, mathematically a matching function can be defined as $\mu : T \cup S \Rightarrow 2^{T \cup S}$ such that:

- $\mu(s) \subseteq T$ such that $|\mu(s)| \leq r_s, \forall s \in S$, where $|\mu(s)|$ represents the collective resources of all tasks that are matched to s.
- $\mu(t) \subseteq S$ such that $|\mu(t)| = r_s$, or $|\mu(t)| = 0, \forall t \in T$ and $s \in S$, where $|\mu(t)|$ is the server resources of s that is matched to t and $|\mu(t)| = 0$ means that task t is unassigned.
- $t \in \mu(s)$ if and only if $\mu(t) = s$, $\forall t \in T$ and $s \in S$,

Here, the definition describes that matching is defined to be a *many-to-one* relation, where each cloud server is matched to a subset of tasks. The objective of matching is to obtain an efficient and stable matching. In such matching game, each player specifies their preferences over the other depending on its objective in the mobile cloud computing environment.

Definition 2 A matching μ can be blocked through a agents pair (t, s) if there exists a (t, s) pair with $t \notin \mu(s)$ and $s \notin \mu(t)$ then such kind of pair is termed as blocking pair.

Definition 3 The obtained matching μ is stable if (a) No blocking pair is exist and (b) Each of the tasks are embedded to cloud servers.

Theorem 1 *Stable matchings always exist for a set of marriages.*

Proof This theorem can be proven through the classical deferred acceptance algorithm (DAA) known as the *Gale − Shapley's* algorithm [25] for a stable marriage problem [27]. It applies an iterative procedure and finds a stable set of marriages. To begin with this procedure, let us assume a set of players, say men propose to women based on their set of preferences. It continues until there exists a man who is available and not yet proposed to all women of his set. Then he can propose to the highly preferred woman of his set who also has not yet rejected his proposal. If the woman is available, she *holds* the received proposal on a string to ensure the possibility of some better proposal. If she already received the proposal, then she rejects the least preferred proposal. This procedure is repeated until no further proposal can be formed since no men can propose to the woman more than once. Once the last women receive her proposal, the algorithm stopped and matched each woman to the man (if there exist any) whose proposal she is still holding in her string. The woman-oriented model also operates in a similar fashion by changing the roles of man and woman [26].

For general settings, the marriage model can be extended to the *college admissions* problem [25], where each college is looking for multiple students to admit, and each student aspires to be matched with one college. It is a prominent extension of many-to-one. The resource allocation problem in the cloud environment can be naturally cast as a *stable matching* problem, which resolves the conflicting interests amid all of the stakeholders and achieving stability. Here, we can model mobile tasks as 'students' and servers as 'colleges,' where both are wishing to be matched with each other. The preferences can be transformed to distinct policies. Due to the size heterogeneity of mobile tasks (i.e., CPU, memory, bandwidth, and storage, etc.), the task allocation problem is modeled as a *job-machine stable matching* problem [26], where machines have heterogeneous capacities, and jobs have different sizes. Each machine can contain multiple jobs ensuring the total size of jobs should not exceed its total capacity. Each machine possesses transitive preference with respect to all the acceptable jobs whose size is smaller than the capacity of machine. Equivalently, each job also possesses transitive preference with respect to all the acceptable machines having sufficient capacities to accommodate the job. The job-machine model is a more general type of *many-to-one matching*, and the problem of college admissions can be seen as a special case where all the jobs are having same size representing students [25, 26]. □

5 Problem formulation

The computational tasks offloading problem is formulated over various decision variables. We define energy to run task j at mobile device i locally as $e^{loc}_{(i,j)}$,energy for offloading the task j from mobile device i to server k as $e^{off}_{(i,j,k)}$. Let us define a variant,

$$z_{i,j,k} = \begin{cases} 1, & \text{if task is offloaded} \\ 0, & \text{otherwise} \end{cases} \tag{10}$$

The total energy consumed by tasks of different applications can be defined as

$$\sum_{j=1}^{T}\left(1-\sum_{k=1}^{|S|} z_{i,j,k}\right)e^{loc}_{i,j} + \sum_{j=1}^{T}\sum_{k=1}^{|S|} z_{i,j,k}e^{off}_{i,j,k} \tag{11}$$

The total energy consumption of mobile devices can be defined as

$$\sum_{i=1}^{D}\sum_{j=1}^{T}\left(1-\sum_{k=1}^{|S|} z_{i,j,k}\right)e^{loc}_{i,j} + \sum_{i=1}^{D}\sum_{j=1}^{T}\sum_{k=1}^{|S|} z_{i,j,k}e^{off}_{i,j,k} \tag{12}$$

The total monetary cost by tasks at task $t_{i,j}$ can be defined as

$$\sum_{k=1}^{|S|} z_{i,j,k}MC_{i,j,k} \tag{13}$$

The total monetary cost at the mobile devices from all the tasks can be defined as

$$\sum_{i=1}^{D}\sum_{j=1}^{T}\sum_{k=1}^{|S|} z_{i,j,k}MC_{i,j,k} \tag{14}$$

Now, we formulate the overall objective function of MinEMC problem as follows:

$$\min \sum_{i=1}^{D}\sum_{j=1}^{T}\left(1-\sum_{k=1}^{|S|} z_{i,j,k}\right)e^{loc}_{i,j} + \sum_{i=1}^{D}\sum_{j=1}^{T}\sum_{k=1}^{|S|} z_{i,j,k}e^{off}_{i,j,k} + \alpha\left(\sum_{i=1}^{D}\sum_{j=1}^{T}\sum_{k=1}^{|S|} z_{i,j,k}MC_{i,j,k}\right) \tag{15}$$

subject to

$$\sum_{k=1}^{|S|} z_{i,j,k} \leq 1, \forall t_{i,j} \in \bigcup_{i=1}^{D} MD_i \tag{16}$$

$$\sum_{j=1}^{T}\sum_{k=1}^{|S|} z_{i,j,k}MC_{i,j,k} \leq MC_i^{budget}, \forall i \in (1,2,....D) \tag{17}$$

$$\left(1-\sum_{k=1}^{|S|} z_{i,j,k}\right) T_{i,j}^{loc} + \sum_{k=1}^{|S|} z_{i,j,k} T_{i,j,k}^{off} \leq T_i^{max} \quad (18)$$
$$, \forall i \in (1,2,....D), \forall j \in (1,2,....T)$$

$$\sum_{j=1}^{T}\left(1-\sum_{k=1}^{|S|} z_{i,j,k}\right) e_{i,j,k}^{loc} \leq E_i^{rem}, \forall i \in (1,2,....D) \quad (19)$$

$$\sum_{i=1}^{D}\sum_{j=1}^{T} z_{i,j,k} r_{i,j} \leq R_k, \forall s_k \in S \quad (20)$$

$$z_{i,j,k} \in \{0,1\}, \forall t_{i,j} \in \bigcup_{i=1}^{D} MD_i, \forall s_k \in S \quad (21)$$

In the formulated problem, α is a trade-off preference parameter, allowing the designer to weigh energy consumption and monetary cost differently. Equation 16 represents the offloading decision variable $z_{i,j,k}$ need to be 0 or 1. Equation 17 guarantees that the monetary cost should be less than the available budget. Equation 18 defines the total completion time of the task can not exceed the threshold of maximum completion time. Equation 19 ensures that the local energy to run the tasks of mobile devices at d_i can not overreach the residual energy of d_i. Equation 20 ensures all the resources (i.e., CPU, memory, bandwidth, and storage, etc.) required by mobile devices $r_{i,j}$ should not exceed the available limit of resources represented by R_k. Equation 21 denotes the value of offloading decision variable $z_{i,j,k}$ should be 0 or 1. It also ensures that the task t_j of device i belongs to the set of mobile devices MD and server s_k belongs to the set of servers S.

6 Proposed algorithms

Firstly we discuss the complexity of the derived problem and present a specialized case after relaxing some constraints and then proceed to solve the general case using distributed algorithms. The proposed algorithm is an improved version of Gale and Shapley's DAA for many-to-one matching and similar to the college-admission problem [16]. At last, we discuss the complexity analysis of the proposed algorithm.

6.1 Problem complexity

To derive the complexity of a defined problem, we use the well known multiple knapsack problem [41].

Theorem 2 *MinEMC problem is NP − Hard.*

Proof Let's derive the NP-hardness of the formulated problem by assuming a special case, where $e_{i,j,k}^{off} = 0$ for $\forall i,j,k$ and $E_i^{rem} \geq \sum_{i=1}^{D}\sum_{j=1}^{A}\sum_{k=1}^{T} e_{i,j,k}^{off}$. It means if we ignore the offloading energy i.e. $e_{i,j,k}^{off} = 0$ and assuming that each mobile device exhibit ample amount of energy to run the tasks then the objective function in Equation (15) will become to

$$\sum_{i=1}^{D}\sum_{j=1}^{A}\sum_{k=1}^{T} z_{i,j,k} e_{i,j,k}^{loc} \quad (22)$$

NP-hardness of the optimization problem can be proven through a reduction from a well-known problem of multiple knapsacks. Given a set I of n items with weight w_i and profit p_i where w_i and $p_i \in (1,2,....n)$ and a set of m knapsacks with capacity c_j where $c_j \in (1,2,....m)$. Now the optimization problem is to pick T disjoint items subsets with weight w_i, such that the overall profit p_i of selected items can be maximized. Each subset of items can be allocated to a knapsack with capacity c_j, which can not be less than the total weight of selected items. Similar to the problem of multiple knapsack, an another instance can be created for the decision form of proposed optimization problem to solve in polynomial time as follows: Given a set of T of n tasks of mobile agents with required amount of resources w_i and energy to locally run the tasks p_i where w_i *and* $p_i \in (1,2,....n)$ and cloud servers set m with available resources c_j where $c_j \in (1,2,....m)$. Now the multiple knapsack optimization problem is to choose I disjoint subsets of tasks, such that the overall profit of the selected tasks is maximized. More specifically, the profit maximization problem of chosen items is equivalent to maximize the derive objective function.

The reduction is polynomial. Due to the hardness of multiple knapsack problem, hence we get the multiple knapsack problem's instances, which is analogous to another instance of maximizing the objective function. Thus the formulated optimization problem in Equation (22) is *NP*-hard. □

6.2 Optimal solution for special case

Let us assume the special case, where the resources needed to run each task in the mobile devices is equal, i.e., $res_{i,j} = r$ and the residual energy is unlimited for each mobile device; it means that each mobile device can run all of its tasks locally without energy constraints. Assuming this configuration, we present a polynomial time solution via minimum weight bipartite matching. Firstly, we build a bipartite graph $G(S_1 \cup S_2, L)$, as shown in Fig. 5. The set of vertices, S_1, S_2 of the graph and set of edges L, can be transformed as follows.

– There is a vertex $v_{i,j}$ corresponding to each task $t_{i,j}$ in S_1. That is each task has a vertex in S_1, i.e., $S_1 = \{ v_{i,j} | \forall t_{i,j} \in \cup_{i=1}^{n} MD_i \}$

- There is a vertex $v'_{i,j}$ corresponding to each task $t_{i,j}$ in S_2 also. For each server k, we add |R/r| vertices in S_2 and denote them as $v''_{k,k'}$. That is S_2 comprises of vertices corresponding to each task and |R/r| vertices for each server, i.e., $S_2 = \{ v'_{i,j} | \forall t_{i,j} \in \cup_{i=1}^{n} MD_i \} \bigcup \{ v''_{i,j} | \forall s_k \in S, 1 \leq k \leq |R/r| \}$
- Considering any two vertices $v_{i,j} \in S_1$ and $v'_{i,j} \in S_2, \forall i,j$, a link $(v_{i,j}, v'_{i,j})$ can be added to L and give a weight of $w_{i,j} = e^{local}_{i,j}$ to it.
- For any two vertices $v_{i,j} \in S_1$ and $v''_{k,k'} \in S_2, \forall i,j,k,k'$, a link $(v_{i,j}, v''_{k,k'})$ can be added to L and give a weight of $w_{i,j} = e^{off}_{i,j,k} + \alpha MC_{i,j,k}$ to it, i.e. L={ $(v_{i,j}, v'_{i,j}) | \forall v_{i,j} \in S_1 and \forall v'_{i,j} \in S_2 \} \bigcup \{ (v_{i,j}, v''_{k,k'}) | \forall i,j,k \}$.

Theorem 3 *The MinEMC problem with the same resource requirement for all the tasks and unbounded energy at the mobile device could be transformed towards obtaining a minimum-weighted bipartite matching in graph $G(S_1, S_2, L)$.*

Proof We show that any matching in a given graph G is a feasible solution for proposed problem, i.e.

- if link $(v_{i,j}, v'_{i,j})$ is included in matching, then $\sum_k = 1^{|S|} x_{i,j,k} = 0$ in our solution.
- if link $(v_{i,j}, v''_{k,k'})$ is included in matching, then task $t_{i,j}$ can be offloaded to run on server s_k.

The constraints 19 and 20 and are satisfied as we have $\sum_{i=1}^{n} \sum_{j=1}^{T} x_{i,j,k} \leq |R/r|$ an unlimited energy as our relaxations. We now show that a feasible solution $\{z_{i,j,k}\}$ can be transformed into a weighted matching in graph G as follows: From constraint 16, we can say that for vertex $v_{i,j}$, the total matchings can only be one, i.e. either the link $(v_{i,j}, v'_{i,j})$ if $\sum_{k=1}^{|S|} x_{i,j,k} = 0$ or $(v_{i,j}, v''_{k,k'})$ if $x_{i,j,k} = 1$. Which means at most one incoming link is chosen in matching. So, feasible solution of proposed problem is transformed into a feasible matching.

We can prove that the weight acquired in a minimum weighted matching bipartite matching problem is equivalent to an optimal outcome. From Eq. 15 we get,

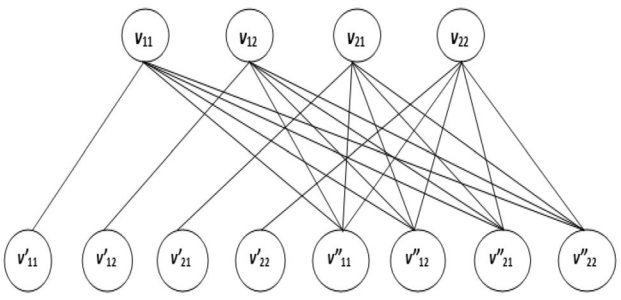

Fig. 5 Bipartite graph modelling for computational tasks offloading

$$\begin{aligned} &\sum_{i=1}^{D} \sum_{j=1}^{T} \left(1 - \sum_{k=1}^{|S|} z_{i,j,k}\right) e^{loc}_{i,j} + \sum_{i=1}^{D} \sum_{j=1}^{T} \sum_{k=1}^{|S|} z_{i,j,k} e^{off}_{i,j,k} \\ &\quad + \alpha \left(\sum_{i=1}^{D} \sum_{j=1}^{T} \sum_{k=1}^{|S|} z_{i,j,k} MC_{i,j,k} \right) \\ &\quad = \sum_{i=1}^{D} \sum_{j=1}^{T} \sum_{k=1}^{|S|} z_{i,j,k} (e^{off}_{i,j,k} + \alpha MC_{i,j,k}) \\ &\quad + \sum_{i=1}^{D} \sum_{j=1}^{T} \left(1 - \sum_{k=1}^{|S|} z_{i,j,k}\right) e^{loc}_{i,j}. \end{aligned} \tag{23}$$

From our weights assignment, it can be observed that the sum of weights with the minimum weighted matching problem in a bipartite graph is equivalent to the optimal solution of the proposed problem.

From this, it is concluded that the proposed special case could be solved using polynomial time-optimal solution.

6.3 Designing algorithm for general case

Distributed algorithm for computation offloading: For the first phase, we have designed the distributed algorithm for computation offloading inspired by [28]. In the distributed version, each node can asynchronously execute its computation. The proposed algorithm work in two procedures for a pair of the mobile device and on the agent lies in the coverage area of the mobile client i.e. *MobileDevice*() and *Agent*() as discussed in Algorithm 1. The algorithm works in following phases:

The MobileDevice() procedure: This procedure is performed for every mobile device.

1. Firstly, in line $5-7$ it initializes the EnergyList and updates the information by available local energy e^{local}_t for each task of mobile device *MD*. The CostList is used to rank the costs for each task t at *MD* based on their monetary cost. After that it sums the respective ranks and creates a OffloadList of tasks at each device.
2. It iteratively applies the constraints till the *offloadList* becomes empty. From line $10-11$ it checks the offloading energy, local energy, and required bandwidth for task t. If it is satisfied, then it sends an offload message for task t and wait for reply.
3. From line $12-20$, if the reply is *accepted* then it updates E^{rem}_i and remove the task from the *OffloadList*. If the reply is *rejected* then it runs the task locally on the device and updates E^{rem}_i and removes the task from the *OffloadList* else it doesn't run the task.
4. If the *OffloadList* becomes empty then a stop message is received.

The Agent() procedure: This procedure is performed for every agent lies in the coverage area of mobile device *MD*

1. Wait for messages from mobile device *MD*
2. In line $28-32$, it accepts the message received from *MD* and check the monetary cost, completion time constraints and required resource constraints and if it satisfied then send the "accepted message" else it sends the "rejected message" to *MD*
3. After completing the procedure, a stop message is received.

Algorithm 1: Distributed algorithm for computational offloading

```
1  Procedure MobileDevice():
2  Input: Tasks of this mobile device which are ready to offload
3  Output: Offloading decision for each of the task in MD
4  begin
5      Create an EnergyList in which we rank the tasks in the
         MD by sorting in decreasing order of e_t^local where t is
         tasks in MD.
6      Create a CostList in which we rank the tasks in MD by
         sorting in increasing order of MC_t where t is tasks in
         MD.
7      Create a OffloadList of tasks where sorting in increasing
         order is done based on sum of respective ranks in
         EnergyList and CostList.
8      while OffloadList ≠ ∅ do
9          t ← first element OffloadList
10         if e_t^off < e_t^local or e_t^local > E_i^rem and
             r_t^B ≤ r_avail^B then
11             sendMsg(offload,t)
12         msg ← getMsg()
13         switch msg.type do
14             accepted: E_i^rem = E_i^rem - e_t^off
15             rejected: if e_t^local ≤ E_i^rem then
16                 Run the task locally on the device.
17                 E_i^rem = E_i^rem - e_t^local
18             else
19                 We don't run this task.
20         remove task t from OffloadList
21 end
22 Procedure Agent():
23 Input: Set of resources available from servers.
24 Output: Offloading decision for each of the task in MD
25 begin
26     stop ← false
27     while ¬ stop do
28         msg ← getMsg()
29         switch msg.type do
30             offload: t ← msg.task
31             if MC_t ≤ MC_t^Budget and T_t^off ≤ T_t^max
                 and r_t ≤ r^avail then
32                 sendMsg(accepted)
33             else
34                 sendMsg(rejected)
35             stop: stop ← true
36 end
```

Distributed algorithm for stable matching Our objective is to create the preference sets and to generate a stable matching between tasks of different sizes as VMs and cloud servers. We do that with the help of the policies at both the server and the mobile end. As the model under consideration uses a multi-tenant, we get tasks that are sharing resources of the cloud instance. So, we have to ensure that isolation is achieved. This can be done with the help of having defined policies that give preference lists for each of the sets. Let us identify our two sets here which are used in matching, the solution for which will ensure that both energy and monetary cost are minimized at the mobile end.

Set of servers As defined in the model, we have k servers labeled from 1 to k. Here we add a special server s_0, which helps us in matching. This s_0 has no policies as to prefer one task over other, i.e., all tasks are preferred equally, and this server has a quota as infinity, which means this special server accepts all the tasks proposing it. This special server s_0 is defined so that all the tasks which are matched to this server are executed locally. All the other servers will have their policies which they follow to give their preference for the tasks along with their quota limitation.

Set of tasks As defined in the model, we have a set of tasks from each of the mobile devices labeled as $t_{i,j}$. These tasks will have to ensure that their policy would try to decrease overall energy consumption while reducing total monetary cost as well.

The preferences sets for task and servers can be defined as follows:

(i) Servers preference list function The multi-tenant cloud provider, generally aims to consolidate the mobile client workload onto a minimum number of hugely occupied servers so that the idle servers can be switch-off to minimize the operational cost and maximize the revenue. Each server can accommodate multiple VMs based on quota q_{max} of the maximum number of VMs. The servers create their preferences based on the function called $P_S(s)$ based on the policies they employed, where s denotes the server. The preference set for servers with different policies can be defined as:

policy 0 Server 0 employs a policy as all tasks are equal irrespective of the incentive or the size of tasks, which means s_0 prefers all the tasks equally.

policy 1 Some server between 1 to k might choose to follow this policy which is revenue-maximizing, which means they choose those tasks which give maximum incentives for them monetarily.

$$P_S(s) = \xi(Incentive) \tag{24}$$

where *Incentive* is a monetary benefit for providing the services.

policy 2 Some server between 1 to k might choose to follow this policy which is to choose maximum size tasks first, which means they choose those tasks which have big sizes. The reason for this might be if the task sizes are large, then they might execute for a long time and earn them more incentives while decreasing the maintenance costs.

$$P_S(s) = \xi(tasksize) \tag{25}$$

where *tasksize* is the size of the tasks under consideration.

This way, there can be a large variety of policies depending on CPU, RAM, memory, which the servers can employ depending on their situation to have maximum incentives to serve the multi-tenant model.

The server always prefers to match with the tasks providing higher $P_S(s)$.

(ii) Tasks preference list function From the perspective of mobile clients and resource demand of tasks. The tasks create their preferences based on the function called $P_T(t)$, where t denotes the task. Each task can be assigned to one server. The tasks have one policy, which is to minimize their monetary cost and total energy.

Mathematically the matching function for tasks can be defined as follows: **For server** s_0 :

$$P_T(t) = e_{i,j}^{loc} \tag{26}$$

For servers 1 to k:

$$P_T(t) = e_{i,j,k}^{off} + \alpha MC_{i,j,k} \tag{27}$$

After this each task will have a preference list $P_T(t)$ which sorts all the servers from 0 to k in ascending order of preference.

The tasks always prefer to match with the server providing higher $P_T(t)$.

Now we discuss our proposed algorithm, as shown in Algorithm 2, which is inspired from the *Gale* and *Shapley's* DAA [25] for many-to-one stable matching.

Algorithm 2: Distributed algorithm for stable matching

Input:
For each task: the set of available servers.
For each Server : the set of available tasks.
Output:
A stable matching between tasks and servers.
begin
  Phase 1:*Initialization:*
  begin
    Phase 1.i: All servers and tasks exchange information and all are marked unengaged. $W(s) = D(s) = \emptyset \ \forall s$.
    Phase 1.ii: Every task computes its preferences using the function $P_T(t)$
    Phase 1.iii: Every server computes its preferences using the function $P_S(s)$
  end
  Phase 2:*Matching:*
  begin
    while *there are unengaged tasks that can propose* **do**
      Phase 2.i: Proposals: Every unengaged task t proposes to its most preferred and not yet proposed server from its list of preferences. If this server is already engaged to some tasks, all those tasks and this server are marked unengaged.
      Phase 2.ii: Update waiting list: Every server s updates its waiting list of proposals, $W(s) \leftarrow W(s) \cup proposers$ and dynamic list of proposals, $D(s) \leftarrow W(s)$.
      Phase 2.iii: Counter-proposals: Every server s find the set of most preferred tasks from $D(s)$ using its preference list $P_S(s)$ and its quota q_{max}. Each server counter-proposes the set of tasks which they computed.
      Phase 2.iv: Accept or reject: Upon receiving the counter-proposals, the tasks accept or reject it by checking if there is a better preferred, not yet proposed server.
      Phase 2.v: If all tasks of the computed set accept the counter-proposal received from a server(s), then all the tasks and servers who were engaged to this set of tasks and this server(s) are marked unengaged.
      Now the accepted set of tasks and that server are engaged.
      Phase 2.vi: All unengaged servers update its dynamic list $D(s)$ by deleting those that are engaged to another server and those that rejected the counter-proposal.
      $D(s) \leftarrow D(s) \setminus \{\text{engaged rejectors}\}$
      Phase 2.vii: Go to phase $2.iii$ if $D(s)$ of atleast one server is strictly decreased.
    Phase 2.viii: All engaged servers and tasks are matched.
  end
end

We execute this algorithm on each agent. It works as follows: In (Phase 1.i) All servers and tasks exchanged information and marked as unengaged. In *Phase*1.*ii* The task computes their preference lists by using function $P_T(t)$. In *Phase*1.*iii* Every server computes its preferences using function $P_S(s)$. The matching algorithm then begins with rounds in the course of which tasks send proposals, servers reply with counter-proposals, and tasks either reject or accept the proposal (Phase 2.i to Phase 2.viii). Each server that collects a new proposal can reassess its chances and consequently marked unengaged (Phase 2.i). $W(s)$ contains the list of those tasks that have proposed at least once to server s. There is a dynamic list denoted as $D(s)$ again initialized to $W(s)$ prior to receiving counter-proposal from any server (Phase 2.ii). For each round, every unengaged task further proposes to its highest favorable server for which it hasn't proposed yet(Phase 2.i). Each server collecting proposals includes the players to its progressive proposer's list and again initializes its dynamic list (Phase 2.ii). With the help of the dynamic list, it explores for its most favorable one, including only tasks and releases a counter-proposal to such tasks.(Phase 2.iii). Each task matches the received counter-proposals with its preference list obtained with the servers; it has not proposed yet (Phase 2.iv). If one of the servers is more preferred than the most desirable counter-proposal, subsequently, the task rejects the proposals while carrying on with proposing (Phase 2.iv, Phase 2.v). Else, the task accepts its most favorable counter-proposal (Phase 2.iv). For particular counter-proposal, if each of the tasks accepts it, then they become engaged with the server. From the whole computed set in which the set of tasks and cloud servers were previously engaged are shattered, and all of their corresponding players marked as unengaged (Phase 2.v). If at least one task doesn't permit, subsequently the server is set to be unengaged (Phase 2.v), Its dynamic list is upgraded via eliminating tasks found rejected its counter-proposal also currently engaged with some other server (Phase 2.vi). The counter-proposals continue to execute until no more server can issue any new counter-proposal (Phase 2.vii). The ongoing round stops and the algorithm get into a new round (Phase 2.viii). The algorithm terminates when no additional tasks can be rejected. Hence the outcome is stable matching.

6.4 Algorithm analysis

This section discusses a brief complexity analysis for the proposed algorithms.

Theorem 4 *The total run time complexity of the offloading algorithm* **Algorithm**1 *is* $O(TlnT)$.

Proof We give different tasks in all the mobile devices as input to the algorithm and get the decision for each of the tasks as to offload, run locally, or get rejected by the algorithm. The total run time complexity of the offloading algorithm is $O(TlnT)$, where T indicates the total number of tasks, which is calculated as the sum of the number of tasks in each of the mobile devices (MDs). The number of tasks in one MD is calculated as the sum of tasks in each of its applications.

Theorem 5 *On the basis of proposals received from players,the complexity of* **Algorithm**2 *is* $O(\lambda^5)$, *where* $\lambda = \max(T, S)$.

Proof Let's begin with an upper bound on the proposals generated through the tasks of mobile devices, then an upper bound for cloud servers are also considered. In at most T proposals, each task has proposed to each of the cloud servers. Hence, in at most $T \times S$ proposals, the tasks have proposed to all cloud servers. For no more than S counter-proposals, each cloud server has proposed to all of the tasks. Moreover, each server counter proposes in each round. Therefore, in at-most $T \times S \times T$, the cloud servers released all of their counter-proposals. Hence, we can derive that the proposals should not exceed $T^3 \times S^2$. The overall complexity of proposed algorithm is $O(\lambda^5)$, where $\lambda = \max(T, S)$.

We are using counter proposals to eliminate the problem of complementaries. Now, once we obtain the solution to our stable matching, we execute all the tasks which are matched to s_0 locally and offload all others to their own cloud.

Theorem 6 **Algorithm**2 *converges i.e., give a matching outcome within a finite number of iterations.*

Proof Following the initialization phase, we enter into a matching phase for all the unengaged tasks. The matching phase is composed of two different loops. The first one represents proposals from tasks. For each iteration of this exist an outer loop, there is a counter-proposal from the servers to the set of tasks. Both loops terminate after executing a finite number of iterations. During the phase of counter-proposals, the following cases can arise:

Case 1 If an engaged server is still engaged, then its dynamic list $D(S)$ will remain unchanged. (As mentioned in Phase 2.vi, only the list of all unengaged servers is updated.)

Case 2 If an unengaged server has become engaged, then its dynamic list will remain unchanged. (As mentioned in Phase 2.vi, only the list of all unengaged servers are updated)

Case 3 If an unengaged server is still unengaged. This case may arise when a few of the tasks it counter proposed

during Phase 2.iv and rejected its proposal, and either (i) No task is engaged with another server or (ii) only a few tasks are engaged. In (i) the list will be left unmodified only while in (ii) it decreases.

Case 4 If an engaged server becomes unengaged. This may be only possible if all tasks of the computed set accept the counter-proposal of a server(s), then all the tasks and servers who were engaged previously are marked unengaged. Hence the dynamic list $D(S)$ of the server will be decreasing. Thus for all the above cases, the inner loop of counter proposals will converge into finite steps.

Let's consider the outer loop. Here the convergence due to the finite number of cloud servers each task can propose to and also another certainty that no task can propose more than once to any of the servers. Hence the proposed algorithm will definitely converge into a finite number of iterations. □

7 Simulation setup and experimental results

First, we illustrate the simulation setup environment and performance metrics. Then, we analyze the experimental results to show the efficacy of the Off-Mat method. In the experiments, we have compared the proposed algorithm with some other methods. Finally, we discuss how our proposed solution minimizes energy consumption, delay, and monetary cost through the 'Off-Mat' framework for multi-tenant mobile clients.

7.1 Simulation setup

In this study, all of the algorithms are executed on a local terminal having an Intel Core i7 processor with 3.4GHz and 8GB RAM using Java 14.0.2. Workload parameters and simulation settings are summarized in Table 4. Similar to the simulation settings of [40], we have used the real-world parameters following the random uniform distribution. We have randomly generated mobile devices (n) between 50 to 100 and the number of applications (M) per device between 1 to 10. The total number of computational tasks T is generated between 50 to 800. The data size of tasks (B^{send}) lies in the interval 10 kilobytes (KB) to 1 megabytes (MB), and the computation (C^{local}) of executing each task distributed in the range of 200 to 2000 megacycles. Similarly, the mobile device CPU frequency (S^{local}) is generated between 1 to 1.5 GHz at random, and the result data size (B^{rec}) is set to 1 to 10 KB. We assume the data receiving power consumption rate (P^{rec}) is between 257 to 325 MW, and data transmitting power consumption rate (P^{send}) is set to 257 to 325 MW. The data network charge rate per MB(γ) is set to $ 0.02 to 0.03 per MB. To simulate a cloud data center, we have configured the hosts between 10 to 100. The characteristics of these servers are listed in Table 5 [67]. Corresponding to Amazon EC2, the four types of VM instances are used, and their characteristics are described in Table 6 [67]. The CPU frequency of Cloud VM (S^{cloud}) is 3.4 GHz, and the charge rate of Cloud VM (β) is set to $ 0.84 per unit time. The active CPU power consumption rate (P_{local}) is between 644 to 700 MW and the idle CPU power consumption rate (P_{idle}) is set to 5 to 10 MW. The total number of agents is set between 2 to 10. For modeling the agents, we set the available bandwidth (r_{send}) and (r_{rec}) between mobile devices and agents between 100 to 800kbps. The maximum time limit (T_{max}) is set from 1.0 to 2.0, and the total bandwidth ranges between 10 to 20 Mbps. The total budget (MC_{budget}) is between $100 to 3000. To characterize the task offloading behaviour, we have adopted the ratio of load-input data (LDR) [40], where the LDR = $\frac{B^{send}}{C^{local}}$. Thus if the LDR value is high, then the task is compute-intensive and preferred for remote execution in the cloud; otherwise, the task is communication-intensive and suitable for local execution.

7.2 Performance metrics

The following performance metrics are applied to assess the efficiency of the proposed 'Off-Mat' approach.

7.2.1 Request filtering

The goal of request filtering metric is to minimize the offloading requests that cannot meet budget and deadline constraints so that the offloading decision-making latency can be improved. We have also analyzed the influence of different LDRs on the filtering of requests.

7.2.2 Energy consumption

Energy consumption measures the amount of energy used for serving the requests. To analyze the total energy consumption, we have analyzed the local energy consumption and offloading energy consumption.

7.2.3 Request delay

To measure the delay of request-response, we have used the Ping tool to check the request transmission delay between the mobile device and agent. It measures the time of ICMP-request packets when sent from mobile device to any agent or cloud server and then receiving the packets sent back from the agent or cloud server.

Table 4 Simulation setting

Parameter	Value
Number of applications per device	1 to 10
Number of tasks (T)	50 to 800
Total mobile devices (D)	50 to 100
Number of cloud servers (S)	10 to 100
Number of agents	2 to 10
Number of VM instances	4
Task data size (B^{send})	10 KB to 1 MB
Task computation (C^{local})	200 to 2000 mega cycles
Mobile device CPU frequency (S^{local})	1 to 1.5 GHz at random
Task result data size (B^{rec})	1 to 10 KB
Data receiving power consumption rate (P^{rec})	257 to 325 MW
Data transmitting power consumption rate (P^{send})	257 to 325 MW
CPU frequency of Cloud VM (S^{cloud})	3.4 GHz
Charge rate of Cloud VM (β)	\$0.84 per unit time
Active CPU power consumption rate (P_{local})	644 to 700 MW
Idle CPU power consumption rate (P_{idle})	5 to 10 MW
Data transmission rate (r_{send})	100 kbps to 800 kbps
Data receiving rate (r_{rec})	100 kbps to 800 kbps
Total bandwidth	10 to 20 Mbps
Data network charge rate per MB(γ)	\$0.02 to 0.03 per MB
Maximum time limit (T_{max})	1.0 to 2.0
Budget (MC_{budget})	\$100 to 3000

Table 5 Configuration of hosts

Type of host	Type of CPU	No. of cores	Frequency (MHz)	RAM (GB)
HP ProLiant G4	Intel Xeon 3040	2	1860	4
HP ProLiant G5	Intel Xeon 3075	2	2660	4

Table 6 Configuration of VM instances

Type of VM	CPU (MIPS)	RAM (MB)
Micro	500	613
Small	1000	1740
Extra large	2000	1740
High-CPU medium	2500	870

7.2.4 Monetary cost

The monetary cost denotes the sum of data transferring cost and pubic cloud services cost. Each device has an offloading budget for mobile cloud services set by mobile clients. We have analyzed the monetary cost by varying the budget of mobile clients and evaluating the total savings by varying the offloading requests.

7.2.5 Fitness cost

We have used the weighted-sum-method (WSM) to find the fitness function for Eq. (15). It applies an aggregation function to transform a multi-objective function into one scalar objective function. Using WSM, we have reformulated Eq. (15) as follows:

$$\min w_1 \left(\sum_{i=1}^{D} \sum_{j=1}^{T} \left(1 - \sum_{k=1}^{|S|} z_{i,j,k} \right) e_{i,j}^{loc} + \sum_{i=1}^{D} \sum_{j=1}^{T} \sum_{k=1}^{|S|} z_{i,j,k} e_{i,j,k}^{off} \right) + w_2 \left(\sum_{i=1}^{D} \sum_{j=1}^{T} \sum_{k=1}^{|S|} z_{i,j,k} MC_{i,j,k} \right) \tag{28}$$

where w_1 and w_2 indicates the weights of the energy consumption and monetary cost objectives, respectively, the sum of both weight parameters is equal to 1. The objective aims to reduce the fitness cost.

7.2.6 Throughput

Throughput indicates the total number of computational tasks that receive their service in per unit time. We have used average throughput time to analyze the performance of different approaches. Throughput time depends on different parameters, such as network delays, processing power, etc. If the optimization rate of the algorithm is higher, then the throughput is faster.

7.2.7 Happiness performance

The happiness metric measures the advantage in resolving the conflicts between the mobile devices tasks and cloud servers using stable matching. We utilize the rank percentile of the selected partner, i.e., tasks or servers, to measure the "happiness" of matching. For cloud servers, happiness indicates the average rank obtained through the matched number of tasks. By varying the total number of tasks and servers, we evaluate happiness performance.

7.3 Baseline approaches

We have measured the performance of the Off-Mat algorithm with the following two baseline algorithms:

- Traditional offloading: In the traditional offloading framework, the mobile devices directly send the tasks to a remote cloud. The remote cloud makes an offloading decision and returns the decision results to the mobile device. There are no agents in the offloading framework.
- Agent-based offloading: It uses agents, where the device sends its offloading request to the agent for performing the offloading decision rather than the distant cloud.

7.4 Experimental results

7.4.1 Impact of request filtering

In this experiment, we have evaluated the performance of the Off-Mat algorithm with respect to the filtering of offloading requests. The primary task of request filtering is to reject the computational offloading requests that cannot satisfy the budget and deadline constraints so that the overall delay of decision making can be minimized. In Fig. 6, we have shown the impact of request filtering by varying the offloading requests. We have considered the requests of different mobile users from 100 to 800, which is equivalent to cases, i.e., case 1 to case 8, respectively. From Fig. 6, it can be identified that the filtering process performs better when the LDR value is low (LDR=1.0) as most of the offloading tasks are communication-intensive and suitable for local execution due to the completion time constraint. Thus, in each scenario, it can be observed that when the LDR value is high (LDR=1.5), then the task is more likely to be offloaded due to its computation-intensive nature.

7.4.2 Performance on energy consumption

In this experiment, we evaluate the energy consumption of the three computation offloading approaches. For this study, we consider that the available bandwidth is sufficient, and all of the offloading tasks can be offloaded directly to the cloud. From Fig. 7, it can be observed that energy consumption rises with the increase of offloaded tasks. In comparison to traditional and agent-based offloading, the proposed approach outperforms and gives better results. As all of the computational tasks can not get advantage via remote execution because of lower LDR value. While proposed 'Off-Mat' approach schedules the computational tasks on the agents so that the mobile devices consume the least amount of energy.

Further, we set the number of offloading requests to 800 and task size to 10KB. From Fig. 8, it can be identified that the total energy consumption of the traditional offloading scheme is approximately three times more than the agent-based offloading. The reason behind this is the longer RTT (round-trip-time) in traditional offloading. Thus it results in more energy consumption than an agent-based scheme. Our proposed approach gives better performance as the device and agent check the constraints and only offload the valid requests to the cloud. Hence it considers only the beneficial offloading tasks for remote execution and saves more energy.

7.4.3 Impact of request delay

Figure 9 shows the average request-response delay of the proposed framework and compares its performance with the traditional and agent-based offloading frameworks. In this comparison, we send the ICMP-request packets through the mobile devices to the agent or cloud and receive back the computation results. For delay analysis, we set the number of offloading requests to 300 and task size to 10KB. In Fig. 9, it can be identified that the average request delay of the proposed approach and agent-based offloading approach is much shorter than the traditional offloading approach. Specifically, the average request delay for the agent is less than 10 ms, and the request delay of the traditional offloading scheme is nearly 30 ms. In contrast, the Off-Mat approach takes less than 5ms in comparison to the other two frameworks. The reason behind the better performance is that the agent is located

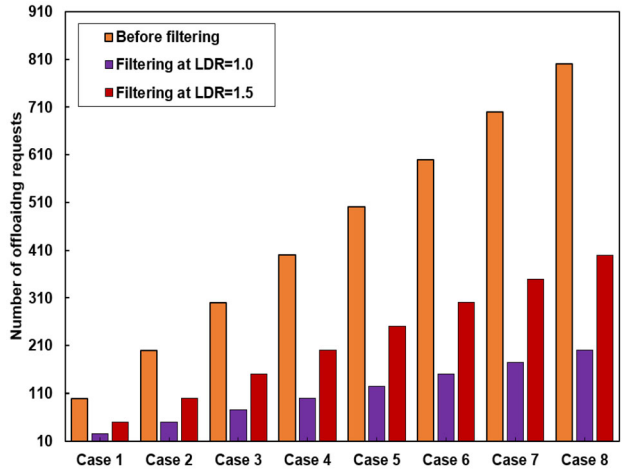

Fig. 6 Effect of LDR on request filtering

one-hop near to the mobile devices. Thus the average latency is much shorter than the cloud where the latency increases due to the complex networks. Further, the Off-Mat approach makes better decision making as it only sends the valid offloading requests to the agent.

7.4.4 Performance on monetary cost

To analyze the monetary cost, we have varied the user budget from 100$ to 3000$ and offloading tasks from 100 to 900. As depcited in Fig. 10, when the budget of mobile devices increases for Off-Mat algorithms, the total cost of offloading tasks will increase accordingly as the more significant number of computation-intensive tasks are offloaded to the servers.

In Fig. 11, we have varied the user budget and set the request size to 500. It can be identified that the total cost of traditional offloading is nearly four times higher than the Off-Mat approach. The reason behind the lower monetary cost is the less number of offloaded tasks due to beneficial offloading. In Fig. 12, we repeat the same experiment and set the request size to 800. Finally, Figs. 13 and 14 demonstrate the total cost saving for the request size 500 and 800, respectively. From the figures, it can be identified that the Off-Mat algorithm can save more cost than the traditional and agent-based offloading algorithms.

7.4.5 Fitness cost performance

The fitness cost performance depends exclusively on the preferences of weight parameters. Thus, after performing some initial experiments, we found that the best performance was obtained by assigning equal weights to each objective. Figure 15 shows the analysis of average fitness value for the different number of offloading tasks. It can be identified that the Off-Mat scheme outweighs all baseline algorithms. The reason for better performance is the optimal number of offloaded tasks, which also reduces the monetary cost in terms of data transfer cost and public cloud cost. After the off-Mat, the agent-based approach shows better performance for the fitness value.

7.4.6 Throughput performance

In Fig. 16, we analyze the average throughput time by increasing the offloading tasks from 100 to 800.

It can be noticed that the Off-Mat scheme has demonstrated maximum throughput time along with varying number of offloaded tasks. Due to the earliest response time of Off-Mat, it generates faster throughput, whereas the response time of traditional and agent-based offloading is higher, which results in low throughput (higher values).

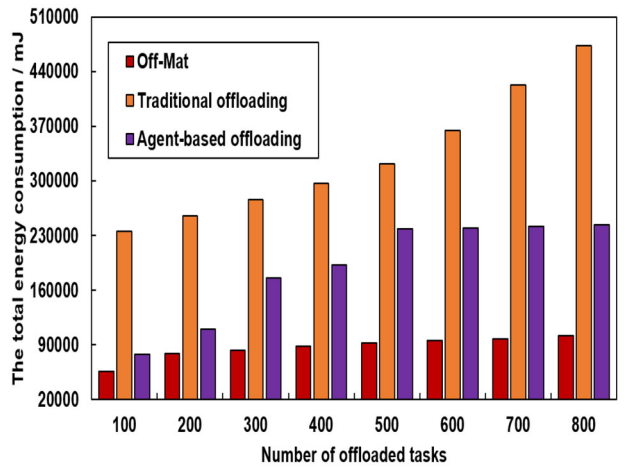

Fig. 7 Total energy consumption analysis of offloading approaches

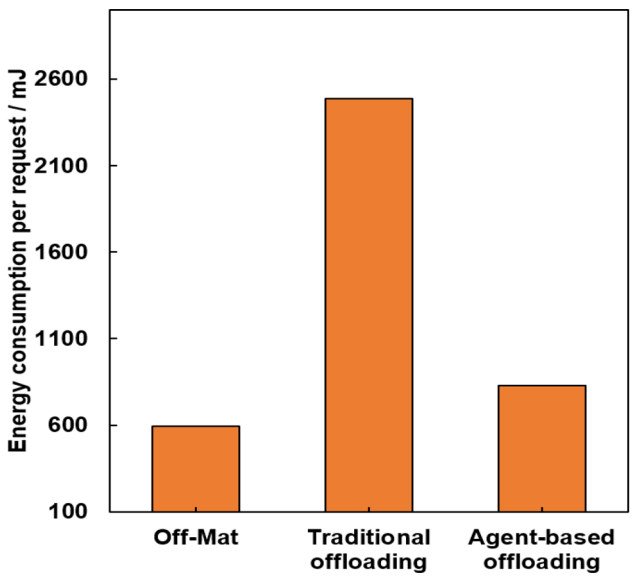

Fig. 8 Total energy consumption analysis for request size 800

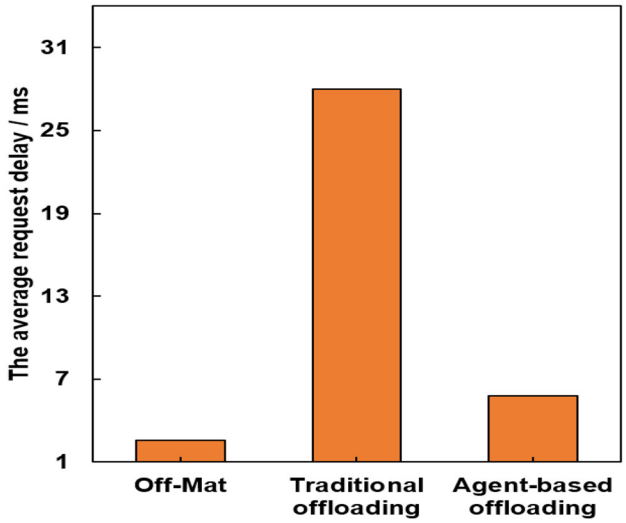

Fig. 9 The average request delay

Throughput time depends on various parameters, such as delays in network, processing power, and hardware resources. It is observed that while varying the number of computational tasks, the net throughput of Off-Mat is getting better and producing stable behavior over other approaches.

7.4.7 Happiness performance

Figures 17, and 18 depict the happiness percentages of tasks and servers, respectively. For this experiment, we consider 10 cloud servers and increase the offloading tasks ranging from 50 to 300. Cloud servers are initially empty, and each can accommodate 10 VMs of different sizes. For computational tasks, we apply different allocation policies, and for servers, we perform a consolidation policy. To measure the matching happiness, we use the average rank of the matches, tasks, and servers.

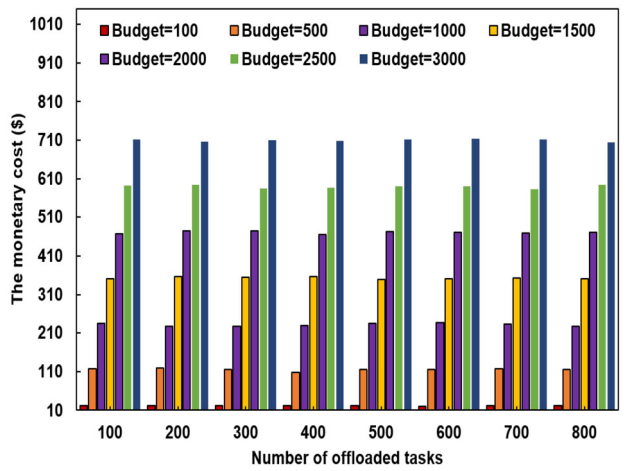

Fig. 10 Monetary cost with varying budget

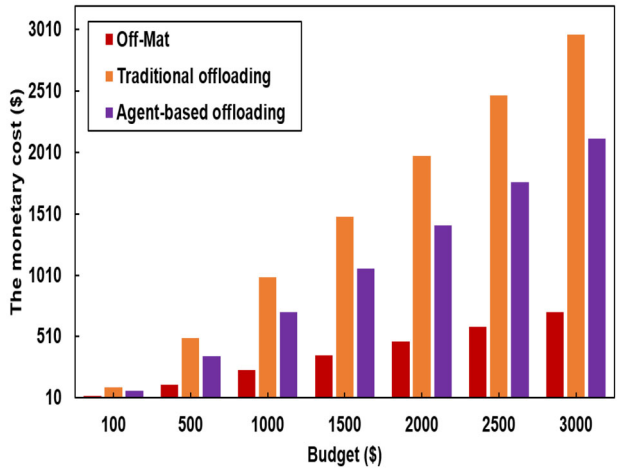

Fig. 11 Monetary cost for request size 500

Compared to the First-Fit benchmark algorithm, the proposed distributed stable matching algorithm provides a substantial improvement in servers' performance. It demonstrates the advantage in resolving the conflicts between the mobile devices tasks and cloud servers using stable matching. First-Fit only allows an individual uniform ranking of tasks for all listed servers; on the other hand, the proposed distributed stable matching approach permits cloud servers to reveal their preferences. Additionally, First-Fit is not able to match a task to a server with inadequate capacity. It means no further rejection from servers, while the proposed algorithm grants rejections if a task is more preferable than other server's tasks during its entire execution. Distinctly, this enhances the task's and server's happiness. Through the analysis of results, we find that the computational tasks can obtain top 10% companion on average while cloud servers are only able to acquire their top 50% tasks. As the number of available VMs is too small in comparison with total capacity of cloud servers,

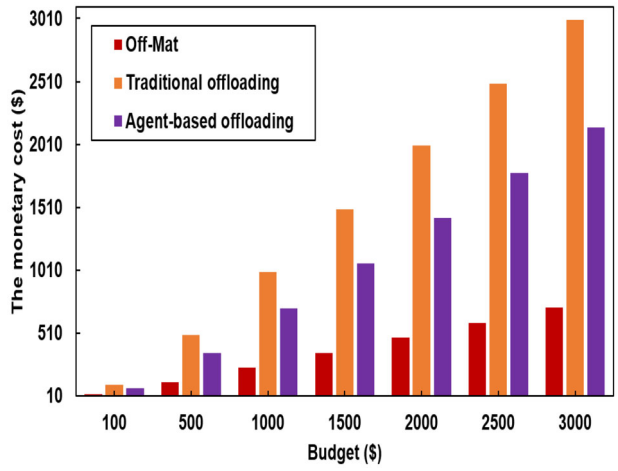

Fig. 12 Monetary cost for request size 800

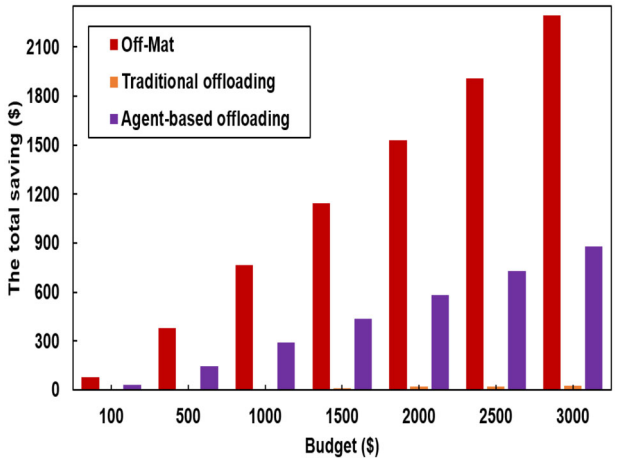

Fig. 13 Total saving when request size is 500

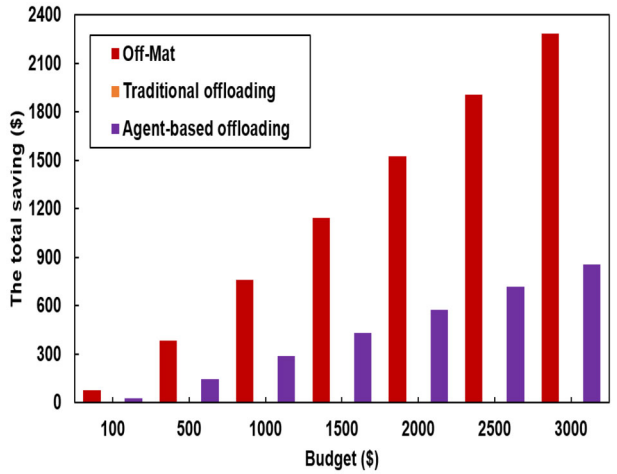

Fig. 14 Total saving when request size is 800

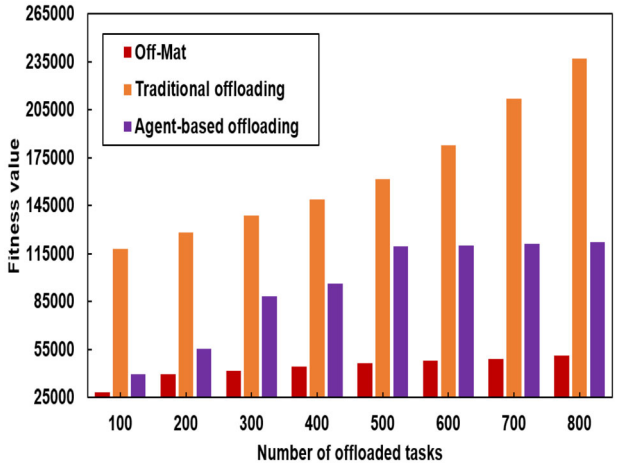

Fig. 15 Average fitness cost

and most of the proposals from VMs' can be directly approved by cloud servers.

For large-scale simulations, we vary the number of offloading tasks and servers and analyze overall happiness performance. As shown in Fig. 19, the tasks get their top 6–8% preferences, while the cloud servers obtain their 13–18% preferences. Thus, we can observe that the Off-Mat effectively analyzes the policies and able to resolve the conflicts for large-scale scenarios.

7.5 Statistical analysis

To evaluate the reliability of the 'Off-Mat' approach, we have applied statistical evaluations. The statistical analysis is conducted to investigate whether the experimental results are statistically significant, and not by coincidence [52, 53].

For different applications, we have several types of statistical tests suitable for heterogeneous data such as homoscedasticity and normality. Thus, to perform the statistical test analysis, we use the StatService toolkit [68, 69], which offers a smart model to select the best statistical test according to the features of data. Based on the evaluation, the suggested statistical test for data analysis is the T-test. Thus, we conduct a paired T-test of the Off-Mat approach and other baseline approaches using statistics calculators available at social science statistics [70]. A T-test is conducted by constructing the following hypotheses:

- **Hypothesis 1:** No difference is observed between Off-Mat and baseline algorithms.
- **Hypothesis 2:** Significant difference is observed between Off-Mat and baseline algorithms.

The primary purpose of this statistical analysis is to validate that the obtained experimental results are statistically significant and not by chance [52, 53]. We evaluate all

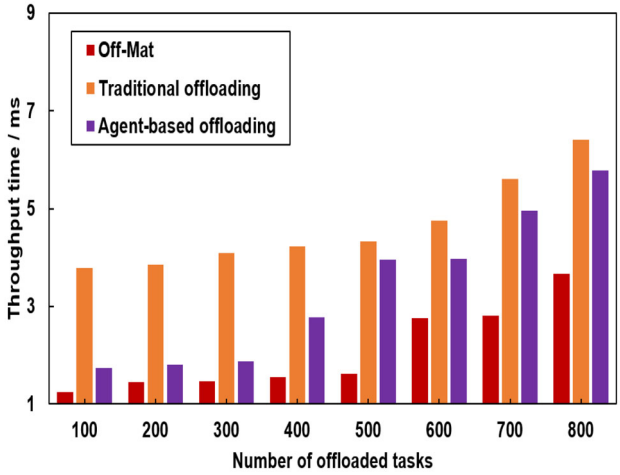

Fig. 16 Throughput time for different approaches

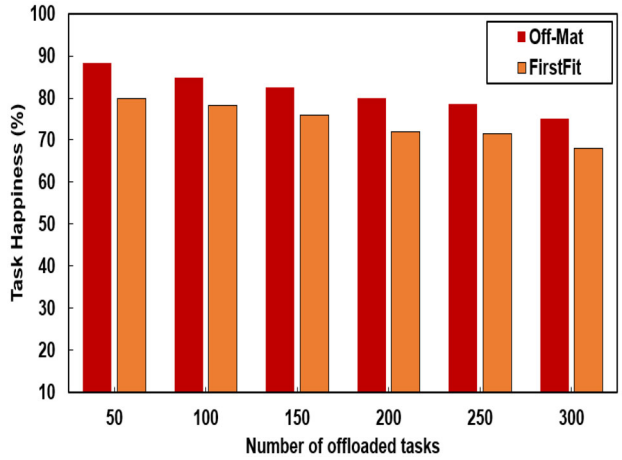

Fig. 17 Task happiness

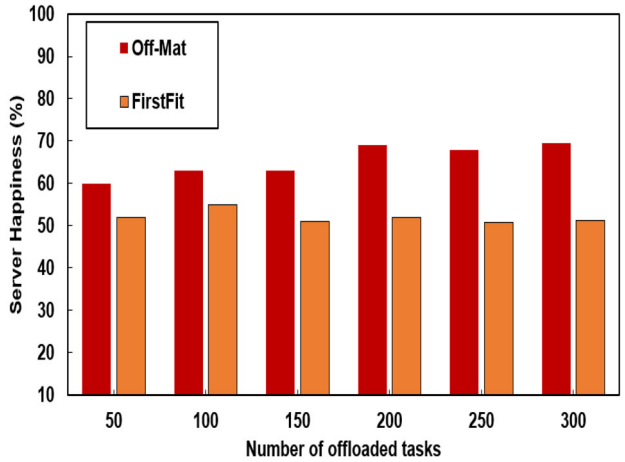

Fig. 18 Server happiness

performance parameters using four standard statistical tests, including total samples, mean value, standard deviation (SD), t-value, p-value, and degrees of freedom (df).

Table 7 represents the statistical analysis of the paired t-test. It shows the significance level of Off-Mat compared with the other benchmark strategies concerning request filtering, energy consumption, total delay, monetary cost, total cost saving, throughput time, task happiness, and server happiness. The objective of the t-test is to validate the correctness of the stated hypothesis. To perform the test, we use the significance level of $p<0.05$. It can be observed from Table 7 that all the p-values are less than 0.05. Hence, it shows that there exist a significant difference between Off-Mat, and other benchmark approaches, as indicated using t-values. As the significance level of variance for all the performance parameters is less than 0.05 in the t-test, we can say that hypothesis 1 is discarded, and hypothesis 2 is accepted.

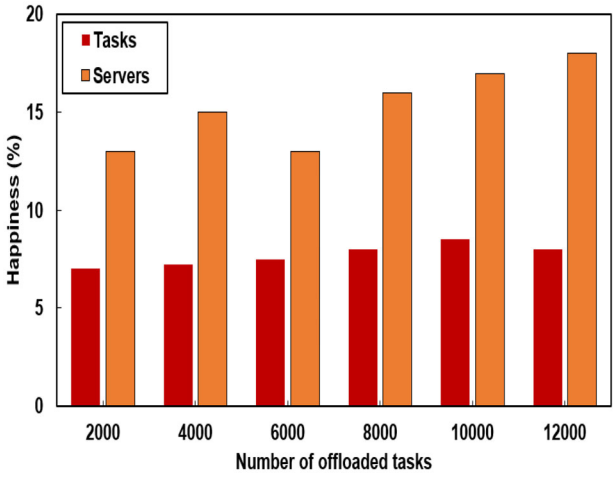

Fig. 19 Overall happiness performance

In Table 8, we perform a comparison of Off-Mat with all the algorithms for all the performance parameters using the ANOVA test. ANOVA test performs multiple comparisons at once for each performance measure for all benchmark approaches. It compares the mean value of two or more groups to identify whether the difference is statistically significant. We apply the same hypothesis case study that we have used earlier for the paired t-test. It can be observed that the p-values with respect to all the F-values such as 4.05 in total cost, 9.59 in total cost saving, 31.48 in energy consumption, 25.80 in total delay, 9.76 in throughput time, 9.00 in request filtering, 7.26 in task happiness, and 60.57 in server happiness are less than significance $p < 0.05$. Thus, we can conclude that the hypothesis 1 is again rejected and hypothesis 2 is accepted. It shows that the overall differences, compared with the benchmark approaches, are statistically significant.

7.6 Overall analysis

To further analyze the efficiency of proposed Off-Mat algorithm, we have compared the best, average, and worst-case value of all performance parameters. These values show the minimum, average, and maximum value for all the experiments. We have used the following gap value formula to identify the gap between the performance values of different approaches.

$$\text{Gap} = \frac{(\text{Average case value} - \text{Best case value})}{\text{Best case value}} \tag{29}$$

The lower gap value of any performance parameter shows that the average-case value of the offloading algorithm is closer to the best-case value. Table 9 shows the different values of the performance parameters. From the table, it can be identified that the Off-Mat approach generates the lowest gap values for request filtering, monetary cost,

Table 7 Statistical comparison of the Off-Mat approach for computational tasks offloading with benchmark approaches

Criteria	Algorithm	N	Mean	SD	t-value	p-value
Request filtering	Off-Mat	20	112.5	61.2372		
	Before filtering	20	450	244.949	3.78076	0.002026
Energy consumption	Off-Mat	20	86,792.1899	14,701.3749		
	Traditional	20	330,809.5898	83,753.5757	8.11658	< 0.00001
	Agent-based	20	190,049.3958	64,862.2789	4.39132	0.000615
Total delay	Off-Mat	20	10.6279	0.5129		
	Traditional	20	1256.1705	690.1886	5.10429	0.00016
	Agent-based	20	690.1886	1.0647	29.20413	< 0.00001
Monetary cost	Off-Mat	20	359.7386	251.8424		
	Traditional	20	1510.9021	1057.738	2.80114	0.01601
	Agent-based	20	1079.2158	755.5271	2.39022	0.034122
Fitness cost	Off-Mat	20	43,511.6861	7350.642		
	Traditional	20	165,890.2778	41,877.1359	8.14112	< 0.00001
	Agent-based	20	95,371.4714	32,432.4242	4.41082	0.000592
Cost saving	Off-Mat	20	1154.5471	805.5585		
	Traditional	20	3.3836	2.1475	3.78083	0.00262
	Agent-based	20	435.0699	301.8784	2.21276	0.047047
Throughput time	Off-Mat	20	2.0712	0.8856		
	Traditional	20	4.6343	0.9257	5.65884	0.000059
	Agent-based	20	3.3561	1.5478	2.03794	0.040912
Task happiness	Off-Mat	20	81.55	4.7534		
	First-Fit	20	74.3	4.5651	2.6946	0.022521
Server happiness	Off-Mat	20	65.4167	52		
	First-Fit	20	3.9296	1.5453	7.78303	0.000015

fitness cost, and second-best for energy consumption, total delay, cost-saving, throughput-time, task happiness, and server happiness.

Further, we analyze the improvement rate of proposed Off-Mat over other baseline approaches. For this, we have first analyzed the mean values of all performance parameters for all offloading approaches and calculated the improvement rate using the following formula:

$$\text{Improvement rate}(\%) = \frac{\text{Baseline} - \text{Off-Mat}}{\text{Off-Mat}} \times 100 \tag{30}$$

The improvement rate measures the gain for any performance parameter. For example reduction in the monetary cost or energy consumption of the Off-Mat over other offloading algorithms. For this, In Table 10, we have recorded the mean values of overall performance results for all experiments and calculate the improvement rate %, as shown in Table 11. The Off-Mat approach shows 281.15% and 118.97% energy consumption reduction over traditional and agent-based offloading approaches. Similarly, for average request delay, Off-Mat shows 994.29% and 79.29% reduction over traditional and agent-based offloading approaches. For cost analysis, results are encouraging and show 320% and 200% reduction over traditional and agent-based offloading approaches. In the case of cost-saving, the Off-Mat maximizes the cost-saving by 99.70% and 62.32% over traditional and agent-based techniques. Overall, fitness cost is also reduced by 281.25% and 119.18% over traditional and agent-based approaches. In the case of throughput time, the performance is improved by 123.74% and 62.03% over the traditional and agent-based offloading scheme. We have further analyzed the improvement for happiness metric and obtained that compared with the First-Fit approach, the Off-Mat shows 8.89% improvement for task happiness and 93.99% for server happiness. In Table 12, we have summarized the overall performance of proposed and existing schemes for different performance parameters.

7.7 Performance comparison with existing approaches

To further show the characteristics and advantages of the proposed study, we compare our results with some other existing techniques, i.e., Multi-Tenant Mobile Offloading [43], ENGINE [55], and Centralized broker-node based offloading [54] of literature as discussed in Table 13. Table 13 shows that [43] uses cloudlet-based offloading, [55] uses fog-based offloading, and [54] uses broker-node

Table 8 ANOVA test for all benchmark algorithms

Source of variation	SS	df	MS	F	p-value
ANOVA test for total cost					
Between groups	4.73E+6	2	2.36E+6	4.05128	0.035262
Within groups	1.05E+7	57	5.84E+5	–	–
Total	1.52E+7	59	–	–	–
ANOVA test for cost saving					
Between groups	4.73E+6	2	2.36E+6	9.59669	0.001456
Within groups	4.44E+6	57	2.46E+5	–	–
Total	9.17E+6	59	–	–	–
ANOVA test for energy					
Between groups	2.40E+11	2	1.20E+11	31.48128	< 0.00001
Within groups	8.00E+10	57	3.81E+09	–	–
Total	3.20E+10	59	–	–	–
ANOVA test for total delay					
Between groups	8.19E+6	2	4.09E+6	25.80103	< 0.00001
Within groups	3.33E+6	57	1.58E+5	–	–
Total	1.15E+6	59	–	–	–
ANOVA test for fitness cost					
Between groups	6.03E+10	2	3.01E+10	31.6673	< 0.00001
Within groups	2.00E+10	57	9.53E+8	–	–
Total	8.03E+10	59	–	–	–
ANOVA test for throughput-time					
Between groups	26.2778	2	13.1389	9.76397	0.001004
Within groups	28.2587	57	1.3457	–	–
Total	54.5364	59	–	–	–
ANOVA test for request filtering					
Between groups	472500	2	236250	9	0.001504
Within groups	551250	57	26250	–	–
Total	1023750	59	–	–	–
ANOVA test for task happiness					
Between groups	157.6875	1	157.6875	7.26085	0.022521
Within groups	217.175	38	21.7175	–	–
Total	374.8625	39	–	–	–
ANOVA test for server happiness					
Between groups	540.0208	1	540.0208	60.57554	0.000015
Within groups	89.1483	38	8.9148	–	–
Total	629.1692	39	–	–	–

based offloading. However, in the proposed approach, we have used agent-based offloading. The agent-based offloading makes faster decision making and improves the total delay and energy consumption. With respect to computing environment, [43] and [54] uses centralized model while [55] and proposed scheme used distributed computing environment. [43] and proposed approach support multi-tenancy and heterogeneity. [43, 55], and [54] have formulated either single objective or bi-objective optimization problem, while proposed approach is a multi-objective approach. The proposed Off-Mat scheme supports beneficial offloading via request filtering, which ensures the offloading under budget and maximum completion time constraint. Thus it improves the delay, monetary cost, and energy consumption performance. To improve stability, the Off-Mat provides task and server happiness parameters while other approaches have neglected such functionality. The overall time complexity of the Off-Mat approach is $O(TlnT)$ for offloading phase and $O(T^3 \times S^2)$ for allocation phase. It takes $2T$ messages to reach any offloading decision. The proposed Off-Mat approach is based on the concept of matching theory

Table 9 Overall analysis of algorithms

Performance parameter	Algorithm	Best	Average	Worst	Gap
Request filtering	Off-Mat	400	225	50	0.4375
	Agent-based	200	112.5	25	0.4375
Energy consumption	Off-Mat	55,922.9688	86,792.18991	102,148.1484	0.551995392
(mJ)	Traditional	234,876.432	330,809.5898	473,269.652	0.40844097
	Agent-based	78,292.144	190,049.3958	243,732.4364	1.427438899
Monetary cost	Off-Mat	23.34430714	359.7385983	714.0379762	14.41012102
(Dollar)	Traditional	98.04609	1510.902113	2998.9595	14.41012102
	Agent-based	70.03292143	1079.215795	2142.113929	14.41012102
Total delay	Off-Mat	10.0134	10.62787125	11.3998	0.061364896
(ms)	Traditional	281.927	1256.170525	2232.1265	3.455658823
	Agent-based	21.927	22.8296	24.423	0.041163862
Cost saving	Off-Mat	2285.962024	1154.547116	76.65569286	0.494940378
(Dollar)	Traditional	7.2464	3.383601429	1.0405	0.533064497
	Agent-based	857.8860714	435.0699194	29.96707857	0.492858162
Fitness cost	Off-Mat	51,187.90763	43511.68611	28,078.51854	0.149961619
	Traditional	237,112.9264	165,890.2778	117,929.7594	0.300374382
	Agent-based	122,207.7185	95,371.47137	39,497.17443	0.219595353
Throughput time	Off-Mat	1.240315963	2.071245499	3.666171	0.669933761
(ms)	Traditional	3.792454	4.634334966	6.40927	0.221988445
	Agent-based	1.735666333	3.356124856	5.7844075	0.933623296
Task happiness (%)	Off-Mat	88.3	81.55	75	0.076443941
	First-Fit	80	74.3	68	0.07125
Server happiness (%)	Off-Mat	69.5	65.41666667	60	0.058752998
	First-Fit	55	52	50.8	0.054545455

Table 10 Mean values of overall performance results of all experiments

Performance parameter	Off-Mat	Traditional	Agent-based	First-Fit
Energy consumption (mJ)	86792.189	330,809.589	190,049.395	
Average request delay (ms)	2.561	28.025	5.8025	
Monetary cost (Dollar)	359.7386	1510.9021	1079.2158	
Cost saving (Dollar)	1154.5471	3.3836	435.0699	
Fitness cost	43511.686	165,890.277	95,371.471	
Throughput time (ms)	2.0712	4.6343	3.3561	
Task happiness (%)	81.55			74.3
Server happiness (%)	65.4167			3.9296

applying the weighted-bipartite matching and stable matching in a distributed environment, which shows the novelty and advantages of the proposed Off-Mat strategy over the other existing schemes as illustrated in Table 13.

8 Conclusions and future work

This study analyzes the computational tasks offloading problem for mobile multi-tenant clouds. We first formulated it into an ILP model by using the objectives of energy and monetary cost under multi-constraints. The proposed $Off - Mat$ framework effectively minimizes the request-response time and improves the offloading performance. It also balances the trade-off of energy and monetary cost. We have first proved the problem complexity of the proposed optimization problem. Subsequently we proceed to solve the special case which is formed after relaxing certain conditions and solved it in polynomial time. Then we went ahead to solve the generalized case, where we have performed offloading and calculated the preferences for each of the servers and tasks selected by agents and perform a stable matching based heuristic. In general, the complexity

Table 11 Improvement in Off-Mat algorithm over other approaches

Performance parameter	Traditional	Agent-based	First-fit
Energy consumption			
Improvement % of Off-Mat over	+281.15%	118.97%	
Average request delay			
Improvement % of Off-Mat over	+994.29%	+79.29%	
Monetary cost			
Improvement % of Off-Mat over	+320%	+200%	
Cost saving			
Improvement % of Off-Mat over	+99.70%	+62.31%	
Fitness Cost			
Improvement % of Off-Mat over	+281.25%	+119.18%	
Throughput time			
Improvement % of Off-Mat over	+123.74	+62.03%	
Task happiness			
Improvement % of Off-Mat over			+8.89%
Server happiness			
Improvement % of Off-Mat over			+93.99%

of the $Off - Mat$ approach is analyzed and evaluated through extensive experiments. Simulation results verify that the 'Off-Mat' achieves superior performance compared with the other computation offloading strategies.

8.1 Future directions and open challenges

The applicability of the proposed solution can be explored in real-time systems and extended in the following main directions.

1. Mobile edge computing (MEC): MEC extends the application services and cloud capabilities to the edge of the network. It can be achieved through the dense deployment of servers or small-cell (pico, femto) base stations (BS) equipped with storage and computation resources. Mobile edge environment ensures efficient network operation and service distribution, minimize latency, and offers an enhanced service user experience. Despite the benefits of MEC, there are still major challenges such as the real-time mobile applications are extremely time-sensitive and energy-sensitive. Thus due to the dynamics of edge networks, the long execution times of such applications can lead to high energy consumption. Therefore, there is a requirement to design an efficient MEC framework for computation offloading [56].
2. Fog computing: Fog computing extends the cloud model to the network edge to enable Internet of Things (IoT) based services. For future Internet, the mobile fog technology is an integral framework of fog computing supporting seamless mobile computing and latency-enabled services. Nonetheless, the critical challenges for mobile fog-based computation offloading are: (i) Which process or module of the application to be offloaded? (ii) How to offload computation?, and (iii) Where to offload? Moreover, the geographical distribution, mobility, and heterogeneity of mobile devices also impose some additional challenges in mobile-fog [57].
3. Multi-Tier Edge-Clouds: To accomplish low end-to-end latency, multi-tier edge-clouds pushes units of computation i.e., cloudlets to the edge of the network in the coverage area. Hence, some recent works have adopted a hierarchical cloudlets arrangement in different edge-tiers. In such architecture, the higher tiers comprises some more powerful edge cloudlets. So, whenever any case of overloading occur, the higher tiers can receive migration demands from the lower-tier cloudlets. Thus, the cost and energy-efficiency issues in hierarchical edge-clouds is an open problem and well suited with the 5G business models [42].
4. Fault-tolerance: During run-time network contexts such as strength of the signal, bandwidth, latency, etc. are periodically changed, and short-term failures can occur for short-time span. Hence, the failure aware partitioning of offloading applications during run-time is a critical component [45].
5. Privacy and Security: A compromised mobile device can perform malicious activities without the user's knowledge, encompassing the access of sensitive information. In consequence, the offloading of mobile applications to the remote server requires a more secure environment. In fact, mobile offloading offers a computation mechanism. Thus, due to the limited security of the mobile OS, partial or full offloading,

Table 12 Comparison of performance metrics for proposed and existing method

Approach	Number of tasks	Monetary cost ($)	Total energy consumption (mJ)	Average request delay (ms)	Fitness cost	Throughput time (ms)	Task happiness (in %)	Server happiness (in %)
Off-Mat	100	23.34430714	55,922.9688	2.5033	28,078.51854	1.240315963	88.3	60
Traditional	100	98.04609	234,876.432	28.192	117,929.7594	3.792454		
Agent-based	100	70.03292143	78,292.144	5.2192	39,497.17443	1.735666333		
First-Fit	100						80	52
Off-Mat	200	118.1309048	78,783.0983	3.4969	39,505.15916	1.455705	85	63
Traditional	200	496.1498	255,367.3863	26.6444	128,160.8552	3.850158125		
Agent-based	200	354.3927143	110,237.806	6.6444	55,459.73304	1.80569125		
First-Fit	200						78.3	55
Off-Mat	300	237.1990952	82,557.9506	2.561	41,392.25657	1.461334	82.5	63
Traditional	300	996.2362	276,142.2386	28.025	138,546.9007	4.092763667		
Agent-based	300	711.5972857	175,821.9879	5.8025	88,250.83777	1.8704684		
First-Fit	300						76	51
Off-Mat	400	355.4175238	88,571.3293	2.8191	44,400.61217	1.5488232	80	69
Traditional	400	1492.7536	297,155.6173	28.049	149,060.5883	4.226318286		
Agent-based	400	1066.252571	192,203.9043	5.8049	96,446.79472	2.774147667		
First-Fit	400						72	51.9
Off-Mat	500	475.8500714	92,855.8646	2.6029	46,545.34137	1.6210575	78.5	68
Traditional	500	1998.5703	321,749.0892	28.291	161,367.6627	4.3311994		
Agent-based	500	1427.550214	238,440.1526	5.8291	119,572.3035	3.952032571		
First-Fit	500						71.5	50.8
Off-Mat	600	594.1903095	95,350.586	2.8499	47,793.62387	2.766423333	75	69.5
Traditional	600	2495.5993	364,510.0581	28.7244	182,752.0187	4.75993625		
Agent-based	600	1782.570929	239,934.874	5.7244	120,322.4296	3.967658125		
First-Fit	600						68	51.3
Off-Mat	700	714.0379762	98,147.5733	2.7334	49,190.06957	2.810134		
Traditional	700	2998.9595	423,406.2451	28.8491	212,191.5108	5.61258		
Agent-based	700	2142.113929	241,731.8613	5.8491	121,214.7794	4.958927	73.5	71
First-Fit	700						66.5	52.2
Off-Mat	800	830.704642	102,148.1484	2.5631	51,187.90763	3.666171	71	72.5
Traditional	800	3488.9595	473,269.652	27.901	237,112.9264	6.40927		
Agent-based	800	2492.113929	243,732.4364	5.7901	122,207.7185	5.7844075		
First-Fit	800						65	53

Table 13 Off-Mat comparison with the existing works

Features	Existing offloading methods			Proposed approach
	Multi-tenant mobile offloading [43]	ENGINE [55]	Centralized broker-node based offloading [54]	
Working model	Scheduling methods are proposed to enhance serving performance for multi-tenant mobile offloading systems	Computation is offloaded in a mobile-edge computing environment considering user budget and service cost	Centralized broker-node based task scheduling is developed to minimize the energy consumption and monetary cost	Agent based distributed computation offloading strategies are proposed for mobile multi-tenant clouds
Offloading type	Cloudlet-based	Fog-based	Broker-node based	Agent-based
Computing environment	Centralized	Distributed	Centralized	Distributed
Performance parameters	Accuracy, delay	Task completion time	Energy consumption, monetary cost	Energy consumption, delay, monetary cost, total saving, happiness
Multitenancy	✓	×	×	✓
Heterogeneity	✓	×	✓	✓
Task and server preferences	×	×	×	✓
Heuristic	Clock synchronization, Approximate sorting, Deep reinforcement learning	Greedy technique, Simulated annealing, Brute Force	Linear programming	Matching theory (weighted bipartite matching, and stable matching)
Budget	×	✓	×	✓
Maximum completion time	✓	×	✓	✓
Request filtering	×	×	×	✓
Offloading complexity	Not considered	$O(T)$	Not considered	$O(TlnT)$
Resource allocation complexity	Not considered	Not considered	Not considered	$O(T^3 \times S^2)$
Message complexity	Not considered	Not considered	Not considered	$2T$

allows us to run security applications in a more robust and secure environment [51]. However, privacy leakage, network attack, information theft during data transmission are some of the open issues during the computation offloading [65].

6. Deep learning-based offloading: Deep learning enables mining & processing of a variety of unstructured data collected through smartphones. So that more intelligent cognitive and robust services can be offered to the MEC systems. Thus, some of the popular deep learning techniques such as CNN, GRU, RNN, and LSTM can offer cognitive services for network functions, traffic, load, including other system measures for enhancing the quality of service (QoS). However, designing a user mobility prediction using deep learning algorithm enabling some decision-making mechanisms is one of the open problems for computation task offloading and migration [65].
7. Blockchain: Data integrity violation is one of the drawbacks of computation offloading. Most of the classical integrity preservation methods are mostly rely on the central entity and not necessarily convenient for the 5G network due to the single point of failure. Blockchain is become an emerging disruptive paradigm which guarantees data completeness. Thus, the blockchain is a promising future for data integrity preservation during the computation offloading [66].

Acknowledgements We would like to thank the editor and anonymous reviewers for their valuable and insightful comments, which

have significantly improved the quality and acceptability of the paper. The work is partially supported by the Department of Science and Technology (DST), Government of India under ICPS Programme through the Project No.: DST/ICPS/CPS-Individual/2018/403(G), "Low-cost Energy-Efficient Cloud for Cyber-Physical Disaster Management Systems." The first author, Yashwant Singh Patel, acknowledges Visvesvaraya PhD Scheme, MeitY, Govt. of India<-MEITY-PHD-2525> for supporting this research.

References

1. Dinh, H.T., Lee, C., Niyato, D., Wang, P.: A survey of mobile cloud computing: architecture, applications, and approaches. Wirel. Commun. Mob. Comput. **13**, 1587–1611 (2013)
2. Satyanarayanan, M.: Mobile computing: the next decade. SIGMOBILE Mobile Computing and Communications Review, pp. 2–10 (2011)
3. Satyanarayanan, M.: Fundamental challenges in mobile computing. In: Proceedings of the fifteenth annual ACM symposium on Principles of distributed computing (PODC '96), pp. 1–7. Association for Computing Machinery, New York (1996)
4. Qi, H., Gani, A.: Research on mobile cloud computing: review, trend and perspectives. In: Second International Conference on Digital Information and Communication Technology and It's Applications (DICTAP), pp. 195–202. Bangkok (2012)
5. Chun, B.-G., Ihm, S., Maniatis, P., Naik, M., Patti, A.: CloneCloud: elastic execution between mobile device and cloud. In: Proceedings of the Sixth Conference on Computer systems (EuroSys '11), pp. 301–314. Association for Computing Machinery, New York (2011)
6. 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: 2012 Proceedings INFOCOM, pp. 945–953. IEEE, Orlando (2012)
7. Satyanarayanan, M., Bahl, P., Caceres, R., Davies, N.: The case for VM-based cloudlets in mobile computing. IEEE Pervasive Comput. **8**(4), 14–23 (2009)
8. Rim, H., Kim, S., Kim, Y., Han, H.: Transparent method offloading for slim execution. In: 1st International Symposium on Wireless Pervasive Computing, pp. 1-6. Phuket (2006)
9. Cuervo, E. et al.: MAUI: making smartphones last longer with code offload. In: Proceedings of the 8th international conference on Mobile systems, applications, and services (MobiSys '10), pp. 49–62. Association for Computing Machinery, New York (2010)
10. Wang, X., Wang, J., Wang, X., Chen, X.: Energy and delay tradeoff for application offloading in mobile cloud computing. IEEE Syst. J. **11**(2), 858–867 (2017)
11. Wen, Y., Zhang, W., Luo, H.: Energy-optimal mobile application execution: taming resource-poor mobile devices with cloud clones. In: Proceedings IEEE INFOCOM, pp. 2716–2720. IEEE, Orlando (2012)
12. Zhang, W., et al.: Energy-optimal mobile cloud computing under stochastic wireless channel. IEEE Trans. Wireless Commun. **12**(9), 4569–4581 (2013)
13. Song, J., Cui, Y., Li, M., Qiu, J., Buyya, R.: Energy-traffic tradeoff cooperative offloading for mobile cloud computing. In: 22nd international symposium of quality of service (IWQoS), pp. 284–289. IEEE, Hong Kong (2014)
14. Xia, F., Ding, F., Li, J., et al.: Phone2Cloud: Exploiting computation offloading for energy saving on smartphones in mobile cloud computing. Inf. Syst. Front. **16**, 95–111 (2014)
15. Xu, H., Li, B.: Egalitarian stable matching for VM migration in cloud computing. In: IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), pp. 631–636. IEEE, Shanghai (2011)
16. Kim, G., Lee, W.: Stable matching with ties for cloud-assisted smart TV services, IEEE International Conference on Consumer Electronics (ICCE), pp. 558–559. IEEE, Las Vegas (2014)
17. Oualhaj, O. A., Sabir, E., Kobbane, A., Ben-Othman, J., Koutbi, M. E.: A college admissions game for content caching in heterogeneous delay tolerant networks. In: 23rd International Conference on Telecommunications (ICT), pp. 1–5. Thessaloniki (2016)
18. Saad, W., Han, Z., Zheng, R., Debbah, M., Poor, H. V.: A college admissions game for uplink user association in wireless small cell networks. In: IEEE INFOCOM- IEEE Conference on Computer Communications, pp. 1096–1104. IEEE, Toronto (2014)
19. Clinch, S., Harkes, J., Friday, A., Davies, N., Satyanarayanan, M.: How close is close enough? Understanding the role of cloudlets in supporting display appropriation by mobile users. In: International Conference on Pervasive Computing and Communications, pp. 122–127. IEEE, Lugano (2012)
20. Xia, Q., Liang, W., Xu, W.: Throughput maximization for online request admissions in mobile cloudlets. In: 38th Annual IEEE Conference on Local Computer Networks, pp. 589–596. IEEE, Sydney (2013)
21. Huang, D., Wang, P., Niyato, D.: A dynamic offloading algorithm for mobile computing. IEEE Trans. Wireless Commun. **11**(6), 1991–1995 (2012)
22. Zhang, W., Wen, Y., Wu, D.O.: Energy-efficient scheduling policy for collaborative execution in mobile cloud computing. In: Proceedings IEEE INFOCOM, pp. 190–194. IEEE, Turin (2013)
23. Gu, Y., Saad, W., Bennis, M., Debbah, M., Han, Z.: Matching theory for future wireless networks: fundamentals and applications. IEEE Commun. Mag. **53**(5), 52–59 (2015)
24. Barou, M., Balinski, M.: Erratum: the stable allocation (or ordinal transportation) problem. Math. Oper. Res. **27**, 662–680 (2002)
25. Gale, D., Shapley, L.S.: College admissions and the stability of marriage. Am. Math. Mont. JSTOR. **69**(1), 9–15 (1962)
26. Xu, H., Li, B.: Anchor: a versatile and efficient framework for resource management in the Cloud. IEEE Trans. Parallel Distrib. Syst. **24**(6), 1066–1076 (2013)
27. Mairson, H.: The stable marriage problem. The Brandeis Review. **12**, (1992)
28. Brito, I., Meseguer, P.: Distributed stable matching problems. In: Principles and Practice of Constraint Programming-CP 2005. Springer, pp. 152–166 (2005)
29. Meng, T.: Security and performance tradeoff analysis of offloading policies in mobile Cloud Computing (2017)
30. Barbera, M. V., Kosta, S., Mei, A., Stefa, J.: To offload or not to offload? The bandwidth and energy costs of mobile cloud computing. In: Proceedings IEEE INFOCOM, pp. 1285–1293. IEEE, Turin (2013)
31. Huang, D., Wang, P., Niyato, D.: A dynamic offloading algorithm for mobile computing. IEEE Trans. Wireless Commun. **11**(6), 1991–1995 (2012)
32. Zheng, K., et al.: Delay-optimized offloading for mobile cloud computing services in heterogenous networks. In: Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, LNICST, 133, pp. 122–131 (2014)
33. Kwak, J., Kim, Y., Lee, J., Chong, S.: DREAM: dynamic resource and task allocation for energy minimization in mobile cloud systems. IEEE J. Sel. Areas Commun. **33**(12), 2510–2523 (2015)
34. Nir, M., Matrawy, A., St-Hilaire, M.: An energy optimizing scheduler for mobile cloud computing environments. In: IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), pp. 404–409. IEEE, Toronto (2014)

35. Liu, X., Li, Y., Chen, H.: Wireless resource scheduling based on backoff for multiuser multiservice mobile cloud computing. IEEE Trans. Veh. Technol. **65**(11), 9247–9259 (2016)
36. Meskar, E., Todd, T.D., Zhao, D., Karakostas, G.: Energy aware offloading for competing users on a shared communication channel. IEEE Trans. Mob. Comput. **16**(1), 87–96 (2017)
37. Chen, X.: Decentralized computation offloading game for mobile cloud computing. IEEE Trans. Parallel Distrib. Syst. **26**(4), 974–983 (2015)
38. Chen, X., Jiao, L., Li, W., Fu, X.: Efficient multi-user computation offloading for mobile-edge cloud computing. IEEE/ACM Trans. Network. **24**(5), 2795–2808 (2016)
39. Liu, Y., Lee, M.J., Zheng, Y.: Adaptive multi-resource allocation for cloudlet-based mobile cloud computing system. IEEE Trans. Mob. Comput. **15**(10), 2398–2410 (2016)
40. Kuang, Z., Guo, S., Liu, J., Yang, Y.: A quick-response framework for multi-user computation offloading in mobile cloud computing. Fut. Gener. Comput. Syst. **81**, 166–176 (2018)
41. Chekuri, C., Khanna, S.: A PTAS for the multiple knapsack problem. In: Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms (SODA '00), Society for Industrial and Applied Mathematics, pp. 213–222. ACM (2000)
42. El Haber, E., Nguyen, T.M., Assi, C.: joint optimization of computational cost and devices energy for task offloading in multi-tier edge-clouds. IEEE Trans. Commun. **67**(5), 3407–3421 (2019)
43. Fang, Z., Lin, J.-H., Srivastava, M.B., Gupta, R.K.: Multi-tenant mobile offloading systems for real-time computer vision applications. In: Proceedings of the 20th International Conference on Distributed Computing and Networking (ICDCN '19), pp. 21–30. Association for Computing Machinery, New York (2019)
44. Ghobaei-Arani, M., Souri, A., Rahmanian, A.A.: Resource management approaches in fog computing: a comprehensive review. J. Grid Comput. **18**, 1–42 (2020)
45. Lakhan, A., Li, X.: Transient fault aware application partitioning computational offloading algorithm in microservices based mobile cloudlet networks. Computing **102**, 105–139 (2020)
46. Verma, R.K., Panigrahi, C.R., Pati, B., Sarkar, J.L.: An efficient approach for running multimedia applications using mobile cloud computing. In: Pati, B., Panigrahi, C., Buyya, R., Li, K.C. (eds.) Advanced Computing and Intelligent Engineering, Advances in Intelligent Systems and Computing, vol. 1089. Springer, Singapore (2020)
47. Shakarami, A., Shahidinejad, A., Ghobaei-Arani, M.: A review on the computation offloading approaches in mobile edge computing: a game-theoretic perspective. Softw: Pract Exper. 1–41 (2020)
48. Nagasundari, S., Ravimaran, S., Uma, G.V.: Enhancement of the dynamic computation-offloading service selection framework in mobile cloud environment. Wireless Pers. Commun. **112**, 225–241 (2020)
49. De, D., Mukherjee, A., GuhaRoy, D.: Power and delay efficient multilevel offloading strategies for mobile cloud computing. Wireless Pers. Commun. **112**, 2159–2186 (2020)
50. Ghobaei-Arani, M., Khorsand, R., Ramezanpour, M.: An autonomous resource provisioning framework for massively multiplayer online games in cloud environment. J. Netw. Comput. Appl. **142**, 76–97 (2019)
51. Derhab, A., Belaoued, M., Guerroumi, M., Khan, F.A.: Two-factor mutual authentication offloading for mobile cloud computing. IEEE Access. **8**, 28956–28969 (2020)
52. Ghobaei-Arani, M., Rahmanian, A.A., Souri, A., Rahmani, A.M.: A moth-flame optimization algorithm for web service composition in cloud computing: simulation and verification. Softw. Pract. Exper. **48**, 1865–1892 (2018)
53. Ghobaei-Arani, M., et al.: CSA-WSC: cuckoo search algorithm for web service composition in cloud environments. Soft. Comput. **22**, 8353–8378 (2018)
54. Nir, M., Matrawy, A., St-Hilaire, M.: Economic and energy considerations for resource augmentation in mobile cloud computing. IEEE Trans. Cloud Comput. **6**(1), 99–113 (2018)
55. Chen, L., et al.: ENGINE: cost effective offloading in mobile edge computing with fog-cloud cooperation. arXiv:1711.01683 (2017)
56. Alfakih, T., Hassan, M.M., Gumaei, A., Savaglio, C., Fortino, G.: Task offloading and resource allocation for mobile edge computing by deep reinforcement learning based on SARSA. IEEE Access **8**, 54074–54084 (2020)
57. Alam, M.G.R., et al.: Autonomic computation offloading in mobile edge for IoT applications. Fut. Gener. Comput. Syst. **90**, 149–157 (2019)
58. Enayet, A., et al.: Mobility-aware optimal resource allocation architecture for big data task execution on mobile cloud in smart cities. IEEE Commun. Mag. **56**(2), 110–117 (2018)
59. Islam, M.M., Razzaque, M.A., Hassan, M.M., Ismail, W.N., Song, B.: Mobile cloud-based big healthcare data processing in smart cities. IEEE Access. **5**, 11887–11899 (2017)
60. Bedi, R.K., Singh, J., Gupta, S.K.: Design and implementation of an efficient multi cloud storage approach for resource constrained mobile devices. Cluster Comput. **22**, 13143–13157 (2019)
61. Durga, S., Mohan, S., Peter, J.D., et al.: Context-aware adaptive resource provisioning for mobile clients in intra-cloud environment. Cluster Comput. **22**, 9915–9928 (2019)
62. Saleem, M., Saleem, Y., Hayat, M.F.: Stochastic QoE-aware optimization of multisource multimedia content delivery for mobile cloud. Cluster Comput. **23**, 1381–1396 (2020)
63. Elashri, S., Azim, A.: Energy-efficient offloading of real-time tasks using cloud computing. Cluster Comput (2020)
64. Milan, S.T., Rajabion, L., Darwesh, A., et al.: Priority-based task scheduling method over cloudlet using a swarm intelligence algorithm. Cluster Comput. **23**, 663–671 (2020)
65. Miao, Y., et al.: Intelligent task prediction and computation offloading based on mobile-edge cloud computing. Fut. Gener. Comput. Syst. **102**, 925–931 (2020)
66. Xu, X., Chen, Y., Zhang, X., Liu, Q., Liu, X., Qi, L.: A blockchain-based computation offloading method for edge computing in 5G networks. Softw. Pract. Exper. 1–18 (2019)
67. Beloglazov, A., Buyya, R.: Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers. Concurr. Comput. **24**(13), 1397–1420 (2012)
68. Statservice. http://moses.us.es/statservice
69. Parejo, J.A., Garca, J., Ruiz-Cortl's, A., Riquelme, J.C.: Statservice: Herramienta de analisis estadstico como soporte para la investigacin con metaheursticas. In: Actas del VIII Congreso Expaol sobre Metaheursticas, Algoritmos Evolutivos y Bio-inspirados (2012)
70. Social Science Statistics. https://www.socscistatistics.com/

Yashwant Singh Patel is currently pursuing his Ph.D. in Computer Science and Engineering at the Indian Institute of Technology Patna, India. His Ph.D. work is Supported by Visvesvaraya PhD Scheme, MeitY, Govt. of India. He received B.E. degree in Computer Science and Engineering from Rajiv Gandhi Proudyogiki Vishwavidyalaya, Bhopal, India, in 2011 and M.Tech. degree in Computer Science and Engineering from KIIT University, Bhubaneswar, India, in 2014. His research interests include Cloud Computing, Edge Computing, Distributed Algorithms, and Grid Computing.

Manoj Reddy is an undergrad student in the Computer Science Department at Indian Institute of Technology Patna, India. His research investigates the energy and cost tradeoff for computational tasks offloading in mobile multi-tenant clouds. Manoj's background spans a diverse range of disciplines and mediums: web design, project management and software development.

Rajiv Misra is currently working as an Associate Professor in the Department of Computer Science and Engineering, Indian Institute of Technology (IIT) Patna, India. He received M.Tech. degree in Computer Science and Engineering from IIT Bombay and Ph.D. degree in the area of mobile computing from IIT Kharagpur. His research interests include Distributed Systems, Cloud Computing, Big Data Computing, Consensus in Blockchain, Cloud IoT Edge Computing, Adhoc Networks and Sensor Networks. He has contributed significantly to these areas and published more than 90 papers in high quality journals and conferences. His h-index is 13 with more than 800 citations. He has authored papers in IEEE Transactions on Mobile Computing, IEEE Transactions on Parallel and Distributed Systems etc. He is a senior member of IEEE.