MobiQoR: Pushing the Envelope of Mobile Edge Computing via Quality-of-Result Optimization

Yongbo Li, Yurong Chen, Tian Lan, Guru Venkataramani
George Washington University, Washington, DC
{lib, gabrielchen, tlan, guruv}@gwu.edu

Abstract—**Mobile edge computing aims at improving application response time and energy efficiency by deploying data processing at the edge of the network. Due to the proliferation of Internet of Things and interactive applications, the ever-increasing demand for low latency calls for novel approaches to further pushing the envelope of mobile edge computing beyond existing task offloading and distributed processing mechanisms. In this paper, we identify a new tradeoff between Quality-of-Result (QoR) and service response time in mobile edge computing. Our key idea is motivated by the observation that a growing set of edge applications involving media processing, machine learning, and data mining can tolerate some level of quality loss in the computed result. By relaxing the need for highest QoR, significant improvement in service response time can be achieved. Toward this end, we present a novel optimization framework, MobiQoR, which minimizes service response time and app energy consumption by jointly optimizing the QoR of all edge nodes and the offloading strategy. The proposed MobiQoR is prototyped using Parse, an open source mobile back-end tool, on Android smartphones. Using representative applications including face recognition and movie recommendation, our evaluation with real-world datasets shows that MobiQoR reduces response time and energy consumption by up to 77% (in face recognition) and 189.3% (in movie recommendation) over existing strategies under the same level of QoR relaxation.**

I. Introduction

The growing popularity of smartphones, along with the rapid emergence of applications from big data analytics to media processing, calls for ever-increasing data processing capabilities, yet satisfying low response time and energy consumption in mobile applications. This gives rise to edge computing [19], [53] (also known as fog computing [9], [11]), a new computing paradigm that addresses the concerns by shifting data processing and computation to the edge of the network. As smartphones and other mobile devices are becoming both data producers and consumers (e.g., recording and augmenting a video stream for visual enhancement), edge computing enables workload processing at the proximity of data sources and end devices. It not only reduces response time compared to traditional cloud-based computing platforms that are constrained by the speed of data transmission, but also overcomes persistent resource limitations on mobile devices that necessitate computation offloading. Prior work has demonstrated the potential benefits of edge computing [22], [60] and developed tools for task offloading [2], [4], [29], [51].

However, emerging mobile applications on the horizon require renewed efforts to push the envelope of mobile edge computing. In particular, ultra-low response time is crucial for object recognition and image processing algorithms, which are widely used in popular contemporary virtual reality and augmented reality systems [56], as well as in interactive and real-time services [26], [33]. For instance, a wearable camera offering visual map services has a preferred response time between 25ms to 50ms [5]. For gaming applications, between 45 and 75 ms latency, there exists a linear correlation between increased latency and decreased game session length [8], suggesting an inverse relationship between response time and user experience. Furthermore, battery life has always been a big concern for mobile users. Exploring energy saving opportunities requires new engineering artifacts for workload offloading and processing in this heterogeneous environment [31].

In this paper, we explore a novel dimension - *Quality of Results (QoR)* - in mobile edge computing and propose a systematic optimization framework, MobiQoR, to trade QoR for reduced response time and extra energy saving. Our key idea is motivated by the observation that for a growing set of edge applications involving media processing, machine learning, and data mining, a perfect result is not always necessary, while a lower-quality or less-than-optimal result is sufficient [44]. Thus, relaxing QoR in such applications alleviates the required computation workload and enables a significant reduction of response time and energy consumption in mobile edge computing, beyond the boundary of existing task offloading and distributed processing mechanisms. The metric QoR depends on the application domain and the corresponding algorithm. As examples, for online recommendation algorithms, QoR can be measured by the normalized mean error between the predicted and actual scores; for object recognition algorithms, QoR can be defined as the percentage of correctly identified objects or patterns. In Section III, we formally define the coherent QoR metric for different classes of applications.

We consider a mobile edge environment where computing tasks can be divided, offloaded and processed in parallel by distributed edge nodes, including neighboring mobile devices, micro-datacenters, routers and base stations. The edge nodes are equipped with heterogeneous network and computing resources. Our MobiQoR optimization aims to minimize the response time and energy consumption of mobile apps by jointly optimizing the task offloading and the selection of edge nodes' QoR levels. While the focus of this paper is not on implementing QoR relaxation for different mobile apps, our

MobiQoR framework is easily extensible to different QoR relaxation techniques [44] that offer varied control knobs to tune the achievable QoR levels. MobiQoR leverages the mapping of QoR to tunable control knobs in algorithms of different applications. Thus, it enables the search for optimal offloading and QoR decisions, while guaranteeing that the quality of the computed results meets end users' expectations. An efficient optimization algorithm is developed to solve the proposed MobiQoR optimization. In particular, we show that the optimization can be cast into linear program (LP) when the tradeoff between QoR and processing speed is approximated as a linear function. The proposed MobiQoR framework and optimization algorithms are prototyped on Android smartphones using Parse open source back-end and mobile SDK [4]. We deploy MobiQoR in our edge testbed consisting of five heterogeneous nodes and build a database for representative applications including face recognition and movie recommendation, measuring their QoR tradeoffs and energy profiles. Using real-world datasets and in realistic network environments, we validate significant improvements in response time and energy saving for different QoR constraints.

In summary, the contributions of our work are:

1) We exploit a new tradeoff between QoR and computing speed in mobile edge computing. Taking advantage of the fact that many mobile applications can tolerate minor QoR loss, the tradeoff allows us to minimize both response time and energy consumption for a large set of applications beyond the conventional boundaries.
2) To leverage this tradeoff, we propose MobiQoR, an optimization framework that jointly determines task offloading strategies and edge nodes' QoR levels for minimizing response time and energy consumption. It enables us to tap into heterogeneous resources offered at network edge in a QoR-aware fashion.
3) We implement and evaluate MobiQoR on Android using representative applications including face recognition and movie recommendation. Significant improvements (up to 77% and 189.3% for the two applications, respectively) are validated using real-world datasets.

II. Related Work

With the proliferation of Internet of Things and interactive applications, the computing resources are brought closer to users by a new computing paradigm, edge computing [19], [53], also known as fog computing [9], [11]. The heterogeneity of edge computing environment and mobile users' ever-increasing demands for ultra-low latency make the traditional cloud-assisted mobile computing techniques such as workload offloading [14], [15], [21], [32], [41], [51], [59], server selection [18] not directly applicable to this new paradigm. The energy tradeoff [7], [38] between local and remote computing has been studied when making mobile offloading decisions. Our work is proposed to further push the envelope of mobile edge computing beyond existing work by introducing a new optimization dimension, QoR, and jointly optimizing the energy consumption and service latency in an online manner.

Especially, an area that attracts enormous research interests is workload partitioning techniques to allow workload offloading, such as method level offloading [29], VM-based synchronization [20], code refactoring [63], middleware-assisted method [30], [42], offloading-aware development [28], and state checkpointing [32]. When determining which portion of workload to offload, Zhang et al. [62] utilize a graph-based method, Rong et al. [48] adopt a Markov process approach, and Chun et al. [14] propose to combine offline analyzer and dynamic profiler in CloneCloud. Most of these works consider single traditional remote server as the offloading destination. Although the focus of our paper is orthogonal to this line of work, considering different workload partitioning techniques within the joint optimization framework we propose in MobiQoR opens up interesting avenues for researches in the context of edge computing.

Existing work has shown the feasibility to save energy and/or reduce latency by relaxing the workload computation accuracy requirement [24], [45], either via software-based techniques [44] (e.g., workload discarding [54], parameter tuning [13], [55]), hardware-based approximation techniques [49], [58], or combination of both [12]. MobiQoR serves a goal that is complementary to this direction of work focusing on approximated computing techniques. The techniques developed in these works generally give no special consideration on the unique property of heterogeneous edge computing environment. In contrast, MobiQoR proposes an algorithmic solution which makes separate QoR level selection for each edge node. On the other hand, techniques proposed in these works can help extend MobiQoR to more ways of QoR tuning.

III. Problem Formulation

Consider a mobile edge network with a client device and N wirelessly connected edge nodes, such as neighboring mobile devices, micro-datacenters, routers and base stations. To process a user request with total workload W (e.g., W images to tag or W recommendation scores to compute), we partition the workload into N tasks of size $w_1, \ldots, w_N$, satisfying $\sum_{i=1}^{N} w_i = W$, where each task of size w_i is offloaded to edge node i and processed in a distributed fashion. Our goal in MobiQoR optimization is to minimize both response time and energy consumption by jointly determining QoR and task assignments to all edge nodes.

Modeling the QoR-speed Tradeoff. Many emerging applications in mobile edge computing such as recommendation, data mining, object recognition, media (e.g., video and image) processing and data analytics expose different control knobs that allow us to exploit the tradeoff between QoR and processing speed. For example, object recognition algorithms [57] often require extraction of a given number of layers with different wavelengths and orientations from the original input images for analysis. By adjusting the number of layers extracted, we can relax the achieved QoR to achieve a speedup in the object recognition. For example, our evaluation in Figure 1 shows a $1.5\times$ speedup when the QoR measured by recognition accuracy slightly reduces from 0.99 to 0.89. In recommendation algorithms [50], the amount of reference data used to make the recommendation can serve as the control knob for achieving this tradeoff. These methods are referred to as parameter level substitution in prior work [44]. Other techniques of approximated computing include discarding or substituting certain different subsets of tasks in non-approximated computation, to achieve varied QoR-speed tradeoff. Our MobiQoR framework is flexible and can utilize such approximated computing techniques. With elastic QoR, we can significantly speed up the processing of user requests, while tolerating slight QoR degradation if permitted by such applications.

In our proposed framework, we allow each edge node to select a different QoR level, exploiting the tradeoff between QoR and processing speed. Let $A^\star$ be the highest accuracy that can be achieved by an application, and A be the actual accuracy achieved with QoR-relaxed (or approximated) computation. We define QoR as "1 minus the normalized error" or equivalently "normalized accuracy", i.e.,

$$q = 1 - (A^\star - A)/A^\star = A/A^\star \in [0, 1]. \tag{1}$$

When $q = 0$, the QoR is the lowest and the approximated results fully diverge from non-approximated results, whereas $q = 1$ means the QoR is the highest and all approximated results exactly match the non-approximated results. The measuring metric for the accuracy A and $A^\star$ is specific to the application domain. For example, for an object recognition application that identifies the tags for a batch of 50 objects, $A^\star = 98\%$ means on average only 1 object is falsely identified out of the 50 objects by the non-approximated algorithm. If by approximated computing, on average, 45 out of the 50 objects are correctly identified, A is 90% and the resulted QoR q is measured by normalized accuracy $90\%/98\% = 0.9184$. For another example that predicts the score rating for 10 user-movie entries, where the score ranges from 0 to 5, the prediction error is often quantified by the Normalized Mean Absolute Error (NMAE). Thus, we define accuracy A and $A^\star$ as "1 minus NMAE". If the NMAE achievable by the approximated and non-approximated algorithms are 0.1 and 0.05 respectively, we have $A^\star = 1 - 0.05$ and $A = 1 - 0.1$, resulting in $q = 0.9474$ in this case.

Formulating the QoR Constraint. For a given application, we use Q to denote the minimum QoR that is acceptable to an end client. Let $q_i \in [0, 1]$ be the QoR level assigned to edge node i. Based on our workload distribution, the aggregated QoR collectively achieved by all nodes in the system must be at least Q, i.e.,

$$\sum_{i=1}^{N} w_i q_i / W \geq Q, \tag{2}$$

where $w_i q_i / W$ is weighted QoR obtained on node i that processes workload w_i (out of total W). In practice, the QoR constraint Q can vary from one application/context to another and also depends on user preferences. For example, an average $Q = 0.80$ could be sufficient for some Netflix-type movie recommendation app, whereas a face recognition app may

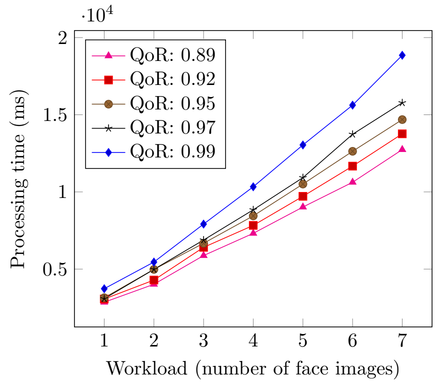

Fig. 1. Illustration of the linear relationship between processing time and size of workload for different QoR values (using object recognition).

need higher QoR. Assigning elastic QoR to different nodes, we can exploit the tradeoff between QoR and process speed. More precisely, for a given Q, our goal is to find the optimal QoR assignment q_i and workload share w_i for each node, to minimize both service latency and energy.

To leverage the tradeoff between QoR and processing speed, we use $T_i(q_i)$ to denote the time required to complete a unit workload (e.g., processing 1 image or calculating 1 recommendation score) by edge node i with desired QoR q_i. Then, the total processing time for w_i units of workload can be simply estimated by $w_i T_i(q_i)$ plus a constant overhead. For an object recognition app, we run batches of workload with different sizes on four heterogeneous edge nodes, and record the average processing time consumed, which are depicted in Figure 1 for one representative node. It can be seen that for each given QoR level, the total processing time is a linear function of the underlying workload. Besides, it is also easy to see that $T_i(q_i)$ is an increasing function - larger q_i means higher QoR, but also requires increased processing time $T_i(q_i)$ per unit workload. In this paper, we profile popular applications in mobile edge computing, including object recognition and online recommendation, to measure the tradeoff function $T_i(q_i)$ used in our optimization. Task profiling and execution time estimation can be assisted by techniques available in [39], [46].

Modeling the Response Time. The end-to-end response time for client applications on node i consists of data transfer time, processing time and a constant overhead τ_i. The τ_i captures the round trip time (RTT) between client device and node i at the network edge, as well as the setup time for computing the required tasks on node i, e.g., loading data into memory, task initialization and launching. Let D_i be the required size of input (and output) data chunk that needs to be sent to nodes (and mobile device) for a unit workload, B_i the network speed available for data transfer between client device and node i. The total data transfer time for workload of size w_i is then $w_i D_i / B_i$. A request with total workload W is completed only if all the tasks of sizes $w_1, \ldots, w_N$ are processed by individual nodes and the required results are aggregated by the mobile device. Therefore, the overall workload processing latency T is defined by the maximum response time of all N nodes:

$$T = \max_{\forall i=1,\ldots,N} \left[\frac{w_i D_i}{B_i} + w_i T_i(q_i) + \tau_i \right], \quad (3)$$

where the response time consists of data transfer time $w_i D_i / B_i$, task processing time $w_i T_i(q_i)$, and a constant overhead τ_i.

Modeling the Energy Consumption. We consider energy optimization from an edge client perspective and focus on the data transmission energy consumed by the client device for offloading workload and receiving results. Thus, we focus on energy saving for edge clients and leave system-level energy optimization, pricing issues [52], and incentive mechanisms [27] to future work. Based on prior study [61], for cellular networks, when transmission data size and rate are larger than certain thresholds, the cellular network interface's power consumption can be estimated by a constant that depends only on the phone model and network operator. For Wi-Fi, the power consumption during data transmission only depends on the channel rate and packet rate. Although the channel rate and packet rate may vary, for a single batch of workload in edge computing, of which the data transmission normally finishes in milliseconds or seconds, we can assume that the channel rate and the transmission protocol remain stable and are known to our optimization. Further, data transmission for a single batch of workload normally lasts for a duration which is smaller than network interface switch period [17]. We can simply use P_i to denote the average transmission power of the active network interface being used on the client mobile device. The total energy consumption is then determined by P_i and active time for transmission to each edge node:

$$E = \sum_{i=1}^{N} \frac{P_i w_i D_i}{B_i}. \quad (4)$$

Formulating the MobiQoR Optimization. Our goal is to jointly minimize mobile energy consumption and response time. We consider an objective function $E + \alpha T$, which combines the two objectives through a tradeoff factor (or weight) α, a commonly-used method for multi-objective optimization problems in networking [35]. The weight α reflects clients' preferences towards the relative importance of energy consumption and latency. It offers a control knob to be tuned by edge providers or edge clients to adjust the optimization for different applications and scenarios. In particular, we can increase α for latency sensitive applications where faster response is more critical than energy saving. A smaller α should be chosen for green computation where energy reduction is the main objective. We will investigate different application scenarios and the impact of α values in Section VII.

We formulate the MobiQoR optimization to minimize total energy consumption and response time under QoR and workload constraints in mobile edge computing. The optimization must determine two sets of closely coupled decision variables: (i) assigning QoR level q_i separately to each edge to achieve the optimal tradeoff between computation quality and processing speed, and (ii) load-balancing among different nodes with respect to their new processing speed under QoR relaxation. The MobiQoR optimization problem is formulated as follows:

$$\begin{aligned}
\min \quad & E + \alpha T & (5)\\
\text{s.t.} \quad & E = \sum_{i=1}^{N} \frac{P_i w_i D_i}{B_i}, & (6)\\
& T = \max_{\forall i=1,\ldots,N} \left[\frac{w_i D_i}{B_i} + w_i T_i(q_i) + \tau_i \right], & (7)\\
& \sum_{i=1}^{N} w_i = W, & (8)\\
& \sum_{i=1}^{N} \frac{w_i}{W} q_i \geq Q, & (9)\\
& 0 \leq q_i \leq 1, \ \forall i = 1, \ldots, N & (10)\\
\text{var.} \quad & \{q_i, w_i, \forall i = 1, \ldots, N\}. & (11)
\end{aligned}$$

IV. Our Proposed Solution

It is obvious that MobiQoR optimization is non-convex since constraint (7) contains a non-convex component $w_i T_i(q_i)$. While it can be numerically computed using solvers such as the genetic algorithm [6], [10], we propose an efficient algorithm to solve the optimization when the tradeoff function $T_i(q_i)$ is approximated by a linear function and the QoR optimization can be cast into linear programming (LP).

We consider a linear approximate tradeoff function $T_i(q_i) = \alpha_i q_i + \beta_i$ for some constants $\alpha_i, \beta_i \ \forall i$. While such an approximation may be difficult for the entire range of QoR $q_i \in [0, 1]$, we can identify different regions of interest on the QoR-speed tradeoff curve and use piecewise linear functions to provide an accurate estimate for different portions of $T_i(q_i)$. For example, our later evaluation results in Section VII-B show that when considering a feasible range of QoR level $q_i \in [0.89, \ 1]$, the linear approximation can be very accurate (Seen in Figure 4). Our main result to recast the QoR optimization into a LP program is stated in the following theorem.

Theorem 1: When $T_i(q_i) = \alpha_i q_i + \beta_i$ for some constants $\alpha_i, \beta_i \ \forall i$, the QoR optimization is equivalent to the following LP:

$$\begin{aligned}
\min \quad & \sum_{i=1}^{N} (P_i D_i / B_i) w_i + \alpha T & (12)\\
\text{s.t.} \quad & (D_i / B_i) w_i + \alpha_i y_i + \beta_i w_i + \tau_i \leq T, \ \forall i & (13)\\
& \sum_{i=1}^{N} w_i = W, & (14)\\
& \sum_{i=1}^{N} y_i \geq QW, & (15)\\
& 0 \leq y_i \leq w_i, \ \forall i & (16)\\
\text{var.} \quad & \{y_i, w_i, T\} & (17)
\end{aligned}$$

where $y_i = q_i w_i$ is an auxiliary variable, representing the quality-workload product of node i.

Proof 1: Plugging $y_i = q_i w_i$ into the QoR optimization (5), it is easy to see that constraints (15) and (16) are equivalent to (9) and (10), respectively. Next, due to the minimization over T, we can show that at optimum, at least one of inequalities in (13) must be satisfied with equality (otherwise, a smaller T is feasible and will further reduce the objective). This implies that T must be equal to the highest latency on the left-hand-side, which is equivalent to the $\max$ over i in (7). Thus, the two optimization problems, (5) and (12), are equivalent.

Theorem 1 shows that the QoR optimization can be solved via LP using a new variable, quality-workload product y_i. Let w_i^* and y_i^* be the optimal solution of (12). The optimal QoR level for each node is given by $q_i^* = y_i^*/w_i^*$. Furthermore, when only discrete QoR levels are allowed, we can first solve a relaxed, continuous problem (with continuous QoR levels) and then quantize the result to map optimal q_i^* to discrete levels, satisfying the constraints in the MobiQoR optimization. Using the analysis of LP algorithms in [37], we obtain the complexity of the proposed algorithm:

Remark 1: To solve the MobiQoR optimization via the equivalent LP, we need to consider an optimization over $2N+1$ variables (i.e., $w_i, y_i, T \ \forall i$), which has a complexity of $O[\sqrt{2N+1} \cdot \log(1/\epsilon)]$ for a given tolerance ϵ using the well-known Homogeneous Self-Dual Algorithm [34].

V. System Design

MobiQoR includes 5 key modules: Workload Manager, Optimization Engine, Decision History Database, Energy Meter, and Workload History Database. Figure 2 shows the major modules and workflow.

The *Workload Manager* plays the coordinator role between mobile apps and our framework. The workload generated by the applications on mobile devices are first submitted to Workload Manager ①. Upon receiving the workload, the Workload Manager performs the following set of tasks: (i) pre-processes workload by organizing the data that can be later submitted to other edge nodes, (ii) handles the app-specific mapping for the required QoR level (defined by Equation (1)), (iii) records important timestamps such as the workload arrival time, transmission finished time, and result receiving time.

To make workload offloading decisions, the Workload Manager first queries the *Decision History Database* to leverage any similarity in offloading decisions made previously ②. If there exists a history entry that matches both current workload size and offloading environment (e.g., network condition and numbers of different types of available edge nodes), the decision is directly used and the following steps (③, ④, and ⑤) are skipped. Otherwise, the Workload Manager requests an optimal decision from *Optimization Engine* ③, which will query *Workload History Database* ④ for the statistics about past offloaded workload, such as energy consumed for data transmission, and processing time taken by different types of edge nodes for various QoR levels. This step is to estimate the parameters needed for solving the QoR optimization problem

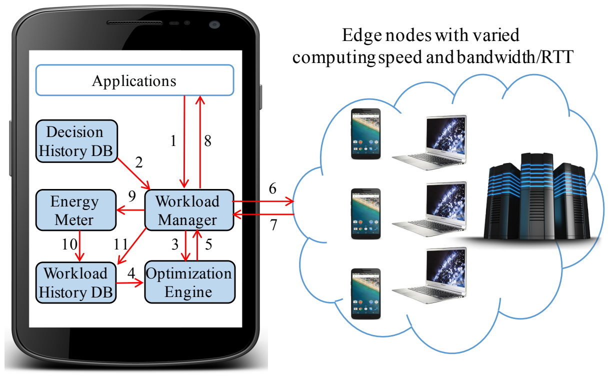

Fig. 2. MobiQoR architecture overview and workflow

in (5). By solving the problem, Optimization Engine determines (i) the assignment of workload, and (ii) the selection of QoR level for each edge node, thereby, jointly optimizing energy and latency. The optimal decisions are fed back to Workload Manager ⑤, which will dispatch the workload to different edge nodes with selected QoR levels ⑥. Once the computation results are sent back to Workload Manager ⑦, they will be aggregated and made available to the applications ⑧. The aggregated QoR level and timestamps for the workload will be calculated and stored in Workload History Database ⑪ for future references, along with the transmission energy measured by software-based *Energy Meter* ⑩.

Our MobiQoR system incorporates various types of devices in a distributed mobile edge environment, ranging from remote powerful workstations to nearby smart devices with less computing capacities. Two tradeoffs exist in the QoR optimization: (i) for a selected edge node, processing a certain workload with higher QoR level generally requires more resources or computing iterations, thus generating longer processing time; (ii) submitting the workload to remote edge nodes can introduce larger data transmission latency and energy consumption, however, remote edge nodes with high computing capacities are potential to reduce the time consumed for workload processing, thus reducing the overall time latency before the results become available to the applications.

VI. Implementation

Parse [4] is an open source mobile back-end framework, which consists of Parse back-end and Parse mobile SDK. Developers can write customized code, and deploy on Parse back-end to process certain workload. Parse mobile SDK is used to configure the mobile apps to transmit data to Parse back-end, call remote functions and receive processed results. Natively, for a given mobile app, only one Parse back-end instance is supported and all requests from mobile side are processed by the same instance.

We implement a prototype of our design of the QoR-aware computing offloading system based on Parse. We first modify the Parse mobile SDK to support multiple workload offloading destinations. The mobile SDK is incorporated in the Workload Manager module of MobiQoR, and the workload

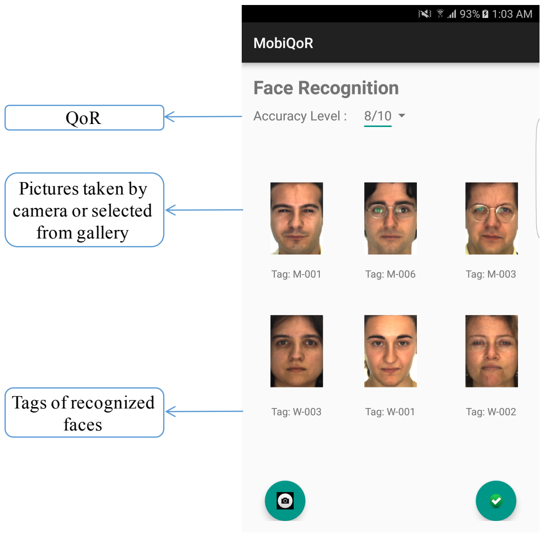

Fig. 3. Our demo app using mobile edge computing for face recognition

offloading tasks are implemented using the modified Parse mobile SDK. For each application, we implement the workload processing code in JavaScript on Parse back-end to coordinate the application's program. An instance of the Parse back-end is deployed on each of our edge nodes. These workload processing functions can be triggered from other edge nodes. The details of applications we select for evaluations will be discussed in details later in Section VII-A.

The Energy Meter module uses regression for energy estimation, which is similar to the approach adopted in [61]. The energy-related parameters are tuned using training data obtained from real measurements by external Power Monitor [3]. Based on the tuned parameters and network interface state transfer patterns for different networks, the Energy Meter estimates and records the power traces that consist of timestamps and power. Since the Workload Manager records the timestamps for data transmissions and result computations, the Energy Meter utilizes these to measure the energy consumed for data transmission of each batch of workload. The measurements are stored in the History Database, and made available to the Optimization Engine.

VII. Evaluation

A. Experiment Preparation

*1) **Evaluation App and Dataset Selection**:* We use two representative applications for evaluation: face recognition that computes a tag for each face image, and movie recommendation to predict users' rating for movie entries. For evaluation purpose, we design a simple UI for each application, through which workload is selected and computed results are displayed. Figure 3 is a screenshot of the face recognition application. A batch of images (representing total workload W) are taken by the camera or selected from photo gallery. Once a user clicks the confirmation button, the images will be offloaded and processed at edge nodes selected by the MobiQoR's Optimization Engine, which will also optimize the QoR levels at the edge nodes jointly. The results will be shown below the corresponding images on the UI when available.

TABLE I
Setup of edge nodes, to which workload can be offloaded

Node index	1	2	3	4
Type	workstation	laptop	workstation	laptop
OS	CentOS	CentOS	CentOS	OS X
Bandwidth to smartphone (Kbps)	751.0	1450.9	3065.4	3559.6
CPU base frequency (GHz)	3.4	2.4	3.4	2.9
CPU max turbo frequency (GHz)	3.9	3.4	3.9	3.2

Eigenface-based face recognition is a popular approach in the domain of human face recognition [57], in which each image is represented by a linear combination of eigenvectors of the whole vector space with different weights. For face recognition, the decision is made by referring the distances between weights of the new image and those of the training sets. With a smaller number of eigenfaces to represent a face image, the computation complexity is reduced while the percentage of correctly identified faces also drops and vice versa. The number of eigenfaces selected can be used to trade QoR for computing speed. The test images for face recognition application are collected from [36].

We implement the movie recommendation application using item-based collaborative filtering algorithm [50]. By discovering the pairwise similarity of movies, this method predicts the rating of user-movie pairs according to the ratings of similar movies. The number of referred user-movie pairs when predicting a new score can be tuned to achieve varied QoR-speed tradeoff. A larger number of referred user-movie pairs results in smaller prediction error (measured by Normalized Mean Absolute Error) as well as longer computation time. The test data for movie recommendation application is from MovieLens [23].

*2) **Experiment Setup**:* We use a Samsung Galaxy Note 5 Android smartphone as the end client node that generates edge computing workload. Taking into account of the high degree of heterogeneity in edge environments [53], we select four different devices and configure them as edge nodes in our testbed. To further tune the level of heterogeneity, we adopt *Linux TC* [25] and *cpupower* [43] tools to change the computing speed and the bandwidth available at different edge nodes. The default parameters (except for experiments in Section VII-E) configured for these edge nodes are listed in Table I with CPU max turbo option enabled by default.

To minimize environment variation when comparing different groups of experiments, we leverage the Android automated testing framework *Espresso* [1] to iteratively submit batches of workload via the app's UI. The ground-truths for testing data are pre-configured for later evaluation. Using the *onView* [1] method, Espresso can automatically wait until all the results are returned before proceeding to next testing action. The com-

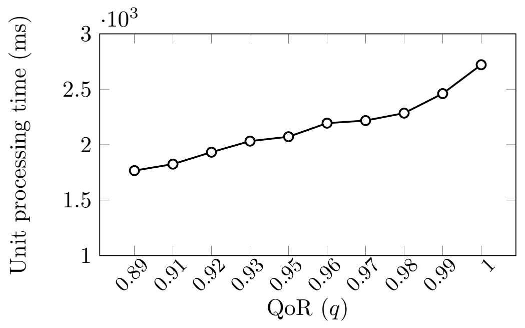

Fig. 4. The QoR-speed tradeoff for the face recognition app. As target QoR increases, the processing time of a unit workload increases. A linear approximation of the tradeoff has less than 11% error on average.

puted results collected via the UI by Espresso are compared against the ground-truths to calculate the achieved QoR. The end-to-end response time is measured by the time lapsed from the confirmation button is clicked to all results are received by the end device. We focus on energy consumption from an edge client perspective and develop MobiQoR's Energy Meter module at the end client device to measure the energy consumed for offloading workload and receiving results.

3) ***Baseline Selection****:* To evaluate the effectiveness of MobiQoR, we select and implement four existing workload offloading strategies from literature as our baselines:

Strategy I: *Even workload distribution with highest QoR*. The workload is evenly distributed to available nodes [29], and processed in a traditional non-approximated fashion, using the default implementation of the data processing program.

Strategy II: *Even workload distribution with reduced QoR*. The workload is evenly distributed [29], but different from Strategy I, this strategy allows QoR relaxation by the approximated computing techniques [44]. The QoR level assigned to each node is equal to an edge client's acceptable, target QoR.

Strategy III: *Energy-aware workload assignment*. In this strategy, the workload is partitioned and assigned to each available node such that the overall energy consumed for data and results transmission is minimized [16]. Edge nodes allowing more energy-friendly transmission are assigned more workload.

Strategy IV: *Latency-aware load balancing*. This strategy aims to minimize the overall response time for a batch of workload. It balances the workload on selected edge nodes in a way to minimize the maximum latency required to process all workload at edge nodes. Such strategy is widely formulated as minimax optimization [40], [47] in distributed workload processing problem.

B. Quantifying QoR-speed Tradeoff

We run a preliminary experiment to quantify the tradeoff between QoR and computing speed. We define 10 different QoR levels and adjust the corresponding parameters in the face recognition application. We test all the different QoR levels and for each level we run 20 batches of workload. The results are plotted in Figure 4. As the QoR (measured by Equation (1)) increases from 0.89 to 1, the processing time per image taken by this node increases from 1766 ms to 2721 ms. A linear approximation of the tradeoff curve can be accurate and our testing results show that the mean absolute percentage error introduced by linear approximation is only 10.8%.

C. Comparing MobiQoR against Baseline Strategies

The preliminary experiment in Figure 4 shows that there exists a tradeoff between the computing speed and the achieved QoR. In the current experiment, we compare the four baseline strategies and MobiQoR, in terms of the effectiveness to jointly minimize the energy consumption and response time. We plug in the four strategies discussed in Section VII-A and strategy of MobiQoR (Strategy M) into our system to consist of five experiment setups.

For each group of experiments, we use Espresso to submit the pre-configured batches of workload for the face recognition and movie recommendation apps. We consider a minimum QoR constraint from the applications that are expected by edge clients. The images for face recognition are submitted in the batch size of 20. For movie recommendation application, 1000 candidate movie entries are considered each time for a user to make a recommendation, in other words, each batch of movie recommendation workload needs to predict the scores given by a user for 1000 movie entries, so the workload size W is fixed as 1000. To take into account of different user preferences towards the relative significance of energy consumption E and overall latency T, we adjust the tradeoff factor α in the MobiQoR optimization (5) to capture a range of scenarios from "energy constrained" (α = 50) to "deadline sensitive"(α = 500) edge environments.

The results averaged overall multiple runs (40 runs for face recognition and 100 runs for movie recommendation) are shown in Figure 5, which demonstrates that MobiQoR consistently outperforms the other four strategies with respect to the optimization objective, i.e., the weighted sum of energy consumption and response time. In particular, the improvement of Strategy II (with relaxed QoR) over Strategy I (with highest QoR) illustrates the benefit received from relaxing QoR in the optimization. More importantly, MobiQoR achieves 13.4% to 77.0% improvement when comparing with the other four baseline strategies for face recognition application. For movie recommendation application, the improvement even reaches 189.3%, and is at least 43.0%.

Breakdown of the Optimized Offloading Decisions: To provide insights into MobiQoR's superior performance, we give in-depth comparisons of the decisions made by different strategies. We show a breakdown of the workload and QoR assignment for each edge node, under different strategies. Because of space limitation, we only present that for face recognition application as Table II, in which the three different α values correspond to the three scenarios in Figure 5. Here q_i is the QoR value selected for edge node i and w_i is the amount of workload assigned to node i. We can see that

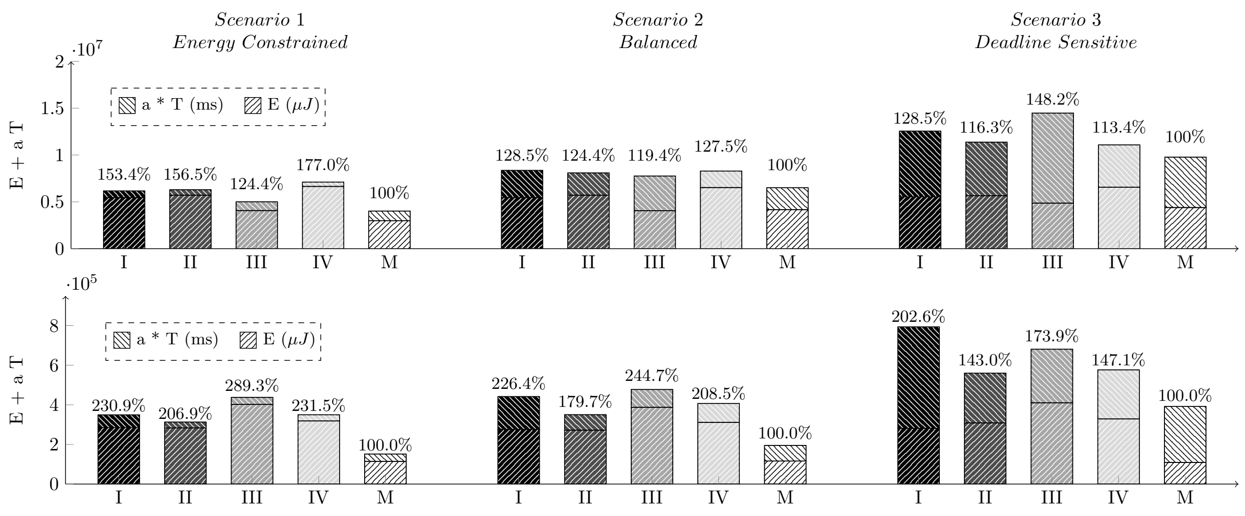

Fig. 5. Weighted sum of response time (patterned with ▧) and energy consumption (patterned with ▨) of 5 strategies when required QoR q is 0.96 for face recognition application (above) and 0.97 for movie recommendation application (below). The $E + \alpha T$ achieved by other strategies are normalized by that of MobiQoR (Strategy M).

TABLE II
BREAKDOWN OF WORKLOAD AND SELECTED QOR FOR EACH EDGE NODE USING DIFFERENT STRATEGIES UNDER VARIOUS SCENARIOS.

Strategy	α	w_1	q_1	w_2	q_2	w_3	q_3	w_4	q_4
I	any	5	1	5	1	5	1	5	1
II	any	5	0.96	5	0.96	5	0.96	5	0.96
III	any	2	0.96	4	0.96	6	0.96	8	0.96
IV	any	6	0.96	8	0.96	3	0.96	3	0.96
M	50	0	NA	0	NA	10	1	10	0.91
	200	0	NA	9	1	6	0.92	5	0.92
	500	0	NA	10	0.97	6	0.92	4	0.98

our algorithm (Strategy M) achieves the overall improvement of energy saving and latency reduction shown in Figure 5 by jointly optimizing the workload assignment and selecting different QoR values for available edge nodes.

We note that the difference between Strategy I and Strategy II shows the benefit obtained via approximated computing by relaxing the QoR constraint from 1 to 0.96. Further, comparing Strategy II to IV with Strategy M illustrates the improvement due to the joint optimization of energy and latency that we proposed in MobiQoR.

D. Evaluating Effects of Weight Factor between Energy and Latency

As shown in Figure 5, MobiQoR is able to outperform all other strategies in different scenarios that are modeled by tuning the weight factor α. In reality, an edge client or operator may want to understand the entire range of latency-energy tradeoff that is enabled by MobiQoR framework. Toward this end, we find the optimal frontier of the latency-energy tradeoff curve by varying the value of the tradeoff factor α and solving the corresponding MobiQoR optimization. The results are shown in Figure 6. It characterizes the best response time and energy consumption (tradeoff) that can be achieved

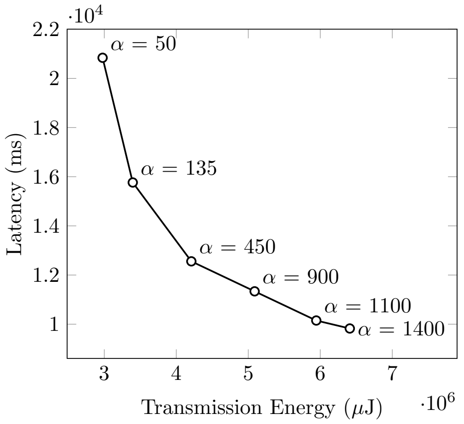

Fig. 6. Using different α values, the measured optimal transmission energy and overall latency achieved for face recognition application by MobiQoR.

when relaxing the QoR constraint to $Q = 0.96$ in mobile edge computing – any further latency improvement beyond this curve is impossible without causing higher energy consumption. Quantifying the optimal frontier of this tradeoff is extremely valuable to edge operators and clients, who can not only derive useful insights on setting the appropriate α value based on application scenarios and user preferences, but also impose arbitrary pricing/cost models on top of this energy-latency tradeoff to determine the optimal operating point.

E. Study of MobiQoR's Sensitivity to Edge Heterogeneity

Heterogeneity is a key feature of edge computing, especially in terms of network bandwidths and computing capacities of different edge nodes. To evaluate the impact of heterogeneity levels on MobiQoR optimization, we use *Linux TC* [25] and *cpupower* [43] tools to change the default setups for the edge

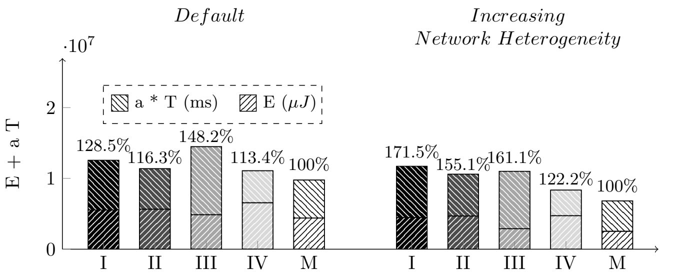

Fig. 7. Impact of different network conditions for the achieved weighted sum of response time (patterned with ▧) and energy consumption (patterned with ▨) of 5 strategies when required QoR q is 0.96 for face recognition application. The $E + \alpha T$ achieved by other strategies are normalized by that of MobiQoR (Strategy M).

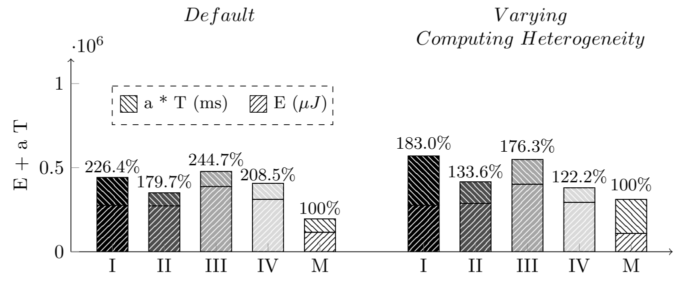

Fig. 8. Impact of varying computing capacities for the achieved weighted sum of response time (patterned with ▧) and energy consumption (patterned with ▨) of 5 strategies when required QoR q is 0.95 for movie recommendation application. The $E + \alpha T$ achieved by other strategies are normalized by that of MobiQoR (Strategy M).

nodes listed in Table I, and compare the results with the previous setup.

We first investigate the history traces of experiments conducted in Section VII-C using the default setup listed in Table I. The traces reveal that for the same selected QoR level q, the unit processing time taken by the four edge nodes sorted in ascending order are: $T_2(q) < T_1(q) \approx T_3(q) < T_4(q)$, where node 1 and node 3 are very close, and node 2 is the fastest one. Next, we configure two new groups of experiment setups based on the default one in Table I by either changing the network bandwidth or the CPU frequency:

Increasing Network Heterogeneity: We keep all other settings in Table I, but change the network bandwidths of edge nodes to 3484.8, 780.8, 3512.0, 4144.0 Kbps, respectively. In particular, we pick the one with fastest processing speed (i.e., node 2), and intentionally decrease its bandwidth to a smaller value. Then we perform the same 40 batches of face recognition experiments conducted in Section VII-C. The comparison of the five strategies is depicted in Figure 7 along with the experiment results using the default setup. As seen from the figure, after increasing network heterogeneity by changing the available bandwidth, our MobiQoR strategy achieves higher (relative) improvements over the other 4 strategies, demonstrating its ability to exploit an increasing level of network heterogeneity.

Varying Computing Heterogeneity: Similar to the previous experiment, we keep all other setups in Table I, but disable the default CPU turbo option, and set the CPU frequency of the four edge nodes to 2, 2.4, 3.4, 3.2 GHz, respectively. In this way, we make the CPU processing speed more consistent with the bandwidth setup in Table I, i.e., the fastest node is with the largest bandwidth and vice versa. We rerun the same 100 batches of movie recommendation experiments in Section VII-C. The comparison of the five strategies is depicted in Figure 8 along with the experiment results using the default setup. Even with decreased heterogeneity, MobiQoR performs 22.2% to 83.0% better than all other 4 strategies.

The above groups of experiments not only show that MobiQoR is able to achieve significant latency-energy improvement at different heterogeneity levels in the edge computing environment. More importantly, its superiority over other strategies is amplified as the computing environment becomes more heterogeneous. In terms of the heterogeneity property of edge computing, it is generally true that remote nodes are more likely to be equipped with more computing power, but smaller bandwidth [53]. MobiQoR's capability to jointly optimize the energy and response time makes it a perfect candidate to exploit such unique property of an edge environment.

VIII. Conclusion

The enlarging gap between contemporary mobile devices' resource constraints and the ever increasing user demands for ultra-low latency calls for rethinking mobile task offloading and distributed processing mechanisms. Inspired by the fact that increasing number of edge applications, such as media processing and machine learning, can tolerate some levels of quality loss, we propose a new optimization horizon, Quality-of-Result (QoR), and present MobiQoR for jointly optimizing the service latency and the mobile energy consumption under given QoR constraints. The proposed MobiQoR is implemented on real mobile devices. Using representative face recognition and movie recommendation applications, evaluations with real-world datasets show that MobiQoR outperforms existing strategies by up to 77.0% for face recognition and 189.3% for movie recommendation. More interestingly, the heterogeneity of edge computing environments further amplifies the benefits of MobiQoR over existing QoR-oblivious policies.

References

[1] Android testing tool - Espresso. https://developer.android.com/training/testing/ui-testing/espresso-testing.html.

[2] Firebase server. https://www.firebase.com/.

[3] Monsoon powermeter. https://www.msoon.com/LabEquipment/PowerMonitor/.

[4] Parse open source bankend. https://www.parse.com/.

[5] IDC futurescape: Worldwide internet of things 2016 predictions. *Available at: https://www.idc.com/research/viewtoc.jsp?containerId=259835*, 2015.

[6] O. Babaoglu, H. Meling, and A. Montresor. Anthill: A framework for the development of agent-based peer-to-peer systems. In *ICDCS*. IEEE, 2002.

[7] M. V. Barbera, S. Kosta, A. Mei, and J. Stefa. To offload or not to offload? the bandwidth and energy costs of mobile cloud computing. In *INFOCOM*. IEEE, 2013.

[8] Y. Bernier. Latency compensating methods in client/server in-game protocol design and optimization. In *Game Developers Conference*, 2011.

[9] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli. Fog computing and its role in the internet of things. In *MCC workshop*. ACM, 2012.

[10] R. Cheng, M. Gen, and Y. Tsujimura. A tutorial survey of job-shop scheduling problems using genetic algorithms I. Representation. *Computers & industrial engineering*, 30(4):983–997, 1996.

[11] M. Chiang. Fog networking: An overview on research opportunities. *arXiv preprint arXiv:1601.00835*, 2016.

[12] V. K. Chippa, S. Venkataramani, S. T. Chakradhar, K. Roy, and A. Raghunathan. Approximate computing: An integrated hardware approach. In *ASILOMAR*. IEEE, 2013.

[13] D. Chu, N. D. Lane, T. T.-T. Lai, C. Pang, X. Meng, Q. Guo, F. Li, and F. Zhao. Balancing energy, latency and accuracy for mobile sensor data classification. In *SenSys*. ACM, 2011.

[14] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti. Clonecloud: elastic execution between mobile device and cloud. In *EuroSys*. ACM, 2011.

[15] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl. MAUI: making smartphones last longer with code offload. In *MobiSys*. ACM, 2010.

[16] S. Cui, A. J. Goldsmith, and A. Bahai. Energy-efficiency of MIMO and cooperative MIMO techniques in sensor networks. *IEEE Journal on selected areas in communications*, 22(6):1089–1098, 2004.

[17] H. Falaki, R. Mahajan, S. Kandula, D. Lymberopoulos, R. Govindan, and D. Estrin. Diversity in smartphone usage. In *MobiSys*. ACM, 2010.

[18] J. Flinn, S. Park, and M. Satyanarayanan. Balancing performance, energy, and quality in pervasive computing. In *ICDCS*. IEEE, 2002.

[19] P. Garcia Lopez, A. Montresor, D. Epema, A. Datta, T. Higashino, A. Iamnitchi, M. Barcellos, P. Felber, and E. Riviere. Edge-centric computing: Vision and challenges. *ACM SIGCOMM Computer Communication Review*, 45(5):37–42, 2015.

[20] M. S. Gordon, D. A. Jamshidi, S. Mahlke, Z. M. Mao, and X. Chen. COMET: code offload by migrating execution transparently. In *OSDI*. USENIX, 2012.

[21] X. Gu, K. Nahrstedt, A. Messer, I. Greenberg, and D. Milojicic. Adaptive offloading for pervasive computing. *IEEE Pervasive Computing*, 3(3):66–73, 2004.

[22] K. Ha, Z. Chen, W. Hu, W. Richter, P. Pillai, and M. Satyanarayanan. Towards wearable cognitive assistance. In *MobiSys*. ACM, 2014.

[23] F. M. Harper and J. A. Konstan. The movielens datasets: History and context. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, 5(4):19, 2016.

[24] H. Hoffmann. Jouleguard: energy guarantees for approximate applications. In *SOSP*. ACM, 2015.

[25] B. Hubert, T. Graf, G. Maxwell, R. van Mook, M. van Oosterhout, P. Schroeder, J. Spaans, and P. Larroy. Linux advanced routing & traffic control. In *Ottawa Linux Symposium*, 2002.

[26] K. R. Jackson, L. Ramakrishnan, K. Muriki, S. Canon, S. Cholia, J. Shalf, H. J. Wasserman, and N. J. Wright. Performance analysis of high performance computing applications on the amazon web services cloud. In *CloudCom*. IEEE, 2010.

[27] C. Joe-Wong, Y. Im, K. Shin, and S. Ha. A performance analysis of incentive mechanisms for cooperative computing. In *ICDCS*. IEEE, 2016.

[28] R. Kemp, N. Palmer, T. Kielmann, and H. Bal. Cuckoo: a computation offloading framework for smartphones. In *MobiCASE*. Springer, 2010.

[29] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang. Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading. In *INFOCOM*. IEEE, 2012.

[30] D. Kovachev, T. Yu, and R. Klamma. Adaptive computation offloading from mobile devices into the cloud. In *ISPA*. IEEE, 2012.

[31] K. Kumar and Y.-H. Lu. Cloud computing for mobile users: Can offloading computation save energy? *Computer*, (4):51–56, 2010.

[32] Y.-W. Kwon and E. Tilevich. Energy-efficient and fault-tolerant distributed mobile execution. In *ICDCS*. IEEE, 2012.

[33] A. Li, X. Yang, S. Kandula, and M. Zhang. CloudCmp: comparing public cloud providers. In *IMC*. ACM, 2010.

[34] D. G. Luenberger and Y. Ye. *Linear and Nonlinear Programming*. Springer Science & Business Media, 2008.

[35] R. T. Marler and J. S. Arora. The weighted sum method for multi-objective optimization: new insights. *Structural and Multidisciplinary Optimization*, 41(6):853–862, 2010.

[36] A. M. Martinez. The AR face database. *CVC Technical Report*, 24, 1998.

[37] N. Megiddo. Linear programming in linear time when the dimension is fixed. *Journal of the ACM*, 31(1):114–127, 1984.

[38] A. P. Miettinen and J. K. Nurminen. Energy efficiency of mobile clients in cloud computing. In *HotCloud*. USENIX, 2010.

[39] B. C. Mochocki, K. Lahiri, S. Cadambi, and X. S. Hu. Signature-based workload estimation for mobile 3D graphics. In *DAC*. ACM, 2006.

[40] R. R. Nadakuditi and M. Liu. Latency-optimizing file splitting for transmission over a large multi-hop network. In *ITA workshop*. IEEE, 2011.

[41] V. Namboodiri and T. Ghose. To cloud or not to cloud: A mobile device perspective on energy consumption of applications. In *WoWMoM*. IEEE, 2012.

[42] S. Ou, K. Yang, and J. Zhang. An effective offloading middleware for pervasive services on mobile devices. *Pervasive and Mobile Computing*, 3(4):362–385, 2007.

[43] V. Pallipadi and A. Starikovskiy. The ondemand governor. In *Proceedings of the Linux Symposium*, 2006.

[44] P. Pandey and D. Pompili. MobiDiC: Exploiting the untapped potential of mobile distributed computing via approximation. In *PerCom*. IEEE, 2016.

[45] J. Park, J. H. Choi, and K. Roy. Dynamic bit-width adaptation in DCT: an approach to trade off image quality and computation energy. *IEEE transactions on very large scale integration (VLSI) systems*, 18(5):787–793, 2010.

[46] P. Puschner and A. Burns. Guest editorial: A review of worst-case execution-time analysis. *Real-Time Systems*, 18(2):115–128, 2000.

[47] S. Ramakrishnan, I.-H. Cho, and L. A. Dunning. A close look at task assignment in distributed systems. In *INFOCOM*. IEEE, 1991.

[48] P. Rong and M. Pedram. Extending the lifetime of a network of battery-powered mobile devices by remote processing: a markovian decision-based approach. In *DAC*. ACM, 2003.

[49] J. San Miguel, J. Albericio, A. Moshovos, and N. E. Jerger. Doppelganger: A cache for approximate computing. In *MICRO*. ACM, 2015.

[50] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. In *WWW*. ACM, 2001.

[51] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies. The case for vm-based cloudlets in mobile computing. *IEEE pervasive Computing*, 8(4):14–23, 2009.

[52] S. Sen, C. Joe-Wong, S. Ha, and M. Chiang. *Smart Data Pricing*. John Wiley & Sons, 2014.

[53] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu. Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, 2016.

[54] S. Sidiroglou-Douskos, S. Misailovic, H. Hoffmann, and M. Rinard. Managing performance vs. accuracy trade-offs with loop perforation. In *FSE*. ACM, 2011.

[55] A. Sinha, A. Wang, and A. P. Chandrakasan. Algorithmic transforms for efficient energy scalable computation. In *ISLPED*. ACM, 2000.

[56] T. Soyata, R. Muraleedharan, C. Funai, M. Kwon, and W. Heinzelman. Cloud-Vision: Real-time face recognition using a mobile-cloudlet-cloud acceleration architecture. In *ISCC*. IEEE, 2012.

[57] M. A. Turk and A. P. Pentland. Face recognition using eigenfaces. In *CVPR*. IEEE, 1991.

[58] S. Venkataramani, V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan. Quality programmable vector processors for approximate computing. In *MICRO*. ACM, 2013.

[59] L. Xiang, S. Ye, Y. Feng, B. Li, and B. Li. Ready, set, go: Coalesced offloading from mobile devices to the cloud. In *INFOCOM*. IEEE, 2014.

[60] S. Yi, Z. Hao, Z. Qin, and Q. Li. Fog computing: Platform and applications. In *HotWeb workshop*. IEEE, 2015.

[61] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, and L. Yang. Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In *CODES+ISSS*. ACM, 2010.

[62] W. Zhang, Y. Wen, and D. O. Wu. Energy-efficient scheduling policy for collaborative execution in mobile cloud computing. In *INFOCOM*. IEEE, 2013.

[63] Y. Zhang, G. Huang, X. Liu, W. Zhang, H. Mei, and S. Yang. Refactoring android java code for on-demand computation offloading. In *ACM SIGPLAN Notices*, volume 47, pages 233–248. ACM, 2012.