Pervasive and Mobile Computing 57 (2019) 49–63

Contents lists available at ScienceDirect

Pervasive and Mobile Computing

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

Adaptable mobile cloud computing environment with code transfer based on machine learning

P. Nawrocki *, B. Sniezynski, H. Slojewski

AGH University of Science and Technology, Faculty of Computer Science, Electronics and Telecommunications, Department of Computer Science, al. A. Mickiewicza 30, 30-059 Krakow, Poland

HIGHLIGHTS

- Analysis of the solutions in the area of optimizing task execution in MCC.
- The offloading strategy is learned autonomously and online on mobile devices.
- The implementation uses a cross-platform technology.
- The experiments demonstrate the effectiveness of the solution developed.

ARTICLE INFO

Article history:
Received 19 December 2018
Received in revised form 29 April 2019
Accepted 5 May 2019
Available online 16 May 2019

Keywords:
Machine learning
Mobile cloud computing
Code offloading
Energy optimization

ABSTRACT

The growing importance of mobile devices has caused the need to develop solutions (such as Mobile Cloud Computing) that make it possible to optimize their operation. The main objective of this article is to investigate the possibilities for using machine learning and the code offloading mechanism in the Mobile Cloud Computing concept which may enable the operation of services to be optimized, among others, on mobile devices. We have proposed a formal model of the solution and created its prototype implementation. The adaptable Mobile Cloud Computing environment developed has been implemented using a cross-platform technology for designing Internet applications (Ionic 2). This technology enables hybrid applications to be built with code transfer that run on different operating systems (such as Android, iOS or Windows), which decreases the amount of work required from developers, as the same code is executed on a mobile device and in the cloud. It also makes this solution significantly more universal. The experiments conducted with respect to our solution showed its effectiveness, especially in the case of services which require complex calculations. Test results (for the Face Recognition and Optical Character Recognition services) showed that service execution time and energy consumption decreased significantly during the performance of tasks on a mobile device.

1. Introduction

In recent years, we have seen the increased importance of solutions that make it possible to optimize the operation of mobile devices. Among these solutions, the Mobile Cloud Computing (MCC) concept plays an important role, which enables tasks/services to be sent from the mobile device to the cloud and the result to be returned to the mobile device. This makes it possible, among other things, to reduce the time of task/service performance and to reduce the energy consumption of mobile devices.

* Corresponding author.
E-mail address: piotr.nawrocki@agh.edu.pl (P. Nawrocki).

https://doi.org/10.1016/j.pmcj.2019.05.001
1574-1192/

In addition to the work carried out by the authors [1–3], there are only a few studies on the use of machine learning algorithms in the MCC environment (such as MALMOS [4] and IC-Cloud [5]). The platforms developed as part of these studies only make it possible to optimize the task/service execution time and do not take into account energy aspects related to power consumption on the mobile device.

Novelty of the studies carried out by the authors and described in this paper consists in the following: we have developed an original offloading mechanism based on resource consumption models created autonomously and online on a mobile device, in which we take into account the time and energy aspects of the optimization of operation of mobile devices in the MCC and use a different machine learning algorithm. In addition, an extension of the MCC concept to the use of a PC placed in the local network as an additional node to perform computations is proposed. This idea is consistent with the new approach of Mobile Edge Computing (MEC) [6], which assumes running mobile applications and performing the related processing tasks closer to the cellular customer. In our research, we have adapted the idea of MEC (which was designed for cellular base stations) to the MCC environment. In the case where the user uses his/her local network, this enables the costs of using the cloud to be reduced by remotely performing tasks/services on a local PC. At the same time, in order to make the solution developed more universal, the authors developed a code offloading mechanism. In most existing MCC solutions, service implementations are separate on the mobile device and in the cloud. The use of the code offloading (code transfer) technique [7–9] makes it possible to develop a single service implementation whose code can be sent to a cloud or to another location (for example, a PC). This significantly simplifies the implementation of such a solution and reduces its implementation and management costs. What is more, we propose a novel technique to measure battery consumption, which replaces PowerTutor [10], a very good but outdated solution.

This paper is structured as follows: Section 2 contains a description of related work. Section 3 is concerned with employing the learning module in the process of optimizing the service and describes in detail the adaptable MCC environment. Section 4 describes performance evaluation and Section 5 contains the conclusion.

2. Related work

The analysis of solutions in the area of optimizing task execution in Mobile Cloud Computing can be presented in various contexts. It appears very important that this analysis should cover the taxonomy of such solutions, issues of code offloading and remote procedure calls, the possibility of using machine learning and aspects of energy consumption by mobile devices.

2.1. Taxonomies of MCC solutions

The review and classification of MCC solutions have became the subject of many studies. Results of such studies have been presented in the form of taxonomies. One of the papers [11] proposes a division into three categories: mono-objective optimization, bi-objective optimization and multi-objective optimization systems. The most common approach appears to be bi-objective optimization (of execution time and energy consumption).

Another distinction concerns decision-making methods: history based, parametric (actual conditions) or stochastic [12]. In this context, machine learning can be considered to be a combination of two or even all three abovementioned categories. The first category applies, since machine learning models can be trained with historical data. The second category applies to online learning. In addition, if a probabilistic classifier has been used, then the model falls into the third category.

There are also studies of existing MCC solutions which present a comparison and evaluation of available solutions from the point of view of different parameters. One of them is [8] which compares several selected MCC environments (such as MAUI, CloneCloud, ThinkAir, COMET and EMCO) in terms of code offloading strategies and the mobile/cloud perspective.

Important issues in the context of access to services using mobile devices and the MCC concept are their security and the ability to use them in a situation where the demand for cloud resources is high. These issues are discussed, among others, in [13] where the authors propose the concept of a service broker (MUBaaS) which, through a distributed Sub Proxy, is able to provide services to mobile devices using various cloud resources. In this approach, all services are located in the cloud and the broker appropriately distributes customer requests (from mobile devices) for services and provides them in a secure manner using different cloud resources. This allows for load balancing as regards access to cloud resources. However, there are some disadvantages of this solution. First of all, there is only one instance of the cloud access gateway and thus its failure or overloading may result in the inability to access services. In addition, there are various solutions (for example [14]) which allow for such load balancing when using cloud resources which makes it possible to reduce the costs of using such resources. In the solution presented, just a simple FIFO queue was used which does not allow such optimization of cloud resource consumption. At the same time, the possibility of running services locally (on a mobile device) is not considered at all. Such functionality may be beneficial depending on the context in which the mobile device operates, involving, inter alia, energy consumption. The use of wireless communication modules always results in high energy consumption [15] and it is not always advantageous to provide services remotely (in the cloud). The solution proposed by us takes into account the operating context of the mobile device and makes it possible, using machine learning algorithms, to determine the optimal location to run the service from the point of view of speed and/or energy consumption.

2.2. Code offloading and remote procedure calls

Important terms in the context of MCC are code offloading and Remote Procedure Calls (RPC). Running complex applications/services on mobile devices is a considerable challenge, among others due to hardware limitations (CPU, memory) and energy consumption. Therefore, the MCC concept provides for the possibility of sending the application/service code or part thereof to the cloud for efficient execution. An important article dealing with code offloading is [16]. In that article, the authors propose an adaptive offloading system which includes a distributed offloading platform and an offloading inference engine. Their tests demonstrate that the adaptive offloading system developed can effectively relieve memory constraints for mobile devices with far lower overhead than other common approaches. However, the research presented in the article does not take into account the operating context of the mobile device, including the aspects related to energy consumption during code offloading. The authors plan future studies in this area. Another issue is the use of Java Virtual Machine (JVM) in the developed system. The operating systems currently used on mobile devices (such as iOS or Android) do not allow the easy implementation of the solution developed by the authors of the article discussed.

In order to optimize the offloading process, research is being carried out on how to automatically choose methods which should be offloaded to a cloud/edge. This issue was presented, among others, in [17] where the authors propose an innovative task selection algorithm which autonomously parses a real-world mobile Android app and retrieves the list of methods that are suitable for offloading to a cloud. However, the current state of work does not yet allow for measuring latency and energy savings when applying the solution proposed. The authors plan to do this and conduct such tests in the future. This area of research appears very important to us, which is why we want to include it in our further work.

The second important concept is the ability to remotely perform services/ applications. Environments such as Cuckoo [9], MAUI [18] or ULOOF [19] take advantage of Remote Procedure Calls (RPC). Cuckoo integrates well-known Android platform tools, i.e. activities and services. Services are used to define computationally complex tasks, while activities describe user interactions. Activities obtain access to services throughout proxy objects. Classes of proxy objects are generated from interfaces defined with Interface Definition Language (IDL). Such classes are then enriched with decision logic by Cuckoo. This logic determines whether a task should be executed locally or remotely. The decision is made using historical data and simple heuristics. Remote calls are preferred only if there is a connection to a server. The local service implementation was left to a programmer, whereas the remote implementation is partially generated by Cuckoo. The authors' aim was to automate the process of building mobile applications and reduce the frequency of repetitive development activities.

On the MAUI platform, which is designed for the .Net Framework, the authors have provided the `[Remotable]` annotation which should be placed before a method signature to mark computationally intensive methods. All methods implementing functionalities other than IO or GUI can be annotated. Such a solution does not increase the size of source code and does not involve excessive effort from a programmer. MAUI has a 0–1 integer linear programming solver that finds a program partitioning strategy which minimizes energy consumption and execution time based on a profiler, which gathers and analyzes runtime data. The offloading decision is taken by a decision engine at runtime. Proxies running both on a mobile device and on server side serialize transferred data. Energy consumption, execution time, resource demands and network conditions like throughput, latency or packet loss are analyzed. To build a smartphone's energy profile, a hardware power meter is used. The decision engine runs on the server side to save energy. Furthermore, the server side is equipped with a coordinator which handles authentication and allocates resources to particular tasks.

Another solution using annotations is AgileRabbit [20] which employs the concept of feedback to decide whether to offload a specific task under specific network conditions. The results show that this solution can speed up processing as much as three times for a speech recognition application and reduce energy consumption by up to 50% with less than 2% overhead in CPU load for a smartwatch and a mobile phone. However, this solution has some limitations. First of all, it operates in just one Android environment and is not cross-platform. In addition, the authors use an offloading decision algorithm using feedback data with a set objective, i.e. minimizing the task completion time or minimizing the total power consumption of smartwatches and mobile devices. In order to improve the operation of this algorithm, it would be advisable to use machine learning. Also, the method of energy consumption measurement used by the authors is not accurate although other methods could be used, for example Power Profiles.

Typically, remote invocation methods (described above) are used in the MCC environment. However, in heterogeneous environments with high dynamics of change, such methods may turn out to be insufficient. That is why a different solution is needed such as the code transfer [21] used by the authors of this paper. One of the environments used for code transfer is Javascript. However, this is most often considered in the context of web pages rather than mobile applications [22]. Currently, some research is being carried out on the use of JavaScript in the context of MCC. One of these solutions is MACO (Manageable Application Code Offloading) [23] where the author optimizes execution time (and energy consumption) of the mobile application through code offloading. However, this solution lacks any complex selection mechanisms (e.g. using machine learning) to decide when part of the application is to be sent from the mobile device to be executed. Simple rules are set for offloading the computational part of the application and there is a preference for executing the elements related to the user interface locally. More complex tests using a mobile phone are missing as well, only tests using a tablet were carried out. The author also did not provide any information on the operating system of the mobile device tested and on the energy measurement method. Most of the research conducted does not take into account energy consumption (just the execution time of the application / service) and does not leverage machine learning mechanisms, such as the MOJA system [24] and the solution for offloading mobile Web applications (MobWeb) [25].

Some modern MCC (and Mobile Edge Computing) systems use rule-based strategies to make decisions about offloading. In [26], the authors propose a multiplatform offloading system, which allows task offloading for mobile applications using the MCC and cloudlet concepts. The decision when to offload tasks is made by the Dynamic Decision System which uses fixed rules. Another article [27] proposes a multi-agent-based flexible IoT edge computing system which makes it possible to optimize operation and reduce the amount of energy consumed. That system also uses a rule-based engine with a fixed rule set.

In our research, we have taken into account energy aspects, developed machine learning to optimize offloading and tested the system in a real-life environment. We have also compared it to a different rule-based offloading system. Also importantly, our solution makes innovative use of TypeScript annotations. None of the previous surveys take TypeScript into account.

2.3. Machine learning

A modern approach to MCC environments is the adaptation of machine learning to optimize execution time and energy consumption. To make the solution efficient, online learning performed autonomously on mobile devices should be applied. It is a problem similar to that which appears in agent-based systems that learn. Surveys [28,29] show that reinforcement learning is usually applied in such conditions. However, our research demonstrates that with a larger parameter space, supervised learning is better than reinforcement learning [30]. It is a case in our domain where the decision should take into account the type of the task, its main parameters and the context. There are also other approaches to this problem in the literature and they are discussed below.

An interesting solution is the EMCO environment [7]. It is a framework designed to simplify the development of hybrid applications by delivering automated tooling and reusable components. This framework locally utilizes data and information gathered by the cloud to make decisions on whether to offload a task or not and how to execute the task. Its decision engine estimates whether it is cost-effective to offload the tasks to the cloud using a fuzzy logic reasoner.

Systems combining MCC and machine learning usually contain an offloading scheduler with offline or online (incremental) training. Offline type schedulers are more straightforward and do not have performance overheads. However, they have to be trained with a previously collected data set, which may hinder adaptation to new conditions. On the other hand, in online schedulers, a model is trained with new data which are collected during runtime. For training, it requires the entire dataset each time (which is less efficient) or a single training example if the algorithm supports it.

An example environment that benefits from an online scheduler is MALMOS [4]. It works alternately in two phases: online training and runtime scheduling. In the first phase, tasks are executed locally and remotely. Information from both executions is passed to the training module to improve the classifier. If classification accuracy exceeds a given threshold, the system moves to the scheduling phase. The length of this phase depends on the number of classification errors made during the training. In this phase, tasks are performed in the location indicated by the classifier. During system evaluation, the following algorithms were compared: Instance-based learning, perceptron and Naive Bayes classifier. One of the IBL algorithms is K-nearest neighbors. A verification of several classification methods is justified because of the algorithms' lack of versatility. Their accuracy may depend on the type and amount of training data.

In our earlier work, we applied reinforcement learning [3] and supervised learning [2]. An agent running in a mobile environment obtains information about the operation to execute and its context. Subsequently, it consider one or more criteria like reducing energy consumption, time or cost to select the execution location: in the cloud or locally.

2.4. Energy consumption

An important issue related to the MCC concept is taking into account the context of mobile device operation, including its energy consumption. The ability to take into account the energy aspects of mobile device operation makes it possible to select the best strategy for using local or cloud resources during the starting of services. One of the articles dealing with the aspect of energy consumption by a mobile device is [31]. The authors of that article investigate the energy consumption patterns of multiple encryption–decryption methods (such as RSA, DES, and AES) in securing electronic health records. They argue that some methods tend to consume more energy on mobile devices than others. However, the article does not contain any descriptions of the test environment in which the experiments were carried out or descriptions of mobile device energy consumption measurements. This is a very important issue due to the difficulty of accurately measuring (or estimating) energy consumption in mobile devices.

A tool for estimating the energy consumption of a mobile device which has gained significant popularity is PowerTutor [10]. It depends on three basic energy characteristics of subassemblies referred to as Dream, Passion and Sapphire. These are not in line with the characteristics of state-of-the-art mobile devices, however, causing significant inaccuracies. Probably the most accurate tool currently available is Battery Historian.[1] It cannot conduct real-time measurements, however; it only performs a static analysis of the reports generated from a device.

Another article which deals with the aspects of energy consumption when running mobile applications is [15]. In the article, the authors propose eprof, the first fine-grained energy profiler for smartphone apps which makes it possible to

[1] Official source code repository of Battery Historian – https://github.com/google/battery-historian.

Table 1
Summary of different MCC solutions.

	MAUI	MALMOS	MOJA	MACO	MobWeb	Agile Rabbit	Authors' solution
Cross-platform ready	No	No	Yes	Yes	Yes	No	Yes
Machine learning	No	Yes	No	No	No	No	Yes
Code offloading	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Energy measurements	Yes	No	No	Yes	No	Yes	Yes
JavaScript (JS)/TypeScript (TS)	No	No	JS	JS	JS	No	TS

measure the energy consumption of individual device components (such as the CPU, WiFi, GPS and 3G/4G) for a given mobile application. However, the solution developed uses built-in Android energy measurement mechanisms which may not ensure adequate measurement accuracy. At the same time, the solution developed presents energy consumption for individual components, allowing developers to determine which application elements are worth optimizing. However, it does not introduce any automated online methods to optimize the operation of mobile applications/services such as those proposed in our solution.

Preliminary work [32] describes contemporary methods of estimating energy consumption for Android devices: Delta-B method, reading discharge current from a file and Power Profiles.

The Delta-B method hinges on calculating the difference in battery charge level at the beginning and at the end of the measurement. Unfortunately, this method has insufficient (1%) accuracy. Reading discharge current from the file gives better results because the results are refreshed every second. However, this accuracy is still insufficient in the context of a single invocation. The file mentioned may not be available on all devices and its location depends on the model. In addition, these methods do not allow the extraction of estimation results for specific components like the screen, CPU or communication modules. The last method is based on so-called Power Profiles which are provided by a producer of a device. It makes it possible to determine the current required to run a particular component in a given state.

An alternative method of energy measurement is the physical measurement of battery energy consumption [33]. This is the most accurate measurement method but it requires physical access to the inside of the device and the appropriate measuring equipment. Therefore, due to the nature of this measurement method, it is not commonly used.

2.5. Summary

Existing MCC solutions focus on the Android ecosystem. They are usually highly complicated and unsupported. Most of them do not use machine learning. These are the reasons why introducing them is often not cost-effective. There are some solutions which use JavaScript, but none of them have advanced machine learning capabilities and most of them do not take into account the power consumption of the mobile device when offloading tasks. A great advantage of our MCC environment is that it can be run regardless of the operating system and takes into account the application/service execution time and the power consumption of the mobile device in the optimization process. Complexity is reduced by annotating methods for possible cloud execution instead of preparing separate versions of code for the mobile device and the server. To increase adaptability, online machine learning was used. The use of TypeScript (instead of JavaScript) also allowed for static typing which gives certainty as to object type and prevents potential errors regarding data types during programming. To the best of our knowledge, our solution is the first to use TypeScript for offloading in MCC. Table 1 summarizes the features of individual MCC solutions.

The Power Profiles method is increasingly used in various studies to estimate energy consumption of mobile devices with the Android system. In [34], the authors use in the system they developed PETrA (*Power Estimation Tool for Android*) information from the power profile file for measuring the energy consumption of Android apps. The results of PETrA system tests (using the LG Nexus series device) have shown that the use of Power Profiles allows for an accurate estimation of energy consumption in mobile devices and the average estimation error achieved using PETrA is only 4% compared to the actual value calculated using the MONSOON hardware toolkit.[2] Currently, Power Profiles are among the most accurate software energy consumption measurement methods in mobile devices with the Android system. The design of modern mobile devices, and in particular the fact that batteries are not user-replaceable (but rather are permanently attached to the device) significantly hinders hardware measurement of energy consumption using, for example, the MONSOON hardware toolkit.

3. Adaptable MCC environment

In this section, we present our solution for the adaptable task allocation problem. We begin with a formal model and then we present its implementation.

The adaptable MCC task execution framework based on machine learning receives task $t \in T$ in context $c \in C$. It returns location for execution $l \in L$ determined by strategy $s : T \times C \rightarrow L$. The task is executed in this location and the

[2] MONSOON Power Monitor – http://www.msoon.com.

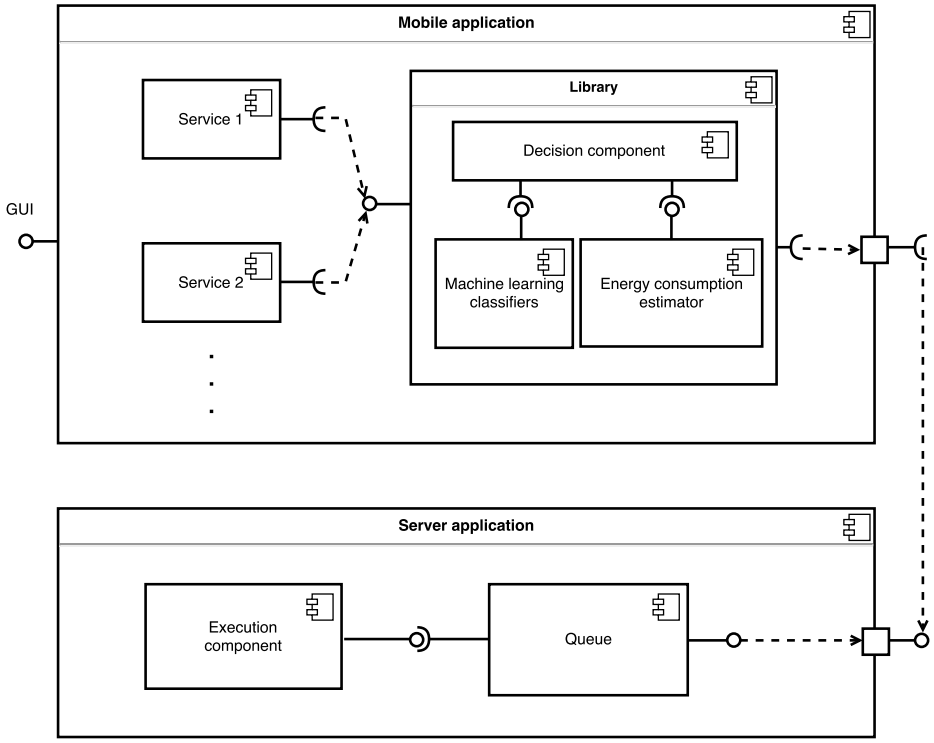

Fig. 1. Solution architecture.

framework observes usage of resources R during the execution (e.g. computation time or battery usage) represented by a set of usage values $\{u_r\}_{r\in R}$. These data form the example:

$$e = (t, c, l, \{u_r\}). \tag{1}$$

Examples are collected in training data set D. Machine learning algorithm m is used to process D and generate models M predicting R that are used by strategy s to determine the location. Every model $M \ni M_r : T \times C \times L \to \mathcal{R}$ is used to predict result r for a given task and context under condition that it is executed in a given location. The framework may be modeled as the following tuple:

$$F = (T, C, L, s, R, D, m, M). \tag{2}$$

The strategy s may be modeled as a function which selects a location with the minimal predicted cost $p \in \mathcal{R}$. The predicted cost is a weighted sum of predicted results:

$$p(t, c, l) = \sum_{r\in R} w_r * M_r(t, c, l), \tag{3}$$

where $w_r \in [0, 1]$ are weights of resources chosen by the user. Therefore s may be defined as:

$$s(t, c) = \arg\min_{l\in L} p(t, c, l). \tag{4}$$

As a result, s depends on these weights and current models M.

Fig. 1 shows the architecture of the proposed solution. The system consists of two main modules: a mobile application and server back-end. The former provides the user functionality of an adaptable MCC task execution framework. The latter processes tasks that were delegated to a PC or a cloud. Mobile application details are described in Section 3.1. The server module is described in Section 3.2.

3.1. Mobile application

The main part of the framework is a library for code offloading. It can be used to adapt task allocation in MCC in order to efficiently provide services that require complex computations such as face recognition, OCR or multimedia processing. Such services are composed of one or more methods. The method which is the entrance point for other methods has to be marked with the `@Offloadable` annotation. In fact, annotated methods are replaced with their enriched versions.

Table 2
Machine learning attributes.

Attributes	Values
Execution location	−1: local (mobile device); 0: PC computer; 1: cloud server
Hashed source code	[−1, 1]
Size of arguments	in kilobytes
Current hour (divided by 23)	[0, 1]
Active transmission module	0: WiFi; 1: mobile network (3G)

Such proxies additionally measure execution time and energy consumption as well. It is worth noting that annotations are strongly supported by IDEs and if they are set up improperly the application will not start. The decision component invokes the computational task in the location where it should be completed in the shortest time with the lowest energy consumption. Resource consumption during execution is predicted by a machine learning classifier. K-nearest neighbors, a feedforward neural network and CART decision tree algorithms are supported but other classification or regression methods may also be applied. Machine learning is executed on the mobile device. Therefore, it consumes energy and time. However, learning is executed rarely (when a sufficient number of new examples are stored) and may be delayed to a moment when the mobile device is not used and connected to a charger. Moreover, our results from other experiments [3] show that this process is not resource demanding.

In the case of remote execution, the method's source code and arguments are serialized (to the JSON[3] format) and then offloaded to a server using the HTTP protocol. Source code of the offloaded method can be transferred one-to-one and simply executed because client-side scripting programming language (TypeScript) has been chosen for implementation. Such serialized source code then becomes placed in a queue on the server side. In fact, it is JavaScript to which TypeScript is transpiled. JavaScript is effortless to transfer due to the fact that it is interpreted, not compiled. We assume that offloaded functions must not have external dependencies other than arguments. It simplifies the code of our framework and encourages the user to apply more functional style of programming. In the future this assumption may be weakened. What is more, `eval` function should be used with care and should not be used to evaluate complex expressions (this approach is also encouraged by the programming community).

The authors provided a library for code offloading which consists of a power consumption estimator based on Power Profiles, a decision component which implements strategy S and two machine learning classifiers M_{et}, M_{ec} predicting execution time and energy consumption, respectively. The M_{ec} model provides approximate energy consumption values of individual components of the mobile device. Measurements start when the method is invoked and end when a response is received. The decision component runs the algorithm presented in Fig. 2, collecting experience online. It is represented by examples stored in training data set D. Currently, to make predictions (lines 5–6) and to create example e (line 30), t, c, l, r are described by attributes shown in Table 2. Execution location represents where the task is executed (-1: mobile device, 0: PC, 1: cloud server). Hashed source code is a unique number identifying the code to be executed. Argument size represents how big the arguments used in task execution are. Current hour represents time and allows daily changes in bandwidth and cloud responsiveness to be taken into account. Active transmission module represents the type of network connection. Additional attributes (more detailed task description, battery level, current mobile CPU voltage, geographical location, etc.) may also be taken into account. Training happens after each method execution.

Both classifiers employ the same machine learning algorithm. They predict discretized resource consumption values. Classes with appropriate intervals are presented in Tables 3 and 4. The intervals have been optimized by consistently shrinking or expanding them to achieve the best results. If the results deteriorated, changes were made in the other direction. The classes are used to calculate the predicted execution cost p, introduced in Eq. (3):

$$p(t, c, l) = w_{et} * M_{et}(t, c, l) + w_{ec} * M_{ec}(t, c, l) \tag{5}$$

Where:
w_{et} - execution time prediction's weight,
w_{ec} - energy consumption prediction's weight,
$M_{et}(t, c, l)$ - discretized execution time prediction, $M_{et}(t, c, l) \in \{-2, -1, 0, 1, 2\}$,
$M_{ec}(t, c, l)$ - discretized energy consumption prediction, $M_{ec}(t, c, l) \in \{-2, -1, 0, 1, 2\}$.

The location with the lowest cost is selected. Draws are resolved in favor of the nearest device starting from local execution and ending with the server.

3.2. Server application

The server implemented by the authors contains a set of FIFO queues. The first queue is used to enqueue computational tasks. Other queues store results. The execution component is the only part of the server which was created by the authors from scratch. It dequeues tasks, which are then deserialized and run. The result can be obtained from the specified

[3] JavaScript Object Notation – https://www.json.org.

```
perform an initial round of 10 random executions and train the classifier (one-time)
get task t ∈ T and observe context c ∈ C
D ← ∅
minimal cost ← 2
execution location ← −1
foreach l ∈ {−1, 0, 1} do
    predict the execution time for t, c, l
    predict the energy consumption for t, c, l
    p ← calculate the predicted cost
    if p <minimal cost then
        minimal cost ← p
        execution location ← l
    end
end
start execution time and energy consumption measurements
switch execution location do
    case −1
        {u_r} ← run on the mobile device
    end
    case 0
        task ← serialize the method and arguments
        insert the task into the PC's queue
        {u_r} ← dequeue the result
    end
    case 1
        task ← serialize the method and arguments
        insert the task into the cloud server's queue
        {u_r} ← dequeue the result
    end
endsw
finish execution time and energy consumption measurements
add e = (t, c, l, {u_r}) to training data D
train the classifier
return result
```

Fig. 2. The MCC library's algorithm.

Table 3
Classes for execution time predictions.

Interval [ms]	Class
(0, 500)	−2
[500, 1000)	−1
[1000, 2000)	0
[2000, 5000)	1
[5000, +∞)	2

Table 4
Classes for energy consumption predictions.

Interval [mAh]	Class
(0, 0.025)	−2
[0.025, 0.05)	−1
[0.05, 0.1)	0
[0.1, 0.25)	1
[0.25, +∞)	2

queue by the library running on the mobile device. The execution component supports libraries which facilitate the implementation of services for mobile devices. When the component is started, sandboxes become created for each supported library. After a method has been deserialized, a similar sandbox is created for the method's execution. The script including all sandboxes is run in the current context. This script can contain callbacks which are the basis for programming in JavaScript.

Server instances may be running on a computer in a local area network or in the cloud. In the case of the cloud, the system should be prepared to process a number of requests simultaneously (in parallel). Such a solution has not been applied in the current implementation, however. Currently, tasks are executed sequentially with the next one being sent from the mobile device when the previous one has been finished. FIFO queues are added to take into account the overhead generated by queuing multiple tasks. Further scaling may be achieved in the future by multiplexing instances or load balancing. Cloud providers like Amazon provide *Platform as a Service* solutions such as load balancers, which go beyond the standard *Infrastructure as a Service* framework.

3.3. Energy consumption measurements

Due to the need for high measurement accuracy and mobility of devices, the authors implemented an energy consumption estimator based on Power Profiles (see Section 2.5). Knowing the state of an individual component of the mobile device and the time it spends in this state, it is possible to calculate its energy drain. Possible states are: inactive, idle, active. Discharge current for each state can be read from the *power_profile.xml* file. Access to this file can be obtained via the internal API of the Android operating system. On the basis of documentation,[4] the formula for total energy consumption Eq. (6) can be derived. It is the sum total of energy consumption for all components. For each component, its energy consumption is calculated by multiplying the current for its state and the duration of this state:

$$u_{ec} = \sum_{i \in C_{ec}} \sum_{cs \in S_i} f(i, cs) * t(i, cs) \tag{6}$$

Where:
C_{ec} – set of components for which energy consumption is calculated,
S_i – set of states of component i,
$f(i, cs)$ – function returning discharge current value for component i and its state cs,
$t(i, cs)$ – function returning the time spent by component i in state cs.

The time spent by a component in a given state can be read from the file system or using the API (for example *TrafficStats* class on Android). Data referring to CPUs are placed in public files */proc/[pid]/stat* where pid is the process identifier. According to Linux documentation,[5] key columns used for calculations are: user time, kernel time, child times and idle time.

4. Performance evaluation

The experiments were conducted using an LG Nexus 5 (2013) with a 4-core Krait 400 2.26 GHz CPU and 2 GB of RAM. The device is equipped with a 2300 mAh battery. As a computer in the local network, a Lenovo ThinkPad S440 (2013) was used. This laptop contains a 2-core low-voltage Intel Core i7-4500U processor clocked up to 3 GHz, SSD and 8 GB of RAM. At first, the tests were carried out on Amazon AWS EC 2 instances, but due to high fees, tests were continued on Digital Ocean's cloud. The cloud machine's specifications were as follows: 4 GB of RAM, 2 cores of an Intel hexacore CPU clocked to 3 GHz and an SSD. Local devices used a 4G Internet connection through a router with a 4G modem.

A mobile application was prepared for experimental purposes. It enables computational tasks to be run on specified devices or according to the classifier's decision. Three machine learning algorithms were tested: a decision tree, k-nearest neighbors (KNN) and a neural network. The CART decision tree[6] had the default gain function (Gini impurity) and the maximum depth of ten. The k-nearest neighbors method[7] was examined for k equal to three. The feedforward neural network[8] had one hidden layer with ten neurons and *tanh* activation function. It was trained using the back propagation algorithm with the learning rate parameter set to 0.01. Several algorithms were examined since they can exhibit different accuracies for a given problem.

The framework implemented was tested with two services: Face Recognition and Optical Character Recognition. The libraries used to provide these services are based on different algorithms, but share at least two characteristics: they are built with the same blocks such as loops and can be resource-intensive depending on the input parameters. Hence, the results can be generalized to other algorithms. Each experiment consisted of 30 series and each series began with the

4 Power Profiles documentation – https://source.android.com/devices/tech/power.
5 *proc* documentation – http://man7.org/linux/man-pages/man5/proc.5.html.
6 CART decision tree – https://github.com/mljs/decision-tree-cart.
7 KNear – https://github.com/NathanEpstein/KNear.
8 Feedforward NN – https://github.com/mljs/feedforward-neural-networks.

initial round of 10 random tasks (FR or OCR). This number of series made it possible to confirm statistical significance with high probability. The initial round resulted in more reasonable values within fewer rounds compared to randomized initial values. Subsequently, random pictures were selected as input (1 out of 10) and random locations were chosen for execution. In the case of face recognition, objects like face, eyes and mouth were random as well. Later on, six rounds consisting of two tasks each were run. The first task was FR and the second was OCR. Every task involved a different picture with a resolution ranging from 320×240 to 640×480. Pictures were repeated in subsequent series. Classifiers for predicting execution time and energy consumption were trained after the completion of each task in a selected location. Each task was run according to the classifier's prediction and then also in other locations. In the case of WiFi transmission there were three locations: the mobile device (local execution), the computer in the local network and the machine in the Digital Ocean cloud. During 3G transmission, computations were not performed on the local PC. This could be achieved by assigning a public IP address to the PC and ensuring that it was not behind NAT/PAT services. Both on the PC and in the cloud, one instance of the server was available as tests were executed synchronously.

Experiments were conducted for various weights assigned to the cost function Eq. (5). Energy consumption depends on execution time. In the case of local execution, the longer the calculations, the more energy the CPU consumes. In the case of PC or cloud execution, energy consumption is related to radio transmission and the longer it takes, the more energy is consumed. Therefore, there is a correlation between execution time and energy consumption. Thus the results for different weights are similar; however, there are certain differences. For 3G connection and weights $w_{et} = 0.3$ and $w_{ec} = 0.7$, the Decision tree learning algorithm, which gives good results, makes it possible to decrease energy consumption by 65% and execution time by 72% between the first two and last two rounds, while for weights $w_{et} = 0.7$ and $w_{ec} = 0.3$, energy consumption is reduced by 57% and execution time by 75%. As we can see, a higher weight results in reducing the corresponding resource more.

Due to lightweight machine learning models implemented in JavaScript the cost of predicting and updating classifiers turned out to be insignificant and impossible to measure using the methods available. On the other hand, the services implemented or their input data (pictures) could influence the results. Where only larger pictures were handled, all tasks were executed remotely. Therefore, results for different transmission types are shown. The input pictures present human faces or text. They come from the Internet and the authors' private repositories.

Before performing the relevant experiments, the authors checked whether the learning process itself would not significantly affect the results. The test results showed that with respect to both execution time and energy consumption, the results for different machine learning algorithms did not significantly affect experiment outcomes. The resulting execution time and energy consumption were very small and in fact difficult to detect given the accuracy of the measurement methods available. For example, during experiments, the use of machine learning algorithms consumed less than 0.08% of battery capacity. These results are consistent with the observations and experiments presented by the authors in an earlier paper [3]. Also importantly, classifiers may be trained less frequently than after each task execution. In experiments described in [3], the learning process was executed after processing each package of 24 tasks. It is also possible to postpone the learning process until the device is connected to the charger. As a result, the learning cost may be further reduced.

4.1. Results for WiFi transmission

In this section, results for WiFi transmission are presented and discussed. Charts presenting execution time, energy consumption and accuracy of classifiers are shown. Fig. 3 consists of two charts. The first of them illustrates the average total execution time (left bar) and the average total energy consumption (right bar) for the first two rounds, whereas the second presents them for the last two rounds. The results were collected for each of the three learning algorithms and also separately for fixed execution location in the absence of learning (charts on the right).

After the first two rounds, execution time and energy consumption started to decrease thanks to machine learning. Results from the first two rounds indicate low classifier accuracy and exhibit significant standard deviations. This was caused by the insufficient training of classifiers at that stage. Results improved over the next few rounds. The best results for the last two rounds were recorded for the neural network. The results obtained by the decision tree may be explained by the relatively simple structure of the model. The average neural network results from the last two rounds proved superior in terms of execution time as well as energy consumption compared to every fixed execution location (fixed locations are presented on the right half of the charts in Fig. 3). It is also evident that local execution for all tasks is completely inefficient. Machine learning-based executions were 10 s faster on average and consumed 0.2 mAh less than local ones. After recognizing 100 pictures the difference would be comparable to 1% of battery capacity.

Fig. 4 shows the accuracy of the Decision module. To determine the accuracy of a given algorithm, execution cost Eq. (5) was calculated, substituting real time execution and approximate energy consumption values. If correct location is selected, the system is awarded a point. If location choice is incorrect, no points are awarded. The correct location is one with the minimum cost and this was established because the task was executed in each location during the experiment. To calculate final accuracy, the score obtained was divided by the maximum score possible.

Accuracy from the first round refers to how well the classifier was trained after the initial round. Each classifier attained higher accuracy in the last round than in the first one. Accuracy drop was observed in the fifth round, which could have been caused by classifier overfitting. This is probably due to the likeness of input data, e.g. similar photo resolutions, in the

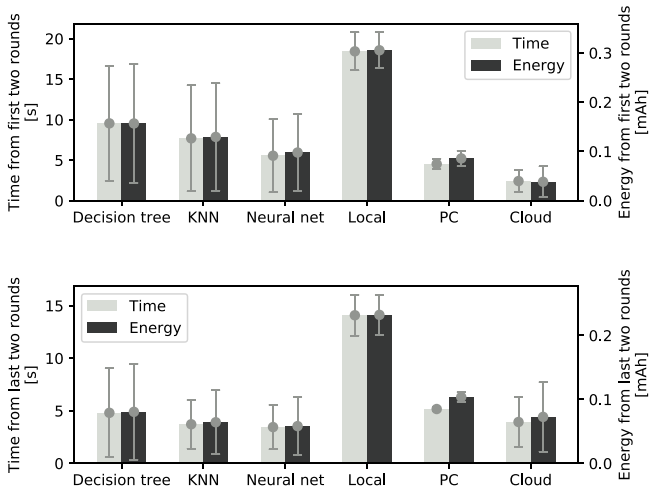

Fig. 3. Total execution time and energy consumption for WiFi transmission.

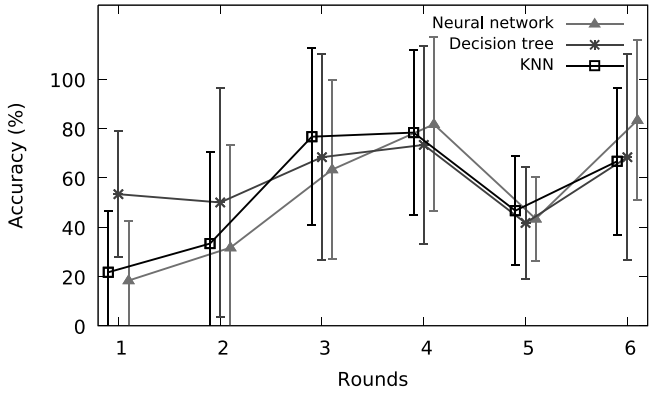

Fig. 4. Accuracy of the Decision module (Neural network, Decision tree, KNN) for WiFi transmission.

earlier phase of the experiment. The decision tree obtained the worst learning results as its accuracy oscillated around 50%. The highest score of about 75% was obtained by the neural network. The statistical significance of improvement between the first and last rounds (*imp*) was checked using the t-Student test. Results of the t-Student test (p-values) are: for Neural network: ≤ 0.0001 (which means that *imp* is statistically significant), for Decision tree: 0.0991 (which means that *imp* is not statistically significant) and for KNN: ≤ 0.0001 (which means that *imp* is statistically significant).

4.2. Results for 3G transmission

In this section, results for 3G transmission are presented. Charts similar to those in the previous section are shown. The only difference is that the computer in the local network was not available.

According to Fig. 5, in the first two rounds, the KNN classifier and the neural network achieved better results than the decision tree and fixed locations. At this stage, high standard deviations are visible. After six rounds, each classifier achieved shorter execution times and lower energy consumption than executions in fixed locations. Execution time in the case of classifiers was nearly four times (3 s per execution) shorter than for executions in the cloud only and up to ten times (13 s per execution) shorter than local executions. Energy consumption for classifiers was about four times (1.5 mAh per execution) lower than when executing in fixed locations. Again, the neural network proved to be the most effective. The negligible standard deviation in the case of this classifier is worth noting. This manner of transmitting data increased energy costs for remote executions. Although processing in the cloud is much faster for most tasks, their remote execution could become undesirable due to higher battery usage.

The chart presented in Fig. 6 is similar to Fig. 4. Higher classifier accuracy when using 3G is clear. Presumably this is related to simplified models with only two execution locations: local and the cloud. The classifiers improved their accuracy in each subsequent round. In the last round, the neural network achieved almost 100% accuracy. The rest obtained satisfactory results exceeding 75%. Statistical significance of the improvement between the first and last rounds (*imp*) was

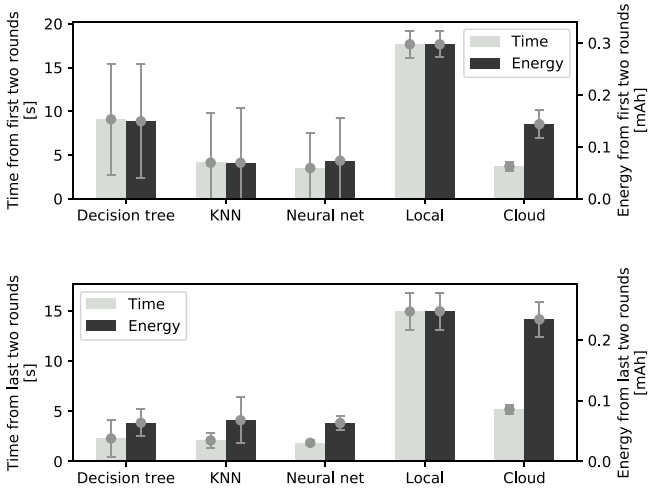

Fig. 5. Total execution time and energy consumption for 3G transmission.

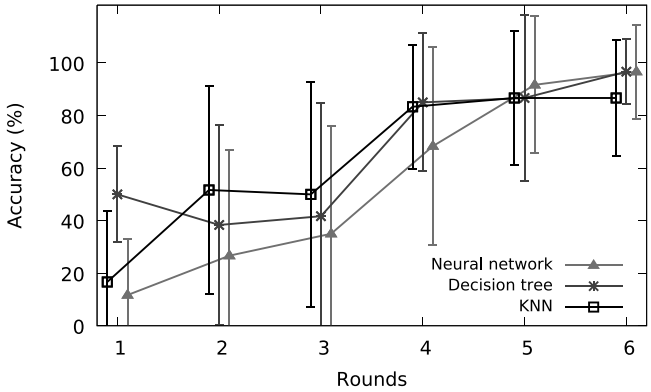

Fig. 6. Accuracy of the Decision module (Neural network, Decision tree, KNN) for 3G transmission.

checked using the t-Student test. Results of the t-Student test (p-values) for all classifiers (Neural network, Decision tree and KNN) are ≤ 0.0001, which means that *imp* was statistically significant.

4.3. Mixed connection type

A comparison of execution locations determined by classifiers for the tasks offloaded using mixed transmission technologies (3G and WiFi) is shown in Fig. 7. We have also applied rule-based and random strategies for comparison. The rule-based strategy is used in MCC systems [26,27] (see Section 2 for more information). In our tests, the rules were defined as follows:

- IF the image is less than or equal to 10 kB AND smaller than 76,800 pixels (320 $\times$ 240) THEN execute task locally;
- IF the image is larger than 10 kB OR greater than or equal to 76,800 pixels (320 $\times$ 240) AND connection type is WiFi THEN offload the task to the PC;
- IF the image is larger than 10 kB OR greater than or equal to 76,800 pixels (320 $\times$ 240) AND connection type is 3G THEN offload the task to the cloud.

The threshold values used in the rules (10 kB and 76,800) were selected manually to minimize execution time for the set of tasks in question. The second strategy selects a random execution location: local or the cloud. After the first two rounds, all the classifiers and the random strategy were better than the rule-based strategy, but the neural network yielded the best results. At the end of the experiment, after learning, all classifiers were able to achieve shorter execution times and lower energy consumption than both the rule-based and random strategies. The rule-based strategy yields better results than in the first two rounds. This is caused by the fact that the set of tasks randomly selected for these rounds was different. This example shows a significant advantage of the machine learning solution, which is able to adapt its

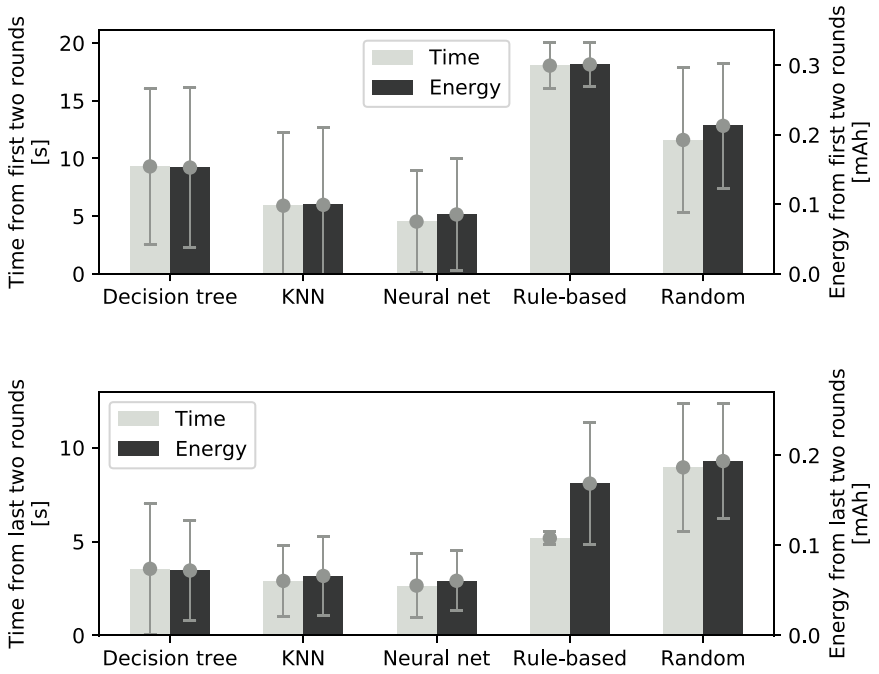

Fig. 7. Total execution time and energy consumption for mixed connection type and ML/rule-based/random strategy.

strategy to task characteristics and the context. The rule-based system may include rules that optimize its operation, but it is not able to respond to changes online.

5. Conclusions

The research conducted has demonstrated that the use of machine learning (within the MCC paradigm) together with the possibility of code offloading enabled the development of an environment which allows the operation of services to be optimized, among others, on mobile devices. This solution is not applicable to native applications. It was implemented using a cross-platform technology for designing Internet applications (*Ionic 2*). It enabled hybrid applications to be built which run on different operating systems (such as Android, iOS or Windows), which also makes the developed solution significantly more universal and allows less complex implementation because offloadable code is created only once. The source code of our solution is available in the following repository: https://github.com/Hoobie/ml-offloading.

The energy consumption estimator supports only Android but, in our opinion, can be extended to other platforms as well by implementing an equivalent to Power Profiles. According to our knowledge there is no Power Profiles on iOS, the second most popular mobile OS. However, because of smaller hardware variety, it is possible to create such equivalent for most popular iOS devices by measuring battery consumption in appropriate tests.

The experiments conducted on the solution developed showed that it is effective, especially for services which require complex calculations. Test results (for the implemented Face Recognition and Optical Character Recognition services) demonstrated that thanks to the use of the environment developed, service execution time and energy consumption decreased significantly during task performance on a mobile device. In order to compare the solution developed with other strategies (not involving machine learning) used in the MCC, a rule-based decision system and a system allowing for random selection of the execution location (local or the cloud) were implemented. Additional experiments were carried out in which both these strategies (without machine learning) were tested. The results of the experiments demonstrated the advantage of the ML solution over the rule-based and random strategies.

In further research, machine learning settings will be tuned and other algorithms will be compared (especially regression should be checked). We also plan to take into account other attributes related to the context and implement other types of tasks.

Acknowledgment

The research presented in this paper was supported by funds from the Polish Ministry of Science and Higher Education assigned to the AGH University of Science and Technology.

Declaration of competing interest

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

References

[1] P. Nawrocki, B. Sniezynski, J. Czyzewski, Learning agent for a service-oriented context-aware recommender system in heterogeneous environment, Comput. Inform. 35 (5) (2016) 1005–1026.
[2] P. Nawrocki, B. Sniezynski, Autonomous context-based service optimization in mobile cloud computing, J. Grid Comput. 15 (3) (2017) 343–356.
[3] P. Nawrocki, B. Sniezynski, Adaptive service management in mobile cloud computing by means of supervised and reinforcement learning, J. Netw. Syst. Manage. (2017) 1–22.
[4] H. Eom, R. Figueiredo, H. Cai, Y. Zhang, G. Huang, MALMOS: Machine learning-based mobile offloading scheduler with online training, in: 2015 3rd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering, 2015, pp. 51–60.
[5] C. Shi, P. Pandurangan, K. Ni, J. Yang, M. Ammar, M. Naik, E. Zegura, IC-cloud: Computation offloading to an intermittently-connected cloud, Tech. rep., Georgia Institute of Technology, 2013.
[6] S. Nunna, K. Ganesan, Mobile edge computing, in: C. Thuemmler, C. Bai (Eds.), Health 4.0: How Virtualization and Big Data are Revolutionizing Healthcare, Springer International Publishing, Cham, 2017, pp. 187–203.
[7] H. Flores, S. Srirama, Adaptive code offloading for mobile cloud applications: Exploiting fuzzy sets and evidence-based learning, in: Proceeding of ACM MCS 2013, ACM, NY, USA, 2013, pp. 9–16.
[8] H. Flores, P. Hui, S. Tarkoma, Y. Li, S. Srirama, R. Buyya, Mobile code offloading: from concept to practice and beyond, IEEE Commun. Mag. 53 (3) (2015) 80–88.
[9] R. Kemp, N. Palmer, T. Kielmann, H. Bal, Cuckoo: A computation offloading framework for smartphones, in: M. Gris, G. Yang (Eds.), Mobile Computing, Applications, and Services, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, pp. 59–79.
[10] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R.P. Dick, Z.M. Mao, L. Yang, Accurate online power estimation and automatic battery behavior based power model generation for smartphones, in: Proceedings of IEEE International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS) 2010, ACM, NY, USA, 2010, pp. 105–114.
[11] E. Ahmed, A. Gani, M. Sookhak, S.H.A. Hamid, F. Xia, Application optimization in mobile cloud computing: Motivation, taxonomies, and open challenges, J. Netw. Comput. Appl. 52 (2015) 52–68.
[12] N. Fernando, S.W. Loke, W. Rahayu, Mobile cloud computing: A survey, Future Gener. Comput. Syst. 29 (1) (2013) 84–106.
[13] R.K. Lomotey, R. Deters, MUBaaS: Mobile ubiquitous brokerage as a service, World Wide Web 19 (1) (2016) 5–19.
[14] P. Nawrocki, W. Reszelewski, Resource usage optimization in mobile cloud computing, Comput. Commun. 99 (2017) 1–12.
[15] A. Pathak, Y.C. Hu, M. Zhang, Where is the energy spent inside my app?: Fine grained energy accounting on smartphones with eprof, in: Proceedings of ACM EuroSys 2012, ACM, 2012, pp. 29–42.
[16] X. Gu, K. Nahrstedt, A. Messer, I. Greenberg, D. Milojicic, Adaptive offloading for pervasive computing, IEEE Pervasive Comput. 3 (3) (2004) 66–73.
[17] A. Zanni, S.Y. Yu, P. Bellavista, R. Langar, S. Secci, Automated selection of offloadable tasks for mobile computation offloading in edge computing, in: Proceedings of CNSM 2017, 2017, pp. 1–5.
[18] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, P. Bahl, MAUI: Making smartphones last longer with code offload, in: Proceedings of MobiSys 2010, ACM, 2010, pp. 49–62.
[19] J.L.D. Neto, S.-Y. Yu, D.F. Macedo, J.M.S. Nogueira, R. Langar, S. Secci, ULOOF: a user level online offloading framework for mobile edge computing, IEEE Trans. Mob. Comput. (2018).
[20] M. Yu, Y. Ma, X. Liu, G. Huang, X. Chen, AgileRabbit: A feedback-driven offloading middleware for smartwatch apps, in: Proceedings of the 9th Asia-Pacific Symposium on Internetware, Internetware'17, ACM, New York, NY, USA, 2017, pp. 6:1–6:10, http://dx.doi.org/10.1145/3131704.3131709.
[21] M. Kemppainen, Mobile Computation Offloading: a Context-driven Approach, 2011.
[22] J. Zhang, W. Liu, W. Zhao, X. Ma, H. Xu, X. Gong, C. Liu, H. Yu, A webpage offloading framework for smart devices, Mobile Netw. Appl. 23 (5) (2018) 1350–1363.
[23] S. Yang, Manageable granularity in mobile application code offloading for energy savings, 2012 IEEE International Conference on Green Computing and Communications, 2012, pp. 611–614, http://dx.doi.org/10.1109/GreenCom.2012.93.
[24] C. Xu, Y. Qiao, B. Lee, N. Murray, MOJA - Mobile offloading for JavaScript applications, in: 25th IET Irish Signals Systems Conference and China-Ireland International Conference on Information and Communications Technologies, ISSC 2014/CIICT 2014, 2014, pp. 59–63.
[25] M. Yu, G. Huang, X. Wang, Y. Zhang, X. Chen, JavaScript offloading for web applications in mobile-cloud computing, 2015 IEEE International Conference on Mobile Services, 2015, pp. 269–276.
[26] P.A. Rego, P.B. Costa, E.F. Coutinho, L.S. Rocha, F.A. Trinta, J.N. de Souza, Performing computation offloading on multiple platforms, Comput. Commun. 105 (C) (2017) 1–13.
[27] T. Ogino, S. Kitagami, T. Suganuma, N. Shiratori, A multi-agent based flexible IoT edge computing architecture harmonizing its control with cloud computing, Int. J. Netw. Comput. 8 (2) (2018) 218–239.
[28] L. Panait, S. Luke, Cooperative multi-agent learning: The state of the art, Auton. Agents Multi-Agent Syst. 11 (2005) 2005.
[29] P. Stone, M. Veloso, Multiagent systems: A survey from the machine learning perspective, Auton. Robots 8 (2000) 345–383.
[30] B. Sniezynski, A strategy learning model for autonomous agents based on classification, Int. J. Appl. Math. Comput. Sci. 25 (3) (2015) 471–482.
[31] J.C. Pry, R.K. Lomotey, Energy consumption cost analysis of mobile data encryption and decryption, in: Proceedings of IEEE International Conference on Mobile Services 2016, 2016, pp. 178–181.
[32] M. Josefiok, M. Schröder, A. Winter, An energy abstraction layer for mobile computing devices, Softwaretechnik-Trends 33 (2013).
[33] A. Schulman, T. Schmid, P. Dutta, N. Spring, Demo: Phone power monitoring with battor, in: ACM Mobicom, 2011.
[34] D. Di Nucci, F. Palomba, A. Prota, A. Panichella, A. Zaidman, A. De Lucia, Software-based energy profiling of Android apps: Simple, efficient and reliable? in: 2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER), 2017, pp. 103–114, http://dx.doi.org/10.1109/SANER.2017.7884613.

Piotr Nawrocki, Ph.D., is Assistant Professor in the Department of Computer Science at the AGH University of Science and Technology, Krakow, Poland. His research interests include distributed systems, computer networks, mobile systems, mobile cloud computing, Internet of Things and service-oriented architectures. He has participated in several EU research projects including MECCANO, 6WINIT, UniversAAL and national projects including IT-SOA and ISMOP. He is a member of the Polish Information Processing Society (PTI).

Bartlomiej Sniezynski received his Ph.D. degree in Computer Science in 2004 from the AGH University of Science and Technology in Krakow, Poland. In 2004, he worked as a Postdoctoral Fellow under the supervision of Professor R. S. Michalski at the Machine Learning and Inference Laboratory, George Mason University, Fairfax, VA, USA. Currently, he is Assistant Professor in the Department of Computer Science at AGH. His research interests include machine learning, multi-agent systems, and knowledge engineering.

Hubert Slojewski is a Computer Science graduate of the AGH University of Science and Technology in Krakow, Poland. His fields of interests include web technologies, machine learning, agent-based systems and MCC.