Computing and Informatics, Vol. 38, 2019, 790–816, doi: 10.31577/cai_2019_4_790

AGENT-BASED SYSTEM FOR MOBILE SERVICE ADAPTATION USING ONLINE MACHINE LEARNING AND MOBILE CLOUD COMPUTING PARADIGM

Piotr Nawrocki, Bartłomiej Śnieżyński

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
e-mail: {piotr.nawrocki, bartlomiej.sniezynski}@agh.edu.pl

Jakub Kołodziej

CRIF Sp. z o.o.
Lublańska 38
31-476 Krakow, Poland
e-mail: jakub.lukasz.kolodziej@gmail.com

Abstract. An important aspect of modern computer systems is their ability to adapt. This is particularly important in the context of the use of mobile devices, which have limited resources and are able to work longer and more efficiently through adaptation. One possibility for the adaptation of mobile service execution is the use of the Mobile Cloud Computing (MCC) paradigm, which allows such services to run in computational clouds and only return the result to the mobile device. At the same time, the importance of machine learning used to optimize various computer systems is increasing. The novel concept proposed by the authors extends the MCC paradigm to add the ability to run services on a PC (e.g. at home). The solution proposed utilizes agent-based concepts in order to create a system that operates in a heterogeneous environment. Machine learning algorithms are used to optimize the performance of mobile services online on mobile devices. This guarantees scalability and privacy. As a result, the solution makes it possible to reduce service execution time and power consumption by mobile devices.

In order to evaluate the proposed concept, an agent-based system for mobile service adaptation was implemented and experiments were performed. The solution developed demonstrates that extending the MCC paradigm with the simultaneous use of machine learning and agent-based concepts allows for the effective adaptation and optimization of mobile services.

Keywords: Agent-based system, machine learning, adaptation, mobile service, mobile cloud computing

1 INTRODUCTION

The immense popularity of mobile devices, mainly smartphones, has brought about a rapid development of mobile services. At the same time, this development of mobile devices, including the availability of various sensors, has caused that the services offered are becoming still smarter. They can adapt to the context in which they are located and are able to learn some of their users' behavioral patterns. The user can obtain certain hints and context-based information from the device (location, time, network communication method, light level, pulse, etc.) and from the knowledge acquired in the learning process. The user may use this information for a great variety of purposes, e.g. determining the best route or evaluating his/her health. However, some mobile services overload the mobile device, especially in the context of energy consumption. In such a situation, the Mobile Cloud Computing (MCC) paradigm can be used. This paradigm makes it possible to execute these services (or elements) in the cloud. Utilizing cloud resources allows service-oriented computations (which would otherwise be resource-intensive) to be performed faster using cloud infrastructure, and the results are sent to the mobile device. The ability to change the location where the service is run makes it possible to optimize the execution time of the service, also taking into account the energy consumption of the mobile device. In MCC, in order to decide whether the mobile service should run on the mobile device or in the cloud in the case in question, machine learning algorithms can be used. An analysis of existing MCC solutions has shown that the concept of using machine learning to optimize such systems has not yet been thoroughly investigated. There are a few articles dealing with this topic such as [1] and [2], also including the work conducted by the authors of this article: [3, 4] and [5]. In addition, there are no comprehensive studies discussing the possibility of extending the MCC paradigm to service-oriented computations on local PCs, which has been demonstrated in a novel way by the authors of this article. Such an extension allows the operating costs of a system to be decreased, but the system becomes more complex. In this case, the use of machine learning is even more justified.

A distributed system, like the one considered in this paper, is a natural environment for agent-based solutions. If the environment is complex, it is very difficult to design all system details *a priori*. To overcome this problem, one can apply a learning algorithm which makes it possible to adapt agents to the environment. In multi-agent systems, most applications use reinforcement learning [6, 7, 8]. However, in a complex environment (where the state-space is large), reinforcement learning needs time to reach satisfactory performance and the knowledge learned is very difficult to analyze. These problems suggest that other solutions, like supervised learning, should be sought. It appears that this method can be also applied to multi-agent systems and it yields results faster and in a human-readable form [9].

The main novelty of the solution proposed in this paper is the application of the agent-based framework with agents which learn autonomously on a mobile device. This makes it possible to achieve scalable learning, because no computations are performed on the data collected from mobile devices. It also protects privacy, because the information collected about task execution is processed locally instead of being sent to a server. An agent adapts the mobile service execution strategy on-line. Mobile services may be executed on a computational cloud, a local PC (personal computer) or a mobile device. The experiments are performed in a real environment.

The structure of the article is as follows: Section 2 presents the analysis of research in the field of agent-based systems in the context of mobile devices, Section 3 presents the concept of agent-based system for mobile service adaptation, Section 4 describes the implementation of this system, Section 5 introduces the results of the experiments conducted and Section 6 contains conclusions.

2 RELATED WORK

Nowadays, an increase in the importance of agent-based systems can be noticed, especially when they are used as parts of heterogeneous environments. This approach to developing systems facilitates communication between their elements located in different places. Benefits of agent-based systems are that they are composed of intelligent elements which are capable of learning and adapting. These elements are able to communicate with one another and share experiences, which further helps the evolution of the system to adapt to the current state of the environment. For this reason, agent-based systems are among the best platforms for building intelligent adaptive systems.

2.1 Learning Agents

The development of a complex, heterogeneous system is very difficult. It is practically impossible to foresee and design optimal behavior for all situations that may arise. Therefore the agents must adapt to the situations they encounter. One of the most commonly used adaptation mechanisms that may be applied for this purpose is agents learning optimal behaviors.

The most common technique used for learning strategies in agent-based systems is reinforcement learning [6, 7, 8]. The learning agent model assumes that the agent interacts with the environment in discrete steps, sets its state s by observing the environment, executes actions and receives a reward $r \in R$. The reward is high if its actions are good, and low if they are bad. The agent has to learn which action should be executed in a given state. The formal model of learning is based on a Markov process. Reinforcement learning algorithms are simple and computationally inexpensive. However, the process of learning requires a lot of trials and without complex extensions it is inefficient in large state spaces [10]. This is a result of the *curse of dimensionality*, a well-known problem in dynamic programming [11], on which reinforcement learning is based. Another disadvantage of reinforcement learning is the unreadability of the knowledge generated because the knowledge learned is represented in a low-level manner (e.g. Q-value tables). Limitations of reinforcement learning can be overcome by adding extensions to the basic algorithm. One of the most impressive results has been achieved by combining reinforcement learning and neural networks [12]. The Deep Q-Network developed as a result of this approach allows the processing of large-dimensional spaces and even image processing. However, it should be noted that such learning exhibits high computational complexity. Therefore, currently it cannot be implemented on mobile devices.

Evolutionary computation, which is a second popular method of adaptation used in agent-based systems, relies on using multiple agent generations to improve performance [6]. This approach is computationally expensive, because many populations of agents have to be maintained. As a result, it is not an appropriate choice for mobile devices. Therefore this type of adaptation is not considered here.

There are few works on supervised learning applications in multi-agent systems. In [13], rule induction is used in a multi-agent system for vehicle routing problems. However, in that work the learning is done offline. First, rules are generated by the AQ (algorithm quasi-optimal) algorithm (the same as used in this work) from global traffic data. Next, agents use these rules to predict traffic. An extension of that approach is [14]. Agents use a hybrid learning algorithm, which is executed online. Rule induction is used to decrease the size of the search space for reinforcement learning. Another approach is applied in [15, 16], where Airiau et al. add learning capabilities to the Belief-Desire-Intention model. Decision tree learning is used to support plan applicability testing. In [17], the C4.5 algorithm is used by agent to build a model of teammates.

2.2 Mobile Cloud Computing

The development of technology, including the increase in processing power, the ongoing miniaturization, and improving availability of various sensors, means that mobile devices, including smartphones, have better technical parameters and more computing power each year, which results in a broader range of applications. The development of mobile devices has improved our ability to determine the context of the device and its user. Information on the context has enabled the development of solutions that are capable of learning how they should operate in order to optimize the use of mobile device resources and also meet user expectations as best as possible.

A large number of existing solutions only utilize the resources of mobile devices and information about context [18]. Devices learn how to best perform services based on the experience acquired. However, these solutions do not enable a permanent reduction in mobile device resource consumption or a significant increase in the execution speed of mobile services. This is why another idea – the Mobile Cloud Computing paradigm – is becoming increasingly popular. In this concept, cloud computing is used for the purpose of offloading mobile services to the cloud in order to increase operating speed and reduce the load on mobile devices. There are many solutions using the MCC paradigm such as Adaptive Code Offloading for Mobile Cloud Applications, AIOLOS, AlfredO (An Architecture for Flexible Interaction with Electronic Devices), CACTSE (Cloudlet Aided Cooperative Terminal Service Environment), COMET (Code Offload by Migrating Execution Transparently), COSMOS (Clouddb for Seamless Mobile Services), Cuckoo, Elijah, EMCO, IC-Cloud (Computation Offloading to an Intermittently-Connected Cloud), MALMOS (Machine Learning-based Mobile Offloading Scheduler), MAUI (Mobile Assistance Using Infrastructure), Mirroring Mobile Device, Mobile Cloud Execution Framework, Mobility Prediction Based on Machine Learning, MOCHA (Mobile Cloud Hybrid Architecture), Replicated Application Framework, ThinkAir, VMCC (Virtualization in Mobile Cloud Computing), MpOS (Multiplatform Offloading System), VCLA (Virtual Cloud Learning Automata), Service-Oriented Context-Aware Recommender System, Mobile Multimedia Processing System and Service-Oriented Mobile Processing System (SMPS). However, only a few solutions (MALMOS, IC-Cloud, VCLA, Service-Oriented Context-Aware Recommender System, Mobile Multimedia Processing System and SMPS) utilize machine learning in order to determine the optimal location for executing the service (the mobile device or the cloud)

The architecture proposed in the MALMOS solution [1] enables decisions to be made, with the application of machine learning technologies, when to offload applications from the mobile device to the cloud. For the offloading, the solution uses the DPartner environment (Java-based on-demand offloading framework). In order to properly teach the system where it should run applications so that they launch faster, the solution uses an online training mechanism. This solution only determines the optimal location for executing the application/service and completely fails to address the important aspect of energy consumption during the launching of the application and transferring the data to the cloud and also the amount of energy used by the online learning mechanism.

Another solution, IC-Cloud [2], uses machine learning to estimate task execution times for the mobile device and for the cloud. These are used to determine the location where the task is to be executed. The time estimation system uses two components simultaneously: the offline component, which prepares the execution model for each task for a specific device prior to the launch of the application, and the online component, which utilizes the online training mechanism with machine learning algorithms on an ongoing basis. In addition, the solution also estimates the quality of network connection based on historical data and on radio signal strength. This solution requires the prior offline setup of the task execution model for each mobile device separately, which may make broader application of the solution difficult. Neither of the solutions (MALMOS, IC-Cloud) are being developed any longer.

The idea of using MCC to optimize the operation of mobile devices is still valid and important [19]. In [20], the authors propose the MpOS system (Multiplatform Offloading System), which allows offloading for mobile applications (Android/Windows Phone) using the MCC and cloudlet concepts. The decision when to offload tasks is made by the Dynamic Decision System (DDS) on the basis of predefined simple metrics such as RTT (round-trip time) or the type of network connection. At the same time, the authors investigate the impact of various types of serialization methods on the performance of the offloading process itself. The DDS system does not take into account the specificity of individual mobile applications nor their requirements related to the use of the processor or energy consumption.

Another article [21] presents an extension of the MCC concept using the cloud learning automata algorithm (VCLA). Some mobile devices are selected using Learning Automata (LA) as ad hoc virtual cloud elements and used to perform calculations. This complements the MCC concept when there is no connection to the cloud. The results of tests conducted using the QualNet 4.5 simulator have shown that the use of VCLA makes it possible to optimize the choice of the number of remote devices (in an ad hoc virtual cloud) on which the calculations are carried out. However, no experiments were conducted in a real-life testing environment which would take into account, among other things, the actual energy consumption of mobile devices.

Recently, the Deep Neural Network (DNN) mechanism has been commonly used in mobile applications, for example for speech recognition purposes (Apple Siri). The use of DNNs in the MCC often involves transferring large amounts of data between the mobile device and the cloud. In [22], the authors study the possibility of offloading only part of the DNN calculation to the cloud. The decision what calculations to send to the cloud takes into account the energy consumption of mobile devices and limited cloud resources. Test results demonstrated an increase in calculation speed and a reduction in the energy consumption of the mobile device. However, the authors only conducted simulation experiments without testing their solution under real-life conditions.

Another article dealing with certain aspects of energy consumption by mobile devices in the context of MCC is [23]. The authors propose an agent-based MCC framework using the Dynamic Programming After Filtering (DPAF) algorithm to enable the optimization of the offloading strategy, taking into account the energy consumption of mobile devices. The experiments conducted demonstrated the usefulness of the framework developed in reducing energy consumption. However, they were only performed in a simulation environment without verifying the solution developed in real-life mobile devices.

An important aspect of using the MCC concept to optimize the operation of mobile devices is security. In [24], the authors propose a secure and efficient offloading scheme which employs a combination of regular rekeying and random padding.

However, in the research conducted, the aspect of optimizing the operation of services/applications on mobile devices was not addressed and the experiments were only performed in a simulation environment.

The other solutions developed by the authors of this article such as the Service-Oriented Context-Aware Recommender System – SoCaRS [25], Mobile Multimedia Processing System – MMPS [4] and SMPS [3] make it possible to optimize (in terms of execution time and the energy used) the location where services are executed (locally or in the cloud) using MCC and machine learning algorithms.

MCC studies have been further developed in work related to Mobile Edge Computing (MEC) [26, 27] in which cloud resources are complemented by edge devices (servers) located close to the infrastructure which enables wireless transmission. This concept is used primarily in cellular networks (including 5G). Most often, systems using MEC implement optimizations on the infrastructure side and use MCC on the device and mobile application side. In MEC research, machine learning algorithms are sometimes used to optimize the use of resources [28, 29].

In [28], a novel post-decision state (PDS) based learning algorithm was used to optimize the operation of MEC. This enabled a significant improvement in edge computing performance with regard to energy aspects. The research conducted only concerned the optimization of operation and energy consumption by edge devices and did not take into account aspects related to the optimization of mobile device operation. Moreover, the authors' experiments were only conducted in a simulation environment.

In [29], the authors propose solutions for offloading in the MEC environment in the context of IoT applications (IoT-Q-L). For this purpose, they use learning agents and the Q-learning algorithm which improves the offloading of computing tasks and reduces energy consumption. Experiments confirming the possibility of optimizing task offloading were only performed in a simulation environment. Another article [30] proposes a multi-agent based flexible IoT edge computing system (F-IoT-EC) which makes it possible to optimize operation and reduce the amount of energy consumed. The system uses a rule-based engine with a fixed rule set. The tests were carried out in a simulation environment.

2.3 Agent-Based Platforms for Mobile Devices

In order to choose a framework for our system, we have analyzed the agent-based platforms available on mobile devices.

In [31], authors describe the JADE-LEAP platform as an agent-based technology for use in connection with mobile devices. JADE-LEAP is a modified version of the JADE platform that can be run on both PCs and servers, but also on mobile devices using the Java environment (e.g., on Android mobile devices). JADE is a Java framework designed for developing agent-based applications that are compliant with FIPA (Foundation of Intelligent Physical Agents) specifications [32]. JADE-LEAP message exchanges are ACL standard compliant. The exchange of messages is done asynchronously. Each agent has its own message queue, to which data are sent from the other agents. The main limitation of JADE-LEAP is the inability to run more than one agent in a separate container on the mobile device and the fact that the main container cannot be created on the mobile device. The JADE-LEAP design requires it always to be located on a PC.

In [33], authors present an agent-based software development platform called JaCaMo. JaCaMo's operating environment can be defined as a designed and programmed set of computing units called artifacts, collected in workspaces that can be distributed across the network. Agents can communicate with each other and use artifacts. In order for the agents to use artifacts, each of them should provide an appropriate interface composed of a set of operations and properties, where operations are actions that allow agents to interact with the environment. Properties define the state of a given artifact, which can be read and modified by the agent through corresponding operations.

In [34], the author describe the μ^2 platform, which is an environment used to build applications that use an effective communicating μ-agent. The platform provides a comprehensive network support. Applications developed for desktop can be (in most cases) launched on mobile devices. The main component of the μ^2 platform are agents and their roles. In contrast to conventional agent-based solutions, the μ^2 platform is strongly focused on efficiency and the minimization of performance problems. Application development is therefore primarily about implementing roles and modeling agent organizations. Java provides the basis for working in the μ^2 platform environment (the alternative is using Clojure). Jetlang is used as an internal messaging mechanism between agents. XStream is used to serialize objects over the network in XML. Netty is used for establishing connections, agent discovery and agent communication over the network.

2.4 Mobile Service Adaptation

In Sections 2.2 and 2.3, the authors primarily analyzed various existing solutions which enable service adaptation using, among others, machine learning and the MCC paradigm. The result of this analysis is the comparison of existing solutions, which is presented in Table 1.

As we can see, some systems are rule-based with manually created rules and mostly apply machine learning to train the model that is used for service adaptation. The latter solution is better because it makes it possible to adapt the system to changing conditions by executing the machine learning algorithm again. Considering all possible mobile devices, tasks and conditions would result in a very complex set of rules.

Analyzing this table, it can also be observed that many state-of-the-art solutions are tested in simulation environments. It should be noted that simulations of wireless networks are simplified and do not account for all the technical problems which occur in the real world [35].

Only a few systems use agents. However, agents are often used on the modeling level rather than at the implementation stage (SoCaRS, MMPS, SMPS, IoT-Q-L, DPAF). Moreover, there is one type of the agent defined in these systems, and that is an abstract entity which encapsulates learning and decision algorithms. It interacts with environment rather than with other agents. Also importantly, these systems are developed in a standard way, without using an agent-based framework, and certain advantages of applying an agent-based methodology (like interoperability or the ability to operate in heterogeneous environments) are not exploited. The only solution known to the authors which employs a multi-agent system in this domain is F-IoT-EC. However, it is implemented in the cloud and on the edge rather than on mobile devices and no machine learning algorithm is used.

Solutions	**Env**	**Energy**	**Time**	**ML/Rules**	**Agents**	**Tests**
MALMOS	MCC	no	yes	ML	no	real
IC-Cloud	MCC	no	yes	ML	no	real
MpOS	MCC	no	yes	Rules	no	real
VCLA	MCC	no	yes	ML	no	simulation
DNN	MCC	yes	yes	ML	no	simulation
DPAF	MCC	yes	yes	Rules	one	simulation
SoCaRS	MCC	yes	yes	ML	one	real
MMPS	MCC	no	yes	ML	one	real
SMPS	MCC	yes	yes	ML	no	real
PDS	MEC	yes	yes	ML	no	simulation
IoT-Q-L	MEC	yes	yes	ML	one	simulation
F-IoT-EC	MEC	yes	yes	Rules	multi	simulation

Table 1. Comparison of existing service adaptation solutions

Several important elements of our research constitute contributions to the area of MCC. The important aspect is that solution performance has been tested in a real-life environment. Additionally, the experience gained by the authors in developing their solutions (SoCaRS, MMPS, SMPS) has allowed them to develop the concept of an agent-based system for mobile service adaptation. However, in contrast to previous research, the authors used an agent-based platform for mobile devices (μ^2) to optimize the performance of mobile services. As a result, a genuine multi-agent system with learning agents was built. The final contribution is that the solution developed extends (in comparison to previous work) the ability to perform services on local PCs, which makes the environment more complex and heterogeneous. Currently, it consists of mobile devices, the mobile cloud and the PCs accessible via a local Wi-Fi network.

In summary, when compared with the solutions analyzed, the solution developed by the authors is the only one that uses a multi-agent system, takes into account time and energy consumption, applies ML (Machine Learning) online and has been tested in a real-life environment.

3 AGENT-BASED SYSTEM FOR MOBILE SERVICE ADAPTATION

The purpose of the research was to develop, implement and test the concept of an agent-based system for mobile service adaptation using online machine learning and the MCC paradigm. The system developed should allow adaptation in order to select the optimal place of service execution from the point of view of execution time and power consumption by a mobile device. The system keeps track of the service being performed on a current basis and selects the place of its execution using the machine learning concept on the basis of defined parameters.

The concept of system architecture (Figure 1) assumes that agents perform the services commissioned on a mobile device, on a PC and in the cloud. The operation of the system is based on interaction between agents in a heterogeneous environment. The management agent containing the service adaptation module is located on the mobile device. At the same time, there are agents responsible for launching the mobile service on the mobile device (locally), on the PC and in the cloud. By using this approach, it is possible to define a common way of exchanging messages and data, which allows communication between different service locations and the mobile device itself.

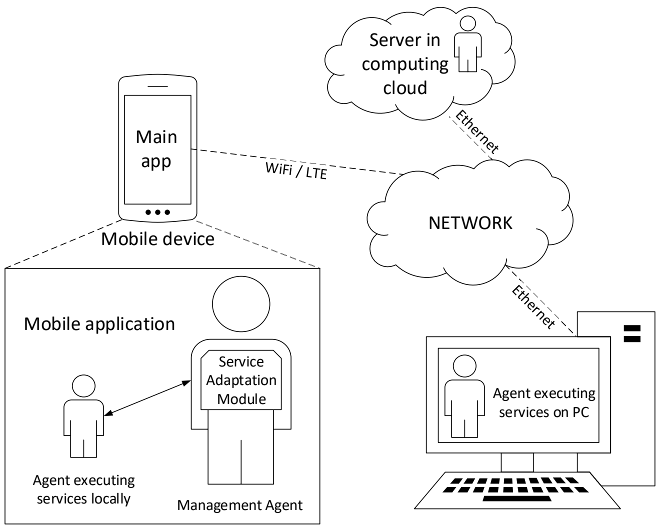
Server in computing cloud
Ethernet
Main app
WiFi / LTE
NETWORK
Mobile device
Ethernet
Mobile application
Service Adaptation Module
Agent executing services on PC
Agent executing services locally
Management Agent

Figure 1. Concept of system architecture

The Service Adaptation Module (Figure 2) consists of four main elements:

- The Manager whose task is to receive a service request, collect training data, cooperate with the Learning Module and return a response that represents the location selected.

- The Learning Module, which builds the knowledge model K from training data. K is used to select the service execution location.
- Training data – examples representing requests, locations of their execution and results of their execution as described by attributes *Attr* collected during service execution.
- Knowledge (K) – models allowing the prediction of request execution results in a given location.

The Manager receives the service request and possible locations from the Management Agent, describes them with *Attr*, creating a *problem*, and passes it to the Learning Module. The Learning Module applies K and returns a *response* representing the predicted outcome. The Manager's task is to select the best location based on such predictions. It also collects data from the execution of services and stores them in T (Training Data). It is also responsible for passing T to the Learning Module when necessary to build new knowledge K. The structure of knowledge is closely related to the type of machine learning algorithm used. Model examples include a set of regression parameters, a set of neural network parameters or a decision tree.

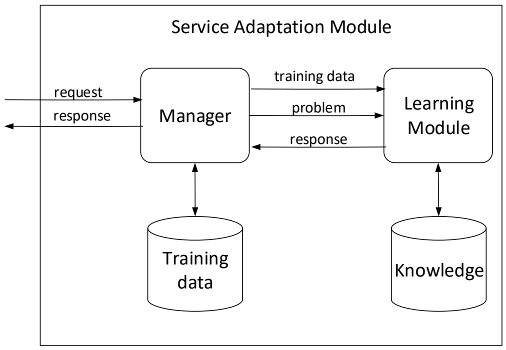

Figure 2. Service Adaptation Module architecture

More formally, the Service Adaptation Module may be defined as the following tuple:

$$SAM = (A, K, T, L) \tag{1}$$

where A is a set of attributes that are used to describe tasks and the context, K is *generated knowledge*, T is *training data*, which is a set of *examples*, and L is a set of possible execution locations.

Input data for the module is a pair $x = (t, c)$ representing the task t which should be executed in the context c. The aim of the module is to return a location $l \in L = \{l_1, l_2, \dots l_{ns}\}$, which corresponds to engaging one of ns services. The processing module describes x with observation attributes $O = \{o_1, o_2, \dots o_n\} \subset A$, which yields $x^O = (o_1(x), o_2(x), \dots o_n(x))$, i.e. a description of the *problem*. Next, using the knowledge stored in K, it solves the *problem* by selecting $l \in L$, which has the minimum predicted cost. If K is empty, l is randomized.

The decision about execution location l is then applied and the task is run using the corresponding service (i.e. locally or in the mobile cloud). After the execution, the module obtains the execution result $r(x,l)$, which is described by $Res = \{r_1, r_2, \dots r_m\} \subset A$ attributes. For example, the four following attributes may be used: whether execution was successful $es(x,l)$, power consumption $b(x,l)$, calculation time $ct(x,l)$ and user's dissatisfaction $dis(x,l)$, which may be measured by observing if the user overrode the module's decision. Therefore, the set of all attributes used to describe the input data is the sum of O and Res

$$A = O \cup Res. \tag{2}$$

The module stores these results together with x^O and location l in T. Therefore the complete training example t stored in T has the form

$$t = (o_1(x), o_2(x), \dots o_n(x), r_1(x,l), r_2(x,l), \dots r_m(x,l), l). \tag{3}$$

The models $\{M_{r_i}\}$ to predict Res values are constructed using supervised learning algorithms and stored in K. These models influence the decision selected. There are $|Res|$ models stored in K. Every model $M_{r_i} : X, L \to [0,1]$ calculates the normalized r_i value for given x and l.

Using predictions of the r_i value returned by the learned model: $M_{r_i}(x,l)$, the module may rate its decisions about all locations $l \in L$ by calculating predicted expenses $e(x,l)$:

$$e(x,l) = \sum_{i=1}^{m} w_i * M_{r_i}(x,l) \tag{4}$$

where w_i are weights of the result r_i. The weights represent user's preferences and are also related to the specificity of application which executes services. In case of an interactive application, the execution time may be more important and its weight may be higher. If mobile services are executed in a background process, the weight of battery usage may have higher value. In the production version of the system, weights can be modified by the user in the settings.

The module selects the location $l^* \in L$ for which execution is predicted to be successful and the expense is predicted to be the lowest:

$$l^* = \begin{cases} \arg\min_{l \in L}\{e(x,l) | M_{es}(x,l) = 1\}, & M_{es} \in K, \\ \arg\min_{l \in L}\{e(x,l)\}, & M_{es} \notin K. \end{cases} \tag{5}$$

The full algorithm of the *Service Adaptation Module* is presented in Figure 3. At the beginning, K and T are set to empty (lines 1–2). Next, algorithm waits for the task (line 4). If there is no learned knowledge, the decision is randomized (lines 5–6). Else there is some knowledge, therefore expenses are calculated for all possible locations (line 8). Next, the best location is selected (line 9). To provide exploration, l^* is replaced by a random location with ε probability (line 10). The task is executed in the selected service (line 12). Results of the execution are observed (line 13) and the example is stored in T (line 14). After processing a given number of tasks (line 15), *Learning Module* is called to generate new knowledge from T and the learned knowledge is stored in K (line 16).

```
1: $K := \emptyset$;
2: $T := \emptyset$;
3: while module is working do
4:    wait for $x = (t, c)$;
5:    if $K = \emptyset$ then
6:       $l$ := random decision;
7:    else
8:       calculate $e(x, l)$ according to Equation (4) for all $l \in L$ applying
         models from $K$ to predict $r_i$;
9:       $l^*$ := best location according to Equation (5);
10:      replace $l^*$ by a random location with $\varepsilon$ probability;
11:   end if
12:   execute task at the service determined by $l^*$;
13:   observe execution results;
14:   store example $t$ (see Equation (3)) in the $T$;
15:   if it is learning time (e.g. every 100 steps) then
16:      learn $K$ from $T$;
17:   end if
18: end while
```

Figure 3. Algorithm of the *Service Adaptation Module* allowing the adaptation of mobile service using machine learning

4 IMPLEMENTATION

The μ^2 platform was used for implementation because it is lightweight and allows the easy building of agents working in a heterogeneous environment. Each agent operating within the μ^2 platform must be on the same network (e.g. the same Wi-Fi network). For this reason, it is not possible to establish direct communication between an agent operating on a mobile device and an agent operating in the public cloud. Exchanges of messages between agents are accomplished using the TCP (Transmission Control Protocol).

The most important agent in the system is the ManagementAgent, which receives requests for service execution. The ManagementAgent selects the optimal location for the service and then passes the service's startup parameters to one of the service agents:

- AndroidProviderAgent – the agent responsible for the execution of the service commissioned on the mobile device;
- AWSProviderAgent – the agent responsible for the execution of the service commissioned in the computing cloud;
- PCProviderAgent – the agent responsible for the execution of the service commissioned on the desktop device.

The communication between agents is based on exchanging MicroMessage messages. The message includes information about the sender and *Intent* (data). In the Agent-based System for MCC Service Adaptation, the *Intent* object is composed of:

- the data needed for service execution;
- information about the time when service execution started;
- battery status information at the start of service execution;
- information about the type of service executed;
- the result of service execution.

The detection of active agents is accomplished through a polling mechanism – Heartbeat. The XML configuration file defines the frequency with which an agent should query the presence of other agents.

At the time of receiving the service request, the ManagementAgent selects where to execute the service, then creates a MicroMessage message with the data necessary to execute it and the information described above. The ManagementAgent sends the message to the recipient who should execute the service via the *send*(*MicroMessage message*) function. If the service should be executed by all agents available, then the *sendGlobalBroadcast*(*MicroMessage message*) function is used.

The Mobile application was written for Android mobile devices in accordance with API 17. During the implementation work, the following technologies and tools were used:

- Java language version 1.7;
- Gradle for building and managing application dependencies;
- Power Tutor for measuring the energy consumption of mobile device;
- WEKA (Waikato Environment for Knowledge Analysis) – a library providing machine learning algorithms;
- Tess4J – Java interface for the Tesseract library for text recognition (Optical Character Recognition – OCR);
- iText – a library for creating PDF documents;
- AWS (Amazon Web Services) Android SDK – a library for communicating between a mobile device and the AWS computing cloud;

- μ^2 – a platform for multi-agent systems.

WEKA was selected as the machine learning library because it has been ported to the Android system used in the development and it provides a broad range of algorithms, thus enabling comparisons between different solutions. In the production version of the system, it may be replaced by a more modern machine learning library like TensorFlow.

The Amazon AWS cloud has been used in the Agent-based System for MCC Service Adaptation. For service execution, the AWS Lambda solution is used, which is based on the IaaS (Infrastructure as a Service) model. AWS Lambda enables the implementation of features that can be remotely invoked by web applications, desktop applications and mobile applications. Due to the limitations related to μ^2 platform operation in a local area network, it was not possible to run the agent in the AWS Lambda cloud computing environment.

Two AWS Lambda functions have been created for MCC Service Adaptation: the *OCR* and *convertingPNGToPDF* functions. The OCR service uses the Tess4J API for Tesseract library operations. The same mechanism was used to implement the service on a desktop. For the OCR service using the Tesseract library, it was not possible to use the same source code as for the mobile app where the dedicated tess-two library was used, which is a modified version of the Tesseract library that can run on Android mobile devices. The same source code (using the iText library) was used for the conversion of the image file to PDF. The AWS Lambda cloud computing environment in which the functions are executed is a Java-based one. The maximum memory for the functions was set at 512 MB and the timeout was set to five minutes.

The desktop application within the Agent-based System for MCC Service Adaptation uses the μ^2 environment and Java. The most important part of the application is the agent that works on the desktop device. The agent communicates with the Management Agent on the mobile device through the network. As with other agents operating within the framework of the system, it receives the service request, executes it on the desktop device and then sends the result of its execution to the Management Agent.

In order to objectively compare the services offered by cloud computing and the desktop application, we have decided to run the desktop application under conditions that are as close as possible to those in the cloud computing environment. For this purpose, the Docker tool is used to place the program and all its dependencies, such as additional libraries, in a lightweight, portable and virtual container that can be run on a Linux server. In order to run a container on a Docker, an image must be created with boot parameters that install the appropriate libraries and other dependencies.

In order to map the AWS Lambda environment on a PC, a desktop application is started via Docker using a docker-lambda image. It has the same software and libraries installed, the same file structure and permissions, and the same environmental variables as the AWS Lambda environment. The image makes it possible to run AWS Lambda functions in the Node.JS and Python environments. Currently, the Java environment is not yet fully supported, but it is possible to launch Java programs manually.

5 PERFORMANCE EVALUATION

A series of experiments were conducted to verify the operation of the Agent-based System for MCC Service Adaptation. As test services, text recognition (OCR) and PNG transformation to PDF format were selected due to the complexity of the service and its demand for mobile device hardware resources.

Eight graphic files with varying sizes, resolutions, and amounts of text contained in them were selected to test the service with different input parameters. Each of the experiments was designed to perform a defined set of services under certain conditions. The tests included service execution, time measurements, energy consumption measurements by the mobile device while the service was being executed, and the recording of results. Observed attributes were $O = \{tt, s, res, con, sig\}$. They correspond to task type, file size, number of pixels in the picture, connection type and network connection signal strength, respectively. Execution results observed were $Res = \{ct, b\}$ (calculation time and battery usage). Result expenses were calculated according to the following formula:

$$e(x,l) = w_{ct}ct(x,l) + w_b b(x,l) \tag{6}$$

where:

- w_{ct} – weight of service execution time;
- w_b – weight of device energy consumption.

In experiments, sets of service requests were executed as consecutive rounds. During the first round, the location is selected in a random way. During each subsequent round, the Service Adaptation Module is trained on the data collected from previous rounds $1 \dots r-1$. To ensure exploration, the location is randomly selected with a probability of 10% even if knowledge is not empty.

After each service execution, the results are recorded in Training Data. After each round, the Service Adaptation Module builds new knowledge using one of the following algorithms provided by the WEKA library: J48, RandomForest, KStar, MultilayerPerceptron and SimpleLogistic. For the J48, RandomForest and KStar algorithms, numerical values of parameters are discretized into 6 compartments of equal frequency.

Every experiment consists of 8 rounds, repeated 10 times. The measurement involves total service execution time and total energy consumption of the mobile device during the round in question. The results are presented as charts representing average values and standard deviations of these measures over ten repetitions for each of the rounds.

The experiment was carried out using a mobile device – LG G2 (CPU[1] – 2.26 GHz, memory – 2 GB), router – Wi-Fi TP-Link TL-WR842N (802.11bgn), cloud – AWS Lambda platform and PC (CPU – Intel Core i5-3320M 2.6 GHz, memory – 8 GB).

5.1 Initial Tests

Initially, we measured values for cases when all tasks were executed on the mobile device. These values are 364 806 ms for average execution time and 399 556 mJ for average battery power consumption. Similarly, for executing all requests in the cloud we get an average execution time of 340 714 ms and an average battery power consumption of 408 212 mJ. It means that execution in the cloud is faster, but requires somewhat more energy because of data transmission. There are no results related to the execution of all tasks on the desktop device since the solution implemented does not enable the execution of tasks on a PC when the mobile device uses HSDPA (High-Speed Downlink Packet Access). In the scenario simulated, the user can only use the desktop device at home.

5.2 Energy Optimization

The experiment was aimed at comparing the effectiveness of machine learning algorithms in optimizing the power consumption of the mobile device. For comparison, we selected five classification algorithms: J48, RandomForest, MultilayerPerceptron, logistic regression and k-NN (k-Nearest Neighbors). The set of service requests consists of 16 individual OCR service requests with various input data. Eight of them are executed using the Wi-Fi connection, and eight using HSDPA.

Coefficient values are: the cost of service execution time w_{ct} at 0.1 and the cost of device energy consumption w_b at 0.9. Figure 4 shows the average results for each round together with standard deviations.

The logistic regression algorithm was the worst from the point of view of optimization. The results obtained by this algorithm in the rounds where location was selected were worse than in the first round where the choice of execution location was random.

The k-NN algorithm enabled a significant optimization of energy consumption, but it was susceptible to overfitting. Initially, the algorithm was getting better with each round until the fifth one, after which it started to oscillate. The statistical significance of the improvement between the first and last rounds (d) was checked using t-Student test. The p-value equals to 0.1196, which means that d is not statistically significant.

Three other algorithms (J48, Random Forest and MultilayerPerceptron) enabled similar levels of optimization. For each of them, a significant reduction in energy consumption could be observed in the early rounds. After the fifth round had been

[1] Central Processing Unit

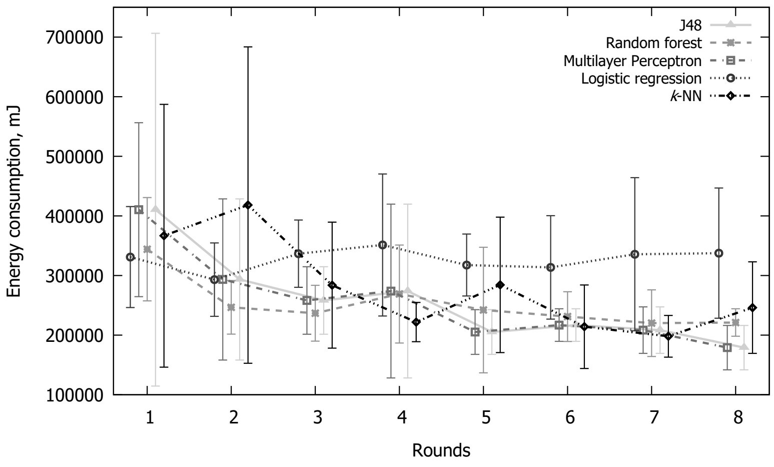

Figure 4. Optimizing energy consumption for coefficients $w_{ct} = 0.1$ and $w_b = 0.9$

completed, the process of reducing energy consumption slowed down, but oscillations were smaller. Results of the t-Student test for these classifiers (p-value) are as follows:

- for J48: 0.0245, which means that d is statistically significant;
- for RandomForest: 0.0004, which means that d is statistically significant;
- for MultilayerPerceptron: ≤ 0.0001, which means that d is statistically significant.

The experiment also examined the level of optimization in execution time with the same cost factors. The results are shown in Figure 5.

As in the case of energy consumption optimization, the logistic regression algorithm exhibited the worst results because no improvement could be observed. The k-NN and MultilayerPerceptron algorithms made it possible to decrease execution time. However, it should be noted that oscillations appeared after several rounds. The best results were obtained by the J48 and RandomForest algorithms, which reduced execution time with small standard deviations in each round and oscillations were small. The results of the t-Student test (p-value) are as follows:

- for logistic regression algorithm: 0.3501, which means that d is not statistically significant;
- for k-NN: 0.0190, which means that d is statistically significant;
- for MultilayerPerceptron: 0.0024, which means that d is statistically significant;
- for J48: ≤ 0.0001, which means that d is statistically significant;
- for RandomForest: ≤ 0.0001, which means that d is statistically significant.

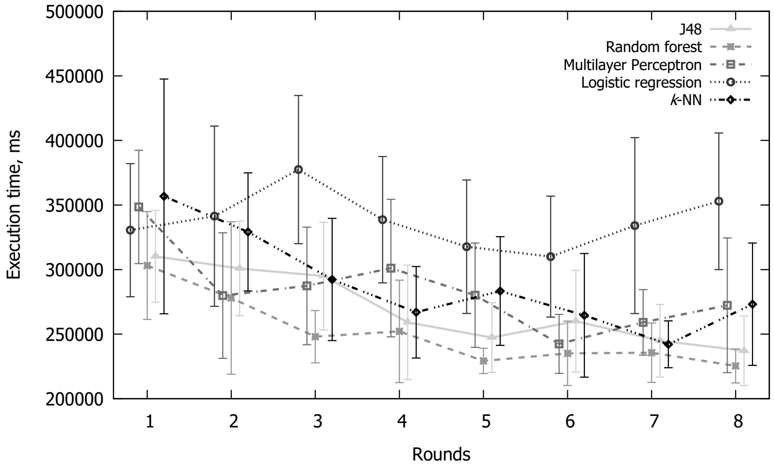

Figure 5. Optimizing execution time for coefficients $w_{ct} = 0.1$ and $w_b = 0.9$

5.3 Time Execution Optimization

The experiment methodology was analogous to the one described above. The experiment was designed to compare machine learning algorithms for selecting the service execution location in order to optimize execution time. For comparison, the three best algorithms were selected: J48, RandomForest and MultilayerPerceptron. The set of service requests is defined in the same way as in the previous experiment. The coefficients are: the cost of service execution time w_{ct} at 0.9 and the energy consumption of the device w_b at 0.1. Average results together with standard deviations are shown in Figure 6.

The J48 and RandomForest algorithms exhibited similar results. Between the second and fourth rounds, the improvement was almost linear. From round four onwards, the reduction in execution time was no longer present, and then in rounds from five to eight the execution time increased slightly. Standard deviations for these two algorithms were significantly higher than for the MultilayerPerceptron algorithm. The MultilayerPerceptron algorithm behaved similarly to the other two, with the difference that from the second to fourth rounds it enabled better optimization of execution time, and from the fifth round onwards the service execution time for the MultilayerPerceptron algorithm oscillated around the same level and standard deviations were small. The results of the t-Student test (p-value) are as follows:

- for J48: 0.0002, which means that d is statistically significant;
- for RandomForest: ≤ 0.0001, which means that d is statistically significant;
- for MultilayerPerceptron: 0.0033, which means that d is statistically significant.

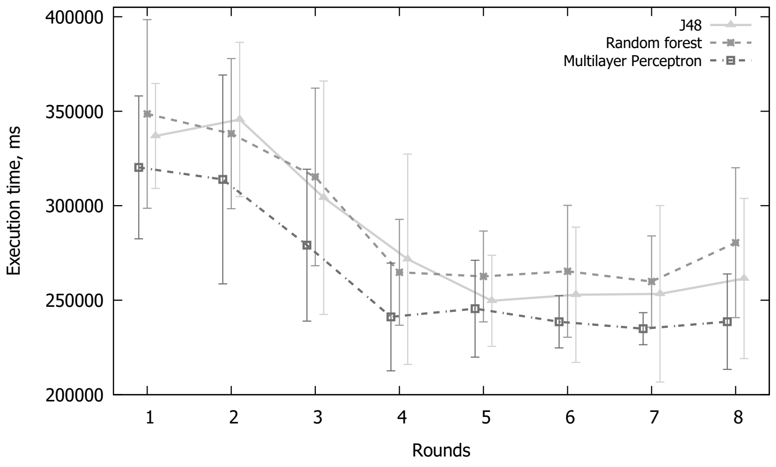

Figure 6. Optimizing execution time for coefficients $w_{ct} = 0.9$ and $w_b = 0.1$

The experiment also examines the level of energy consumption optimization on the mobile device with the same cost factors. The results are shown in Figure 7.

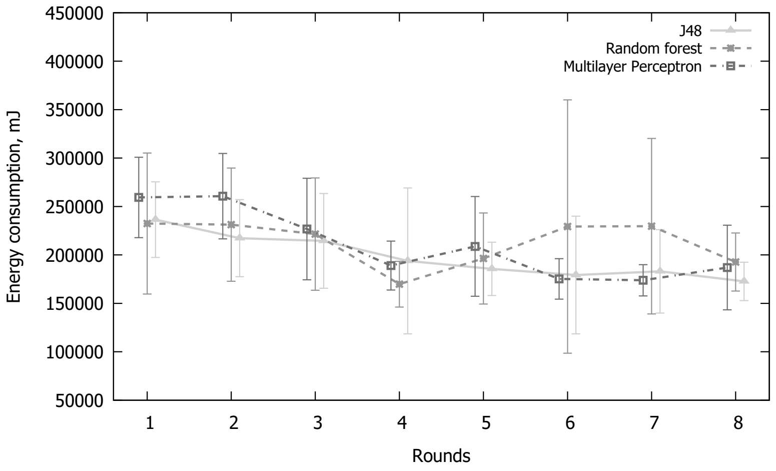

Figure 7. Optimizing energy consumption for coefficients $w_{ct} = 0.9$ and $w_b = 0.1$

In the fourth round, energy consumption by the mobile device was the lowest for the Random Forest algorithm. However, from the fourth round onwards there was an increase in energy consumption by the device and a significant increase in standard deviation could be observed in rounds six and seven. The significant increase in standard deviation was related, among others, to the specificity of the

HSDPA wireless transmission technology used in the tests, which does not allow to guarantee the quality of the network connection (including delay and bandwidth). The J48 and MultilayerPerceptron algorithms proved better in the end and they also had lower standard deviations. The results of the t-Student test (p-value) are as follows:

- for J48: 0.0002, which means that d is statistically significant;
- for RandomForest: 0.1757, which means that d is not statistically significant;
- for MultilayerPerceptron: 0.0013, which means that d is statistically significant.

5.4 Time and Energy Optimization

The experiment methodology was analogous to the previous experiments. The experiment aimed at comparing machine learning algorithms for selecting service execution location in order to optimize both mobile device execution time and energy consumption. The same algorithms were compared as in the previous experiments: J48, Random Forest and MultilayerPerceptron. The values of coefficients were as follows: service execution time cost w_{ct} at 0.5 and device energy consumption cost w_b at 0.5. Average results together with standard deviations are shown in Figures 8 and 9.

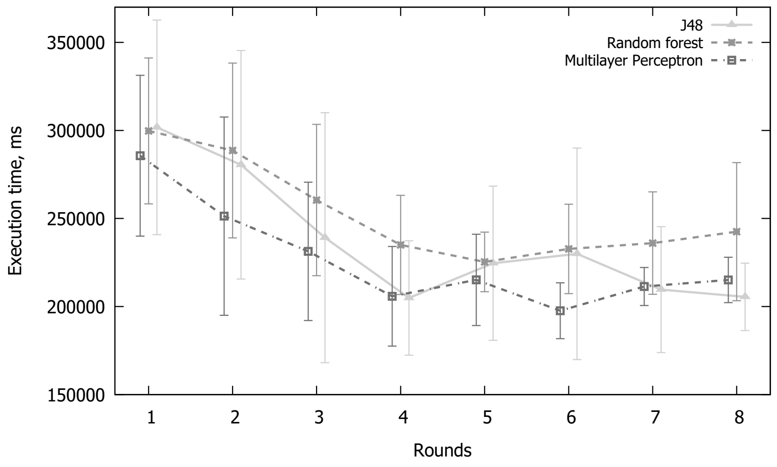

Figure 8. Optimizing execution time for coefficients $w_{ct} = 0.5$ and $w_b = 0.5$

The J48 algorithm enabled the optimization of both mobile device execution time and energy consumption. In the case of energy consumption, until the fourth round of the algorithm a significant reduction in energy consumption was noted. From the fifth round onwards, the process slowed down, but continued. A similar situation occurred with the optimization of execution time, with the difference that

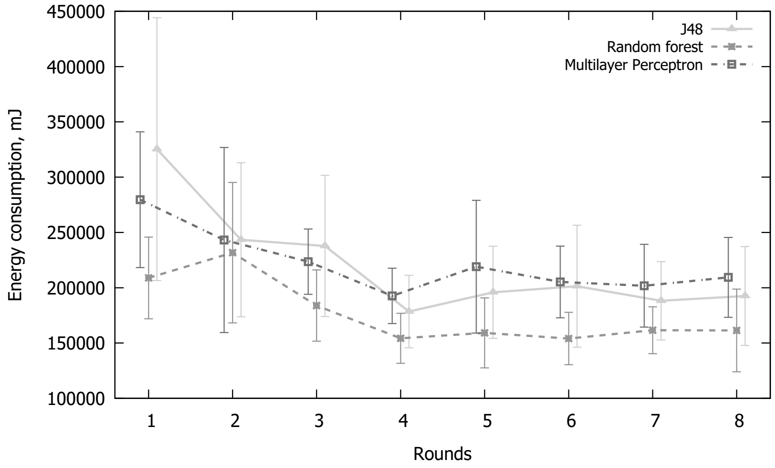

Figure 9. Optimizing energy consumption for coefficients $w_{ct} = 0.5$ and $w_b = 0.5$

from round four to six there was an increase in execution time but from round six to eight the execution time decreased again. The Random Forest algorithm behaved in the same manner with respect to both measurements. Until the fourth round, there was a significant improvement in both parameters, and from the fourth to the eighth rounds the values of the parameters oscillated around the same level. In addition, the algorithm was characterized by the fact that the standard deviation value over all rounds remained at a similar, low level. The MultilayerPerceptron algorithm also optimized both the execution time and energy consumption of the mobile device well. In both cases, the reduction was considerable until the fourth round. From round four to round eight, the values of the metrics oscillated. The feature that distinguished this algorithm from the others was also the low value of standard deviation for all rounds. The results of the t-Student test for energy consumption optimization (p-value) are as follows:

- for J48: 0.0039, which means that d is statistically significant;
- for RandomForest: 0.0059, which means that d is statistically significant;
- for MultilayerPerceptron: 0.0105, which means that d is statistically significant.

The results of the t-Student test for execution time optimization (p-value) are as follows:

- for J48: 0.0002, which means that d is statistically significant;
- for RandomForest: 0.0002, which means that d is statistically significant;
- for MultilayerPerceptron: 0.0053, which means that d is statistically significant.

5.5 Analysis of Results

As we can see, the J48, RandomForest and MultilayerPerceptron algorithms are useful for service adaptation, while k-NN and SimpleLogistic yield poor results. It should be noted that a careful choice of parameters with values different from default ones might improve their results, but the first three algorithms listed work well without such tuning.

The best optimization results were achieved by the J48 algorithm. It exhibited the lowest energy consumption or execution time in the last round in three out of six cases. This learning algorithm is relatively simple and has a low computational complexity. What is also important, the knowledge learned has a form of a decision tree, so it may potentially be analyzed by a human. Thus it appears to be the best choice for mobile cloud computing and service adaptation.

To determine which algorithms yielded the fastest improvements, the results in round three were compared. The fastest learning algorithm was Random Forest, since it won in three out of six cases. Unfortunately, this learning algorithm is more complex and needs more resources. The models learned are also difficult to analyze in this case.

All algorithms yielded rapid improvements in performance in the first three rounds. The reason for this is that during these rounds, various new examples were added to the training set. During later rounds, the examples added were more frequently similar to the previous ones and as a result, progress was much slower: results were sometimes even worse than in earlier rounds or they oscillated.

6 CONCLUSIONS

In this paper, we have proposed an agent-based solution for mobile service adaptation using machine learning. This approach enables online, autonomous adaptation on a mobile device. Through the use of machine learning algorithms as well as the Mobile Cloud Computing concept, various mobile services and applications may become smarter, faster and more energy efficient. The local execution of the learning algorithm allows for scalability because each device learns on its own. This solution also ensures privacy protection as no usage data are being sent.

We have also shown that it is possible to add one more location to execute computation, a PC computer that may by located at home or at work and used instead of the cloud. Such solution makes task-related communication much faster, because only local area network may be used. Increased complexity of the system did not influence the adaptation capability. Application of the multi-agent platform helped to design and implement PC-mobile devices cooperation.

Experimental results obtained in a real-life environment demonstrate that the application of this approach brings improvements in energy consumption and execution time that are statistically significant. The best results are obtained by the J48 algorithm. This algorithm creates models in a form of decision tree. Therefore these models can be analyzed what may be important in some applications.

In future research we would like to address, among other things, the use of localization data and the building of user models. The first improvement would make it possible to take into account user needs specific to his/her location, e.g. the ability to recharge the device while at home. On the other hand, building a user model would enable a faster start. Instead of an empty knowledge set, an initial knowledge base matching the owner profile could be used. We would also like to incorporate other metrics such as the quality of the result and its cost in the cost function (Equation (6)). We will be also investigating possibilities to use agent-based platform on a cloud by extending the functionality of the μ^2 platform with the possibility of unicast communication.

Acknowledgements

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.

REFERENCES

[1] Eom, H.—Figueiredo, R.—Cai, H.—Zhang, Y.—Huang, G.: MALMOS: Machine Learning-Based Mobile Offloading Scheduler with Online Training. Proceedings of the 2015 3rd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MOBILECLOUD '15), 2015, pp. 51–60, doi: 10.1109/mobilecloud.2015.19.

[2] Shi, C.—Pandurangan, P.—Ni, K.—Yang, J.—Ammar, M.—Naik, M.—Zegura, E.: IC-Cloud: Computation Offloading to an Intermittently-Connected Cloud. Technical report, School of Computer Science, Georgia Institute of Technology, 2013.

[3] Nawrocki, P.—Sniezynski, B.: Autonomous Context-Based Service Optimization in Mobile Cloud Computing. Journal of Grid Computing, Vol. 15, 2017, No. 3, pp. 343–356, doi: 10.1007/s10723-017-9406-2.

[4] Nawrocki, P.—Sniezynski, B.: Adaptive Service Management in Mobile Cloud Computing by Means of Supervised and Reinforcement Learning. Journal of Network and Systems Management, Vol. 26, 2018, No. 1, pp. 1–22, doi: 10.1007/s10922-017-9405-4.

[5] Nawrocki, P.—Sniezynski, B.—Slojewski, H.: Adaptable Mobile Cloud Computing Environment with Code Transfer Based on Machine Learning. Pervasive and Mobile Computing, Vol. 57, 2019, pp. 49–63, doi: 10.1016/j.pmcj.2019.05.001.

[6] Panait, L.—Luke, S.: Cooperative Multi-Agent Learning: The State of the Art. Autonomous Agents and Multi-Agent Systems, Vol. 11, 2005, No. 3, pp. 387–434, doi: 10.1007/s10458-005-2631-2.

[7] Sen, S.—Weiss, G.: Learning in Multiagent Systems. In: Weiss, G. (Ed.): Multiagent Systems. Chapter 6. MIT Press, Cambridge, MA, USA, 1999, pp. 259–298.

[8] Tuyls, K.—Weiss, G.: Multiagent Learning: Basics, Challenges, and Prospects. AI Magazine, Vol. 33, 2012, No. 3, pp. 41–52, doi: 10.1609/aimag.v33i3.2426.

[9] Sniezynski, B.: A Strategy Learning Model for Autonomous Agents Based on Classification. International Journal of Applied Mathematics and Computer Science, Vol. 25, 2015, No. 3, pp. 471–482.

[10] Kaelbling, L. P.—Littman, M. L.—Moore, A. W.: Reinforcement Learning: A Survey. Journal of Artificial Intelligence Research, Vol. 4, 1996, pp. 237–285, doi: 10.1613/jair.301.

[11] Bellman, R. E.: Dynamic Programming. A Rand Corporation Research Study. Princeton University Press, 1957.

[12] Mnih, V.—Kavukcuoglu, K.—Silver, D.—Rusu, A. et al.: Human-Level Control Through Deep Reinforcement Learning. Nature, Vol. 518, 2015, No. 7540, pp. 529–533, doi: 10.1038/nature14236.

[13] Gehrke, J. D.—Wojtusiak, J.: Traffic Prediction for Agent Route Planning. In: Bubak, M., Dick van Albada, G. D., Dongarra, J., Sloot, P. M. A. (Eds.): Computational Science – ICCS 2008. Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 5103, 2008, pp. 692–701, doi: 10.1007/978-3-540-69389-5_77.

[14] Sniezynski, B.—Wojcik, W.—Gehrke, J. D.—Wojtusiak, J.: Combining Rule Induction and Reinforcement Learning: An Agent-Based Vehicle Routing. Proceedings of the 2010 Ninth International Conference on Machine Learning and Applications (ICMLA 2010), 2010, pp. 851–856, doi: 10.1109/icmla.2010.132.

[15] Airiau, S.—Padgham, L.—Sardina, S.—Sen, S.: Incorporating Learning in BDI Agents. Proceedings of the ALAMAS + ALAg Workshop at AAMAS, Estoril, Portugal, May 2008.

[16] Singh, D.—Sardina, S.—Padgham, L.—Airiau, S.: Learning Context Conditions for BDI Plan Selection. Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS '10), Toronto, Canada, 2010, Vol. 1, pp. 325–332.

[17] Barrett, S.—Stone, P.—Kraus, S.—Rosenfeld, A.: Learning Teammate Models for Ad Hoc Teamwork. International Conference on Autonomous Agents and Multiagent Systems (AAMAS), Adaptive Learning Agents (ALA) Workshop, Valencia, Spain, June 2012.

[18] Korpipaa, P.—Mantyjarvi, J.—Kela, J.—Keranen, H.—Malm, E. J.: Managing Context Information in Mobile Devices. IEEE Pervasive Computing, Vol. 2, 2003, No. 3, pp. 42–51, doi: 10.1109/mprv.2003.1228526.

[19] Zhou, B.—Buyya, R.: Augmentation Techniques for Mobile Cloud Computing: A Taxonomy, Survey, and Future Directions. ACM Computing Surveys (CSUR), Vol. 51, 2018, No. 1, pp. 13:1–13:38, doi: 10.1145/3152397.

[20] Rego, P. A. L.—Costa, P. B.—Coutinho, E. F.—Rocha, L. S.—Trinta, F. A. M.—de Souza, J. N.: Performing Computation Offloading on Multiple Platforms. Computer Communications, Vol. 105, 2017, pp. 1–13, doi: 10.1016/j.comcom.2016.07.017.

[21] Krishna, P. V.—Misra, S.—Saritha, V.—Raju, D. N.—Obaidat, M. S.: An Efficient Learning Automata Based Task Offloading in Mobile Cloud Comput-

ing Environments. 2017 IEEE International Conference on Communications (ICC), 2017, pp. 1–6, doi: 10.1109/icc.2017.7997139.

[22] Eshratifar, A. E.—Pedram, M.: Energy and Performance Efficient Computation Offloading for Deep Neural Networks in a Mobile Cloud Computing Environment. Proceedings of the 2018 Great Lakes Symposium on VLSI (GLSVLSI '18), ACM, 2018, pp. 111–116, doi: 10.1145/3194554.3194565.

[23] Kuang, Z.—Guo, S.—Liu, J.—Yang, Y.: A Quick-Response Framework for Multi-User Computation Offloading in Mobile Cloud Computing. Future Generation Computer Systems, Vol. 81, 2018, pp. 166–176, doi: 10.1016/j.future.2017.10.034.

[24] Meng, T.—Wolter, K.—Wu, H.—Wang, Q.: A Secure and Cost-Efficient Offloading Policy for Mobile Cloud Computing Against Timing Attacks. Pervasive and Mobile Computing, Vol. 45, 2018, pp. 4–18, doi: 10.1016/j.pmcj.2018.01.007.

[25] Nawrocki, P.—Śnieżyński, B.—Czyżewski, J.: Learning Agent for a Service-Oriented Context-Aware Recommender System in a Heterogeneous Environment. Computing and Informatics, Vol. 35, 2016, No. 5, pp. 1005–1026.

[26] Dolui, K.—Datta, S. K.: Comparison of Edge Computing Implementations: Fog Computing, Cloudlet and Mobile Edge Computing. 2017 Global Internet of Things Summit (GIoTS), June 2017, pp. 1–6, doi: 10.1109/giots.2017.8016213.

[27] Mao, Y.—You, Ch.—Zhang, J.—Huang, K.—Letaief, K. B.: A Survey on Mobile Edge Computing: The Communication Perspective. IEEE Communications Surveys and Tutorials, Vol. 19, 2017, No. 4, pp. 2322–2358, doi: 10.1109/comst.2017.2745201.

[28] Xu, J.—Chen, L.—Ren, S.: Online Learning for Offloading and Autoscaling in Energy Harvesting Mobile Edge Computing. IEEE Transactions on Cognitive Communications and Networking, Vol. 3, 2017, No. 3, pp. 361–373, doi: 10.1109/tccn.2017.2725277.

[29] Alam, M. G. R.—Hassan, M. M.—Uddin, M. Z.—Almogren, A.—Fortino, G.: Autonomic Computation Offloading in Mobile Edge for IoT Applications. Future Generation Computer Systems, Vol. 90, 2019, pp. 149–157, doi: 10.1016/j.future.2018.07.050.

[30] Ogino, T.—Kitagami, S.—Suganuma, T.—Shiratori, N.: A Multi-Agent Based Flexible IoT Edge Computing Architecture Harmonizing Its Control with Cloud Computing. International Journal of Networking and Computing, Vol. 8, 2018, No. 2, pp. 218–239.

[31] Bergenti, F.—Caire, G.—Gotta, D.: Agents on the Move: JADE for Android Devices. In: Santoro, C., Bergenti, F. (Eds.): WOA 2014. CEUR Workshop Proceedings, Vol. 1260, 2014, Art. No. 9.

[32] Bellifemine, F.—Poggi, A.—Rimassa, G.: JADE: A FIPA2000 Compliant Agent Development Environment. Proceedings of the Fifth International Conference on Autonomous Agents (AGENTS '01), ACM, 2001, pp. 216–217, doi: 10.1145/375735.376120.

[33] Boissier, O.—Bordini, R.—Hubner, J.—Ricci, A.—Santi, A.: Multi-Agent Oriented Programming with JaCaMo. Science of Computer Programming, Vol. 78, 2013, No. 6, pp. 747–761, doi: 10.1016/j.scico.2011.10.004.

[34] Frantz, Ch.: Micro-Agents Revisited: A Modern Reimplementation of the Micro-Agent Layer of the Otago Agent Platform (OPAL). Master thesis, University of Koblenz-Landau, Germany, 2010.

[35] Haq, F.—Kunz, T.: Simulation vs. Emulation: Evaluating Mobile Ad Hoc Network Routing Protocols. Proceedings of the International Workshop on Wireless Ad-Hoc Networks (IWWAN 2005), London, 2005.

Piotr Nawrocki is Assistant Professor in the Department of Computer Science at the AGH University of Science and Technology, Poland. His research interests include distributed systems, computer networks, mobile systems, service-oriented architectures and Mobile Cloud Computing. He obtained his Ph.D. in computer science from AGH Krakow.

Bartłomiej Śnieżyński 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 Postdoctoral Fellow under the supervision of the 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 the AGH. His research interests include machine learning, multi-agent systems, and knowledge engineering.

Jakub Kołodziej is a computer science graduate student at the AGH University of Science and Technology in Krakow, Poland.