Computers and Electrical Engineering 80 (2019) 106497

Contents lists available at ScienceDirect

Computers and Electrical Engineering

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

Proactive machine learning-based solution for advanced manageability of multi-persona mobile computing☆

Hanine Tout[a], Nadjia Kara[a], Chamseddine Talhi[a], Azzam Mourad[b,*]

[a] École de Technologie Supérieure (ETS), Montreal, Canada
[b] Lebanese American University (LAU), Beirut, Lebanon

ARTICLE INFO

Article history:
Received 12 December 2017
Revised 25 October 2019
Accepted 28 October 2019
Available online 1 November 2019

Keywords:
Mobile device
Multi-persona mobile computing
Mobile cloud computing
Offloading
Optimization
Dynamic programming
Machine learning
Artificial intelligence

ABSTRACT

Latest mobile virtualization techniques have opened the door for multi-persona mobility to overcome security and privacy concerns of bring-your-own devices practice. Multi-persona allows a physical device to co-host multiple virtual phones with impenetrable walls among them. However, physical resources should be always enough to support virtual instances and applications needs without performance degradation or system crash. Though computation offloading can augment devices resources, yet some applications are not offloadable. Additionally, idle applications and virtual environments impose high overhead on the device. Through machine learning, this work predicts future context and resource needs of currently running virtual environments and potential future active ones. It provides advanced manageability strategies, formulated in an optimization model, which appropriately turn off applications and switch off virtual environments to release device resources when needed. A dynamic programming algorithm is advocated to find the adequate strategies. Extensive experiments conducted demonstrate the efficiency of our proposition.

1. Introduction

The emergence of smartphones as the linchpin of everyday computing and communication, has forced one of the most major shifts that corporates have ever seen. Mobile computing has gone from a niche market to the fastest growing, and often most popular, way to do business computing. Mobile computing is becoming not only a new computing platform, but the dominant one for many enterprises. To cope with this wave, Bring Your Own Device (BYOD) [1] has been embraced, as a practice that allows employees to use their personal mobile devices to access corporate data and applications. The trade-off, of course, is that the corporate data therein, is being accessed and stored on personal devices, which raises many security concerns. With employee-owned devices at work, the chances of confidential company data mixing with personal employee information are high, and the instances of data leakage and loss are even higher. Along the way, corporate culture had to change in order to accommodate the always-present nature of the modern smartphone, and security practices have been completely rethought to deal with the challenge of alien, uncontrolled devices being brought inside the corporate firewall. Many enterprises have dealt with this issue by requiring employee devices to adhere to certain security policies. Yet, this in turn has raised some privacy issues for the end users who might sacrifice their personal data to comply with such policies.

☆ This paper is for regular issues of CAEE. Reviews processed and recommended for publication to the Editor-in-Chief by Associate Editor Dr. M. H. Rehmani.

* Corresponding author.
E-mail address: azzam.mourad@lau.edu.lb (A. Mourad).

https://doi.org/10.1016/j.compeleceng.2019.106497
0045-7906/

Controlling these concerns from different perspectives has proven to be just too difficult for both the company and the user [2]. This will be also further demonstrated in the rest of the article. Fortunately, with the option to tap into a virtualization solution, a user can differentiate between these contexts through virtual phone dedicated for each. Similar to virtualization on servers and desktop machines, mobile virtualization allows several virtual phones (VPs) to run simultaneously on the same physical mobile device with a clear isolation among them [2]. Leveraging mobile virtualization, dual persona devices have been already released, enabling two phones-in-a-phone, to support BYOD needs [3]. Yet, satisfying many of our daily-life needs nowadays, urges mobile devices to broaden their capabilities to support more than just two contexts and drive multi-persona device to be the new game changer. Multi-persona creates the same impenetrable walls between an employee's applications and an organization's data and applications, yet allowing one phone to co-host "multiple completely independent and secure virtual environments" [4]. But why would anybody want more than just two virtual phones? With multi-persona, a user could isolate private banking services and e-commerce, sensitive corporate data, social networking and games in separate VPs, in order to efficiently manage financial transactions, prevent untrusted applications from accessing critical information and share the device with other family members without ending up with accidental phone calls, unintended in-app purchases or even access to restricted content. Even more interesting use cases for multi-persona are those that necessitate multiple VPs for work with different levels of security. While working at their private clinic and at multiple hospitals, doctors are subject to different mobile policies, reflecting each of the different institutions. With personal, clinic and hospitals VPs, multi-persona allow doctors to comply with the policy of each and effectively treat their patients while maintaining their own unburdened personal use of the device [2]. As a matter of fact, the success of multi-persona lies in its capability of supporting multiple virtual devices concurrently on a single terminal, while making the latter able to clearly distinguish between the different contexts in which it is used.

Though its isolation competency, mobile virtualization imposes significant overhead on the mobile terminal with limited computation capabilities, memory capacity and battery lifetime. In one hand, previous works [5,6] have proved that even a lightweight virtualization architecture can be costly for the mobile device resources, causing performance degradation and more critically shorter viability of the system. On the other hand, technological advances have markedly shaped the way computations are performed. With the abundance of cloud resources, many computation offloading approaches have been proposed to support mobile devices needs by migrating the computations from an end mobile device to remote resourceful infrastructure. All these approaches have indeed proved their competency to reduce the resource consumption on the mobile terminals and enhance the latter performance [7–9]. From these premises, we proposed in previous work offloading-based approach to support multi-persona [6]. Through optimization and heuristics, the solution was capable to find the dissemination of computations, in each virtual phone, between local and remote execution capable of minimizing the resource consumption and augmenting the applications' performance. However, in many scenarios, the capability of only offloading computations might just not be sufficient to manage the device resources.

"Why offloading might not be enough as a resource management solution?" Starting with the first scenario (Sc_1) where typically some computations are not offloadable whether based on their type (native tasks), security level, or even their need to call device-related functionalities (e.g., camera); when more resources are needed on the device, offloading these components is not an option and different management decisions should be taken. In another scenario (Sc_2), the device can run out of resources (e.g., battery) while some applications have been idle for a period of time and will not be used in the near future, yet still consuming part of the device resources. Offloading such applications will not be efficient as it might cost the device more energy for transmitting the needed data for remote processing. Moreover, imagine the device is running out of resources while some VPs are idle (i,e., either running idle applications or none) and will not get active in the near future; also here just an offloading decision will not be able to efficiently cope with such situation as the virtual phones are still stressing out the device resources (Sc_3). Further, typically, local and remote application execution are adopted based on their capability to enhance the performance of an application or reduce the energy consumed on the device. However, in some use cases, one of these strategies should be enforced independently of the cost that they impose on the performance of the applications. For instance, the case when the device is running out of battery, while local execution of the running components outperforms their remote execution (Sc_4).

"So what kind of advanced manageability is effectively needed besides offloading?" Along with offloading, other manageability solutions should be available. Shutting down, whether idle applications which will not be used in the next period of time or active non offloadable components, can cope with the first two aforementioned scenarios (Sc_1, Sc_2). While the ability to switch off particular virtual phones can release the device resources and hence cope with (Sc_3). Finally, prioritizing system viability over individual app performance when evaluating offloading cost is critically needed in (Sc_4) as the viability of the whole system is more critical than the performance of particular application(s).

"How does this work address these research problems and offer the needed manageability solutions?" We propose in this work a proactive solution with advanced manageability to control the virtual instances running on the mobile device. Through machine learning intelligence [10], our proposition is able to predict beforehand the context and the resource needs for future execution and manage the device resources accordingly. It is able to predict the resource needs of the currently running VPs and applications as well as those VPs and applications that may run in the future and their resource demands, all based on the usage pattern of the device. Additionally, the solution offers advanced management strategies, particularly, offloading the execution, turning off components, switching off personas and prioritizing device survivability over applications performance, all according to the contextual environment that might affect the application of these strategies. Based on the predicted context and resource needs, detected hotspot (active and resource intensive) and coldspot (idle) compo-

nents and virtual phones, classification scheme, network conditions and device state, an optimization model is formulated. We propose respectively a dynamic programming algorithm to solve this model, where the solution indicates for each computation the adequate strategy to handle the resource and performance needs on the end terminal. We performed thorough experiments and the results prove the efficiency of our proposition to manage VPs needs.

This work offers the following contributions:

- Proactive machine learning technique to predict future context and resource needs. Based on VPs statistics, our proposition predicts the device context in future time slot, which includes timestamp, location, running VPs, components and resource needs accordingly aiming to provide proactive solution that avoids performance degradation or system crash.
- Advanced strategies to manage VPs, where idle applications that will not be used in the near future or in case the whole system is in critical situation, VPs and/or some applications can be switched off to save resources and ensure longer viability on the physical device.
- Novel optimization model to meet with the resource needs and enhance the performance on end devices with multiple VPs by minimizing the resource usage on the device while assuring availability of predicted future context and resource requirements.
- Efficient dynamic programming algorithm to solve the optimization model aiming to find the adequate strategies to be applied by the end terminal.

The rest of the article is organized as follows: Section 2 illustrates the proposed system model, while Section 3 provides background about different machine learning techniques. Next, in Section 4, we provide the formal formulation of our problem and in Section 5, we present the proposed algorithm to solve it. Afterwards, in Section 6, we present the thorough evaluation conducted. In Section 7, we discuss relevant literature and finally, in Section 8, we conclude the paper and draw some future directions.

2. System model

The capacity of the physical device should be sufficient to satisfy the resource needs of all VPs it is hosting. Otherwise, the physical mobile device is overloaded and will lead to degraded performance of its VPs or even system crash when there is no enough resources to support future VPs needs. Besides the offloading strategy offered in this work, some green computing actions can be taken; VPs and applications can be switched off to save resources and ensure longer viability on the physical device as long as they are idle and won't be used in the near future or whenever the whole system is in critical situation. The proposed system model is depicted in Fig. 1.

A set of profilers are installed in each VP to monitor different aspects. The program profiler monitors tasks' energy consumption, execution time and size of data to be transmitted in case of remote processing. The network profiler monitors

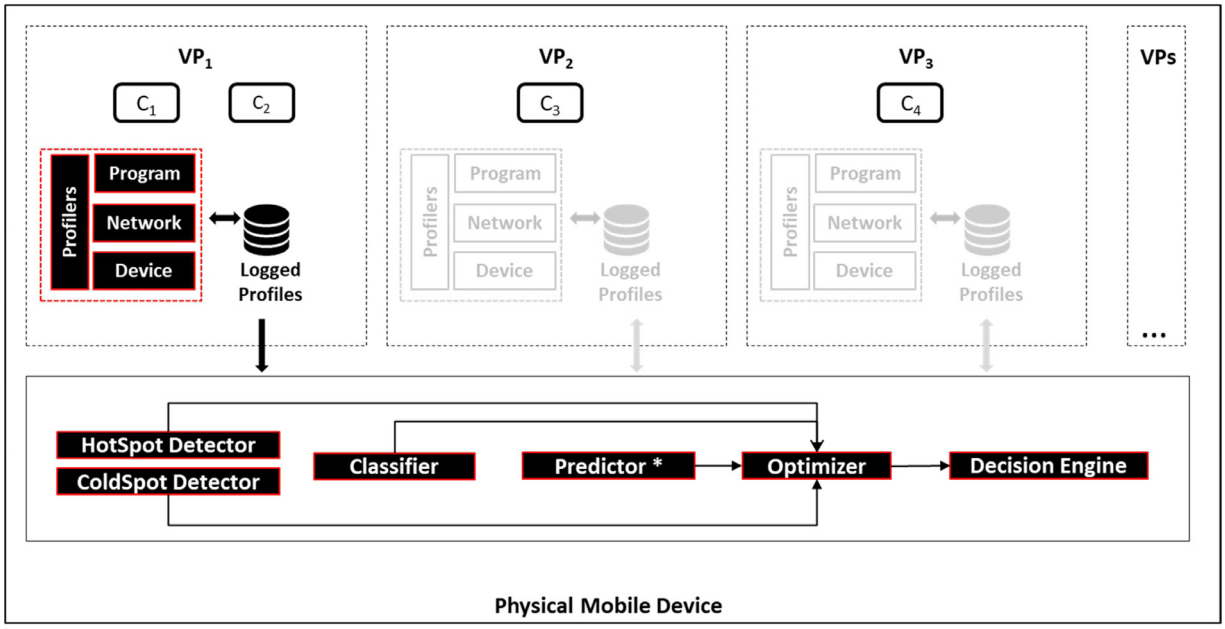


Fig. 1. System model.

Table 1
Prioritization scheme.

Class				Priority
Critical	AC	Active	Foreground	1
Critical	AC	Idle	Foreground	2
Critical	AC	Active	Background	3
Critical	AC	Idle	Background	4
Critical	DC	Active	Foreground	5
Critical	DC	Idle	Foreground	6
Critical	DC	Active	Background	7
Critical	DC	Idle	Background	8
Uncritical	AC	Active	Foreground	9
Uncritical	AC	Idle	Foreground	10
Uncritical	AC	Active	Background	11
Uncritical	AC	Idle	Background	12
Uncritical	DC	Active	Foreground	13
Uncritical	DC	Idle	Foreground	14
Uncritical	DC	Active	Background	15
Uncritical	DC	Idle	Background	16

the network characteristics in terms of availability, type (e.g., wifi, 3G), bandwidth, latency and energy consumed on transmission. The device profiler inspects the energy consumption on the device as well as the battery level, CPU utilization, memory consumption, and keep track of the location and time where each VP and application is used. The gathered information helps predicting future usage context and resource needs.

The statistics collected at each VP are forwarded to the predictor component, which applies machine learning techniques to predict the context of the device in the future time slot. The context includes timestamp, location, VPs running, components running and resource needs accordingly. In other words, this includes the resource needs of the currently running VPs and applications as well as those will run in the future and their resource demands. The predictor aims to provide proactive solution capable of avoiding critical situations of performance degradation or system crash. Any of the local or remote predictors can be invoked to train the model and perform the prediction. The performed experiments presented later in this work investigates remote prediction while the evaluation of processing training and prediction locally and their effect on the device accordingly are to be investigated in future work.

To manage limited system resources, the Android system can terminate running applications. If the Android system needs to free up resources and terminate processes, it uses the following priority system [11]. Based on its status, each process is assigned a priority level. Foreground, visible, service, background and empty processes are assigned priorities from 1 to 5 respectively. Those processes with higher priority levels have higher chance to be terminated. To turn off particular applications and VPs we apply the following classification scheme for prioritization. We define 5 classes to categorize a **component**

- Offloadable/Notoffloadable
- Background/Foreground
- Active/Idle: doesn't actively utilize system resources
- Belong to actual context VP (AC)/Not (DC)
- Critical/Not: based on user preferences

As for **VPs**, we define:

- Background/Foreground
- Active/Idle: is not running any component which actively utilize system resources
- Define actual context (AC)/Not (DC)
- Critical/Not: based on user preferences

Table 1 shows the prioritization scheme applied.

Frequently invoked, resource-intensive and/or time consuming components are designated as hotspots. Such criteria and the thresholds used for hotspot selection are adjusted automatically based on the device state, instrumented by the profilers. The hotspot detector analyzes the logged profile information and collects all the components whose processing, memory, data size, execution time and frequency of call are greater than given threshold values respectively. These components are marked as hotspots. Higher thresholds will keep the hotter components. For more information about the hotspot detector and its algorithm, please refer to [12]. In contrast, the coldspot detector detects idle components and VPs. An idle VP is one that is not running any active component or just idle ones.

According to the predicted context and resource needs, classification scheme, hotposts, colspots, and instrumented data a multi-objective optimization problem of three vectors of variables is formulated in the optimizer module. The objective functions aim to manage the resource needs of VPs, avoid performance degradation and system crash. Finally, for the decision engine, we propose a novel algorithm based on dynamic programming to find the adequate strategies to be applied

by the physical device. The decision implies which VPs should remain on and/or those to be switched off, and for each component whether it should be powered-off, executed locally or offloaded.

3. Machine learning prediction

To predict future context and usage behavior, we study a variety of machine learning techniques [10], namely, LR [13], SVR [14], NN [15] and DNN [16], which are well known algorithms in machine learning and used nowadays to solve problems in many research areas. We compare the accuracy of these techniques and the one found to be with the highest accuracy is to be used throughout this work.

3.1. Linear regression

Linear regression [13] is a linear model that assumes a linear relationship between the input variable(s) x and the single output variable y. In other words, y can be calculated from a linear combination of the input variables x. The method is referred to as simple linear regression, when there is one input variable and as multiple linear regression method when there are multiple input variables.

The linear equation assigns one scale factor to each input, called a coefficient (β). One additional coefficient is also added, giving the line an additional degree of freedom (e.g. moving up and down on a two-dimensional plot), often called the bias coefficient or intercept (β_0). In a simple regression problem, the form of the model would be: $y = \beta_0 + \beta_1 * x$

In higher dimensions when more than one input variable x are used, the line is called a plane or a hyper-plane and the linear relationship can be expressed as: $y = \beta_0 + \beta_1 * x_1 + \beta_2 * x_2 + \ldots + \beta_n * x_n$

3.2. Support vector regression

Support vector machine analysis is a popular machine learning tool for classification and regression [14]. SVR is considered a non-parametric technique because it relies on kernel functions. The model generated by SVR depends only on a subset of the training data, because the cost function for building the model ignores any training data close to the model prediction.

The objective is to

Minimize $\frac{1}{2}||w||$

Subject to $\begin{cases} y_i - \langle w, x_i \rangle - b \leq \varepsilon \\ \langle w, x_i \rangle + b - y_i \leq \varepsilon \end{cases}$ where x_i is a training sample with target value y_i. The inner product plus intercept $\langle w, x_i \rangle + b$ is the prediction for that sample, and ε is a free parameter that serves as a threshold; all predictions have to be within an ε range of the true predictions.

3.3. Neural network

Neural Network [15] also called Artificial Neural Network, is a learning algorithm inspired by the structure and functional aspects of biological neural networks. Computations are structured in terms of an interconnected group of artificial neurons, processing information using a connectionist approach to computation. Modern neural networks are usually used to model complex relationships between inputs and outputs, to find patterns in data, or to capture the statistical structure in an unknown joint probability distribution between observed variables.

3.4. Deep neural network

A deep neural network [16] is an ANN with multiple hidden layers between the input and output layers. Similar to ANN, DNN can model complex non-linear relationships. DNNs are typically feed-forward networks in which data flows from the input layer to the output layer without looping back.

4. Problem formulation

In the following, we mathematically formulate the Multiple Virtual Phones (MVPs) problem as a multi-objective optimization model with three vectors decision variables.

- Decision Variables:
  $I_{vp_j} = \{I_{vp_1}, \ldots I_{vp_m}\}$
  $I_{c_i,vp_j} = \{I_{c_1,vp_1}, \ldots I_{c_n,vp_m}\}$
  $x_{c_i,vp_j} = \{x_{c_1,vp_1}, \ldots x_{c_n,vp_m}\}$
  where,

$$\forall vp_{j,j:1\rightarrow m}, I_{vp_j} = \begin{cases} 0, & \text{if } vp_j \text{ should be switched off} \\ 1, & \text{if } vp_j \text{ should remain active} \end{cases}$$

$$\forall c_{i,i:1\to n}, \forall vp_{j,j:1\to m}, I_{c_i,vp_j} = \begin{cases} 0, & \text{if } c_i \in vp_j \text{ should be turned off} \\ 1, & \text{if } c_i \in vp_j \text{ should remain on} \end{cases}$$

$$\forall c_{i,i:1\to n}, \forall vp_{j,j:1\to m}, x_{c_i,vp_j} = \begin{cases} 0, & \text{if } c_i \in vp_j \text{ is to be executed locally} \\ 1, & \text{if } c_i \in vp_j \text{ is to be offloaded} \\ -(indifferent), & \text{if } I_{c_i,vp_j} = 0 \end{cases}$$

- Notations:

m	number of virtual phones
n	number of components in a virtual phone
j	1,... m
i	1,... n
vp_j	virtual phone
c_i	component
p_{cpu}	power consumed by vp_j on processing
p_s	power consumed by vp_j on the screen
p_{cpu}^{idle}	power consumed by vp_j on idle CPU
p_{net}^{active}	power consumed on network being active
p_{tr}	power consumed for data transmission
$Data_{c_i,vp_j}$	size of data transmitted for offloading c_i, $c_i \in vp_j$
CPU_{c_i,vp_j}^{local}	cpu usage in vp_j by c_i executed locally
CPU_{c_i,vp_j}^{remote}	cpu usage in vp_j by c_i offloaded
$Memory_{c_i,vp_j}^{local}$	memory usage in vp_j by c_i executed locally
$Memory_{c_i,vp_j}^{remote}$	memory usage in vp_j by c_i offloaded
t_{c_i,vp_j}^{local}	time to execute c_i locally, $c_i \in vp_j$
t_{c_i,vp_j}^{remote}	round trip time to process c_i remotely, $c_i \in vp_j$
W_{cpu}	weight for objective function (1)
W_{memory}	weight for objective function (2)
W_{energy}	weight for objective function (3)
W_{time}	weight for objective function (4)
$\tilde{T}_{cpu}$	threshold for cpu usage
$\tilde{T}_{memory}$	threshold for memory usage

- Mathematical Model: $F = Minimize \begin{bmatrix} F_{cpu} \\ F_{memory} \\ F_{energy} \\ F_{time} \end{bmatrix}$

subject to

$F_{cpu} < \tilde{T}_{cpu}$	(c_1)
$F_{memory} < \tilde{T}_{memory}$	(c_2)

where,

$$F_{cpu}\left(I_{vp_j}, I_{c_i,vp_j}, x_{c_i,vp_j}\right) = W_{cpu} \times \left[\sum_{j=1}^{m} I_{vp_j} \sum_{i=1}^{n} I_{c_i,vp_j} \times \left(1 - x_{c_i,vp_j}\right) \times CPU_{c_i,vp_j}^{local} + \sum_{j=1}^{m} I_{vp_j} \sum_{i=1}^{n} I_{c_i,vp_j} \times x_{c_i,vp_j} \times CPU_{c_i,vp_j}^{remote} \right] \quad (1)$$

$$F_{memory}\left(I_{vp_j}, I_{c_i,vp_j}, x_{c_i,vp_j}\right) = W_{memory} \times \left[\sum_{j=1}^{m} I_{vp_j} \sum_{i=1}^{n} I_{c_i,vp_j} \times \left(1 - x_{c_i,vp_j}\right) \times M_{c_i,vp_j}^{local} + \sum_{j=1}^{m} I_{vp_j} \sum_{i=1}^{n} I_{c_i,vp_j} \times x_{c_i,vp_j} \times M_{c_i,vp_j}^{remote} \right] \quad (2)$$

$$F_{energy}\left(I_{vp_j}, I_{c_i,vp_j}, x_{c_i,vp_j}\right) = W_{energy} \times \left[\sum_{j=1}^{m} I_{vp_j} \sum_{i=1}^{n} I_{c_i,vp_j} \times \left(1 - x_{c_i,vp_j}\right) \times \left(\left(P_{cpu} + P_s\right) \times t_{c_i,vp_j}^{local}\right) + \sum_{j=1}^{m} I_{vp_j} \sum_{i=1}^{n} I_{c_i,vp_j} \times x_{c_i,vp_j} \times \left(\left(P_{cpu}^{idle} + P_s + P_{net}^{active}\right) \times t_{c_i,vp_j}^{remote} + \left(P_{tr} \times \left(Latency + \frac{Data_{c_i,vp_j}}{Bandwidth}\right)\right)\right)\right] \quad (3)$$

$$F_{time}\left(I_{vp_j}, I_{c_i,vp_j}, x_{c_i,vp_j}\right) = W_{time} \times \left[\sum_{j=1}^{m} I_{vp_j} \sum_{i=1}^{n} I_{c_i,vp_j} \times \left(1 - x_{c_i,vp_j}\right) \times t_{c_i,vp_j}^{local} + \sum_{j=1}^{m} I_{c_i,vp_j} \sum_{i=1}^{n} I_{c_i,vp_j} \times x_{c_i,vp_j} \times \left(t_{c_i,vp_j}^{remote} + Latency + \frac{Data_{c_i,vp_j}}{Bandwidth}\right)\right] \quad (4)$$

The aim of this model is to find the strategies (determined by the decision variables I_{vp_j}, I_{c_i,vp_j} and x_{c_i,vp_j}) able to minimize the resource usage on the physical mobile device in order to avoid performance degradation, while assuring the availability of predicted future context and resource needs (identified in constraints (c_1) and (c_2)). Eq. (1) measures the processing needed to execute services locally on the device and the one needed waiting for remote computations and/or processing the response back on the terminal. Eq. (2) calculates the memory needed to process local computations and the one consumed while waiting and/or processing the response back. Eq. (3) is used to determine the energy consumed by the CPU and on the screen brightness for local processing as well as the energy spent on idle CPU, screen brightness and active network while waiting the execution of offloaded services. It also includes the energy consumed by the terminal for data transmission. Finally, Eq. (4) calculates the duration of local and remote processing taking into account the latency for data transmission.

5. Proposed algorithm based on dynamic programming

Dynamic programming (DP) [17] is a powerful technique that can be used to solve many problems, including optimization, for which a naive approach would take exponential time to finish the process. We present in this section our proposed dynamic programming algorithm, which aims to find the strategy to be applied for the VPs and their components, while meeting with the objective functions defined in previous section.

5.1. DP Table Filling

Since at the lower level, strategies should be determined for each service in the running personas, we need to generate a bit stream of N bits, where N is the number of components on the device. We use an $N + 1 * N + 1$ DP table to store the bit-streams showing which personas are to be switched off and which components to be offloaded, executed locally and those to be turned off. For the first step, a random bit stream of size N*3 (number of decision variables) is generated that determines a first solution. However, different **Rules apply when generating potential random solutions**:

- $\forall c_{i,i:1\to n}, \forall vp_{j,j:1\to m}$, If $I_{vp_j} = 0$ then $I_{c_i,vp_j} = 0$ and $x_{c_i,vp_j} = -$
- $\forall c_{i,i:1\to n}, \forall vp_{j,j:1\to m}$, If $I_{vp_j} = 1$ and $I_{c_i,vp_j} = 0$ then $x_{c_i,vp_j} = -$
- $\forall c_{i,i:1\to n}, \forall vp_{j,j:1\to m}$, If $I_{vp_j} = 1$ and $I_{c_i,vp_j} = 1$ then $x_{c_i,vp_j} = 0$ or 1(if c_i is offloadable)
- $\forall c_{i,i:1\to n}, \forall vp_{j,j:1\to m}$, I_{vp_i} is constant $\forall c_i \in vp_j$
- Based on the prioritization scheme in Table 1, the probability of generating a 0/1 bit is adapted.
- Based on whether c_i is offloadable or not, x_{c_i,vp_j} is generated.

Rules to fill the DP Table: This stream is assigned to the table such that 1s for x_{c_i,vp_j} are assigned to the next horizontal cell, and the 0s are assigned to the next vertical cell. If x_{c_i,vp_j} in the stream is 1, the starting cell is (1, 2) and if it is 0/−, the starting cell is (2, 1). This approach will avoid extra computations for common bit strings [18]. Since the first cell is left empty, and the stream size is N, we need $N + 1 * N + 1$ Table to fit all possible potential generated streams. According to the bits generated and to the defined rules to fill the columns and rows of this table, the process might/might not leave some cells with NULL values.

Example: A 2D 6*6 table is shown in Table 2. To clarify, assume that $N = 5$ (2 components in vp_1 and 3 components in vp_2) and the first random stream is 00 − 00 − 10 − 110111 (non bold vectors). Assume that the second random bit stream is 11111010 − 10 − 110. The starting cell of the second stream is (1, 2) since the third bit is 1. By following the aforementioned rules to fill the table, the resulting bold stream is shown in the table.

Whenever a bit stream is generated randomly, we calculate the consumed CPU, memory, energy and time of each cell (i.e., each component) in the table, and also at the same time calculate the total of each of these metrics of this bit stream, which formulate the defined objective functions. However, if a random bit stream is generated which has some common

Table 2
DP Table Filling.

NULL	[1,1,1]	NULL	NULL	NULL	NULL
[0,0,-]	**[1,1,0]**	NULL	NULL	NULL	NULL
[0,0,-]	**[1,0,-]**	NULL	NULL	NULL	NULL
[1,0,-]	**[1,0,-]**	NULL	NULL	NULL	NULL
[1,1,0]	[1,1,1]/**[1,1,0]**	NULL	NULL	NULL	NULL
NULL	NULL	NULL	NULL	NULL	NULL

cells with an existing string in the table; we replace that cell with the new value only if it's able to offer better trade-off with respect to the defined objective functions, for that cell, conforming with the weight of each metric. We then update the metrics of the remaining cells for the existing bit streams, based on the new values at this common cell. Every time a new stream is generated, we keep tracking the arrangement of the stream in Table 2.

When to terminate this process? Once a solution that meets with the device needs is generated with the least loss possible (i.e, least number of switched off personas and components). Therefore, we define the latter as the hamming distance between the I_{vp_j} and I_{c_i,vp_j} in the generated bit stream solution and $I_{vp_j} = 1$ and $I_{ci,vp_j} = 1$.

The full process described above is depicted in Algorithm 1.

Algorithm 1 Dynamic programming algorithm.

do
  setRC(*ds*, *frc*) ▷ Set the resource constraints based on the device state and predicted future context
  Initialize $T_{bitstreams}$
  generateRBS(*rules*) ▷ Generate a random bit stream that conforms with the rules
  findFB(*dptab*) ▷ Check the first bit to specify the starting cell in the table
  for $i = 1$ to N **do**
    positionB(b_i, *dptab*) ▷ Put each bit of the bit stream in the correct position in table
    calcsCPU(*cell*) ▷ Calculate self-cpu of each cell
    calcSMemory(*cell*) ▷ Calculate self-memory of each cell
    calcSEenergy(*cell*) ▷ Calculate self-energy of each cell
    calcSTime(*cell*) ▷ Calculate self-time of each cell
    calcObjFuncs() ▷ and Calculate their corresponding totals (Objective Functions)
    if this specific cell *cell* in table *dptab* is visited before **then**
      compare(*nscpu*, *oscpu*) ▷ Compare the new self-cpu of this cell with the previous one
      compare(*nsmemory*, *osmemory*) ▷ Compare the new self-memory of this cell with the previous one
      compare(*nsenergy*, *osenergy*) ▷ Compare the new self-energy of this cell with the previous one
      compare(*nstime*, *ostime*) ▷ Compare the new self-time of this cell with the previous one
      if *nscpu*, *nsmemory*, *nsenergy*, *nstime* of *cell* offer better trade-off than *oscpu*, *osmemory*, *osenergy*, *ostime* **then**
        Replace *oscpu* of this *cell* with *nscpu*
        Replace *osmemory* of this *cell* with *nsmemory*
        Replace *osenergy* of this *cell* with *nsenergy*
        Replace *ostime* of this *cell* with *nstime*
        Update the remaining amounts *updateDBTable*(*dptable*) in the remaining cells ▷ previous bit stream based on the new amount of this common cell
        calcCPU(*nbs*) ▷ Calculate the cpu of the remaining bits of the new bit stream
        calcMemory(*nbs*) ▷ Calculate the cpu of the remaining bits of the new bit stream
        calcEnergy(*nbs*) ▷ Calculate the cpu of the remaining bits of the new bit stream
        calcTime(*nbs*) ▷ Calculate the cpu of the remaining bits of the new bit stream
        Track the position of all bits in the table in a matrix
      else
        Keep previous totals for *cell*
        track(*dptab*) ▷ Track position of all bits in the table *dptab* in a matrix
      end if
    end if
  end for
  return bit stream *bs* with least loss and its corresponding F_{cpu}, F_{memory}, F_{energy}, F_{time}
while No feasible stream is found and $NF_{bitstreams} < T_{bitstreams}$

6. Evaluation

In the sequel, we first evaluate the accuracy of the machine learning techniques discussed in Section 3 and the one with the least error rate is adopted. According to the future context, predicted by the latter, we evaluate the efficiency of our proposed algorithm to find the adequate strategies that meet with the optimization objectives and comply with the future resource needs.

6.1. Setup

First, we evaluate LR, SVR, NN and DNN in two phases context-aware prediction:
Phase #1: Predict resource needs (CPU and Memory) for the running personas/apps

Table 3
Device usage behavior.

Time	# of VPs	VP(s)	Location	# of Components
8:00-10:00	1	Personal	Home	1–5
10:00-12:00	2	Personal + Clinic	Clinic	3–8
12:00-15:00	2	Personal + Hospital#1	Hospital#1	3–8
15:00-18:00	3	Personal + Hospital#1 + Hospital#2	Hospital#2	5–12
18:00-20:00	1	Personal	Home	1–5

Table 4
ML techniques setup for phase #1.

LR		SVR		NN		DNN	
Window size	1h	**Kernel**	Radial	**Hidden layers**	1	**Hidden layers**	2
Train	50 min	**Cross-validation**	10	**Nodes**	5	**Nodes**	5
Test	10 min	**Cost**	300	**Window size**	1h	**Threshold**	0.01
		Window size	1h	**Train**	50 min	**Window size**	1h
		Train	50 min	**Test**	10 min	**Train**	50 min
		Test	10 min			**Test**	10 min

Table 5
ML techniques setup for phase #2.

LR		SVR		NN		DNN	
Window size	4 days	**Kernel**	Radial	**Hidden layers**	1	**Hidden layers**	2
Train	4 days	**Cross-validation**	10	**Nodes**	4–7	**Nodes**	4–7
Test	1 day	**Cost**	300	**Window size**	4 days	**Threshold**	0.01
		Window size	4 days	**Train**	4 days	**Window size**	4 days
		Train	4 days	**Test**	1 day	**Train**	4 days
		Test	1 day			**Test**	1 day

- T_0: predict resource needs at T_1 for currently running personas/apps.

Phase #2: Predict future running/switched off personas/apps and their resource needs (CPU and Memory)

- T_0: predict behavior (Location, VPs and Components) at T_1 then resource needs accordingly.

For phase #1, we generate two data sets (DS_1 and (DS_2)) for doctor work shifts during one day from 8 am to 8 pm. Both data sets have 14,400 rows with data being generated each 3 seconds. The dataset consists of timestamp, which allows daily pattern recognition, coordinates to detect the location of the user (e.g., home, clinic, hospitals), number of VPs running, number of components, their CPU usage and memory consumption. The data set disregards the time spent to move from one location to another. The usage behavior in this data set is described in Table 3. DS_1 reflects normal behavior while DS_2 includes some peaks in the resource consumption to see how would that affect the prediction accuracy of these resources.

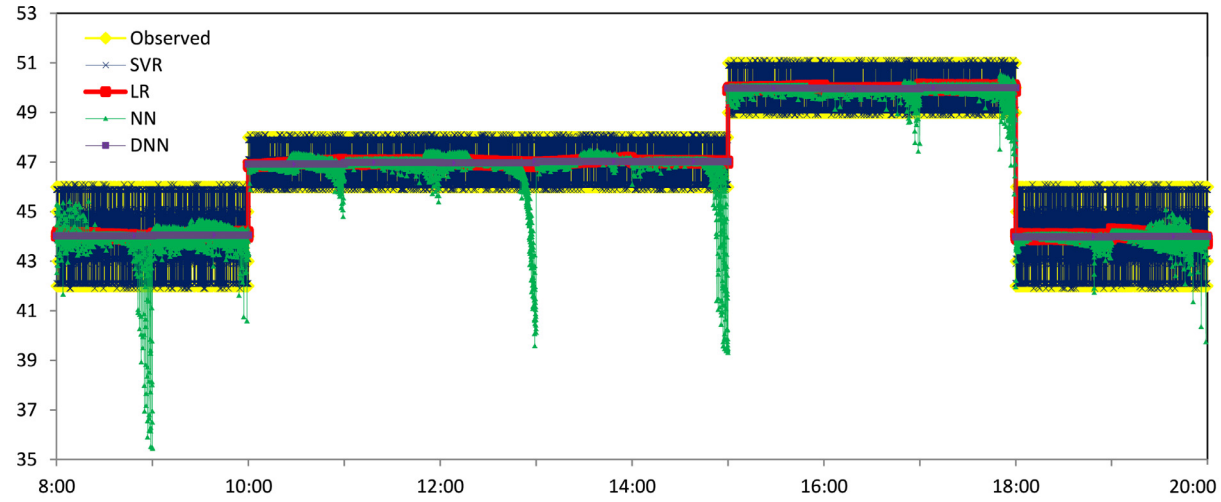
As for phase #2, we generate 4 datasets (DS_3, DS_4, DS_5 and DS_6) for daily prediction. Data is generated every 30 min to reduce the data size to be analyzed. Future Context, behavior and resource needs are all to be predicted in this phase. In DS_3, the same usage pattern is shown daily (day 1, 2, 3 and 4 and prediction applied on day 5) In DS_4, same usage pattern is shown daily (day 1, 2 and 4, Spikes on day 3 and prediction applied on day 5). In DS_5, similar daily usage pattern (day 1, 2, 3 and 4 and prediction applied on day 5). In DS_6, similar daily usage pattern (day 1, 2 and 4, Spikes on day 3 and prediction applied on day 5).

Various machine learning techniques are used and compared: LR=Linear regression model, SVR=support vector regression, NN=neural network and DNN=Deep Neural Network. Tables 4 and 5 show the setup of each machine learning technique studied in each prediction phase.

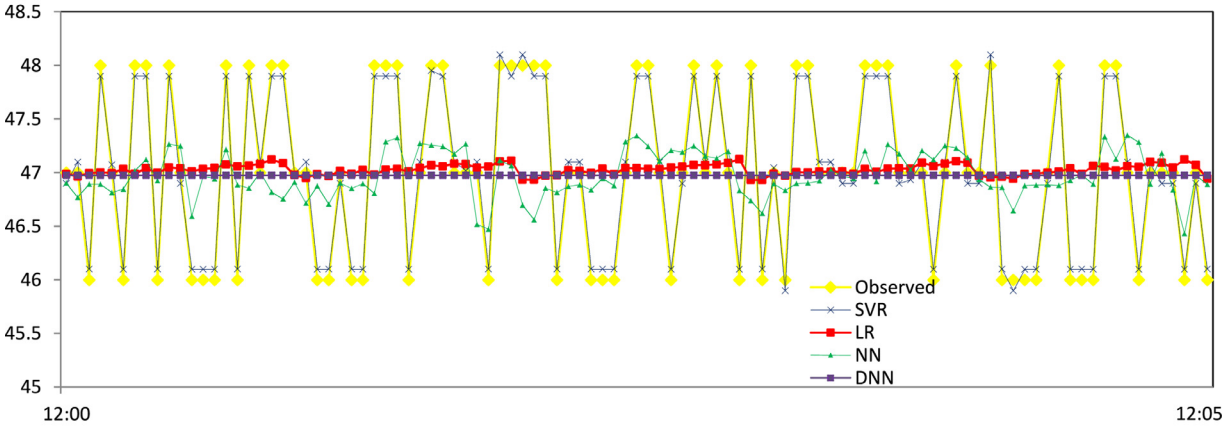
6.2. Numerical analysis

For each dataset in both phases #1 and #2, we examine the accuracy of LR, SVR, NN and DNN techniques to predict resource needs in terms of CPU and memory, to define the constraints of the formulated optimization model.

Fig. 2 shows the results of Dataset DS_1. Particularly, Fig. 2a depicts the CPU values over the day from 8:00am till 8:00pm, while in Fig. 2b, only a subset of these values is illustrated for the convenience of the reader when examining the accuracy of each technique. *Observed* values are the real data. Comparing the latter with those predicted by each machine learning technique, the results show that SVR outperforms the others with the highest accuracy (SVR predicted values almost matches the Observed values). In Fig. 2c, we analyze the error rate of these predictions based on the Root Mean Square Error metric (RMSE). SVR shows the minimum RMSE among the other techniques with only 0.098 error rate. The same analysis applies on the memory results depicted in Fig. 3. SVR shows the highest accuracy with 0.0982 RMSE.



(a) CPU values over the day (8:00-20:00)



(b) Subset of CPU values (12:00-12:05)

RMSE-CPU			
LR	**SVR**	**NN**	**DNN**
1.008667	0.098083	1.331917	1.020627

(c) RMSE

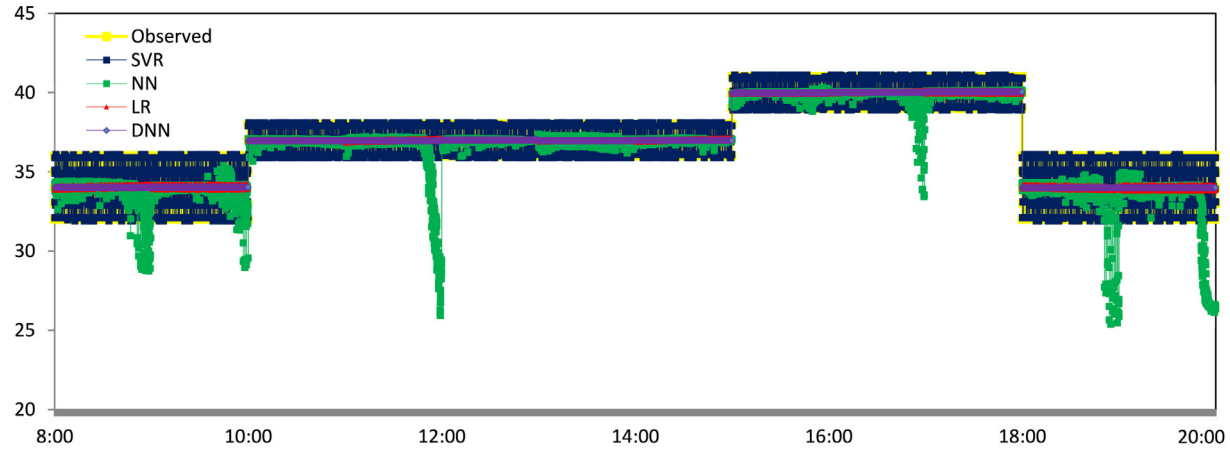
Fig. 2. DS1: observed CPU vs. predicted.

For DS_2, which includes some unusual behavior (peaks) in the resources consumption, SVR outperformed the other techniques as well for both CPU and memory needs prediction, showing the least error rate with 0.098083 and 0.09825 RMSE accordingly as depicted in Figs. 4 and 5.

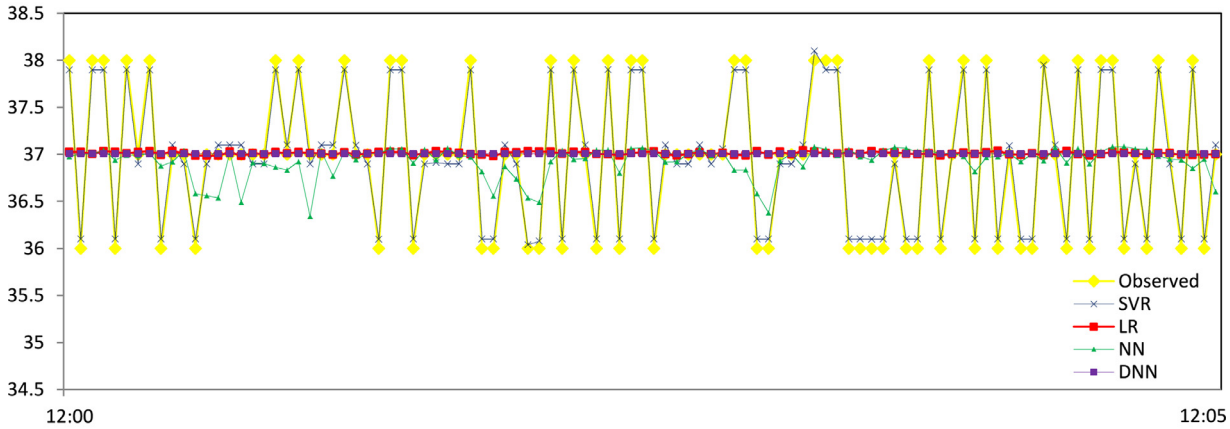
DS_3, DS_4, $DS_5 and DS_6$, all imply multi-stage prediction, which includes prediction of the location followed by the virtual phones and components and finally the resource needs by the latter.

When exactly the same context and usage pattern are repeated from day 1 to day 4, SVR was able to predict that the same applies on the fifth day with no errors detected in the predicted CPU and memory values as depicted in Fig. 6. However, when exactly the same usage pattern is shown daily (day 1, 2 and 4) with some spikes observed on day 3, the error rate for all the machine learning techniques has increased for the predicted values on day 5 as depicted in Fig. 7. In this case, LR, SVR, NN and DNN have showed error rates of 3.62, 1.79, 2.11 and 3.26 respectively for the CPU values and 3.91, 1.65, 1.44 and 3.37 RMSE for the memory values, while higher accuracy was observed in previous case (DS_3) with 2.37, 0, 1.89 and 2.38 RMSE respectively for CPU and 2.47, 0, 1.34 and 2.52 RMSE for the memory values.

The same analysis applies when comparing the results in Figs. 8 and 9, where similar, but not exactly the same, context and usage behavior are observed on daily basis for DS_5 and some spikes injected in DS_6.



(a) Memory values over the day (8:00-20:00)



(b) Subset of memory values (12:00-12:05)

RMSE-Memory			
LR	**SVR**	**NN**	**DNN**
1.015917	0.09825	1.623208	1.021837

(c) RMSE

Fig. 3. DS1: observed memory vs. predicted.

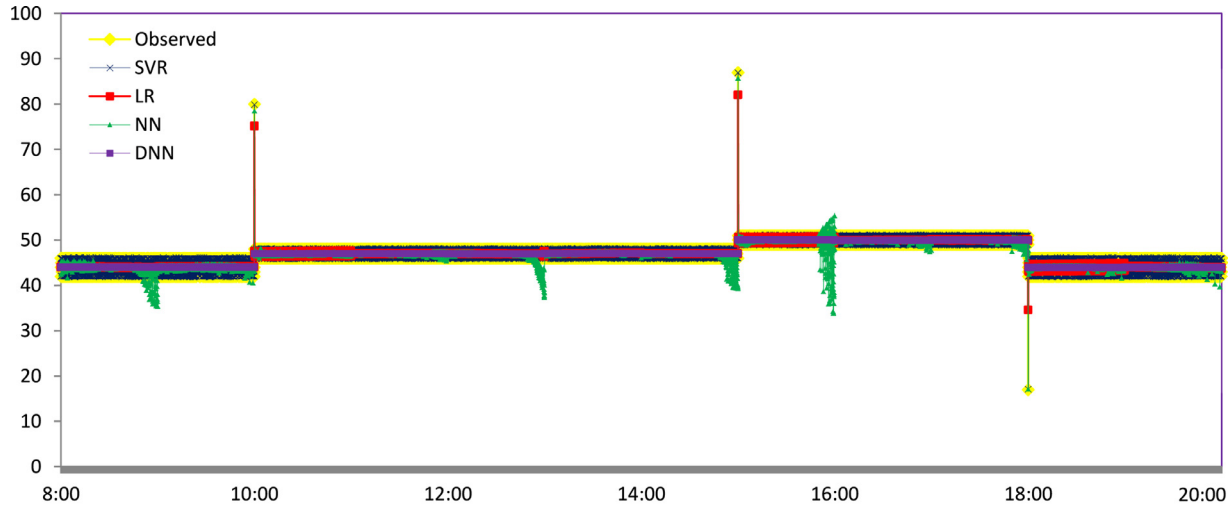
Table 6
Usage scenarios.

	Scenario-1	Scenario-1	Scenario-3	Scenario-4
Total VPs Running	1	3	2	3
Total Components Running	2	10	8	10
Predicted CPU Needs	10%	80%	60%	30%
Predicted Memory Needs	20%	70%	70%	30%

The next experiments are performed based on the values predicted by SVR since the latter had the highest accuracy in all previous conducted analysis. In what follows, we study the efficiency of the proposed optimization and dynamic programming algorithm to find efficient management strategies in different scenarios that reflect various usage pattern of the mobile device.

We compare the results with previous existing work, which adopts heuristics to find the strategy for each component. Table 6 illustrates the configuration of each scenario.

Table 7 shows the strategies generated for each scenario, by both heuristic and dynamic programming decision engines. To recall, a 0 bit in the heuristic approach is for local execution of the component while a bit of 1 is for offloading it. In

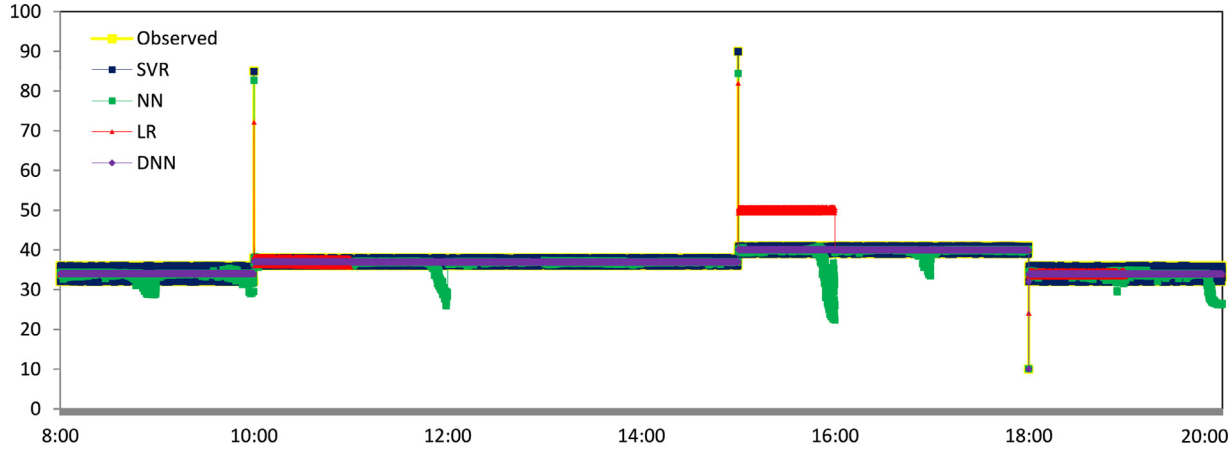


(a) CPU values over the day (8:00-20:00)

RMSE-CPU			
LR	SVR	NN	DNN
1.008667	0.098083	1.331917	1.020627

(b) RMSE

Fig. 4. DS2: observed CPU vs. predicted.



(a) Memory values over the day (8:00-20:00)

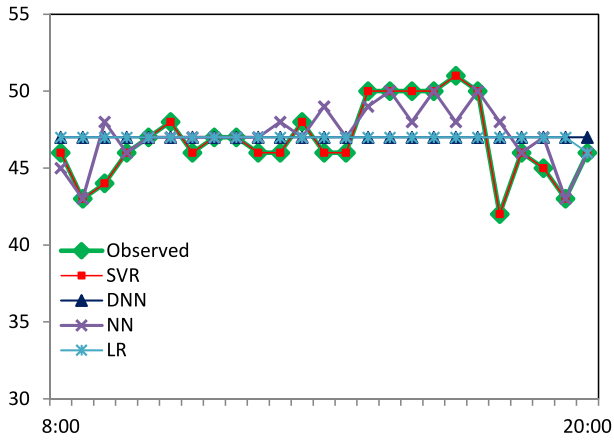
RMSE-Memory			
LR	SVR	NN	DNN
1.015917	0.09825	1.623208	1.021837

(b) RMSE

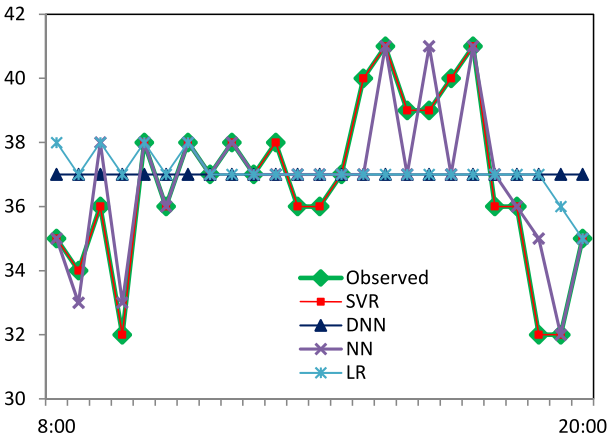
Fig. 5. DS2: observed memory vs. predicted.

Table 7
Generated strategies.

Scenario	Heuristic	DPDE
Scenario-1	1 1	[1,1,1] | [1,1,1]
Scenario-2	0 1 0 0 1 1 1 1 0 0	[1,0,-] || [1,1,1] | [1,1,1] | [1,1,1] | [0,0,-] | [0,0,-] | [0,0,-] | [0,0,-] | [0,0,-] | [0,0,-]
Scenario-3	0 0 1 0 0 1 0 0	[1,1,0] | [1,1,0] | [1,0,-] || [1,1,0] | [1,0,-] | [1,1,0] | [1,0,-] | [1,0,-]
Scenario-4	0 0 0 0 1 1 1 1 1 1	[1,0,-] || [0,0,-] | [0,0,-] | [0,0,-] || [0,0,-] | [0,0,-] || [0,0,-] | [0,0,-] | [0,0,-] | [0,0,-]



(a) CPU



(b) Memory

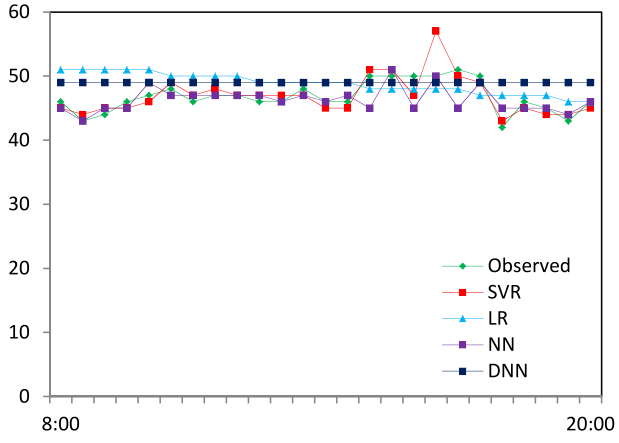
RMSE-CPU			
LR	SVR	NN	DNN
2.3749	0	1.8868	2.3833

(c) RMSE: CPU

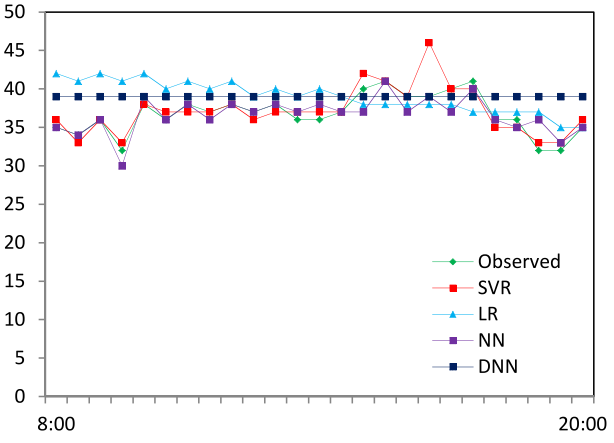
RMSE-Memory			
LR	SVR	NN	DNN
2.4739	0	1.3416	2.5298

(d) RMSE: Memory

Fig. 6. DS3: observed vs. predicted values.



(a) CPU



(b) Memory

RMSE-CPU			
LR	SVR	NN	DNN
3.616628	1.788854	2.107131	3.261901

(c) RMSE: CPU

RMSE-Memory			
LR	SVR	NN	DNN
3.908864	1.649242	1.442221	3.37046

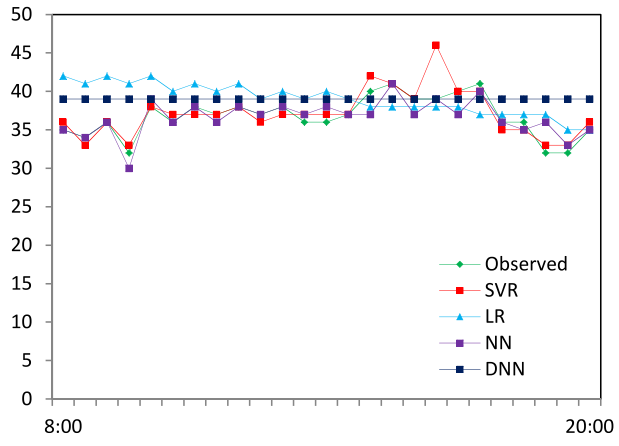
(d) RMSE: Memory

Fig. 7. DS4:Observed vs. Predicted values.

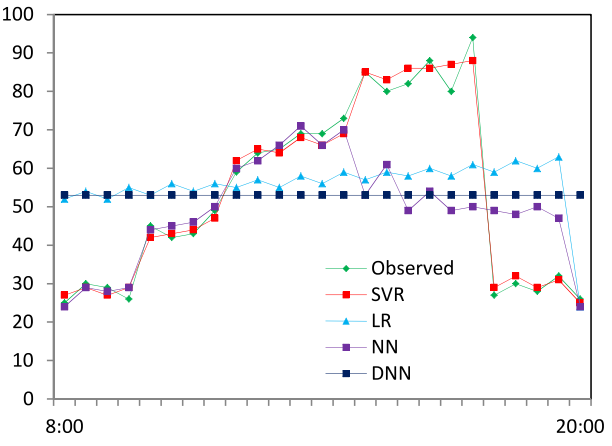
the dynamic programming decision engine, the '||' is to separate strategies between VPs and '|' to separate strategies of components in the same VP.

The resource consumption and performance of each of the applied strategies are compared in Fig. 10.

The figure shows that the algorithm proposed in this work is able to find better strategies with less CPU, memory and energy consumption and better performance compared to the heuristic approach. But how efficient are these strategies with respect to the predicted resource needs in each scenario? To answer this question, we examine Figs. 10a and 10 b. For Scenario-1, the results show that both approaches, i.e., Heuristic and DPDE are able to find a strategy that meets with



(a) CPU



(b) Memory

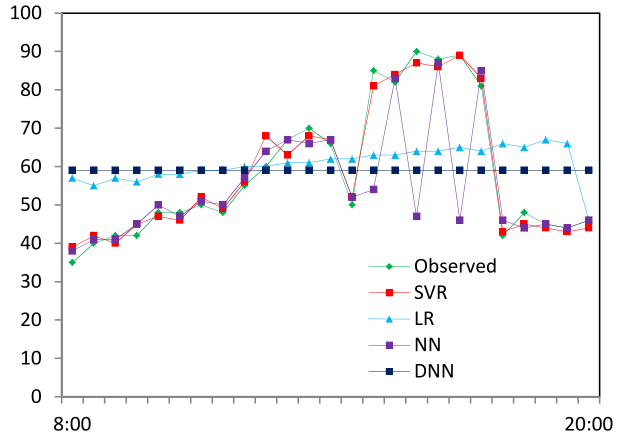
RMSE-CPU			
LR	SVR	NN	DNN
16.28619	2.481935	17.78089	17.5294

(c) RMSE: CPU

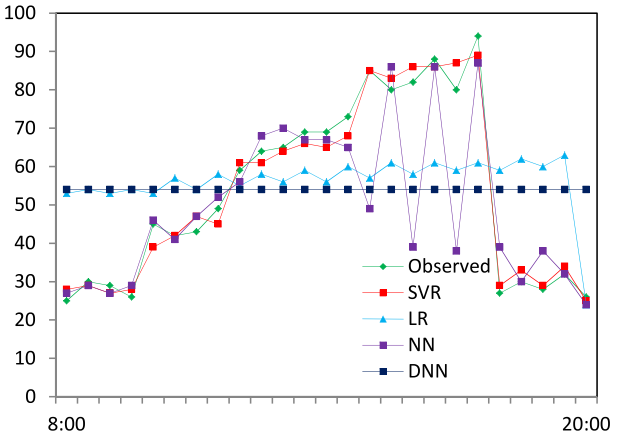
RMSE-Memory			
LR	SVR	NN	DNN
21.86413	2.792848	18.00889	23.11796

(d) RMSE: Memory

Fig. 8. DS5:Observed vs. Predicted values.



(a) CPU



(b) Memory

RMSE-CPU			
LR	SVR	NN	DNN
16.28373	2.727636	13.82751	17.5294

(c) RMSE: CPU

RMSE-Memory			
LR	SVR	NN	DNN
21.72464	3.298485	14.71462	23.11363

(d) RMSE: Memory

Fig. 9. DS6:Observed vs. Predicted values.

both CPU and memory requirements. Particularly, The strategies found by both approaches imply 7.83% CPU and 13.634% memory which guarantees the availability of the needed 10% and 20% of CPU and memory respectively. For Scenario-2, 80% available CPU and 70% available memory are required as depicted in Table 6. The strategy found by the heuristic approach consumes 56.96% CPU and 63.39% memory which does not meet with the future resource needs, yet with DPDE, the strategy found only consumes 15.17% of the CPU and 20.34% of the memory, which keeps more available resources that meet with the resource required in future context. The same applies on Scenario-3. Finally for Scenario-4, the results show that both approaches are able to find good strategies that meet with the device needs yet with better results of the management

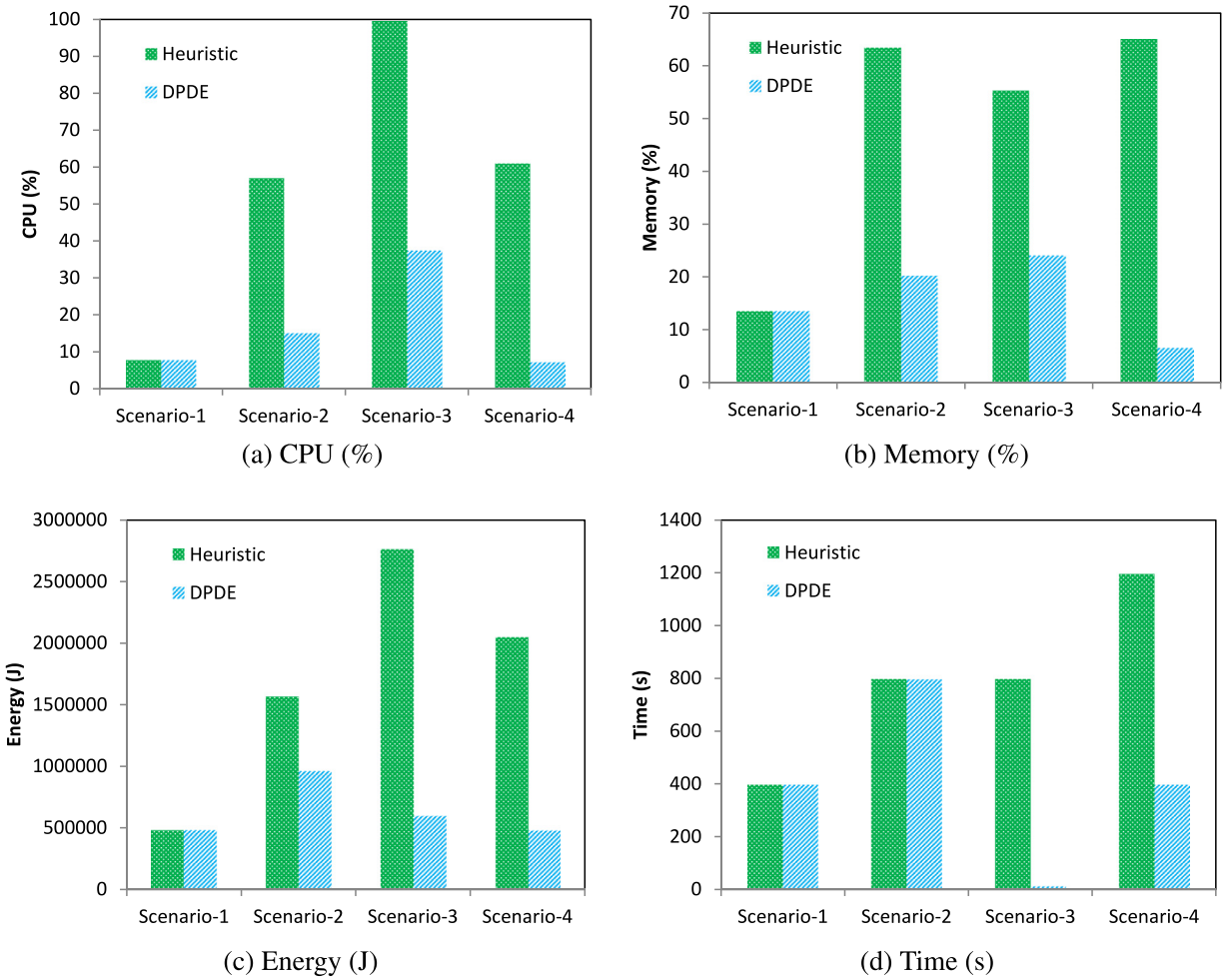


Fig. 10. Strategies efficiency.

strategy generated by DPDE, which guarantees more resource availabilities of 92.67% CPU and 93.3% of the memory usage. These results prove the efficiency of the advanced management strategies proposed in this work to manage predicted future resource needs.

Finally, we study the performance of heuristic and DPDE algorithms in terms of execution time. Fig. 11 shows that for all scenarios in question, DPDE shows faster execution.

7. Related works

We survey in this section predictive strategies for virtual machines management, recent techniques for mobile computations offloading, and algorithms for dynamic offloading decisions.

7.1. Predictive virtual instances management strategies

Sharing an end terminal between several virtual machines raises many problems and autonomic load balancing of resources is one of these open key challenges to be resolved. In this context, different approaches have been proposed for load prediction on a physical machine. Predicting future load enables proactive consolidation of VMs on the overloaded and under-loaded physical machines. In [19], the authors have proposed regression methods to predict CPU utilization of a physical machine. The work uses K-nearest neighbor (KNN) regression algorithm to approximate a function based on the data collected during the lifetimes of the VMs. The formulated function is then used to predict an overloaded or an under-loaded machine. Bala et al. [20] have proposed a proactive load balancing approach that based on prior knowledge of the resource utilization parameters, applies machine learning techniques to predict future resource needs. Various techniques have been studied in their work, such as KNN, Artificial Neural Networks (ANN), Support Vector Machines (SVM) and Random Forest and the one with maximum accuracy has been utilized as prediction-based approach. Xiao et al. [18] have also used a load

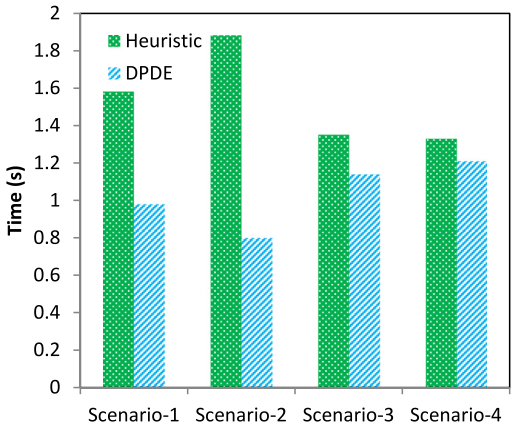


Fig. 11. Decision engine performance.

prediction algorithm to capture the rising trend of resource usage patterns and help identifying hot spots and cold spots machines. After predicting the resource needs, Hot spot and cold spot machines are identified. When the resource utilization of any physical machine is above the hot threshold, the latter is marked as hotspot. If so, some VMs running on it will be migrated away to reduce its load [18]. On the other hand, cold spot machines either idle or having the average utilization below particular threshold, are also identified. If so, some of those physical machines could potentially be turned off to save energy [18,21].

7.2. Dynamic offloading algorithms

A Dynamic Programming (DP) algorithm was proposed in [22], where a two dimensional DP table was used in order to determine what to offload. However, a backtracking algorithm was needed to find the final decisions, which was time consuming. A dynamic offloading algorithm based on Lyapunov optimization was presented in [23]. The algorithm is based upon a relationship between the current solution and the optimal solution requiring a considerable amount of execution time and many iterations to converge upon a solution [24]. A semi-definite relaxation approach for the offloading problem was presented by Chen et al., [25]. Their work considered a mobile cloud computing scenario consisting of one nearby computing access point, and a remote cloud server(s). Their proposition is based on an algorithm that solves a linear program through randomization and relaxation to generate an integer solution. The algorithm can find a near-optimal solution when using about 100 trials of relaxation. In [24], a dynamic programming algorithm called DPH is proposed. The algorithm generates periodically random bit strings of 0s and 1s, for remote and local execution of tasks, and utilize sub-strings when they improve the solution. The algorithm can find a nearly optimal solution after several iterations. The authors use a Hamming distance criterion to terminate the search process and hence obtain the final decision quickly. The stopping criterion is met when a given fraction of tasks are offloaded.

7.3. Analysis

The main highlights of this work are pro-activity, new management strategies for multi-persona mobile computing and accelerated decision making. In the context of offloading, none of the existing works have already addressed the raised multi-persona mobile computing problems, none has presented a relevant proactive intelligent decision maker, or even proposed similar management strategies as those presented in this manuscript. Thinking about the multiple VPs case as the load balancing problem of virtual machines in the cloud, we propose in this work an analogous proactive solution with advanced manageability strategies. Our proposition includes first a prediction-based proactive approach using machine learning techniques, which have been found suitable from the literature review discussed above. As pro-activity requires a prior knowledge of the device context and resource utilization, we devote a set of profilers to log the relevant data. Gathered data are trained using machine learning algorithms and predictions about future context and resource needs are made. Various machine learning techniques are evaluated, namely, LR [13], Support Vector Regression (SVR) [14], Neural Network (NN) [15] and Deep Neural Network (DNN) [16], and the one found to be with highest accuracy (SVR) has been utilized as prediction-based technique. Moreover, advanced management solution is proposed, which includes different strategies; namely, offloading and turning off tasks and switching off personas. To help adopting such strategies, we propose hotposts and coldspots detectors along with a classification scheme to categorize and prioritize tasks and VPs. Based on the predicted context and resource needs, identified hotspot and coldspot tasks, prioritization, device state and network connection properties, a multi-objective optimization model is formulated. Finally, the model is solved with a dynamic programming algorithm that finds the adequate strategies to be applied for each task and VP aiming to attain the objective functions in

the formulated model. The experimental results clearly demonstrate the effectiveness of this work, showing better resource management solution with improved performance.

8. Conclusion and future directions

Managing virtual phones running on an end physical mobile device with limited resources is challenging. In this context, we proposed in this work novel approach able to predict future context and resource needs of these VPs and apply advanced management strategies accordingly, aiming to avoid any performance degradation or system crash in the system. Various machine learning techniques are studied and the one with the highest accuracy is adapted to meet the problem requirements. Additionally, new management strategies are proposed and novel algorithm based on dynamic programming is presented to generate the adequate strategies that meet with the resource needs according to different usage scenarios. Thorough analysis was conducted to study the efficiency of this proposition. The results proved the efficiency of the predictor, the adequacy of the new strategies proposed and the competency of the algorithm performance. As for the research community, studying the effect and overhead of training the model and performing the prediction on the end terminal would be interesting. Also, examining the effect of the window size value on the accuracy of the prediction model and proposing a dynamic generic approach to adapt this value to different data sets would be a valuable future research track.

Declaration of Competing Interest

The work has not been already presented, published or is now under consideration elsewhere.

Acknowledgments

The work has been supported by École de Technologie Supérieure (ETS), NSERC Canada, the Associated Research Unit of the National Council for Scientific Research CNRS Lebanon and the Lebanese American University (LAU).

References

[1] Rouse M.. BYOD (bring your own device). 2012a. Accessed: 2019-05-10; http://whatis.techtarget.com/definition/BYOD-bring-your-own-device.
[2] Cellrox. Cellrox Thinvisor. Accessed: 2019-05-10; http://www.cellrox.com/product.
[3] Rouse M.. dual persona (mobile device management). 2012b. Accessed: 2019-05-10; http://searchconsumerization.techtarget.com/definition/Dual-persona.
[4] Andrus J, Dall C, Hof AV, Laadan O, Nieh J. Cells: a virtual mobile smartphone architecture. In: Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles. ACM; 2011. p. 173–87.
[5] Tout H, Talhi C, Kara N, Mourad A. Towards an offloading approach that augments multi-persona performance and viability. In: 2015 12th Annual IEEE Consumer Communications and Networking Conference (CCNC); 2015. p. 455–60. doi:10.1109/CCNC.2015.7158018.
[6] Tout H, Talhi C, Kara N, Mourad A. Selective mobile cloud offloading to augment multi-persona performance and viability. IEEE Trans Cloud Comput 2019;7(2):314–28. doi:10.1109/TCC.2016.2535223.
[7] Kemp R. Programming frameworks for distributed smartphone computing. Vrije Universiteit; 2014. Ph.D. thesis.
[8] Alameddine HA, Sharafeddine S, Sebbah S, Ayoubi S, Assi C. Dynamic task offloading and scheduling for low-latency iot services in multi-access edge computing. IEEE J Sel Areas Commun 2019;37(3):668–82.
[9] Dbouk T, Mourad A, Otrok H, Tout H, Talhi C. A novel ad-hoc mobile edge cloud offering security services through intelligent resource-aware offloading. IEEE Trans Netw ServManage 2019. doi:10.1109/TNSM.2019.2939221. 1–1.
[10] Kodratoff Y. Introduction to machine learning. Elsevier; 2014.
[11] Android. The activity lifecycle. 2016. Accessed: 2019-05-10; https://developer.android.com/guide/components/activities/activity-lifecycle.html.
[12] Tout H, Talhi C, Kara N, Mourad A. Smart mobile computation offloading: centralized selective and multi-objective approach. Expert Syst Appl 2017;80:1–13.
[13] Seber GA, Lee AJ. Linear regression analysis, 329. John Wiley & Sons; 2012.
[14] Drucker H, Burges CJ, Kaufman L, Smola AJ, Vapnik V. Support vector regression machines. In: Advances in neural information processing systems; 1997. p. 155–61.
[15] Demuth HB, Beale MH, De Jess O, Hagan MT. Neural network design. 2nd. USA: Martin Hagan; 2014. ISBN 0971732116, 9780971732117.
[16] Goodfellow I, Bengio Y, Courville A. Deep learning. MIT Press; 2016. http://www.deeplearningbook.org.
[17] Bertsekas DP, Bertsekas DP, Bertsekas DP, Bertsekas DP. Dynamic programming and optimal control, 1. Athena scientific Belmont, MA; 1995.
[18] Xiao Z, Song W, Chen Q. Dynamic resource allocation using virtual machines for cloud computing environment. IEEE Trans Parallel Distrib Syst 2012;24(6):1107–17.
[19] Farahnakian F, Pahikkala T, Liljeberg P, Plosila J. Energy aware consolidation algorithm based on k-nearest neighbor regression for cloud data centers. In: Proceedings of the 2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing. IEEE Computer Society; 2013. p. 256–9.
[20] Bala A, Chana I. Prediction-based proactive load balancing approach through vm migration. Eng Comput 2016;32(4):581–92.
[21] Beloglazov A, Buyya R. Energy efficient allocation of virtual machines in cloud data centers. In: 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing. IEEE; 2010. p. 577–8.
[22] Liu Y, Lee MJ. An effective dynamic programming offloading algorithm in mobile cloud computing system. In: 2014 IEEE Wireless Communications and Networking Conference (WCNC). IEEE; 2014. p. 1868–73.
[23] Huang D, Wang P, Niyato D. A dynamic offloading algorithm for mobile computing. IEEE Trans Wirel Commun 2012;11(6):1991–5.
[24] Shahzad H, Szymanski TH. A dynamic programming offloading algorithm for mobile cloud computing. In: 2016 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE). IEEE; 2016. p. 1–5.
[25] Chen M-H, Liang B, Dong M. A semidefinite relaxation approach to mobile cloud offloading with computing access point. In: 2015 IEEE 16th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC). IEEE; 2015. p. 186–90.

Hanine Tout received the Ph.D. degree in software engineering from cole de Technologie Suprieure (TS), Montreal, Canada. She is a Postdoc Fellow between ETS and Ericsson, Canada, where she is leading two industrial projects in the areas of AI, federated learning, machine learning, security, 5G and cloud-native IMS. She is a TPC member and reviewer of prestigious conferences and journals.

Nadjia Kara received her Ph.D., in Electrical Engineering from Ecole Polytechnique of Montreal. She is an associate professor in software engineering and IT at TS and an affiliate professor at Concordia, Montreal, Canada. She spent around 10 years in industry working as researcher and system architect. Her research interests include network and service architectures, next generation networks, distributed systems and traffic engineering.

Chamseddine Talhi received the Ph.D degree in computer science from Laval University, Quebec, Canada. He is an associate professor in the department of software engineering and IT at ETS, University of Quebec, Montreal, Canada. He is leading a research group that investigates Smartphone, embedded systems and IoT security. His research interests include cloud security and secure sharing of embedded systems.

Azzam Mourad received his Ph.D. degree in electrical and computer engineering from Concordia University. He is an associate professor of computer science at the Lebanese American University. He served/serves as Associate Editor for IET Quantum Communication and IEEE Communications Letters, and General-Chair, Track Chair, TPC member and reviewer of several prestigious conferences and journals. He is an IEEE senior member.