Expert Systems With Applications 80 (2017) 1–13

Contents lists available at ScienceDirect

Expert Systems With Applications

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

Smart mobile computation offloading: Centralized selective and multi-objective approach

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

[a] Department of Software Engineering and Information Technologies, École de Technologie Superieure, Montreal, Canada
[b] Department of Computer Science and Mathematics, Lebanese American University, Beirut, Lebanon

ARTICLE INFO

Article history:
Received 10 November 2016
Revised 2 January 2017
Accepted 3 March 2017
Available online 9 March 2017

Keywords:
Mobile device
Mobile cloud computing
Computation offloading
Selective offloading
Hotspots
Optimization

ABSTRACT

Although mobile devices have been considerably upgraded to more powerful terminals, yet their lightness feature still impose intrinsic limitations in their computation capability, storage capacity and battery lifetime. With the ability to release and augment the limited resources of mobile devices, mobile cloud computing has drawn significant research attention allowing computations to be offloaded and executed on remote resourceful infrastructure. Nevertheless, circumstances like mobility, latency, applications execution overload and mobile device state; any can affect the offloading decision, which might dictate local execution for some tasks and remote execution for others. We present in this article a novel system model for computations offloading which goes beyond existing works with smart centralized, selective, and optimized approach. The proposition consists of (1)hotspots selection mechanism to minimize the overhead of the offloading evaluation process yet without jeopardizing the discovery of the optimal processing environment of tasks, (2)a multi-objective optimization model that considers adaptable metrics crucial for minimizing device resource usage and augmenting its performance, and (3)a tailored centralized decision maker that uses genetics to intelligently find the optimal distribution of tasks. The scalability, overhead and performance of the proposed hotspots selection mechanism and hence its effect on the decision maker and tasks dissemination are evaluated. The results show its ability to notably reduce the evaluation cost while the decision maker was able in turn to maintain optimal dissemination of tasks. The model is also evaluated and the experiments prove its competency over existing models with execution speedup and significant reduction in the CPU usage, memory consumption and energy loss.

© 2017 Elsevier Ltd. All rights reserved.

1. Introduction

While smartphone usage continues in a fast-paced mode, developers are provisioning more advanced applications toward all-in-one computing device. However, no matter how sophisticated smartphones are growing, their hardware is still limited in terms of processing power, memory capacity and battery lifetime, compared to their counterparts of desktop machines. With the advancements in wireless communications and the abundance of cloud computing resources, many computations offloading techniques (Chae et al., 2014; Chen, Ogata, & Horikawa, 2012; Chun, Ihm, Maniatis, Naik, & Patti, 2011; Cuervo et al., 2010; Flores, Srirama, & Buyya, 2014; Gordon, Jamshidi, Mahlke, Mao, & Chen, 2012; Hung, Shieh, & Lee, 2012; Kemp, 2014; Kosta, Aucinas, Hui, Mortier, & Zhang, 2012; Shi et al., 2014; Xia et al., 2014) have been proposed allowing mobile devices to migrate the execution and throw the burdens of computations to remote infrastructure. Typically, either full application or fine-grained tasks like services, methods, or threads, are migrated to be executed on remote server, releasing the mobile device from intensive processing. The offloading decision is based upon number of factors. The network bandwidth and latency, the resource demands of tasks and the mobile device state all form a context that influences offloading efficiency. According to these aspects, a decision engine analyzes the cost of both local and remote execution and dictates what tasks to be offloaded correspondingly. Though the success of offloading to enhance performance, decrease energy loss and augment resource availabilities, offloading evaluation has its own consequences on the mobile terminal.

Multitasking is a trending action on the mobile operating systems, where multiple applications run simultaneously on the device to meet with the mobile user demands in daily life (Xiang, Ye, Feng, Li, & Li, 2014). However, when more than one application are running on the mobile terminal, offloading cannot be evaluated in-

* Corresponding author.
E-mail addresses: hanine.tout.1@ens.etsmtl.ca (H. Tout), chamseddine.talhi@etsmtl.ca (C. Talhi), nadjia.kara@etsmtl.ca (N. Kara), azzam.mourad@lau.edu.lb (A. Mourad).

http://dx.doi.org/10.1016/j.eswa.2017.03.011
0957-4174/© 2017 Elsevier Ltd. All rights reserved.

dependently for each, as migrating one application or running it locally would affect the execution of the others. Existing offloading techniques necessitate invoking the decision engine for each application separately, which make them unable to efficiently handle such circumstances. In addition, analyzing local and remote execution costs is an iterative process, therefore running such decentralized evaluation imposes in turn significant overhead and forms itself a bottleneck on the end terminal. Although taking the decision remotely might overcome such problem, yet it requires sending relevant data to remote server which imposes more overhead, besides the additional monetary fees to be carried out. Further, existing works assume offloading to be productive whenever remotely executing an application is able to save energy without degrading its normal response time (Flores et al., 2015). However, taking an offloading decision is more complex and additional metrics have to be considered. There is already a lot of understanding in the literature regarding the impact of communication latency and bandwidth, code execution, energy consumption, execution time, CPU and memory in the offloading process, yet existing decision models are limited such that none of them considers all these aspects and tries to reach a tradeoff among them.

This article emphasizes theoretically and experimentally on these limitations and advances relevant prominent solutions. We propose in this work new system model for computations offloading that differs from existing approaches in different aspects and further contributions. This proposition includes first a novel selective mechanism to reduce the search space in offloading evaluation. The mechanism considers only frequently invoked, resource-intensive and time consuming tasks, called hotspots, input for the decision model, in order to reduce the overhead of the decision engine. The work also presents a decision model that considers all of the connectivity properties, energy consumption, CPU and memory usages and execution time of tasks in order to refine the execution environment of tasks. We emphasized in previous work (Tout, Talhi, Kara, & Mourad, 2016) the effect of such metrics on the device performance and system survivability. We refine the optimization model in this work to make it resilient not only to the device state but also to the detected hotspots that vary with the device usage, and to strategies that can be enforced on the device through the proposed system to control offloading prioritization and execution suspension of tasks. We also redesign a genetic-based algorithm with tailored adaptive fitness evaluation for intelligent offloading decision making process. The evaluation is centralized to collectively evaluate offloading tasks from different applications running on the mobile terminal. The results of our experiments demonstrate the capability and highlight the efficiency of our proposition. In the following, we use the terms computations, tasks, and components interchangeably.

The originality and novelty of this work are emphasized by the following contributions:

- Novel system model for computations offloading which goes beyond existing techniques with selective, centralized and optimized approach.
- A selective mechanism that minimizes the search space and significantly reduces the overhead of offloading decision evaluation.
- An optimization model with metrics crucial to the device resources and applications performance, adaptable to resource usage, detected hotspots and execution strategies.
- An intelligent centralized decision engine which evaluates the optimization model through tailored genetic-based algorithm able to reach optimal dissemination of tasks with minimal evaluation cost.

The rest of the article is structured as follows. We review in Section 2 the basic mobile computation offloading architecture including the role of each component. In Section 3, we discuss existing computation offloading strategies while highlighting our contributions. We emphasize on the problems in Section 4 and present insights about our approach in Section 5. We detail our proposition in Sections 6–8. In Section 9, we evaluate our proposition and finally in Section 10, we conclude the article and draw some future directions.

2. Computations offloading overview

Mobile cloud computing has integrated cloud computing capabilities into the mobile environment to support mobile devices, ranging from outsourcing software and platforms all the way to infrastructure. In this context, different offloading concepts have been studied. The explosive growth of mobile internet applications, like social networking services, online gaming, audio and video streaming, is the main reason behind the significant overload on the cellular networks. In this context, traffic offloading (Andreev, Pyattaev, Johnsson, Galinina, & Koucheryavy, 2014; Fiandrino, Kliazovich, Bouvry, & Zomaya, 2015; Han et al., 2010) has been proposed, which is the use of complementary networks like Wi-Fi for data transmission in order to reduce the data carried on the cellular network. On the other hand, the limited resources of mobile devices have triggered another research domain of computation offloading, subject of this work, which has different concept and objective compared to traffic offloading. Code offloading is an opportunistic process that leverages cloud resources (e.g., servers) to execute computation-intensive components designated by a mobile terminal. In this process, an offloading decision is taken based on a cost model that can estimate where the execution is more effective for the end device. Due to mobility and changes in the network conditions, device resources, and computation requirements, the evaluation of this model changes from one execution to another and hence lead to different decisions. Whether the communication between the mobile terminal and the cloud resources is done through the cellular network or Wifi hotspots, the mobile device user is the one responsible for the additional cost imposed by using these channels, which does not raise any economic conflict.

The common architecture of computation offloading is depicted in Fig. 1. Set of profilers are installed on the terminal to monitor the mobile applications, the environment characteristics, and the device state. The mobile also contains a solver (i.e., decision maker) that based on the information gathered by the profilers, evaluates a cost model and generates an efficient distribution of components (i.e., decides about portions to be executed locally and those to be offloaded). On the other hand, the cloud infrastructure offers the servers where the offloaded components are to be executed. Hereafter, we describe each component of this architecture. The mobile terminal includes *profilers* to monitor different aspects. The program profiler is responsible of monitoring multiple parameters of the component (C) which is candidate for offloading, like energy consumption, execution time and size of data to be transmitted. The component, which is the offloading unit, can be a service, method or thread inside the application or even a full app. Different methods can be used to identify an offloading candidate, which is also called code partitioning. For instance, some approaches (Cuervo et al., 2010; Kemp, 2014; Kosta et al., 2012) rely on the developers to statically annotate explicitly the application source code (e.g., [Remoteable], strategy=remote, @Remote), while others (Chun et al., 2011) provide automatic mechanism capable of analyzing the code and generating potential migration points. The network profiler is responsible of monitoring 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 and CPU utilization to predict critical situations that

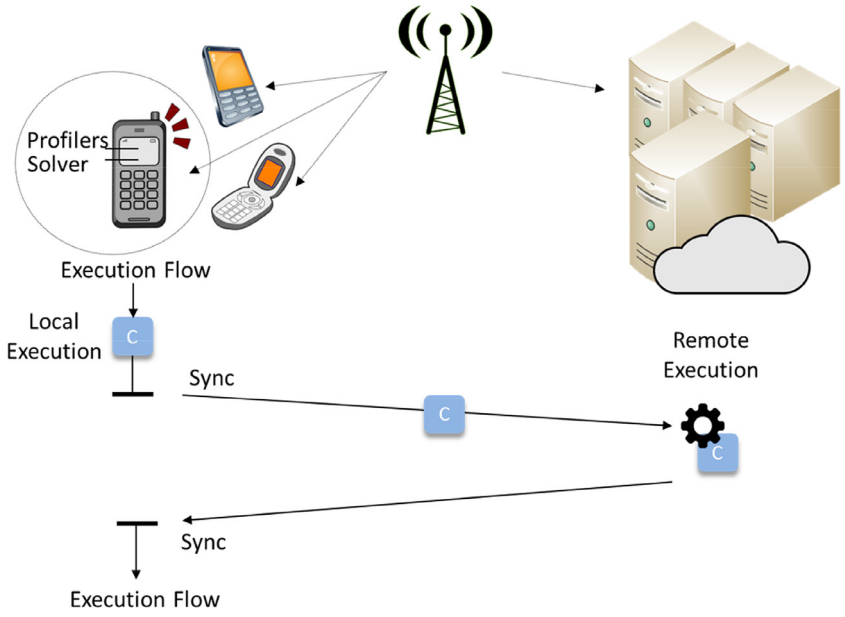

Fig. 1. Mobile code offloading architecture.

require offloading, and hence trigger the solver. Based on a cost model, a *solver* evaluates the information gathered by the profilers. It compares the benefit of local and remote execution and a decision is taken accordingly. If offloading is more beneficial, the code is invoked remotely; otherwise, it is executed locally. The *remote platform* consists of server(s) having higher processing power and more resource competency compared to mobile devices, located in the vicinity like cloudlets (Satyanarayanan, Bahl, Caceres, & Davies, 2009) or in the cloud (Amazon, 2016; Google, 2016), which are responsible of executing the offloaded code.

3. Related work

In this section, we review existing offloading approaches and we classify them based on different key factors to distinguish our proposition and highlight its contributions.

Cuervo et al. (2010) have proposed MAUI, an offloading framework that aims to reduce the energy consumption of mobile applications. The framework consists of a proxy server responsible of communicating the method state, a profiler that can monitor the device, program and network conditions, and a solver that can decide whether to run the method locally or remotely. MAUI uses its optimization framework to decide which method to send for remote execution based on the information gathered by the profiler. The results show the ability of MAUI to minimize the energy consumption of a running app.

CloneCloud is another offloading approach that has been presented by Chun et al. (2011) in order to minimize the energy consumption and speedup the execution of the running application. A profiler collects the data about the threads running in this app and communicates the gathered data with an optimization solver. Based on cost metrics of execution time and energy, the solver decides about the best partitioning of these threads between local and remote execution. This approach does not require modification in the original application since it works at the binary level. The experiments of CloneCloud showed promising results in terms of minimizing both execution time and energy consumption of an application. However, only one thread at a time can be encapsulated in a VM and migrated for remote execution, which diminishes the concurrency of executing the components of an application.

Gordon et al. (2012) proposed COMET that rely on distributed shared memory (DSM) systems and virtual machine (VM) synchronization techniques to enable multithreaded offloading and overcomes the limitations of MAUI and CloneCloud, which can offload one method/thread at a time. To manage memory consistency, a field-level granularity is used, reducing the frequency of required communication between the mobile device and the cloud.

Following different strategy, Kemp (2014) has proposed Cuckoo that assumes compute intensive code to be implemented as an Android service. The framework includes sensors to decide, at runtime, whether or not to offload particular service since circumstances like network type and status and invocation parameters of the service call on mobile devices get changed continuously, making offloading sometimes beneficial but not always. Cuckoo framework has been able to reduce the energy consumption and increase the speed of computation intensive applications.

Chen et al. (2012) have proposed a similar framework that automatically offloads heavy back-end services of a regular standalone Android application in order to reduce the energy loss and execution time of an application. Based on a decision model, the services are offloaded to an Android virtual machine in the cloud.

ThinkAir has been introduced by Kosta et al. (2012) as a technique to improve both computational performance and power efficiency of mobile devices by bridging smartphones to the cloud. The proposed architecture consists of a cloud infrastructure, an application server that communicates with applications and executes remote methods, a set of profilers to monitor the device, program, and network conditions, and an execution controller that decides about offloading. ThinkAir applies a method-level code offloading. It parallelizes method execution by invoking multiple virtual machines (VMs) to execute in the cloud in a seamless and on-demand manner achieving greater reduction in execution time and energy consumption.

Considering user delay-tolerance threshold, new offloading-decision making algorithm has been proposed by Xia et al. (2014). The proposed tool predicts the average execution time and energy of an application when running locally on the device, then compares them to cloud-based execution cost in order to decide where the application should be executed.

CMcloud is a new scheme that aims to maximize the throughput or minimize the server cost at cloud provider end by running

Table 1
Classification of offloading approaches.

Approach	Offloading Unit	Mobile Cost Model Metrics				Model Evaluation	Evaluation Overhead	Generated Dissemination Savings			
		Energy	Time	Processing	Memory			Energy	Time	Processing	Memory
Cuervo et al. (2010)	Method	✓	x	x	x	Independent	High	O/S	SO/S	SO/S	SO/S
Chun et al. (2011)	Thread	✓	✓	x	x	Independent	High	O/US	O/US	O/US	O/US
Gordon et al. (2012)	Multi-Thread	–	–	–	–	–	–	O/US	O/US	O/US	O/US
Kemp (2014)	Service	✓	✓	x	x	Independent	High	O/US	O/US	O/US	O/US
Chen et al. (2012)	Service	✓	✓	x	x	Independent	High	O/US	O/US	O/US	O/US
Kosta et al. (2012)	Method	✓	✓	x	x	Independent	High	O/US	O/US	O/US	O/US
Xia et al. (2014)	Method	✓	✓	x	x	Independent	High	O/US	O/US	O/US	O/US
Chae et al. (2014)	Method	x	✓	x	x	Independent	High	SO/S	O/S	SO/S	SO/S
Shi et al. (2014)	Method	x	✓	x	x	Independent	High	SO/S	O/S	SO/S	SO/S
Tout et al. (2016)	Generic	✓	✓	✓	✓	Collective	High	O/US	O/S	O/S	O/S
Our Proposition	Generic	✓	✓	✓	✓	Selective Collective	Low	O/US	O/S	O/S	O/S

as many mobile applications as possible per server and offer the user's expected acceleration in the mobile application execution. Proposed by Chae et al. (2014), CMcloud seeks to find the least costly server which has enough remaining resources to finish the execution of the mobile application within a target deadline.

COSMOS system has been presented by Shi et al. (2014) with the objective of managing cloud resources to reduce their usage monetary fees while maintaining good offloading performance. Through its master component, COSMOS collects periodically information of computation tasks and remote VMs workloads. Based on the gathered information, COSMOS is able to control the number of active VMs over time. Particularly, whenever VMs are overloaded, the system turns on new instance to handle the upcoming requests. It can also decide to shut down unnecessary instances to reduce the monetary cost in case the rest are enough to handle the mobile devices requests.

From Table 1, offloading unit identifies the code level granularity where offloading is applied. The mobile cost model metrics represent the decision model aspects used to evaluate offloading productivity. Model evaluation show the characteristics of the offloading evaluation process, while evaluation overhead reflects the decision making cost. Finally, savings highlight the gain obtained by tasks dissemination generated by the decision engine after cost model evaluation (O and SO stand for optimal and suboptimal while S and US stand for stable and unstable respectively).

Discussion. Significant attention has been turned toward computation offloading to support mobile devices. Existing approaches focused on enhancing performance and saving energy on the mobile terminal (Chen et al., 2012; Chun et al., 2011; Cuervo et al., 2010; Gordon et al., 2012; Kemp, 2014; Xia et al., 2014), others targeted the additional fees imposed by remote execution (Chae et al., 2014; Kosta et al., 2012; Shi et al., 2014) and in our turn we focused in previous work (Tout et al., 2016) on addressing performance and survivability with multiple virtual environments running on the mobile device. Evaluating the cost model independently for each task or even collectively cause significant overhead and create itself a bottleneck on the mobile terminal with resource constraints, which is a common limitation in all these approaches. Along with this overhead, these approaches fall in suboptimal and unstable savings in the dissemination of tasks due to limited aspects considered in the cost model.

Differently, this paper goes beyond existing works by proposing new system model for computations offloading. As independent offloading evaluation, proposed in existing approaches, fails to handle circumstances when multiple tasks are running simultaneously and hence the decision on a task affects the others, we present a centralized approach able to collectively evaluate offloading tasks from different applications. Also, we propose a selective mechanism to process offloading evaluation only for selected tasks, designated as hotspots, which is capable of significantly reducing the overhead of the decision engine compared to existing approaches. Moreover, the latter evaluates an optimization model that includes energy loss, CPU usage, memory consumption and performance of each component, essential metrics to augment mobile device resources and performance and not being all considered in any of the existing works. We emphasize the efficiency of these metrics where we prove their ability to overcome suboptimal and unstable savings of tasks dissemination. Differently from exiting models, we also refine the metrics making them resilient not only to the device state but also to the hotspots and to strategies that can be enforced on the mobile terminal like offloading prioritization and suspension of tasks. The decision engine decodes, for the designated components, the execution strategy that achieves a balance between all the metrics considered in the model, reaching stability with high savings in local resource usage and significant execution speedup. Essentially and differently, the proposed computation offloading system model is able to intelligently reduce the overhead of offloading evaluation without jeopardizing optimality in tasks dissemination savings.

4. Technical problems

We highlight in this section the technical problems subject of this work.

4.1. Accuracy and overhead of decision model evaluation

The decision making is an iterative process invoked by the decision engine (solver) and triggered to handle critical situations like processing power degradation, storage inefficiency and dying battery. This process evaluates the decision model to determine whether to offload particular components or not. In existing approaches, the cost model is evaluated for each task independently, which has its own consequences when multiple applications run simultaneously on the mobile terminal to meet with daily life needs. In one hand, with independent offloading evaluation, the system lacks global view of the execution environment which results in inaccurate and faulty decisions. On the other hand, repeating the same process for each task is not reasonable and imposes in turn considerable overhead on the device, higher than the savings achieved by the dissemination if tasks.

The alternative is a centralized collective decision maker that considers the running components from different applications in the evaluation process. Nonetheless, the overhead of such decision engine increases along with the number of components candidates for offloading. Theoretically, deciding what to offload suffers from an exponential search space in the number of different possibilities in which the components can be distributed (i.e., local or remote

execution). It is similar to the various ways n distinct objects (components) can be distributed into m different bins with k_1 objects in the first bin, k_2 in the second one, etc. and $k_1 + k_2 + \ldots + k_m = n$. By applying the multinomial theorem, this can be calculated as $\sum_{k_1+k_2+\ldots k_m} = n\binom{n}{k_1,k_2,\ldots,k_m}$. In our multi-apps offloading problem, $m = 2$, one bin is the mobile terminal and the other is the remote server thus, for n components, there are 2^n different distribution possibilities. This number will dramatically increase with fine grained components like services and will be more crucial with methods and threads as their number is greater inside the applications. Our previous work (Tout et al., 2016) suffered also from this dilemma, where the results revealed that the evaluation process in such case can consume up to 6x more CPU usage and 1.25× more energy consumption than the services themselves, and takes up to 3 s to find the distribution. These results pushed towards sacrificing optimality of tasks dissemination savings to mitigate the overhead of the evaluation process by accepting suboptimal solution. A possible alternative to overcome such overhead is to take the decision remotely, yet this requires sending relevant data to remote server which not only imposes in turn more overhead, but also additional monetary fees are to be carried out. Therefore the question is how to decrease the overhead of decision making without sacrificing optimal distribution of tasks?

4.2. Decision model metrics

In the presence of network connectivity and available remote resources, the offloading decision is based upon number of factors. The available resource on the mobile terminal, the latency and bandwidth of the network, the resource demands and performance of the component, any of them can form a context that influences offloading feasibility and efficiency. Besides, energy and execution time, processing power and memory capacity would critically influences the mobile device when running multiple applications simultaneously. As the energy is being consumed on many elements other than CPU and memory, essentially, graphics, screen and network, decreasing the power consumption does not guarantee more processing power or even more memory availability. As long as the device is on and new applications are running, both CPU and memory usage levels continue to change. Therefore, for example, in case the device is running slow or out of memory, migrating components that require intensive processing and lot of memory would be efficient respectively, regardless of the energy consumption level. Thus, each time an offloading decision is to be made, these factors should be taken into consideration and an explicit evaluation of these metrics should be done. As none of the existing techniques has considered and examined a tradeoff among all these metrics, what is the effect of such multi-objective optimization on the optimality and stability of tasks dissemination savings?, is a question to be answered in this work as well.

5. Centralized selective and multi-objective offloading: insights

The proposed system model is depicted in Fig. 2. We devote a set profilers on the mobile terminal to monitor the relevant resources. The profilers examine the availability of connectivity, beside its data rate and latency. They also monitor the energy loss, CPU usage, memory consumption , and execution time for all components. A component can be service, method, or even thread that implements certain functionalities. It should be clear that this does not mean that the system deals with components of heterogenous nature but it is generic enough to be applied independently of the offloading unit considered. The gathered profiled data are then communicated with the rest of the modules on the device. Capturing different criteria that influence the resource consumption and performance of the components, the profiled data serve as input for the detector to identify hotspot components forming a subset of the offloading candidates. Frequently invoked, resource-intensive and/or time consuming components are marked as hotspots. Such criteria and the thresholds used for hotspot selection are adjusted by the strategies controller based on the device state, instrumented by the profilers. Gathered data are also used by a centralized, selective and intelligent decision engine, which evaluates a cost model to analyze tasks offloading. Only selected hotspots (C_2, C_3 and C_4) go into the offloading evaluation process, while the rest of the components (C_1), not included, continue their execution locally on the mobile terminal. The decision model consists of essential metrics that guarantee resource availabilities and enhanced performance on the mobile device. The controller can enforce different strategies in the model to manage offloading evaluation. Such strategies include weights between the model metrics to adapt it with the device state. For example, additional weights can be given to the memory metric when the device is running out of memory. Other strategies include priorities between components like prioritizing offloading from foreground application or even based on criticality classification. Defining such strategies at high level in the controller form part of future track, yet this work considers and adapt to such strategies at the low level of the decision model. The latter is hence adaptable with these controls as well as with the selected hotspots which vary with the device usage. The decision engine includes also a tailored algorithm that examines a tradeoff between the model metrics and dictates accordingly what components to be offloaded and those to run on the mobile terminal (only C_4 is to be offloaded). Offloading computations can be done through the cellular network of Wifi hotspots. When Wifi capacity is purchased by the cellular network provider, both cellular and Wifi capacities should be considered to evaluate the possibility to process the issued offloading requests as well as to predict the channels ability to handle upcoming requests. Yet, in this work we assume enough resources to handle mobile computations offloading regardless the wireless technology employed (Wifi or cellular network), which is a fair assumption in the light of the resourceful infrastructure of cloud service providers. The following sections detail our proposition.

6. Selective mechanism

To reduce the search space of the decision maker, only hotspot components from the offloading candidates are selected and considered in the evaluation model, while the rest of the components are to be executed locally.

6.1. Hotspots profiling

Profilers on the device generate instrumented data for each and every component. The produced profiles serve as input data used by the detector to identify hot components for selective offloading evaluation. The profilers capture different criteria that influence the resource availabilities and performance on the mobile device. Generated profile includes for every component, the CPU and memory utilization, size and execution time and its frequency of invocation, which are the criteria we use for hotspots selection. The detector modules 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 to be considered for offloading evaluation. The higher thresholds will keep the hotter components.

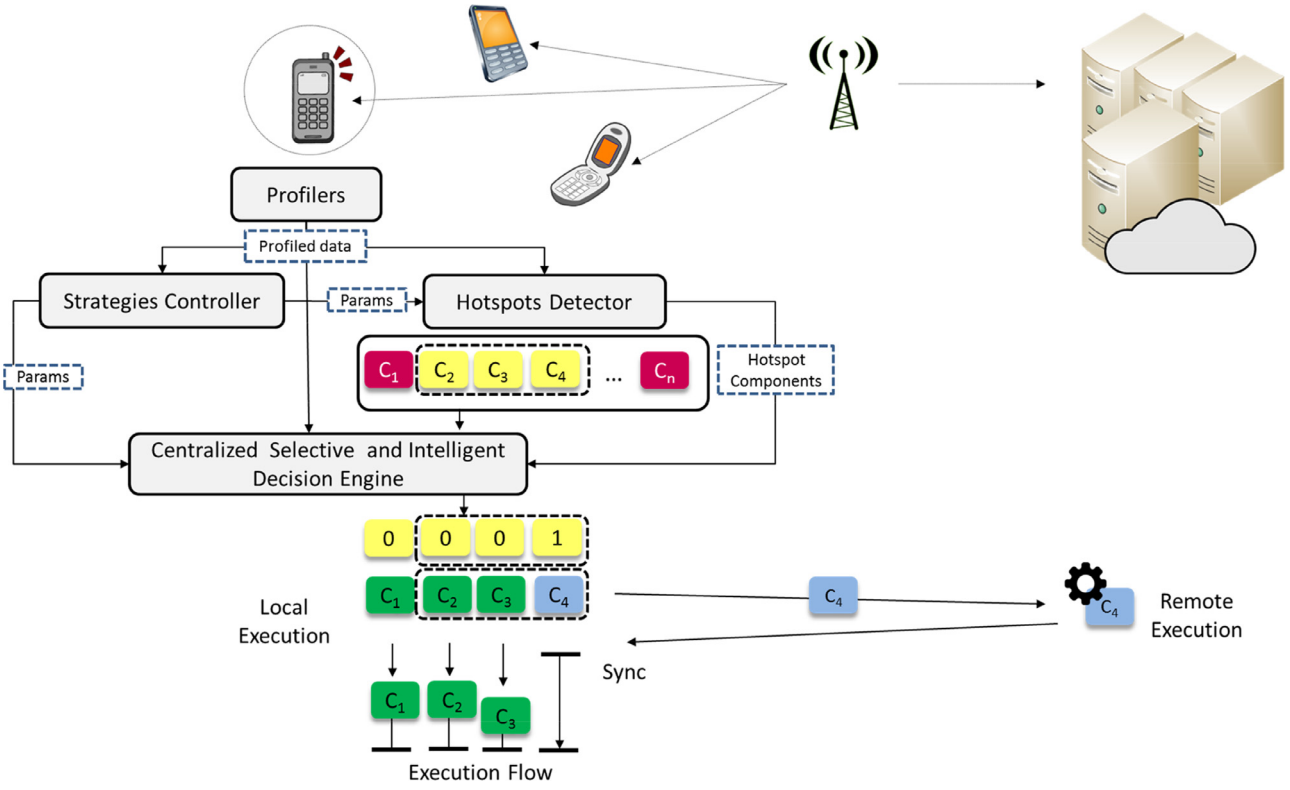

Fig. 2. System model.

6.2. Hotspots detection

By default, threshold values are set based on Bayesian average according to the following formula:

$$\bar{V} = \frac{|C| * m + \sum_{i=1}^{|C|} v_i}{|C| + m} \quad (1)$$

where, $|C|$ is the components data set size, m is the prior mean value and v_i is the profile data value. However, to provide efficient decision for the distribution of components, the strategies controller can tune the selection criteria as well as their threshold values according to the mobile device state instrumented by the profilers. The device for instance might suffer from high CPU usage, while after a period of time might run out of memory based on the number and complexity of the applications on the mobile terminal. Thus, the priority of the selection criteria can be tuned according to each situation and some of the criteria might not be considered in the selection of hotspots correspondingly. The thresholds values are also adaptable. For instance, normally, the battery level will clearly drain along with the usage of the device and the execution of applications. Therefore with dying battery, even components with low energy consumption should be considered as hotspot. For this end, the energy threshold value can be reduced by the controller so that even components with small energy cost on the device would be considered. In this article, we refine the decision model to consider these adaptations, yet, at high level, defining such strategies in the controller module is part of future track.

6.3. Selection algorithm

Algorithm 1 illustrates the process to select hotspots. Components from different applications, their profiled data, and the thresholds values and a selection ratio to limit the number of components marked as hotspots, all form the input of this algorithm.

Algorithm 1 Hotspots selection.

1: Input: Component set $C = \{C_i, \ldots, C_n\}$, each of which is characterized by size s_i, invocation frequency f_i, CPU usage c_i, Memory usage m_i , Energy consumption e_i, execution time t_i; thresholds for each criteria respectively $T_s, T_f, T_c, T_m, T_e, T_t$, and selection ratio r
2: Output: A subset of hotspot components $H \subseteq C$
3: $H \leftarrow \emptyset$
4: **for** $i = 1$ **to** n **do**
5: isHotspot(C_i, s_i, T_s, H)
6: isHotspot(C_i, f_i, T_f, H)
7: isHotspot(C_i, c_i, T_c, H)
8: isHotspot(C_i, m_i, T_m, H)
9: isHotspot(C_i, e_i, T_e, H)
10: isHotspot(C_i, t_i, T_t, H)
11: **end for**
12: **if** $H = \emptyset$ **then**
13: $H \leftarrow H \cup \{C\}$
14: **end if**
15: **if** $|H| > r$ **then**
16: $NH \leftarrow chooseRand(|H| \times r, H)$
17: $H = NH$
18: **end if**
19: **return** H

The process starts by initializing an empty set for the hotspots (Line 3) then loops over all the components calling *ISHOTSPOT* procedure in Algorithm 2 for hotspots identification (Line 4 till Line 11). This procedure fills the hotspot set with components having profiled data exceeding the predefined threshold value. Those criteria which are not considered in the selection process have a threshold value of -1, as an indicator used by the controller module to vary the selection criteria. *ISHOTSPOT* returns the set

Algorithm 2 isHotspot.

```
1: procedure isHotspot(C, v, T, H)
2:     if T! = −1 & C∄H then
3:         if v > T then
4:             H ← H ∪ {C}
5:         end if
6:     end if
7:     return H
8: end procedure
```

of hotspot components back to Algorithm 1. If no hotspots were found (Line 12), the set is filled with the initial list of components (Line 13). In case the number of hotspots exceeds a predefined ratio (Line 15), which is by default $n/2$ defined based on empirical study, the number of hot components are limited to that value (Lines 16) with random choice from the components set to avoid selecting the same values. Finally the algorithm returns the hotspots set to be considered in the evaluation model (Line 19).

Lemma 1. *The time complexity of Algorithm 1 is $O(n)$.*

Proof. The time complexity of the selection algorithm depends solely on the number of the entries n in the components set C. Initializing the empty set H (Line 3) takes $O(1)$. Next, running over all the elements in the C set to identify hotspot components (Line 4–Line 11), takes $O(n+1)$ and the inner procedure call *isHotspot* is of $O(1)$. Checking if the hotspot set is empty (Line 12) and subsequently assigning the elements of C to H (Line 13), each takes O(1). Finally, in case the number of elements in H exceeds the ratio r for hotspot components (Line 15), the algorithm will loop again on the elements in H to select the appropriate portion (Line 16), which takes $O(|H| \times r + 1)$. With $H \subseteq C$, $O(|H| \times r + 1)$ has lower order compared to $O(n+1)$, the former can be dropped and thus hotspots can be selected in $O(n)$. □

7. Centralized selective offloading decision model

We devote this part to present the proposed centralized, selective and optimized offloading decision model.

Assumptions. We assume in the proposed system model that offloading can be prioritized between components. These priorities are assigned and enforced by the strategies controller. For example, foreground components can have higher priorities to use local/remote resources over background ones. This can be also based on appropriate criticality classification scheme, where more critical apps will be given for example higher priorities to have better performance. Also, the execution of tasks might be suspended based on the device resources. We consider all these strategies at low level in the decision model while their definition at higher level is to be addressed in future work. Additionally, components are assumed to be independent and some of them might not be offloadable like tasks that require local device data. We also consider dynamic environment where the network might not be available and its data rate and latency may vary.

7.1. Definition

Definition 1. Considering the set of hotspot components, generated by the selection process, the device state instrumented by the profilers and the management parameters values of the controller, the problem is to find components that should be offloaded and those to be executed locally for a tradeoff of enhanced computing capacity, minimal memory and battery usages on the device and better performance. The problem defined as follows:

Given a set of hotspot components $H = \{c1, c_2, \ldots, c_k\}$ on a device D, each of which c_i needs $CPU_{c_i}^{local}$ processing unit, $Memory_{c_i}^{local}$ memory, $Energy_{c_i}^{local}$ energy and spends $ExecTime_{c_i}^{local}$ period of time when executed on the device; while when offloaded, each consumes $CPU_{c_i}^{Remote}$ processing unit, $Memory_{c_i}^{Remote}$ memory, $Energy_{c_i}^{Remote}$ on the device and needs $ExecTime_{c_i}^{Remote}$, waiting and processing the remote response; α_{c_i} an indicator whether the component c_i is offloadable or not, offloading priorities p_{c_i}; network bandwidth *Bandwidth* and latency *Latency*, weights $w_{(F_P)}$, $w_{(F_M)}$, $w_{(F_E)}$ and $w_{(F_T)}$ for the evaluation metrics; the decision should dictate for each component whether it should be executed locally or remotely in a way to minimize the overall energy loss F_E, CPU F_C and memory F_M usages on the mobile device and speedup the execution F_T for better experience.

Theorem 1. *Offloading optimization decision making is NP-Hard*

Proof. Offloading optimization can be easily seen in the NP-class; as once a dissemination of components is found, it can be verified in polynomial time. Next, we will prove that this problem is NP-Hard via a reduction from the NP-hard multi-objective-m-dimensional Knapsack Problem (MOMKP) (Lust & Teghem, 2012). We aim in what follows to prove that a solution found for a case of our multi-apps code offloading optimization problem can be used to solve multi-objective-m-dimensional Knapsack. Given the MOMKP - a collection of n items $a_1, \ldots, a_n$, where each item a_i has m weights $w_{ki} \in \mathbb{N}, k = 1, \ldots, m$ and t values $p_{ki} \in \mathbb{N}, k = 1, \ldots, t$ and a knapsack of m capacities $c_k \in \mathbb{N}$, $k = 1, \ldots, m$ - we can build the offloading optimization decision making problem as follows: Having x applications $A = \{a_1, \ldots, a_x\}$ with y components in each, forming a set of x^*y representing k components $H = \{c_1, \ldots, c_k\}$, each corresponding to an item in MOMKP: Set the resource demands of each component c_i in terms of memory and CPU as the weights of the items in the sack. $CPU_{c_i}^{local}$, and $CPU_{c_i}^{remote}$, $Memory_{c_i}^{local}$ and $Memory_{c_i}^{remote}$, are CPU and memory usages when C_i is running locally or executed remotely, respectively. Then, set f_E, f_T, f_P and f_M as the values of each item, where they form the cost of each component in terms of energy consumption, execution time, CPU usage and memory needs respectively. So that $\sum_{i=1}^{x} f_j$ where $j = 1, \ldots, 4$ formulates each of objective functions in the model correspondingly. Accordingly, the knapsack content is a portion of components selected to run on the mobile device such that the total of each value (cost) is minimized, and vice versa, and a solution to our problem yields a solution to the MOMKP. After this complete proof of the reduction, we conclude that the this problem is NP-Hard. □

7.2. Model formulation

In the following, we mathematically formulate the proposed offloading evaluation model based on the definitions provided. Built on top of previously proposed optimization metrics in our latest achievement (Tout et al., 2016), the model in this work is now resilient not only to the device state but also with the hotspots selection and the execution management strategies defined above.

- Decision Variables:

  $x = \{x_{c_1}, \ldots x_{c_k}\}$

  where,

  $$\forall c_{i,i:1\to k}, x_{c_i} = \begin{cases} 0, & \text{if } c_i \text{ is to be executed locally} \\ 1, & \text{if } c_i \text{ is to offloaded} \end{cases}$$

- Parameters:

D	mobile device
H	set of hotspot components
c_i	hotspot component
γ_{c_i}	offloadable component indicator
$ExecutionTime_{c_i}^{local}$	time to execute c_i locally
$ExecutionTime_{c_i}^{remote}$	round trip time to process c_i remotely
$CPU_{c_i}^{local}$	cpu usage on D by c_i executed locally
$CPU_{c_i}^{remote}$	cpu usage on D by c_i offloaded
$Memory_{c_i}^{local}$	memory usage on D by c_i executed locally
$Memory_{c_i}^{remote}$	memory usage on D c_i offloaded
$Data_{c_i}$	size of data transmitted for offloading c_i
$power_{cpu}$	power consumed by D on precessing
$power_{screen}$	power consumed by D on the screen
$power_{idle}$	power consumed by D on idle CPU
$power_{transmission}$	power consumed by D for transmission
p_{c_i}	offloading priority for component c_i
s_{c_i}	suspension indicator for component c_i
$w_{(F_E)}$	weight for function F_E
$w_{(F_T)}$	weight for function F_T
$w_{(F_C)}$	weight for function F_C
$w_{(F_M)}$	weight for function F_M
$\tilde{T}_{F_P}$	threshold for processing load
$\tilde{T}_{F_M}$	threshold for memory consumption
$\tilde{T}_{F_E}$	threshold for energy loss

- Mathematical Model:
$Minimize(F_E, F_T, F_C, F_M)$ where,

$$F_E = \left[\sum_{i=1}^{|H|} s_{c_i}(1-x_{c_i}) \times ((Power_{cpu} + Power_{screen}) \times ExecTime_{c_i}^{local}) + \sum_{i=1}^{|H|} s_{c_i}x_{c_i} \times \left((Power_{idle} \times ExecTime_{c_i}^{remote}) + \left(Power_{transmission} \times \left(Latency + \frac{Data_{c_i}}{Bandwidth}\right)\right)\right) \times \gamma_{c_i} \times p_{c_i}\right] \times w_{(F_E)} \quad (2)$$

$$F_T = \left[\sum_{i=1}^{|H|} s_{c_i}(1-x_{c_i}) \times (ExecTime_{c_i}^{local}) + \sum_{i=1}^{|H|} s_{c_i}x_{c_i} \times \left(ExecTime_{c_i}^{remote} + Latency + \frac{Data_{c_i}}{Bandwidth}\right) \times \gamma_{c_i} \times p_{c_i}\right] \times w_{(F_T)} \quad (3)$$

$$F_C = \left[\sum_{i=1}^{|H|} s_{c_i}(1-x_{c_i}) \times CPU_{c_i}^{local} + \sum_{i=1}^{|H|} s_{c_i}x_{c_i} \times CPU_{c_i}^{remote} \times \gamma_{c_i} \times p_{c_i}\right] \times w_{(F_C)} \quad (4)$$

$$F_M = \left[\sum_{i=1}^{|H|} s_{c_i}(1-x_{c_i}) \times Memory_{c_i}^{local} + \sum_{i=1}^{|H|} s_{c_i}x_{c_i} \times Memory_{c_i}^{remote} \times \gamma_{c_j} \times p_{c_i}\right] \times w_{(F_M)} \quad (5)$$

Subject to

$$F_P < \tilde{T}_{F_P} \quad (c_1)$$
$$F_M < \tilde{T}_{F_M} \quad (c_2)$$
$$F_E < \tilde{T}_{F_E} \quad (c_3)$$

The model aims to decrease energy loss, speed up the execution, minimize processing and memory usage on the mobile device, objectives which are defined in Eqs. (2), (3), (4) and (5) respectively with p_{c_i} and s_{c_i} to control prioritization and suspension of tasks accordingly. The model is subject to several constraints; (c_1) forces the decision maker to look for solutions that do not overload processing on the mobile device, (c_2) ensure the candidate solutions do not excess the memory on the end terminal and (c_3) represents the energy constraint that guarantees a threshold for available power on the device.

8. Intelligent decision making process

The proposed intelligent decision maker exploits the smart evolution of solutions in genetic algorithms (GAs) (Deb, 1999), which have been able to solve complex optimization problems in many areas (Cai & Chen, 2014; Grefenstette, 2013; Wu, Chiang, & Fu, 2014) through their method of evolution inspired search. Based on natural selection, GAs simulate the propagation of the fittest individuals over consecutive generations to determine the best solution. We investigated different algorithms for the decision making process. The first algorithm is NSGA-II (Deb, Pratap, Agarwal, & Meyarivan, 2002), a multi-objective genetic algorithm which uses pareto ranking mechanism for classification of solutions and crowding distance to define proximity between them. Next, SPEA2 (Zitzler et al., 2001), which is another multi-objective evolutionary algorithm based on pareto dominance, yet characterized by its strength scheme that not only takes into account the number of solutions that dominate particular solution, but also the number of solutions by which it is dominated. The third algorithm is SMSEMOA (Emmerich, Beume, & Naujoks, 2005), which is a steady state algorithm, in which a random selection of individuals is done for the mating process and the offspring replaces the individuals of the parent population. Further, IBEA (Zitzler & Künzli, 2004) is an algorithm that employs a quality indicator in the selection process. Finally, MOCell algorithm (Nebro, Durillo, Luna, Dorronsoro, & Alba, 2009) which is characterized by both decentralized population and archive to store non-dominated solutions. We implemented these algorithms as introduced by their authors yet with the adequate mapping. An extensive study presented in previous work (Tout et al., 2016) proves the efficiency of NSGA-II (Deb et al., 2002) over other algorithms. In this work, we redesign NSGA-II with adaptive fitness evaluation and evolution-based crossover. According to the optimization model that we presented in Section 7, the intelligent decision maker is capable of generating the distribution of hot components that minimizes the resource usage and enhance the performance. The process adopted by the decision maker is depicted in Algorithm 3. Hereafter, we detail the solution encoding as well as the genetic operators, then in Section 9, we study its efficiency.

8.1. Solution encoding

Each chromosome also called individual in a population forms a candidate solution in GAs. For offloading decision optimization, the algorithm starts with population of N randomly generated individuals according to the number of components marked as hotspots (Line 5). Each individual represents the distribution of components. Every individual has a size $|H|$ and it is encoded as a set of binaries $x = \{x_{c_1}, \ldots, x_{c_k}\}$, where k is the total number of components involved. Each gene x_{c_i} of an individual represents a component on the mobile device and its value dictates whether this component should be executed locally on the end terminal ($x_{c_i} = 0$) or offloaded ($x_{c_i} = 1$).

Algorithm 3 Intelligent decision maker.

1: **Input:** Set of hotspot components $H = \{c_1, c_2, \ldots c_k\}$, each of which is characterized by local and remote execution time, cpu usage and memory consumption, network characteristics BL in terms of bandwidth and latency, number of possible solutions N, mutation rate μ_m, crossover rate μ_c and number of generations λ.
2: **Output:** Distribution set of hotspots H'.
3: $i \longleftarrow 0$ ▷ i is the population index
4: $H' \longleftarrow \emptyset$
5: $G_i \longleftarrow Random[N][|H|]$ ▷ generates random population
6: **for** $k = 1$ to r **do**
7: Calculate F_E:=$CalcEnergy(H, G_i, BL)$
8: Calculate F_T:=$CalcTime(H, G_i, BL)$
9: Calculate F_C:=$CalcProcessing(H, G_i)$
10: Calculate F_M:=$CalcMemory(H, G_i)$
11: **end for**
12: **for** $g = 1$ to λ **do**
13: {
14: Propagate b best candidates distributions BC for next generation $G_{i+1} \longleftarrow BC$
15: select two solutions from G_i, X_A and X_B;
16: Generate X_C by evolution-based crossover to X_A, X_B;
17: Add X_C to G_{i+2};
18: Select a solution X_b from G_{i+2};
19: Mutate X_b and generate new feasible solution $X_{j'}$;
20: **for** $k = 1$ to r **do**
21: Reexamine F_E:=$CalcEnergy(H, G_{i+1}, BL)$
22: Reexamine F_T:=$CalcTime(H, G_{i+1}, BL)$
23: Reexamine F_C:=$CalcProcessing(H, G_{i+1})$
24: Reexamine F_M:=$CalcMemory(H, G_{i+1})$
25: **end for**
26: Update generation $G_i = G_{i+1} + G_{i+2}$
27: Update generation index $i \longleftarrow i + 1$
28: **if** Same fitness is detected in G_{i+1} and G_{i+2} **then**
29: break;
30: **end if**
31: }
32: **end for**
33: $H' \longleftarrow$ optimal components distribution in G_i
34: **return** H'

8.2. Fitness evaluation

In GAs, a score/fitness is designated for each chromosome simulating the propagation of the fittest individuals over consecutive generations in order to determine the best solution. The fitness varies based on how efficiently each individual can solve the problem. In our case, the fitness of each candidate solution is determined by F_E, F_T, F_C and F_M functions that constitute the model proposed in Section 7 (Line 6 to Line 11). With all these metrics are to be minimized, the solutions are ranked according to their ability to speedup the execution and reduce the resources usage on the mobile terminal.

The fittest b individuals in the population are then selected to go through a process of evolution (Line 14). In the latter, crossover (Line 15–Line 17) and mutation (Line 18–Line 19) operations are applied to produce next generation of individuals for new possible distribution solutions of components. The algorithm reassesses the model metrics to calculate the fitness of these generated individuals (Line 20–Line 25). The evaluation process continues over and over until any of the stopping criteria is met, where either the fitness of the best individual in successive populations did not improve (Line 28–Line 30) or the defined number of iterations δ is reached. Finally, the decision maker returns the fittest distribution from the last generation having the optimal tradeoff between the defined metrics (Line 33–Line 34).

8.3. Evolution process

8.3.1. Selection

In this phase, chromosomes are selected to go through the evolution process. We use bit tournament selection, which involves running several rounds over randomly chosen chromosomes from the population. The winner in each round, which has the best fitness is then selected for crossover.

8.3.2. Crossover

Crossover is achieved by exchanging genes between two individuals with the intent to produce better offspring. With a μ_c rate, crossover usually occurs when regions of a chromosome break and reconnect to the other chromosome. In contrast, we propose an evolution-based crossover operator. This operator is based on the differential evolution of individuals that optimizes offloading. Taking two parents individuals, genes that produce better fitness (smaller fitness value based on the evaluation presented in Section 8.2) when compared to their parents are used to form the offspring.

8.3.3. Mutation

Mutation operation is to apply additional modifications in the chromosomes that improve their fitness. With a μ_m rate, we apply standard bit flip mutation.

Lemma 2. *The time complexity of Algorithm 3 is* $O(\lambda N|H|)$

Proof. The complexity of this algorithm is determined by the fitness function evaluation, the population size, the individual length, variation and selection operators and the number of iterations or generations. Initializing generation index, solution set H' and generating the first random population has each time complexity $\mathcal{O}(1)$. The evaluation of the fitness function has time complexity of $\mathcal{O}(N+1)$ where N is the population size. The tournament selection, evolution-based crossover and bit flip mutation, have time complexity of $\mathcal{O}(N|H|\lambda)$ where $|H|$ is the size of an individual and λ is the number of generations. Reassessing the model is of $\mathcal{O}(\lambda N)$. Finally, the return statement has $\mathcal{O}(1)$. Subsequently, the time complexity of the algorithm is $\mathcal{O}(1) + \mathcal{O}(N+1) + \mathcal{O}(\lambda N|H|) + \mathcal{O}(\lambda N) + \mathcal{O}(1)$. With lower orders are to be dropped, this is equivalent to $\mathcal{O}(\lambda N|H|)$. □

9. Numerical analysis

We devote this section to present the experimental results that demonstrate the efficiency of our proposition.

9.1. Testbed setup

In the following experiments, the mobile terminal is running Android operating system with quad-core processor and 1GB of RAM. The implementation of a mobile application should follow first particular design pattern in order to make it offloadable. The activity/service model in android allows clear separation between the application code and its user interface. Particularly, the logic code of the computation-intensive tasks can be implemented as services through an interface definition language (AIDL) while the user interface is defined using activities. Applying such model would facilitate the offloading task as services and activities are already isolated. For this end, we developed three mobile services of

different weights that we use in our experiments to map different usage scenarios of the device.

- Zip/Unzip is a lightweight service that allows creating archive folder from files and extracting back the content.
- Virus Scanning is a moderate service that scans files of 100KB against a library of 1000 viruses signatures, 1 file at a time.
- NQueens Puzzle is a computation intensive service that implements an algorithm capable of finding all possible solutions of the typical NQueens problem and returns the number of solutions found. In our version we use N = 13 to create heavy app.

To make these services offloadable, we take advantage of the offloading libraries provided in Kemp (2014) as it applies the same design pattern (i.e., activity/service). On first invocation, the offloadable services are sent for remote execution on pre-configured server. Only the .class files of the remote implementation are automatically packaged as jars and transmitted to the server. With just few kilobytes, the transmission of such package drives negligible overhead over the network. Whenever the requested services are already hosted on the server, only relevant parameters are communicated. Yet, the relevant overhead is included in the results. To profile services, we implemented Linux-based commands to monitor CPU and memory usages, while we used PowerTutor (Zhang et al., 2010) to monitor the energy consumption on different aspects like CPU, screen and network. As for the power consumption on idle CPU, active network and during transmissions, we took advantage of the power profile in android (Android, 2016) to get the relevant information. Finally, we embedded a timer to monitor the execution of each service. As for the decision making algorithm, the configuration is based on empirical study of 50 times of execution. We set $\mu_c = 0.6$, $\mu_c = 1/n$, where n is the number of decision variables, $N = 100$ and the number of generations λ to be 20, 45, 70, 75, 100 for 10 to 50 services respectively. The connection between the mobile terminal and the server is achieved through WiFi network in infrastructure mode. It is characterized by the IEEE 802.11n wireless networking standard. The server side is running Ubuntu 12.04 with 7.3GB memory and quad core AMD Phenom(tm) II X4 B95 processor.

9.2. Results

9.2.1. Decision model efficiency

This set of experiments aims to study the efficiency of the proposed offloading optimization model, which considers energy, execution time, CPU and memory metrics in the evaluation process and its effect on the optimality and stability of tasks dissemination savings. Running the developed services, we compare the savings and performance improvement achieved by our proposition to the those provided by existing models presented in Table 1. The energy model adopted by Cuervo et al. (2010), the time model adopted by Shi et al. (2014) and Chae et al. (2014) and the energy/time model used by Chen et al. (2012); Chun et al. (2011); Kemp (2014); Kosta et al. (2012); Xia et al. (2014). Considering the three services described in the setup, we run each of the decision models on the mobile device to make the offloading evaluation and generate the distribution of these services accordingly. Fig. 3 highlights the savings of the services dissemination found by each model in terms of resource savings and execution speedup.

The energy model shows stable results. However, in terms of optimality, this model can lead to the tasks dissemination that offers the best energy savings yet the minimal in terms of processing, memory and execution speedup. The results show that this model is able to offer 99% less energy with just 33% less CPU usage, 10% less memory consumption and 59% speedup in the execution, with average savings values of 99%, 33%, 10% and 58% respectively. Similarly, the time model shows stability with 485 tasks dissemination that provides the best speedup possible of 63% yet worst in terms of energy with 97%, processing with 33% and memory usage with 10% savings, with average of 63%, 97%, 33%, and 10% savings

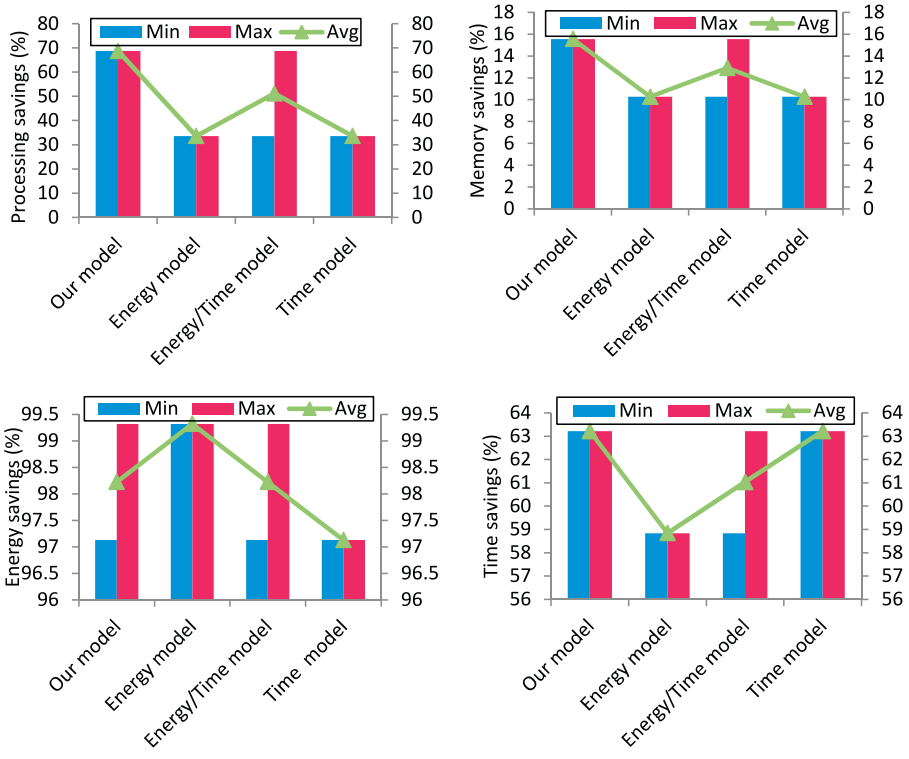

Fig. 3. Decision savings.

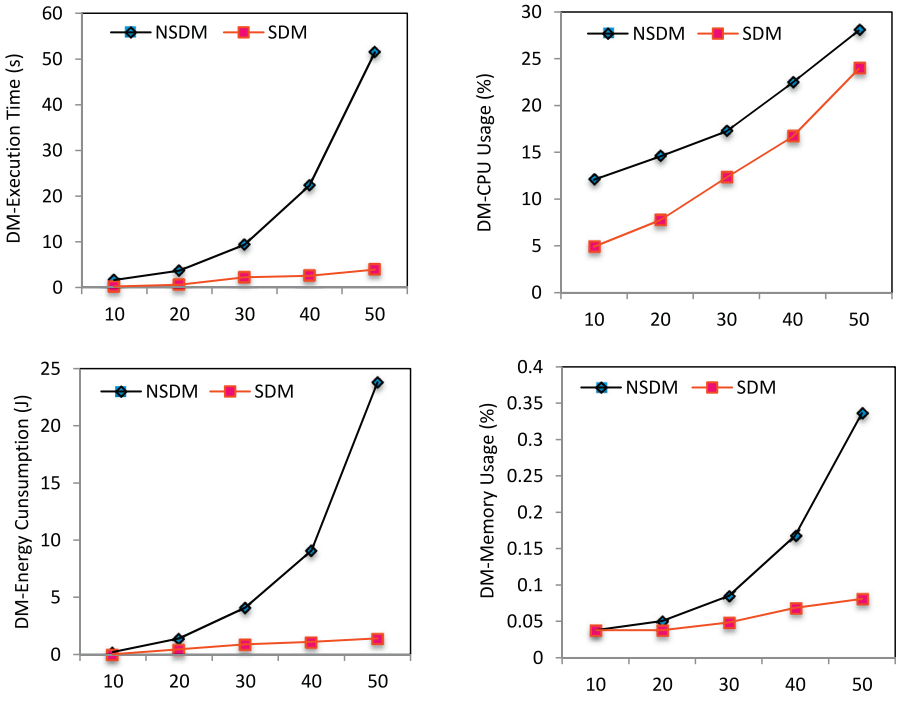

Fig. 4. Decision maker overhead.

respectively. On the other hand, the energy/time model is able to reach optimality with respect to processing, memory, energy and executions speedup with a dissemination that offers average savings of 51%, 12%, 98% and 61% savings respectively. However, this model shows instability and hence risks finding the dissemination with such values and fall in suboptimal results of 33% 10% 97% and 58% savings accordingly. Per contra, our proposition outperforms the other models and shows stable results in terms of CPU, memory and execution time with 68%, 15.5% and 63% as average reductions respectively. As for the energy, our model can reach up to 99% reduction and 97% in the worst case, with an average of 98%, which is still comparable to the energy/time model. These results confirm that the proposed model is more adequate to offer better trade-off of resources and performance on the device with higher stability.

9.2.2. Selective mechanism and intelligent decision maker efficiency

The second set of experiments is intended to study the efficiency of the selective method in reducing the overhead of the evaluation process and the ability of the decision maker to adapt to it without jeopardizing finding the optimal distribution of components. The results of these experiments are depicted in Figs. 4–6. We increment the number of services stressing the mobile device to study the scalability and cover the case of more fine grained components like methods and threads. We cloned the applications defined in the testbed setup not to implement such large number of services.

We examine first the overhead of the decision making process (DM) that evaluates our multi-objective optimization model to find the optimal distribution of services (Fig. 4). The results show that increasing the number of components (i.e., services), imposes significant overhead by the decision maker on the mobile terminal when no selective method is applied (NSDM). Specifically, they show drastic increase in the CPU usage of the decision maker that was 12% with 10 services and reached 28% with 50 services. In addition, its memory usage increased 7 times, its energy consumption increased 114 times and its speed decreased 31 times. Compared to NSDM, our selective method (SDM) was able to reduce 1.2 times the CPU usage of DM, 4 times its memory consumption, 17 times the energy consumption and speedup 13 times its execution.

Following the distribution generated by the decision maker in each scenario, we measured the overhead of the services dissemination as well (Fig. 5). The objective here is to check how the distribution found by the DM, without considering all the components (i.e., using our selective method SDM) in the cost-benefit analysis, can affect the services overhead. Notably, SDM was able to find the optimal distribution in many scenarios, namely with 20, 30 and 50 services, and hence did not cause any overhead in terms of CPU usage, memory consumption, energy and execution time of the services in these cases.

Another interesting observation can be highlighted when comparing the overall overhead caused by both the DM and the components on the device (Fig. 6). The results show that even when SDM caused more services overhead than NSDM (scenarios where SDM could not find the optimal solution like when running 40 services), the overall overhead remained better for SDM. This is due to the notable improvement SDM was able to reach in terms of decreasing the overhead of the offloading decision evaluation process.

Finally, we examine the error rates of non-beneficial offloading. The results are illustrated in Table 2. The results prove that our proposition is always capable of finding the trade-off that optimizes the device resources and performance based on the proposed multi-objective optimization model, when other existing models risk finding the adequate computations dissemination.

Table 2
Decision error rate.

	Our model	Energy model	Energy/Time model	Time model
Decision error rate	0%	80%	50%	80%

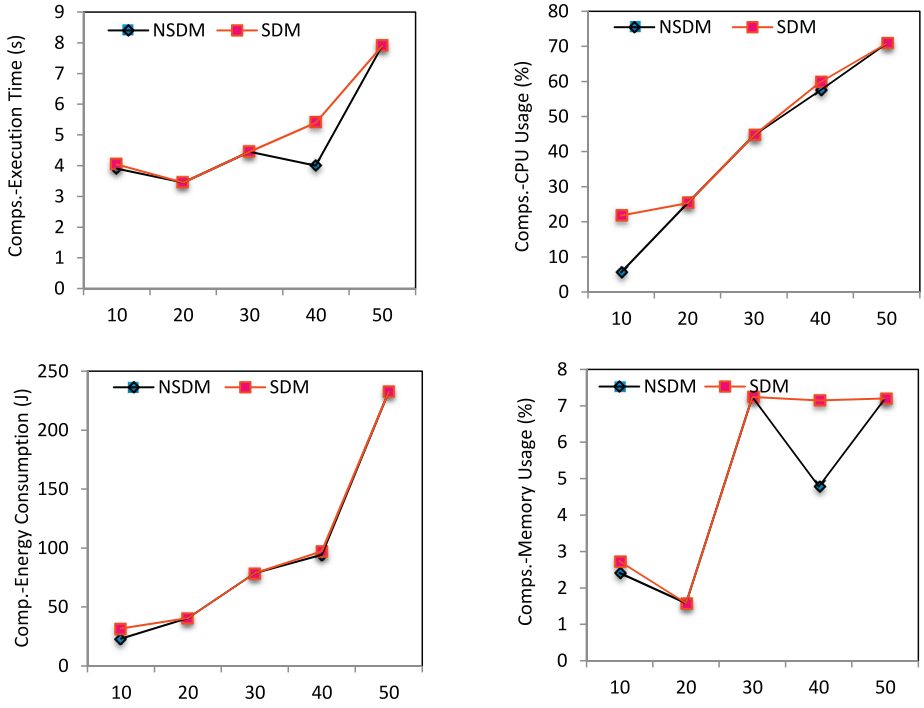

Fig. 5. Components overhead.

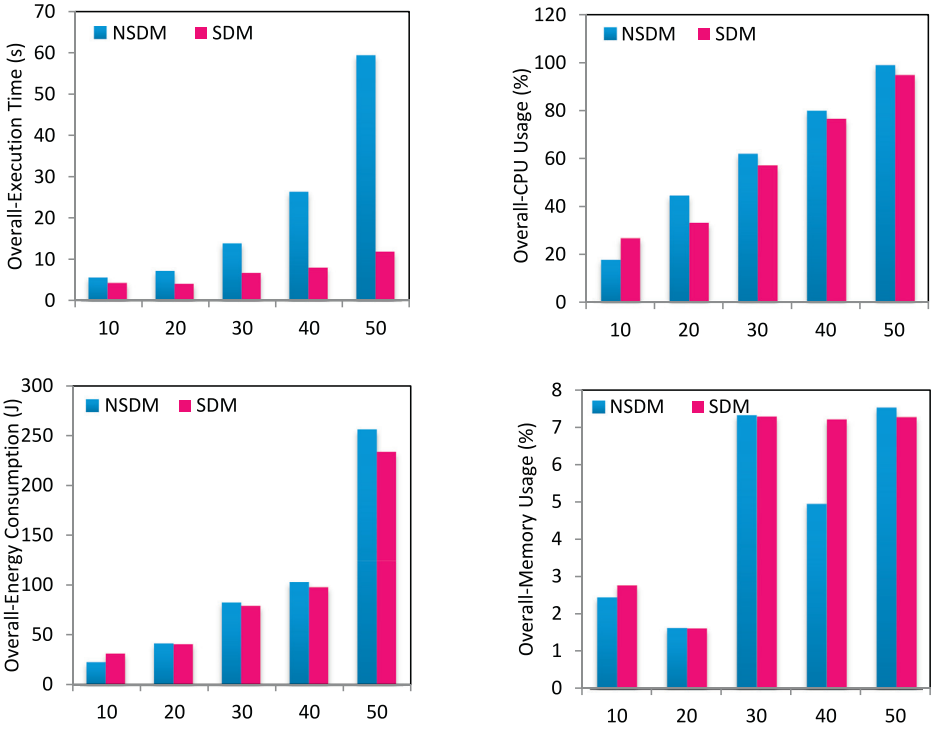

Fig. 6. Overall overhead.

10. Conclusion and future directions

This article goes beyond existing approaches with intelligent system model for computations offloading. The system is able to collectively evaluate offloading tasks from different applications that run on the mobile terminal through centralized selective decision engine. With only hotspots considered in the offloading evaluation process, the proposition is capable of significantly reducing the overhead of the decision engine. The latter evaluates a multi-objective optimization model that includes essential metrics to augment mobile device resources and quality of experience. The model is resilient not only to the device state, but also to the detected hotspots and to strategies that can be enforced on the mobile terminal to prioritize offloading and control the execution of tasks. The model was able to offer optimal dissemination of tasks with 68% reduction in the CPU usage, 15.5% in the memory consumption, 99% in the energy and up to 63% execution speedup, with higher stability compared to existing models. According to the

model, the decision engine decodes, for the designated hotspots, the execution strategy in order to achieve a tradeoff between the proposed metrics. The selective mechanism was able to notably reduce the overhead of the offloading evaluation process with 1.2 times less CPU, 4× less memory consumption, 17× less energy and 13× speedup while the intelligent decision maker was able to adapt to this mechanism and generate the dissemination of tasks with optimal overall savings.

Promisingly the results give guidance to selective optimized system that can run on resource constrained mobile devices to manage applications executions while alleviating the overhead of the offloading evaluation process without jeopardizing the optimal distribution of tasks that minimizes processing, memory, energy loss and speedup the execution on the device. This work opens several research directions that can be considered by the research community. The main objective is to maintain good quality of experience on the device and ensure longer survivability. Therefore, defining and enforcing management policies and rules between components on the mobile device in order to refine the decision would be a valuable track. While there is still no works that give informative decisions in mobile cloud offloading, adaptive policy-based approaches (Cimino, Lazzerini, Marcelloni, & Ciaramella, 2012; Fang et al., 2012) allow managing situations with awareness for proactive and instructive recommendations. For instance, rather than dictating what components to offload, more valuable recommendations can be taken based on user preferences, resource availabilities and device state. Such decisions include suspending and shutting down some applications due to resource scarcity and direct decisions to prioritize the mobile device survivability over applications performance. While assuming independent components on the device reduces cost model complexity, considering the dependencies between components of an application is important. With only few works have been proposed in this regard (Chun et al., 2011; Mahmoodi, Uma, & Subbalakshmi, 2016), dynamic analysis of potential execution flow paths of different tasks in a mobile application has direct impact on the distribution decision where the decision to offload or locally execute particular components can influences the execution of other dependents components.

Acknowledgment

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

Amazon (2016). Amazon EC2 - virtual server hosting. https://aws.amazon.com/ec2/. Accessed: 2016-11-10.

Andreev, S., Pyattaev, A., Johnsson, K., Galinina, O., & Koucheryavy, Y. (2014). Cellular traffic offloading onto network-assisted device-to-device connections. *IEEE Communications Magazine, 52*(4), 20–31.

Android (2016). Power profiles for Android. https://source.android.com/devices/tech/power/index.html. Accessed: 2016-11-10.

Cai, Z., & Chen, C. (2014). Demand-driven task scheduling using 2d chromosome genetic algorithm in mobile cloud. In *Progress in informatics and computing (PIC), 2014 international conference on* (pp. 539–545). IEEE.

Chae, D., Kim, J., Kim, J., Kim, J., Yang, S., Cho, Y., Known, Y., & Paek, Y. (2014). Cmcloud: Cloud platform for cost-effective offloading of mobile applications. In *Cluster, cloud and grid computing (CCGrid), 2014 14th IEEE/ACM international symposium on* (pp. 434–444). IEEE.

Chen, E., Ogata, S., & Horikawa, K. (2012). Offloading android applications to the cloud without customizing android. In *Pervasive computing and communications workshops (PERCOM workshops), 2012 IEEE international conference on* (pp. 788–793). IEEE.

Chun, B.-G., Ihm, S., Maniatis, P., Naik, M., & Patti, A. (2011). Clonecloud: Elastic execution between mobile device and cloud. In *Proceedings of the sixth conference on computer systems* (pp. 301–314). ACM.

Cimino, M. G., Lazzerini, B., Marcelloni, F., & Ciaramella, A. (2012). An adaptive rule-based approach for managing situation-awareness. *Expert Systems with Applications, 39*(12), 10796–10811.

Cuervo, E., Balasubramanian, A., Cho, D.-k., Wolman, A., Saroiu, S., Chandra, R., & Bahl, P. (2010). Maui: Making smartphones last longer with code offload. In *Proceedings of the 8th international conference on mobile systems, applications, and services* (pp. 49–62). ACM.

Deb, K. (1999). An introduction to genetic algorithms. *Sadhana, 24*(4–5), 293–315.

Deb, K., Pratap, A., Agarwal, S., & Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *Evolutionary Computation, IEEE Transactions on, 6*(2), 182–197.

Emmerich, M., Beume, N., & Naujoks, B. (2005). An emo algorithm using the hypervolume measure as selection criterion. In *Evolutionary multi-criterion optimization* (pp. 62–76). Springer.

Fang, B., Liao, S., Xu, K., Cheng, H., Zhu, C., & Chen, H. (2012). A novel mobile recommender system for indoor shopping. *Expert Systems with Applications, 39*(15), 11992–12000.

Fiandrino, C., Kliazovich, D., Bouvry, P., & Zomaya, A. Y. (2015). Network-assisted offloading for mobile cloud applications. In *2015 IEEE international conference on communications (ICC)* (pp. 5833–5838). IEEE.

Flores, H., Hui, P., Tarkoma, S., Li, Y., Srirama, S., & Buyya, R. (2015). Mobile code offloading: From concept to practice and beyond. *Communications Magazine, IEEE, 53*(3), 80–88.

Flores, H., Srirama, S. N., & Buyya, R. (2014). Computational offloading or data binding? Bridging the cloud infrastructure to the proximity of the mobile user. In *Mobile cloud computing, services, and engineering (MobileCloud), 2014 2nd IEEE international conference on* (pp. 10–18). IEEE.

Google (2016). Google cloud platform. https://cloud.google.com/. Accessed: 2016-11-10.

Gordon, M. S., Jamshidi, D. A., Mahlke, S., Mao, Z. M., & Chen, X. (2012). Comet: Code offload by migrating execution transparently. In *Presented as part of the 10th USENIX symposium on operating systems design and implementation (OSDI 12)* (pp. 93–106).

Grefenstette, J. J. (2013). Genetic algorithms and their applications. In *Proceedings of the second international conference on genetic algorithms*. Psychology Press.

Han, B., Hui, P., Kumar, V., Marathe, M. V., Pei, G., & Srinivasan, A. (2010). Cellular traffic offloading through opportunistic communications: A case study. In *Proceedings of the 5th ACM workshop on challenged networks* (pp. 31–38). ACM.

Hung, S.-H., Shieh, J.-P., & Lee, C.-P. (2012). Virtualizing smartphone applications to the cloud. *Computing and Informatics, 30*(6), 1083–1097.

Kemp, R. (2014). Programming frameworks for distributed smartphone computing.

Kosta, S., Aucinas, A., Hui, P., Mortier, R., & Zhang, X. (2012). Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading. In *INFOCOM, 2012 proceedings IEEE* (pp. 945–953). IEEE.

Lust, T., & Teghem, J. (2012). The multiobjective multidimensional knapsack problem: A survey and a new approach. *International Transactions in Operational Research, 19*(4), 495–520.

Mahmoodi, S. E., Uma, R., & Subbalakshmi, K. (2016). Optimal joint scheduling and cloud offloading for mobile applications.

Nebro, A. J., Durillo, J. J., Luna, F., Dorronsoro, B., & Alba, E. (2009). Mocell: A cellular genetic algorithm for multiobjective optimization. *International Journal of Intelligent Systems, 24*(7), 726–746.

Satyanarayanan, M., Bahl, P., Caceres, R., & Davies, N. (2009). The case for vm-based cloudlets in mobile computing. *Pervasive Computing, IEEE, 8*(4), 14–23.

Shi, C., Habak, K., Pandurangan, P., Ammar, M., Naik, M., & Zegura, E. (2014). Cosmos: computation offloading as a service for mobile devices. In *Proceedings of the 15th ACM international symposium on mobile ad hoc networking and computing* (pp. 287–296). ACM.

Tout, H., Talhi, C., Kara, N., & Mourad, A. (2016). Selective mobile cloud offloading to augment multi-persona performance and viability. *Cloud Computing, IEEE Transactions on, PP*(99), 1. doi:10.1109/TCC.2016.2535223.

Wu, C.-W., Chiang, T.-C., & Fu, L.-C. (2014). An ant colony optimization algorithm for multi-objective clustering in mobile ad hoc networks. In *Evolutionary computation (CEC), 2014 IEEE congress on* (pp. 2963–2968). IEEE.

Xia, F., Ding, F., Li, J., Kong, X., Yang, L. T., & Ma, J. (2014). Phone2cloud: Exploiting computation offloading for energy saving on smartphones in mobile cloud computing. *Information Systems Frontiers, 16*(1), 95–111.

Xiang, L., Ye, S., Feng, Y., Li, B., & Li, B. (2014). Ready, set, go: Coalesced offloading from mobile devices to the cloud. In *INFOCOM, 2014 proceedings IEEE* (pp. 2373–2381). IEEE.

Zhang, L., Tiwana, B., Qian, Z., Wang, Z., Dick, R. P., Mao, Z. M., & Yang, L. (2010). Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In *Proceedings of the eighth IEEE/ACM/IFIP international conference on hardware/software codesign and system synthesis* (pp. 105–114). ACM.

Zitzler, E., & Künzli, S. (2004). Indicator-based selection in multiobjective search. In *Parallel problem solving from nature-PPSN VIII* (pp. 832–842). Springer.

Zitzler, E., Laumanns, M., Thiele, L., Zitzler, E., Zitzler, E., Thiele, L., & Thiele, L. (2001). Spea2: Improving the strength pareto evolutionary algorithm.