A Novel Model for Computational Offloading on Autonomic Managers: a Mobile Test Bed

Guilherme Antonio Borges, Claudio F. R. Geyer
Institute of Informatics
Federal University of Rio Grande do Sul
Porto Alegre, Brazil 91509–900
Email: {geyer, gaborges}@inf.ufrgs.br

Rômulo Reis de Oliveira, Tiago Coelho Ferreto
School of Computer Science
Pontifical Catholic University of Rio Grande do Sul
Porto Alegre, Brazil 90619–900
Email: romulo.reis@acad.pucrs.br, tiago.ferreto@pucrs.br

***Abstract*—The technological advance of mobile devices, networks and cloud technologies progressed and reduced their access costs to the whole human population. Besides that, mobile devices are still limited in the battery capacity, storage, and connectivity. An efficient way to manage their resources is to make the applications self-adaptive and context-aware using the MAPE-K loop model. However, even this method can add a considerable processing cost to their devices. This paper proposes to reduce such costs by applying the computational offloading technique into the classical MAPE-K loop. In this way, we analyzed it based on literature evidence to find a suitable process that allows offloading to remote and cloud servers. The results through experimentation on the proposed model show that there is a substantial performance increasing in the planning activity remote executions compared to the local ones, what is also affected by the distance from the servers.**

***Index Terms*—Autonomic Computing; MAPE-K; Code Offloading; Mobile Cloud Computing**

I. Introduction

Nowadays, the execution of parallel applications and background services, both with access to high-speed networks and sensors, have been feasible to most of the new generations of mobile devices. These factors are contributing to more complex and highly distributed applications being studied and built in both, industry and academy.

Despite these technological advances in mobile devices, the ubiquity and sophistication of new applications are leading them towards the limits of performance, storage, and energy usage [1]. Especially concerning the battery lifetime improvement, both processor and network speeds increased coarsely one thousand fold since 2001, but typical rechargeable battery energy density has only doubled in the same period [2]. As most of the mobile device operations depend on its energy capacity, a naive usage of the available resources can quickly drain its energy. A context-aware software may drive efficient resource management in these situations.

Even when we achieve an efficient management through context-awareness, as the system complexity grows, the human effort to deal with system tasks (i.e., configuration, execution, and maintenance) and the error-prone also grow. It may escalate until it is virtually impossible for users and administrators manually manage the entire system [3], [4]. A way to reduce this human effort is making it a self-adaptive software (SAS).

A SAS aims to find better runtime configurations in response to environmental changes [5]. This type of software usually follows the autonomic computing reference model, which, according to Kephart and Chess [6], it conceptually separates the managed elements from the engine that performs the management, also known as the autonomic manager. The managers use an optimization process to provide an efficient decision making. However, such process is also affected by the growth of system complexity, what can be very costly for resource-limited devices executing real-time adaptations.

For a better understanding of the problem, let be consider the results presented by Pascual et al. [7]. They have executed experiments in a smartphone LG Nexus 5 to optimize the metrics of battery, memory, and usability by applying the algorithms PAES [8] and NSGA-II [9]. The goal was to analyze the optimization process. Their results showed that the average time range from 2 to 5 seconds when considering small benchmarks of feature models. Though, when considering a larger benchmark of feature model (i.e., 500 features), the average time reached approximately 32 seconds.

How can we reduce the computational time and resource cost of this process on mobile devices? In the self-adaptive literature, many approaches exist aiming to make the optimization process less costly. However, to the best of our knowledge, the computational offloading approach was not used to reduce the cost in autonomic managers for mobile devices.

The Mobile Cloud Computing (MCC) research field has shown results in the reduction of processing time and energy usage in mobile devices for processing-intensive applications through of offloading functions to an external computer with higher processing capacity (e.g., cloud, cloudlet) [10]. To exhibit its advantage, Ha et al. [11] demonstrate experiments in ideal conditions for two intensive functions using a mobile device Samsung Galaxy S2. To augmented reality functions were measured the average energy usage of 1.1 J (Joule) offloading to a Cloudlet, 3.1 J to the closest Amazon computational cloud, and 3.3 J executing only in the mobile device. For face recognition functions a more significant difference was measured considering the average energy usage: 5.4 J offloading to a Cloudlet, 6.6 J to the closest Amazon computational cloud, and 16.4 J executing only in the device.

These two functions are typically used as specific parts of applications, and they can affect the overhaul performance of the whole application if not offloaded. Similarly, the before mentioned optimization process can cost a substantial processing time and battery to its mobile device, and it is a part of the

MAPE-K's planning activity. Thus, it is intuitive to infer that this process also can be remotely executed to use less local resources and time. Although, would be advantageous offload the optimization process of the autonomic manager?

To answer such a question, in the present work we propose an extension of the MAPE-K architecture aiming to verify the viability of the offloading technique applied on the autonomic manager optimization process. As our focus is to evaluate it in ideal conditions, we do not approach the decision-making module to offload in this paper in a dynamically way.

For that, the paper is organized as follows. In Section 2 reviews the background of offloading and self-adaptive systems. Section 3 discusses the related works that treat the space complexity from the planning activity. Section 4 presents an offloading model incorporated into the autonomic manager model. Section 5 shows the proof-of-concept, preliminary experiments; as well the results discussion. Section 6 discusses the conclusion and future works.

II. Background

In this section the basics of Self-Adaptation and Mobile Cloud Computing are addressed, both are used to compose the approach presented in Section 4.

A. Self-Adaptive Systems

Self-adaptive systems (SAS), also referred to as autonomic computing, is the research area that seeks to treat the growth of system complexity by making them monitoring itself and the operational environment, taking appropriate actions when circumstances change [6], [4]. The model usually separate the managed elements from the autonomic managers, which perform the management.

Each autonomic manager allows adaptation through four activities/steps: monitor, analyze, plan and execute, all of them with access to a knowledge base. These activities and the knowledge base comprise an adaptation control loop model known as MAPE-K. In the monitoring activity, elements collect relevant data via sensors to reflect the current state of the observable system, including the managed elements. In the analyzing activity, states are monitored and evaluated; if undesired states are detected, one or more new (desired) target states are specified. The planning activity decides the necessary steps to adapt, moving the software towards the desired state; it usually is done through techniques of planning and optimization. The execution activity performs the decided adaptation actions into the managed element through actuators or effectors [12].

B. Mobile Cloud Computing and Mobile Edge Computing

Mobile Cloud Computing is the research area that integrates the cloud computing technology with mobile devices to make them resource-full regarding computation power, memory, storage, energy and context awareness [13]. For this purpose, the MCC uses computational offloading operations to migrate resource-intensive functions from mobile devices to more resource-rich computers located in the cloud [14].

In this way, a process is expected to perform better in the cloud infrastructure than a mobile device. However, the network access delay between the device and the cloud has also to be taken into consideration, since the cloud infrastructure can often be physically far from the device, and further it is, longer is the access delay.

Mobile Edge Computing (MEC) is an emerging paradigm that brings substantial computing and storage resources at the edge of the network, close to mobile devices or sensors, what allows lower latency times for operations as a computational offloading compared to those performed on cloud servers [15]. These closer computers are variously referred in the literature as cloudlets, micro datacenters, or fog nodes [15].

However, not all functions have advantages to being offloaded into external computers. Even those who do it, the variations in the operating environment are recommended to be considered by them. Otherwise, they can lead to poorer performance than local executions. Some examples of unfavorable conditions are: (1) when the device does not have the network connection, and (2) when the latency is higher than the local execution time. For these reasons, an engine is usually used to evaluate the conditions and decide whether is or not favorable to execute the offload operation [10].

III. Related Work

The MAPE-K planning activity performs the generation of configurations for decision making. Ideally, an optimal configuration generated by an exact algorithm would be the best; nevertheless, it is a practice only for small scales of P problems. In larger scales, when it considers all possible combinations of actions and functional requirements, the number of configurations can become quite large; which makes it unfeasible to evaluate all of them at runtime. As it is an NP-hard problem, it requires heuristic approaches to achieve near-optimal results at runtime [16], [17]. Table I presents these approaches found in the literature of self-adaptive systems. We selected these works by their relevance to the research field of SAS and their similarity with the subject of this paper.

For reducing the number of possible combinations, the works [21], and [23] use plans entirely defined in development time. At runtime time, they make decisions by using action policies, which also must be previously defined. Differently, Rouvoy et al. [18] reduce the configuration space by chosen a subset of all possible combinations of development time, which is evaluated in real time through a utility function. However, to ensure that such set comprises the optimal values according to the defined objectives and all the possible context changes that can occur in real time tend to become harder to identify in development time as the space of possible configurations increases.

The works [19], [27], [20], and [22] use a method partially at runtime. Saller et al. [19] reduce the spatial complexity of the possible configurations and contexts by using development time steps based on the results of runtime steps aiming to apply algorithms closer to the exact ones. Alfrez et al. [22] use semantic logic rules refined on development time to restrict the range of configurations for selection at runtime. Similarly, Rosa et al. [20] use off-line steps to analyze the performance of the rules, excluding those who does not reach the goal. Although, at runtime, they allow to chose the performance evaluation of all rules against the context changes by brute

TABLE I
RELATED WORKS

Approach	Year	Generation of Configuration	Analysis Technique	Optimization Mechanism
[18]	2009	Development time	Utility function	Brute search and heuristic
[19]	2013	Partially at development time	Cost function	Reduction of space configuration
[20]	2013	Partially at development time	Rules (Action policies)	Brute force to optimization and objective function
[17]	2013	Runtime	Utility function	Prediction through stochastic algorithms and heuristic search
[21]	2014	Development time	Rules (Action policies)	Without support
[22]	2014	Partially at development time	Rules (Action policies)	Objective function by using constraint logic
[23]	2014	Development time	Rules (Action policies)	Without support
[24]	2015	Runtime	Utility function	Genetic algorithm (DAGAME)
[7]	2015	Runtime	Multi-objective function	Multi-objective evolutionary algorithms
[25]	2015	Runtime	Utility function	Discrete swarm particle
[1]	2016	Runtime	Cost/benefit function	Prediction through stochastic algorithms and reinforcement learning
[26]	2017	Runtime	Rules and utility functions	Reinforcement learning and case base reasoning

force or by using heuristic search, which executes until the first rule achieves the goal.

Despite the previous methods being alternatives from the full generation of configurations at development time, the partial generation also can be a time-consuming task, since it still requires human participation. Furthermore, it is still difficult to predict which configurations will include the optimum values based on the defined objectives and all the possibilities of context changes.

Otherwise, the works [17], [7], [24], [25], [26], and [1] generate the configurations at runtime, once specified the environment model. All of them use heuristics to find approximations of the optimal configurations, avoiding the needed for human participation in the adaptation policy refinement. To evaluate their environment, they perform, in the analysis activity, functions to express quality values in the form of fitness, cost, or utility, allowing the search for better results.

Specifically, the works [17], [7], [25], [26], and [1] use those functions to solve multi-objective problems by looking for selected configurations that meet non-functional requirements of many resources at runtime. However, these types of solutions tend to have scalability issues due to the growth of goals and actions, which is especially worrying in resource-contained devices such as mobile devices.

Although these works tried to make a complete optimization and configuration generation in the planning phase, the smartphone has resource limits to execute such techniques, what restrict their possibilities of applications. As we mentioned in the introduction, the computational offloading technique can increase the performance of resource-intensive functions, and it can be an alternative to overcome the resource limitation. However, to the best of our knowledge, none of the related works tried to apply it aiming to improve the time performance of the planning phase. Then, to expand the perspective of these works, we propose in the next section a novel model to explore the new possibilities and limitations of applying such technique.

IV. MODEL

Figure 1 illustrates the proposed model based on the MAPE-K loop. To distinguish the classical model from the one proposed in this section, we use the term **phase** instead of **activity**. The process starts at the monitoring phase, which collects contexts from the environment and managed elements, generating events. The analysis phase processes such events, firstly, to identify if any reconfiguration criteria are satisfied, stopping the process if no adaptation is needed. Otherwise, it decides whether the computational offloading criteria is satisfied. It results in two possible planning executions: (1) when not satisfied, the planning phase is executed locally at the host device, otherwise, (2) the planning is offloaded to a remote server. In both cases, a selected configuration is sent to the device's execution engine to apply the required changes at the managed elements. All these steps have access to a knowledge base, which contains all previous loaded environment models and information that are used to assist the runtime reconfigurations.

From the aforementioned process, it is important to highlight that the alterations added to MAPE-K loop focus on the analysis and planning activities. Thus, all other activities performed in the classic model, and their variations that preserve the original purpose, can be adapted for the context presented by this paper. On the other hand, despite the knowledge base being unaltered in our proposed model, in its implementation, its information still needs to be evaluated and replicated in the remote server related to the planning phase to reduce data transported by the network in runtime. Sections IV-A and IV-B present the proposed modifications in more detail.

A. *Offloading decision process*

The offload decision process was added to the analysis activity of the original MAPE-K loop, right after an adaptation request occurs - i.e., in the "Is offloading criteria satisfied" box from Figure 1. It is particularly relevant to be considered on the model because not all situations benefit from performing a computational offloading. For example, the unstable bandwidth of wireless networks and intermittent connections can reduce the gains derived from using offloading what can lead to worse execution times compared to local executions [10]. Also, the heterogeneity of devices and cloud servers is a significant matter of concern to the decision process, once it is available in the market hardware from low-cost to the more expensive and high-technological.

In this way, for real-world applications, there is no doubt that the context-awareness is needed to ensure a better decision-making process [1]. However, proposing a state of the art decision process requires its own research and experimentation, as it is found in the survey of Bhattacharya

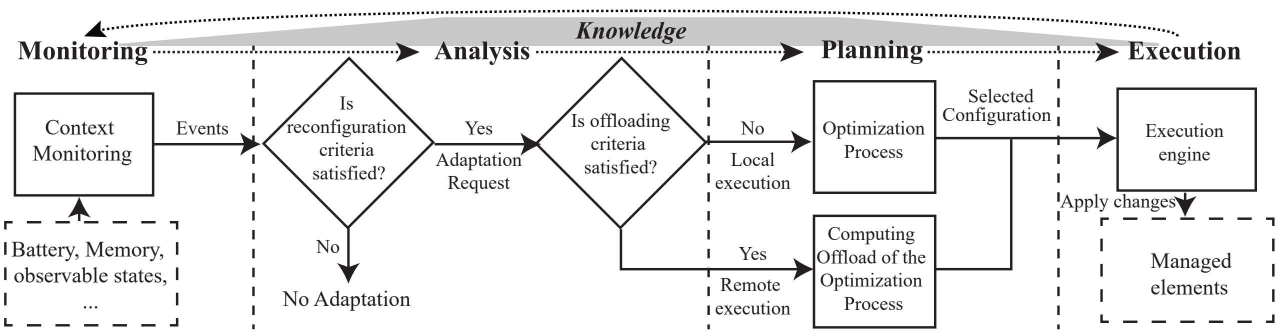

Fig. 1. Proposed general model of the autonomic manager with computational offloading support

and De [10]. Thus, since our contribution is to validate the proposed model and evaluate the computational (code) offloading technique performance of the optimization process of the MAPE-K planning activity in ideal conditions, for the sake of our evaluation, a naive approach is used. That is, when the offloading feature is enabled, it will always perform the offloading process; otherwise, it will perform the local process.

B. Optimization Process

At the planning phase, some of the previously mentioned related works apply an optimization mechanism at runtime to achieve better configuration plans. Among them, the works [17], [7], [25], and [1] use the optimization functions to solve multi-objective problems by looking for selected configurations that meet non-functional requirements of many resources at runtime. However, these types of solutions tend to have scalability issues due to the growth of goals and actions, which is a matter of concern in resource-contained devices.

To address this issue, we propose the use of the computational offloading technique in the planning activity optimization process for configuration generation, which is the most resource-intensive process from such model. This affirmation is based on the result times presented in Pascual et al. [7] that ranged roughly from 2 to 32 seconds according to the feature model used in the performance evaluation.

To better understand our approach, Figure 2 illustrates the proposed modification for offloading the optimization mechanism presented by Pascual et al. [7], which uses Multi-Objective Evolutionary Algorithms (MOEA). As beforementioned, the analysis phase has to choose between local (light gray) and remote (dark gray) execution. Both processes need parameters to execute and generate a set of valid and optimal Feature Model (FM) configurations considering multiple objectives. The set is evaluated by the autonomic manager's decision-making process to find an appropriated configuration to the adaptation. It is executed locally to allow a more flexible decision-making process. Thus, applications can prioritize different objectives depending on their current contexts. For example, if the device is running on low battery power, the application can prioritize the configuration that best saves device's energy. Otherwise, if such device is fully charged, it is likely to choose a balanced configuration from the result set. Once the configuration has been selected, the

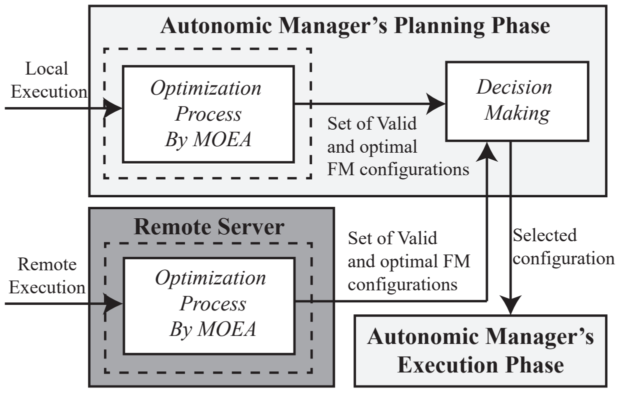
Autonomic Manager's Planning Phase
Local Execution
Optimization Process By MOEA
Set of Valid and optimal FM configurations
Decision Making
Remote Server
Remote Execution
Optimization Process By MOEA
Set of Valid and optimal FM configurations
Selected configuration
Autonomic Manager's Execution Phase

Fig. 2. Example of autonomic manager planning phase with offload support

Autonomic Manager's Execution phase can reconfigure the managed elements.

The main difference from remote execution to the local one is that it requires data transportation to execute the optimization process and return its results to the mobile device for decision-making. Thus, evaluating the data dependence of the offloaded algorithm can lead to better performance, because some of the required data may already be located on the server at the time of the computational offloading.

To illustrate it with more detail, Algorithm 1 presents the NSGA-II pseudocode with the Fix operation added by [7] in the original version of the algorithm [9] to avoid that configurations violate the FM constraints by fixing them. Such algorithm requires at least the specification of the parameters *psize*, *pcrossover*, *pmutation* and *evalmax*. *Psize* determines the size of the population; the *pcrossover* is the reference to which crossover method is going be used; *pmutation* indicates which mutation method is applied to the populations, and *evalmax* determines the maximum number of evaluated generations. They all are integer parameters and do not add much data to network transport; so they can be transported on the fly to the server.

In the case of [7], an FM must be specified to generate a new solution (line 4), select new configurations (line 13), and perform Fix operations (lines 5 and 16). Since FMs usually do not change at runtime, they can be previously located on

Algorithm 1 NSGA-II with the fix operator proposed by [7].

```
Require: $psize$, $pcrossover$, $pmutation$, $evalmax$
Ensure: $PF$ (a set of non dominated solutions)
 1: $P = \emptyset$
 2: $evaluations = 0$
 3: for $i$ = 1 to $Psize/2$ do
 4:     $s$ = NewSolution()
 5:     $s$ = Fix($s$)
 6:     EvaluateFitness()
 7:     $evaluations = evaluations + 1$
 8:     $P = P + s$
 9: end for
10: while $evaluations < evalmax$ do
11:     $PO = \emptyset$
12:     for $i$ = 1 to $psize/2$ do
13:         $parents$ = Selection(P)
14:         $offspring$ = Crossover($parents$, $pcrossover$)
15:         $offspring$ = Mutation($offspring$, $pmutation$)
16:         $offspring$ = Fix($offspring$)
17:         EvaluateFitness($offspring$)
18:         $evaluations = evaluations + 1$
19:         $PO = PO$ + $offspring$
20:     end for
21:     $P = P \cup PO$
22:     RankingAndCrowdingDistance($P$)
23: end while
24: $PF$ = BestFront($P$)
```

the server at development time. Thus, just the identification can be transported at runtime instead of a whole model. For the fitness evaluation of the configurations (lines 6 and 17), a fitness function is required for each FM. These functions can also be previously loaded on the server; however, in the applications that need to change them at runtime, a better strategy is to upload them from the device.

All the aforementioned FMs as well the cross-tree constraints used in the Fix operations are located on the knowledge base and are used in both the remote server and the local device. These models are examples of parts of the knowledge base that after evaluated can be replicated into the remote server to reduce data sent by the network in real time.

Finally, in the case of the autonomic manager optimization, the current configuration is usually used as initial population for the algorithm. This occurs to challenge the current configuration, what add more chances to get better results than the already obtained. In this way, the current configuration can also be transported to the server at runtime to replace the new configuration at line 4.

V. Environment Setup

A. Implementation and infrastructure

The model was implemented as an extension from MO-DAGAME[1] proposed by Pascual et al. [7]. The MO-DAGAME is a framework based on the JMetal framework with the focus on Dynamic Software Product Lines problems using feature models. The available Android code was extended to support both offload and local execution in mobile devices. The available source code to desktop was extended to act as an offload server built using the Java language, receiving by socket connections the input data in JSON (JavaScript Object Notation) format. The extended codes are available in our repository on github[2] under the GPLv3 license.

The infrastructure used in the experiments is composed of smartphones and servers connected by a dedicated D-Link DIR-615:T1 wireless router. Table II shows the specifications of the smartphones, labored by letters A and B. Their hardware and software differences are used to reduce the evaluation bias, and they are not proper to compare themselves. It is also important to note that our focus is comparing their processing time with those from the remote servers.

As remote servers to process the offloading operations, we used two computers with the OS Ubuntu 16.04 64 bits. The first computer is a virtual machine F2 of the nearest Microsoft Azure Cloud service (CS), located in the South of Brazil. The second computer is a dedicated remote server used to simulate an environment of Mobile Edge Computing, which we label as Edge Server (ES). Table III detail their hardware.

B. Algorithms and data input

The algorithms and the data described in this section are based on those presented in [7]. Each line informs the FM name, the number of features and the number of possible configuration results. For the evaluation, we selected nine feature models, as showed in Table IV. From 1 to 8 are real-world feature models, including FMs specifics for mobile devices (5, 7 and 8). However, the FM 9 is one of the benchmarks randomly generated from SPLOT[3] and include more features than those found in the other FMs.

For the evaluation, each FM uses three optimization objectives as attributes. By considering them, the framework should: maximize the usability, minimize the battery consumption and minimize memory usage. Each of them can assume the following values:

- *Usability* measures, among others, how easy and satisfying to use is the application. It takes real values between 0 and 10, according to a normal distribution;
- The *Battery consumption* introduced by the feature, measured in milliamps. It takes real values between 10.0 and 20.0 according to a normal distribution;
- The *Memory footprint*, in megabytes, introduced by the feature. It takes real values between 0.0 and 10.0 according to a normal distribution.

As they do not change at runtime, both FM and its objective functions are available on the server and the mobile device in development time. In this way, they are loaded without the need of data transportation to perform the optimization process. For the evaluation, we chose the two faster Multi-objective Evolutionary Algorithms from MO-DAGAME framework to mobile devices presented by Pascual et al. [7]: (1) Nondominated Sorting Genetic Algorithm II (NSGA-II) [9]; (2) Pareto Archived Evolution Strategy (PAES) [8].

[1]http://caosd.lcc.uma.es/projects/famware/tools.htm

[2]https://github.com/gaborges/MO-DAGAME-Offload

[3]http://www.splot-research.org/

TABLE II
SPECIFICATION OF THE SMARTPHONES USED IN THE EXPERIMENTS

Label	Model	Operational System	CPU	RAM
Smartphone A	Moto E2 (2015)	Android 5.1.1	Quad-Core 1.2 GHz ARM Cortex-A7	1 GB
Smartphone B	Moto G5 Plus (2017)	Android 7.1	Octa-Core 2.0 GHz ARM Cortex-A53	2 GB

TABLE III
SERVERS' HARDWARE SPECIFICATION

Label	CPU	RAM	Storage
Cloud Server (CS)	2 cores of a Intel Xeon E5-2673 v3, 2.4 GHz (3.2 GHz Intel Turbo Boost)	4GB	SSD (Solid-State Drive) 32GB
Edge Server (ES)	Intel Core i5-4460 with 4 cores, 3.2 GHz (3.4 GHz Intel Turbo Boost)	16GB	HD (Hard-Disk) 1TB

TABLE IV
FEATURE MODELS FOR EVALUATION

	Name	Features	Possible Configurations
1	x264	17	2048
2	Wget	17	8192
3	Berkeley DB Memory	19	3840
4	Sensor Network	27	19152
5	Mobile Game	33	9198
6	Tank War	37	1,741,824
7	Mobile Media	43	2,128,896
8	Mobile Visit Guide	51	33,800,000
9	SPLOT-3CNF-500	500	3.779e15

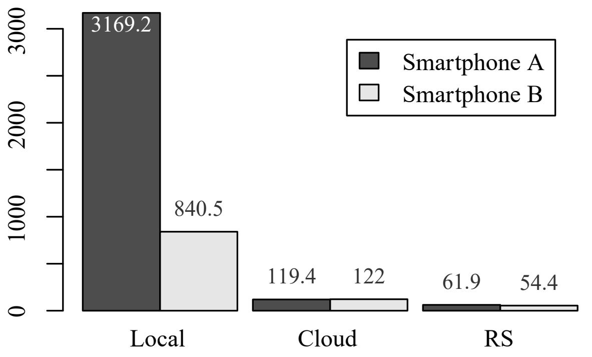

Fig. 3. Summary of the average execution times in milliseconds from the FM 2 using the algorithm NSGA-II

Furthermore, in order to improve the quality and reliability of the solutions generated by these MOEAs, we also have applied a seeding technique used by [7], which in our approach works as follows:

1) A valid configuration (the seed), which includes around 50% of the features in the FM, is pre-computed for each FM. They are initially located into the mobile device, however, when an offloading operation is performed, the respective FM seed is transported to the server;
2) The initial population is filled using the seed: (a) A mutation is introduced in the seed. (b) The resulting configuration is repaired using the fix operator. (c) The fixed configuration is added to the initial population.

We applied the following parameters for all algorithms. The Single-Point Crossover parameter for executing the crossover operator, with a crossover probability of 0.8. The BitFlip Mutation operator with a probability of 0.1 was applied for the mutation. The number of allowed fitness evaluations is 5000, and the population size is 100. Specifically to the PAES, the parameter of archive size is 100, and the bisections are 5.

As the objective of this paper is not tuning the values to improve the performance of particular algorithms and FMs, we compare the local and remote executions using default parameter values.

VI. EXPERIMENTAL RESULTS

In this section, we present the experimental results from the modification of the planning phase and discuss the limitations of the proposed model. Table V shows the average execution time results in milliseconds, where each line represents the combination of the mobile devices (MD) with an algorithm executed locally, in the CS or in the ES. The columns from 1 to 9 represent the evaluated feature model. The average execution time was calculated using the function "summarize" from the R language based on 100 independent runs.

As expected, in the evaluation that only considers the local execution, the high-technological smartphone B has better performance than the low-cost smartphone A.

By considering the local execution results, it is possible to note that the algorithm NSGA-II has better processing times in most cases than the algorithm PAES for the used FMs. Despite this, they are both slow when compared to the remote execution times. As they are the fastest algorithms from the MO-DAGAME benchmark [7], we can infer that the remaining ones can have a speed up when offloaded.

In both cases, the remote executions show performance improvement. For the sake of argument, we selected the fastest local average execution times from Table V using the algorithm NSGA-II, which belongs to the FM2. Figure 3 summarizes them with the corresponding CS and ES execution times. When we compare these local results to the results from the CS, we measure a speedup of 6.88 times to smartphone B and 26.54 times to smartphone A. On another hand, When we compare the same local times to those of the ES, we measure a speedup of 15.44 times to smartphone B and 51.17 times to smartphone A. This happens because both remote servers have more processing capacity and fewer limitations on storage and memory than both smartphones.

Besides the heterogeneity of hardware, software, and manufacture among the smartphones used in the experiment, the results showed that they present a similar behavior when offloading to a remote server. That is, all offloading processing

TABLE V
AVERAGE TIME OF THE EXPERIMENT'S RESULTS IN MILLISECONDS (MS)

MD	Execution	1	2	3	4	5	6	7	8	9	Algorithm
A	Local	3192.49	3169.26	3429.43	3961.05	4329.04	4625.00	5124.90	5615.28	55395.39	NSGA-II
B	Local	851.21	840.50	904.05	1076.99	1177.04	1262.14	1417.94	1563.32	19368.03	NSGA-II
A	CS	142.39	119.40	125.58	147.87	164.32	171.33	195.51	195.73	1836.23	NSGA-II
B	CS	140.93	122.04	149.91	149.15	165.90	175.85	200.86	211.39	1842.20	NSGA-II
A	ES	88.98	61.93	64.18	94.83	79.57	101.83	116.16	120.25	1524.06	NSGA-II
B	ES	66.46	54.43	61.02	78.54	96.49	100.98	118.34	118.47	1529.62	NSGA-II
A	Local	3699.99	4722.11	3463.69	5904.09	5045.67	5107.39	5243.13	6266.90	52906.77	PAES
B	Local	1325.22	1775.04	1173.59	2096.18	1709.73	1743.49	1736.89	2029.93	18152.89	PAES
A	CS	244.87	226.55	168.87	272.04	232.31	235.56	246.70	283.50	1797.98	PAES
B	CS	198.10	215.85	189.59	260.51	231.74	249.51	240.56	261.34	1789.46	PAES
A	ES	109.83	117.28	85.70	163.09	136.89	138.34	153.86	168.73	1416.34	PAES
B	ES	100.84	118.38	91.72	151.15	139.46	137.11	146.98	155.94	1402.85	PAES

TABLE VI
AVERAGE LATENCY TIME OF OFFLOADING OPERATIONS IN MS

FMs	Minimum	Average	Maximum	SD	Target
1-8	62.20	88.79	5110.71	99.11	Cloud
9	124.98	268.05	5335.07	215.48	Cloud
1-8	5.92	17.29	3173.31	50.48	ES
9	14.42	110.96	3221.49	176.34	ES

times to the same remote servers among all experiments have fewer execution time variations when compared to local ones. It happens because the hardware in the servers is always the same to all smartphones. In this way, we can infer that the processing capacity of each server plus the network latency time define most of the offloading execution time.

The latency time is a relevant information about the comparison among the remote servers, which is the network delay time caused by the distance between end-to-end pairs. For the CS, it is higher than for the ES because the CS is more centered and far away from the requester than the ES. For more detail, see Table VI, which shows the minimum, average, maximum and standard deviation (SD) of the latency time, analyzed by the method summarize from the language R. We separated the processed sets in two group of FMs, 1-8 and 9. Once the FM 9 has far more data from its current state to send by the network than the others, merge all of them would add bias to the evaluation. The results confirm the expected lower latency of the ES experiments due to its closer distance from the smartphones.

However, there is still a difference in the execution times between both servers that are greater than the latency time. It can be explained by the difference of hardware among them, where the ES can processing 3.4 GHz and the CS 3.2 GHz by using the Turbo Boost Technology. Also, as the ES is not virtualized, it does not add the virtualization overhead in its times as the CS does. As the process of optimization was executed in memory, the difference of SSD in the CS does not seem to have influenced in the processing time.

A. Discussion and Limitations

As general guidelines, the results presented in this paper show that the proposed model exhibits a performance increase in ideal conditions when performing the offloading operations. These results validate our model, nevertheless, it is necessary to highlight that this is the first model witch explores such technique to decentralize the adaptation logic of the MAPE-K model, so further investigations are needed to find the possibilities of derived models, frameworks, and middleware. In time, it will benefit the resource usage management in resource-constrained devices as we can find in the pervasive and mobile environment.

Considering the offloading technique, the latency time added to the response times can lead to worse response times when an offloaded function has to communicate often to obtain the desired result. It is particularly worrying for far away servers as can be found in some cloud computing scenarios. The ideal situation is that the needed data, such as the user profile history, should be transported to the server before its offloading operation. In these scenarios, we can use the monitoring pattern presented in [28], which is easier to implement in cloud servers due to its centered characteristic. However, the implementation in the MEC server needs a more complex deployment due to its restricted coverage and decentralized nature, which leads to the need for mobility management of the monitored data to perform offloading operations. Thus, experiments are required to evaluate the cost of transporting this data using such pattern in both cases.

We achieved positive results due to some facts: the optimization process of the Planning activity being a computational-intensive function, that - after the analysis in the Section IV-B - can be remotely executed without intermediate data dependencies, and it was implemented (and deployed) in an ideal environment to offloading operations. Unfortunately, the real-world is far from always being a perfect environment to offload. However, we needed to consider it to validate the problem and the model before being justifiable the usage of a more sophisticated decision-making mechanism on it. As explained in the Section IV-A, it was not the focus of this paper, also because it requires proper research and experimentation.

The scalability of the offloading operations on the server side is also a major topic to be evaluated in future research. The reason for this happens because there are conditions in the real-world environment that both of our remote servers do not always will show better time results than the local ones. For example, multiple concurrent clients can access the same server, overloading it and increasing the response time.

Also, if this server is capable of processing different types of offloading operations, even if they focus on using other resources (i.e., CPU, memory, i/o,...), they still can interfere in the others offloading operations.

VII. Conclusion and Future Works

In this paper, we proposed a modification of the MAPE-K loop architecture to support the computational offloading technique for resource-poor devices, such as smartphones, tablets and the Internet of Thing devices. To evaluate this approach, we extended the framework and benchmark MO-DAGAME to allow computational offloading to cloud and remote servers.

Through experiments in ideal conditions, we have verified the viability of this technique applied to the autonomic manager's optimization process. The gathered results showed that there is a significant performance improvement by using the proposed approach in both ES and cloud servers. We also confirmed the low-latency characteristic of the ES over the cloud with similar processing capacity, which can benefit latency-sensitive applications.

The proposed model opens new possibilities to explore combinations of the self-adaptation processes and computational offloading techniques to reduce the resource usage on resource-poor devices. In future works, we intend to evaluate the scalability support of the approach in the cloud computing and MEC environments, as well as consider the concurrency with other applications. The mobility is also a significant concern, specifically to MEC. We also intend to seek out a more efficient and context-aware decision-making process, which is a real-world requirement for resource management.

VIII. Acknowledgment

The authors would like to thank Dell Inc and the following Brazilian Agencies:FAPERGS Projects "GREEN-CLOUD - Computação em Cloud com Computação Sustentável" (#16/2551-0000 488-9) and "SmartSent" (#17/2551-0001 195-3), CAPES, CNPq and PROPESQ-UFRGS-Brasil for supporting this work.

References

[1] N. Z. Naqvi, J. Devlieghere, D. Preuveneers, and Y. Berbers, "MasCOT: Self-adaptive opportunistic offloading for cloud-enabled smart mobile applications with probabilistic graphical models at runtime," in *Proceedings of the 49th Hawaii International Conference on System Sciences (HICSS)*. IEEE, 2016, pp. 5701–5710.

[2] R. Chandra, S. Hodges, A. Badam, and J. Huang, "Offloading to improve the battery life of mobile devices," *International Journal of Pervasive Computing*, vol. 15, no. 4, pp. 5–9, 2016.

[3] K. Saller, "Model-based runtime adaptation of resource constrained devices," Ph.D. dissertation, Technische Universität, Darmstadt, 2015. [Online]. Available: http://tuprints.ulb.tu-darmstadt.de/4322/

[4] C. Krupitzer, F. M. Roth, S. VanSyckel, G. Schiele, and C. Becker, "A survey on engineering approaches for self-adaptive systems," *Journal of Pervasive and Mobile Computing*, vol. 17, pp. 184–206, 2015.

[5] R. Laddaga, P. Robertson, and H. Shrobe, "Self-adaptive software," *Proposer Information Pamphlet BAA*, 1997.

[6] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[7] G. G. Pascual, R. E. Lopez-Herrejon, M. Pinto, L. Fuentes, and A. Egyed, "Applying multiobjective evolutionary algorithms to dynamic software product lines for reconfiguring mobile applications," *Journal of Systems and Software*, vol. 103, pp. 392–411, 2015.

[8] J. D. Knowles and D. W. Corne, "Approximating the nondominated front using the pareto archived evolution strategy," *Journal of Evolutionary Computation*, vol. 8, no. 2, pp. 149–172, 2000.

[9] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE transactions on evolutionary computation*, vol. 6, no. 2, pp. 182–197, 2002.

[10] A. Bhattacharya and P. De, "A survey of adaptation techniques in computation offloading," *Journal of Network and Computer Applications*, vol. 78, pp. 97–115, 2017.

[11] K. Ha, P. Pillai, G. Lewis, S. Simanta, S. Clinch, N. Davies, and M. Satyanarayanan, "The impact of mobile multimedia applications on data center consolidation," in *Proceedings of the International Conference on Cloud Engineering (IC2E)*. IEEE, 2013, pp. 166–176.

[12] J. M. Portocarrero, F. C. Delicato, P. F. Pires, N. Gámez, L. Fuentes, D. Ludovino, and P. Ferreira, "Autonomic wireless sensor networks: a systematic literature review," *Journal of Sensors*, vol. 2014, 2014.

[13] M. Othman, S. A. Madani, S. U. Khan *et al.*, "A survey of mobile cloud computing application models," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 1, pp. 393–413, 2014.

[14] L. Ricci and E. Carlini, "Distributed virtual environments: From client server to cloud and p2p architectures," in *Proceedings of the International Conference on High Performance Computing Simulation (HPCS)*, July 2012, pp. 8–17.

[15] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, 2017.

[16] J. White, B. Dougherty, and D. C. Schmidt, "Selecting highly optimal architectural feature sets with filtered cartesian flattening," *Journal of Systems and Software*, vol. 82, no. 8, pp. 1268–1284, 2009.

[17] D. Cooray, E. Kouroshfar, S. Malek, and R. Roshandel, "Proactive self-adaptation for improving the reliability of mission-critical, embedded, and mobile software," *IEEE Transactions on Software Engineering*, vol. 39, no. 12, pp. 1714–1735, 2013.

[18] R. Rouvoy, P. Barone, Y. Ding, F. Eliassen, S. Hallsteinsen, J. Lorenzo, A. Mamelli, and U. Scholz, "Music: Middleware support for self-adaptation in ubiquitous and service-oriented environments," in *Software Engineering for Self-Adaptive Systems*. Springer, 2009, pp. 164–182.

[19] K. Saller, M. Lochau, and I. Reimund, "Context-aware dspls: model-based runtime adaptation for resource-constrained systems," in *Proceedings of the 17th International Software Product Line Conference co-located workshops*. ACM, 2013, pp. 106–113.

[20] L. Rosa, L. Rodrigues, A. Lopes, M. Hiltunen, and R. Schlichting, "Self-management of adaptable component-based applications," *IEEE Transactions on Software Engineering*, vol. 39, no. 3, pp. 403–421, 2013.

[21] R. Mizouni, M. A. Matar, Z. Al Mahmoud, S. Alzahmi, and A. Salah, "A framework for context-aware self-adaptive mobile applications spl," *International Journal of Expert Systems with Applications*, vol. 41, no. 16, pp. 7549–7564, 2014.

[22] G. H. Alférez, V. Pelechano, R. Mazo, C. Salinesi, and D. Diaz, "Dynamic adaptation of service compositions with variability models," *Journal of Systems and Software*, vol. 91, pp. 24–47, 2014.

[23] S. K. Datta, C. Bonnet, and N. Nikaein, "Self-adaptive battery and context aware mobile application development," in *Proceedings of the International Wireless Communications and Mobile Computing Conference (IWCMC)*. IEEE, 2014, pp. 761–766.

[24] G. G. Pascual, M. Pinto, and L. Fuentes, "Self-adaptation of mobile systems driven by the common variability language," *Journal of Future Generation Computer Systems (FGCS)*, vol. 47, pp. 127–144, 2015.

[25] N. Ali and C. Solis, "Self-adaptation to mobile resources in service oriented architecture," in *Proceedings of the International Conference on Mobile Services*. IEEE, 2015, pp. 407–414.

[26] T. Zhao, W. Zhang, H. Zhao, and Z. Jin, "A reinforcement learning-based framework for the generation and evolution of adaptation rules," in *Proceedings of the International Conference on Autonomic Computing (ICAC)*. IEEE, 2017, pp. 103–112.

[27] M. G. Xavier, K. J. Matteussi, G. R. Frana, W. P. Pereira, and C. A. F. d. Rose, "Mobile application testing on clouds: Challenges, opportunities and architectural elements," in *Proceedings of the 25th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*, March 2017, pp. 181–185.

[28] R. De Lemos, H. Giese, H. A. Müller, M. Shaw, J. Andersson, M. Litoiu, B. Schmerl, G. Tamura, N. M. Villegas, T. Vogel *et al.*, "Software engineering for self-adaptive systems: A second research roadmap," in *Software Engineering for Self-Adaptive Systems II*. Springer, 2013, pp. 1–32.