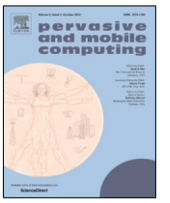
Pervasive and Mobile Computing 84 (2022) 101627

Contents lists available at ScienceDirect

Pervasive and Mobile Computing

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



Perses: A framework for the continuous evaluation of the QoS of distributed mobile applications

Sergio Laso [a,b,*], Javier Berrocal [a], Pablo Fernández [c], Antonio Ruiz-Cortés [c], Juan M. Murillo [a]

[a] Universidad de Extremadura, Cáceres, Spain
[b] Current Affiliation: Global Process and Product Improvement S.L., Cáceres, Spain
[c] I3US Institute, SCORE Lab. University of Sevilla, Sevilla, Spain

ARTICLE INFO

Article history:
Received 28 March 2022
Received in revised form 3 June 2022
Accepted 6 June 2022
Available online 18 June 2022

Keywords:
Distributed Computing
Mobile applications
Quality of Service
Evaluation
Virtual scenarios

ABSTRACT

The increasing capabilities of mobile devices have led to the emergence of new paradigms exploiting them. These paradigms foster the onload and distribution of functionalities on mobile devices, allowing the development of distributed mobile applications. This distribution reduces the latency and the data traffic overhead and improves privacy. As in any other mobile application, their success largely depends on the quality of service (QoS) they offer. Nevertheless, the evaluation of distributed mobile applications is particularly complex due to the number, heterogeneity, and interactions between the devices involved. Current techniques allow developers to assess the quality of a single device, but they are not designed for highly heterogeneous, distributed, and collaborative environments. This paper presents a framework called Perses, which allows the creation of virtual scenarios with multiple heterogeneous mobile devices to launch end-to-end tests to evaluate not only each device but also the interactions among them. The framework was evaluated against a real deployment, showing that the behavior and the quality attributes measured are similar to those of the real deployment, allowing developers to evaluate these applications before launching them. Finally, Perses was integrated into a DevOps methodology to automate its execution and further facilitate its adoption by software companies.

1. Introduction

We have witnessed a massive deployment of mobile applications since the arrival of the Apple App Store and Google Play [1]. This has generated a proliferation of companies dedicated to the development of mobile applications, leading to a great economic impact [2]. The success or failure of mobile applications largely depends on the quality of experience (QoE) [3] they offer, i.e., the satisfaction of a user with the provided functionality. The QoE is usually considered a broad term evaluating both the quality of service (QoS) [4] – quantitative measures of the performance of a service – and the user experience (UX) — efficiency of the interaction between the end-user and the service. Many of the companies behind mobile applications are startups so that the success of an application is an omen of the future of the company.

To ensure their success, mobile applications usually follow a pure client–server architecture. All computing demanding components (back end) are offloaded to cloud environments. Only the user interface and some basic components (front

* Corresponding author.
E-mail address: slasom@unex.es (S. Laso).

https://doi.org/10.1016/j.pmcj.2022.101627

end) are deployed on mobile devices. This architectural style allows developers to reduce the resource consumption on these devices; thus, allowing their deployment on almost any device with a minimum set of characteristics — limiting the problems caused by the great heterogeneity of devices in this market.

To evaluate the required quality, developers can assess each side (back end and front end) independently, but also end-to-end (E2E) testing are also performed. E2E tests assess the correct integration of the two sides and the correct functioning of the main functionalities, replicating the behavior of the users [5].

Nevertheless, in recent years, mobile devices have considerably increased their computing and storage capabilities [6], enabling the development of more computing-demanding applications in order to meet more stringent QoS requirements. New paradigms and architectural designs have emerged for developing them with distributed computing (in the following distributed mobile applications), such as Human Microservices [7], the Internet of People [8] or mist computing [9]. In addition, these paradigms allows one to address some challenges such as privacy awareness [10] and distribute the learning and intelligence among several nodes [11]. In these new paradigms, a functionality, or parts of it, may be distributed among different devices (mobile devices, IoT devices, fog or edge nodes, etc.). To consume this distributed functionality, usually some of the involved devices should interact to get all the required data and complete its workflow.

Therefore, these paradigms allow developers to relegate more functionality and on-load some computationally demanding components on mobile devices to further improve the QoS by reducing the latency caused by the network communication and data traffic overhead, increasing the control of the user over their data, and improving their privacy.

However, the QoS of distributed mobile applications is complex to evaluate. End-to-end evaluation in such architectures is not as trivial as in more traditional architectures. In a client–server architecture, it is only necessary to have the back end available and execute the end-to-end tests from a device. With these new paradigms, it is necessary to deploy a considerable set of heterogeneous devices close to real scenarios to properly evaluate the QoS — the latency and the execution time can vary greatly depending on the devices involved. This is especially relevant in today's ecosystem of mobile devices, due to the market fragmentation. This makes it even more necessary to know the expected quality in advance.

The application that does not meet the expected QoS is more likely to be rejected by users, with the financial and image impact that this may cause to a company. Currently, some commercial tools allow the evaluation of applications on different devices. AWS Device Farm [12] or Azure App Center Test [13] are platforms that provide a farm of real mobile devices. However, they do not allow the deployment of several mobile devices and the launch of E2E tests triggering different interactions among them, which is required to properly evaluate these distributed applications. Therefore, new techniques and tools are needed to evaluate the QoS of distributed mobile applications with E2E testing, enabling large-scale simultaneous evaluation of several highly interacting mobile devices and considering the heterogeneity of today's ecosystem.

In this paper, we present a framework called Perses. This framework allows the creation of virtual scenarios for the large-scale simultaneous deployment of virtualized mobile devices. Developers can simulate the deployment of distributed mobile applications in these heterogeneous scenarios. To evaluate the QoS correctly, the framework launches E2E tests that allow the whole system to be tested. In addition, this tool can be fully integrated into a software development process such as DevOps, automating the entire process of creating, deploying, and launching E2E tests, which reduces the effort invested by software companies to validate the QoS of their applications before deploying them into the production environment.

The main contributions of this works are as follows:

1. Perses allows to evaluate the QoS of distributed mobile applications through E2E tests in heterogeneous virtual scenarios with large-scale mobile devices.
2. Integration into common software development practices. Different methods have been developed to ease the integration of Perses into a DevOps methodology and in the Continuous Integration practices. This reduces the effort invested by software companies to validate the QoS of their applications before deploying them in the production environment.
3. The presented framework has been evaluated with a real application, comparing the real vs the virtual scenarios. In addition, different scalability tests have been performed, identifying that Perses is highly scalable and that the performance is maintained when the number of simulated devices increases.

The rest of the paper is structured as follows. Section 2 describes the motivations for this work. Section 3 explains the Perses framework. Section 4 validates the framework through a case study. Section 5 presents several related works. Finally, Section 6 details the conclusions.

2. Motivation

The growth of the mobile applications market has generated a multitude and variety of applications. Among the different companies that develop applications, there is a fierce fight for their applications to be the most downloaded and used to obtain greater benefits through different forms of revenue. There are different dimensions, such as advertising, originality, and virality for an application to be more or less successful. This success also lies in end-user satisfaction by offering a good QoE [14].

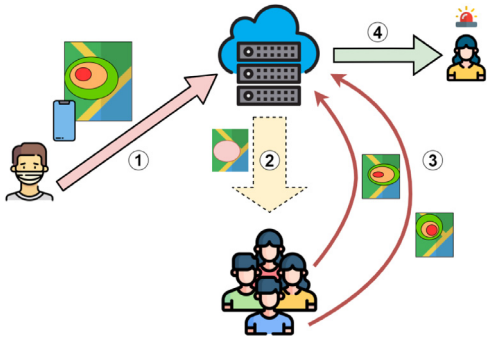


Fig. 1. COVID-Heatmaps app.

To achieve this QoE, it is necessary to apply software development methodologies and tools that allow developers to evaluate the system and to obtain rapid feedback. The QoE not only depends on the usability and accessibility that the interface consumes by end users but also depends on the QoS offered by the whole application and the infrastructure supporting it, especially in the case of distributed mobile applications in which computing is not primarily performed in a single location (cloud environment in traditional architectures) but is performed on a set of distributed devices.

During the last few years, we have witnessed applications (or specific releases) that either failed or had poor acceptance by users due to the QoS provided. For instance, Pokemon Go, a fairly successful mobile augmented reality game, may have failed in its early days. After an avalanche of downloads of the game in its presentation due to people's interest, the servers where the game's functionalities were hosted were saturated, causing the vast majority of users to be unable to play [15]. Another example is Auctionata — Online, which proposed live-streamed auctions of fine art and collectibles by broadcasting bids via mobile devices. The early attempts at broadcasting events failed to meet the expected QoS, limited by slow broadband speeds and delivery concerns [16]. In the latter case, the company was a start-up and had to close down.

These problems can be avoided by evaluating the QoS of the whole system before deployment. In client–server architectures, QoS is usually evaluated at the back-end, for example, by performing load tests simulating the connection of multiple clients with tools such as Apache JMeter[1] or Postman,[2] as this is where most of the computation and storage is located. Nevertheless, in distributed mobile applications, the computation and storage tasks are performed on different devices (smartphones, IoT devices, edge or cloud nodes, or any other device managing the system). Therefore, QoS evaluation must be performed jointly, involving all the different (or, at least, a good representation) devices that collaborate in the real deployment. This will allow developers to obtain results that can more accurately predict the expected behavior in a real production environment.

As a running example, let us present a distributed mobile application to detect close contacts of positive COVID-19 users, called COVID-Heatmaps. This application is composed, on the one hand, of a mobile application that stores the location of users on their devices and processes and generates heatmaps that are used to detect close contacts. On the other hand, a cloud node compares the heatmap generated by users with that of a COVID-19-positive user to detect whether they are close contacts. Fig. 1 shows an overview of how this application works. First (step 1), when a COVID-19-positive user is registered, the mobile application processes its heatmap with the stored traces and sends them to the cloud node. Second, the cloud node delimits the user's movement areas according to the heatmap and sends them to the other mobile devices (step 2). These devices check with their location history to determine whether they have been in the received areas. The devices that have been in these areas process and send their heatmap to the cloud node (step 3). The cloud node compares the heatmaps of the users with one of the COVID-19 positive, and the users who are considered close contacts are notified (step 4).

A potential solution for evaluating the QoS in these distributed applications is to use a real device farm that allows deploying the application on a large set of devices, and then, together with the cloud node, launch end-to-end tests, simulating the behavior of any user and the QoS obtained.

Several platforms offer physical mobile device farms. Among them are the AWS Device Farm and Azure App Center Test, which allow us to create highly customized test runs, indicating different types of devices, OS versions, etc. One of the disadvantages of these platforms is the number of devices that can be run simultaneously; they offer a very limited number and at a rather high cost. In addition, these platforms focus on launching UX tests, such as unit and adaptivity tests of graphical elements, navigation through different screens, compatibility with different OS versions, etc. (Appium,[3] Espresso,[4] etc.).

1 https://jmeter.apache.org.
2 https://www.postman.com.
3 http://appium.io.
4 https://developer.android.com/training/testing/espresso.

Furthermore, in distributed mobile applications, all the distributed components (mobile devices and cloud nodes in this case) must be fully available to execute E2E tests analyzing the QoS. For instance, to search for the close contacts of a COVID-19-positive user in our running example, the positive user identifies his or her movement areas in the previous few days to the cloud, the cloud sends these areas to the other users, these users validate whether they have been in these areas, and sends these evaluations to the cloud — which checks whether they are close contacts. This E2E test involves the evaluation of several components on different devices. Traditional testing platforms are not intended for testing scenarios with multiple interacting devices. They are mainly focused on running isolated tests on each device.

Therefore, new techniques are needed to evaluate the QoS of distributed mobile applications simulating a near-reality scenario. This will allow developers to launch E2E tests and measure the obtained QoS. These techniques will also be able to be integrated into a software development process, such as DevOps, so that they can be widely adopted by enterprises, saving the effort and cost required to perform these tests.

3. Perses: QoS evaluation in distributed applications

The estimation of QoS attributes in distributed architectures is particularly complex, and frameworks/tools are needed to help developers measure them to increase the success likelihood. Perses is a framework that allows developers to easily deploy a virtual scenario with multiple heterogeneous virtual mobile devices to evaluate the QoS of a distributed mobile application. These applications are characterized by distributed computing, in which some or all of the main computation that significantly affects QoS is performed on the end devices (executing activities such as data processing, running an AI model, etc.). In addition, these applications seek to enhance user privacy by storing data on end devices locally. Given an initial file with the configuration of the virtual scenario (infrastructure, network, devices, tests, etc.), Perses deploys it in a cloud instance.

Perses is fully scalable, allowing the evaluation of virtual scenarios formed by a large number of virtualized heterogeneous devices. The heterogeneity of virtual mobile devices is characterized by the possibility of deploying them with different hardware, operating system and configurations, being able to simulate close-to-real scenarios where there is a multitude of devices with different hardware and software characteristics.

Once deployed, Perses launches the tests, collects the logs from the devices, and analyzes the results. To support the evaluation of different dimensions of the QoE, Perses allows the execution of QoS tests and UX tests. First, it allows the configuration and definition of E2E tests evaluating different devices and the interactions among them, which are essential for evaluating QoS in distributed mobile applications. The UX tests are focused on evaluating the user interface with Espresso. Thus, Perses covers the most important dimensions evaluated by developers.

Finally, to save the time required for the manual execution of Perses, it is fully integrated into the DevOps methodology, automating the different steps during the evaluation process, and allowing software companies to easily integrate Perses into their continuous integration pipeline.

During the following subsections, first, the architecture of Perses is presented, detailing the characteristics of its modules; second, the configuration file is showed, where the characteristics of the virtual scenario, tests, etc. are defined, and finally, the integration of Perses in the DevOps methodology and its execution flow are described.

3.1. Architecture

Fig. 2 shows the architecture of Perses, formed by different modules focused on specific functionalities that complement each other. The most important modules are described below.

Setup: Perses requires a set of credentials and a configuration file as input. This file contains the characteristics of the virtual scenario to be deployed, the tests to be launched and the desired QoS attributes to be evaluated during the test execution. This module checks both the credentials to connect to the cloud infrastructure provider and the configuration file with the different parameters defined in the virtual scenario.

Deployment: The deployment module creates and deploys the entire virtual scenario taking the configuration file as input. To create and deploy the scenario, an abstraction layer is defined that encapsulates Terraform [17] — a framework with a high-level language that is used to define the deployment infrastructure of an application for cloud providers. Terraform has been extended so that in addition to deploying the cloud infrastructure, it also orchestrates the virtual scenario by installing the necessary resources, creating virtual mobile devices, and deploying distributed mobile applications on those devices. To host and deploy the virtual scenario, Amazon Web Services (AWS) is used as the cloud infrastructure provider. For the creation and deployment of virtual mobile devices, Docker[5] is used, where mobile devices are deployed in containers [18]. The management of these devices and the installation and deployment of the distributed application is performed through an Android Debug Bridge (ADB).[6]

Due to the network infrastructure provided by AWS, data is transmitted directly over the wired network. This is a significant difference to real mobile-cloud communication via WIFI or LTE. Kathara [19], a framework that enables network

5 https://www.docker.com.

6 https://developer.android.com/studio/command-line/adb.

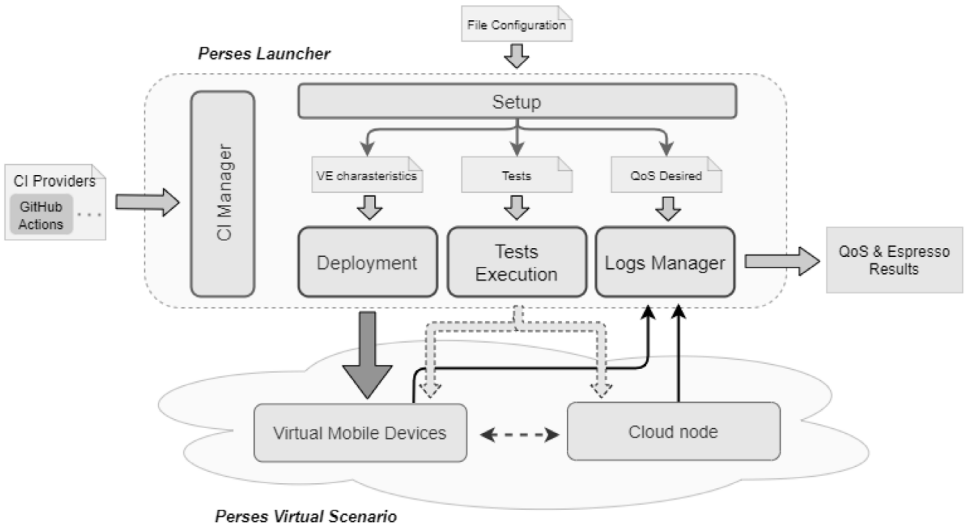


Fig. 2. General diagram of Perses.

emulation in Docker containers, has therefore been integrated. This allows us to emulate the network infrastructure of the virtual devices with the cloud environment to replicate real mobile-cloud communications.

Tests Execution: this module executes the tests defined in the configuration file. Perses allows two different types of tests to be run. *E2E tests*, these tests evaluate the QoS attributes of the application by executing the core functionalities that trigger the interactions among different devices. For this, Perses integrates APIPecker [20] — a simple API performance tester where different attributes (concurrent users, iterations and delay) can be defined to customize the tests, stress the devices, and obtain more information about the QoS. In addition, *user interface tests*, Perses allows developers to run Espresso tests to evaluate the UX and specific traditional functionalities.

Logs Manager: after launching the tests, this module collects the results obtained by aggregating the system logs of each of the virtual devices. After this, it analyzes the results and determines whether the desired QoS defined in the configuration file is achieved.

CI Manager: this module integrates with DevOps. This module autonomously manages the execution of the different Perses actions and modules. This makes it possible to automate the entire process of creating and deploying virtual scenarios and all the management related to the launch and analysis of tests. This automation is carried out through a workflow defined with GitHub Actions [21].

3.2. Defining virtual scenarios

Perses needs a configuration file (*.yaml* extension)[7] where the characteristics of the virtual scenario, the tests, and the desired QoS are defined. Fig. 3 shows a conceptual model with all the parameters in the configuration file. The explanation of each of the parameters and the user guide is detailed on Github.[8]

3.3. Integrating Perses in a DevOps methodology

One of the features of Perses is the full integration into the DevOps methodology. For this purpose, it is currently integrated with GitHub Actions, which makes it easy to automate software workflows applying CI/CD. To perform this integration, it is necessary to define a file with the different steps that the workflow must follow. This workflow has been developed to run any application, i.e., it is not necessary to define a workflow file for each application. Listing 1 shows an extract of the workflow, and the complete file is available on GitHub.[9]

Due to the integration of Perses in the DevOps methodology, the effort required by applying the defined framework is reduced. In addition, by automating the entire execution flow, the effort for developers is minimal, and repeatability is encouraged.

7 https://github.com/slasom/COVID-Heatmaps/.
8 https://github.com/perses-org/perses.
9 https://github.com/perses-org/gha/blob/master/workflow/perses-workflow.yml.

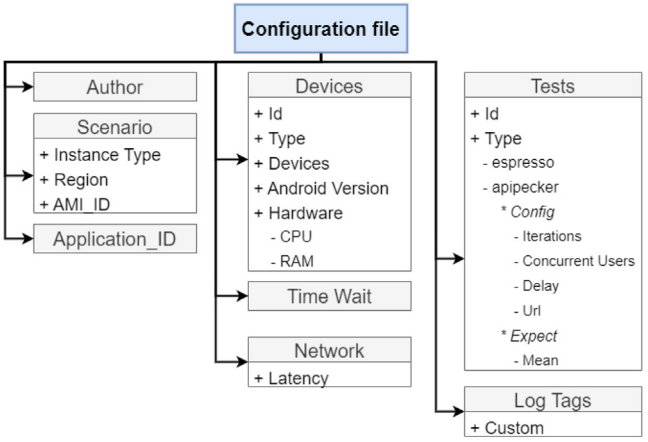


Fig. 3. Conceptual Model. Configuration File.

```
- name: "Build Android project"
uses: vgaidarji/android-github-actions-build@v1.0.1
with:
args: "./gradlew assembleDebug assembleAndroidTest"...
- name: "Perses Setup"
run: |
cd.perses_runner
node index -a setup -g../.perses-full.yml -c../.perses-credentials.yml
...
```

Listing 1: Perses Workflow

4. Experimental evaluation

Perses is a framework that allows the deployment of virtual scenarios to assess the QoS of distributed computing applications. However, although it can configure scenarios with heterogeneous devices, these devices are not real; they have virtualized images with limitations to consume a similar amount of resources as real devices. In this section, we present the evaluation of a case study in a real scenario with physical devices and with Perses to study the feasibility of the proposal and the accuracy of the results obtained.

In this section, first, the characteristics of both scenarios are presented (devices used, characteristics of the cloud node, characteristics of the Perses's virtual scenario, etc.). Second, the tests to be launched and the QoS parameters to be measured are explained. Finally, the results obtained, the comparison of both scenarios in terms of executing time, transfer time, etc., and their operational cost are analyzed and discussed.

The case study focuses on the evaluation of the viability of the proposal. For this purpose, the distributed computing application mentioned in Section 2 will be used in which several QoS parameters will be evaluated.

4.1. Experiment set-up

The experiment was carried out with seven physical mobile devices for the real scenario and seven virtual mobile devices for the virtual scenario. To make the test fair, each of the seven devices in each scenario was loaded with a specific set of locations so that the same volume of data was processed and transferred in both scenarios in the different tests.

The real scenario is composed of three Xiaomi Mi 9, two Mi 9T, OnePlus 6T, and Huawei Mate 20, which have similar hardware characteristics. All have 6 GB of RAM, and the processors are very similar (Qualcomm 855, 845, and 730 and Kirin 980). All of them have Android 9 as their OS.

The virtual scenario has been defined in the configuration file with all the features required to use Perses. More information about the configuration file can be found in the link.[10] To host and deploy the virtual scenario, we use a C5.metal EC2 instance of AWS. The set of virtual devices consists of seven devices, as in the real scenario. The hardware

[10] https://github.com/slasom/Covid-Heatmaps/.

is limited to similar characteristics to those of real devices. Otherwise, as they are located in a fairly powerful instance, they clearly outperform physical devices. Each of them is composed of three CPUs, 6 GB of RAM, and Android 9 as the OS.

The cloud node is the same for both scenarios. It has been deployed on a T2.large EC2 of AWS. For communication and data transfer with mobile devices, the MQTT [22] communication protocol was used.

4.1.1. Scenario evaluation set-up

To evaluate the QoS parameters, a set of tests with different configurations is launched based on the main functionality of the application, i.e., to search for close contacts after a COVID-19-positive user registration.

The different configurations in the tests are linked to the definition of different 'positive users' with the set of synthetic locations that has been created to obtain more varied results. When a positive user is registered, one, three, five and seven devices real/virtual respond to the request with a heatmap of 100, 1,000, 5,000 and 10,000 location points. These increments are defined to evaluate Perses in the face of increased mobile device and data/computer complexity. Each of the tests was repeated four times to obtain consistent results. To check the dispersion of the mean of the test repetitions, the coefficient of variation was calculated, obtaining a maximum value of 0.078 among all the results.

Finally, three QoS parameters are measured that allow us to compare Perses with a real scenario. **Mobile-Side execution time** is the time it takes for mobile devices to process their heatmaps. **Transfer time** measures the time it takes to send the data from the cloud side to the mobile side and vice versa. **Aggregation time** captures the time the cloud spends processing and comparing the heatmaps to detect possible close contacts.

4.1.2. Results of the experiments

This last subsection presents the results obtained after the evaluation of the COVID-Heatmaps app in both scenarios.

Mobile-Side Execution time: Fig. 4(a) shows the computing time on real and virtual mobile devices for the different numbers of devices involved and location points exchanged. It can be seen that they follow the same trend in both scenarios. There is a small gap between both scenarios; this is due to the existing hardware inequality, and the processors of real mobile devices are less powerful due to their size, architecture, etc.

To better analyze the differences between the two scenarios and the existing gap, Figs. 4(b) and 4(c) show the difference between the plane for the real scenario and the plane for the virtualized scenario. Fig. 4(b) shows a 2D graph with different lines for the different data consumption as the number of devices increases. Fig. 4(c) shows different lines for the different number of mobile devices as the quantity of data to be processed increases.

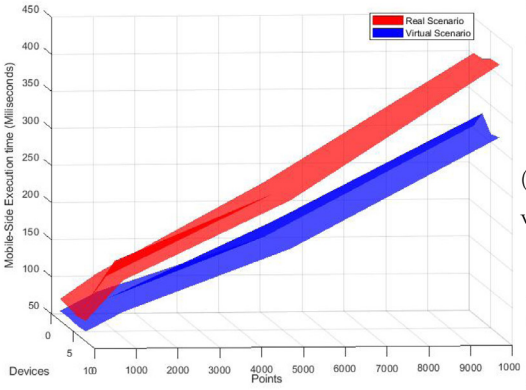
The graph 4(b) shows that for 100 location points, the difference is approximately 28 ms on average for the whole set of devices, as each device computes its heat map concurrently and therefore the computation time is very similar. The same is true for the other sets of points sent; for 1,000 location points, the difference is slightly higher, 46 ms. For 5,000 location points, the difference increases to 80 ms. Finally, for 10,000 location points sent, the difference is 134 ms on average. This increase can be better seen in Fig. 4(c), showing that for every configuration (number of devices), the execution time increases with the same trend. However, the increase is the same with respect to the total time, and the dashed pink line shows the percentage of the difference in the execution times between both scenarios. As seen in Figs. 4(b) and 4(c), percentage-wise, the difference is constant at approximately 34%. This gap cannot be reduced due to the minimum hardware requirements for virtual devices to be deployed. If they are further constrained to match real devices, they do not have sufficient capacity to deploy the docker container. However, as can be seen this gap constant, therefore extrapolation can be applied in order to obtain results closer to the real devices.

Finally, Fig. 5(a) shows the results obtained with Perses after a test with different sets of virtual mobile devices to evaluate scalability, deploying up to 50 simultaneous devices. An increase in execution time can be observed as the number of points increases. However, increasing the number of devices does not affect the performance because the execution is performed in parallel on each device.

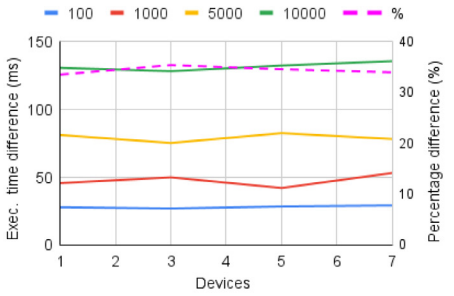
Transfer time: Fig. 6(a) shows the results obtained on the data transfer time. As with Mobile-Side Execution time, they follow the same trend with the constant difference between the two scenarios. In this case, the difference is minimal, the integration of Kathara with Perses allows emulating the communication of real mobile devices. In the real scenario, the mobile devices receive the request and transfer the data via WiFi technology plus the existing distance to the location of the cloud node deployed on the AWS servers. In the virtual scenario, the data is transmitted directly over the wired network. So without network emulation, the transfer time of virtual mobile devices would not come close to that of the real scenario.

As before, Figs. 6(b) and 6(c) show the difference between the real and virtualized scenario planes. Fig. 6(b) shows different lines for the different numbers of points. Fig. 6(c) shows different lines for each set of responding devices. Both figures show a difference between ± 25 ms. This means that in certain cases, the virtual scenario transfers data faster than the real scenario and vice versa. The percentage of difference between both scenarios is between 3%–5%, a minimal variation. Therefore, Perses allows developer to simulate the network infrastructure and execute the tests over it.

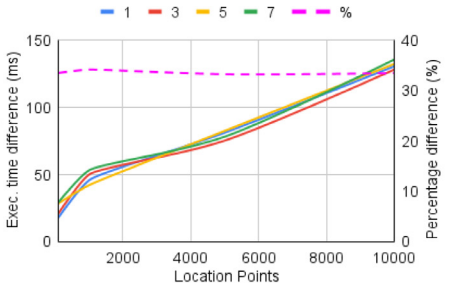
Finally, Fig. 5(b) shows the results obtained with the scalability test. We can observe a growth of the transfer time both when increasing the number of points and the number of virtual devices. The growth due to the increase in the amount of virtual devices is caused by the increase in the number of virtual devices and the increase in the coordination time of all the responses received by the cloud.



(a) Execution time

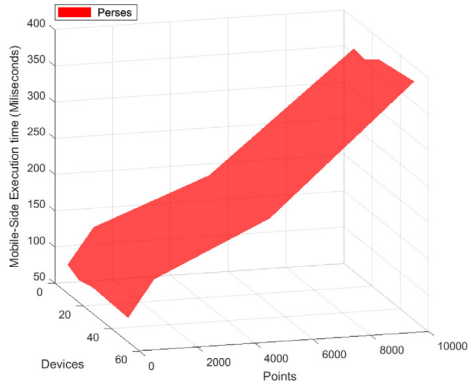


(b) Differences depending on the num of devices

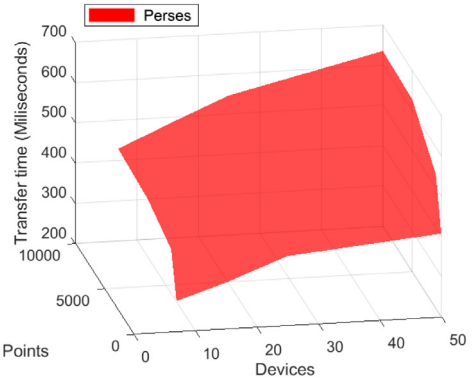


(c) Differences depending on the num of points

Fig. 4. Mobile-Side Execution time results.



(a) Mobile-Side Execution time large-scale



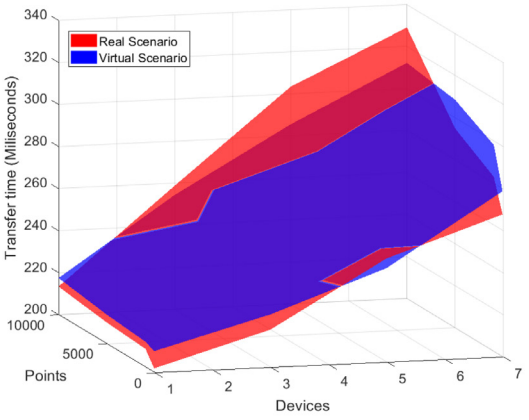
(b) Transfer time large-scale

Fig. 5. Large-scale test results.

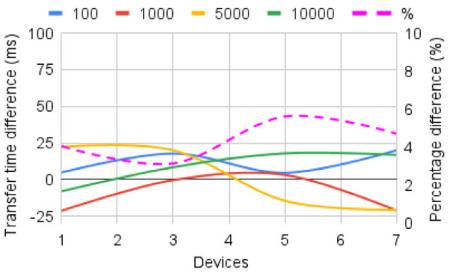
Aggregation time: Fig. 7(a) shows the aggregation time in the cloud node of the heatmaps for the different numbers of devices involved and location points sent. In this case, it can be seen that the planes obtained from both scenarios are practically the same. The cloud node is the same for both scenarios, the number of devices involved and the volume of data are the same. Figs. 7(b) and 7(c) show the differences in aggregation time between the two scenarios as a function of the number of devices and the amount of information shared, respectively. A constant trend is observed horizontally, and there is hardly any difference (between 0.01 and −0.01), as expected.

Operational costs: This last section shows the differences concerning the operational costs required for testing the application in both scenarios.

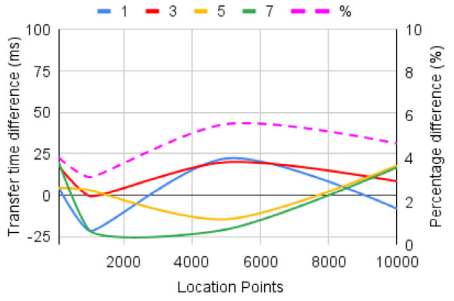
The real scenario had a cost composed of the physical smartphones and the AWS infrastructure. The price of the smartphones (launch price) gives a total amount of $2,804. With respect to the AWS infrastructure, a T2.large instance was used in the Ireland region at a cost of $0.1008/hours. The duration of the tests in this scenario lasted 5 min or 0.083 h, resulting in $0.009. Therefore, the real scenario had a cost of $2,804.01



(a) Transfer time

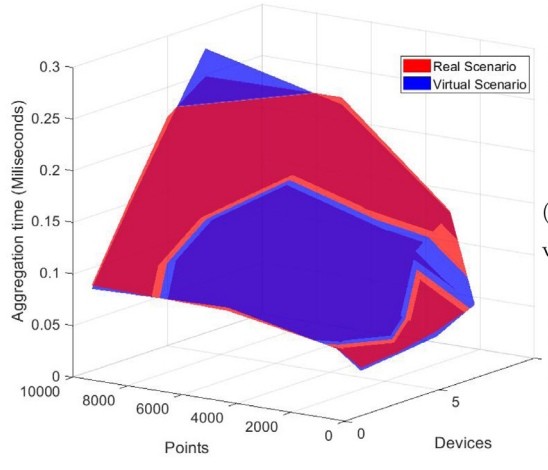


(b) Differences depending on the num of devices

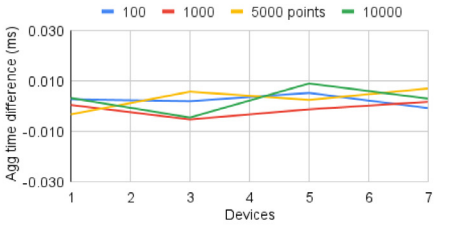


(c) Differences depending on the num of points

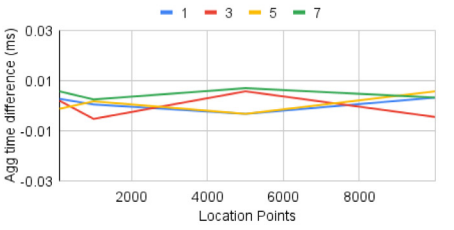
Fig. 6. Transfer time results.



(a) Aggregation time



(b) Differences depending on the num of devices



(c) Differences depending on the num of points

Fig. 7. Aggregation time results.

The virtual scenario had a cost composed of the AWS infrastructure used to host and use the scenario with the virtual devices, the cloud node, and the MQTT communication protocol. The virtual scenario was hosted on a C5.metal instance in the Ireland region for $4.608/hours. As before, the cloud node and the MQTT protocol were hosted on the same instance. These tests took 20 min or 0.33 h. Therefore, a cost of $1.536 for the virtual scenario and $0.033 for the cloud node and the MQTT protocol was obtained. Therefore, the virtual scenario had a cost of $1.569.

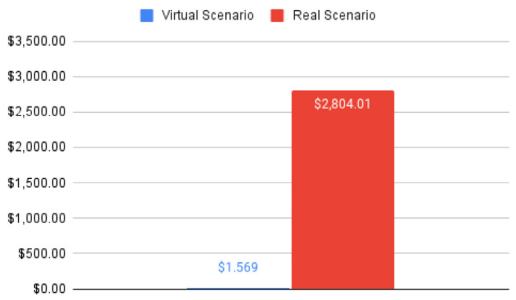


Fig. 8. Operational costs.

Fig. 8 shows the total costs for each of the scenarios. There is a large difference between testing in a real scenario and testing with the Perses framework. For the cost of deploying the real scenario, we performed Perses's evaluation on applications with similar characteristics 1,787 times.

5. Related works

There are different frameworks and commercial platforms for testing mobile applications. In addition to the aforementioned AWS Device Farm and Azure App Center Test, there are other tests, such as the Firebase Test Lab [23] and Perfecto [24]. All of them offer different functionalities and features for testing mobile applications with real and virtual devices (selection of different models and hardware characteristics, video reports of results, etc.). They are also easily integrated into different development tools and in CI/CD pipelines of the software development cycle of companies. However, they are focused on UI testing, i.e., they do not evaluate QoS parameters such as response time and latencies, as they are designed for mobile applications with client–server architectures.

At the research level, there are some proposals detailing frameworks, tools, and techniques for evaluating distributed applications focused on the IoT. In [25], the authors proposed a framework for developing and evaluating component-based distributed systems for heterogeneous scenarios, considering mobile and fixed networks. It is an interesting proposal and similar to our work focusing on distributed applications. However, it is not clear what the workflow of the platform was and what the development and evaluation process was. Moreover, it is not possible to evaluate standard applications, only those developed on the platform. The focus was specifically on the evaluation of applications developed with the proposed framework. Therefore, approaches are needed that can also evaluate any applications that can be deployed on the target devices.

In [26], the authors presented a framework for automated IoT application testing. The framework allows the automated execution of user-defined experiments and can generate virtual testbeds with adjustable network properties. To do so, they virtualize devices acting as fog and edge devices using the QEMU emulator. The authors developed a prototype of the framework and, in the experiment, performed experiments with physical and emulated Raspberry Pi 3. The framework is promising; however, it does not allow the creation of heterogeneous testbeds. All emulated devices run under the same operating system and the same defined features. Finally, they do not integrate the defined framework in a development cycle for companies.

In [27], the authors presented a simulator that enables the design and analysis of large-scale IoT systems for smart city applications consisting of mobile devices. It supports the simulation of devices, gateways, and applications. It also supports environment modeling (including movement, interference, etc.). They explain and simulate an IoT system that is deployed in Leuven. Nevertheless, it is intended for specific case studies of smart city applications.

In [28], the authors proposed a system for testing the usability and performance of Android mobile applications with virtual reality. This system simulates different network and mobility behaviors to evaluate applications in close-to-real environments with different network configurations. However, they focused on manually testing the application using a single device, which makes it impossible to evaluate distributed mobile applications. Furthermore, the tests performed focused on the user experience, and no QoS-related parameters were collected.

6. Conclusion

The market for mobile applications has grown at a frenetic pace. Moreover, the capacities of mobile devices have increased considerably. With this, new paradigms and distributed architectural designs have emerged for developing mobile applications exploiting these capabilities to further improve the QoS. However, new tools are needed to easily assess these distributed and highly heterogeneous environments and to ensure application quality.

In this paper, we presented a framework called Perses, which allows developers to create virtual scenarios and can deploy a large number of virtual mobile devices to correctly evaluate the QoS. This framework was fully integrated into a DevOps software development flow, which allows automating the tasks of creating and deploying the virtual scenario,

launching E2E tests, collecting results, etc., which reduces the effort required to perform these tests. This is a fundamental aspect so that it can be better embraced by software development companies. Finally, the framework was evaluated with a case study, and the results obtained were comparable to those obtained from a real scenario.

In future work, we are working on allowing the emulation of new devices (IoT devices). This will increase the heterogeneity of the scenarios. We are also working on providing different network topologies with Kathara (distribution of devices connected to different nodes simulating the connection to different WIFI or LTE connections). Finally, we are also working on automatically generating a cost analysis to allow developers to visualize the costs that will be generated before the deployment of the virtual scenario, being able to adjust it to the budget they have defined.

Declaration of competing interest

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

Acknowledgments

This work has been partially funded by the DIN2020-011586 grant, funded by MCIN/AEI/10.13039/501100011033 and by the European Union "NextGenerationEU/PRTR", by projects RTI2018-094591-B-I00 (MCI/AEI/FEDER,UE) and RTI2018-101204-B-C21, the 4IE+ Project (0499-4IE-PLUS-4-E) funded by Interreg V-A España-Portugal (POCTEP) 2014–2020 program, by the RCIS network (TIN2016-81978-REDT), by the Department of Economy, Science and Digital Agenda of the Government of Extremadura (GR21133, IB18030), by the Government of Andalusian (US-1264651) and by the European Regional Development Fund.

References

[1] Statista Research Department, Number of apps available in leading app stores 2020, 2021, URL https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/. (Accessed April 13 2021).
[2] J. Parker, 10 Years of growth of mobile app market, 2018, https://www.knowband.com/blog/es/mobile-app-es/10-years-of-growth-of-mobile-app-market/. (Accessed 13 February 2021).
[3] K. De Moor, I. Ketyko, W. Joseph, T. Deryckere, L. De Marez, L. Martens, G. Verleye, Proposed framework for evaluating quality of experience in a mobile, testbed-oriented living lab setting, Mob. Netw. Appl. 15 (3) (2010) 378–391.
[4] H.J. Kim, D.H. Lee, J.M. Lee, K.H. Lee, W. Lyu, S.G. Choi, The QoE evaluation method through the qos-qoe correlation model, in: 2008 Fourth International Conference on Networked Computing and Advanced Information Management, vol. 2, 2008, pp. 719–725, http://dx.doi.org/10.1109/NCM.2008.202.
[5] B. Lima, Automated scenario-based integration testing of distributed systems, in: Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2018, pp. 956–958.
[6] J. Berrocal, J. Garcia-Alonso, C. Vicente-Chicote, J. Hernández, T. Mikkonen, C. Canal, J.M. Murillo, Early analysis of resource consumption patterns in mobile applications, Pervasive Mob. Comput. (ISSN: 1574-1192) 35 (2017) 32–50, http://dx.doi.org/10.1016/j.pmcj.2016.06.011, URL http://www.sciencedirect.com/science/article/pii/S1574119216300797.
[7] S. Laso, J. Berrocal, J. García-Alonso, C. Canal, J. Manuel Murillo, Human microservices: A framework for turning humans into service providers, Softw. - Pract. Exp. (2021).
[8] J. Miranda, N. Mäkitalo, J. Garcia-Alonso, J. Berrocal, T. Mikkonen, C. Canal, J.M. Murillo, From the internet of things to the internet of people, IEEE Internet Comput. 19 (2) (2015) 40–47.
[9] A. Yousefpour, C. Fung, T. Nguyen, K. Kadiyala, F. Jalali, A. Niakanlahiji, J. Kong, J.P. Jue, All one needs to know about fog computing and related edge computing paradigms: A complete survey, J. Syst. Archit. (2019).
[10] Y. Qian, L. Hu, J. Chen, X. Guan, M.M. Hassan, A. Alelaiwi, Privacy-aware service placement for mobile edge computing via federated learning, Inform. Sci. 505 (2019) 562–570.
[11] K. Lin, C. Li, Y. Li, C. Savaglio, G. Fortino, Distributed learning for vehicle routing decision in software defined internet of vehicles, IEEE Trans. Intell. Transp. Syst. 22 (6) (2020) 3730–3741.
[12] Amazon Web Service, AWS device farm, 2021, https://aws.amazon.com/device-farm/. (Accessed 23 March 2022).
[13] M. Azure, App center test, 2021, https://docs.microsoft.com/en-us/appcenter/test-cloud/. (Accessed 28 May 2021).
[14] W.N. Picoto, R. Duarte, I. Pinto, Uncovering top-ranking factors for mobile apps through a multimethod approach, J. Bus. Res. 101 (2019) 668–674.
[15] A.P. Gamerant, Pokemon go worldwide launch halted to fix server problems, 2016, https://gamerant.com/pokemon-go-server-launch-problems/. (Accessed 28 October 2021).
[16] S. Insights, Top 7 biggest flops in the mobile app industry, 2019, https://www.smartinsights.com/mobile-marketing/top-7-biggest-flops-mobile-app-industry/. (Accessed 28 January 2021).
[17] HashiCorp, Terraform, 2021, https://www.terraform.io/. (Accessed 22 April 2021).
[18] Budtmo, Docker android, 2021, https://github.com/budtmo/docker-android. (Accessed 23 March 2022).
[19] M. Scazzariello, L. Ariemma, T. Caiazzi, Kathará: A lightweight network emulation system, in: NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium, IEEE, 2020, pp. 1–2.
[20] P. Fernandez, API pecker, 2020, https://www.npmjs.com/package/apipecker. (Accessed 10 March 2022).
[21] Github, GitHub actions, 2021, https://github.com/features/actions. (Accessed 27 July 2021).
[22] MQTT, MQTT, 2021, URL http://mqtt.org/. (Accessed 22 March 2021).
[23] Firebase, Firebase test lab, 2021, https://firebase.google.com/products/test-lab. (Accessed 13 March 2022).
[24] Perforce Software, Perfecto, 2021, https://www.perfecto.io/. (Accessed 23 March 2022).
[25] B. Richerzhagen, D. Stingl, J. Ruckert, R. Steinmetz, Simonstrator: Simulation and prototyping platform for distributed mobile applications, in: The 8th EAI International Conference on Simulation Tools and Techniques (ACM SIMUTOOLS 2015), IMDEA Networks Institute Publications Repository, 2015.

[26] I. Behnke, L. Thamsen, O. Kao, Héctor: A framework for testing IoT applications across heterogeneous edge and cloud testbeds, in: Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing Companion, 2019, pp. 15–20.
[27] M. Provoost, D. Weyns, DingNet: A simulator for large-scale IoT systems with mobile devices, in: EWSN, 2019, pp. 267–269.
[28] T. Amano, S. Kajita, H. Yamaguchi, T. Higashino, M. Takai, Smartphone applications testbed using virtual reality, in: Proceedings of the 15th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services, 2018, pp. 422–431.