2017 25th Euromicro International Conference on Parallel, Distributed and Network-Based Processing

Mobile application testing on Clouds: challenges, opportunities and architectural elements

Miguel G. Xavier, Kassiano J. Matteussi, Gabriel R. França, Wagner P. Pereira, and Cesar A. F. De Rose
Pontifical Catholic University of Rio Grande do Sul (PUCRS)
Faculty of Informatics, Porto Alegre, Brazil
miguel.xavier@acad.pucrs.br

***Abstract*—With mobility increasing more each year, mobile devices and operating system (OS) fragmentation are increasing at an even faster pace. A multitude of screen sizes, network connection types, and OS versions have emerged in the market and led mobile developers to rethink testing practices to ensure quality and a good experience for increasingly demanding users who crave highly reliable and stable applications. Such best practices come at the high cost for testing infrastructure and maintenance, making most development teams bypass test cycles and deliver applications before they are thoroughly validated. And when teams pursue automated test cycles on clouds, expenses due to high-cost services are not always worth the investment in application development phases. As a result, test cycles which are essential to validate application reliability and stability, such as regression, functional, and leakage tests are left out, primarily affecting user experience. This paper shows the challenges intrinsic to mobile application testing in consonance with the opportunities provided by clouds. We explored the state of the art to synthesize current cloud-based mobile application testing architectures to convey the need for a new concept and platform to minimize maintenance and make testing infrastructures more cost-effective. Hence, we proposed an alternative architecture using emulated devices for testing automation, which aims for massive test cycles at a lower cost.**

***Keywords*—Mobile application, application testing, cloud computing**

I. Introduction

The world has witnessed an explosive growth of mobile devices and applications driven by increased hardware capabilities and network communications globally. This growing number of devices has led developers to be concerned about applications stability. Since device manufactures have designed a variety of hardware architectures that differ in screen sizes, input methods (touchscreen, physical keyboard), sensors, etc., an application that is designed to run on one particular mobile device does not necessarily work well in another. In addition, each OS has its own characteristics and limitations, so that testing an application across multiple different OS versions is indispensable to ensure stability, reliability, and interoperability. Otherwise, applications tend to fail, primarily affecting user experience.

The simplest and most common way to test a mobile application involves running it on a number of devices and OSs, and analyzing its behavior during run-time. This task is toilsome given the multitude of hardware architectures and OS fragmentation. The time-consuming nature of manual testing and the number of OS versions make this solution unfeasible and no longer cost-effective during development stages. A set of testing tools has been developed to overcome this problem by providing emulated devices on which applications can be tested, stressed, and/or evaluated. Furthermore, some projects have focused on functional tests, providing frameworks for testing automation. Mobile developers now have a lot of opportunities, merging powerful frameworks for functional tests with emulated platforms for stability analysis at a low infrastructure cost.

On the other hand, cloud computing has evolved at a rapid pace and has become an alternative platform for systems that require scalability and high performance with unlimited hardware resources. Such platforms have paved the way for developers to migrate software development stages from local to decentralized environments where distributed teams develop, deploy, and manage applications without the complexity of building and maintaining the infrastructure. This cloud computing delivery model is called Platform as a Service (PaaS) [1], and has been implemented by Internet players such as Amazon, [2], IBM [3], Google [4], and Heroku [5].

Offering testing frameworks with mobile-emulated devices in PaaS platforms as a new type of service will likely be a trend in the near future. Hence, this paper introduces architectural elements necessary to execute not just tasks focused on developing or deploying on clouds, but an ecosystem to test, analyze, and diagnose potential issues proactively while evaluating mobile applications on distinct hardware configurations and under different conditions. Arguably, cloud platforms can orchestrate many testing life cycles from different users to provide a dynamic, resilient, and cost-effective testing infrastructure. As such, allow developers to submit their applications, hardware requirements, and gather valuable information to help them during the development phase.

Most research works have explored clouds as testing platforms to orchestrate tests using actual devices. These solutions look very attractive at first, but incur high infrastructure costs and demands even more costs for maintaining. TestFairy [6], for instance, depends on the collaboration of users to introduce their own devices into a shared testing network to be used as targets where applications can be tested, stressed and finally evaluated. Further, Xamarin [7] is a complete testing platform which uses dozens or even thousands of real devices in a data center to support developer's requests as they arrive. Is evident that real devices produce more realistic results, however, for high performance projects that execute many test cycles in short time period, these platforms become very costly and may be not preferable over simpler approaches as those based on emulation. These scenarios have motivated our research and have raised many fundamental issues that we will discuss and show later how they will be treated in our architecture.

2377-5750/17 $31.00 © 2017 IEEE
DOI 10.1109/PDP.2017.96

II. Mobile Application Testing: Challenges and Opportunities on Clouds

An abundance of mobile devices and applications have come into the market over the past few years. The set of display resolutions, aspect ratios, processing capacity, and amount of memory are all mandatory requirements for the design phase of an application. The increasing number of requirements have forced developers rethink testing practices to build high-quality applications at low infrastructure costs. Testing mobile applications is not a straightforward task and has challenged developers due to the multitude of devices, OS fragmentation, and the sheer number of test interfaces. Development and Quality Assurance (QA) teams can not guarantee that an application that works well on one device will work properly on another because the screen resolution, CPU, memory, and hardware differ in uncountable ways. Hence, many developers have focused on high-end technologies, overlooking cheaper and less powerful devices.

OS fragmentation poses a challenge for mobile application testing. Engineering teams face many issues to ensure a quality and consistent user experience given the variety of OS versions. For example, when an Android application is developed, it is necessary to deploy and test it across all Android's versions to ensure portability. When an application supports multi-platforms, the number of versions grows even further, making it very time-consuming and expensive for developers. Also, the devices themselves can be expensive to purchase.

Cloud computing platforms have been utilized to capitalize on test automation and make the testing process less time-consuming and more cost-effective. Distributed teams can perform test cycles on the same physical device shared through the cloud, instead of needing to buy the same hardware for all distributed offices. Cloud platforms have to face all of the challenges of testing mobile applications at a high scale using device clusters. We have observed purpose-build platforms, but related studies do not provide accurate nomenclature or explanations for the functionality of these solutions. Hence, we intend to classify these solutions according to the architectures we have found, which employee real, emulated, and collaborative device clusters.

A. Cloud-based Real Device Clusters

This platform presents a set of physical mobile devices linked on top of a datacenter as a device cluster, allowing for the execution of multiple tests over a pool of real devices, as depicted in Figure 1(a). Real device clusters are a simple way to test any mobile application with distinct requirements and configurations. Developers can upload test scripts without having to worry about the installation of testing tools or acquiring devices. Thus, the bugs can be caught through performance statistics, logs, videos, and a set of screenshots in runtime, and fixed before they are discovered by the user. Xamarin [7], AWS Device Farm [8], PCloud [9], Sauce Labs [10] are examples of this type of cloud platform. In contrast, infrastructures composed of real devices may limit the number of tests per user. Since there is a limited number of available devices, the job's makespan is steadily affected by the cloud throughput and request arrival times, harming the test cycles.

B. Cloud-based Emulated Device Clusters

Similar to device clusters, this platform allows mobile applications to be tested in a shared environment on the cloud using emulated devices instead (Figure 1(b)). Devices are created on-demand in an emulated form, enabling velocity and dynamic delivery for teams that need automated tests at a low cost. Emulators are wrapped into VMs, which are kept alive only during the test cycle. When the tests are completed, the VMs are destroyed and queued requests are scheduled. The tests and results are projected in the same way as device clusters. A comprehensive variety of devices and configurations can be chosen. However, some compatibility issues may occur when using an emulated cluster. For example, when performing different versions of an OS, such as Android, IOS, Windows Phone, and so on. In addition, it is necessary to provide resources (CPU, storage, memory, network) elastically to avoid potential performance problems during mobile application testing. As stated, emulation-based testing produces less realistic results, but makes up for this with low maintenance and infrastructure costs.

C. Cloud-based Real Device Collaborative Clusters

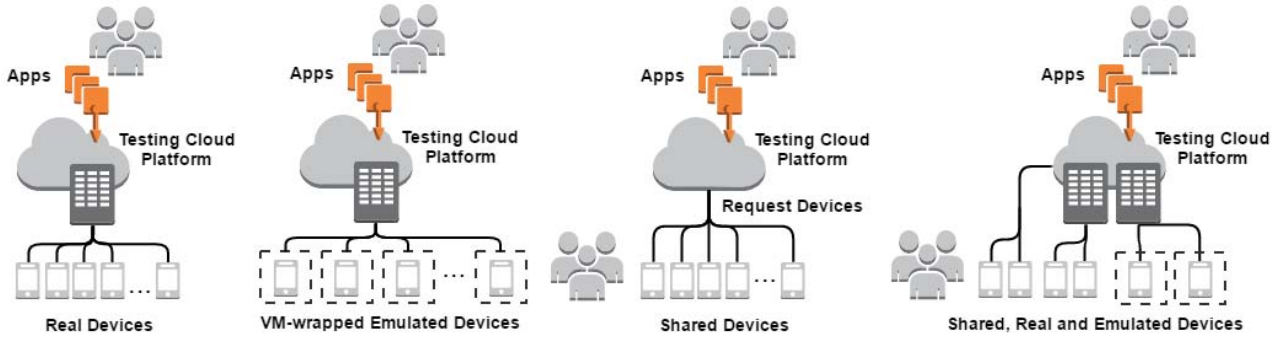
In collaborative device clusters, developers create groups of devices distributed geographically, allowing real mobile devices to be shared through the cloud (Figure 1(c)). This saves infrastructure costs because the device is bought just once and shared between distributed developers. The manual tests for their applications are made on top of real devices picked by users between available devices in the cloud (as a request) and accessed by a browser without any software installation. PCloud and AWS Device Farm are examples of this platform. This cluster does not have the same devices available all the time and also may present some instabilities due to hardware failures, power outages, malicious applications, connection problems (caused by the user or network issues), among other things. Moreover, the number of available devices is driven by the number of users sharing the network. The greater the number of users, the greater the number of devices on the clusters.

D. Cloud-based Device Heterogeneous Clusters

Device heterogeneous clusters link the three types of platforms previously presented, as a solution for large-scale testing, where developers may choose to run their tests on real, emulated or request access to collaborative devices, Figure 1(d)) demonstrates this concept. This architecture allows high availability, distributed processing, lower cost and also resource sharing (real or emulated). Sauce Labs is a representative of these solutions. Unfortunately, in this scenario, we found a lot of problems such as unstable collaborative mode condition, it can happen due the service to depend of the user; still, the resource contention leading to the interference because tests demand many parallel requests searching for computational resources and finally, queues of tests may be generated when existing only a small pool of available resources (devices).

III. Cloud-based Emulated Device Architecture Statement

The cloud computing paradigm is composed essentially of three service models that enable ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources: Software-as-a-Service (Saas), Platform-as-a-Service (PaaS), and Infrastructure-as-a-Service (IaaS) [1]. In addition, other platforms have been proposed to supply scenario-specific requirements, such as Datacenter-as-a-Service (DaaS), Database-as-a-Service (DBaaS), and so forth.



(a) Real Device Clusters (b) Emulated Device Clusters (c) Real Device Collaborative Clusters (d) Device Heterogeneous Clusters

Fig. 1. Cloud-based Device Clusters

Here we present architectural elements necessary to supply not just mobile application testing requirements, but also demands from distributed testing teams that do not fit in any of the current service models. Though we have previously presented four platforms for mobile test automation on clouds, we focused exclusively on emulated device clusters, since we have not seen any reference system using emulated devices.

We defined a cloud-based system architecture composed of high-level client-server modules. The server module sits on top of a virtualization system (hypervisor) and uses a cloud engine to control the VMs to start the device emulators. On the other hand, the client module is a REST-like interface through which users submit their testable applications with hardware/software requirements. REST APIs have been gaining ground for the past few years as predominant Web service design models. The workflow, from the user's application submission to the report the users receive, is presented in Figure 2.

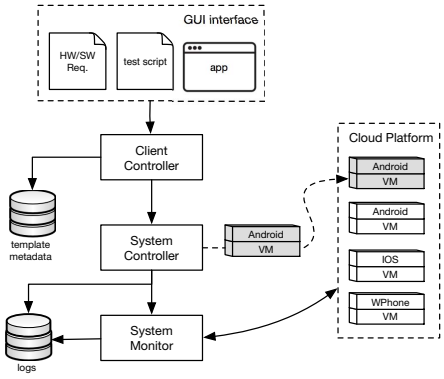


Fig. 2. Architecture Workflow Design

The workflow consists of: (1) receiving the application, the test script and hardware/software requirements from the user (OS, OS version, amount of memory, CPU, and network bandwidth); (2) selecting from storage the template (flavor) that fulfills the submitted requirements; (3) sending the hypervisor information on the template to be used; (4) starting the VMs containing the device emulators from the chosen template and running the application on them; and (5) sending reports back to the user during run-time.

The next sections describe the system's low-level components in detail.

A. Client Controller

The client controller aims to deliver an easy to use interface to manage mobile application test cycles. The idea behind the module is to provide an interface and work as a cloud-based scalable service for testing applications, using dozens or even hundreds of different emulated devices at the same time. The interface defines a set of architectural principles by which makes it possible to design Web services that focus on the system's resources, including how resource states are addressed and transferred via a wide range of clients written in different languages. In this context, we defined an interface that allows anyone to check whether their applications have been carried out correctly and efficiently across a variety of device configurations with minimal effort. The designed interface's endpoints are as follows:

Resource request: The endpoint returns the amount of available resources and the mobile OS versions currently supported by the cloud platform. This is fundamental to ensure applications' execution and to avoid resource contention in our proposed Cloud-based Emulated Device Clusters.

Application and test script submission: The service receives a JSON file describing the hardware/software requirements, the testable application, and the test script. Immediately after the script is sent, the API starts parsing the file, thereafter generating an output according to the chosen OS (e.g., MonkeyRunner python script for Android), and then it sends the final script to the controller. The service receives the binary file in Base64 format and converts it back to binary as soon as all bytes are transferred to the controller.

Monitoring: The service notifies users of the status of the submitted tests. It returns information such as elapsed time, completed percentage, and number of provisioned VMs. Also, it displays the executed instructions of the submitted test in real time, presenting exceptions that might have occurred during the test and showing step-by-step execution times.

Management: The service is responsible for handling tests in progress. It provides functions to cancel, pause or resume a given test cycle at any time. To guarantee the appropriate life cycle of each test, the application management must be done all the time, providing and deallocating resources when necessary.

Result request: The service gives users test statistics and the results generated by the test script when it has finished.

Event results such as screen capturing or file creation can as well be authenticated on the service.

Test script descriptor: To make the definition as generic as possible, the input script must be written in only one language, and also must work on any platform. Based on the work by Jun-fei Huang et al. [11], we defined requirements for a code generator which users XML as the input language, where the customer can program it with universal commands. It basically interprets the markup language and generates an output descriptor script, according to the test tool used, e.g., a python output compatible with MonkeyRunner [12]. The greatest advantage of this is that the customer does not have to worry about the system characteristics before creating the test cases, i.e., delays, dynamic storage paths, etc. The commands we defined are shown in Table I.

TABLE I. COMMANDS TO DESCRIBE FUNCTIONAL TEST STEPS OF AN APPLICATION

Command	Description
touch	touches a specific point on the screen, given by X and Y coordinates
screenshot	takes a snapshot of the screen
install app	installs an app on the device
remove app	removes an app off the device
slide	simulates a slide movement between two points (X1; Y1) and (X2; Y2)
press	presses a specific button, e.g., VOLUMEUP
upload file	uploads a file to the device

B. System Controller

A template image is used to clone, convert or deploy more VMs. Within these images are included the VM's virtual disks and settings. Templates have a great importance in the architecture. They contain pre-installed mobile OS emulators that prevent the controller from installing virtual device emulators one by one. The controller can deploy each OS version by using the template specifically made for that version.

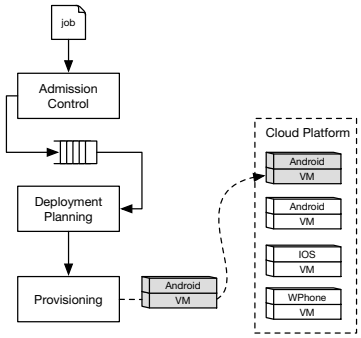


Fig. 3. Components of the system's controller

The cloud platform controls the hypervisor which in turn controls the VMs containing the mobile emulators. From the architecture point of view, The cloud platform acts like the resource manager and provides manageability features for the components presented in Figure 3 and described below:

Admission Control: An admission control phase is required to check if current resources are sufficient for the new request. The arrived request contains the user's mobile application, the test script and information about hardware requirements and mobile platform version on which the application will be evaluated. The Admission Control module checks resource availability across the cloud compute nodes. If resources are not available, then the job is placed in a scheduling queue until the resources are released and become available. It is up to the user the creation of the test script before submitting the application to the platform.

Deployment Planning: Queued jobs are scheduled as they arrived. In this process, the system must prepare a deployment plan by choosing most suitable compute nodes from the pool of available nodes. The scheduling decision may vary depending on the hardware requirements and application design. Distributed applications convey data flows through the network and have bandwidth limits that are connection-specific so that different application instances are preferable to run on multiple different compute nodes. Moreover, there may be scenarios where a controlled network path between the nodes become necessary to control either network latency or packet losses with no disturbance from other applications.

Provisioning and Resource Allocation: The system allocates resources to jobs based on the deployment plan. A resource manager system is necessary to manage the available resources and to guarantee resource isolation between multiple-consolidated test cycles. To this end, the cloud platform is used to orchestrate the pool of compute nodes and assign them to the jobs. The cloud platform can control a huge amount of computing resources (processing, storage, and network) in a data center. Once the resources are assigned to the jobs, the system starts the device emulators within each compute node and places the test application to run on those. Because emulators are OS-specific, different image templates must be build to support each OS independently. For example, there must be a template for Android, iOS, Windows Phone and so forth. Additionally, the templates must also be partitioned per OS version for ease of maintaining. Since the templates are built, the system chooses one based on the user's received software requirements.

C. System Monitor

The system monitors test's stages in all compute nodes to improve agility and responsiveness. The system gives users the possibility to track the progress of the submitted tests during run-time and receive notifications when they are finished or some test's event are raised. Since the tests have been finished, the user can visualize the results per stages through the interface.

IV. EVALUATION

Experiments were conducted to compare a non-cloud testing platform running in a single workstation versus the proposed cloud-based platform with emulated devices. It enabled testing for distributed and parallel mobile applications. In addition, we analyzed performance, scalability, and the effects caused by the time spent to deploy multiple tests and execute their life cycle.

A. Experimental Environment

Our cloud setup consists of three identical nodes equipped with two Intel Xeon x6550 processors (32 threads). They communicate through a Gigabit ethernet switch interconnected with a 800GB iSCSI SAN storage. For the local experiments, we used a workstation equipped with two Intel Xeon E5530 processors (8 threads).

As a workload, we chose a simple 3-stage flashlight application. Our test case is described as follows: in the first stage, the user chooses the color of the light. In the second, the user pushes the power button that matches the color previously chosen. Finally, the application turns the flashlight on. The stages were described through the commands we defined in Table I. The VM-wrapped emulators have a unique life cycle. They are provisioned as new requests arrive and after the deployment planning is completed. When the test cycle is finished, the corresponding VMs are deallocated and the resources are released. The number of experimental test rounds was given by a confidence interval of at least 95%. The metrics collected were execution time per test cycle and maximum number of requests.

B. Performance Analysis

By comparing the performance results of the emulated platform in Figure 5 with the tests performed in the workstation as presented in Figure 4, the difference in test cycle execution times become noticeable. This difference occurs because the workstation does not impose any resource limitation to the tests because they were performed sequentially—running sequentially is a restriction imputable to the Android emulator when running standalone. However, when the test cycles were provisioned on the cloud infrastructure, it was possible to consolidate them on the same physical machine and perform them in parallel. Since the VMs start to compete for the same hardware resource, it leads to a contention scenario where performance may vary unpredictably.

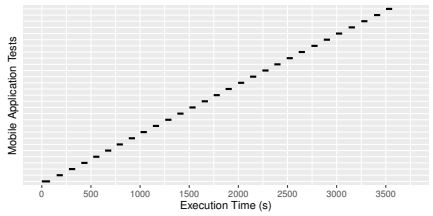


Fig. 4. Performance evaluation of tests running sequentially in a single machine

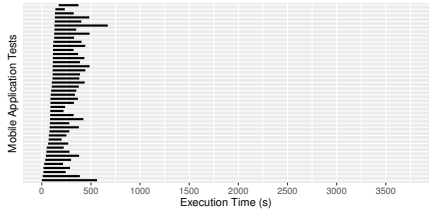


Fig. 5. Performance evaluation of multiple tests running simultaneously on cloud.

Additionally, we also evaluated the CPU consumption while the test cycles were carried out in the workstation. As can be seen in Figure 6, the test cycles consumed the maximum processing capacity of the machine throughout execution, making it unusable for other tasks.

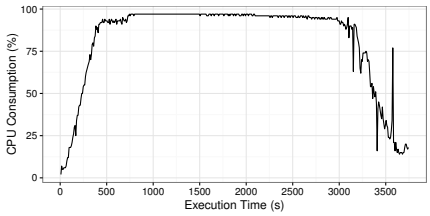


Fig. 6. CPU consumption throughout the tests in a single machine

It is worth noting that cloud computing is beneficial to local test environments due to a number of factors, such as higher scalability, lower infrastructure costs, and easier maintenance. Moreover, the results showed a super-linear behavior when running tests locally, presenting about 80% performance degradation compared to the cloud platform.

V. Conclusion and Future Directions

Software testing tools make life easier for many developers. Most of the testing methods and practices are not very different from years ago. Although there are many tools and techniques available, the goal of the proposed architecture is to use cloud computing to automatically test mobile applications at a low cost. Our experience conveys that the decision between emulation versus real-based cloud solutions might depends on the application development lifecycle. For example, developers can use real-based ones to verify the application look and feel, then move on and do large-scale tests on emulated devices.

Cloud computing has evolved at a rapid pace. We believe this work is complementary for a new cloud computing model generation which aims to deliver mobile application testing as a service independent of mobile OS versions or resource characteristics at a low cost.

References

[1] P. Mell and T. Grance, "The nist definition of cloud computing," 2011.

[2] "Amazon Web Services," 2016. [Online]. Available: http://aws.amazon.com/

[3] P. Iannucci, M. Gupta *et al.*, *IBM SmartCloud: Building a Cloud Enabled Data Center*. IBM Redbooks, 2013.

[4] "Google Cloud," 2016. [Online]. Available: https://cloud.google.com/

[5] N. Middleton, R. Schneeman *et al.*, *Heroku: Up and Running*. " O'Reilly Media, Inc.", 2013.

[6] J. J. Arbon, "App quality: Secrets for agile app teams," 2014.

[7] "Xamarin test cloud: Mobile app testing made easy," 2016. [Online]. Available: https://xamarin.com/test-cloud/

[8] "Amazon device farm," 2016. [Online]. Available: https://aws.amazon.com/pt/device-farm/

[9] "Pcloudy App Testing Tool," 2016. [Online]. Available: https://www.pcloudy.com//

[10] "Sauce Labs Mobile Testing," 2016. [Online]. Available: https://saucelabs.com/

[11] J.-F. Huang and Y.-z. Gong, "Remote mobile test system: a mobile phone cloud for application testing," in *Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on*. IEEE, 2012, pp. 1–4.

[12] "Monkey Runner," 2016. [Online]. Available: https://developer.android.com/studio/test/monkeyrunner/index.html