Kongdroid: A proposal for a Cloud Service for Stress Testing on Android Applications

Leonardo M. A. Sodré, Felipe Silva Ferraz, Gustavo Henrique da Silva Alexandre, Ana C. L. De Carvalho

CESAR
Centro de Estudos e Sistemas Avançados do Recife
Recife, Brazil
{lmas, fsf, ghsa, alcl}@cesar.org.br

Informatics Center
Federal University of Pernambuco
Recife, Brazil
{fsf3, ghsa}@cin.ufpe.br

***Abstract* – This work proposes a new and scalable service for stress testing on Android applications. This tool is available through cloud computing resources to support developers in their applications validation, aiming robustness, stability and compatibility, in different devices before commercial deployment. The solution focuses on the generation of a certain number of pseudo-random user interface events in the installed application in an emulator. This emulator is created from real images, of customized versions of the Android platform, running in well known devices. This execution results in a report containing the events that were successfully and those that failed due to any specific reason.**

Keywords-cloud computing; stress testing; remote testing; mobile applications; Android

I. INTRODUCTION

The software development, for mobile devices, and the conduction of large-scale experimental developing studies using real person, have become easier through the creation of app stores, and by using those stores as a mechanism for a significant number of users, to publish applications they have authored. An example of this was the emergence of the Apple store, which popularized this type of service. Unlike Apple's iOS platform, Google's Android open platform does not impose restrictions on its operating system; thereby, creating favorable conditions for various hardware manufacturers to adopt these devices. However, this benefit comes at a price: the challenge has become to develop interactive applications that need to run on a variety of these manufacturers' items of hardware equipment, each with its own customized version of the operating system, different hardware resource capabilities and screen resolutions and functionalities. Another relevant factor is the evolution of the Android version, where the application needs to track changes on the platform to keep operating properly.

Taking advantage of this benefit of the open platform and manufacturers' mass launch of more affordable Android devices, according to a recent survey, last year, the Google Store tripled in size, with its stock in 2013 amounting to about 800 (eight hundred) thousand applications [1], and recorded more than 25 (twenty five) billion downloads in 2012 [2].

Even though this demand has created the benefit of a proliferation of applications, it has also presented the need to address a growing issue: they have difficulty in generating various user events to stress the application and check if any exception occurs; in testing the capacities and resolutions of Android devices on different models; and there are few physical models available for testing. This difficulty of having an insufficient number of devices is also a reality faced by organizations.

Among the techniques used in this study was that of using an Android emulator instead of the physical model. This is because unlike the iOS simulator [3] and its resource constraints, the emulator reproduces a real device efficiently. This decision to use an emulator was further strengthened when it became feasible to configure the emulator, released by the manufacturer, with real versions of the Android platform. Given the support of cloud computing resources, it was possible to pre-configure these emulators in a scalable environment, thus enabling it to be used in parallel, so as to meet users' requirements as to running their application on several mobile devices. By means of an Application Programming Interface (API) accessed through an Internet browser, the user accesses this cloud environment to subject his/her application to testing, for which a script will be generated automatically to install the app in the emulator and apply the stress command using the Android Monkey tool, native to the platform. If the processing of the test demands a high consumption of infrastructure resources so as not to compromise the run quality, a new instance may be used to balance these resources in order to ensure the delivery of the results.

The program put forward in this paper to tackle these difficulties is called a Kongdroid. This enables the developer to use a prepared and configured environment in which to conduct stress testing [4]. It is hoped that, by having this facility, the knowledge of test development that a developer needs will be reduced and that time will be gained as there is no need to prepare an infrastructure since this is provided by this service. As a result of using the Kongdroid, it is estimated and it will permit the publication of more robust applications that are compatible with various Android device models, i.e., that it will indicate possible areas for improvement, so as to anticipate corrections, while

 ISBN: 978-1-61208-304-9

the model is still in the development phase. Problems of the type in which the application is unexpectedly closed are among the situations that are not so easy to spot, unless features such as the Android Monkey [5] are used.

This was a structured study, which began with the authors deepening their knowledge of the technologies used and a review of the literature so as to be able to cite related studies. This introductory section draws attention to the state of the art with the issues related to Kongdroid. Then the proposed solution is detailed by describing the techniques used to create the service and matters to be careful about and points to consider. These strongly guided the study while it was being developed. After recording the approach to finding a solution, an account is given of the planning, implementation and the comparison of two experiments undertaken in which the solution was applied so as to give evidence of how important it is to use it. To summarize all the work done, the concluding section indicates the improvements achieved in the state of the art, the advantages and limitations of Kongdroid, its possible applications and ideas on how it may evolve.

This paper is divided as follows: The first section presents a short introduction, section II presents the state of art about topics used in this paper, section III presents the proposed solution and how it was developed, the experiments and results used to validate the tool are depicted in section IV, finally section V presents some conclusions about this work.

II. STATE OF THE ART

A. Cloud Computing

Cloud Computing [6] is the representation of the applications made available as a service on the Internet and by *software* and *hardware* in the data centers that make these services feasible [7]. There are many definitions of Cloud Computing, but some features are held in common by most of them, for example, virtualized environments and providing computing resources on demand. This type of service is commonly called a public cloud. A private cloud is a center with data restricted to a specific company or of limited access [8].

Cloud Computing is divided into three main types to offer services, as shown in Figure 1: Software as a Service (SaaS) [9], Platform as a Service (PaaS) [10], and Infrastructure as a Service (IaaS) [11].

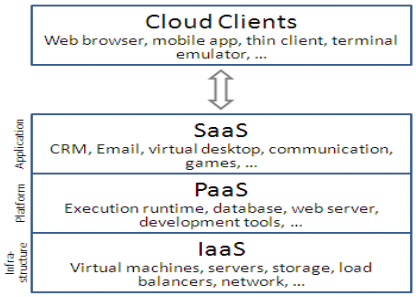

Figure 1. Cloud computing at different levels

Related to this work, there is a type of cloud computing called Testing as a service (TaaS), which offers users testing services, such as the automatic generation of test cases, automated conduct of tests and evaluation of test results [12]. Testing tasks can be modeled using ontology techniques, and they can be combined based on a shared ontology model, along side with TaaS, there are, other subtypes: Development as a Service (DaaS) [13], Communications as a Service (CaaS) [14] and Everything as a Service (EaaS), which are not part of this scope.

B. Android Platform

Android [15,16] is a platform for mobile devices that runs on the nucleus of the Linux operating system but developed into a structure external to this nucleus [17]. The Android operating system was initially developed by Google and later by the Open Handset Alliance (OHA), which is a group of large companies in the telephone mobile market such as HTC, LG, Motorola, Samsung, Sony Ericsson, Toshiba, Nextel, China Mobile, T-Mobile, ASUS, Intel, Garmin and others. OHA is led by Google and the group's goal is to define a single open platform for mobile phones; thus, making consumers more satisfied with the final product. Another goal of the group is to create a flexible platform on which to develop applications. The birth of Android came about based on these objectives for which OHA is responsible for maintaining a standard platform where all the new market trends are present in a single solution [17, 18].

Android applications in [19] are built using Java language; but, there is no Java virtual machine in the operating system, only a virtual machine optimized for mobile devices called Dalvik [20, 21].

C. Monkey Test

Android Software Development Kit (SDK) [22] makes a Monkey test tool available to generate pseudo-random user events such as clicks, touches, or gestures and other events at the system level. As the guide to the Android platform itself says, "You can use the Monkey to stress-test applications that you are developing, in a random yet repeatable manner "[5].

The Monkey is a tool accessed via the command line that can be run on an instance of the emulator or mobile device. There are four main categories of options: basic configuration, such as the definition of the number of attempts for random events; operational restrictions, such as

 ISBN: 978-1-61208-304-9

restricting the test to a single package; event types and frequencies; and debugging options [5]. During these events the tool observes three conditions, which deal specifically with the following: if it is restricted to execution in one or more specific packages, it watches for attempts to browse for other packages, and blocks them; if the application crashes or receives any type of *not-dealt-with* exception, the Monkey will stop and report the error, and if the application generates an *application not responding* error, the Monkey will stop and report the error [5]. Other types of behaviors of defects that the Monkey does not detect can be mapped by other types of smart Monkey tools [23]. Other related studies use stress testing: AASanbox [24, 25] and model-based GUI for Android applications [26].

D. Testdroid

The Testdroid is a useful tool for Android application developers who can validate if their application is compatible with several other types of devices [27, 28]. It is proposed to perform a specific set of user actions on one or more real device and collect and report test results. It is a service that is available on the Internet, for which the steps: Record your test, Run test on real devices and check reports.

III. PROPOSED SOLUTION

The proposed service is committed to providing a check on the user's application, using stress testing [4], based on the native Monkey Test tool of the Android platform, to validate the robustness and compatibility in various telephone options and other mobile devices. After it has been run, reports of the results are generated for data analysis and emailed to the client. With such data, the client will obtain valuable information to support improving the application and ensuring quality, as shown in the proposed high-level architecture in Figure 2. It will also lead to a better understanding of the flow of the run and the entities involved. In the following sections, this paper will describe this solution in greater detail so as to understand its methodology, structure and development decisions. Real devices are dispensed with because the tests are run exclusively on emulators.

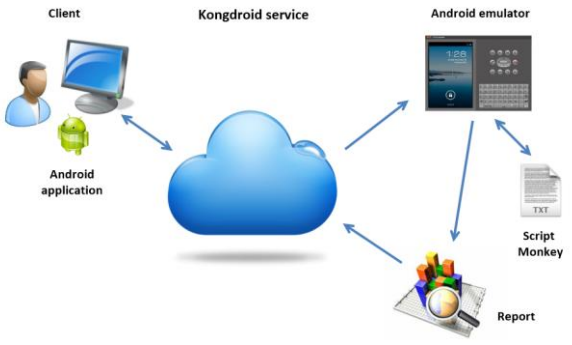

Figure 2. High-level definition of the architecture proposed

A. Definition of the architecture based on the cloud

After a detailed study of the necessary functionalities of the solution, it became very clear that to meet the user demand, the architecture should have the following quality attributes:

- Availability: The system will be available 7 days a week and 24 hours a day;
- Integrity/security [29, 30]: Only users with access privileges may configure and run tests. Every application transferred to the service and tested will be discarded at the end;
- Interoperability: The solution should be able to operationalize its being implemented in different modules, management and others, to conduct testing processes, running on different operating systems, Windows and Linux, respectively;
- Usability: A new user should be able to conduct a test of an application without the need for guidance, only with the support of tips on the filter options of the commands;
- Scalability [31]: The service should scale computing resources whenever there is a need to ensure the correct balancing of the processing of users' requests.
- Use of standards [32, 33]: The solution should support pre-established script models for running tests, so that they are dynamically created from the selection of the options of mobile devices.

To meet these requirements, an infrastructure benchmark on the market was adopted and widely used by several companies, *Amazon Web Services* (AWS) [34]. AWS offers a variety of cloud services, of which the one that stands out is Amazon Elastic Computer Cloud (EC2) [35], which permits the rental of instances of virtual servers that can be scalable to the extent that the solution needs both processing and to place limitations on software.

B. Definition of the standards used

In order to structure a better service, it was very important to define models and nomenclature standards and the target of the resources. In the presentation of the solution to the user, there is the possibility of selecting more than one type of mobile device. This feature led to a considerable complexity, since the architecture should be flexible enough to allow the addition of new devices without causing the work developed to be reworked. In meeting this functionality, for each model made available, a base script model to carry out the commands of the test is defined. After the executive action of the user, this standard model is used, based on the filters selected by the user, to generate dynamically the final script for conducting the test.

 ISBN: 978-1-61208-304-9

1) Nomenclature of commands and resources

First of all, the target folder for each authenticated user was identified, named by his/her identification in the system. In this folder, what are stored are all the resources to be used such as: application to be tested; test script commands for mobile and log files of the relevant events, commonly called a log.

A unique identifier is assigned to each mobile device option. Thus, the identifier is used for all command scripts generated. For the identifier "001", the script will have to be generated with the following format: Script_Monkey_001.bat. It is also used to generate the logcat (relevant phone events) with the following format: LogCat_001.log and generation of the Monkey test log (relevant events of the stress test command) with the following format: LogMonkey_001.txt.

One of the commands carried out by the command script is the startup of an Android *Virtual Device* (AVD) to start the emulator. For each type of mobile device, an AVD was created with the following format: avd [identifier], ie, for the device with the ID "001", the nomenclature is avd001.

C. Preparation of the environment

As cited in the definition of the architecture, the AWS infrastructure was chosen to make the service available in the cloud. For this setting to work properly, an extensive level of knowledge of managing servers or operating system processes was not required, but some settings are essential so it works correctly.

1) EC2 Structure

For this study, a new account was created on the AWS and the EC2 service used to create the instance of the "t1.micro" type on the platform of the Windows operating system, Server 2008 R2. This type of instance has limited resources (CPU and RAM memory) and its use is free for one year, i.e., payment for its use is not required unless use exceeds some preset limits. When the instance is available, access can be gained through a *Domain Name System* (DNS) with a dynamic IP (internet protocol) address. This is not the best option because at every reboot of the instance, this address is modified. To overcome this drawback, the EC2 service has an elastic IP resource, i.e., for the public DNS, it is assigned a static IP address, thus ensuring there is always access to the same address.

2) Configuration of the Android platform

To use the Android emulator platform and to carry out the Monkey commands, the Android *Software Development Kit* (SDK), version 21, and the Java *Runtime Environment* (JRE), version 1.7 have to be installed in the AWS. Environment variables were created: "ANDROID_SDK_HOME" containing the path of the Android SDK and "JAVA_HOME", containing the path to the JRE.

After properly installing the Android, the Android SDK *Manager* had to be run to complete the upgrades of associated tools. Among these updates, one requires special attention, namely, the Google APIs Add-On. The add-on provides system images compatible with Android that runs on the Android emulator, thus enabling the application to be debugged, run and tested before publishing it to users. Several mobile phone manufacturers have these images on their web pages targeted on application developers. For this study, the images used were from the Motorola manufacturers: Atrix 2 and Razr and LG 3D Optimus model [36,37].

3) Configuration of the Microsoft platform

To implement the solution developed in ASP.NET MVC 4 [38], it was necessary to install the *Internet Information Service* (IIS) version 7.5 and the Microsoft. NET Framework 4.5 in the AWS. For IIS, it was necessary to create an application called "monkey", where the implementation of the solution was stored and the right of full access to the folder called "Content" of the application was assigned to the user of the IIS (DefaultAppPool), so that "Content" allows the resources used to be stored and altered.

D. Model of Monkey script

As previously mentioned, to make the service flexible as to replacing and/or adding new options for mobile devices, a script model was created to run the commands needed to perform the stress test. This stage of the project required close attention and simulations to determine the optimal sequence of actions to ensure better efficiency in the results hoped for. The use of Android emulators involves a series of difficulties when they are in an automation process, since the ability to foresee the time needed to trigger each command is not precise, and, therefore, auxiliary actions were used to minimize this uncertainty. Other resources were also taken advantage of to have the emulator perform better, since there was not the need for a graphical display. The automated commands in this script can be run manually in the user's environment, but they involve complexity in configuring the necessary tools and environment variables.

The identifier of each mobile device option was parameterized in this model so that all resources accessed and generated are easily referenced, based on the data selected by the user, the script is easily generated and applied in the environment of the solution.

1) Selection of port

For this automation would function properly, the environment was totally controlled, i.e., for each script generated a known number of the network port is generated and later will be attributed to the Android emulator. This strategy is of fundamental importance to free the memory of the emulator at the end of the test. The generation of the port number is made at random between 5554 and 5584, this range being reserved for this type of program. By default, if the port does not specify it, it is associated with the generating the numbers 5554 and 5555 (this second port is

 ISBN: 978-1-61208-304-9

reserved for the *android debug bridge* (ADB)) and should another emulator be run in parallel, the next port is that of the number 5556, and so on, successively.

The following is an example of a command linking port 5558 to the emulator:

```
emulator -ports 5558,5559 -avd avd001
```

In the example below, there is a sample command for a specific emulator, for the installation of the user's APK on the emulator:

```
adb -s emulator-5558 install helloWorld.apk
```

2) Estimate of time for each step

When using the mobile device or emulator, the focus of this study, in a process of stress test automation [39], the time it is estimated time for each command to be executed should be taken into account. If there is not enough time left over for the next command to be applied under favorable conditions, the procedure, as a whole, will be compromised and aborted. To ensure the efficiency of the script actions, possible points of delay were identified and auxiliary commands were defined in order to be used following these main ones, i.e., promoting a longer time so that the environment is in a fit state for the next step. One difficulty found was that the operating system does not provide a specific command for this situation, where, to solve this limitation, another command was used to obtain the same result. The following is an example:

```
ping 1.1.1.1 -n 1 -w %_timer% >NUL
```

For an operating system with a TCP/IP client, the PING command can be used to delay the run by a number of seconds. If specified (-w), the PING will wait for a number of milliseconds between two pings before giving a time limit. The environment variable, represented by (*% timer%)*, contains the time that the action will imply.

This feature was used to overcome three points of slowness:

- Running the Emulator: Estimated time of 240,000 (two hundred forty thousands) milliseconds;
- Installing the Android Application Package (APK): Estimated time of 20,000 (twenty thousand) milliseconds;

Conduct of the Monkey test: Estimated time of 120,000 (one hundred and twenty thousand) milliseconds. A fixed value was used due to the project being limited to 500 (five hundred) random events. In an environment with a high processing infrastructure, without limitation on events, this estimate would need to use a formula such that the time might vary proportionally.

3) Tool for recovering the APK package

In order for the command for the stress test to be able to restrict the target application, it is fundamental to know the name of the package that will be used as a parameter. For the purposes of promoting a better experience for the user, when using the service to enter and select data to perform the test, there is no need to register this package in order to avoid errors when typing manually.

To meet this situation, a tool called *android-apktool* was used. This is a tool available in the repository of Google projects under the *Apache License 2.0*, which undertakes reverse engineering on Android APK files. It can decode resources to nearly their original form and rebuild them after some modifications have been made. Thus it was possible, starting with the APK user, to decode the information of the package and use it as a parameter in the command of the stress test.

4) Definition of variables

When defining the script model definition, some temporary environment variables were created to make it possible when the script was generated to have a specific one for the mobile device model and for the dynamic use of information in the commands to be executed.

The example below better illustrates the need to use these variables:

```
%_adbPath% -s %_serialEmulator% install
%_apkPath%
```

The same command used in session 3.5.1 to install the user's APK, but this time the variable *%_adbPath%* was used, which identifies the path of the Android ADB program to carry out the commands, *%_serialEmulator%*, which identifies the serial or port in which the emulator is running, and *%_apkPath%*, which identifies the path of the user's APK stored on the server.

5) Command to optimize the emulator

To avoid overloading the server, should more than one instance of the emulator be run, unnecessary features in an environment may be discarded without interacting with the user. Thus, the options of initial animation, graphical and audio interface were disregarded. The following is an example of the command:

```
emulator -ports 5554,5555 -no-boot-anim -no-window -
noaudio -avd avd001
```

6) Command to instal the Apk

After the above command to run the emulator, the next to be auctioned is to install the user's APK. The ADB provides an option so that this action occurs only when the emulator is "ready", thus avoiding error and the script being interrupted. The following is an example of the command used:

```
adb -s emulator-5558 -e wait-for-device install -r
helloWorld.apk
```

 ISBN: 978-1-61208-304-9

7) Command to unblock the screen

During the period of testing the solution it was realized that the emulator on being started, by default, is left with its screen blocked. Thus the command of the stress test was discarded. To resolve this issue, a command was included in the script to send a screen unblock event. Below is an example of the command used:

```
adb -s emulator-5558 shell input keyevent 82
```

8) Command to conduct the Monkey

For the main action of the script, the following command to run the stress test was specified:

```
%_adbPath% -s %_serialEmulator% shell monkey -v -v
-p %_apkPackage% " + monkeyParameters + "
%_monkeyEvents% > %_logMonkeyPath%
```

The arguments-*v*-*v* promote greater information in the tests run. The *%_apkPackage%* variable stores the name of the package, extracted as described in section 3.5.3. The *monkeyParameters* development variable stores the set of options for user-selected parameters. The *%_monkeyEvents%* environment variable stores the number of interface pseudo events reported by the user. The *%_logMonkeyPath%* variable stores the service path to record the results of the stress test.

E. Conduct of the test

To run the tests it was defined that the service should possess a modularized and simplified flow. Using few steps, the application meets a demand and then has the capacity to quickly return to the initial state for a new request from the user.

As shown in Figure 3, after the user obtains his/her authentication, he/she is directed to the starting point of the service. The first piece of information requested is the submission of the application package to be tested, the Apk Android. This transfer is performed securely and at the end of the process it should be discarded. Still on the main screen the user will need to provide other important pieces of information, such as the email to, which results should be sent, to select which application of the device models should be validated, the number of pseudo events and other optional choices regarding the stress test, these being Monkey event options and Monkey debugging options. After completing the data and confirming the start of the operation, the system will validate them and if there is no criticism, the service will be started.

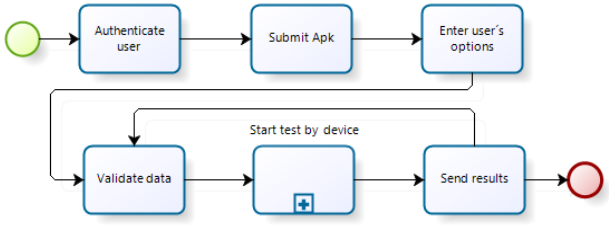

Figure 3. Main flow of the conduct of the test

From this point on, the system has already allocated the user´s physical space and the Apk Android is available for use in the emulator, such that the script should be generated and executed by the device model chosen. After each run of this script has been concluded, an email will be sent to the user and the result attached.

Figure 4 below shows each step of the test run by the model of the device selected. This process is performed in parallel so that the service does not take up the hardware resources of the AWS infrastructure for a lengthy period of time. Each run of an emulator requires a high level of processing and memory, in which the orchestration of these elements monitors the need to allocate more resources, i.e., whether another server will need to be initialized to balance and ensure the quality of the system.

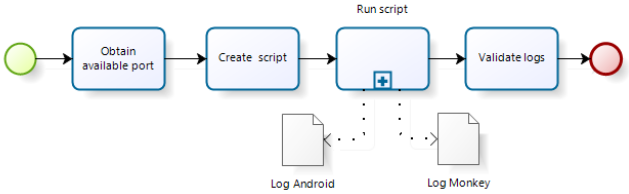

Figure 4. Secondary flow of the test per device

In the flow of Figure 4, the first step is to check and select the port number of the server where the emulator will be allocated. This port is one of the parameters used for the next step, the creation of the script. At this point, what are defined are the times between each execution of a command are defined, the parameters entered by the user to compose the command Android Monkey command, the physical path in the server of the user's location to generate the results and the path for the Apk of the target application of the tests. During this run, a log file of the events generated from the emulator and another log file of the events of the stress test are generated with the test result. The flow is finalized with the validation of these files.

Figure 5 below shows the sequence of the commands that make up the test script. The run starts by using environment variables used during actions in the emulator. Via the *android-apktool* tool, the name of the Apk package is recovered and stored in an environmental variable to be used later in the command of the stress test. The next step is to run the emulator, in which, to ensure optimum performance, parameters are used to bypass the startup animation, the audio and screen. At this point, the

 ISBN: 978-1-61208-304-9

generation of events of the emulator is also triggered. The next action is to install the Apk of the application which, via the *wait-for-device* parameter, is only run after the emulator is found in the *device* state, i.e., its instance is prepared to respond to the user's actions. After the Apk has been installed, but before running the stress test command, the screen must be unblocked, since without this step the pseudo events of the Monkey android are prevented from interacting with the application. From this point on, the emulator has the necessary condition for the stress test to start and to record on file the events in order to compose the result, whether there was a failure or success.

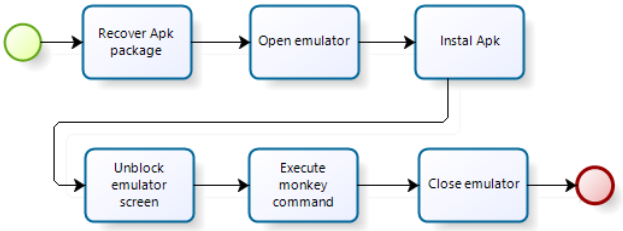

Figure 5. Secondary flow of the execution of the test script

After all the commands have been carried out, the Apk application is uninstalled and deleted and the instance of the emulator is closed.

F. *Summary of the solution*

Figure 6 illustrates the architecture of the solution at a more detailed level, where the user, via a web browser, submits his/her application and informs the proposed cloud service of the parameters desired, which are loaded to run and scale in an orchestrated way all the resources required, such as to ensure the expected results.

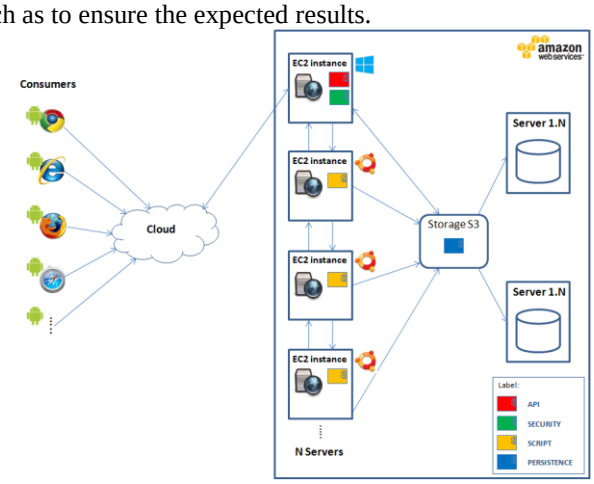

Figure 6. Low-level definition of the architecture proposed

As Figure 6 shows, the first instance used of a virtual server is that of a Windows Server, which is responsible for starting the service and using the data selected by the user, it dynamically generates scripts, per device, to be run. In the second moment, another instance of a virtual server is initialized, but this time is used for the option of a Linux Ubuntu machine. The scripts of the stress test are run in parallel in this new instance; should it be necessary, another instance with the same settings can be used without compromising the total flow of the solution. After finalizing the conduct of the stress test, a report is stored and sent to the user so he/she can analyze it.

IV. EXPERIMENTS AND RESULTS

To prove the correct functioning of the entire solution, two examples run on Kongdroid will be described. The input parameters and the expected result will be specified, as well as a comparative analysis to prove why using the tool as a support tool for developers of applications is important before publication to future users.

A. *Experiment undertaken*

In the selection of the applications, the following strategy was used: both should appear as published in the Google Store (Google Play), an example of a simpler application with a satisfactory result, and another example of an application of more moderate complexity with a fault in the test of the application not responding (ANR) type.

For the simple application, one was selected from the calculator type, categorized as a utility, called *Shake Calc*. It is proposed to be a scientific calculator with the following features: accelerometer to finalize the calculation, basic vision for access to the more frequently used functions and more complex calculations; it can switch to an advanced mode of exhibition with a touch from the user, as shown in Figure 8. For the application of moderate complexity, one was selected of the type with tables, categorized as children's games, called *Smart Bubbles.* Figure 7 shows the mentioned game that is proposed to be a math table with the following features: a game to learn the tables in a fun way, during the game, equations and bubbles with numbers are presented; for each equation, a bubble appears with the correct result and some others with wrong results.

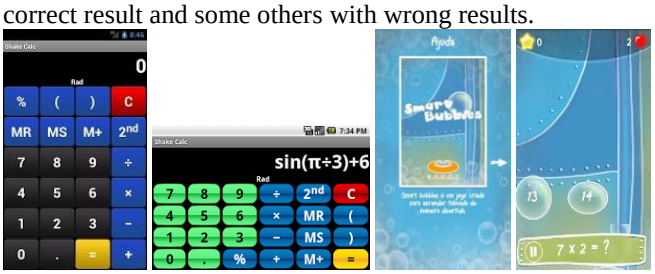

Figure 7. The first two figures represent Shake Calc and the last two Smart Bubbles

The Kongdroid was started after being informed of the following input parameters: Choice of the APK of the application to send to the service, the email to receive the results, the target device of the stress test selected (for the *Shake Calc*, the Motorola Atrix 2 model was used and for

the *Smart Bubbles*,the Motorola Razr was used). It was also informed of 500 (five hundred) pseudo random events, 300 millisecond gaps between each event, to ignore *crashes*, to ignore *timeouts* and to ignore *security exceptions*. These latter three parameters are generally used to provide for the test being run completely, unless it is finalized by the operating system.

B. Metrics used

To better measure and condition the comparison of the results for a more realistic analysis, the metrics were defined of the total number of events per the number of events run and the number of application runs by the number of applications successfully tested.

C. Results

After submitting the APK application to Kongdroid and informing it of the input parameters for each application of the experiment, the cloud service will process the stress test. This step takes less than ten (10) minutes, since every action has a maximum time configured to be run on the application installed on the Android emulator for greater efficiency in allocating and releasing resources, as well as in the response time of the results to the user. Upon completion of the due tests, an email is created with the log files of the environment and the Monkey with a text attached, whether the test was successful or not, and sent to the user´s email address so he /she may investigate and analyze the results.

In the most significant part of the Monkey log file of the stress test done on the *Shake Calc* application, it is observed that the test was successfully completed by the text "// Monkey finished", where all five hundred pseudo random events were run without an exception having occurred.

In the most significant part of the Monkey log file of the stress test done on the Smart Bubbles application, as cited when planning the experiment, the log highlights the failure of the test by the ANR type of error that occurred, where the application on receiving a given pseudo random user event, a certain time without a response which leads the operating system to cause the error in the application and to close it immediately so as not to compromise other functionalities of the device. If another type of error occurred in the application, it would also be recorded on the Monkey log.

D. Comparison of the results

Metrics were applied with the following evidence:

- Number of events per total number of events run: For the *Shake Calc* application of the five hundred pseudo random events programmed all were successfully run. For the *Smart Bubbles* application of the five hundred scheduled events only twenty-five were run successfully. As shown in the graph in Figure 8:

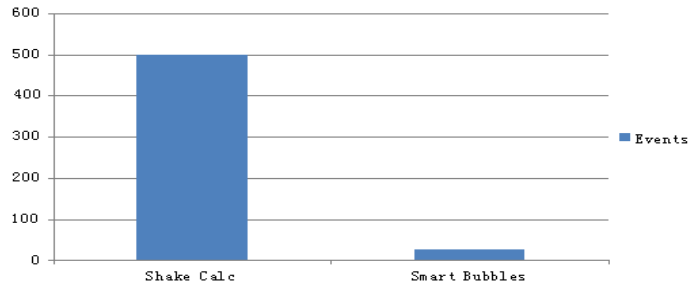

Figure 8. Number of random pseudo events

Number of applications per number of applications successfully tested: Two applications selected for the experiment, where one had a successful test (*Shake Calc*), and one had a failure in the test (*Smart Bubbles*).

By using the results of the metrics, very different scenarios and conclusions can be obtained. While it was attested that the experiment conducted with the *Shake Calc* application, after subjecting it to a significant load of user events, its stability responded effectively, thus ensuring that its publication and other devices running on Android had greater reliability, in the experiment conducted with the *Smart Bubbles* application, it was proven that it does not have the efficiency to withstand a greater number of User Interface events, in which when a severe ANR error occurs, the application needed to be finalized by the operating system.

This type of error could be avoided in the development phase by using a tool like Kongdroid, so that the credibility of the application is not threatened. This is a real threat given that the application is published and the user on downloading it could come across the kind of situation where he/she may suddenly be impeded from continuing to use it and which may easily cause that the application can no longer be used.

To better attest the efficiency of this work, another ten (10) applications from the Google Play store were selected, all of which were downloaded by a significant number of users. The tests were performed on three device models offered by Kongdroid, LG Optimos 3D, Motorola Atrix 2 and Motorola Razr. The following Figure 9 shows the results of the stress tests:

 ISBN: 978-1-61208-304-9

Application	Category	LG Optimus 3D	Motorola Atrix 2	Motorola Razr
Aviary Photo Editor	Photography	Passed.	Failed. Occurred ActivityNotFoundException after 162 events.	Failed. Occurred ANR after 14 events.
Bradesco	Finance	Passed.	Passed.	Passed.
Banco do Brasil	Finance	Passed.	Passed.	Passed.
CPqD Liga +	Communication	Failed. Occurred ANR after 19 events.	Passed.	Passed.
Jogo da velha	Games	Passed.	Passed.	Passed.
Lanterna Led HD	Utilities	Passed.	Passed.	Passed.
Memory	Games	Passed.	Passed.	Passed.
Paint Joy	Games	Failed. Occurred ANR after 207 events.	Failed. Occurred ANR after 18 events.	Passed.
Remember The Milk	Productivity	Passed.	Passed.	Passed.
Shazam	Music and Video	Failed. Occurred ANR after 204 events.	Passed.	Passed.

Figure 9. Results in another 10 applications

Among the related studies presented in this paper and other test automation tools surveyed, characteristics similar to those in Kongdroid were not found. Due to this, it was difficult making it possible to compile a valid comparison test to attest to its efficiency. This is why the focus of the experiments and results was on validating the quality of existing applications in the Google Play store when subjected to stress tests in different mobile device models.

V. CONCLUSION AND FUTURE WORK

In this paper, we have presented the Kongdroid, a cloud computing service to automate stress testing so as to analyze Android applications. It is shown how the Android emulator can be used to run applications in an isolated and pre-configured environment with real images of versions of operating systems released by device manufacturers. The main purpose of this solution is to enable developers to subject their application to a high number of pseudo random user events in various Android devices to assure their effectiveness as to the correct conduct of the functionalities.

Its importance is due to the fact that of its offering a thorough knowledge of stress testing techniques, where the developer will be able to use a pre-prepared environment to validate his/her application in various device models with different capacities and resolutions.

The advantages of using this service are obtained because of the detailed results of the environment and events performed being sent more speedily to the user for his/her analysis. This makes it an important tool in supporting development in order to pinpoint quickly areas to be improved before publication in the Apps store. The previous limitation that the developer had due to restricted use for testing on devices no longer exists.

Among the limitations of the service, there is the difficulty of repeating the test effectively, restricting the stress test to one application screen, the difficult of closing a specific instance of the Android emulator in the Windows environment, the absence of images of the Android platform for a given mobile device model and the high consumption of memory and the limit of instances of the emulator.

The results obtained from the experiments undertaken show there is no effective control by the Google Store as to effective compatibility of their applications in the different models found in the market. In this case, the assurance needs to come from the very author of the application using a tool such as Kongdroid.

One of the main contributions of this paper was that of permitting the developer the facility of lessening his/her need to acquire extensive knowledge of test development. Without requiring complexity when preparing a test environment, the service offers simplicity when generating a considerable number of the user´s interface events in the target application. This initiative enables the publication of the application to be more robust and compatible with various models of Android devices. Another important point is to anticipate improvements and corrections during the development phase, because what are avoided are problems of the type in which the application is ended unexpectedly during use. Besides costing less to correct before publication, this does not adversely affect the credibility of the author of the application.

For future research studies, we plan: adding new options for device models; a new mechanism for freeing the emulator at the end of the test for the Windows environment; a real-time listing of events being run; a test result in a more professional format; comparative results between devices tested; improving the performance of the emulator and; creating an orchestrator to manage cloud computing resources more efficiently

REFERENCES

[1] I. Paul. http://www.rssphone.com/google-play-store-800000-apps-and-overtake-apple-appstore/. Accessed in February 2013.

[2] Z. Lutz. http://www.engadget.com/2012/09/26/google-play-hits-25-billion-app-downloads/. Accessed in November 2012.

[3] M. Goadrich and M. Rogers. Smart smartphone development: iOS versus Android. In SIGCSE, volume 42, 2011.

[4] Ham K.H., Park, Y.B. 2011. Mobile Application Compatibility Test System Design for Android Fragmentation. CCIS 257, pp. 314-320.

[5] UI/Application Exerciser Monkey, http://developer.android.com/guide/developing/tools/monkey.html. Accessed in February 2013.

[6] Parkhill, D. The Challenge of the Computer Utility.

 ISBN: 978-1-61208-304-9

[7] A. Bechtolsheim. Cloud Computing and Cloud Networking. talk at UC Berkeley, December 2008.

[8] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. Above the Clouds: A Berkeley View of Cloud Computing. UC Berkeley, February, 2009.

[9] Candan, K. S., Li, W.-S., Phan, T., and Zhou, M. Frontiers in Information and Software as Services. In Proceedings of the 25th IEEE International Conference on Data Engineering (ICDE2009), pp. 1761-1768, 2009.

[10] D. Cheng. PaaS-onomics: A CIO's Guide to using Platform-as-a-Service to Lower Costs of Application Initiatives while improving the Business Value of IT. Technical Report, LongJump, 2008.

[11] S. Bhardwaj, L. Jain, and S. Jain, "Cloud computing: A study of infrastructure as a service (IAAS)", International Journal of engineering and information Technology, vol. 2, no. 1, 2010, pp.60-63.

[12] L. Yu, S. Su, J. Zhao, et al, "Performing Unit Testing Based on Testing as a Service (TaaS) Approach", Proceedings of International Conference on Service Science (ICSS) 2008, pp. 127-131.

[13] K. Matsumoto, S. Kibe, , M. Uehara, and H. Mori. "Design of Development as a Service in the Cloud" Network-Based Information Systems (NBiS), 15th International Conference on, (2012). Kawagoe, Japan 2012.

[14] L. Youseff, M. Butrico, and D. Da Silva. Toward a unified ontology of cloud computing. In Grid Computing Environments Workshop, 2008. GCE'08.

[15] S. Brahler, Analysis of Android architecture. Karlsruher Institut für Technologie, http://os.ibds.kit.edu/downloads/sa_2010_braehler-stefan_android architecture.pdf, accessed Nov 14, 2010, Outubro 2010.

[16] Android, www.android.com. Accessed in February, 2013.

[17] Google Android, Ricardo R. Lecheta, 2a Edição, Novatec, Junho/2010.

[18] Professional Android Application Development, Reto Meier, Wiley Publishing, Inc., 2009.

[19] How many lines of code does it take to create the Android OS? http://www.gubatron.com/blog/2010/05/23/how-many-lines-of-code-does-it-take-to-create-the-android-os/. Accessed in February 2013.

[20] Dalvik, code.google.com/p/dalvik. Accessed in February 2013

[21] David Ehringer. The dalvik virtual machine architecture. Technical report, Google, March 2010.

[22] Android Application Development, Rick Rogers et al, O'Reilly, 2009.

[23] N. Nyman, "Using monkey test tools," Software Testing and Quality Engineering magazine, vol. 29, no. 2, pp. 18–21, 2000.

[24] A.-D. Schmidt, H.-G. Schmidt, L. Batyuk, J. H. Clausen, S. A. Camtepe, S. Albayrak, and C. Yildizli. Smartphone malware evolution revisited: Android next target? In Proceedings of the 4th IEEE International Conference on Malicious and Unwanted Software (Malware 2009), pp. 1–7. IEEE, 2009.

[25] T. Bläsing, L. Batyuk, and A. Schmidt. An Android Application Sandbox System for Suspicious Software Detection. In 5th International Conference on Malicious and Unwanted Software, Berlin, Germany, 2010.

[26] T. Takala, and M. Katara. Experiences of System-Level Model-Based GUI Testing of an Android Application. In Fourth IEEE International Conference on Software Testing, Verification and Validation, Finland, 2011.

[27] Kaasila, J. Ferreira, D. Kostakos, V & Ojala, T (2012). Testdroid: automated remote UI testing on Android. Proceedings of the 11th International Conference on Mobile and Ubiquitous Multimedia – MUM '12: Art. 28.

[28] Robotium, 2010. It's like Selenium, but for Android. Retrieved on 19th January, 2012 from http://code.google.com/p/robotium/.

[29] T. Mendhe, P. Kamble and A. Thakre, "Survey on Security, Storage, and Networking of Cloud Computing", International Journal on Computer Science and Engineering (IJCSE), vol. 4, no. 11, (2012) November, ISSN : 0975-3397.

[30] R. Byyya, C. Yeo, S. Venugopal, J. Broberg, and I. Brandic. Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing as the 5th Utility. Future Generation of Computer Systems. vol. 25. no. 6. pp. 599- 616. 2009.

[31] L. Jain and S. Bhardwaj, "Enterprise Cloud Computing: Key Considerations for Adoption" International Journal of Engineering and Information Technology Vol 2 , (2010). IJEIT 2010, 2(2), 113-117 ISSN 0976-0253 (Online).

[32] M. Fowler, UML Distilled. Addison-Wesley, 1997.

[33] A. Leff, and J. Rayfield. "Web-Application Development Using the Model-View-Controller Design Pattern," Proceedings of the 5th IEEE Enterprise Distributed Object Computing Conference, 2001, pp. 118-124.

[34] K. Jackson, L. Ramakrishnan, K. Muriki, S. Canon, et al. Performance Analysis of High Performance Computing Applications on the Amazon Web Services Cloud. In CloudCom, pp. 159–168. IEEE, 2010.

[35] W. Vogels. A Head in the Clouds—The Power of Infrastructure as a Service. In First workshop on Cloud Computing and in Applications (CCA '08), October 2008.

[36] Motorola Solutions Developer, developer.motorolasolutions.com. Accessed in October 2013.

[37] LG Developer, developer.lge.com. Accessed in October 2013.

[38] ASP.NET MVC 4, http://www.asp.net/mvc/mvc4. Accessed in February 2013.

[39] M. Grechanik, Q. Xie, and C. Fu, "Creating GUI testing tools using accessibility technologies," in Proc. IEEE International Conference on Software Testing, Verification, and Validation Workshops. Washington, DC, USA: IEEE Computer Society, 2009, pp. 243–250.

 ISBN: 978-1-61208-304-9