IPSJ SIG Technical Report Vol.2013-HPC-138 No.27
2013/2/22

StarCloud: Optimizing a Testing Framework for Android Development

JARED RAVETCH[†1] KENTO AIDA[†2†1]

Abstract: Smartphone development is changing at rate that is becoming increasingly difficult for application developers to develop and test their applications on all relevant versions of the smartphone OS. With the Android OS, we have the unique ability to run the Android Emulator, which creates a virtual device, on either our workstation or cloud environment. We present StarCloud, an Android testing framework that emulates a cloud of connected and distributed Android devices utilizing both public Amazon's Elastic Compute Cloud (EC2) and private OpenStack cloud infrastructure. Using the BOINC distributed computing client for Android, we were able to exemplify a unique use case for StarCloud to further research in grid computing on Android devices.

Keywords: Android, Dalvik VM, EC2, OpenStack, Distributed Computing

1. Introduction

The smartphone has revolutionized communication and the way humans interact with one another. Development and testing for the smartphone, specifically Google's Android[1] platform, is rife with challenge due to the inability to simulate how the application will behave in a distributed environment.

As the Android OS is regularly updated beginning in version 1.5 (Cupcake)[2] to the current version 4.2 (Jelly Bean)[2], forces developers to test their applications on all iterations, which has no easy solution. What has come out of the extremely fast paced evolution of smartphones, specifically Android devices, is the need to continuously keep revising and testing applications. Each subsequent iteration of the Android OS improves upon the last, adding new features, new APIs, and better performance. A developer must be aware of changes in each Android OS. Thus running multiple versions of the OS simultaneously to test one's application is essential to the development and testing process. Moreover, there is no easily accessible sandbox environment for application developers to test the security and performance implications of their application in the cloud.

As smartphone applications on the Android operating system become more and more complex, utilizing distributed computing technology[3] and peer-to-peer networking[4], it is becoming ever more necessary to evaluate these applications on multiple interconnected devices. However, evaluating the efficacy of a distributed computing application for an Android device is cumbersome when using one's own workstation or cost prohibitive purchasing multiple devices. Utilizing the Android Virtual Device (AVD)[5], bundled with the Android SDK[5] is one way to test one's application without the use of physical devices. However, in order to run multiple AVDs much more powerful processing power is required in a cluster type setup.

With public cloud computing resources becoming easier and more inexpensive to use, specifically Amazon Web Services (AWS)[6], the possibility of creating such an environment becomes more of a reality. In addition, with the help of the OpenStack[7] project, creating one's own private cloud on physical servers that can mirror a similar type setup in AWS becomes a powerful tool in application testing. The advantages of OpenStack in this instance allow for more fine-grained control and operation of multiple virtual machines as well as a fully customizable network stack[8].

Deploying AVDs in public/private clouds would give us one solution to tackle the problem of software testing. A developer is able to run multiple AVDs on cloud computing resources and test the application with multiple versions of the Android OS simultaneously. The developer can also evaluate distributed computing applications running on multiple AVDs in the cloud. However, setting up this distributed testing framework for Android Applications on cloud computing resources is not an easy task. This requires a deep understanding of cloud computing technologies and architecture by the developer, e.g. configuration, deployment and software installation. To the best of our knowledge, there is no solution to automatically setup a testing framework on pubic/private clouds.

With smartphones continuously increasing in performance[9], it begs the question for researchers whether or not they are now suitable for distributed computing tasks. The BOINC project[10] is a particularly popular distributed computing effort that aims to consolidate and make it easier for researchers to create and maintain distributed computing projects. This is a very difficult question to answer as it involves testing the BOINC computing client on multiple versions of Android and seeing how well the client performs. One approach to this question is to use virtual devices to get an idea if smartphones can perform such tasks and how to optimize the client to best suit the device.

This combination of factors, from difficulty to testing applications on multiple versions of Android to testing large numbers of devices together in a distributed environment that led

†1Tokyo Institute of Technology
†2 National Institute of Informatics

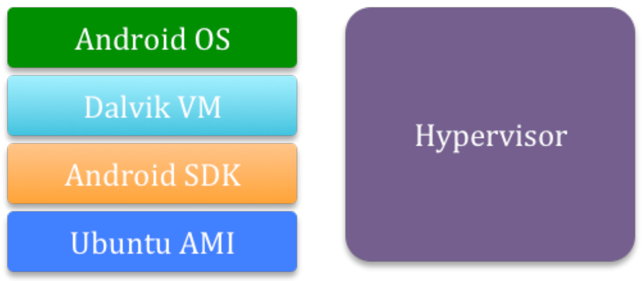


Figure 1. StarCloud Node Application Stack

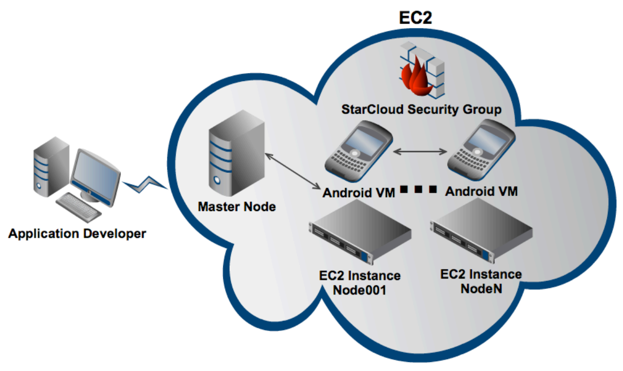
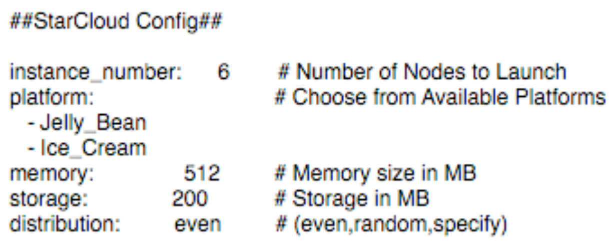


Figure 2. StarCloud Architecture in EC2



```
##StarCloud Config##

instance_number:   6      # Number of Nodes to Launch
platform:                 # Choose from Available Platforms
  - Jelly_Bean
  - Ice_Cream
memory:           512     # Memory size in MB
storage:          200     # Storage in MB
distribution:     even    # (even,random,specify)
```

Figure 3. StarCloud Configuration File

us to develop a new testing framework. This paper presents StarCloud, a testing framework for the development of Android applications. StarCloud allows developers and researchers alike to create a cloud of distributed virtual devices on public or private cloud infrastructure and test applications in an environment that more closely mirrors actual usage.

The rest of this paper is organized as follows: Section 2 briefly discusses related work. Section 3 presents our implementation of StarCloud, and Section 4 shows our experimental results. Section 5 summarizes our contribution and outlines future work.

2. Background

The concept of StarCloud grew from the existing tool StarCluster[11], which gives anyone the ability to launch a cluster of EC2 instances, using AMIs specifically customized for scientific research. In order to run the Android OS in the cloud, we referred to the work done by Byung-Gon Chun et al. that created CloneCloud[12], which elastically offloads computation from the Android device to EC2[7]. A snapshot of the running Android OS is used as a template to launch instances running the Android OS in EC2.

3. Architecture and Implementation

As discussed in the introduction, utilizing public and private cloud architecture is essential to this research as it enables the developer to employ commonly available resources, specifically AWS and OpenStack. The overall functionality of StarCloud does not differ when setup in either EC2 or OpenStack, as the Eucalyptus[13] API used to provision the instances is the same. However, additional time must be spent if an OpenStack cluster is not already setup, in order to have StarCloud at working capacity.

EC2 was chosen as the cloud infrastructure platform due to its full-featured API and seamless ability to scale. EC2 has been utilized in a multitude of research and high performance computing projects[14].

3.1 Middleware Architecture

StarCloud acts as a middleware between a user (or application developer) and EC2 or OpenStack allowing the user to launch a cloud of customized instances; each instance running the Android OS utilizes the Dalvik Virtual Machine[15]. Figure 1 shows the software stack of StarCloud. A security group is created to allow each device to seamlessly communicate with one another and externally. A master instance is also launched, in order for the user to interact with the virtual Android device cloud to deploy applications, collect device logs and performance data as illustrated in Figure 2.

At the heart of StarCloud is the ability to communicate with cloud architecture to provision any number of required instances. StarCloud uses the APIs available for both AWS and OpenStack, to launch instances, each having the required packages to run the Android SDK. We decided early on that each instance will be running only one AVD, as the required CPU and memory is significant enough to segregate each AVD into a separate instance. Moreover, this allows for better networking performance, as each AVD will be using the networking stack on each instance rather then having to share with other AVDs on the same instance. The instance itself is running Ubuntu 12.04 LTS, with all packages required to run the Android SDK and be able to display the emulator in a UI.

One challenge we faced was how to display the emulator to the user, as AWS and OpenStack instances do not have an X Window System[16]. We decided to use VNC [17] to output the emulator's UI so that the user can visually see the AVD in action.

3.2 Configuration

A configuration file in standard YAML[18] format (Figure 3) is used to specify the number of instances, the Android

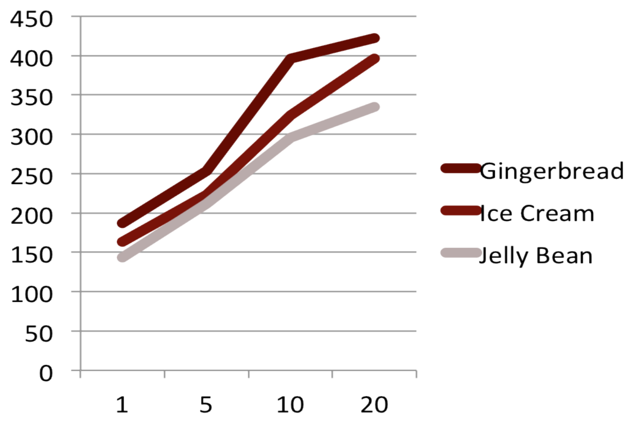


Figure 4. Start Time in Seconds of Nodes. X-axis is number of VMs and Y-axis is time in seconds

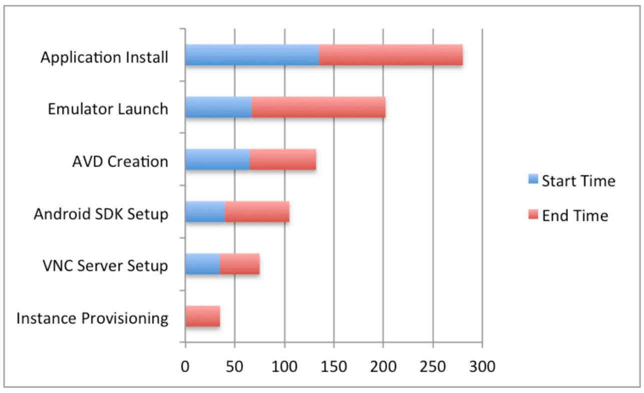


Figure 5. Breakdown of Launch Time for Each Node. X-axis is time in seconds

platform(s) to launch, how much memory and storage should each AVD allocate and what the platform distribution should look like, i.e. launch half of the instances with Jelly Bean[2] and the other half with Ice Cream Sandwich[2]. Once the machine is launched, the Android SDK for Linux is downloaded and uncompressed into an accessible directory.

Next, StarCloud uses the Android SDK to update the list of available Android platforms and System Images[2]. From this list, the specified Android platform and corresponding System Image is downloaded and installed into the SDK directory.

From here, StarCloud is now able to create the necessary AVD as requested initially in the configuration file. When the AVD is completed, the VNC server is setup and the AVD is launched in the VNC session. After booting the AVD, StarCloud uses the Android Debugger[5] to install the required application(s) for testing. When testing is complete, the instances are destroyed

4. Experiment

There are two parts to our experimentation. The first part focuses on launching the Android emulator on three different platforms: Gingerbread, Ice Cream Sandwich and Jellybean and measuring the time taken to complete this task. The second is providing a use case for developers and researching of StarCloud. This involves choosing an application that will test the performance of different versions of Android in a distributed environment. For our server environment we choose m1.small instances in EC2 running Ubuntu 12.04 LTS. Small instances offer a good balance of processor and memory, without worrying about overloading the server. As we are only running one AVD per instance, there was no performance gain by using larger instances, as each AVD occupies one CPU core and allocated memory as defined in the configuration file.

4.1 Launching the Android Emulator

Figure 4 presents the time to launch Android Emulators. The results show that the time for launching Android Emulators linearly increases following the number of emulators launched. What's interesting to point out is that the Gingerbread platform is significantly slower then Ice Cream or Jelly Bean. This fact can be contributed to the time spent in the initial boot and application installation phases. Moreover, the launch times are decreasing as we progress from each platform iteration and thus hope to see continued launch improvements in latter installments of Android.

As discussed in the implementation section, there are many phases to launching the emulator in the cloud. Figure 5 is a breakdown of the phases and respective average time taken for each phase. As one can see from the chart, the most expensive phases are 1) the launching of the emulator, i.e. booting the AVD until the Android OS is fully operational and 2) Android application installation. The instance provisioning or starting the virtual machine on EC2 step is fairly consistent when launching larger number of instances, as each API call to create a server is done in parallel. However, in order to launch the emulator, the server environment must be first completely setup.

As we are using a virtual environment, we do not have the ability to use hardware virtualization as would be available when using physical hardware. Android has made great strides in improving the performance of the emulator by releasing a hardware virtualization of the system image using Intel's Atom x86 processor[19]. In EC2 or OpenStack, we do not have access to the hardware virtualization abilities of the CPU, thus must use the software virtualized ARM[20] processor. This unfortunately greatly increases the time spent starting the emulator and Android OS, as can be seen in (Figure 4). Start time and End time in this case refer to the initial API call to provision the EC2 instance and until the application is fully installed and ready for user input.

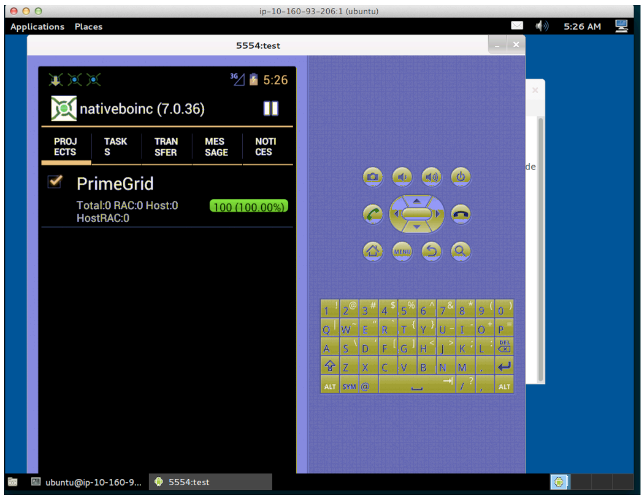


Figure 6. Android Emulator Running in EC2

BOINC Version	CPU	Android OS	Average upload rate	Average download rate	Measured floating point speed
7.0.36	ARMv7 Processor rev 0 (v7l)	Jelly Bean	5.58 KB/sec	196.25 KB/sec	1000 million ops/sec
7.0.36	ARMv7 Processor rev 0 (v7l)	Ice Cream Sandwich	2.27 KB/sec	170.22 KB/sec	1000 million ops/sec
7.0.36	ARM926EJ-S rev 5 (v5l)	Gingerbread	5.98 KB/sec	165.28 KB/sec	1000 million ops/sec

Table 1. List of Devices Connected to BOINC

We also noticed there were variations in the time spent when starting the emulator between the versions of Android we were testing. The start time can be seen to decrease as we go from Gingerbread to Jelly Bean. This is most likely due to the optimization each subsequent Android version has made to reduce boot time.

4.2 Use Case with BOINC

As there are a myriad of applications for StarCloud, from development sandboxing to distributed computing. We will focus on the latter, as smartphones are becoming increasingly more powerful and versatile. One very popular system that allows individuals and organizations to participate in public-resource distributed computing projects is BOINC[10] (Berkeley Open Infrastructure for Network Computing). What makes StarCloud an extremely useful tool is that it allows for the creation of individual BOINC clients on various Android versions and emulated hardware. This enables researchers who are developing clients for Android devices to test their software in a more real-world environment.

For the second phase of the experiment, we focused on running the BOINC client on the Android Virtual Device. There is already an existing application that will install and run BOINC on Android called NativeBOINC[21]. We followed the same steps as described in the first phase of the experiment and adding a last step of installing an application and starting this application. Once NativeBOINC is started, it must first download and install the BOINC client. This phase also takes a significant amount of time, as the client is downloaded, verified, unpacked and installed. Finally, once the BOINC client is installed, the client must then be synchronized with the BOINC Account Manager (BAM), which requires an account to already be setup. Figure 6 shows a screenshot of the VNC session with the BOINC client running in the emulator on an EC2 instance.

For this experiment, we choose the PrimeGrid project[22]. An account on BAM was created and then added the PrimeGrid project to the account. After the BAM credentials were entered into the client, the client synchronizes with the PrimeGrid project and begins running benchmarking tasks to determine the environment the client is being run. These results are uploaded to the PrimeGrid project and are viewable on the project's site. Table 1 shows the results of the benchmarking tests run. We can see slight differences in the networking download speed of each platform, with Gingerbread being the slowest and Jelly Bean being the fastest. Floating-point calculations per second were all the same for each platform.

The very fact that we can 1) launch multiple versions of Android on cloud infrastructure and 2) run the BOINC client on these virtual devices and gather benchmarking results shows that StarCloud did indeed succeed in it's goal to test distributed computing applications in a cloud environment. Even though the overall performance of the virtual devices running the BOINC client was suboptimal, it lays the groundwork for further testing and optimization of StarCloud.

5. Conclusion

In this paper we present the novel StarCloud middleware to aid application developers in testing their applications in a virtual cloud of Android devices. We have shown that StarCloud can be utilized in both public and private cloud infrastructure. StarCloud can be a vital tool for researchers and developers alike that are interested in testing their applications in scenarios where more connected devices in a cloud are required. Moreover, distributed computing on smartphones is becoming a reality now and with the help of StarCloud, highly tuned applications can be developed and tested to meet the various needs for Android platforms and hardware specifications. Future work will focus on improving the launch times of the emulators and VNC sessions to allow developers and researchers to focus their time on application testing and performance monitoring.

Acknowledgments This work was supported in part by JSPS KAKENHI Grant Number 24240006.

References

[1] Burnette, Ed. Hello, Android: introducing Google's mobile development platform. Pragmatic Bookshelf, 2009.

[2] Google Android. http://developer.android.com/

[3] Chou, Wu, and Li Li. "WIPdroid–a two-way web services and real-time communication enabled mobile computing platform for distributed services computing." *Services Computing, 2008. SCC'08. IEEE International Conference on*. Vol. 2. IEEE, 2008.

[4] Ughetti, Marco, Tiziana Trucco, and Danilo Gotta. "Development of agent-based, peer-to-peer mobile applications on ANDROID with JADE." *Mobile Ubiquitous Computing, Systems, Services and Technologies, 2008. UBICOMM'08. The Second International Conference on*. IEEE, 2008.

[5] Google Android SDK. http://developer.android.com/sdk/index.html

[6] Amazon Web Services (AWS). http://aws.amazon.com/

[7] Open source software for building private and public clouds. http://www.openstack.org

[8] OpenStack Quantum. http://wiki.openstack.org/Quantum

[9] Borkar, Shekhar, and Andrew A. Chien. "The future of microprocessors."*Communications of the ACM* 54.5 (2011): 67-77.

[10] Anderson, David P. "BOINC: A system for public-resource computing and storage." *Grid Computing, 2004. Proceedings. Fifth IEEE/ACM International Workshop on*. IEEE, 2004.

[11] StarCluster. http://star.mit.edu/cluster/

[12] Chun, Byung-Gon, and Petros Maniatis. "Augmented smartphone applications through clone cloud execution." *Proc. of the 8th Workshop on Hot Topics in Operating Systems (HotOS), Monte Verita, Switzerland*. 2009.

[13] Nurmi, Daniel, et al. "The eucalyptus open-source cloud-computing system."Cluster Computing and the Grid, 2009. CCGRID'09. 9th IEEE/ACM International Symposium on. IEEE, 2009.

[14] High Performance Computing on AWS. http://aws.amazon.com/hpc-applications/.

[15] Bornstein, Dan. "Dalvik vm internals." *Google I/O Developer Conference*. Vol. 23. 2008.

[16] Scheifler, Robert W., and Jim Gettys. "The X window system." *ACM Transactions on Graphics (TOG)* 5.2 (1986): 79-109.

[17] Richardson, Tristan, et al. "Virtual network computing." *Internet Computing, IEEE* 2.1 (1998): 33-38.

[18] Ben-Kiki, Oren, Clark Evans, and Brian Ingerson. "YAML Ain't Markup Language (YAML™) Version 1.1." *Working Draft 2008-05* 11 (2001).

[19] Intel Atom x86 Image for Android. http://software.intel.com/en-us/articles/intel-atom-x86-image-for-android-ice-cream-sandwich-installation-instructions-recommended

[20] http://developer.android.com/tools/devices/emulator.html

[21] NativeBOINC. http://nativeboinc.org/.

[22] PrimeGrid. http://www.primegrid.com/.