Java(Script) Application Performance on Android

Wybren Kortstra
University of Twente
P.O. Box 217, 7500AE Enschede
The Netherlands
w.kortstra@student.utwente.nl

ABSTRACT

JavaScript is becoming increasingly popular as a language for the implementation of web applications. However, these applications are expected to perform worse than Java programs on Android devices. Depending on how powerful the device is and the tasks the application has to perform we can perceive the performance difference. The choice for developers seems to be simple, namely to implement only Java applications. However, the cost of developing a JavaScript application is much lower compared to the cost of developing a Java application. This paper describes a tool that can measure the performance difference between Java and JavaScript applications for Android. This paper shows that JavaScript applications do not perform much worse than Java applications and that front-end frameworks for mobile web applications still have opportunities to improve.

Keywords

Java, JavaScript, Native, Hybrid, Android, Benchmarking

1. INTRODUCTION

JavaScript or hybrid applications are becoming increasingly popular, but JavaScript is an interpreted programming language, so JavaScript applications tend to be slower when compared to Java applications (native) [11]. Furthermore, web applications run inside a webview on mobile platforms, which is an additional layer between the application and the hardware, causing additional delays.

Hybrid applications consist of HTML, JavaScript and an interfacing component to interact with the underlying hardware. A popular interfacing component is Apache Cordova [4].

Hybrid applications are quicker and cheaper to develop than native applications [10]. Since it is possible to develop an application for both iOS and Android at the same time. Another reason they are cheaper is because there are more developers for JavaScript than for Java [1].

Therefore when developing an application one needs to decide to apply the native or the hybrid approach. In this research we created a benchmarking tool to test the performance difference between JavaScript and Java applications on mobile devices. The tool can help developers decide between native and hybrid application development.

Our benchmarking tool tests how quick an application starts up and can load views, this is the first action taken by an application, which is the first impression on the user which is important for the perception of performance. This tool also measures the delay of the JavaScript interface, which is the delay introduced when calling native functions from a hybrid application. Hybrid applications do this often, when using a camera or the filesystem for instance.

Most applications focus on displaying information to the user. For instance, banking applications show the balance or messages from the bank. Facebook, Linked-In and others display messages from other users to the user. All these applications use the Internet to retrieve information. Most of the time information is displayed in a scrollable list.

Our results show that native applications are faster than hybrid applications. However, the difference was not that big and depends on how modern the smartphone is. Hardware seems to be an important factor alongside with the Android version. Other differences lay in the way the user interface is operated. Android defined a standard for user interfaces and this is only partly implemented by hybrid frameworks.

The paper is further structured as follows: section 2 discusses the problem and elaborates on the research questions and then explains how we tackled the problem, section 3 discusses how we built the tool and explains the decision for the tests we used, section 4 explains their implementation, section 5 shows the results, section 6 discusses related work, section 7 explains the results and conclusions and in section 8 we discuss future improvements.

2. PROBLEM ANALYSIS

The first Facebook mobile application was a hybrid application. In 2012 Facebook moved to a native application because the hybrid application did not perform well enough [22].

There is an ongoing debate between mobile application developers whether to build hybrid or native applications. There is no objective answer to the question and the answer depends on the needs of the application and the device the application runs on. Still then, no benchmark is available that tells if your device would be able to give the performance required by an application.

In this paper we created a benchmarking tool to compare the performance of hybrid application against the performance of native applications.

24th Twente Student Conference on IT January 22nd, 2016, Enschede, The Netherlands.

2.1 Background

A **web application** is a mobile web application that is designed to execute in the web browser of the mobile device. It is built using only HTML, CSS and JavaScript [17].

A **native application** (native app) is an application program that has been developed for use on a particular platform or device. In this research a native application is an application developed in Java [19].

Hybrid applications apply both web and native methodology [17]. It uses the webview of the device while at the same time it can access native APIs with the use of an interfacing component like Phonegap [16] or Cordova [4].

All user interface elements in an Android app are built using **View** and **ViewGroup** objects. A View is an object that draws something on the screen with which the user can interact [6]. When referring to a view in this paper, we mean the visible part of the screen with the ViewGroup and View objects.

The most common navigation structures in mobile applications are the **navigation drawer** and the **tabbed navigation**. The navigation drawer is a button in the upper left or right corner that can be pressed which will then show a vertical menu on the left or right side of the screen. The other navigation structure consists of tabs at the top or bottom of the screen that are visible at all times.

2.2 Research goal

The main goal of this research is to compare the performance of hybrid Android applications with respect to native Android applications. Our research question has been formulated as follows: What is the performance difference between Java and JavaScript applications for Android devices and under what circumstances?

This question is divided into these subquestions. Answering these subquestions should give an answer to the main research question.

1. Which parts of an application should be taken into account when measuring the performance? For instance, we could test the start up time or the network delays.
2. How to measure the performance of the different parts in an application?
3. What is the difference in performance of using different devices and Android versions?

To help answer the questions above we built a benchmarking tool. This tool automatically tests the hybrid performance with respect to the native performance. The performance measured by this benchmarking tool is represented as a score. The score has been determined by comparing hybrid applications and native applications. Not all Android devices have the same hardware nor do all Android devices have the same Android version, and both the Android version and the hardware impact the score.

3. TOOL ARCHITECTURE

We built a benchmarking tool to compare the performance of JavaScript applications with Java applications. The tool measures the performance by measuring the time it takes to complete a task in both the native and the hybrid interface. The time difference is used as the performance measurement. This can be justified because this is the time the end-user notices when using the application.

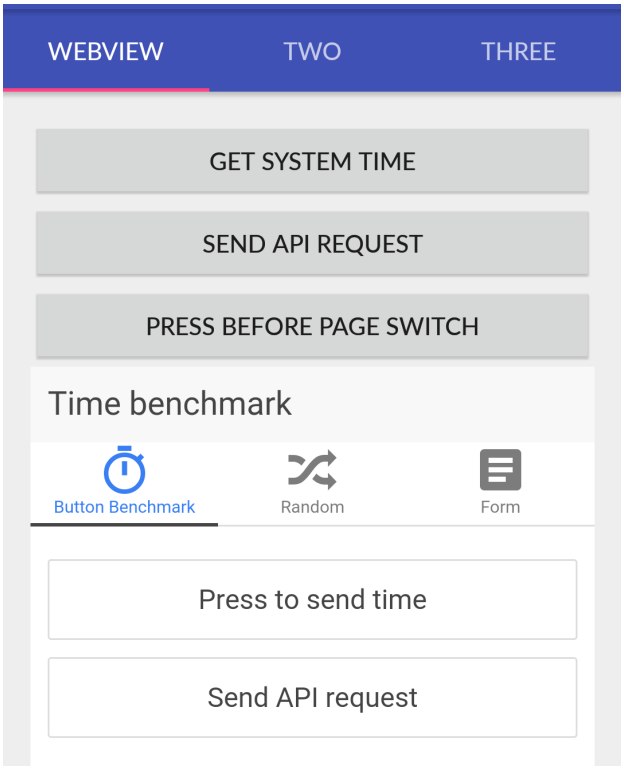

Figure 1. Interface of the benchmarking tool

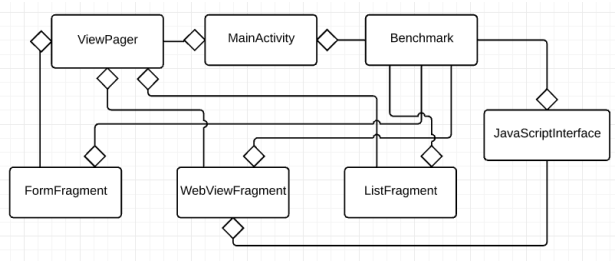

Figure 2. Interface of the benchmarking tool

3.1 Interface

We used tabs for the navigation. This choice is favoured over a navigation drawer, because it is considered as better practise according to UX-designers for a small number of menu items [18]. The tool we created has three different tabs. On the first tab there are three buttons and a WebView. The second tab contains a form and the third tab contains a list. The WebView contains an interface similar to the interface of the tool. This means it has also a tabbed interface with three tabs. Figure 1 shows the tool interface.

3.2 Tool Structure

Figure 2 shows the class diagram of our tool. The MainActivity contains a ViewPager, which contains fragments used to create the tabbed interface. The MainActivity also contains the Benchmark, which is passed through to all the Fragments so every Fragment uses the same Benchmark object. The WebViewFragment creates a JavaScriptInterface which is injected in the WebView. The JavaScriptInterface is used to relay function calls from the JavaScript in the WebView to methods in Java. The JavaScriptInterface also contains the Benchmark object.

3.3 Mobile framework

To create the mobile application with web technology a framework was used. The decision was made to use ionicframework [21]. Ionic is a powerful HTML5 SDK that helps build native-feeling mobile applications using web technologies like HTML, CSS, and Javascript. Ionicframework is growing rapidly with 50.000 new applications per month and having 320.000 applications at the end of 2014 [12]. As ionicframework is quite popular we considered that it is a suitable framework to use in our benchmark.

3.4 Monkeyrunner

Input must be simulated to prevent the user from influencing the benchmark results. Monkeyrunner [9] is a tool to control the Android device from outside the Android code. Monkeyrunner allows scripts to be written that run the benchmarking tool by sending keystrokes to it. Monkeyrunner can also install the application and clear the application cache.

3.5 Benchmark tests

The first subquestion is to determine which application tasks we need to test with the benchmark. We examined other Android applications and identified the following basic actions:

1. Loading first view
2. Moving to another view
3. Loading data from the Internet
4. Scrolling through a list
5. Calling the native interface

An application can be used once the interface is built and displayed to the user. The first test we identified is the time to load the first view.

Applications often offer the following options to their users: click on a button to do some action, load data from the Internet, scroll through a list of data or go to a different view. The second test we identified is to go to a different view.

The third test will be loading data from the Internet. Often data is loaded in applications. Examples are social media application like Facebook and Twitter or news applications like Google News or BBC news.

The data loaded from the Internet is often displayed to the user in a list. The fourth test we identified is to measure the performance of scrolling through a list of items.

Finally we test the delay cause by the JavaScript Interface, which is used to call native Android functions, for example to call the camera or bring up the keyboard.

3.6 Benchmarking Score

The score measured by the benchmark is determined by the performance of the JavaScript application with respect to the Java application. We also took the absolute performance into consideration. For example, if the JavaScript application should start up in 10 seconds and the Java application in 5 seconds then the Java application is 100% faster. However, this is also true if the JavaScript application should start up in 1 seconds and the Java application in 0.5 seconds, but 1 second is much better than 10 seconds. So to calculate the benchmark score we used the relative difference in performance as a percentage and multiplied this by the absolute difference in performance in seconds.

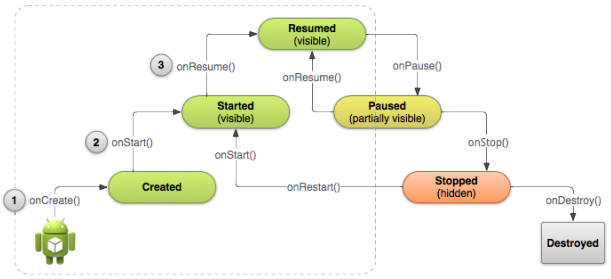

Figure 3. Android life cycle

4. TESTING STRATEGY

The performance of each task is tested by measuring how long the task takes to complete. To measure this we read the system time at the beginning of an action and when the action has finished we read the system time again. The difference between those two time moments is the time taken by the action.

4.1 Start up

An Android application does not start with a main method, but an Activity instance is invoked with a few specific callbacks. The first callback is the *onCreate* method, which is called to do basic application start up logic. Next, when the application is displayed to the user, the *onStart* method is called. Figure 3 shows the full life cycle of an Android application [7]. To measure the start up time we read the start time on the *onCreate* method and read the end time on the *onStart* method.

In a Cordova compiled hybrid application the WebView is the first thing that is loaded and the HTML, CSS and JavaScript code is loaded instantly. However, our tool first needs to start before the WebView can load itself, which happens while the native view is created. The WebView calls the *onPageStarted* method when it starts loading the page and then the start time is read. When the WebView is ready and the page or view is loaded the *onPageFinished* method is called and the end time is read.

4.2 Change View

The time between the start and end of a view change is measured from the moment the navigation button is touched until the moment the view is loaded. Since it is not possible to set an *onClickListener* on the generated tab button, we simulate the start by two touches at nearly the same time, one for touching a button to read the start time, and another to navigate to another tab. When the *onPageSelected* method is called from the viewPager the end time is read.

The time to change the view in a WebView is measured in a similar way: when touching the navigation button the start time is read and when the view is loaded the *ionicView.enter* event fires and the end time is read.

4.3 Network

The goal of this test is not to test network speed or latency, which is independent of the Java and JavaScript application. All the network calls are handled by the underlying Android operating system, so we measure if there is any additional delay caused by the WebView.

The speed is measured on calls made to an API to retrieve information. The moment the request is sent the start time is read, and when the response arrives the end time is read.

Table 1. Device specifications

	Samsung	Huawei
CPU	Quad-core 2.3 GHz Krait 400	Dual-core 1.2 GHz Cortex-A5
Memory	3 GB RAM	512 MB RAM
Android	4.4 (KitKat)	4.1.1 (Jelly Bean)

4.4 Scrolling List

To measure how smooth one can scroll through a list of items we measured the number of frames per second, which is determined by the amount of display refreshes. More frames per second means that one can scroll smoother through a list of items.

4.5 JavaScript Interface

We measure the additional delay needed to relay a call from the JavaScript to the Java code by pressing a button in the tool. When this button is pressed the start time is read, then exactly one second later a button is pressed in the WebView, which calls a function on the JavaScript interface, which forwards the call to Java code. At this point the end time is read. If there were no delay, the time difference would be exactly one second, if the time difference is bigger than one second, a delay is experienced.

5. RESULTS

We used the benchmarking tool on two devices: a Samsung Note 10.1 P605 and a Huewei Acend Mate G510. The specifications of the devices can be found in Table 1. The Samsung uses a newer version of Android and has better hardware compared to the Huawei.

Due to the limited time we had to complete the research, we have not been able to design a representative test to measure the performance of lists. We found that there are applications that are capable of measuring the number of frames per second [8], however we could not reproduce this or read the number of frames per second from this application.

Figure 4 shows the results of the benchmarking tool in terms of the number of seconds it took to complete an action. Figure 5 shows the score as it is calculated by the benchmarking tool. Notice that the score for the network test is so low that it is not visible in this chart.

The start up of a JavaScript application is the largest performance difference we measured. Compared to the native start up time, the WebView takes about 4 times as much time to start. We also see that the performance difference on older devices is larger than on newer devices. The time it takes to change a view in JavaScript applications is almost 0.2 seconds on older devices compared to the 0.1 seconds on newer devices. In Java applications, regardless of the device, it takes about 0.015 seconds.

6. RELATED WORK

There are benchmarking tools for mobile CPU and GPU performance such as, AnTuTu [20], but not for testing or comparing application performance. The study, conducted by Brinkheden [3] is quite similar to ours and focus on the low level API. Brinkheden concluded that hybrid applications use about 8.5 times more storage space, use about 25% more memory and execute code slower in most cases. This research gave an indication of what can be expected from our benchmark, namely that hybrid applications are slower.

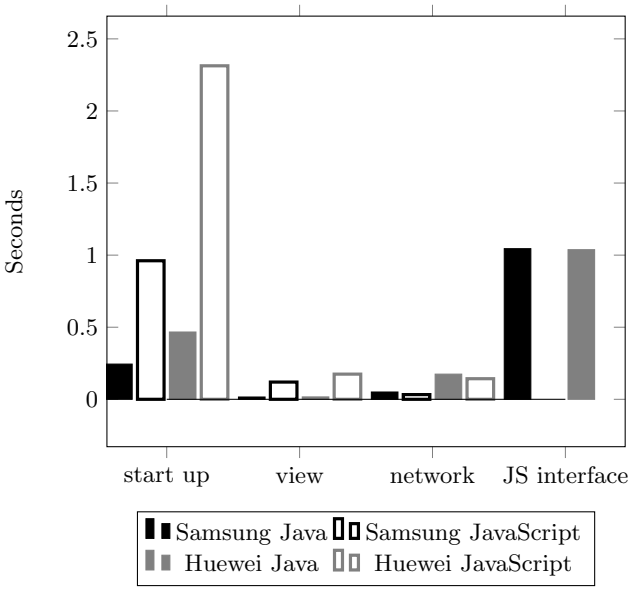

Figure 4. Benchmark results (lower is better)

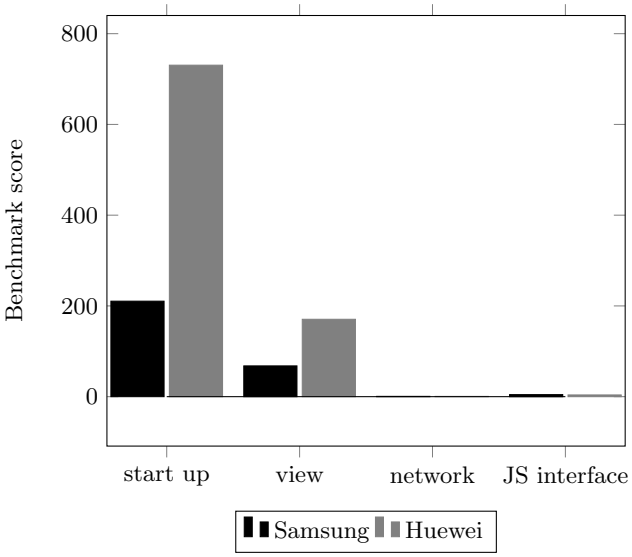

Figure 5. Benchmark score (lower is better)

A few years ago hybrid applications were worse in performance compared to today [14]. JavaScript has gained a lot in performance due to software and hardware improvements. Furthermore native scroll events were not supported and had to be imitated by JavaScript [15]. According to Sencha Michael Mullany [14] JavaScript performance on Android increased progressively between 2009 and 2013. The Sunspider benchmark shows a 4x performance improvementa and the DOM interactions tested by the Dromaeo benchmark show a 3.5x performance improvement [14]. Sunspider [2] and Dromaeo [13] are JavaScript benchmarking tools.

7. CONCLUSION

Reflecting on our research questions we conclude that Java applications have better performance than JavaScript applications, but this performance difference is not big. We cannot explain why JavaScript applications perform better than Java applications on the network test. To find an

explanation we would have to dig deeper into the application structure.

The performance difference on newer devices is smaller than on older devices, which could be because since Android 4.4 a new WebView was introduced, which brought a huge performance improvement [5]. Another reason is most likely the hardware difference between the devices. The Samsung device we used has more modern hardware than the Huawei device we used. However, even though the results do not show large differences in performance we should keep in mind that we only tested 4 basic operations. More comprehensive tests could give other results.

We conclude that one can just as well choose to develop a simple application using JavaScript as with Java. The difference in performance on newer devices is relatively small. We expect the performance of the newer devices to increase, making the performance difference between Java and JavaScript even smaller.

8. FUTURE WORK

Our benchmarking tool only contains 4 basic tests. The benchmarking tool could be extended with tests to measure the performance of list scrolling and user-interface animations.

For a better comparison of Java and JavaScript application we should also take a look at the user-interface frameworks. We could compare the way the user interacts with the Java and JavaScript application. The way the user interacts with an application also affects the user experience.

9. REFERENCES

[1] 2015 Developer Survey. http://stackoverflow.com/research/developer-survey-2015/, 2015. [Online; accessed 07-October-2015].

[2] Apple. SunSpider 1.0.2 JavaScript Benchmark. https://www.webkit.org/perf/sunspider/sunspider.html, 2013. [Online; accessed 21-October-2015].

[3] D. Brinkheden and R. Andersson. A performance study of hybrid mobile applications compared to native applications, 2015.

[4] A. Cordova. Apache Cordova. http://cordova.apache.org/, 2015. [Online; accessed 19-October-2015].

[5] A. Developer. Migrating to WebView in Android 4.4. https://developer.android.com/guide/webapps/migrating.html. [Online; accessed 09-January-2016].

[6] A. Developer. UI Overview. http://developer.android.com/guide/topics/ui/overview.html. [Online; accessed 10-December-2015].

[7] A. Developer. Starting an Activity. http://developer.android.com/training/basics/activity-lifecycle/starting.html, 2015. [Online; accessed 11-December-2015].

[8] EasyAsPieApps. FPS Meter. http://fpsmeter.github.io/, 2015. [Online; accessed 09-January-2016].

[9] R. H. KANAGOUDRA and D. H. R. ARADHYA. Comparative study of bluetooth testing automation tools on android. 2014.

[10] B. Kohan. Apple iPhone iOS App Development, Android Apps Cost Estimate / Average Pricing. http://www.comentum.com/mobile-app-development-cost.html, 2015. [Online; accessed 19-October-2015].

[11] B. Kohan. Native vs Hybrid / PhoneGap App Development Comparison. http://www.comentum.com/phonegap-vs-native-app-development.html, 2015. [Online; accessed 19-October-2015].

[12] M. Lynch. Ionic: Year One in Review. http://blog.ionic.io/ionic-one-year-review/, December 2014. [Online; accessed 13-December-2015].

[13] Mozilla. Dromaeo. https://wiki.mozilla.org/Dromaeo, August 2010. [Online; accessed 30-December-2015].

[14] M. Mullany. 5 Myths About Mobile Web Performance. https://www.sencha.com/blog/5-myths-about-mobile-web-performance-2/, August 2013. [Online; accessed 30-December-2015].

[15] Perry. Native Scrolling in Ionic: A Tale in Rhyme. http://blog.ionic.io/native-scrolling-in-ionic-a-tale-in-rhyme/, May 2015. [Online; accessed 30-December-2015].

[16] Phonegap. About phonegap. http://phonegap.com/about/, 2015. [Online; accessed 07-October-2015].

[17] R. Raj and S. Tolety. A study on approaches to build cross-platform mobile applications and criteria to select appropriate approach. In *India Conference (INDICON), 2012 Annual IEEE*, pages 625–629, Dec 2012.

[18] A. Rose. UX designers: Side drawer navigation could be costing you half your user engagement. http://thenextweb.com/dd/2014/04/08/ux-designers-side-drawer-navigation-costing-half-user-engagement/, April 2014. [Online; accessed 11-December-2015].

[19] M. Rouse. Native app definition. http://searchsoftwarequality.techtarget.com/definition/native-application-native-app, February 2013. [Online; accessed 07-October-2015].

[20] G. P. Store. AnTuTu benchmark. https://play.google.com/store/apps/details?id=com.antutu.ABenchMark, 2015. [Online; accessed 07-October-2015].

[21] I. team. Ionicframework. http://ionicframework.com, December 2014. [Online; accessed 26-December-2015].

[22] D. H. Venturebeat. Facebook's Zuckerberg: 'The biggest mistake we've made as a company is betting on HTML5 over native.'. http://venturebeat.com/2012/09/11/facebooks-zuckerberg-the-biggest-mistake-weve-made-as-a-company-is-betting-on-html5-over-native/, September 2012. [Online; accessed 30-December-2015].