Mobile JavaScript Library Minimization

Milestone one

William Jernigan
Oregon State University
Corvallis, OR 97331
wdcjernigan@gmail.com

William Leslie
Oregon State University
Corvallis, OR 97331
lesliew@onid.oregonstate.edu

Francis Vo
Oregon State University
Corvallis, OR 97331
vof@onid.oregonstate.edu

1. RESEARCH OVERVIEW

Mobile devices load entire JavaScript libraries when users browse the web. This can make the mobile device slow or unresponsive if the library is large and/or if the mobile device's resources are limited. If a website is slow or unresponsive on a mobile device, users may look for another website to use or download an app instead of using the mobile browser. Traditional guidelines were made to improve load times and usability of JavaScript websites. JavaScript libraries are often minified to create smaller files allowing for faster download times for the files. Hybrid apps are mobile apps that use JavaScript where the files are saved on the device. This means that download times are not an issue and therefore file size is less of an issue for performance. 80% of web page loading time is attributed to loading page resources and only 20% is from back-end computations [3]. Hybrid applications do not have to load source from a server and therefore do not have that concern. We plan to show that the traditional guidelines of minifying result in an insignificant improvement. Then we will suggest new guidelines that will work better for hybrid apps.

2. TECHNICAL CHALLENGES

In order to show that the traditional web page guidelines do not apply to hybrid applications, we must use the Google Closure Compiler, KJSCompiler, or something similar to create a version of the hybrid application which is minified. Then we must test its speed with something like PageSpeed Insights or mobileOK before and after minification with the expectation that there will be miniscule improvements. We can use IBM Worklight to simulate the hybrid applications on a laptop or desktop computer. Below is a discussion of the current challenges with each required technology.

2.1 Google Closure Compiler

Google Closure Compiler is a great tool, but there are problems when compiling a large number of js files. JavaScript apps that didn't plan on using the closure compiler might have a hard time with the compiler. The closure compiler only compiles JavaScript code, and therefore the user would need to change HTML scripts and other files to work with the newly compiled files. Also as part of a website, there might be multiple pages that use the same library of files. It would be nice to have file dependencies to help create better compiled files. We found some possible solutions, Google Closure Library and KJScompiler.

2.2 KJScompiler

It is a wrapper around Google Closure Compiler that allows annotations with information about dependencies to be used. KJSCompiler compiles the JavaScript files in the order that the dependencies dictate. This allows for the correct ordering of declarations when compiling over multiple files. The downside is that you need to have dependence information in each file, and hard to know the dependence of a system you don't know.

2.3 PageSpeed Insights

PageSpeed Insights is a Chrome extension that analyzes problems with JavaScript websites. This is helpful to see which traditional guidelines are used. We will be able to see the improvements with JavaScript websites, but since it is a Chrome extension, it cannot be used to directly measure the performance on mobile devices. But we can simulate these hybrid apps with IBM Worklight as described below.

2.4 IBM Worklight

Worklight is an extension to Eclipse which helps with developing mobile applications, including mobile web apps, hybrid apps, and native apps. We can use Worklight to simulate running a hybrid app on a laptop/desktop, then use PageSpeed Insights to measure the app instead of modifying PageSpeed Insights to work on a mobile device. We will have to load all of these applications in Eclipse on a local IBM Application Server, then argue that this configuration is representative of what happens on a phone.

3. PROPOSED PROCESS

3.1 Ineffective traditional guidelines

First, we will use Google Closure Compiler to turn a hybrid app into a minified hybrid app. The closure compiler uses traditional guidelines for optimizing mobile web apps. Then we will test both the normal and minified versions of the hybrid apps. Our hypothesis is that the performance of normal and minified versions will be very similar. The traditional

guidelines are used to create an optimized javascript files that are transferred over the internet to the user device. Unlike mobile web apps, hybrid apps have the JavaScript files on the device on execution. This means that file size and other minification guidelines might not affect performance as much as web apps.

3.2 New Guidelines

Secondly, after proving that traditional guidelines are ineffective, we will test and validate that new guidelines are more effective for hybrid apps. Aharon will have some suggestions for new guidelines.

4. RELATED WORKS

4.1 Digua: Minifier and Obfuscator for Web Resources[1]

Authors Alex Ciminian and Ciprian Dobre discuss their minifier tool Digua. Digua is a minifier which works for CSS, HTML, and JavaScript. It will find correlations between these different languages to further reduce the size of these files. The paper states that 80% of web page loading time is attributed to loading page resources and only 20% is from back-end computations and the reduced file size will result in much increased web page responsiveness. This does not apply to hybrid mobile apps. This is because the web code for the application is already stored in the device. The method presented in the paper would only apply to applications which need to be downloaded every time the website is hit.

4.2 Semi-Automatic Rename Refactoring for JavaScript[2]

Authors Asger Feldthaus and Anders Moller discuss a method for renaming functions using a combination of static analysis and programmer input. While this does not apply directly to our project, it does bring up issues involved with renaming in JavaScript. For instance, similarly named features of the code may be renamed when they should not be. This will lead to a lot of errors in the code. This is a limitation to minification methods as well, since, in order to maintain functionality, certain features such as functions cannot be renamed.

4.3 Modern JavaScript Project Optimizers[4]

The authors of this paper discuss the state of the art of JavaScript optimizers. Tools discussed included JSMin, YUI Compressor, UglifierJS, Google Closure Compiler, and KJS-Compiler and their respective benefits and drawbacks. For instance, KJSCompiler is an extension to Google Closure Compiler where the users can annotate dependencies between files. This is a useful tool to research, due to the fact that it may further reduce the size of JavaScript files.

4.4 Characterizing and Detecting Performance Bugs for Smartphone Applications[?]

This paper begins by presenting common patterns among performance bugs among popular, large scale Android applications. Then, the presents PerfChecker, a static analysis tool that looks for those performance bugs (20 of which were fixed by developers). This paper focuses on bugs relating to a native application's local code and potentially very specific causes for those bugs, such as activating multiple features in a specific sequence. So, the guidelines presented here do not directly address hybrid mobile applications, although the approach or results could in some cases be adapted for hybrids. There is little mention of web language bugs.

4.5 High Performance Web Sites[3]

The paper discussed the Importance of Frontend Performance. It also list best practices and shows why they are important.

1. Make fewer HTTP requests
2. Use a content delivery network
3. Add an Expires header
4. Gzip components
5. Put stylesheets at the top
6. Put scripts at the bottom
7. Avoid CSS expressions
8. Make JavaScript and CSS external
9. Reduce DNS lookups
10. Minify JavaScript
11. Avoid redirects
12. Remove duplicate scripts
13. Confi gure ETags
14. Make Ajax cacheable

5. REFERENCES

[1] Alex Ciminian and Ciprian Dobre. Digua: Minifier and obfuscator for web resources. 2012.

[2] Asger Feldthaus and Anders Møller. Semi-automatic rename refactoring for javascript. In *Proceedings of the 2013 ACM SIGPLAN international conference on Object oriented programming systems languages & applications*, pages 323–338. ACM, 2013.

[3] Steve Souders. High-performance web sites. *Communications of the ACM*, 51(12):36–41, 2008.

[4] IO Zolotareva, OO Knyga, and . Modern javascript project optimizers. 2014.