Evaluating the Impact of Android Best Practices on Energy Consumption

Sona Mundody
Computer Science Department,
Srinivas Institute of Technology Mangalore, India

ABSTRACT
Android best practices for performance are small code changes proposed by Google to reduce execution time. This paper evaluates and analyzes the impact of two of these best practices on performance and energy consumption. The practices are applied to the code of an Android application and the code efficiency is analyzed. The practices indicate a positive impact on performance and energy consumption.

General Terms
Android Performance; Energy Efficiency; Android Best Practices

Keywords
Android; Best Practices; Performance; Energy Consumption

1. INTRODUCTION
The mobile device market is rapidly developing. Most of these devices run Android Operating System. Android is a development platform for mobile applications based on Linux operating system [1] derived from an open source project led by Google.

The Android application development is simplified by its SDK that provides tools and APIs needed to develop applications, favouring an easy integration with many resources available on the device. Due to limited resources available on mobile devices and the limited battery lifetime, the project of mobile apps have hard constraints specially performance and energy consumption.

Many researchers have focused on evaluation of energy consumption and performance for mobile devices, focusing on hardware components or application code. The performance of C and Java was compared in [2], while a comparison between Dalvik Virtual Machine and JVM is presented in [3]. Recently, different algorithm paradigms are compared regarding performance and energy consumption in [4] and different codes for the same purpose are compared in [5]. An evaluation of the performance of the Android best practices is presented in [6].

Google presents best practices for android development focusing on performance improvement. These practices are simple tips to reduce execution time. The focus of this paper is to evaluate and analyze the impact of two of these best practices on performance and energy consumption.

2. ANDROID BEST PRACTICES
Google proposes several best practices [7] for performance to be incorporated in application development. According to the study conducted by Google, the use of these practices provide better overall performance in application.

One of the best practices suggests that the designer must avoid the creation of unnecessary objects. Creating unnecessary objects in application code causes periodic garbage collection and thereby creating negative impact on application performance. The other best practice indicates the use of static methods instead of virtual ones. Google claims that it brings a speed in invocation from 15% - 20%.

Another practice concerns with the declaration and usage of constants and recommends the use of static final for primitive constants and strings. When using the final keyword the class no longer requires a <clinit> method because the constants go into the static field initializers in the .dex file. This makes the access faster. However, the practice is valid only to primitive types and constant Strings.

Another practice suggests that the use of getters/setters methods, common in object oriented languages, should be avoided to improve Android application performance. According to Google, the time to directly access an attribute is faster than through getter/setter methods.

Concerning the manipulation of arrays, Google best practices also present the suggestion about the use of appropriate for syntax. The For-each syntax, introduced by Java 1.5, can be used to manipulate collections that implement the Iterable interface and for arrays and in these cases Google suggests the use of the For-each syntax by default. However Google suggests a hand written counted loop for performance critical ArrayList iteration. The hand written counted loops are the traditional Java for syntax and can have two variations: For with length and For without length. In For without length, the array size is obtained at each iteration. This syntax is slower than the For with length, where the array size is obtained only one time before iterations, instead at each cycle.

The best practices also indicate the use of package access instead private access in private inner classes. This practice is applied when an inner class needs to access the attributes of external class. The virtual machine considers the direct access of inner class to attributes of an external class as illegal, because they are different classes. Applying this practice, one can avoid overhead in applications that use inner class at critical points of performance.

Another best practice indicates that the use of floating point for Android is not recommended. According to Google, the use of floating point is two times slower than integer.

3. METHODOLOGY
The two Google best practices that are evaluated in this work are: the use of appropriate for syntax and avoiding getters/setters. Our experiments evaluate the impact on performance and energy consumption caused by the Android best practices. For all experiments, the emulator provided into the Android SDK is used. The emulator is configured to run on Android 4.2.2 using API 17, and simulating the ARM EABI v7a processor.

For performance evaluation, the android.os.Debug library is used, which generates an execution trace. The execution time is obtained using the Traceview tool which provides the values for Exclude and Include CPU Time. For energy Consumption evaluation the Android application named...
Power Tutor is used, which estimates the energy consumption for an application executed on a target device from its beginning to its end. To analyze the impact of each best practice, the code without the practice is executed to obtain its evaluation results. After that, this code is modified applying the practice, and its evaluation is performed. Each experiment (with and without practice) is repeated thirty times on the emulator, analyzing the execution times provided by Traceview [8] and also energy consumption by PowerTutor [9]. Results presented in section IV are based on arithmetical means and standard deviation for these thirty executions. The Student t test was used to check the statistical significance of our results.

4. EXPERIMENTAL RESULTS

Two best practices evaluated in this work are: using appropriate for syntax and avoiding getters/setters. Firstly these practices are analyzed using experimental codes and finally these are applied to Android applications and analyzed. OpenSudoku [10] is the application that was chosen initially to demonstrate the impact of two of these Android best practices. These impacts are firstly evaluated separately and after that simultaneously. Later the same experiments were carried on other two open source Android applications Asquare [11] and WordSearch[12].

4.1 Best Practices on Experimental Code

Firstly the practice that suggests the use of appropriate for syntax is evaluated using the code fragments shown in Fig. 1. Different implementations of a loop are represented here. The code fragment used for tracing is shown in Fig.2. These implementations are evaluated using Traceview tool and PowerTutor and the results are presented in Table 1.

![Fig. 2: Code fragment for tracing](image)

Table 1. Results of for practice

<table>
<thead>
<tr>
<th>Methods</th>
<th>Performance Incl. CPU Time(ms)</th>
<th>Energy (Joules)</th>
</tr>
</thead>
<tbody>
<tr>
<td>For without length</td>
<td>53.87327</td>
<td>0.00607</td>
</tr>
<tr>
<td>For with length</td>
<td>50.16987</td>
<td>0.00549</td>
</tr>
<tr>
<td>For each</td>
<td>47.16917</td>
<td>0.00514</td>
</tr>
</tbody>
</table>

![Fig. 1: Experimental code fragment for for practice](image)

The impact of avoiding getters/setters methods is evaluated using the code fragment shown in Fig. 3. The code fragment used for tracing is depicted in Fig. 4. The results of evaluation are presented in Table 2. The experiments show that withoutGetter method is faster than withGetter and the energy consumed by withoutGetter method is less than withGetter method.

![Fig. 3: Code fragment –Avoiding getters/setters](image)

![Fig. 4: Code fragment for Tracing](image)

Table 2. Results of avoiding getter/setter methods

<table>
<thead>
<tr>
<th>Methods</th>
<th>Performance Incl. CPU Time(ms)</th>
<th>Energy (Joules)</th>
</tr>
</thead>
<tbody>
<tr>
<td>withGetter</td>
<td>921.4926</td>
<td>0.128</td>
</tr>
<tr>
<td>withoutGetter</td>
<td>309.9727</td>
<td>0.0422</td>
</tr>
</tbody>
</table>
4.2 Evaluating the Impact on Performance and Energy Consumption of Real Apps

First we used the OpenSudoku[10] application to demonstrate the impact of these two best practices on real applications. The code fragment illustrated in Fig. 5 is used for evaluation. This code illustrates the validate() method after the best practices are applied.

```java
protected boolean validate() {
    boolean valid = true;
    Map<Integer, Cell> cells = new HashMap<Integer, Cell>;
    int value; //***************Modified here***********/
    for (int i = 0; i < cells.length; ++i) {
        for (Cell cell : cells) {
            if (cell.getValue().equals(num.getValue())) {
                value = cell.getValue();
                valid = false;
                //cell.setValue(false);
            } else {
                cells.values().put(cell, cell.getValue());
            }
        }
    }
    return valid;
}
```

Fig. 5: validate() method after best practices are applied

Performance and energy results for both analyzed practices are summarized in Table 3 and Table 4 (mean values), while standard deviations are presented in Table 5 and Table 6. Our results indicate that the code without getters is faster and more energy efficient, compared to the one with getters. In these experiments one can observe reduction in execution times and energy consumption.

Table 3. Performance and energy results of for practice

<table>
<thead>
<tr>
<th>Best Practice</th>
<th>Performance Incl. CPU Time(ms)</th>
<th>Energy (Joules)</th>
</tr>
</thead>
<tbody>
<tr>
<td>For without length</td>
<td>1544.285</td>
<td>0.1435</td>
</tr>
<tr>
<td>For each</td>
<td>1351.957</td>
<td>0.1088</td>
</tr>
</tbody>
</table>

Table 4. Performance and energy results of with/without getter practice

<table>
<thead>
<tr>
<th>Best Practice</th>
<th>Performance Incl. CPU Time(ms)</th>
<th>Energy (Joules)</th>
</tr>
</thead>
<tbody>
<tr>
<td>With Getter</td>
<td>7111.282</td>
<td>0.577</td>
</tr>
<tr>
<td>Without Getter</td>
<td>6908.269</td>
<td>0.477</td>
</tr>
</tbody>
</table>

Our results also demonstrate that the For-each presents the best results regarding performance and energy. All these compared values are means and its differences are statistically indicative according to Student t test.

After the separate evaluation, we applied the two practices simultaneously and the obtained results are illustrated in figures below. Fig 6 presents the CPU include time obtained for the validate() method with and without best practices. Fig.7 represents the energy consumed by original and modified code.

Table 4. Standard deviation of with/without getter practice

<table>
<thead>
<tr>
<th>Best Practice</th>
<th>SD(Performance)</th>
<th>Energy (Joules)</th>
</tr>
</thead>
<tbody>
<tr>
<td>With Getter</td>
<td>277.1676</td>
<td>0.02575</td>
</tr>
<tr>
<td>Without Getter</td>
<td>317.256</td>
<td>0.02539</td>
</tr>
</tbody>
</table>

Table 7. Performance and energy results

We also evaluated other two Android applications: Asqare and WordSearch, to study the impact of these practices. These applications used getter methods which were avoided in our experiments. The performance and energy results obtained for original and modified methods in (1) Asqare and (2) WordSearch applications are shown in Table 7 (mean values) and standard deviations are presented in Table 8.
Table 8. Results-Standard Deviation

<table>
<thead>
<tr>
<th>Method</th>
<th>Performance</th>
<th>Energy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Incl. CPU Time(ms)</td>
<td>Joules (Joules)</td>
</tr>
<tr>
<td>Original</td>
<td>734.33</td>
<td>0.104</td>
</tr>
<tr>
<td>Modified</td>
<td>583.64</td>
<td>0.078</td>
</tr>
</tbody>
</table>

The results we obtained indicate that the code without getters is faster and more energy efficient, compared to the one with getters. All the compared values are means and its differences are statistically significant according to the Student t test.

5. CONCLUSION

In this paper, the two of the best practices for performance proposed by Google are evaluated for performance and energy. The experiments are conducted using an open source Android application. The focus of this study is evaluating the impact of these two practices on performance as well as on energy consumption. The presented results have indicated that the use of most appropriate for syntax and avoiding getters produce efficient code, considering both performance and energy efficiency.

6. REFERENCES


