American Research Journal of Computer Science and Information Technology

Volume 7, Issue 1, 73-77 Pages
Research Article | Open Access
ISSN (Online)- 2572-2921
DOI : 10.21694/2572-2921.24016

AMERICAN RESEARCH JOURNALS
An Academic Publishing House

Approaches to Optimizing Power Consumption of Android Applications: Practical Methods and Tools

Vladislav Terekhov

Mobile Applications Developer, Mobilesource Corp, Boca Raton, Florida, United States.

ABSTRACT

Optimizing the power consumption of Android applications developed on the Java platform is a key factor in enhancing the user experience and extending the life of mobile devices. The high level of application power consumption is due to many factors, including working with sensors, network and processor. The article discusses the main optimization techniques, such as managing background processes, using data caching, reducing sensor and CPU activity, as well as using more efficient APIs and technologies such as WorkManager and App Standby Buckets. Code-level optimization and the competent use of third-party profiling tools can significantly reduce power consumption, extending the operating time of the device without recharging. Efficient energy management is becoming a priority for developers seeking to improve application performance and meet the needs of users in conditions of limited hardware resources.

KEYWORDS: *energy consumption optimization, Android applications, Java, mobile devices, background processes, profiling, API, data caching, energy saving.*

INTRODUCTION

The optimization of energy consumption in mobile applications is one of the priority tasks in modern software development. As the number of mobile devices grows and application functionality increases, the issue of energy efficiency becomes more relevant. Android, as one of the most widely used operating systems, offers users a wide range of applications that can significantly affect the battery life of the device. High energy consumption leads to rapid battery depletion, which negatively impacts the user experience and causes dissatisfaction among consumers.

The task of optimizing energy consumption is especially important for applications developed on the Java platform, as this platform is widely used for creating Android programs. Applications running on Java may require significant computational resources, which can adversely affect the battery life of the device. Therefore, developing approaches to reduce energy consumption without compromising the functionality of the application becomes a crucial task for developers seeking to balance performance and energy savings.

Moreover, improving the energy efficiency of Android applications not only enhances the user experience but also allows developers to implement more complex features without increasing the load on the system. This is particularly important given the increasing diversity of hardware characteristics in mobile devices, which requires the development of universal solutions for managing energy consumption across different platforms.

The aim of this work is to analyze the main factors affecting the energy consumption of Android applications on the Java platform and to propose effective optimization methods aimed at reducing the energy consumption of mobile devices.

Reasons for Increased Energy Consumption in Android Applications

Mobile devices with the Android operating system attract a wide audience due to their intuitive interface and the vast number of available applications and games that can be downloaded from the Play Market with a single tap [1]. As of 2024, 71.74% of mobile phones use the Android operating system. Figure 1 below shows the percentage of phones using different operating systems.

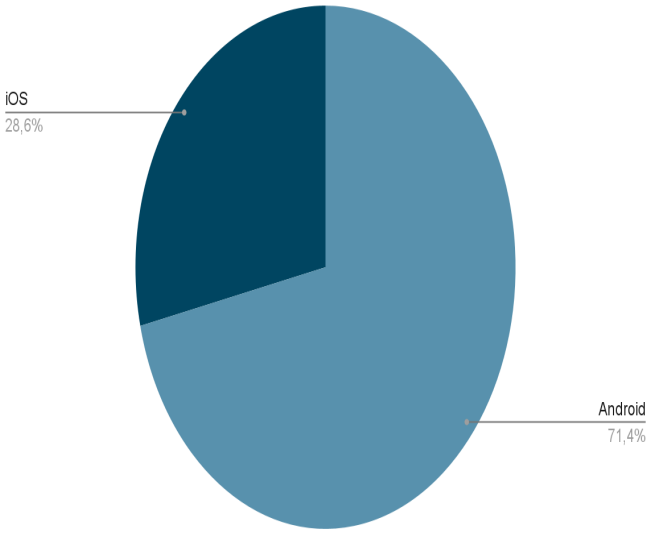

Fig. 1. Percentage of phones using different operating systems [8].

Advanced users appreciate Android for its multitasking capabilities, reliability, lack of system crashes, and the wide range of built-in features, including the ability to replace standard programs with third-party alternatives. Developers value the system for its excellent documentation, cross-platform development tools, and open architecture, which allows flexible management of system functionality.

Manufacturers of smartphones and tablets annually expand their range of models, using displays of various types and sizes, as well as processors with different characteristics, striving to extend the battery life of devices. However, as functional capabilities increase, so does energy consumption, making the task of improving energy efficiency particularly relevant [1]. The energy consumption of mobile devices can be broken down into several key components: the operating system, applications, and hardware sensors. Android is a multilayer operating system based on the Linux kernel with unique extensions for memory management, interprocess communication, and power management. Above the kernel are system libraries and the Dalvik virtual machine, which is optimized for mobile processors with RISC architecture and features low memory consumption.

For developers, Android provides the opportunity to use both native system libraries and development environments, such as ART, which was introduced with Android 4.4 and ensures faster application performance through ahead-of-time code compilation. This enables developers to not only create more energy-efficient applications but also follow design recommendations that minimize system load [2].

Below, Table 1 describes the possible reasons for increased power consumption in Android applications.

Table 1. Possible reasons for increased power consumption of Android applications [3].

Reason	Description
Frequent use of GPS and geolocation	Continuous or overly frequent determination of the user's location via GPS or network data consumes a lot of energy due to the operation of sensors and antennas.
Unnecessary background processes	The application continues to run in the background, updating data or performing tasks, which increases power consumption even when the user is not actively using it.
Frequent data synchronization	Constantly accessing the internet to synchronize data with the server results in high energy consumption due to the use of mobile data or Wi-Fi.
Unoptimized animation and graphics	The use of complex graphic elements and animations without optimization can overload the processor and graphic chip, increasing battery consumption.
Improper sensor management	Continuous or unnecessary use of sensors (accelerometer, gyroscope, etc.) leads to excessive energy consumption.
Frequent interface updates	Frequent updates to the user interface (e.g., widgets, notifications) also lead to increased energy consumption.
Constant network usage	Connecting to the internet to download ads, data, or multimedia elements frequently wakes up the device and drains the battery.
Background music and video usage	Playing multimedia content in the background or with the screen off requires significant resources and increases power consumption.
Unoptimized CPU usage	Applications that overload the processor without proper optimization can cause excessive activity and high energy consumption.
Lack of data caching optimization	Frequent data requests from memory or the network without temporary caching increase the load on the processor and network, leading to higher power consumption.

Techniques for Optimizing Power Consumption in Android Applications

Several optimization strategies can be used to extend battery life in Android applications, each aimed at reducing power consumption and making more efficient use of system resources.

First, it is recommended to separate the data processing logic for mobile networks and Wi-Fi. This is because performance requirements can differ for each type of connection, and configuring separate algorithms can help optimize battery usage under different conditions.

Second, it is necessary to carefully analyze all background

processes to minimize unnecessary operations, thereby reducing system load [4].

It is also important to reduce the number of network requests by using data caching mechanisms. This allows previously loaded information to be accessed during subsequent operations, significantly lowering the frequency of network access.

Batch transmission of network data can be a useful practice. This approach involves grouping network requests, which reduces the number of times the device accesses the network and decreases the frequency of waking from sleep mode.

Attention should also be given to WorkManager, a modern API for managing background tasks. It supports the execution of deferred operations even after device reboot and integrates functionality from earlier APIs like Job Scheduler and FirebaseJobDispatcher, providing better battery management.

Finally, the App Standby Buckets feature, available in Android Pie, improves application management by limiting access to system resources based on app activity [5].

For greater clarity, Table 2 describes other existing techniques for optimizing the energy consumption of Android applications.

Table 2. Techniques for optimizing the energy consumption of Android applications [5].

Technique	Description
Optimization of location services	Limiting the frequency of location updates, using cached location data, or determining location via less energy-intensive network sources.
Limiting background processes	Using Android mechanisms to limit or suspend background processes, such as WorkManager and JobScheduler, to perform tasks only when necessary.
Efficient use of sensors	Disabling unnecessary sensors and setting time intervals for sensor operation, such as the accelerometer or gyroscope, to reduce battery load.
Reducing data synchronization frequency	Adjusting data synchronization based on context, for example, syncing only when Wi-Fi is available or when the device is connected to a charger.
Optimization of animations and graphics	Using simpler animations and graphic elements, reducing the frame rate (FPS), and applying more efficient APIs like Canvas or SurfaceView.
Data caching	Caching network data, images, and other resources to reduce the frequency of server requests and lower network energy consumption.
Optimizing network calls	Combining network requests to perform batch operations, using HTTP/2 or more efficient protocols to reduce energy consumption.
Using push notifications	Replacing frequent server polling with push notifications for alerts and updates, significantly reducing server requests and saving battery power.
Minimizing CPU activity	Limiting CPU wake-ups via WakeLock, performing background tasks using energy-efficient threads.
Task scheduling	Using delayed execution algorithms (lazy loading) and performing tasks at optimal moments, such as only when the device is charging or connected to Wi-Fi.
Using Android battery profiles	Integrating with the Doze and App Standby mechanisms for automatic management of app operation during device inactivity.
Local data processing	Performing operations and processing data on the user's device rather than making constant server requests to reduce network usage and save battery.

Tools and Methods for Analyzing Power Consumption in Android Applications

When developing Android applications on the Java platform, there are several tools available for analyzing power consumption that help optimize device resource usage and extend battery life. Effective power management becomes critically important, especially with the increasing complexity of mobile applications and performance requirements. Below are the main tools for analyzing power consumption in the context of Java application development.

Android Profiler is a built-in tool in Android Studio designed to monitor application performance, including power consumption analysis. For Java applications, Android Profiler allows tracking resource usage, including CPU, memory, network, and battery consumption. The tool provides visual graphs that help developers identify which parts of the code cause significant power consumption. This is especially important when dealing with background services, network request activity, and interface animations.

For more detailed analysis, developers can use the built-in Batterystats utility in conjunction with Battery Historian. Batterystats collects data on the application's energy consumption, including information about the device's state, CPU activity, radio module usage, and the execution

of background processes. These data can be imported into Battery Historian, which converts them into visual graphs and reports, making it easier to identify "energy-intensive" sections of Java code and optimize application performance.

Java developers can also use third-party solutions like Qualcomm's Trepn Profiler. This tool allows for detailed tracking of CPU, GPU, network, and other system components in real-time. It is important to note that Trepn Profiler is particularly useful for devices with Snapdragon chips, where it can collect low-level information on energy consumption and system load. This provides developers with a more accurate picture of how their Java code impacts the overall performance of the device and its power consumption.

For general monitoring and assessment of power consumption, developers can use GSam Battery Monitor. This tool is designed to track the power consumption of applications and the system as a whole. It helps identify Java applications that operate inefficiently, consume too many resources in the background, or make network requests too frequently, which may cause accelerated battery depletion.

In addition to monitoring tools, it is worth noting that the built-in Lint code analyzer in Android Studio also helps identify potential issues related to inefficient resource usage in Java applications. Lint analyzes code for errors that could lead to memory leaks, improper thread handling, or incorrect use of Android components, which can increase power consumption. Integrating this tool into the development process helps prevent performance and power consumption problems at early stages of development [6].

Profiling can also be employed as an important approach. For Android applications running on devices with limited hardware capabilities, such optimization becomes a key task. Efficient use of resources ensures high application performance and a positive user experience. Without the use of profiling methods, achieving optimal performance would be extremely difficult.

When profiling Android applications, several aspects should be considered, including memory management. Incorrect memory handling can lead to failures and errors, such as App Not Responding (ANR). Additionally, data processing directly impacts the user interface: if it is performed inefficiently, the application begins to slow down, which can result in interface freezing or complete stoppage. Network request management should also be considered—delays in data exchange with the server can degrade the user experience. Finally, power consumption optimization is a key factor, as applications with high battery usage are often quickly uninstalled by users.

Android profiling methods may include using developer tools available directly on the device. For example, to analyze GPU performance, rendering profiling can be enabled in the phone settings. The graphical interface displays information about the time spent rendering each frame, which helps identify problem areas.

Memory profiling plays one of the key roles in optimizing applications for mobile devices. With Memory Profiler, memory usage can be analyzed, leaks can be identified, and object allocation can be monitored. CPU profiling allows developers to determine how well the application handles the load, and network profiling helps detect delays in data exchange with servers. Battery consumption profiling helps analyze how energy-intensive the application is and which processes affect battery consumption [7].

Thus, profiling is a necessary step in the Android application development process, allowing developers to create efficient and high-performance applications for users. Summarizing the above, Table 3 describes the tools used to analyze the energy consumption of Android applications.

Table 3. Tools used to analyze the energy consumption of Android applications [7].

Tool/Method	Description
Android Profiler (Android Studio)	A built-in tool in Android Studio for real-time application performance monitoring, including CPU, memory, network, and power consumption analysis.
Battery Historian	A tool that analyzes battery logs and provides information about which applications and processes consume the device's energy.
ADB Shell Dumpsys Battery	A command in Android Debug Bridge (ADB) that provides detailed information about the battery status and the energy consumption of various processes and application components.
System Tracing (Perfetto)	A low-level performance analysis tool that tracks CPU activity, GPU usage, I/O, and other metrics affecting power consumption.
Energy Profiler	Part of Android Profiler, focused on measuring the application's power consumption, tracking battery usage by apps and their components, such as sensors, GPS, and network.
Firebase Performance Monitoring	A remote performance monitoring tool that provides data on CPU load, response time, network delays, and the app's impact on energy consumption.
Trepn Profiler	A tool from Qualcomm that allows developers to analyze CPU, GPU, and memory usage and measure the real-time energy consumption of mobile devices.
GAPID (Graphics API Debugger)	A tool for analyzing and debugging the graphics API on Android devices, enabling the optimization of graphics and reduction of power consumption through rendering optimization.

GFXBench Battery Test	A test for evaluating graphics performance and energy consumption, assessing how long a device can operate under graphical load.
PowerProfile Monitoring	An Android API that allows developers to create energy consumption profiles for device components and adjust optimal power consumption for different application use cases.
Systrace	A system performance analysis tool that tracks CPU, GPU, and thread usage, which can influence power consumption.
LeakCanary	A library for detecting memory leaks that can indirectly impact power consumption, as the app consumes more resources to operate.
LINT (Battery-related checks)	A static code analyzer in Android Studio that checks for issues related to inefficient resource and battery usage.
ACTS (Android Compatibility Test Suite)	A set of tests used to check the device's compatibility with Android and evaluate its behavior, including app energy consumption.

CONCLUSION

In conclusion, optimizing the energy consumption of Android applications on the Java platform is one of the most important tasks for developers aiming to improve the energy efficiency of their products. The application of profiling and code optimization methods, as well as the use of modern Android tools, not only extends the battery life of mobile devices but also significantly enhances the user experience. This is particularly relevant given the growing demands for mobile application functionality and the limited resources of device batteries.

REFERENCES

1. Izergin D. A. et al. Onka computer security of the Android mobile operating system //Russian Journal of Technology. – 2020. – vol. 7. – No. 6. – pp. 44-55.
2. Adeshchenko K. R. Sustainable mobile development: approaches to creating energy-efficient applications // Innovation and investment. - 2024. – No. 3. – pp. 296-300.
3. Novikov K. D., Raskatova M. V. Energy efficiency of mobile web applications //Bulletin of the Russian New University. Series: Complex systems: models, analysis and management. – 2021. – No. 2. – pp. 101-110.
4. Nalivaiko, A. S. Recommendations on memory implementation in Java / A. S. Nalivaiko // Young Scientist. — 2020. — № 24 (314). — Pp. 59-63.
5. Hort M. et al. Review of mobile application performance optimization //IEEE Transactions on Software Engineering. – 2021. – vol. 48. – No. 8. – pp. 2879-2904.
6. Opeyemi B. M. The path to sustainable energy consumption: the possibility of replacing non-renewable energy sources with renewable ones //Energy. – 2021. – vol. 228. – p. 120519.
7. Ulla I. et al. Protection of personal attributes when profiling mobile users based on applications //IEEE Access. – 2020. – Vol. 8. – pp. 143818-143836.
8. 20 Android Statistics For 2024. [Электронный ресурс] Режим доступа: https://www.demandsage.com/android-statistics/ (дата обращения 28.09.2024).

Citation: Vladislav Terekhov, "Approaches to Optimizing Power Consumption of Android Applications: Practical Methods and Tools", American Research Journal of Computer Science and Information Technology, Vol 7, no. 1, 2024, pp. 73-77.

Copyright © 2024 Vladislav Terekhov, This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.