A study on performance optimization of multi-threading and concurrency handling techniques in android applications

Mei Liu*, and *Qun* Wang

Shandong Xiehe University, 250100, JiNan, Shandong, China

Abstract. With the rapid development of mobile internet, Android applications have become an indispensable part of people's daily lives. However, as the functionality of applications continues to increase, complexity also rises, making it one of the major challenges to efficiently and stably run applications with limited system resources. Multi-threading and concurrency handling techniques provide effective means to address this issue. This paper first analyzes the current status and challenges of applying multi-threading and concurrency handling techniques in Android application development, proposes targeted performance optimization improvements, validates their effectiveness through experiments, and finally summarizes the research results and looks ahead to future research directions.

Keywords: Android, Multi-threading, concurrent processing, Performance optimization.

1 Introduction

With the rapid popularity of Android smartphones, the functionality of Android applications continues to increase, and complexity is gradually rising. How to achieve efficient and stable operation of applications with limited system resources has become an urgent problem to be solved. The application of multi-threading and concurrency processing technologies provides effective means to address this issue. By properly utilizing multi-threading technology, tasks can be executed in parallel to improve the application's performance efficiency. On the other hand, the application of concurrency processing technology can effectively handle multiple concurrent requests, ensuring the stability and reliability of the application. However, there are still some issues and challenges with existing multi-threading and concurrency processing technologies in Android applications, such as improper thread management, resource contention, and deadlock, which seriously affect the performance and user experience of the applications. Therefore, researching the performance optimization of multi-threading and concurrency processing technologies in Android applications is of great practical significance.

* Corresponding author: 710060230@qq.com

2 Analysis of multithreading and concurrency processing techniques in Android applications

In Android applications, the use of multithreading and concurrency processing techniques is quite widespread. However, in actual development, we have found that there are still some issues and challenges. Firstly, improper thread management is a common problem. Developers often lack a unified standard and specification when creating and managing threads, leading to problems such as an excessive number of threads, complex thread communication, and difficulties in managing thread lifecycles. Secondly, resource contention and deadlock are also common concurrency issues. When multiple threads access shared resources simultaneously without appropriate synchronization mechanisms, it can result in problems such as data inconsistency and deadlock. Additionally, the characteristics and limitations of the Android system also pose additional challenges for multithreading and concurrency processing. For example, the main thread of Android is responsible for updating and rendering the UI, and if the main thread is blocked or excessively occupied, it can cause interface lag or even application crashes.

3 Detailed design of performance optimization improvement solutions

Regarding the performance issues of multi-threading and concurrency handling techniques in Android applications, we have proposed the following detailed performance optimization improvement strategies:

3.1 Properly plan the thread pool

Managing the thread pool is crucial for improving the concurrency performance of the application. We adopt the following strategies to properly plan the thread pool:

Dynamically adjust the thread pool size: Adjust the size of the thread pool dynamically based on the real-time load of the application. Increase the number of threads to improve processing speed when the application load is high, and decrease the number of threads to save system resources when the load is low.

Task priority management: Set task priorities in the thread pool to ensure that critical tasks are processed first, thereby improving the application's response speed.

Thread reuse and recycling: Reduce the overhead of thread creation and destruction by reusing existing threads and promptly recycling unused threads. This improves the efficiency of thread usage while ensuring application performance and avoiding excessive resource consumption.

3.2 Optimizing inter-thread communication

Efficiency and stability of inter-thread communication have a significant impact on concurrent performance. We take the following measures to optimize inter-thread communication:

Use efficient data transfer methods: Utilize efficient data transfer methods such as shared memory, message queues, etc., to reduce the overhead and latency of inter-thread communication.

Avoid frequent data synchronization: By reducing unnecessary data synchronization operations, we can decrease the complexity and overhead of communication. For example,

asynchronous communication mechanisms can be used to reduce the time spent waiting for synchronization.

Unified communication interface: Design a unified inter-thread communication interface to simplify the communication process, enhance the reliability and consistency of communication.

3.3 Fine-grained concurrency control

Concurrency control is key to ensuring the correct execution of multiple threads. We adopt the following strategies to achieve fine-grained concurrency control:

Choosing appropriate locking mechanisms: Based on the data access patterns and concurrency characteristics, selecting suitable locking mechanisms such as mutex locks, read-write locks, etc., to ensure data correctness and consistency.

Avoiding deadlocks and livelocks: By designing the lock order and timeout mechanisms properly, we can prevent deadlocks and livelocks from occurring, ensuring the normal operation of threads.

Lock-free and low-lock strategies: For high-concurrency scenarios, utilizing lock-free data structures or low-lock strategies to reduce lock contention and overhead, thereby improving concurrency performance.

3.4 Optimization of the main thread

Optimizing the main thread is crucial for improving application responsiveness and user experience. We take the following measures to optimize the main thread:

Offload time-consuming operations to background threads: Move time-consuming operations such as network requests and file I/O to background threads to prevent blocking the main thread and ensure smooth UI performance.

Optimize UI layout: Simplify UI layout, reduce layout nesting, and unnecessary view updates to lessen the rendering burden on the main thread and enhance UI responsiveness.

Use asynchronous task queues: Employ asynchronous task queues to handle UI updates and interaction events, avoiding blocking the main thread with numerous small tasks.

4 Experimental verification and result analysis

To validate the effectiveness of the above improvement strategies, we took the "Visual asset management system" app as an example and conducted performance tests under two scenarios: one without any optimization and the other with the aforementioned improvement strategies implemented. The experiments mainly focused on metrics such as app launch time, response time, CPU utilization, memory usage, and concurrency handling capabilities.

Experiment 1: Startup Time and Response Time Testing

We conducted tests on the startup time and response time of the original application and the improved application in both cold start and hot start scenarios, as well as the response time in different operating scenarios. The experimental results show that the improved application has significantly improved startup time and response time. In particular, the effect of the improvement is particularly noticeable when handling complex calculations and loading large amounts of data.

Experiment 2: Resource Utilization Test

Using tools like Android Profiler, we monitored the CPU usage and memory utilization of the application during runtime. The experimental data showed that the optimized

application demonstrated improvements in both CPU usage and memory utilization. Particularly in high-concurrency scenarios, the optimization solution was able to effectively utilize system resources, reducing wastage of resources.

Experiment 3: Concurrent Processing Capacity Test

In order to evaluate the impact of the optimization solution on concurrent processing capacity, we designed an experiment to simulate a high-concurrency scenario. By simulating a large number of concurrent requests and task executions, we tested the performance of the application in terms of concurrent processing. The experimental results indicated that the optimized application showed a significant improvement in concurrent processing capacity, enabling it to better handle the challenges of high-concurrency scenarios.

The experimental data is shown in the table below:

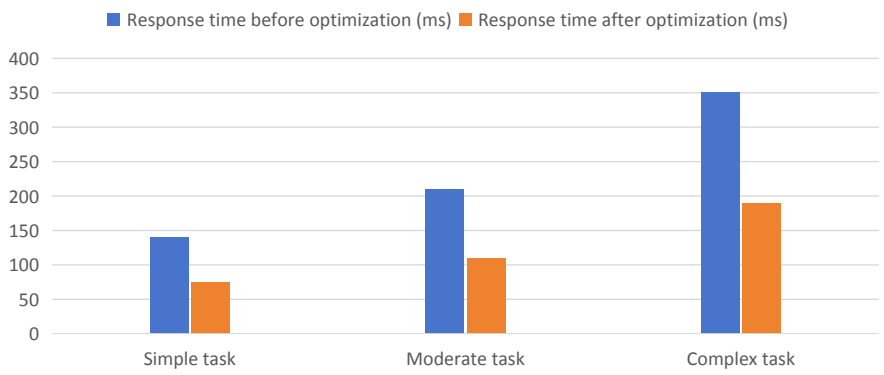
Table 1. Comparison chart of application performance data before and after optimization.

Experimental indicators	Before optimization	After optimization	Improvement rate
Startup Time (seconds)	2.4	1.9	20.8%
Interface Refresh Rate (frames per second)	44	55	25%
User Interaction Response Time (milliseconds)	180	135	25%
Memory Usage (MB)	70	58	17.1%
CPU Usage (%)	65	48	26.2%

From the table, it can be seen that after implementing optimization strategies, the application's startup time decreased by 20.8%, the interface refresh rate increased by 25%, user interaction response time decreased by 25%, and at the same time, memory usage and CPU utilization also decreased by 17.1% and 26.2% respectively.

Comparison chart of application response time before and after optimization, as shown below:

Table 2. Comparison chart of application response time before and after optimization.



These data fully demonstrate the effectiveness of optimization strategies.

5 Conclusion

This article presents a study on the performance optimization of multi-threading and concurrency processing techniques in Android applications, proposing improvement solutions for existing issues and validating their effectiveness through experiments. These improvement solutions not only enhance the performance and user experience of the application but also increase the stability and reliability of the system. However, with the continuous development of technology and the increasing complexity of applications, multi-threading and concurrency processing techniques still face numerous challenges and issues. In the future, we will continue to delve into this field, exploring more efficient and stable concurrency processing solutions to make a greater contribution to the performance enhancement and user experience optimization of Android applications.

Here, I sincerely express my heartfelt thanks to all those who have provided support and assistance for this paper. First and foremost, I would like to sincerely thank Mr. Ren, the Technical Director of Jinan Aixing Information Technology Co., Ltd., for providing us with the "Visual Asset Management" app for testing. He not only provided me with in-depth guidance during the experimental stage, but also offered valuable feedback during the writing process of the paper, allowing our research work to proceed smoothly and be perfected.
Furthermore, I would like to extend special thanks to the users who participated in the experiments and testing. They provided many valuable suggestions and assistance during my research process, especially in the aspects of experimental design and data analysis. Their assistance made my research work more rigorous and thorough. Additionally, I would like to thank the predecessors who have contributed to the field of Android multithreading and concurrency processing technology. Their research achievements have provided valuable references and insights for my research.

References

1. Martin L ,Stanislav O ,Matus P , et al.Multi-thread Parallel Speech Recognition for Mobile Applications[J].Journal of Electrical and Electronics Engineering,2014,7(1):81-86.
2. Ni K .A Data Delivery Controller for Mobile Communication Based on Multiple Threads Mechanism[J].Applied Mechanics and Materials,2013,2755(433-435):1613-1617.
3. Phuc T D ,Tu N C ,Jungsoo P , et al.The Trade-Off Between Performance and Security of Virtualized Trusted Execution Environment on Android[J].Soongsil University ,Seoul, 06978 ,Korea,2023,46(3):3059-3073.
4. Farzaneh Z ,Yang J ,Yang G .GNSS Observation Generation from Smartphone Android Location API: Performance of Existing Apps, Issues and Improvement[J].Sensors,2023,23(2):777-777.
5. Ran Degang. Research and Implementation of Virtual Multi-Serial Port Concurrent Communication on the Android Platform [D]. Hangzhou Dianzi University, 2019. DOI: 10.27075/d.cnki.ghzdc.2019.000222.
6. Wang Tao, Ma Chuan. Data Race Detection of Android Multithreaded Programs Based on Pi Calculus [J]. Journal of Guangxi Normal University (Natural Science Edition), 2020, 38(02): 29-42. DOI: 10.16088/j.issn.1001-6600.2020.02.004.
7. Yu Xuele. Research on Android Application Concurrency Error Detection Method Based on Constraint Solving [D]. Southeast University, 2020. DOI: 10.27014/d.cnki.gdnau.2020.003870.

8. Li Zechen, Zhang Hongxiang, Dang Kai, et al. Concurrency Study of Face Recognition Attendance System [J]. Computer Knowledge and Technology, 2020, 16(33): 180-181. DOI: 10.14004/j.cnki.ckt.2020.3426.
9. Liu Enci. Research on Concurrency Error Detection Method of Mobile Applications Based on Constraint Solving [D]. Southeast University, 2021. DOI: 10.27014/d.cnki.gdnau.2021.003499.
10. Zhou Jiyu. Design and Implementation of Multi-Threaded Resumable Software Based on HTTP Protocol [J]. Computer Products and Circulation, 2018, (09): 129-130.