DroidPerf: Profiling Memory Objects on Android Devices

Bolun Li, Qidong Zhao, Shuyin Jiao, Xu Liu
NC State University

October 5, 2023
Motivations

Existing popular profilers on ART (Android runtime)

- PowerTutor, PowerScope - Energy consumption analysis
- Android profiler - Hotspot analysis
- Perfetto - Trace analysis
- Other profilers - Cross-layer inefficiency or app crash analysis

Why we need a profiler on ART (Android runtime)

- Existing tools mainly focus on hotspot analysis profiling
- Existing tools fail to tell whether a resource is being used productively and contributes to a program's overall efficiencies
- Android applications are highly susceptible to object locality issues

Need a new profiler for profiling memory objects on Android devices
Study of Memory Inefficiencies in Android Applications

- Memory Bloat - High memory usage in loops
- Android persistent running services
- Improper use of Android APIs (e.g., Android broadcast APIs)
- Runtime configuration changes (e.g., screen orientation changes)
- Use Android unoptimized data structure (e.g., HashMap (ArrayMap is optimized))
- Generate a large amount of duplicated objects
Challenges of Building an Android Memory Profiler

- Lack of library to support hook the object allocation on Android
- Limited JVM Tool Interface (JVMTI) support
  - Missing important callbacks, e.g., callbacks when a method is compiled and loaded into memory, or unloaded from memory
- Lack of async unwind facility
  - Missing AsyncGetCallTrace facility, which is used to get calling stack inside a signal handler to avoid the safe point
- Unable to obtain sampled PMU effective address on ARM
  - ARM only provides an instruction pointer on each PMU sample
Methodology

DroidPerf in system stacks

1. Android App (Unmodified)
2. DroidPerf
3. ART Tool Interface (ART TI)
4. Virtual Machine
5. Perf Events Interface
6. Android OS
7. Performance Monitor Unit
8. Android Commodity CPU
Methodology

Capture object creation and destruction

- **JVMTI** ObjectAlloc Callback
  - Allocated thread
  - Reference to the allocated object
  - Reference to the class of the allocated object
  - Object Size

- **JVMTI** ObjectFree Callback
  - Tag of the freed object
Methodology

Construct call path

● Monitor Executed Function: MethodEntry & MethodExit
● Obtain line number: GetLineNumberTable API

main() {
    methodA();
}
methodA() {
    methodB();
}
methodB()
Methodology

DroidPerf’s object-centric analysis
Overview of DroidPerf

Workflow of DroidPerf

- Online Data Collector
- Offline Data Analyzer
- GUI

Memory Inefficiencies Code

Attached as an ART Agent

Android Application

Application Source Code
Evaluation

Time overhead

- Google Pixel 7 (Google Tensor G2 chipset, 8 GB RAM)
Evaluation

Memory overhead

- Google Pixel 7 (Google Tensor G2 chipset, 8 GB RAM)
Case Study of DNS66

VSCode GUI on Android Application DNS66

Optimization result: reduce the total cache misses by 18.97% and the heap memory usage consumption by 6.6%
### Case Studies

<table>
<thead>
<tr>
<th>Android Applications</th>
<th>Inefficiency</th>
<th>Optimization</th>
</tr>
</thead>
<tbody>
<tr>
<td>DNS66</td>
<td>AdVpnThread.java</td>
<td>Excessive memory usage in nested loops</td>
</tr>
<tr>
<td>Rajawali</td>
<td>RenderToTextureFragment.java</td>
<td>Long running services</td>
</tr>
<tr>
<td>RxAndroid</td>
<td>HandlerScheduler.java</td>
<td>Inefficient using Android broadcast API</td>
</tr>
<tr>
<td>Applozic</td>
<td>MessageClientService.java</td>
<td>Runtime configuration change (screen orientation, etc. changes)</td>
</tr>
<tr>
<td>GmsCore</td>
<td>TriggerReceiver.java</td>
<td>Poor data structure</td>
</tr>
<tr>
<td>Twire</td>
<td>Service.java, TopStreams/Activity.java</td>
<td>Generate duplicated objects</td>
</tr>
<tr>
<td>Stream Chat</td>
<td>QueryChannelLogic.java</td>
<td></td>
</tr>
<tr>
<td>Stetho</td>
<td>AsyncPrettyPrinterRegistry.java, ChromeDevtoolsServer.java, JsonRpcPeer.java, ResponseBodyFileManager.java</td>
<td></td>
</tr>
<tr>
<td>MediaPicker</td>
<td>Utility.java</td>
<td></td>
</tr>
<tr>
<td>LeakCanary</td>
<td>AndroidDebugHeapAnalyzer.java</td>
<td></td>
</tr>
<tr>
<td>Foxy Droid</td>
<td>QueryBuilder.java</td>
<td></td>
</tr>
</tbody>
</table>

Many optimization patches were confirmed: Applozic, GmsCore, Twire, etc. Upstreamed optimization patches: DNS66, Rajawali
Conclusions

● Categorize different Android memory inefficiencies

● Develop DroidPerf to pinpoint object-level inefficiencies that occurred on Android Runtime
  ○ Support lightweight novel object-centric profiling
  ○ Apply to unmodified applications
  ○ Obtain nontrivial performance gains

● Contribute to the community
  ○ Many optimization patches are upstreamed to real-world applications
Future Work

- Enhance DroidPerf's Feature
  - Support instruction-level monitoring
  - Support instruction-level inefficiency analysis
- Enhance DroidPerf's Usability
  - Develop an extension for Android Studio to integrate DroidPerf into the Android Development Environment