LE/EECS 4443: Mobile User Interfaces (LAB)

Week 7: Introduction to Profiling & Benchmarking

S.Toyonaga[1]

[1]Lassonde School of Engineering
York University

May 19, 2024

Table of Contents

Introduction

By the end of this tutorial, you will be able to...

1. Understand broadly what performance encompasses
2. How to profile an application
3. How to benchmark an application

Disclaimer

Certain sections of this tutorial will be theory-heavy while others will be highly application-based.
One does not preclude the other in importance.. It is highly recommended that you read through the entire slide-deck to attain a more in-depth understanding of the importance of performance and testing.

Performance

- Broadly speaking, we can think of performance as empirically **evaluating** the **efficiency** of an entity relative to the amount of required **resources**.
- **Goals:**
  1. Proving conformity of system to performance standards
  2. Comparing systems with each other
  3. Identifying components with poor performance

Software Performance

- Measures the efficiency of software with respect to time and resource allocations.

  i.e., Response Time, Throughput, Utilization...
- **Note:** It is possible to define other metrics which support different fine-tuned objectives.
- The scope of performance evaluation can be limited to subsets of the entire system or the whole system.

Flavours of Performance Testing

1. **Load Testing:** Evaluating performance under an **expected** workload.
2. **Endurance Testing:** Load testing, but just for a longer period of time.
3. **Stress Testing:** Evaluating performance under the absolute limit.
4. **Spike Testing:** Evaluating performance under environments with very high entropy.
5. **Capacity Testing:** Slowly increase the workload and find the point where maximum capacity is reached.
6. **Configuration Testing:** Evaluating performance with the goal of comparing different configurations against each other.

Putting It All Together...

Definition

"An app is considered to have poor performance if it responds slowly, shows choppy animations, freezes, or consumes too much power. Fixing performance problems involves identifying areas in which your app makes inefficient use of resources such as the CPU, memory, graphics, network, or the device battery."

https://developer.android.com/studio/profile

Performance Measurements

1. **Monitoring Performance**
   - **Static:** Metrics can be obtained without running the program.
   - **Dynamic:** Metrics can only be obtained by running the program.
2. **Metrics**
   - **Internal:** Metrics of the application which are obtained by event detection and performance data logging. Typically, these metrics are obtained by writing code into the application.
   - **External:** Metrics that are gathered independently from the software being evaluated.

Objectives of Performance Measurements

1. Understanding the System
2. Specification of Expected Workloads & Performance
3. Improvement of Pre-Existing Models
4. Verification & Validation to Goals & Specifications of Expected Performance
5. Evaluation

Bottom Line: You must think about how to create a robust measurement plan that evaluates the system effectively

Problems with Performance Measurements

- Measuring performance is susceptible to noise. If we try to measure performance under very constrained conditions, the results will not be generalizable to the broad public.

  i.e., Internal vs External Validity
- Noise and further complications can be caused by:
  1. System Perturbation with relation to Measurement Software
  2. Tug-of-war between capture rate and system overload
  3. Representative time intervals & workload generation

Addressing the Problems (Green Flags)

1. Always be performance testing! Don't wait until the end of the software development life-cycle to start!
2. Writing many flavours of tests with many scenarios is a MUST. Make your performance tests span a wide variety of cases.
3. Keep an eye on the cache to avoid unrealistic recovery speeds. Restart the computer, clear the cache, etc.,
4. Make use of local resources!

Helpful Resources

1. https://developer.android.com/build/optimize-your-build
2. https://www.youtube.com/watch?v=-Mgy_nTMCtc

Introduction

- Profiling offers a dynamic analysis that measures the overall complexity of an application.
- In Android studio, we can evaluate CPU, Memory, Network, and Energy usage at broad and very deep levels of granularity.
- **Components of Profiling:**
  1. Instrumentation ("Probes")
  2. Sampling

Profiling: Instrumentation

- **Internal** → Existing within the source code, compilation, or binary code
- To effectively use instrumentation, you must identify the metrics of interest, choose the capturing level of granularity, and then use logging or checkpoints to collect the data.
- **Note:** Overhead (noise) caused by profiling is substantial if the recorded operation is too short or fast.

Profiling: Sampling

- **Recall (LE/EECS 3221):** Profiling is achieved by correlating instructions with CPU interrupts. It requires a sufficient amount of sampling to attain statistical significance.
- Sampling is not as accurate as instrumentation, however, it is more efficient and less invasive.
- Sampling can lose information or generally speaking, fail to capture an important interaction if it misses an interrupt.

Rule of Thumb

If the operations are sufficiently slow, it is preferable to use instrumentation. The additional overhead becomes insignificant.

Introduction

Definition

"Benchmarking is a way to inspect and monitor the performance of your app. You can regularly run benchmarks to analyze and debug performance problems and help ensure that you don't introduce regressions in recent changes."

https://developer.android.com/topic/performance/benchmarking/benchmarking-overview

Benchmarking is about comparing performance between applications. Profiling is about understanding why performance occurs.

Flavours of Benchmarks

1. **Specification-Based (≈ Interface)**
2. **Kit-Based (≈ Abstract Class without Overriding)**
3. **Hybrid (≈ Hybrid)**
4. **Synthetic:** Simulated workloads in isolation
5. **Microbenchmark:** Focuses on a specific function enabling the quantification of bottlenecks.
6. **Macrobenchmark:** Allows code to be run and evaluated separately from running the full application itself. You can measure metrics such as the applications startup time and UI-related features (i.e., "Jank").
7. **Kernel:** A broader, more generalizable version of a Microbenchmark. We will consider this a, "basic" simulation, so to speak.
8. **Application:** A real world, real behaviour simulation.

Primary Strategies of Benchmarking

1. Fixed-Work
2. Fixed-Time
3. Variable ($Work \vee Time$) Method

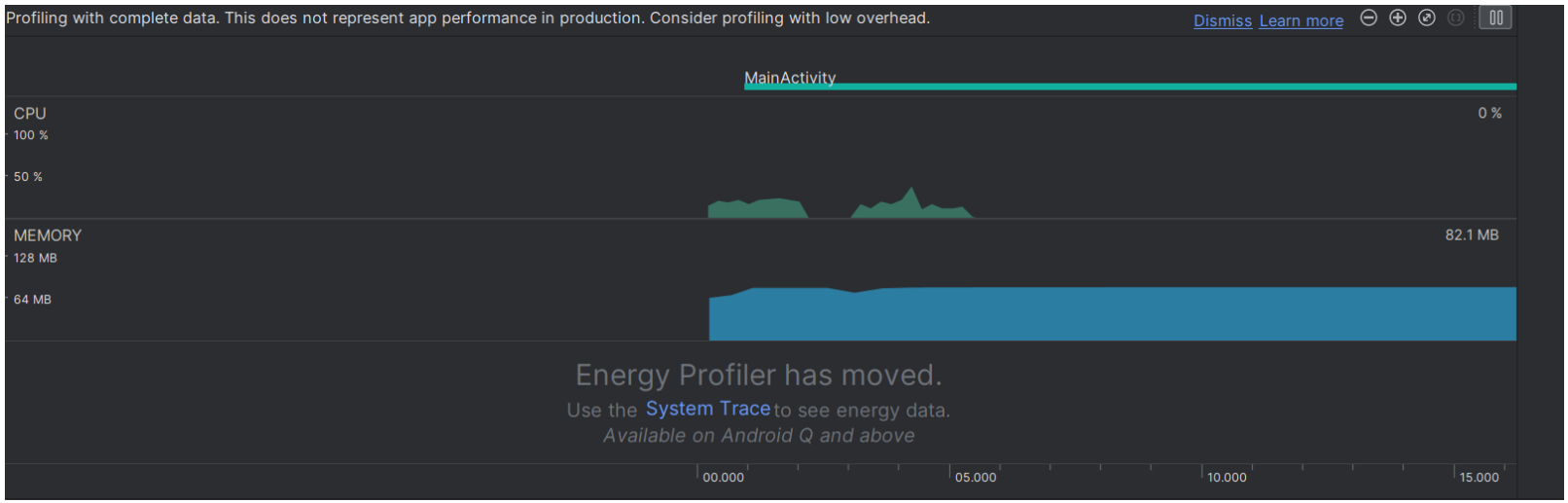
Conclusion

Final Words

A good benchmark has a workload that is **relevant**, producing results which are **reproducible, fair, verifiable, and usable**.

Getting Ready with the Profiler

1. In your application, on the top right, click ":"
2. Begin profiling your application. It should launch the application in your emulator with the following window.



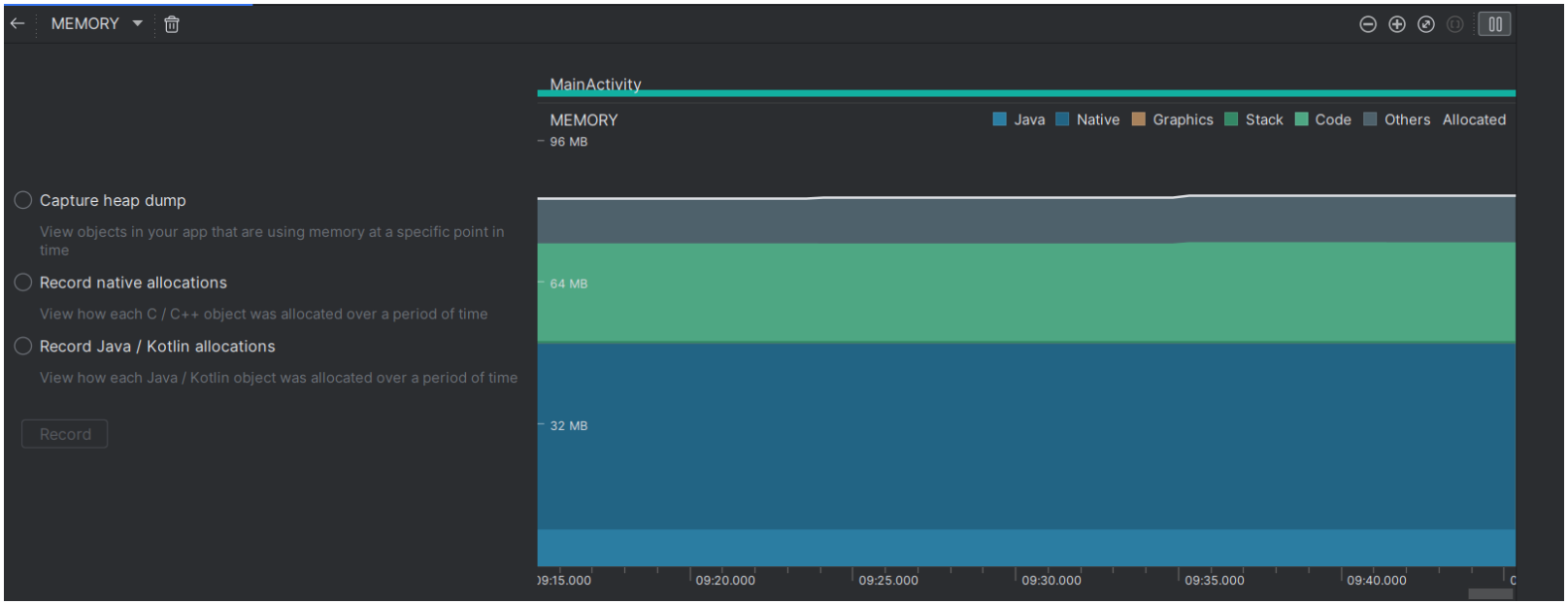
Clicking on the CPU ∨ Memory rows will open up an expanded view where we can run traces (sampling) to view performance metrics.

Profiler: CPU

Read:

https://developer.android.com/studio/profile/cpu-profiler

Profiler: Memory Stack



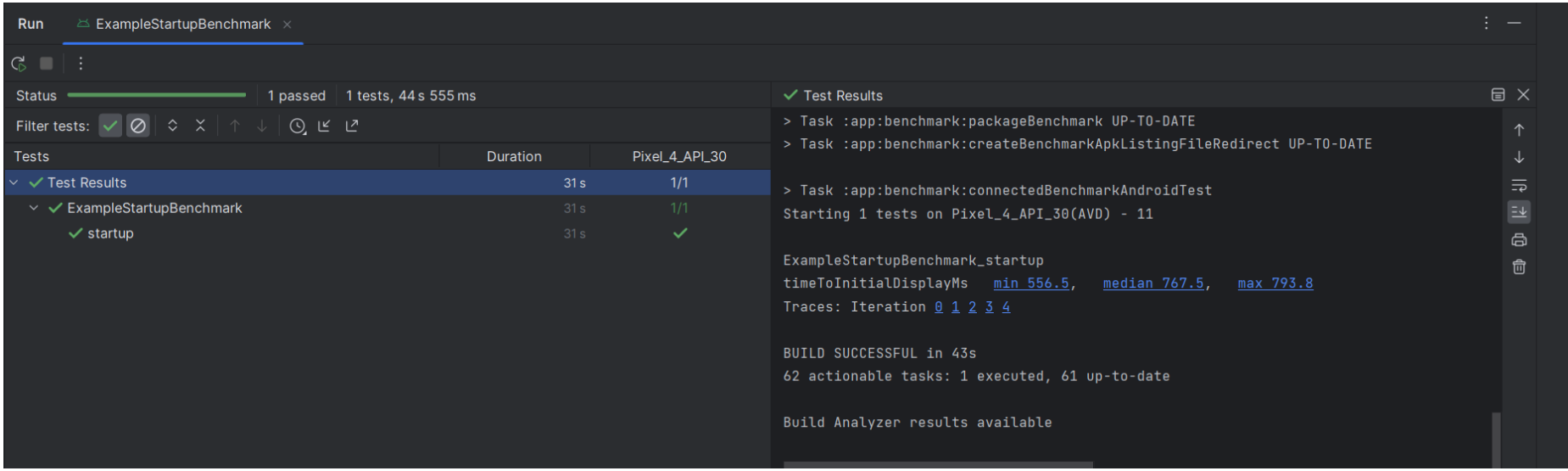
Read:

https://developer.android.com/studio/profile/memory-profiler

Getting Ready with Benchmarking

1. Right Click the **app** directory and select *New* → *Module*
2. From the, "Templates" selection, select, "Benchmark". Ensure that the settings are correct and click, "Finish"
3. An ExampleBenchmark will be configured. Wait for all of the dependencies to complete downloading.
4. When you **run** the application, all of the tests will run in sequence on the emulator. After your tests have completed running, you should see the following image.

Benchmark: Completion



Benchmark: Code

```
@Test
public void startup() {
    mBenchmarkRule.measureRepeated(
            "com.example.touchevents",
            Collections.singletonList(new StartupTimingMetric()),
            CompilationMode.DEFAULT,
            StartupMode.COLD,
            5,
            scope -> {
                scope.pressHome();
                scope.startActivityAndWait();
                return null;
            });
    }
```

It's a Wrap (Summary & Future Resources)

- **Writing a Macrobenchmark:** https://developer.android.com/topic/performance/benchmarking/macrobenchmark-overview
- **Writing a Microbenchmark:** https://developer.android.com/topic/performance/benchmarking/microbenchmark-write
- **Understanding the Profiler & How to Measure Performance:** https://developer.android.com/topic/performance/measuring-performance
- **Improving Performance:** https://developer.android.com/topic/performance/improving-overview
- **Demo (TypingTest):** https://github.com/stoyonaga/EECS4443_W24_Assets/tree/main/TA%20Demos/TypingTest
- **Performance Debugging with SysTrace:** https://www.youtube.com/watch?v=ktknfQykhXU

LASSONDE SCHOOL OF ENGINEERING

Conclusion

Regard

Thank you for your time! :)
Questions?