

CS 326

Performance Tuning

Stephen Freund

1

The Goal

- Ensure system has acceptable
 - throughput
 - latency
 - memory footprint
 - network usage
 - ...
- For anticipated workload

When To Do Performance Tuning

- During design? Nope. But:
 - Understand algorithm running times
 - and make **reasonable** choices
 - Identify design tradeoffs that may impact performance (sorted vs. unsorted list, etc.)
 - Document choices you make, alternatives, assumptions about the workload
 - *Always* favor simple solutions until proven they are insufficient.
- **It is much easier to optimize a well-designed system than a poorly-designed system.**

When To Do Performance Tuning

- During initial implementation? Nope. But:
 - Follow good programming discipline
 - Build abstractions so you can change reps later
 - Adhere to basic common sense:
 - don't re-compute same value unnecessarily
 - don't do more work than necessary
 - ...
- **It is much easier to optimize a well-written system than a poorly-written system.**

When To Do Performance Tuning

- When writing correctness tests? Nope.
 - Focus on correctness tests first
- **It is much easier to optimize a correct system than an incorrect system.**

Performance Tuning



- *Premature optimization is the root of all evil.*
 - Donald Knuth
- Only when you recognize that the system fails to meet desired performance goals
- Methodology is similar to debugging once a failure has occurred.

Experiment-Driven Methodology

- Measure the performance of the system before modification.
- Identify a bottleneck
 - part of the system that is critical for improving the performance.
- Modify the system to remove the bottleneck.
- Measure the performance of the system after modification.
 - improvement => adopt change
 - no improvement => **revert to original**

Reproducible Test Inputs

- Real Data (eg: Marvel Comics)
- Synthetic Data (eg: my "synthetic" graph)
 - Write a program to generate "fake data"
 - Mimics all relevant characteristics of real world
 - Better understanding of structure of data
 - Easier to create different size inputs, inputs w/ different features, etc.
- Typically a combination
- Being able to quickly write data generators is a handy skill.

Identifying Bottlenecks

- "90-10 Rule"
 - 90% of the time is spent in 10% of the code.
- How to find that 10%
 - Log messages
 - Timers/counters in code
 - Profiler
 - dynamic analysis to measure properties of program behavior at run time
 - where does the program spend time?
 - how is memory being used?
 - ...

Algorithmic Bottlenecks

- Implementation contains algorithms with inadequate Big-O run times.
- Example:
 - originally kept an array unsorted with $O(n)$ search
 - change to a sorted array with $O(\log n)$ search
 - but at the cost of slower insertions...
- Requires you to know algorithm design and analysis basics.
- May need to change ADT internal reps, etc.

Implementation Bugs

- Defect leads to unnecessary computation.
- Example:

```
func inTree(root: Tree, v : Int) -> Bool {
  if (v == root.value) {
    return true
  } else {
    let inLeft = inTree(root.left, v)
    let inRight = inTree(root.right, v)
    return inLeft || inRight
  }
}
```
- Standard debugging techniques...

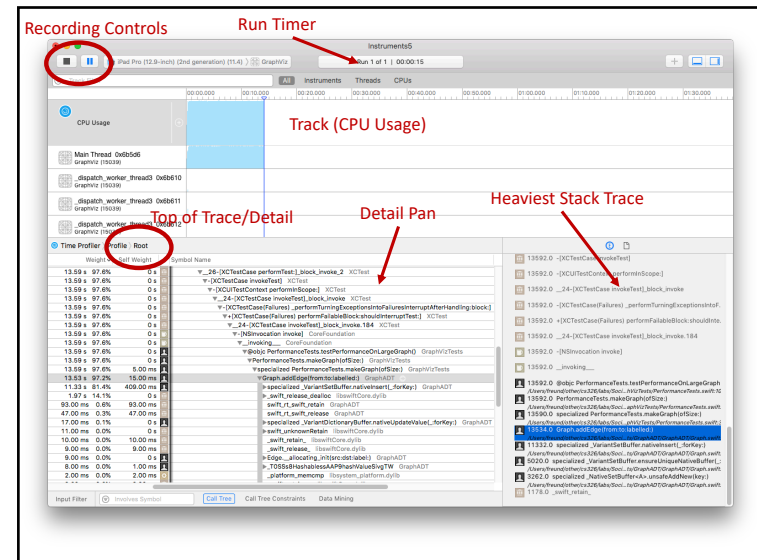
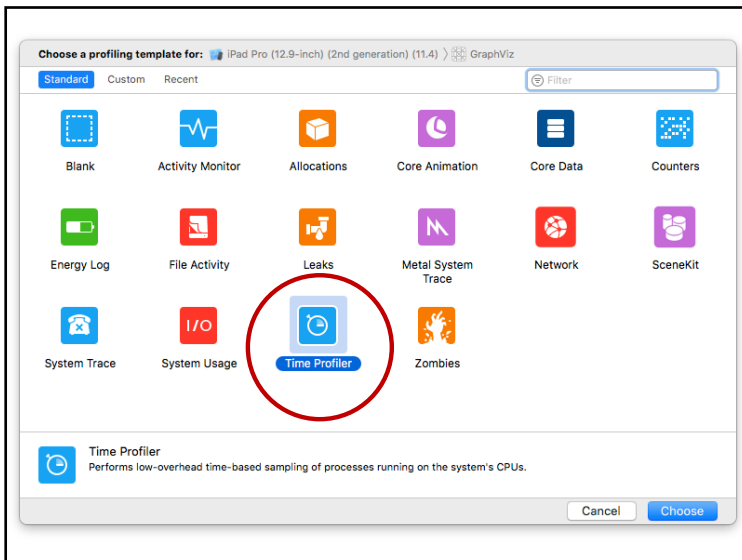
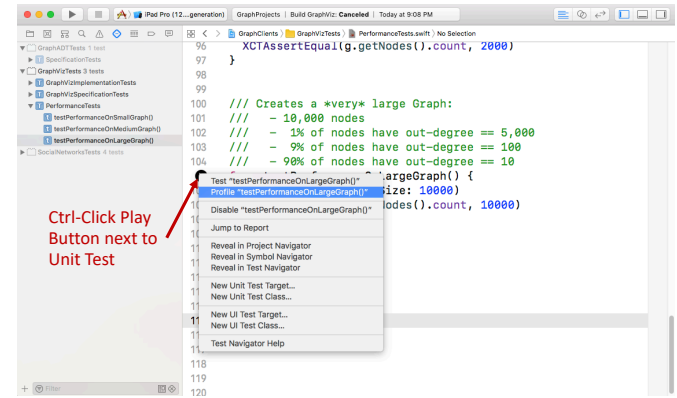
Bad Assumptions / Library Impls.

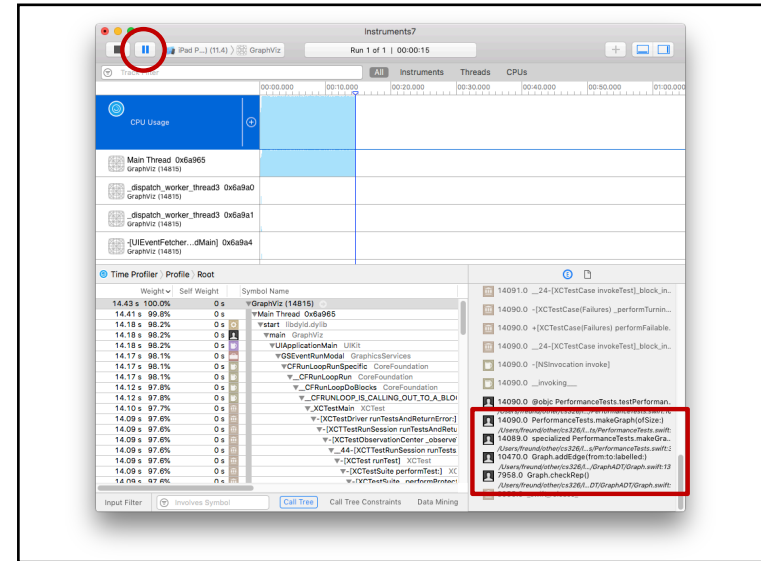
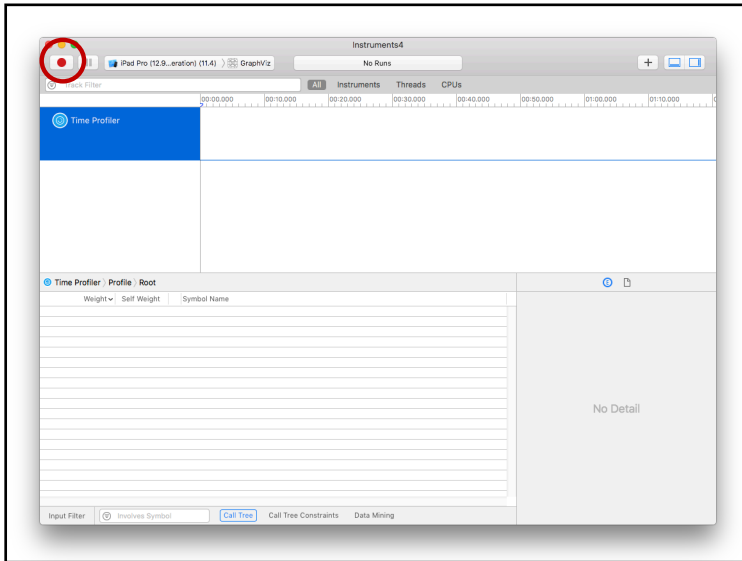
- Code you use may have different performance characteristics than you assumed.
 - Library call creates a new http connection every time, rather than reusing an existing connection
 - Creating UIBezierPaths becomes a bottleneck if you render a few thousand nodes in your GraphView
- Primitive ADTs may have limitations
 - Swift Sets/Dicts/Arrays are values
 - Copy-on-write strategy avoids a lot of copying
 - But how much are you paying for what's left?

Expensive Run-time Checks

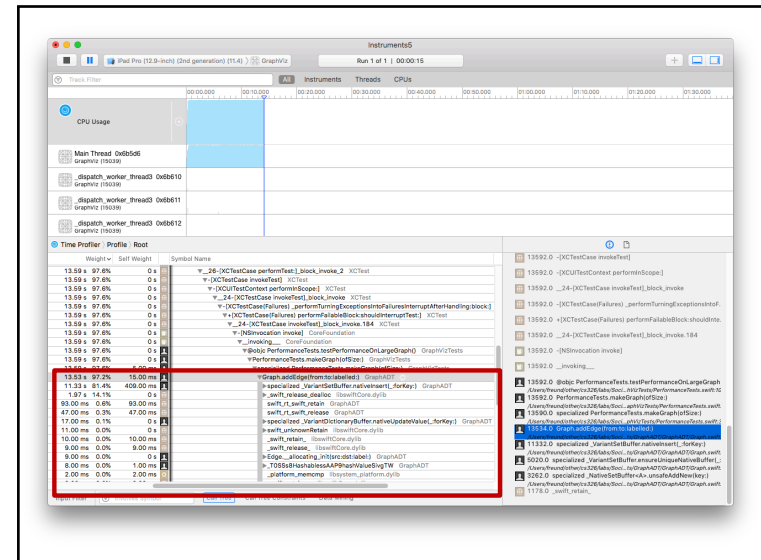
- precondition/invariant assertions
- checkRep() calls
- Never delete checks
 - Add flag to selectively turn them off.
 - You will likely need to do this for SocialNetworks...

Running Profiler in XCode



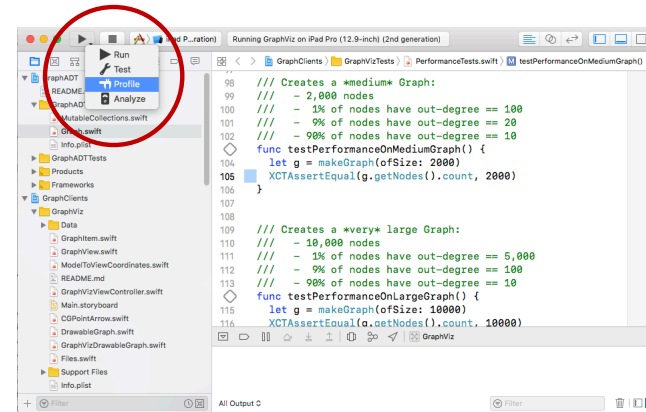


Modify checkRep()...



Change From Set To MutableSet

Running App With Profiler



The screenshot shows the Xcode IDE with a Swift file open. The file contains two performance test functions: `testPerformanceOnMediumGraph()` and `testPerformanceOnLargeGraph()`. The `testPerformanceOnMediumGraph()` function creates a graph with 2,000 nodes and asserts that 1% of nodes have an out-degree of 100, 9% have an out-degree of 20, and 90% have an out-degree of 10. The `testPerformanceOnLargeGraph()` function creates a graph with 10,000 nodes and asserts that 1% of nodes have an out-degree of 5,000, 9% have an out-degree of 100, and 90% have an out-degree of 10. A red circle highlights the `Run` button in the top toolbar.

```
98 // Creates a *medium* Graph:
99 // - 2,000 nodes
100 // - 1% of nodes have out-degree == 100
101 // - 9% of nodes have out-degree == 20
102 // - 90% of nodes have out-degree == 10
103 func testPerformanceOnMediumGraph() {
104     let g = makeGraph(ofSize: 2000)
105     XCTAssertEqual(g.getNode().count, 2000)
106 }
107
108 // Creates a *very* large Graph:
109 // - 10,000 nodes
110 // - 1% of nodes have out-degree == 5,000
111 // - 9% of nodes have out-degree == 100
112 // - 90% of nodes have out-degree == 10
113 func testPerformanceOnLargeGraph() {
114     let g = makeGraph(ofSize: 10000)
115     XCTAssertEqual(g.getNode().count, 10000)
116 }
```