Performance Evaluation and Optimization
Spring 2019
Course Information

Last updated: 2021-Jan-15

Be sure to read **all** of this document!

1 The topics

Computer systems are immensely complex, yet much of that complexity is hidden from the typical programmer (and user). Layers upon layers of abstracted capabilities are employed when the user-level programs are run. When the program performs sufficiently well for its users, then those layers are behavior sufficiently well.

But what happens when a program performs poorly? Sometimes, the answer is provided by asymptotic algorithmic analysis: the programmer used an $O(n^2)$ sorting algorithm on a truly large array. More often, however, the proximate cause of the slowness involves the constant factors that such theoretical analyses ignore. How do we find where those constant factors are larger than they need to be? Through experimentation, coupled with a detailed understanding of those layers, their interfaces, and how their implementations interact. The primary cause of the the slowness could be something direct—a heavily used function that could perform its task in fewer steps—or something caused by unexpectedly problematic interactions indirectly caused by the subtle elements of the program.

The primary focus of this course is simply stated: How do you find the slowness? And how do you then fix it? Our work will bring us to a number of experimental methods that allow us unambiguously to determine whence the inefficiency, as well as the method of optimizations that follow.

Here is a rough list of the topics with which we will spend our time:

- **Profiling and timing:** How to measure the time and memory are used by different parts program and system. Call graph profilers (`gprof`), binary instrumentation tools (*Valgrind*), kernel tracing tools (*DTrace/SystemTap*), and CPU performance counters (*PerfSuite*) can be used to measure performance as a whole and to probe the inner performance of individual components.
- **Benchmarks and models:** How do we come up with a suite of programs and inputs that are representative of how a system will really be used? Can the behavior of programs or systems be reduced to a mathematical model? What happens if our test suite or model are not sufficiently representative?
- **Identifying bottlenecks:** Once we know how to measure performance on representative inputs, how do we pinpoint the parts of the program/system that are limiting

performance? What if the limiting behavior occurs deep in the system in unexpected ways?

- **Performance optimization:** Having identified the bottleneck, in what ways can we try to alleviate the bottleneck and improve the performance?
- **Performance analysis:** How do we know whether our optimization actually improved performance? If the performance seems to improve, how do we know whether our change caused the improvement as intended, or caused it unintentionally by altering the behavior of some other part of the system? What happens when multiple improvements are composed?
- **Parallelism and scalability:** If our code runs in parallel, we hope for it to *scale*—its performance should, in theory, improve linearly with the number of processing units. How do we find the source of the communication/synchronization that limits the scaling? What if the source isn't in our code, but in some behavior deeper in the system that our code is triggering?

This course will be a kind of experimental-methods seminar. We will read a few research papers and analyze them. We will develop, as a class, experiments to test our understanding of performance limitations. And then we will implement, execute, and analyze these experiments in groups, presenting and discussing our results with one another.

2 Lectures, labs, and help

Lectures and labs: This class will meet on **Tuesdays and Thursdays**, from **1:30 pm to 2:50 pm**. We will variously use this time for lectures, group discussions, and labs. Initially, our class meetings will be on Zoom; whether there is any in-person compontent to these class meetings is yet to be determined.

You are expected to be present for **all class meetings**. This is a smaller class that will depend heavily on working together. Be prepared each day to participate fully.

Office hours and meetings: If you seek assistance, reinforcement, review, or other opportunities to discuss the course material or assignments, you should see me. There is a link on the course web page for scheduling times for meetings.

Communications: You should certainly feel free to send **email** (<sfkaplan@amherst.edu>) with your questions or thoughts. I also encourage the use of *Slack*. We will have a channel for the class, as well as a channel for each research group, and it is a great way to ask questions and share answers.

3 Texts and materials

There is no one textbook that covers the material from this class. There will be occassional research papers to read, as well as documentation/tutorials for the system components and measurement tools that we will be using. These will be posted on the class web page as the semester progresses.

4 Projects

Because this course focuses on experimental methods, we will spend significant time designing, implementing, executing, and analyzing our own experiments. Some of this work will involve clearly defined, individual work that you will submit. Other parts of these experiments will be carried out in groups. Your engagement with both the individual and group work is indispensible, both for you and for the class as a whole.

5 Grading

At the end of the course, I will assess your mastery of the course material. To do so, I will consider your individual project work, your group project work, and your participating during in class lectures and discussions. All three of these elements of the course are opportunities for you to demonstrate your understanding of the material.