NYU Distributed Systems: Introduction and Goals

Welcome! This is a distributed systems course targeted at students who have previous experience with computer systems (that is, have taken classes in operating systems and architecture), and with theory (that is, have taken classes on algorithms and understand algorithmic analysis).

Why distributed systems?

Computers and programs running on them can fail: computers can lose power or network connectivity, programs can crash, hardware can degrade, etc. But failures are an issue for computerized services. For some, e.g., when talking about air traffic control, this can be catastrophic[1], while for others, e.g., with GMail or Netflix, failures can have high economic costs. Thus fault-tolerant services, which remain available despite computer or program failures are desirable. Running programs on multiple computers is the only way to achieve this goal. Thus, distributed system, which are programs designed to be run on a collection of computers (connected by a network) and ensure availability despite the failure of one or more computers or the network, underpin all fault tolerant services.

While distributed systems are necessary for fault tolerance, they are more widely used: any webpage or application that requires Internet connectivity is implemented by combining program logic running on two (the client device, e.g., your phone, and the server) or more computers and thus is a distributed system. Similarly, large-scale computation, e.g., big data processing, ML training, etc., run on clusters are another example. While fault tolerance is not the goal for these distributed systems, they must also handle failures and deal with network delays and other problems, and require similar programs.

Our goals

Over this semester our main goal is to learn how to build and reason about distributed systems. To do so, we will look at a few different questions: (a) how do we model distributed systems and reason about their behavior, with a particular focus on the asynchronous model, a set of assumptions that are commonly made when describing and reasoning about distributed systems; (b) what does it mean for a distributed system to be correct, and how do we specify these correctness properties; (c) protocols to solve agreement problems, that is protocols that allow all computers (or processes) running to agree on a value; (d) Byzantine failures and how to deal with them; and (e) failure detectors, and how they help with distributed systems.

This class requires that you read papers: my aim in class is to cover gaps and put the paper in context, but the material assumes you have read the papers. While you can get a base understanding without reading the papers, you are likely to miss important details that are necessary to master this material.

We will also be using a set of programming assignments. These assignments require that you write code in Elixir, which many of you might be unfamiliar with. The first assignments is designed to help you gain familiarity, and the labs use a very small subset of Elixir's features and constructs. We use

[1] In fact, NASA's SIFT program, which was developed in 1978 and provided computerized air traffic control systems was one of the eary distributed systems.

this language because the resulting code closely resembles the pseudocode and algorithms you will encounter in the class. Please talk to the course staff if you need help with the language.