Distributed Computing: Fundamentals, Simulations and Advanced Topics Notes

Joshua Concon

CSC2221 Self-Study Preparation

Not entirely sure about pre-requisites, but I am reading this book with both CSCB63, CSCB58 and STAB52 completed, the complete name of the textbook is "Distributed Computing Fundamentals, Simulations, and Advanced Topics" by Hagit Attiya and Jennifer Welch.
Dr. Vassos Hadzilacos is helping walk me through is happening in this book, so these notes are not my own but knowledge imparted to me from both Dr. Hadzilacos and this textbook. If you find any problems in these notes, feel free to contact me at conconjoshua@gmail.com.

Contents

1 Chapter 1 - Introduction

The section starts with some introduction to distributed systems on what it is, which is a *collection of individual computing devices that can communicate with each other.* It then goes on to list a bunch of real-world applications of Distributed Computing, which I totally skipped since Distributed Systems and Big Data are currently very ”trendy” and in demand in the job market.

The one example of a distributed system that I really liked that really helped me understand what we're looking at is the concept of a computer. A computer is really a bunch of smaller parts running concurrently, communicating with each other to solve a larger task, such as, running your computer.

1.1 Theory of Distributed Computing

Basically the goal of distributed computing is, just as you probably have done in algorithms or data structures courses, identify fundamental problems, state them precisely, create algorithms to solve them, and prove the optimality and correctness of those algorithms, but instead of doing thing as they relate to algorithms or data structures, we will be doing them for distributed systems.

One thing you'll notice that is different however is that **there is no single accepted model of computation for distributed systems**. This is probably due to the major differences between distributed systems and implementation of how they communicate with one another, what kind of information do the systems share with each other, what kind of failures are tolerated, etc.

Another thing is that not only will you be interested in the time and space complexity of these algorithms, but now we will also have to worry about the complexity of communication costs between systems and the number of faulty vs. nonfaulty components.

Due to complications facing distributed systems, it is a lot easier to prove that some problems cannot be solved without a certain amount of some resource, similar to problems being NP-complete, these proofs are important to know which problems should not have any effort put into them to try to solve the problem. Luckily for us, if a problem cannot be solved under a particular set of assumptions, we can change the rules by proposing a weaker problem statement.

1.2 Relationship of Theory to Practice

Definition: Multiple-instruction multiple-data (MIMD)

Multiple-instruction multiple-data are separate hardware running on common software. They are called **tightly coupled** if they have shared memory, are also sometimes called *multiprocessors* in this case. They are called **loosely coupled** if they do not have shared memory.

Examples of MIMD machines include a collection of workstations on a local area network or a collection of processors on a switching network.

You'll notice that distributed computing has a lot of models and that these models do not translate to real-life architectures. Because of this, the chapters in this (not these notes) book are more focused on fundamental problems rather than models, indicating whether a choice of a model has an impact on solvability or complexity of a problem and simulations, showing what properties and similarities are shared between the models.

These are the three main models based on communication and synchrony that we will be looking at.

The Asynchronous Shared Memory Model Which applies to tightly coupled machines where processors do not get the clock signal.

The Asynchronous Message-Passing Model Which applies to loosely-coupled machines and to wide-area networks.

The Synchronous Message-Passing Model Which is an idealization of message-passing systems in which some timing information is known, such as an upper bounds on message delay.
Another important part about our models is the faults that are assumed to occur. Most of these notes will be centred around crash failures, in which we

define as when a processor ceases to operate at some point without warning, which is usually how components fail in real life.

We are also studying the **Byzantine Failure Model**, which include processors that do not have constraints. This is a very strong worst-case assumption that even covers the possibility of human intrusion.