PREFACE

The computational universe surrounding us is clearly quite different from that envisioned by the designers of the large mainframes of half a century ago. Even the subsequent most futuristic visions of *supercomputing* and of *parallel machines,* which have guided the research drive and absorbed the research funding for so many years, are far from today's computational realities.

These realities are characterized by the presence of communities of networked entities communicating with each other, cooperating toward common tasks or the solution of a shared problem, and acting autonomously and spontaneously. They are *distributed computing environments.*

It has been from the fields of network and of communication engineering that the seeds of what we now experience have germinated. The growth in understanding has occurred when computer scientists (initially very few) started to become aware of and study the computational issues connected with these new network-centric realities. The internet, the web, and the grids are just examples of these environments. Whether over wired or wireless media, whether by static or nomadic code, computing in such environments is inherently decentralized and distributed. To compute in distributed environments one must understand the basic principles, the fundamental properties, the available tools, and the inherent limitations.

This book focuses on the *algorithmics* of distributed computing; that is, on how to solve problems and perform tasks efficiently in a distributed computing environment. Because of the multiplicity and variety of distributed systems and networked environments and their widespread differences, this book does not focus on any single one of them. Rather it describes and employes a distributed computing *universe* that captures the nature and basic structure of those systems (e.g., distributed operating systems, data communication networks, distributed databases, transaction processing systems, etc.), allowing us to discard or ignore the system-specific details while identifying the general principles and techniques.

This universe consists of a finite collection of computational *entities* communicating by means of *messages* in order to achieve a common goal; for example, to perform a given task, to compute the solution to a problem, to satisfy a request either from the user (i.e., outside the environment) or from other entities. Although each entity is capable of performing computations, it is the collection

[1] Incredibly, the terms "distributed systems" and "distributed computing" have been for years highjacked and (ab)used to describe very limited systems and low-level solutions (e.g., client server) that have little to do with distributed computing.

of all these entities that together will solve the problem or ensure that the task is performed.

In this universe, to solve a problem, we must discover and design a *distributed algorithm* or *protocol* for those entities: A set of rules that specify what each entity has to do. The collective but autonomous execution of those rules, possibly without any supervision or synchronization, must enable the entities to perform the desired task to solve the problem.

In the design process, we must ensure both *correctness* (i.e., the protocol we design indeed solves the problem) and *efficiency* (i.e., the protocol we design has a "small" cost).

As the title says, this book is on the *Design and Analysis of Distributed Algorithms.* Its goal is to enable the reader to learn how to *design* protocols to solve problems in a distributed computing environment, not by listing the results but rather by teaching how they can be obtained. In addition to the "how" and "why" (necessary for problem solution, from basic building blocks to complex protocol design), it focuses on providing the *analytical tools* and skills necessary for complexity evaluation of designs.

There are several *levels* of use of the book. The book is primarily a senior-undergraduate and graduate textbook; it contains the material for two one-term courses or alternatively a full-year course on Distributed Algorithms and Protocols, Distributed Computing, Network Computing, or Special Topics in Algorithms. It covers the "distributed part" of a graduate course on Parallel and Distributed Computing (the chapters on Distributed Data, Routing, and Synchronous Computing, in particular), and it is the theoretical companion book for a course in Distributed Systems, Advanced Operating Systems, or Distributed Data Processing.

The book is written for the students from the students' point of view, and it follows closely a well defined teaching path and method (the "course") developed over the years; both the path and the method become apparent while reading and using the book. It also provides a self-contained, self-directed guide for system-protocol designers and for communication software and engineers and developers, as well as for researchers wanting to enter or just interested in the area; it enables hands-on, head-on, and in-depth acquisition of the material. In addition, it is a serious sourcebook and referencebook for investigators in distributed computing and related areas.

Unlike the other available textbooks on these subjects, the book is based on a very simple *fully reactive* computational model. From a learning point of view, this makes the explanations clearer and readers' comprehension easier. From a teaching point of view, this approach provides the instructor with a natural way to present otherwise difficult material and to guide the students through, step by step. The instructors themselves, if not already familiar-with the material or with the approach, can achieve proficiency quickly and easily.

All protocols in the textbook as well as those designed by the students as part of the exercises are immediately programmable. Hence, the subtleties of actual implementation can be employed to enhance the understanding of the theoretical

[2] An open source Java-based engine, *DisJ,* provides the execution and visualization environment for our reactive protocols.

design principles; furthermore, *experimental* analysis (e.g., performance evaluation and comparison) can be easily and usefully integrated in the coursework expanding the analytical tools.

The book is written so to require *no* prerequisites other than standard undergraduate knowledge of operating systems and of algorithms. Clearly, concurrent or prior knowledge of communication networks, distributed operating systems or distributed transaction systems would help the reader to ground the material of this course into some practical application context; however, none is necessary.

The book is structured into nine chapters of different lengths. Some are focused on a single problem, others on a class of problems. The structuring of the written material into chapters could have easily followed different lines. For example, the material of *election* and of *mutual exclusion* could have been grouped together in a chapter on *Distributed Control.* Indeed, these two topics can be taught one after the other: Although missing an introduction, this "hidden" chapter is present in a distributed way. An important "hidden" chapter is Chapter 10 on *Distributed Graph Algorithms* whose content is distributed throughout the book: *Spanning-Tree Construction* (Section 2.5), *Depth-First Traversal* (Section 2.3.1), *Breadth-First Spanning Tree* (Section 4.2.5), *Minimum-Cost Spanning Tree* (Section 3.8.1), *Shortest Paths* (Section 4.2.3), Centers and medians (Section 2.6), Cycle and Knot Detection (Section 8.2).

The suggested prerequisite structure of the chapters is shown in Figure 1. As suggested by the figure, the first three chapters should be covered sequentially and before the other material.

There are only two other prerequisite relationships. The relationship between *Synchronous Compution* (Chapter 6) and *Computing in Presence of Faults* (Chapter 7) is particular. The recommended sequencing is in fact the following: Sections 7.1–7.2 (providing the strong motivation for synchronous computing), Chapter 6 (describing fault-free synchronous computing) and the rest of Chapter 7 (dealing with fault-tolerant synchronous computing as well as other issues). The other suggested

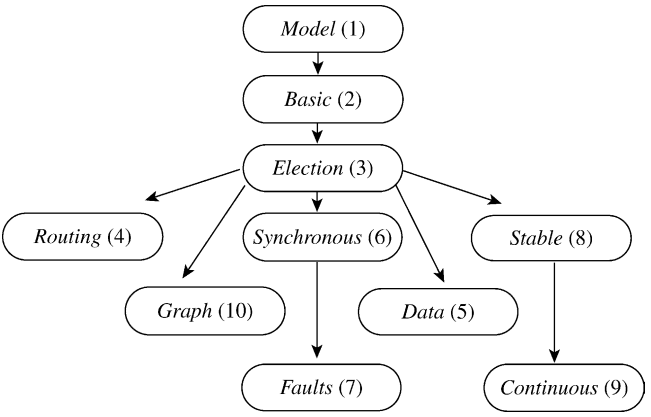

Figure 1: Prerequisite structure of the chapters.

prerequisite structure is that the topic of *Stable Properties* (Chapter 8) be handled before that of *Continuous Computations* (Chapter 9). Other than that, the sections can be mixed and matched depending on the instructor's preferences and interests. An interesting and popular sequence for a one-semester course is given by Chapters 1–6. A more conventional one-semester sequence is provided by Chapters 1–3 and 6–9.

The symbol ($\star$) after a section indicates noncore material. In connection with Exercises and Problems the symbol ($\star$) denotes difficulty (the more the symbols, the greater the difficulty).

Several important topics are not included in this edition of the book. In particular, this edition does not include algorithms on distributed coloring, on minimal independent sets, on self-stabilization, as well as on Sense of Direction. By design, this book does not include distributed computing in the *shared memory* model, focusing entirely on the message-passing paradigm.

This book has evolved from the teaching method and the material I have designed for the fourth-year undergraduate course *Introduction to Distributed Computing* and for the graduate course *Principles of Distributed Computing* at Carleton University over the last 20 years, and for the advanced graduate courses on *Distributed Algorithms* I have taught as part of the Advanced Summer School on Distributed Computing at the University of Siena over the last 10 years. I am most grateful to all the students of these courses: through their feedback they have helped me verify what works and what does not, shaping my teaching and thus the current structure of this book. Their keen interest and enthusiasm over the years have been the main reason for the existence of this book.

This book is very much work in progress. I would welcome any feedback that will make it grow and mature and change. Comments, criticisms, and reports on personal experience as a lecturer using the book, as a student studying it, or as a researcher glancing through it, suggestions for changes, and so forth: I am looking foreward to receiving any. Clearly, reports on typos, errors, and mistakes are very much appreciated. I tried to be accurate in giving credits; if you know of any omission or mistake in this regards, please let me know.

My own experience as well as that of my students leads to the inescapable conclusion that

distributed algorithms are fun

both to teach and to learn. I welcome you to share this experience, and I hope you will reach the same conclusion.

NICOLA SANTORO