Introduction to Reliable and Secure Distributed Programming

Christian Cachin • Rachid Guerraoui
Luís Rodrigues

Introduction to

Reliable and Secure Distributed Programming

Second Edition

Springer

Dr. Christian Cachin
IBM Research Zürich
Säumerstrasse 4
8803 Rüschlikon
Switzerland
cca@zurich.ibm.com

Prof. Luís Rodrigues
INESC-ID
Instituto Superior Técnico
Rua Alves Redol 9
1000-029 Lisboa
Portugal
ler@ist.utl.pt

Prof. Dr. Rachid Guerraoui
Ecole Polytechnique
Fédérale Lausanne (EPFL)
Fac. Informatique et Communications
Lab. Programmation Distribuée (LPD)
Station 14
1015 Lausanne
Bat. INR
Switzerland
Rachid.Guerraoui@epfl.ch

ISBN 978-3-642-15259-7 e-ISBN 978-3-642-15260-3
DOI 10.1007/978-3-642-15260-3
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2011921701

ACM Computing Classification (1998): C.2, F.2, G.2

Cover design: KuenkelLopka GmbH

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

To Irene, Philippe and André.
To Maria and Sarah.
To Hugo and Sara.

Preface

This book provides an introduction to distributed programming abstractions and presents the fundamental algorithms that implement them in several distributed environments. The reader is given insight into the important problems of distributed computing and the main algorithmic techniques used to solve these problems. Through examples the reader can learn how these methods can be applied to building distributed applications. The central theme of the book is the tolerance to uncertainty and adversarial influence in a distributed system, which may arise from network delays, faults, or even malicious attacks.

Content

In modern computing, a program usually encompasses *multiple processes*. A process is simply an abstraction that may represent a physical computer or a virtual one, a processor within a computer, or a specific thread of execution in a concurrent system. The fundamental problem with devising such distributed programs is to have all processes *cooperate* on some *common* task. Of course, traditional centralized algorithmic issues still need to be dealt with for each process individually. Distributed environments, which may range from a single computer to a data center or even a global system available around the clock, pose additional challenges: how to achieve a robust form of cooperation despite process failures, disconnections of some of the processes, or even malicious attacks on some processes? Distributed algorithms should be dependable, offer reliability and security, and have predictable behavior even under negative influence from the environment.

If no cooperation were required, a distributed program would simply consist of a set of independent centralized programs, each running on a specific process, and little benefit could be obtained from the availability of several processes in a distributed environment. It was the need for cooperation that revealed many of the fascinating problems addressed by this book, problems that need to be solved to make distributed computing a reality. The book not only introduces the reader to these problem statements, it also presents ways to solve them in different contexts.

Not surprisingly, distributed programming can be significantly simplified if the difficulty of robust cooperation is encapsulated within specific *abstractions*. By encapsulating all the tricky algorithmic issues, such distributed programming abstractions bridge the gap between network communication layers, which are

usually frugal in terms of dependability guarantees, and distributed application layers, which usually demand highly dependable primitives.

The book presents various distributed programming abstractions and describes algorithms that implement them. In a sense, we give the distributed application programmer a library of abstract interface specifications, and give the distributed system builder a library of algorithms that implement the specifications.

A significant amount of the preparation time for this book was devoted to formulating a collection of exercises and developing their solutions. We strongly encourage the reader to work out the exercises. We believe that no reasonable understanding can be achieved in a passive way. This is especially true in the field of distributed computing, where the human mind too often follows some attractive but misleading intuition. The book also includes the solutions for all exercises, to emphasize our intention to make them an integral part of the content. Many exercises are rather easy and can be discussed within an undergraduate teaching classroom. Other exercises are more difficult and need more time. These can typically be studied individually.

Presentation

The book as such is self-contained. This has been made possible because the field of distributed algorithms has reached a certain level of maturity, where distracting details can be abstracted away for reasoning about distributed algorithms. Such details include the behavior of the communication network, its various kinds of failures, as well as implementations of cryptographic primitives; all of them are treated in-depth by other works. Elementary knowledge about algorithms, first-order logic, programming languages, networking, security, and operating systems might be helpful. But we believe that most of our abstractions and algorithms can be understood with minimal knowledge about these notions.

The book follows an incremental approach and was primarily written as a textbook for teaching at the undergraduate or basic graduate level. It introduces the fundamental elements of distributed computing in an intuitive manner and builds sophisticated distributed programming abstractions from elementary ones in a modular way. Whenever we devise algorithms to implement a given abstraction, we consider a simple distributed-system model first, and then we revisit the algorithms in more challenging models. In other words, we first devise algorithms by making strong simplifying assumptions on the distributed environment, and then we discuss how to weaken those assumptions.

We have tried to balance intuition and presentation simplicity on the one hand with rigor on the other hand. Sometimes rigor was affected, and this might not have been always on purpose. The focus here is rather on abstraction specifications and algorithms, not on computability and complexity. Indeed, there is no theorem in this book. Correctness arguments are given with the aim of better understanding the algorithms: they are not formal correctness proofs per se.

Organization

The book has six chapters, grouped in two parts. The first part establishes the common ground:

- In Chapter 1, we *motivate* the need for distributed programming abstractions by discussing various applications that typically make use of such abstractions. The chapter also introduces the modular notation and the pseudo code used to describe the algorithms in the book.
- In Chapter 2, we present different kinds of *assumptions* about the underlying distributed environment. We introduce a family of distributed-system models for this purpose. Basically, a model describes the low-level abstractions on which more sophisticated ones are built. These include process and communication link abstractions. This chapter might be considered as a reference to other chapters.

The remaining four chapters make up the second part of the book. Each chapter is devoted to one problem, containing a broad class of related abstractions and various algorithms implementing them. We will go from the simpler abstractions to the more sophisticated ones:

- In Chapter 3, we introduce communication abstractions for distributed programming. They permit the *broadcasting* of a message to a group of processes and offer diverse reliability guarantees for delivering messages to the processes. For instance, we discuss how to make sure that a message delivered to one process is also delivered to all other processes, despite the crash of the original sender process.
- In Chapter 4, we discuss *shared memory* abstractions, which encapsulate simple forms of distributed storage objects, accessed by read and write operations. These could be files in a distributed storage system or registers in the memory of a multi-processor computer. We cover methods for reading and writing data values by clients, such that a value stored by a set of processes can later be retrieved, even if some of the processes crash, have erased the value, or report wrong data.
- In Chapter 5, we address the *consensus* abstraction through which a set of processes can decide on a common value, based on values that the processes initially propose. They must reach the same decision despite faulty processes, which may have crashed or may even actively try to prevent the others from reaching a common decision.
- In Chapter 6, we consider *variants of consensus*, which are obtained by extending or modifying the consensus abstraction according to the needs of important applications. This includes total-order broadcast, terminating reliable broadcast, (non-blocking) atomic commitment, group membership, and view-synchronous communication.

The distributed algorithms we study not only differ according to the actual abstraction they implement, but also according to the assumptions they make on the underlying distributed environment. We call the set of initial abstractions that an algorithm takes for granted a *distributed-system model*. Many aspects have a fundamental impact on how an algorithm is designed, such as the reliability of the links,

the degree of synchrony of the system, the severity of the failures, and whether a deterministic or a randomized solution is sought.

In several places throughout the book, the same basic distributed programming primitive is implemented in multiple distributed-system models. The intention behind this is two-fold: first, to create insight into the specific problems encountered in a particular system model, and second, to illustrate how the choice of a model affects the implementation of a primitive.

A detailed study of all chapters and the associated exercises constitutes a rich and thorough introduction to the field. Focusing on each chapter solely for the specifications of the abstractions and their underlying algorithms in their simplest form, i.e., for the simplest system model with crash failures only, would constitute a shorter, more elementary course. Such a course could provide a nice companion to a more practice-oriented course on distributed programming.

Changes Made for the Second Edition

This edition is a thoroughly revised version of the first edition. Most parts of the book have been updated. But the biggest change was to expand the scope of the book to a new dimension, addressing the key concept of *security against malicious actions*. Abstractions and algorithms in a model of distributed computing that allows adversarial attacks have become known as *Byzantine fault-tolerance*.

The first edition of the book was titled "Introduction to Reliable Distributed Programming." By adding one word ("secure") to the title – and adding one co-author – the evolution of the book reflects the developments in the field of distributed systems and in the real world. Since the first edition was published in 2006, it has become clear that most practical distributed systems are threatened by intrusions and that insiders cannot be ruled out as the source of malicious attacks. Building dependable distributed systems nowadays requires an interdisciplinary effort, with inputs from distributed algorithms, security, and other domains.

On the technical level, the syntax for modules and the names of some events have changed, in order to add more structure for presenting the algorithms. A module may now exist in multiple instances at the same time within an algorithm, and every instance is named by a unique identifier for this purpose. We believe that this has simplified the presentation of several important algorithms.

The first edition of this book contained a companion set of running examples implemented in the Java programming language, using the *Appia* protocol composition framework. The implementation addresses systems subject to crash failures and is available from the book's online website.

Online Resources

More information about the book, including the implementation of many protocols from the first edition, tutorial presentation material, classroom slides, and errata, is available online on the book's website at:

`http://distributedprogramming.net`

References

We have been exploring the world of distributed programming abstractions for almost two decades now. The material of this book has been influenced by many researchers in the field of distributed computing. A special mention is due to Leslie Lamport and Nancy Lynch for having posed fascinating problems in distributed computing, and to the *Cornell school* of reliable distributed computing, including Özalp Babaoglu, Ken Birman, Keith Marzullo, Robbert van Rennesse, Rick Schlichting, Fred Schneider, and Sam Toueg.

Many other researchers have directly or indirectly inspired the material of this book. We did our best to reference their work throughout the text. All chapters end with notes that give context information and historical references; our intention behind them is to provide hints for further reading, to trace the history of the presented concepts, as well as to give credit to the people who invented and worked out the concepts. At the end of the book, we reference books on other aspects of distributed computing for further reading.

Acknowledgments

We would like to express our deepest gratitude to our undergraduate and graduate students from the École Polytechnique Fédérale de Lausanne (EPFL) and the University of Lisboa (UL), for serving as reviewers of preliminary drafts of this book. Indeed, they had no choice and needed to prepare for their exams anyway! But they were indulgent toward the bugs and typos that could be found in earlier versions of the book as well as associated slides, and they provided us with useful feedback.

Partha Dutta, Corine Hari, Michal Kapalka, Petr Kouznetsov, Ron Levy, Maxime Monod, Bastian Pochon, and Jesper Spring, graduate students from the School of Computer and Communication Sciences of EPFL, Filipe Araújo and Hugo Miranda, graduate students from the Distributed Algorithms and Network Protocol (DIALNP) group at the Departamento de Informática da Faculdade de Ciências da Universidade de Lisboa (UL), Leila Khalil and Robert Basmadjian, graduate students from the Lebanese University in Beirut, as well as Ali Ghodsi, graduate student from the Swedish Institute of Computer Science (SICS) in Stockholm, suggested many improvements to the algorithms presented in the book.

Several implementations for the "hands-on" part of the book were developed by, or with the help of, Alexandre Pinto, a key member of the *Appia* team, complemented with inputs from several DIALNP team members and students, including Nuno Carvalho, Maria João Monteiro, and Luís Sardinha.

Finally, we would like to thank all our colleagues who were kind enough to comment on earlier drafts of this book. These include Felix Gaertner, Benoit Garbinato, and Maarten van Steen.

Acknowledgments for the Second Edition

Work on the second edition of this book started while Christian Cachin was on sabbatical leave from IBM Research at EPFL in 2009. We are grateful for the support of EPFL and IBM Research.

We thank again the students at EPFL and the University of Lisboa, who worked with the book, for improving the first edition. We extend our gratitude to the students at the Instituto Superior Técnico (IST) of the Universidade Técnica de Lisboa, at ETH Zürich, and at EPFL, who were exposed to preliminary drafts of the additional material included in the second edition, for their helpful feedback.

We are grateful to many attentive readers of the first edition and to those who commented on earlier drafts of the second edition, for pointing out problems and suggesting improvements. In particular, we thank Zinaida Benenson, Alysson Bessani, Diego Biurrun, Filipe Cristóvão, Dan Dobre, Felix Freiling, Ali Ghodsi, Seif Haridi, Matúš Harvan, Rüdiger Kapitza, Nikola Knežević, Andreas Knobel, Mihai Letia, Thomas Locher, Hein Meling, Hugo Miranda, Luís Pina, Martin Schaub, and Marko Vukolić.

Christian Cachin
Rachid Guerraoui
Luís Rodrigues

Contents