Michel Raynal

# Distributed Algorithms for Message-Passing Systems

Springer

# Distributed Algorithms for Message-Passing Systems

Michel Raynal

# Distributed Algorithms for Message-Passing Systems

Springer

Michel Raynal
Institut Universitaire de France
IRISA-ISTIC
Université de Rennes 1
Rennes Cedex
France

ISBN 978-3-642-38122-5 ISBN 978-3-642-38123-2 (eBook)
DOI 10.1007/978-3-642-38123-2
Springer Heidelberg New York Dordrecht London

Library of Congress Control Number: 2013942973

ACM Computing Classification (1998): F.1, D.1, B.3



Printed on acid-free paper

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

# Preface

*La profusion des choses cachait la rareté des idées et l'usure des croyances.*
*[. . . ] Retenir quelque chose du temps où l'on ne sera plus.*
In *Les années* (2008), Annie Ernaux

*Nel mezzo del cammin di nostra vita*
*Mi ritrovai per una selva oscura,*
*Ché la diritta via era smarritta.*
In *La divina commedia* (1307–1321), Dante Alighieri (1265–1321)

*Wir müssen nichts sein, sondern alles werden wollen.*
Johann Wolfgang von Goethe (1749–1832)

*Chaque génération, sans doute, se croit vouée à refaire le monde.*
*La mienne sait pourtant qu'elle ne le refera pas. Mais sa tâche est peut-être plus grande.*
*Elle consiste à empêcher que le monde ne se défasse.*
Speech at the Nobel Banquet, Stockholm, December 10, 1957, Albert Camus (1913–1960)

*Rien n'est précaire comme vivre*
*Rien comme être n'est passager*
*C'est un peu fondre pour le givre*
*Ou pour le vent être léger*
*J'arrive où je suis étranger.*
In *Le voyage de Hollande* (1965), Louis Aragon (1897–1982)

**What Is Distributed Computing?** Distributed computing was born in the late 1970s when researchers and practitioners started taking into account the intrinsic characteristic of physically distributed systems. The field then emerged as a specialized research area distinct from networking, operating systems, and parallel computing.

*Distributed computing* arises when one has to solve a problem in terms of distributed entities (usually called processors, nodes, processes, actors, agents, sensors, peers, etc.) such that each entity has only a partial knowledge of the many parameters involved in the problem that has to be solved. While parallel computing and real-time computing can be characterized, respectively, by the terms *efficiency* and *on-time computing*, distributed computing can be characterized by the term *uncertainty*. This uncertainty is created by asynchrony, multiplicity of control flows,

absence of shared memory and global time, failure, dynamicity, mobility, etc. Mastering one form or another of uncertainty is pervasive in all distributed computing problems. A main difficulty in designing distributed algorithms comes from the fact that each entity cooperating in the achievement of a common goal cannot have instantaneous knowledge of the current state of the other entities; it can only know their past local states.

Although distributed algorithms are often made up of a few lines, their behavior can be difficult to understand and their properties hard to state and prove. Hence, distributed computing is not only a fundamental topic but also a challenging topic where simplicity, elegance, and beauty are first-class citizens.

**Why This Book?** While there are a lot of books on sequential computing (both on basic data structures, or algorithms), this is not the case in distributed computing. Most books on distributed computing consider advanced topics where the uncertainty inherent to distributed computing is created by the net effect of asynchrony and failures. It follows that these books are more appropriate for graduate students than for undergraduate students.

The aim of this book is to present in a comprehensive way basic notions, concepts and algorithms of distributed computing when the distributed entities cooperate by sending and receiving messages on top of an underlying network. In this case, the main difficulty comes from the physical distribution of the entities and the asynchrony of the environment in which they evolve.

**Audience** This book has been written primarily for people who are not familiar with the topic and the concepts that are presented. These include mainly:

- Senior-level undergraduate students and graduate students in computer science or computer engineering, who are interested in the principles and foundations of distributed computing.
- Practitioners and engineers who want to be aware of the state-of-the-art concepts, basic principles, mechanisms, and techniques encountered in distributed computing.

Prerequisites for this book include undergraduate courses on algorithms, and basic knowledge on operating systems. Selections of chapters for undergraduate and graduate courses are suggested in the section titled “How to Use This Book” in the Afterword.

**Content** As already indicated, this book covers algorithms, basic principles, and foundations of message-passing programming, i.e., programs where the entities communicate by sending and receiving messages through a network. The world is distributed, and the algorithmic thinking suited to distributed applications and systems is not reducible to sequential computing. Knowledge of the bases of distributed computing is becoming more important than ever as more and more computer applications are now distributed. The book is composed of six parts.

- The aim of the first part, which is made up of six chapters, is to give a feel for the nature of distributed algorithms, i.e., what makes them different from sequential or parallel algorithms. To that end, it mainly considers distributed graph algorithms. In this context, each node of the graph is a process, which has to compute a result whose meaning depends on the whole graph.

  Basic distributed algorithms such as network traversals, shortest-path algorithms, vertex coloring, knot detection, etc., are first presented. Then, a general framework for distributed graph algorithms is introduced. A chapter is devoted to leader election algorithms on a ring network, and another chapter focuses on the navigation of a network by mobile objects.

- The second part is on the nature of distributed executions. It is made up of four chapters. In some sense, this part is the core of the book. It explains what a distributed execution is, the fundamental notion of a consistent global state, and the impossibility—without freezing the computation—of knowing whether a computed consistent global state has been passed through by the execution or not.

  Then, this part of the book addresses an important issue of distributed computations, namely the notion of logical time: scalar (linear) time, vector time, and matrix time. Each type of time is analyzed and examples of their uses are given. A chapter, which extends the notion of a global state, is then devoted to asynchronous distributed checkpointing. Finally, the last chapter of this part shows how to simulate a synchronous system on top of an asynchronous system (such simulators are called synchronizers).

- The third part of the book is made up of two chapters devoted to distributed mutual exclusion and distributed resource allocation. Different families of permission-based mutual exclusion algorithms are presented. The notion of an adaptive algorithm is also introduced. The notion of a critical section with multiple entries, and the case of resources with a single or several instances is also presented. Associated deadlock prevention techniques are introduced.

- The fourth part of the book is on the definition and the implementation of communication operations whose abstraction level is higher than the simple send/receive of messages. These communication abstractions impose order constraints on message deliveries. Causal message delivery and total order broadcast are first presented in one chapter. Then, another chapter considers synchronous communication (also called rendezvous or logically instantaneous communication).

- The fifth part of the book, which is made up of two chapters, is on the detection of stable properties encountered in distributed computing. A stable property is a property that, once true, remains true forever. The properties which are studied are the detection of the termination of a distributed computation, and the detection of distributed deadlock. This part of the book is strongly related to the second part (which is devoted to the notion of a global state).

- The sixth and last part of the book, which is also made up of two chapters, is devoted to the notion of a distributed shared memory. The aim is here to provide the entities (processes) with a set of objects that allow them to cooperate at

an abstraction level more appropriate than the use of messages. Two consistency conditions, which can be associated with these objects, are presented and investigated, namely, atomicity (also called linearizability) and sequential consistency. Several algorithms implementing these consistency conditions are described.

To have a more complete feeling of the spirit of this book, the reader is invited to consult the section "The Aim of This Book" in the Afterword, which describes what it is hoped has been learned from this book. Each chapter starts with a short presentation and a list of the main keywords, and terminates with a summary of its content. Each of the six parts of the book is also introduced by a brief description of its aim and its technical content.

**Acknowledgments** This book originates from lecture notes for undergraduate and graduate courses on distributed computing that I give at the University of Rennes (France) and, as an invited professor, at several universities all over the world. I would like to thank the students for their questions that, in one way or another, have contributed to this book. I want also to thank Ronan Nugent (Springer) for his support and his help in putting it all together.

Last but not least (and maybe most importantly), I also want to thank all the researchers whose results are presented in this book. Without their work, this book would not exist.

Michel Raynal

Professeur des Universités
Institut Universitaire de France
IRISA-ISTIC, Université de Rennes 1
Campus de Beaulieu, 35042, Rennes, France

March–October 2012
Rennes, Saint-Grégoire, Tokyo, Fukuoka (AINA'12), Arequipa (LATIN'12),
Reykjavik (SIROCCO'12), Palermo (CISIS'12), Madeira (PODC'12), Lisbon,
Douelle, Saint-Philibert, Rhodes Island (Europar'12),
Salvador de Bahia (DISC'12), Mexico City (Turing Year at UNAM)

# Contents

## Part VI Distributed Shared Memory

# Afterword

## The Aim of This Book

The practice of sequential computing has greatly benefited from the results of the theory of sequential computing that were captured in the study of formal languages and automata theory. Everyone knows what can be computed (computability) and what can be computed efficiently (complexity). All these results constitute the foundations of sequential computing, which, thanks to them, has become a *science*. These theoretical results and algorithmic principles have been described in many books from which students can learn basic results, algorithms, and principles of sequential computing (e.g., [99, 107, 148, 189, 205, 219, 258, 270, 351] to cite a few).

Since Lamport's seminal paper "*Time, clocks, and the ordering of events in a distributed system*", which appeared in 1978 [226], distributed computing is no longer a set of tricks or recipes, but a domain of computing science with its own concepts, methods, and applications. The world is distributed, and today the major part of applications are distributed. This means that message-passing algorithms are now an important part of any computing science or computing engineering curriculum.

Thanks to appropriate curricula—and good associated books—students have a good background in the theory and practice of sequential computing. In the same spirit, an aim of this book is to try to provide them with an appropriate background when they have to solve distributed computing problems.

Technology is what makes everyday life easier. Science is what allows us to transcend it, and capture the deep nature of the objects we are manipulating. To that end, it provides us with the right concepts to master and understand what we are doing. Considering failure-free asynchronous distributed computing, an ambition of this book is to be a step in this direction.

M. Raynal, *Distributed Algorithms for Message-Passing Systems*,
DOI 10.1007/978-3-642-38123-2, 

## Most Important Concepts, Notions, and Mechanisms Presented in This Book

Chapter 1: Asynchronous/synchronous system, breadth-first traversal, broadcast, convergecast, depth-first traversal, distributed algorithm, forward/discard principle, initial knowledge, local algorithm, parallel traversal, spanning tree, unidirectional logical ring.

Chapter 2: Distributed graph algorithm, cycle detection, graph coloring, knot detection, maximal independent set, problem reduction, shortest path computation.

Chapter 3: Cut vertex, de Bruijn's graph, determination of cut vertices, global function, message filtering, regular communication graph, round-based framework.

Chapter 4: Anonymous network, election, message complexity, process identity, ring network, time complexity, unidirectional versus bidirectional ring.

Chapter 5: Adaptive algorithm, distributed queuing, edge/link reversal, mobile object, mutual exclusion, network navigation, object consistency, routing, scalability, spanning tree, starvation-freedom, token.

Chapter 6: Event, causal dependence relation, causal future, causal path, causal past, concurrent (independent) events, causal precedence relation, consistent global state, cut, global state, happened before relation, lattice of global states, observation, marker message, nondeterminism, partial order on events, partial order on local states, process history, process local state, sequential observation.

Chapter 7: Adaptive communication layer, approximate causality relation, causal precedence, causality tracking, conjunction of stable local predicates, detection of a global state property, discarding old data, Hasse diagram, immediate predecessor, linear (scalar) time (clock), logical time, matrix time (clock), message stability, partial (total) order, relevant event, $k$-restricted vector clock, sequential observation, size of a vector clock, timestamp, time propagation, total order broadcast, vector time (clock).

Chapter 8: Causal path, causal precedence, communication-induced checkpointing, interval (of events), local checkpoint, forced local checkpoint, global checkpoint, hidden dependency, recovery, rollback-dependency trackability, scalar clock, spontaneous local checkpoint, uncoordinated checkpoint, useless checkpoint, vector clock, Z-dependence, zigzag cycle, zigzag pattern, zigzag path, zigzag prevention.

Chapter 9: Asynchronous system, bounded delay network, complexity, graph covering structure, physical clock drift, pulse-based programming, synchronizer, synchronous algorithm.

Chapter 10: Adaptive algorithm, arbiter permission, bounded algorithm, deadlock-freedom, directed acyclic graph, extended mutex, adaptive algorithm, grid quorum, individual permission, liveness property, mutual exclusion (mutex), preemption, quorum, readers/writers problem, safety property, starvation-freedom, timestamp, vote.

Chapter 11: Conflict graph, deadlock prevention, graph coloring, incremental requests, $k$-out-of-$M$ problem, permission, resource allocation, resource graph, resource type, resource instance, simultaneous requests, static/dynamic (resource) session, timestamp, total order, waiting chain, wait-for graph.

Chapter 12: Asynchronous system, bounded lifetime message, causal barrier, causal broadcast, causal message delivery order, circulating token, client/server broadcast, coordinator process, delivery condition, first in first out (FIFO) channel, order properties on a channel, size of control information, synchronous system.

Chapter 13: Asynchronous system, client-server hierarchy, communication initiative, communicating sequential processes, crown, deadline-constrained interaction, deterministic vs. nondeterministic context, logically instantaneous communication, planned vs. forced interaction, rendezvous, multiparty interaction, synchronous communication, synchronous system, token.

Chapter 14: AND receive, asynchronous system, atomic model, counting, diffusing computation, distributed iteration, global state, $k$-out-of-$n$ receive statement, loop invariant, message arrival vs. message reception, network traversal, nondeterministic statement, OR receive statement, reasoned construction, receive statement, ring, spanning tree, stable property, termination detection, wave.

Chapter 15: AND communication model, cycle, deadlock, deadlock detection, knot, one-at-a-time model, OR communication model, probe-based algorithm, resource vs. message, stable property, wait-for graph.

Chapter 16: Atomicity, composability, concurrent object, consistency condition, distributed shared memory, invalidation vs. update, linearizability, linearization point, local property, manager process, object operation, partial order on operations, read/write register, real time, sequential specification, server process, shared memory abstraction, total order broadcast abstraction.

Chapter 17: Causal consistency, concurrent object, consistency condition, distributed shared memory, invalidation, logical time, manager process, OO constraint, partial order on operations, read/write register, sequential consistency, server processes, shared memory abstraction, total order broadcast abstraction, WW constraint.

## How to Use This Book

This section presents two courses on distributed computing which can benefit from the concepts, algorithms and principles presented in this book. Each course is a one-semester course, and they are designed to be sequential (a full year at the undergraduate level, or split, with the first course at the undergraduate level and the second at the beginning of the graduate level).

- A first one-semester course on distributed computing could first focus on Part I, which is devoted to graph algorithms. Then, the course could address (a) distributed mutual exclusion (Chap. 10), (b) causal message delivery and total order

broadcast (Chap. 12), and (c) distributed termination detection (Chap. 14), if time permits.

The spirit of this course is to be an introductory course, giving students a correct intuition of what distributed algorithms are (they are not simple "extensions" of sequential algorithms), and show them that there are problems which are specific to distributed computing.

- A second one-semester course on distributed computing could first address the concept of a global state (Chap. 6). The aim is here to give the student a precise view of what a distributed execution is and introduce the notion of a global state. Then, the course could develop and illustrate the different notions of logical times (Chap. 7).

  Distributed checkpointing (Chap. 8), synchronizers (Chap. 9), resource allocation (Chap. 11), rendezvous communication (Chap. 13), and deadlock detection (Chap. 15), can be used to illustrate the previous notions.

  Finally, the meaning and the implementation of a distributed shared memory (Part VI) could be presented to introduce the notion of a consistency condition, which is a fundamental notion of distributed computing.

Of course, this book can also be used by engineers and researchers who work on distributed applications to better understand the concepts and mechanisms that underlie their work.

## From Failure-Free Systems to Failure-Prone Systems

This book was devoted to algorithms for failure-free asynchronous distributed applications and systems. Once the fundamental notions, concepts, and algorithms of failure-free distributed computing are mastered, one can focus on more specific topics of failure-prone distributed systems. In such a context, the combined effect of asynchrony and failures create *uncertainty* that algorithms have to cope with. The reader interested in the net effect of asynchrony and failure on the design of distributed algorithms is invited to consult the following books: [24, 67, 150, 155, 219, 242, 315, 316] (to cite a few).

## A Series of Books

This book completes a series of four books, written by the author, devoted to concurrent and distributed computing [315–317]. More precisely, we have the following.

- As has been seen, this book is on elementary distributed computing for *failure-free asynchronous* systems.
- The book [317] is on algorithms in *asynchronous* shared memory systems where processes can commit *crash failures*. It focuses on the construction of reliable concurrent objects in the presence of process crashes.

- The book [316] is on *asynchronous message-passing systems* where processes are prone to *crash failures*. It presents communication and agreement abstractions for fault-tolerant asynchronous distributed systems. Failure detectors are used to circumvent impossibility results encountered in pure asynchronous systems.
- The book [315] is on *synchronous* message-passing systems, where the processes are prone to *crash failures, omission failures, or Byzantine failures*. It focuses on the following distributed agreement problems: consensus, interactive consistency, and non-blocking atomic commit.

*Enseigner, c'est réfléchir à voix haute devant les étudiants*.
Henri-Léon Lebesgue (1875–1941)

*Make everything as simple as possible, but not simpler.*
Albert Einstein (1879–1955)