What can be Computed in a Distributed System?

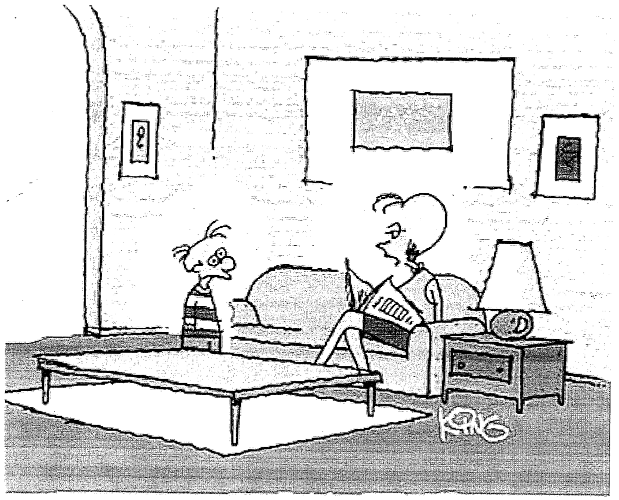
Michel Raynal

Institut Universitaire de France

& IRISA, Université de Rennes, France

& Hong Kong Polytechnic University (PolyU)

Never forget ....



"No, you weren't downloaded. Your were born."

Table of contents

On distributed computing

God made the bits,
all else is the work of man

God made the integers, all else is the work of man (L. Kronecker)

Distributed computing

- A birth certificate: *Time, clocks and the ordering of events in a distributed system*, Leslie Lamport, CACM 1978
- DC arises when one has to solve a problem in terms of entities (processes, agents, sensors, peers, actors, nodes, processors, ...) such that each entity has only a partial knowledge of the many parameters involved in the problem that has to be solved

What is distributed computing about?

- Real-time: masters On-time computing
- Parallelism: provides Efficiency
- Distributed computing:

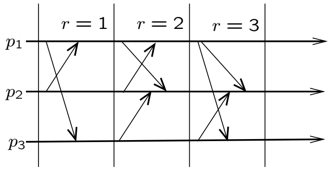
  masters Uncertainty

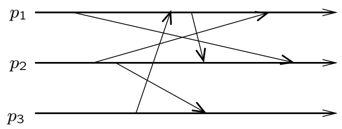
Distributed system

- n sequential deterministic processes: p_1, ..., p_n
- Communication:
  - ⋆ message-passing (send/receive) or
  - ⋆ read/write atomic registers
- Deterministic: the behavior of a process is entirely determined by its initial state, its algorithm, and
  - ⋆ the sequence of values read from atomic registers or
  - ⋆ the sequence of messages it receives

Synchronous vs asynchronous system

- Asynchronous:
  process speed and message transfer delays: arbitrary
- Synchronous
  - ⋆ Round-based computation
  - ⋆ A round is made up of three phases: send, receive, local computation
  - ⋆ A message sent during a round is received during the very same round





Process failure model

- Crash: unexpected halt
- t-resilient model
  Model parameter t= max # processes that may crash
- Wait-free model: $t = n - 1$

- Herlihy M.P., Wait-free synchronization. *ACM Transactions on Programming Languages and Systems*, 13(1):124-149, 1991

Textbooks:

- Attiya H. and Welch J.L., *Distributed computing: fundamentals, simulations and advanced topics*, Wiley-Interscience, 414 pages, 2004

- Lynch N.A., *Distributed algorithms*. Morgan Kaufmann, 872 pages, 1996.

- Raynal M., *Concurrent programming: algorithms, principles, and foundations*. Springer, 530 pages, 2013

Notion of an environment in DC

- Environment: set of failures and (a)synchrony patterns in which the system may evolve
- The system does not master its environment, it only suffers it
- This is a fundamental difference with sequential (or parallel) computing

Computability and complexity in DC

- Computability and complexity are the two lenses that allows us to understand and master computing
- In DC we have the following:

	Synchronous	Asynchronous
Failure-free	complexity	complexity
Crash-prone	complexity	computability

A (the?) fundamental issue

Are asynchronous crash-prone distributed systems universal?

On computability and universal constructions (1)

- In sequential computing, computability is understood through the Church-Turing's thesis (anything that can be computed, can computed by a Turing machine)
- The notion of Computability is intimately related to the notion of universality
- A fundamental issue of DC:

  Is it possible to design a universal construction (algorithm) on top of an asynchronous distributed system prone to crash failure?

On computability and universal constructions (2)

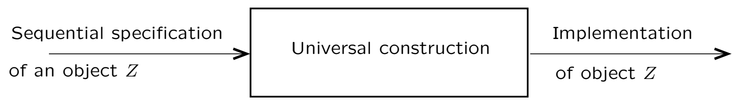
- Due to the environment (asynchrony and failures), distributed computability has a different flavor than sequential computability
- Moreover, this is independent of the fact that communication is by message-passing or read/write registers
- A famous quote of Leslie Lamport on distributed computing:

  A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable
- It follows that the limits of distributed computability reflect the difficulty of making decisions in the face of uncertainty, and has little to do with the computational power of each participant

-Herlihy M.P., Rajsbaum S., and Raynal M., Power and limits of distributed computing shared memory models. *Theoretical Computer Science*, 509:3-24, 2013

A universality notion for distributed computing

- Let Z be any concurrent object, which can be defined by a sequential specification on total operations
- From a practical point of view: Z is a service that we want to make reliable in the presence of failures
- The universality notion in which we are interested concerns the possibility to implement any such object Z despite asynchrony and any number of process crashes



- Herlihy M.P., Wait-free synchronization. *ACM Transactions on Programming Languages and Systems*, 13(1):124-149, 1991

Lamport L., Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, 1978

On the progress conditions of object Z

- The wait-freedom progress condition states that: An invocation of an operation on Z can fail to terminate only if the invoking process crashes

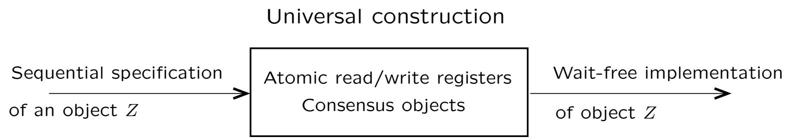
  Corresponding implementations are said "wait-free"
- A wait-free implementation prevents the use of locks!
- Non-blocking and obstruction-freedom are progress conditions weaker than wait-freedom

- Herlihy M.P., Luchangco V., and Moir M., Obstruction-free synchronization: double-ended queues as an example. *Proc. 23th Int'l IEEE Conference on Distributed Computing Systems (ICDCS'03)*, IEEE Press, pp. 522-529, 2003

- Herlihy M.P. and Wing J.M, Linearizability: a correctness condition for concurrent objects. *ACM Transactions on Programming Languages and Systems*, 12(3):463-492, 1990

Consensus-based universal constructions

- The consensus object is universal in the sense it allows the design of wait-free implementations of any object Z defined by a sequential specification



- Herlihy M.P., Wait-free synchronization. *ACM Transactions on Programming Languages and Systems*, 13(1):124-149, 1991

Consensus object

- A consensus object is a one-shot concurrent object that provides processes with a single operation denoted propose(v) where v is an input parameter (called "proposed value")
- A consensus object is defined by the following properties
  - ⋆ Validity. If a process decides a value, this value has been proposed by a process
  - ⋆ Agreement. No two processes decide different values
  - ⋆ Termination. An invocation of propose() by a process that does not crash terminates
- Consensus objects allow the processes to agree on the same sequence of operations applied to Z, despite any concurrency, asynchrony, and failure pattern

A fundamental result of distributed computability

- There is no deterministic algorithm that wait-free implements a consensus object
  - ⋆ Whatever the number of processes $n \geq 2$
  - ⋆ Whatever the communication medium (read/write registers or message-passing)
  - ⋆ Even if a single process may crash
  - ⋆ Even if processes have to agree on a single bit!

- Fischer M.J., Lynch N.A., and Paterson M.S., Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374-382, 1985

- Loui M. and Abu-Amara H., Memory requirements for agreement among unreliable asynchronous processes. *Advances in Comp. Research*, 4:163-183, JAI Press, 1987

Underlying intuition with binary consensus (1)

- Let a global state be 0-valent if only 0 can be decided from this state
- Let a global state be 1-valent if only 1 can be decided from this state
- Univalent state: 0-valent or 1-valent
- Bivalent state: any of 0 or 1 can still be decided "the dice are not yet cast"
- Decision step: carries the construction from a bivalent state to a univalent state

- Fischer M.J., Lynch N.A., and Paterson M.S., Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374-382, 1985

Underlying intuition with binary consensus (2)

- The impossibility theorem (FLP) is by contradiction
- It assumes there is an algorithm and shows that
  - ⋆ There is at least one initial bivalent state
  - ⋆ Among all possible executions there is at least one that makes the algorithm to always progress from a bivalent state to another bivalent state
- This shows that it is NOT always possible to break the non-determinism created by the environment

Sequential vs distributed computability

- A network of asynchronous Turing machines where even only one may crash, connected by a message-passing facility, or a read/write shared memory, is computationally less powerful than a single reliable Turing machine
- The nature of distributed computability issues is different from the nature of Turing's computability issues, namely, it is not related to the computational power of the individual participants

Enriching read/write systems with stronger objects

- Consensus number of an object X = largest n for which consensus can be be wait-free implemented in a read/write system of n processes enriched with objects X
  If there is no largest n, the consensus number is $+\infty$
- Herlihy's hierarchy:
  - ⋆ Consensus number 1: read/write atomic registers, ...
  - ⋆ Consensus number 2: test&set, swap, fetch&add, stack, queue, ...
  - ⋆ Consensus number $+\infty$: compare&swap, LL/SC, mem-to-mem swap, ...
- Any object with consensus number n is universal in a system of $\leq n$ processes

- Herlihy M.P., Wait-free synchronization. *ACM Transactions on Programming Languages and Systems*, 13(1):124-149, 1991

From read/write to message-passing systems

- Whatever the environment, it is possible to simulate message-passing on top of read/write
- It is impossible to simulate read/write on top of message-passing when $t \geq n/2$ (ABD impossibility)
- Intuition: indistinguishability argument
- A variant: CAP theorem
  - ⋆ CAP = Consistency, Availability, Partition-tolerance
  - ⋆ States that, when designing distributed services, it is impossible to design an algorithm that simultaneously ensures the three previous properties
  - ⋆ Impossibility variant of FLP + ABD

- Attiya H., Bar-Noy A. and Dolev D., Sharing memory robustly in message passing systems. *Journal of the ACM*, 42(1):121-132, 1995

- Brewer E.A., Pushing the CAP: strategies for consistency and availability. *IEEE Computer*, 45(2):23-29, 2012

Playing with progress conditions and consensus objects

- Obstruction-freedom: an invocation of an operation on an object is guaranteed to terminate when it executes alone for a "long enough period" (whatever the points at which the other invocations stopped)
- (y, x)-*liveness* if the object can be accessed by a subset of $y \leq n$ processes only, and wait-freedom is guaranteed for $x \leq y$ processes while obstruction-freedom is guaranteed for the remaining $y - x$ processes
- Impossibility to build an $(n, 1)$-live consensus object from read/write atomic registers and $(n-1, n-1)$-live consensus objects
- Another hierarchy: any (n, x)-live consensus object with $x < n$ has consensus number $x + 1$

- Imbs D., Raynal M., and Taubenfeld G., On asymmetric progress conditions. *Proc. 29rd ACM Symposium on Principles of Distributed Computing (PODC'10)*, ACM Press, pp. 55-64, 2010

How to circumvent consensus impossibility

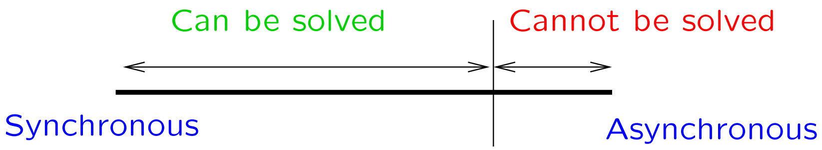
Remark:
No notion of objects with consensus number in MP systems

Three approaches

- Add an oracle
  (which provides additional computational power)
  - ⋆ Failure detectors
  - ⋆ Randomization
- Restrict the set of input vectors

The failure detector approach

- Given a problem: Find the weakest "assumptions" that has to added to an asynchronous system in order problems can be solved



Failure detectors

- Provide each process with a read-only local variable giving (possibly unreliable) information on failures
- Given a problem (object), give as few information as possible while allowing the object to be implemented
- According to the information on failures that is given, several "classes" of failure detectors can be defined

- Chandra T. and Toueg S., Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225-267, 1996

- Raynal M., *Communication and agreement abstractions for fault-tolerant asynchronous distributed systems.* Morgan & Claypool Pub., 251 pages, 2010

The weakest failure detector to solve consensus

- Ω: provides each process p_i with a read-only local variable $leader_i$ such that, after an unknown but finite time, the variables $leader_i$ of the non-crashed processes contain forever the same process identity of a non-crashed process
- Ω: weakest FD that allows consensus to be solved

- Chandra T.D., Hadzilacos V. and Toueg S., The Weakest Failure Detector for Solving Consensus. *Journal of the ACM*, 43(4):685-722, 1996

- Fernández A., Jiménez E., Raynal M., and Trédan G., A timing assumption and two t-resilient protocols for implementing an eventual leader service in asynchronous shared-memory systems. *Algorithmica*, 56(4):550-576, 2010

The notion of an indulgent distributed algorithm

- A distributed algorithm is indulgent with respect to a failure detector FD it uses to solve a problem Pb if
  - ⋆ it always guarantees the safety property defining Pb (i.e., whatever the correct/incorrect behavior of FD),
  - ⋆ and satisfies the liveness property associated with Pb at least when FD behaves correctly
- Hence, when the implementation of FD does not satisfies its specification, the algorithm may not terminate, but if it terminates its results are correct
- All Ω-based algorithms are indulgent
- Notions of stable vs unstable periods

- Guerraoui R., Indulgent algorithms. *Proc. 19th ACM Symposium on Principles of Distributed Computing (PODC'00)*, ACM Press, pp. 289-298, 2000

Randomization

- A classical way to break non-determinism
- Asynchronous round-based algorithms
- Requires to modify the termination property which becomes:

  The probability that a non-faulty process has decided by round r tends to 1, when the number of rounds tends to $+\infty$
- Notion of expected number of rounds to decide

- Ben-Or M., Another advantage of free choice: completely asynchronous agreement protocol. *Proc. 2d ACM Symposium on Principles of Distributed Computing (PODC'83)*, ACM Press, pp. 27-30, 1983

Restrict the set of input vectors

- Intuitively consider that an input vector "encodes" the value that has to be decided
- The consensus algorithm has to "decode" it
- To be possible the input vector has to satisfies some properties

- Friedman R., Mostéfaoui A., Rajsbaum S., and Raynal M., Asynchronous agreement and its relation with error-correcting codes. *IEEE Transactions on Computers*, 56(7):865-875, 2007

- Mostéfaoui A., Rajsbaum S., and Raynal M., Conditions on input vectors for consensus solvability in asynchronous distributed systems. *Journal of the ACM*, 50(6): 922-954, 2003

Examples of objects which can be solved in the basic read/write system

Snapshot object

- A snapshot object is an array of registers $A[1..n]$, where $A[i]$ can be written only by p_i
- It provides the processes with two operations
  - ⋆ $\forall i$: a write that allows p_i to write (only) in $A[i]$
  - ⋆ snapshot() can be invoked by any process and returns the value of the whole array
- These operations are atomic: each appears as if it been executed instantaneously at some point of the time line, between its start and its end events

- Afek Y., Attiya H., Dolev D., Gafni E., Merritt M., and Shavit N., Atomic snapshots of shared memory. *Journal of the ACM*, 40(4):873-890, 1993

One-shot M-renaming object (1)

- Each process p_i has an identity id_i taken from a large name space, whose size is N
- Initially a process knows only n and its initial identity id_i
- The aim is to allow processes to obtain new names in a new name space of size $M << N$
- The object provides processes with a single operation denoted `new_name`(id) where the input parameter is the identity of the invoking process; `new_name`$()$ returns a new name to the invoking process

- Attiya H., Bar-Noy A., Dolev D., Peleg D., and Reischuk R., Renaming in an asynchronous environment. *Journal of the ACM*, 37(3):524-548, 1990

One-shot M-renaming object (2)

- Validity. A new name is an integer in the set $[1..M]$.
- Agreement.
  No two processes obtain the same new name.
- Termination. If a process invokes `new_name`$()$ and does not crash, it eventually obtains a new name.

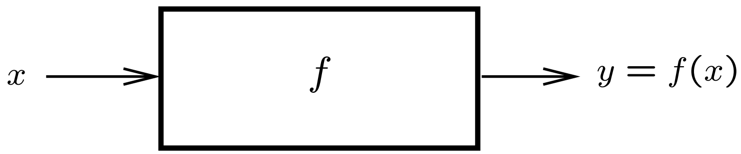
Let p be the number of processes that invoke `new_name`$()$.
$M = 2p - 1$ is a lower bound on the new name space

- Castañeda A. and Rajsbaum S., New combinatorial topology bounds for renaming: The upper bound. *Journal of the ACM*, 59(1), Article 3, 49 pages, 2012

- Herlihy M. and Shavit N., The topological structure of asynchronous computability. *Journal of the ACM*, 46(6):858-923, 1999

Sequential computing

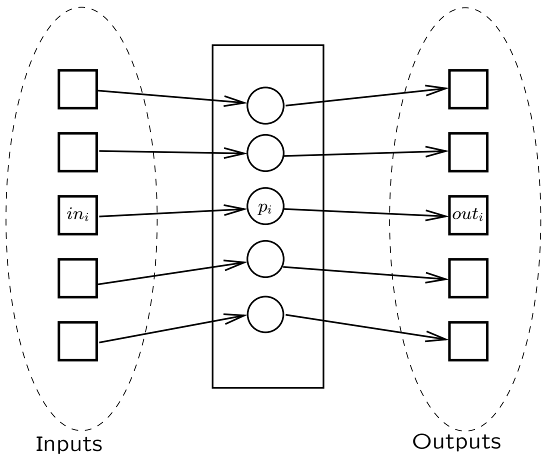
- Power and limit of sequential computing
- Central notion of a function: $y = f(x)$



- Notion of a computable function
- Several formalisms:
  Turing machine,
  Post system,
  Church's lambda calculus, etc.

The notion of a task in DC (1)

The DC counterpart of a function



The notion of a task in DC (2)

- A task T is a triple $(\mathcal{I}, \mathcal{O}, \Delta)$
  - $\star$ $\mathcal{I}$: set of input vectors (of size n)
  - $\star$ $\mathcal{O}$: set of output vectors (of size n)
  - $\star$ Δ: relation from $\mathcal{I}$ into $\mathcal{O}$: $\forall I \in \mathcal{I}: \Delta(I) \subseteq \mathcal{O}$
- $I[i]$: private input of p_i
- $O[i]$: private output of p_i
- $\forall I \in \mathcal{I}$:
  $\Delta(I) = \{$ output vectors that can be decided from I $\}$

Fundamental issues/results in asynchronous DC (1)

- Impossibility for a given process p_i to know if another process p_j has crashed or is only very slow (generates a lot of impossibilities)
- Due to the net effect of asynchrony and crashes the DC model is "weaker" than a Turing machine!
- There are Turing-computable functions that are not computable even in the presence of a single failure

  A lot of tasks cannot be solved in asynchronous crash-prone distributed systems while they can in a reliable distributed system

Fundamental issues/results in asynchronous DC (1)

- The question whether a task is 1-resilient computable can be reduced to a question of graph connectivity
- The question whether a task is computable in the presence of more failures:

  reducible to the question whether an associated geometric structure (called simplicial complex) has higher dimensional "holes", which is known to be undecidable
- Similar to oracles of classic computability, there are tasks which are computable only when given access to a distributed oracle for other tasks (leading to infinite hierarchies of tasks)

M. Herlihy, S. Rajsbaum, and M. Raynal, Power and limits of distributed computing shared memory models. *Theoretical Computer Science*, 509: 3-24 (2013)

A glance at synchronous systems

Complexity issues: gentle reminder

	Synchronous	Asynchronous
Failure-free	complexity	complexity
Crash-prone	complexity	computability

Asynchronous or synchronous failure-free systems

- Power = Turing machine
- Main issue: find the best solutions
- Example (cf. sorting pb)
  - ⋆ Leader election on a non-anonymous uni or bi-directional ring
  - ⋆ Message complexity: $O(n \log n)$
  - ⋆ Time complexity $O(\log n)$

- Dolev D., Klawe M., and Rodeh M., An $O(n \log n)$ unidirectional distributed algorithm for extrema finding in a circle. *Journal of Algorithms*, 3:245–260, 1982

- Higham L. and Przytycka T., A simple efficient algorithm for maximum finding on rings. *Information Processing Letters*, 58(6):319–324, 1996

Failure-prone synchronous systems

- Computability/Complexity results are similar to sequential computing
- Example: consensus problem:

Process failure model	Upper bound on t
crash failure	$t < n$
send omission failure	$t < n$
general omission failure	$t < n/2$
Byzantine failure	$t < n/3$

- In all cases: Lower bound on the number of rounds that the processes have to execute is $t + 1$

- Raynal M., *Fault-tolerant agreement in synchronous distributed systems.* Morgan & Claypool, 167 pages, 2010

Crash-prone synchr. systems with message adversaries

- Fully connected notwork
- Round-based computation
- A message adversary is a daemon which, at every round, is allowed to suppress messages
- No process knows in advance which are the links on which messages are suppressed during a round
- First introduced under the name mobile fault

- Santoro N. and Widmayer P., Time is not a healer. *Proc. 6th Annual Symposium on Theoretical Aspects of Computer Science (STACS'89)*, Springer LNCS 349, pp. 304-316, 1989.

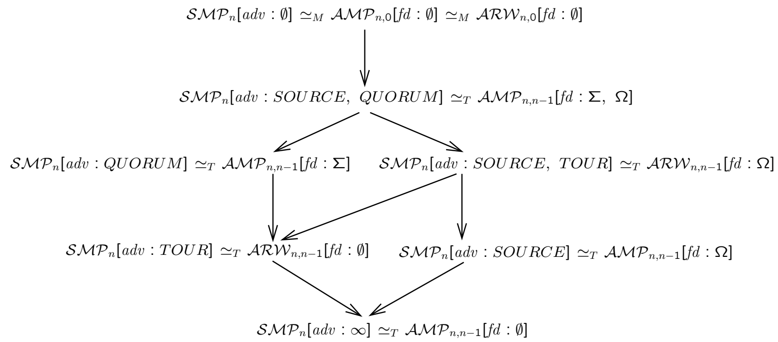
- Santoro N. and Widmayer P., Agreement in synchronous networks with ubiquitous faults. *Theoretical Computer Science*, 384(2-3): 232-249, 2007

The adversary TOUR

- The adversary TOUR is such that, in each round, and for each pair of processes (p_i, p_j), the adversary is allowed to suppress
  - ⋆ the message sent by p_i to p_j or the message sent by p_j to p_i
  - ⋆ but not both
- The synchronous message-passing model weakened by the message adversary TOUR and the asynchronous crash-prone read/write system model have the same computational power for distributed tasks

-Afek Y. and Gafni E., Asynchrony from synchrony. *Proc. Int'l Conference on Distributed Computing and Networking (ICDCN'13)*, Springer LNCS 7730, pp. 225-239, 2013

Message adversaries: a more global picture



- Raynal M. and Stainer J., Round-based synchrony weakened by message adversaries *vs* asynchrony enriched with failure detectors. *Proc. 33rd ACM Symposium on Principles of Distributed Computing (PODC '13)*, ACM Press, pp. 166-175, 2013

Conclusion

- The aim was to understand the power, subtleties and limits of crash-prone asynchronous distributed computing models
- A "Holy Grail" quest: have a view as clear as what we have in sequential computing wrt to computability, complexity, and languages hierarchy

The only slide to remember

Asynchrony and failures do modify

- Our view of synchronization
- The way synchronization has to be solved

The "other" only slide to remember!

Is there an end to the story?

Colorin colorado,

este cuento no se ha acabado ...

A few books on the topic

- Attiya H. and Welch J.L., *Distributed computing: fundamentals, simulations and advanced topics*,
Wiley-Interscience, 414 pages, 2004

- Herlihy M. and Shavit N., *The art of multiprocessor programming*.
Morgan Kaufmann, 508 pages, 2008

- Lynch N.A., *Distributed algorithms*. Morgan Kaufmann, 872 pages, 1996

- Raynal M., *Communication and agreement abstractions for fault-tolerant asynchronous distributed systems*.
Morgan & Claypool Pub., 251 pages, 2010

- Raynal M., *Fault-tolerant agreement in synchronous message-passing systems*.
Morgan & Claypool Publishers, 165 pages, 2010

- Raynal M., *Concurrent programming: algorithms, principles, and foundations*.
Springer, 530 pages, 2013

- Raynal M., *Distributed algorithms for message-passing systems*.
Springer, 515 pages, 2013

- Taubenfeld G., *Synchronization algorithms and concurrent programming*.
Pearson Education/Prentice Hall, 423 pages, 2006

Books