Distributed computing

From Wikipedia, the free encyclopedia

Distributed Computing is a field of computer science that studies distributed systems. A *distributed system* is a model in which components located on networked computers communicate and coordinate their actions by passing messages.[1] The components interact with each other in order to achieve a common goal. Three significant characteristics of distributed systems are: concurrency of components, lack of a global clock, and independent failure of components.[1] Examples of distributed systems vary from SOA-based systems to massively multiplayer online games to peer-to-peer applications.

A computer program that runs in a distributed system is called a **distributed program**, and distributed programming is the process of writing such programs.[2] There are many alternatives for the message passing mechanism, including pure HTTP, RPC-like connectors and message queues.

A goal and challenge pursued by some computer scientists and practitioners in distributed systems is location transparency; however, this goal has fallen out of favour in industry, as distributed systems are different from conventional non-distributed systems, and the differences, such as network partitions, partial system failures, and partial upgrades, cannot simply be "papered over" by attempts at "transparency" (see CAP theorem).

Distributed computing also refers to the use of distributed systems to solve computational problems. In *distributed computing*, a problem is divided into many tasks, each of which is solved by one or more computers,[3] which communicate with each other by message passing.[4]

Contents

Introduction

The word *distributed* in terms such as "distributed system", "distributed programming", and "distributed algorithm" originally referred to computer networks where individual computers were physically distributed within some geographical area.[5] The terms are nowadays used in a much wider sense, even referring to autonomous processes that run on the same physical computer and interact with each other by message passing.[4]

While there is no single definition of a distributed system,[6] the following defining properties are commonly used:

- There are several autonomous computational entities (*computers* or *nodes*), each of which has its own local memory.[7]
- The entities communicate with each other by message passing.[8]

A distributed system may have a common goal, such as solving a large computational problem;[9] the user then perceives the collection of autonomous processors as a unit. Alternatively, each computer may have its own user with individual needs, and the purpose of the distributed system is to coordinate the use of shared resources or provide communication services to the users.[10]

Other typical properties of distributed systems include the following:

- The system has to tolerate failures in individual computers.[11]
- The structure of the system (network topology, network latency, number of computers) is not known in advance, the system may consist of different kinds of computers and network links, and the system may change during the execution of a distributed program.[12]
- Each computer has only a limited, incomplete view of the system. Each computer may know only one part of the input.[13]

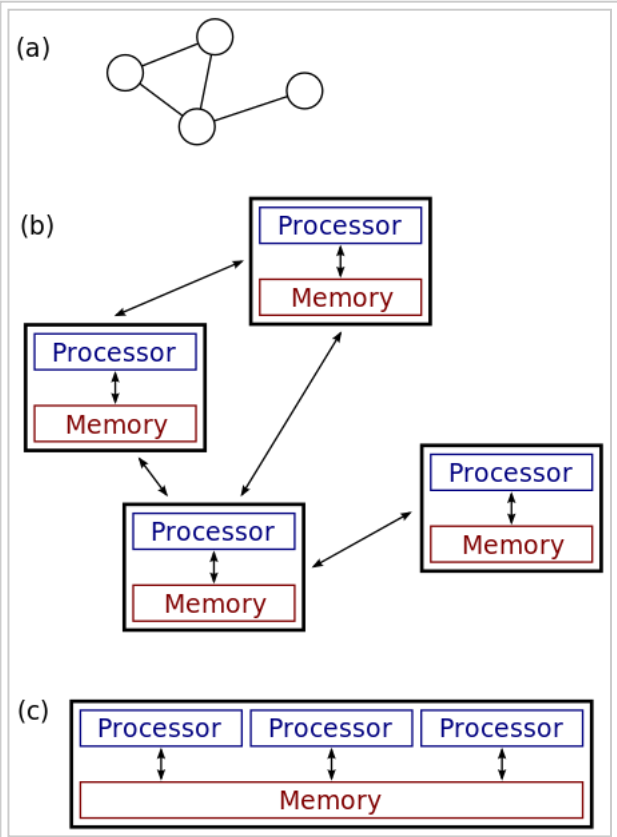
Parallel and distributed computing

Distributed systems are groups of networked computers, which have the same goal for their work. The terms "concurrent computing", "parallel computing", and "distributed computing" have a lot of overlap, and no clear distinction exists between them.[14] The same system may be characterized both as "parallel" and "distributed"; the processors in a typical distributed system run concurrently in parallel.[15] Parallel computing may be seen as a particular tightly coupled form of distributed computing,[16] and distributed computing may be seen as a loosely coupled form of parallel computing.[6] Nevertheless, it is possible to roughly classify concurrent systems as "parallel" or "distributed" using the following criteria:

- In parallel computing, all processors may have access to a shared memory to exchange information between processors.[17]
- In distributed computing, each processor has its own private memory (distributed memory). Information is exchanged by passing messages between the processors.[18]

The figure on the right illustrates the difference between distributed and parallel systems. Figure (a) is a schematic view of a typical distributed system; as usual, the system is represented as a network topology in which each node is a computer and each line connecting the nodes is a communication link. Figure (b) shows the same distributed system in more detail: each computer has its own local memory, and information can be exchanged only by passing messages from one node to another by using the available communication links. Figure (c) shows a parallel system in which each processor has a direct access to a shared memory.

The situation is further complicated by the traditional uses of the terms parallel and distributed *algorithm* that do not quite match the above definitions of parallel and distributed *systems* (see below for more detailed discussion). Nevertheless, as a rule of thumb, high-performance parallel computation in a shared-memory multiprocessor uses parallel algorithms while the coordination of a large-scale distributed system uses distributed algorithms.



(a), (b): a distributed system.
(c): a parallel system.

History

The use of concurrent processes that communicate by message-passing has its roots in operating system architectures studied in the 1960s.[19] The first widespread distributed systems were local-area networks such as Ethernet, which was invented in the 1970s.[20]

ARPANET, the predecessor of the Internet, was introduced in the late 1960s, and ARPANET e-mail was invented in the early 1970s. E-mail became the most successful application of ARPANET,[21] and it is probably the earliest example of a large-scale distributed application. In addition to ARPANET, and its successor, the Internet, other early worldwide computer networks included Usenet and FidoNet from the 1980s, both of which were used to support distributed discussion systems.

The study of distributed computing became its own branch of computer science in the late 1970s and early 1980s. The first conference in the field, Symposium on Principles of Distributed Computing (PODC), dates back to 1982, and its European counterpart International Symposium on Distributed Computing (DISC) was first held in 1985.

Architectures

Various hardware and software architectures are used for distributed computing. At a lower level, it is necessary to interconnect multiple CPUs with some sort of network, regardless of whether that network is printed onto a circuit board or made up of loosely coupled devices and cables. At a higher level, it is necessary to interconnect processes running on those CPUs with some sort of communication system.

Distributed programming typically falls into one of several basic architectures: client–server, three-tier, *n*-tier, or peer-to-peer; or categories: loose coupling, or tight coupling.

- Client–server: architectures where smart clients contact the server for data then format and display it to the users. Input at the client is committed back to the server when it represents a permanent change.
- Three-tier: architectures that move the client intelligence to a middle tier so that stateless clients can be used. This simplifies application deployment. Most web applications are three-tier.
- *n*-tier: architectures that refer typically to web applications which further forward their requests to other enterprise services. This type of application is the one most responsible for the success of application servers.
- Peer-to-peer: architectures where there are no special machines that provide a service or manage the network resources.[22]:227 Instead all responsibilities are uniformly divided among all machines, known as peers. Peers can serve both as clients and as servers.

Another basic aspect of distributed computing architecture is the method of communicating and coordinating work among concurrent processes. Through various message passing protocols, processes may communicate directly with one another, typically in a master/slave relationship. Alternatively, a "database-centric" architecture can enable distributed computing to be done without any form of direct inter-process communication, by utilizing a shared database.[23]

Applications

Reasons for using distributed systems and distributed computing may include:

1. The very nature of an application may *require* the use of a communication network that connects several computers: for example, data produced in one physical location and required in another location.
2. There are many cases in which the use of a single computer would be possible in principle, but the use of a distributed system is *beneficial* for practical reasons. For example, it may be more cost-efficient to obtain the desired level of performance by using a cluster of several low-end computers, in comparison with a single high-end computer. A distributed system can provide more reliability than a non-distributed system, as there is no single point of failure. Moreover, a distributed system may be easier to expand and manage than a monolithic uniprocessor system.[24]

Examples

Examples of distributed systems and applications of distributed computing include the following:[25]

- telecommunication networks:
  - telephone networks and cellular networks,
  - computer networks such as the Internet,
  - wireless sensor networks,
  - routing algorithms;
- network applications:
  - World Wide Web and peer-to-peer networks,
  - massively multiplayer online games and virtual reality communities,
  - distributed databases and distributed database management systems,
  - network file systems,
  - distributed information processing systems such as banking systems and airline reservation systems;
- real-time process control:
  - aircraft control systems,
  - industrial control systems;
- parallel computation:
  - scientific computing, including cluster computing and grid computing and various volunteer computing projects (see the list of distributed computing projects),
  - distributed rendering in computer graphics

Theoretical foundations

Models

Many tasks that we would like to automate by using a computer are of question–answer type: we would like to ask a question and the computer should produce an answer. In theoretical computer science, such tasks are called computational problems. Formally, a computational problem consists of *instances* together with a *solution* for each instance. Instances are questions that we can ask, and solutions are desired answers to these questions.

Theoretical computer science seeks to understand which computational problems can be solved by using a computer (computability theory) and how efficiently (computational complexity theory). Traditionally, it is said that a problem can be solved by using a computer if we can design an algorithm that produces a correct solution for any given instance. Such an algorithm can be implemented as a computer program that runs on a general-purpose computer: the program reads a problem instance from input, performs some computation, and produces the solution as output. Formalisms such as random access machines or universal Turing machines can be used as abstract models of a sequential general-purpose computer executing such an algorithm.

The field of concurrent and distributed computing studies similar questions in the case of either multiple computers, or a computer that executes a network of interacting processes: which computational problems can be solved in such a network and how efficiently? However, it is not at all obvious what is meant by "solving a problem" in the case of a concurrent or distributed system: for example, what is the task of the algorithm designer, and what is the concurrent or distributed equivalent of a sequential general-purpose computer?

The discussion below focuses on the case of multiple computers, although many of the issues are the same for concurrent processes running on a single computer.

Three viewpoints are commonly used:

Parallel algorithms in shared-memory model

- All processors have access to a shared memory. The algorithm designer chooses the program executed by each processor.
- One theoretical model is the parallel random access machines (PRAM) that are used.[26] However, the classical PRAM model assumes synchronous access to the shared memory.
- Shared-memory programs can be extended to distributed systems if the underlying operating system encapsulates the communication between nodes and virtually unifies the memory across all individual systems.
- A model that is closer to the behavior of real-world multiprocessor machines and takes into account the use of machine instructions, such as Compare-and-swap (CAS), is that of *asynchronous shared memory*. There is a wide body of work on this model, a summary of which can be found in the literature.[27][28]

Parallel algorithms in message-passing model

- The algorithm designer chooses the structure of the network, as well as the program executed by each computer.
- Models such as Boolean circuits and sorting networks are used.[29] A Boolean circuit can be seen as a computer network: each gate is a computer that runs an extremely simple computer program. Similarly, a sorting network can be seen as a computer network: each comparator is a computer.

Distributed algorithms in message-passing model

- The algorithm designer only chooses the computer program. All computers run the same program. The system must work correctly regardless of the structure of the network.
- A commonly used model is a graph with one finite-state machine per node.

In the case of distributed algorithms, computational problems are typically related to graphs. Often the graph that describes the structure of the computer network *is* the problem instance. This is illustrated in the following example.

An example

Consider the computational problem of finding a coloring of a given graph G. Different fields might take the following approaches:

Centralized algorithms

- The graph G is encoded as a string, and the string is given as input to a computer. The computer program finds a coloring of the graph, encodes the coloring as a string, and outputs the result.

Parallel algorithms

- Again, the graph G is encoded as a string. However, multiple computers can access the same string in parallel. Each computer might focus on one part of the graph and produce a coloring for that part.
- The main focus is on high-performance computation that exploits the processing power of multiple computers in parallel.

Distributed algorithms

- The graph G is the structure of the computer network. There is one computer for each node of G and one communication link for each edge of G. Initially, each computer only knows about its immediate neighbors in the graph G; the computers must exchange messages with each other to discover more about the structure of G. Each computer must produce its own color as output.
- The main focus is on coordinating the operation of an arbitrary distributed system.

While the field of parallel algorithms has a different focus than the field of distributed algorithms, there is a lot of interaction between the two fields. For example, the Cole–Vishkin algorithm for graph coloring[30] was originally presented as a parallel algorithm, but the same technique can also be used directly as a distributed algorithm.

Moreover, a parallel algorithm can be implemented either in a parallel system (using shared memory) or in a distributed system (using message passing).[31] The traditional boundary between parallel and distributed algorithms (choose a suitable network vs. run in any given network) does not lie in the same place as the boundary between parallel and distributed systems (shared memory vs. message passing).

Complexity measures

In parallel algorithms, yet another resource in addition to time and space is the number of computers. Indeed, often there is a trade-off between the running time and the number of computers: the problem can be solved faster if there are more computers running in parallel (see speedup). If a decision problem can be solved in polylogarithmic time by using a polynomial number of processors, then the problem is said to be in the class NC.[32] The class NC can be defined equally well by using the PRAM formalism or Boolean circuits—PRAM machines can simulate Boolean circuits efficiently and vice versa.[33]

In the analysis of distributed algorithms, more attention is usually paid on communication operations than computational steps. Perhaps the simplest model of distributed computing is a synchronous system where all nodes operate in a lockstep fashion. During each *communication round*, all nodes in parallel (1) receive the latest messages from their neighbours, (2) perform arbitrary local computation, and (3) send new messages to their neighbours. In such systems, a central complexity measure is the number of synchronous communication rounds required to complete the task.[34]

This complexity measure is closely related to the diameter of the network. Let D be the diameter of the network. On the one hand, any computable problem can be solved trivially in a synchronous distributed system in approximately $2D$ communication rounds: simply gather all information in one location (D rounds), solve the problem, and inform each node about the solution (D rounds).

On the other hand, if the running time of the algorithm is much smaller than D communication rounds, then the nodes in the network must produce their output without having the possibility to obtain information about distant parts of the network. In other words, the nodes must make globally consistent decisions based on information that is available in their *local neighbourhood*. Many distributed algorithms are known with the running time much smaller than D rounds, and understanding which problems can be solved by such algorithms is one of the central research questions of the field.[35]

Other commonly used measures are the total number of bits transmitted in the network (cf. communication complexity).

Other problems

Traditional computational problems take the perspective that we ask a question, a computer (or a distributed system) processes the question for a while, and then produces an answer and stops. However, there are also problems where we do not want the system to ever stop. Examples of such problems include the dining philosophers problem and other similar mutual exclusion problems. In these problems, the distributed system is supposed to continuously coordinate the use of shared resources so that no conflicts or deadlocks occur.

There are also fundamental challenges that are unique to distributed computing. The first example is challenges that are related to *fault-tolerance*. Examples of related problems include consensus problems,[36] Byzantine fault tolerance,[37] and self-stabilisation.[38]

A lot of research is also focused on understanding the *asynchronous* nature of distributed systems:

- Synchronizers can be used to run synchronous algorithms in asynchronous systems.[39]
- Logical clocks provide a causal happened-before ordering of events.[40]
- Clock synchronization algorithms provide globally consistent physical time stamps.[41]

Properties of distributed systems

So far the focus has been on *designing* a distributed system that solves a given problem. A complementary research problem is *studying* the properties of a given distributed system.

The halting problem is an analogous example from the field of centralised computation: we are given a computer program and the task is to decide whether it halts or runs forever. The halting problem is undecidable in the general case, and naturally understanding the behaviour of a computer network is at least as hard as understanding the behaviour of one computer.

However, there are many interesting special cases that are decidable. In particular, it is possible to reason about the behaviour of a network of finite-state machines. One example is telling whether a given network of interacting (asynchronous and non-deterministic) finite-state machines can reach a deadlock. This problem is PSPACE-complete,[42] i.e., it is decidable, but it is not likely that there is an efficient (centralised, parallel or distributed) algorithm that solves the problem in the case of large networks.

Coordinator election

Coordinator election (sometimes called *leader election*) is the process of designating a single process as the organizer of some task distributed among several computers (nodes). Before the task is begun, all network nodes are either unaware which node will serve as the "coordinator" (or leader) of the task, or unable to communicate

with the current coordinator. After a coordinator election algorithm has been run, however, each node throughout the network recognizes a particular, unique node as the task coordinator.

The network nodes communicate among themselves in order to decide which of them will get into the "coordinator" state. For that, they need some method in order to break the symmetry among them. For example, if each node has unique and comparable identities, then the nodes can compare their identities, and decide that the node with the highest identity is the coordinator.

The definition of this problem is often attributed to LeLann, who formalized it as a method to create a new token in a token ring network in which the token has been lost.

Coordinator election algorithms are designed to be economical in terms of total bytes transmitted, and time. The algorithm suggested by Gallager, Humblet, and Spira [43] for general undirected graphs has had a strong impact on the design of distributed algorithms in general, and won the Dijkstra Prize for an influential paper in distributed computing.

Many other algorithms were suggested for different kind of network graphs, such as undirected rings, unidirectional rings, complete graphs, grids, directed Euler graphs, and others. A general method that decouples the issue of the graph family from the design of the coordinator election algorithm was suggested by Korach, Kutten, and Moran.[44]

In order to perform coordination, distributed systems employ the concept of coordinators. The coordinator election problem is to choose a process from among a group of processes on different processors in a distributed system to act as the central coordinator. Several central coordinator election algorithms exist.[45]

See also

- AppScale
- BOINC
- Blockchain (database)
- Code mobility
- Decentralized computing
- Dew computing
- Distributed algorithmic mechanism design
- Distributed cache
- Distributed operating system
- Edsger W. Dijkstra Prize in Distributed Computing
- Fog computing
- Folding@home
- Inferno
- Jungle computing
- Layered queueing network
- Library Oriented Architecture - LOA
- List of distributed computing conferences
- List of distributed computing projects
- List of important publications in concurrent, parallel, and distributed computing
- Parallel distributed processing
- Parallel programming model
- Plan 9 from Bell Labs

Notes

1. Coulouris, George; Jean Dollimore; Tim Kindberg; Gordon Blair (2011). *Distributed Systems: Concepts and Design (5th Edition)*. Boston: Addison-Wesley. ISBN 0-132-14301-1.
2. Andrews (2000). Dolev (2000). Ghosh (2007), p. 10.
3. Godfrey (2002).
4. Andrews (2000), p. 291–292. Dolev (2000), p. 5.
5. Lynch (1996), p. 1.
6. Ghosh (2007), p. 10.
7. Andrews (2000), p. 8–9, 291. Dolev (2000), p. 5. Ghosh (2007), p. 3. Lynch (1996), p. xix, 1. Peleg (2000), p. xv.
8. Andrews (2000), p. 291. Ghosh (2007), p. 3. Peleg (2000), p. 4.
9. Ghosh (2007), p. 3–4. Peleg (2000), p. 1.
10. Ghosh (2007), p. 4. Peleg (2000), p. 2.
11. Ghosh (2007), p. 4, 8. Lynch (1996), p. 2–3. Peleg (2000), p. 4.
12. Lynch (1996), p. 2. Peleg (2000), p. 1.
13. Ghosh (2007), p. 7. Lynch (1996), p. xix, 2. Peleg (2000), p. 4.
14. Ghosh (2007), p. 10. Keidar (2008).
15. Lynch (1996), p. xix, 1–2. Peleg (2000), p. 1.
16. Peleg (2000), p. 1.
17. Papadimitriou (1994), Chapter 15. Keidar (2008).
18. See references in Introduction.
19. Andrews (2000), p. 348.
20. Andrews (2000), p. 32.
21. Peter (2004), The history of email (http://www.nethistory.info/History%20of%20the%20Internet/email.html).
22. Vigna P, Casey MJ. *The Age of Cryptocurrency: How Bitcoin and the Blockchain Are Challenging the Global Economic Order* St. Martin's Press January 27, 2015 ISBN 9781250065636
23. Lind P, Alm M (2006), "A database-centric virtual chemistry system", *J Chem Inf Model*, **46** (3): 1034–9, doi:10.1021/ci050360b, PMID 16711722.
24. Elmasri & Navathe (2000), Section 24.1.2.
25. Andrews (2000), p. 10–11. Ghosh (2007), p. 4–6. Lynch (1996), p. xix, 1. Peleg (2000), p. xv. Elmasri & Navathe (2000), Section 24.
26. Cormen, Leiserson & Rivest (1990), Section 30.
27. Herlihy & Shavit (2008), Chapters 2-6.
28. Lynch (1996)
29. Cormen, Leiserson & Rivest (1990), Sections 28 and 29.
30. Cole & Vishkin (1986). Cormen, Leiserson & Rivest (1990), Section 30.5.
31. Andrews (2000), p. ix.
32. Arora & Barak (2009), Section 6.7. Papadimitriou (1994), Section 15.3.
33. Papadimitriou (1994), Section 15.2.
34. Lynch (1996), p. 17–23.
35. Peleg (2000), Sections 2.3 and 7. Linial (1992). Naor & Stockmeyer (1995).
36. Lynch (1996), Sections 5–7. Ghosh (2007), Chapter 13.
37. Lynch (1996), p. 99–102. Ghosh (2007), p. 192–193.
38. Dolev (2000). Ghosh (2007), Chapter 17.
39. Lynch (1996), Section 16. Peleg (2000), Section 6.
40. Lynch (1996), Section 18. Ghosh (2007), Sections 6.2–6.3.
41. Ghosh (2007), Section 6.4.
42. Papadimitriou (1994), Section 19.3.
43. R. G. Gallager, P. A. Humblet, and P. M. Spira (January 1983). "A Distributed Algorithm for Minimum-Weight Spanning Trees" (PDF). *ACM Transactions on Programming Languages and Systems*. **5** (1): 66–77. doi:10.1145/357195.357200.
44. Ephraim Korach, Shay Kutten, Shlomo Moran (1990). "A Modular Technique for the Design of Efficient Distributed Leader Finding Algorithms". *ACM Transactions on Programming Languages and Systems*. **12** (1): 84–101. doi:10.1145/77606.77610.
45. Hamilton, Howard. "Distributed Algorithms". Retrieved 2013-03-03.

References

Books

- Andrews, Gregory R. (2000), *Foundations of Multithreaded, Parallel, and Distributed Programming*, Addison–Wesley, ISBN 0-201-35752-6.
- Arora, Sanjeev; Barak, Boaz (2009), *Computational Complexity – A Modern Approach*, Cambridge, ISBN 978-0-521-42426-4.
- Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L. (1990), *Introduction to Algorithms* (1st ed.), MIT Press, ISBN 0-262-03141-8.
- Dolev, Shlomi (2000), *Self-Stabilization*, MIT Press, ISBN 0-262-04178-2.
- Elmasri, Ramez; Navathe, Shamkant B. (2000), *Fundamentals of Database Systems* (3rd ed.), Addison–Wesley, ISBN 0-201-54263-3.
- Ghosh, Sukumar (2007), *Distributed Systems – An Algorithmic Approach*, Chapman & Hall/CRC, ISBN 978-1-58488-564-1.
- Lynch, Nancy A. (1996), *Distributed Algorithms*, Morgan Kaufmann, ISBN 1-55860-348-4.
- Herlihy, Maurice P.; Shavit, Nir N. (2008), *The Art of Multiprocessor Programming*, Morgan Kaufmann, ISBN 0-12-370591-6.
- Papadimitriou, Christos H. (1994), *Computational Complexity*, Addison–Wesley, ISBN 0-201-53082-1.
- Peleg, David (2000), *Distributed Computing: A Locality-Sensitive Approach*, SIAM, ISBN 0-89871-464-8.

Articles

- Cole, Richard; Vishkin, Uzi (1986), "Deterministic coin tossing with applications to optimal parallel list ranking", *Information and Control*, **70** (1): 32–53, doi:10.1016/S0019-9958(86)80023-7.
- Keidar, Idit (2008), "Distributed computing column 32 – The year in review", *ACM SIGACT News*, **39** (4): 53–54, doi:10.1145/1466390.1466402.
- Linial, Nathan (1992), "Locality in distributed graph algorithms", *SIAM Journal on Computing*, **21** (1): 193–201, doi:10.1137/0221015.
- Naor, Moni; Stockmeyer, Larry (1995), "What can be computed locally?", *SIAM Journal on Computing*, **24** (6): 1259–1277, doi:10.1137/S0097539793254571.

Web sites

- Godfrey, Bill (2002). "A primer on distributed computing".
- Peter, Ian (2004). "Ian Peter's History of the Internet". Retrieved 2009-08-04.

Further reading

Books

- Attiya, Hagit and Jennifer Welch (2004), *Distributed Computing: Fundamentals, Simulations, and Advanced Topics*, Wiley-Interscience ISBN 0-471-45324-2.
- Christian Cachin; Rachid Guerraoui; Luís Rodrigues (2011), *Introduction to Reliable and Secure Distributed Programming* (2. ed.), Springer, ISBN 978-3-642-15259-7
- Coulouris, George; et al. (2011), *Distributed Systems: Concepts and Design (5th Edition)*, Addison-Wesley ISBN 0-132-14301-1.
- Faber, Jim (1998), *Java Distributed Computing*, O'Reilly: Java Distributed Computing by Jim Faber, 1998 (http://docstore.mik.ua/orelly/java-ent/dist/index.htm)
- Garg, Vijay K. (2002), *Elements of Distributed Computing*, Wiley-IEEE Press ISBN 0-471-03600-5.
- Tel, Gerard (1994), *Introduction to Distributed Algorithms*, Cambridge University Press
- Chandy, Mani; et al., *Parallel Program Design*

Articles

- Keidar, Idit; Rajsbaum, Sergio, eds. (2000–2009), "Distributed computing column", *ACM SIGACT News*.

- Birrell, A. D.; Levin, R.; Schroeder, M. D.; Needham, R. M. (April 1982). "Grapevine: An exercise in distributed computing" (PDF). *Communications of the ACM*. **25** (4): 260–274. doi:10.1145/358468.358487.

Conference Papers

- C. Rodríguez, M. Villagra and B. Barán, Asynchronous team algorithms for Boolean Satisfiability (https://dx.doi.org/10.1109/BIMNICS.2007.4610083), Bionetics2007, pp. 66–69, 2007.

External links

- Distributed computing (https://www.dmoz.org/Computers/Computer_Science/Distributed_Computing/) at DMOZ
- Distributed computing journals (https://www.dmoz.org/Computers/Computer_Science/Distributed_Computing/Publications/) at DMOZ

Wikimedia Commons has media related to ***Distributed computing***.

Categories: Distributed computing
