Distributed Algorithms For Message Passing Systems

This dissertation, "A Practical Distributed Garbage Collection Algorithm for Message Passing Network With Message Delay" by ???, Chun-tak, Kwan, was obtained from The University of Hong Kong (Pokfulam, Hong Kong) and is being sold pursuant to Creative Commons: Attribution 3.0 Hong Kong License. The content of this dissertation has not been altered in any way. We have altered the formatting in order to facilitate the ease of printing and reading of the dissertation. All rights not granted by the above license are retained by the author. DOI: 10.5353/th_b3121315 Subjects: Memory management (Computer science) Computer networks Computer algorithms

Both authors have taught the course of “Distributed Systems” for many years in the respective schools. During the teaching, we feel strongly that “Distributed systems” have evolved from traditional “LAN” based distributed systems towards “Internet based” systems. Although there exist many excellent textbooks on this topic, because of the fast development of distributed systems and network programming/protocols, we have difficulty in finding an appropriate textbook for the course of “distributed systems” with orientation to the requirement of the undergraduate level study for today’s distributed technology. Specifically, from - to-date concepts, algorithms, and models to implementations for both distributed system designs and application programming. Thus the philosophy behind this book is to integrate the concepts, algorithm designs and implementations of distributed systems based on network programming. After using several materials of other textbooks and research books, we found that many texts treat the distributed systems with separation of concepts, algorithm design and network programming and it is very difficult for students to map the concepts of distributed systems to the algorithm design, prototyping and implementations. This book intends to enable readers, especially postgraduates and senior undergraduate level, to study up-to-date concepts, algorithms and network programming skills for building modern distributed systems. It enables students not only to master the concepts of distributed network system but also to readily use the material introduced into implementation practices.

Designing distributed computing systems is a complex process requiring a solid understanding of the design problems and the theoretical and practical aspects of their solutions. This comprehensive textbook covers the fundamental principles and models underlying the theory, algorithms and systems aspects of distributed computing. Broad and detailed coverage of the theory is balanced with practical systems-related issues such as mutual exclusion, deadlock detection, authentication, and failure recovery. Algorithms are carefully selected, lucidly presented, and described without complex proofs. Simple explanations and illustrations are used to elucidate the algorithms. Important emerging topics such as peer-to-peer networks and network security are also considered. With vital algorithms, numerous illustrations, examples and homework problems, this textbook is suitable for advanced undergraduate and graduate students of electrical and computer engineering and computer science. Practitioners in data networking and sensor networks will also find this a valuable resource. Additional resources are available online at www.cambridge.org/9780521876346.

This volume contains the proceedings from the workshops held in conjunction with the IEEE International Parallel and Distributed Processing Symposium, IPDPS 2000, on 1-5 May 2000 in Cancun, Mexico. The workshopsprovidea forum for bringing together researchers,practiti-ers, and designers from various backgrounds to discuss the state of the art in parallelism.Theyfocusondi erentaspectsofparallelism,fromruntimesystems to formal methods, from optics to irregular problems, from biology to networks of personal computers, from embedded systems to programming environments; the following workshops are represented in this volume: { Workshop on Personal Computer Based Networks of Workstations { Workshop on Advances in Parallel and Distributed Computational Models { Workshop on Par. and Dist. Comp. in Image, Video, and Multimedia { Workshop on High-Level Parallel Prog. Models and Supportive Env. { Workshop on High Performance Data Mining { Workshop on Solving Irregularly Structured Problems in Parallel { Workshop on Java for Parallel and Distributed Computing { WorkshoponBiologicallyInspiredSolutionsto ParallelProcessingProblems { Workshop on Parallel and Distributed Real-Time Systems { Workshop on Embedded HPC Systems and Applications { Recon gurable Architectures Workshop { Workshop on Formal Methods for Parallel Programming { Workshop on Optics and Computer Science { Workshop on Run-Time Systems for Parallel Programming { Workshop on Fault-Tolerant Parallel and Distributed Systems All papers published in the workshops proceedings were selected by the p- gram committee on the basis of referee reports. Each paper was reviewed by independent referees who judged the papers for originality, quality, and cons- tency with the themes of the workshops.

Fundamentals and Recent Developments

A Locality-Sensitive Approach

Fault-Tolerant Message-Passing Distributed Systems

Elements of Distributed Computing

Impossibility Results for Distributed Computing

In Distributed Algorithms, Nancy Lynch provides a blueprint for designing, implementing, and analyzing distributed algorithms. She directs her book at a wide audience, including students, programmers, system designers, and researchers. Distributed Algorithms contains the most significant algorithms and impossibility results in the area, all in a simple automata-theoretic setting. The algorithms are proved correct, and their complexity is analyzed according to precisely defined complexity measures. The problems covered include resource allocation, communication, consensus among distributed processes, data consistency, deadlock detection, leader election, global snapshots, and many others. The material is organized according to the system model—first by the timing model and then by the interprocess communication mechanism. The material on system models is isolated in separate chapters for easy reference. The presentation is completely rigorous, yet is intuitive enough for immediate comprehension. This book familiarizes readers with important problems, algorithms, and impossibility results in the area: readers can then recognize the problems when they arise in practice, apply the algorithms to solve them, and use the impossibility results to determine whether problems are unsolvable. The book also provides readers with the basic mathematical tools for designing new algorithms and proving new impossibility results. In addition, it teaches readers how to reason carefully about distributed algorithms—to model them formally, devise precise specifications for their required behavior, prove their correctness, and evaluate their performance with realistic measures.

Surveys the theory and history of the alternating direction method of multipliers, and discusses its applications to a wide variety of statistical and machine learning problems of recent interest, including the lasso, sparse logistic regression, basis pursuit, covariance selection, support vector machines, and many others.

AN ELABORATE YET BEGINNER-FRIENDLY GUIDE TO DISTRIBUTED ALGORITHMS Distributed Algorithms, a non-trivial and highly evolving field of active research, is often presented in most publications using a heavy accompaniment of mathematical techniques and notations. Aimed squarely at beginners as well as experienced practitioners, this book attempts to demystify and explicate the subject of distributed algorithms using a highly expansive and verbose style of treatment. Covering scores of landmark algorithms in the field of distributed computing, the approach is to present and analyse each topic using a minimum of mathematical exposition, reverting instead to a fluid style of description in plain English. A mathematical presentation is avoided altogether whenever such a move does not reduce the quality of the analysis at hand. Elsewhere, the effort always is to talk and guide the reader through the relevant math without resorting to a series of equations. To backup such a style of treatment, each topic is accompanied by a multitude of examples, flowcharts, and diagrams. The book is divided into three parts; the first part deals with fundamentals, the second and largest of the three is all about algorithms specific to message passing networks, while the last one focuses on shared memory algorithms. The beginning of the book dedicates a few chapters to the basics - including a quick orientation on the underlying platform, i.e. distributed systems, their characteristics, advantages, challenges, and so on. Some of the earlier chapters also address basic algorithms and techniques relevant to distributed computing environments before moving on to progressively complex algorithms and results - en route to the later chapters in the second part which deal with widely used 'industrial-strength' protocols such as Paxos and Raft. The third part of the book does assume a basic orientation towards computer programming, and presents numerous shared memory algorithms where each one is accompanied by a detailed description, analysis, pseudo code, and in some cases, code (C or C++). Whenever actual code is used, the syntax is kept as basic as possible - incorporating only elementary features of the language - so that newbie programmers can follow the presentation smoothly. Lastly, the target audience of the book is wide enough to cover beginners such as students or graduates joining the industry, experienced professionals wishing to migrate from monolithic frameworks to distributed ones, as well as readers with years of experience on the subject of distributed computing. The style of presentation is selected with the first two classes of readers in mind: those who wish to quickly ramp up on the subject of distributed algorithms for professional reasons or personal ones. While staying true to the stated aim, the book does not shy away from dealing with complex topics. A concise list of content information follows: Introduction to distributed systems Properties of distributed data stores and Brewer's theorem Building blocks: unicast, broadcast, algorithms in cubes Leader election algorithms: for ring/generic networks Consensus algorithms: synchronous/asynchronous variants for message passing and shared memory systems Distributed commits, Paxos, Raft Graph algorithms Routing algorithms Time and order Mutual exclusion: for message passing networks Debug algorithms: snapshot, deadlock/termination detection Shared memory: practical problems, mutual exclusion, consensus, resource allocation About the author Fourré Sigs is an industry veteran with over 25 years of experience in systems programming, networking, and highly scalable and secure distributed service architectures.

An Introduction to Distributed Algorithms takes up some of the main concepts and algorithms, ranging from basic to advanced techniques and applications, that underlie the programming of distributed-memory systems such as computer networks, networks of workstations, and multiprocessors. Written from the broad perspective of distributed-memory systems in general it includes topics such as algorithms for maximum flow, program debugging, and simulation that do not appear in more orthodox texts on distributed algorithms. Moving from fundamentals to advances and applications, ten chapters—with exercises and bibliographic notes—cover a variety of topics. These include models of distributed computation, information propagation, leader election, distributed snapshots, network synchronization, self- stability, termination detection, deadlock detection, graph algorithms, mutual exclusion, program debugging, and simulation. All of the algorithms are presented in a clear, template- based format for the description of message-passing computations among the nodes of a connected graph. Such a generic setting allows the treatment of problems originating from many different application areas. The main ideas and algorithms are described in a way that balances intuition and formal rigor—most are preceded by a general intuitive discussion and followed by formal statements as to correctness complexity or other properties.

From Concepts to Implementations

Introduction to Reliable and Secure Distributed Programming

Concurrent and Distributed Computing in Java

Distributed Algorithms

Self-stabilization

This book presents the most important fault-tolerant distributed programming abstractions and their associated distributed algorithms, in particular in terms of reliable communication and agreement, which lie at the heart of nearly all distributed applications. These programming abstractions, distributed objects or services, allow software designers and programmers to cope with asynchrony and the most important types of failures such as process crashes, message losses, and malicious behaviors of computing entities, widely known under the term "Byzantine fault-tolerance". The author introduces these notions in an incremental manner, starting from a clear specification, followed by algorithms which are first described intuitively and then proved correct. The book also presents impossibility results in classic distributed computing models, along with strategies, mainly failure detectors and randomization, that allow us to enrich these models. In this sense, the book constitutes an introduction to the science of distributed computing, with applications in all domains of distributed systems, such as cloud computing and blockchains. Each chapter comes with exercises and bibliographic notes to help the reader approach, understand, and master the fascinating field of fault-tolerant distributed computing.

A comprehensive guide to distributed algorithms that emphasizes examples and exercises rather than mathematical argumentation. This book offers students and researchers a guide to distributed algorithms that emphasizes examples and exercises rather than the intricacies of mathematical models. It avoids mathematical argumentation, often a stumbling block for students, teaching algorithmic thought rather than proofs and logic. This approach allows the student to learn a large number of algorithms within a relatively short span of time. Algorithms are explained through brief, informal descriptions, illuminating examples, and practical exercises. The examples and exercises allow readers to understand algorithms intuitively and from different perspectives. Proof sketches, arguing the correctness of an algorithm or explaining the idea behind fundamental results, are also included. An appendix offers pseudocode descriptions of many algorithms. Distributed algorithms are performed by a collection of computers that send messages to each other or by multiple software threads that use the same shared memory. The algorithms presented in the book are for the most part “classics,” selected because they shed light on the algorithmic design of distributed systems or on key issues in distributed computing and concurrent programming. Distributed Algorithms can be used in courses for upper-level undergraduates or graduate students in computer science, or as a reference for researchers in the field.

The objective of our monograph is to cover the developments on the theoretical foundations of distributed symmetry breaking in the message-passing model. We hope that our monograph will stimulate further progress in this exciting area.

Multi-Threaded Object-Oriented MPI-Based Message Passing Interface: The ARCH Library presents ARCH, a library built as an extension to MPI. ARCH relies on a small set of programming abstractions that allow the writing of well-structured multi-threaded parallel codes according to the object-oriented programming style. ARCH has been written with C++. The book describes the built-in classes, and illustrates their use through several template application cases in several fields of interest: Distributed Algorithms (global completion detection, distributed process serialization), Parallel Combinatorial Optimization (A* procedure), Parallel Image-Processing (segmentation by region growing). It shows how new application-level distributed data types - such as a distributed tree and a distributed graph - can be derived from the built-in classes. A feature of interest to readers is that both the library and the application codes used for illustration purposes are available via the Internet. The material can be downloaded for installation and personal parallel code development on the reader's computer system. ARCH can be run on Unix/Linux as well as Windows NT-based platforms. Current installations include the IBM-SP2, the CRAY-T3E, the Intel Paragon, PC-networks under Linux or Windows NT. Multi-Threaded Object-Oriented MPI-Based Message Passing Interface: The ARCH Library is aimed at scientists who need to implement parallel/distributed algorithms requiring complicated local and/or distributed control structures. It can also benefit parallel/distributed program developers who wish to write codes in the object-oriented style. The author has been using ARCH for several years as a medium to teach parallel and network programming. Teachers can employ the library for the same purpose while students can use it for training. Although ARCH has been used so far in an academic environment, it will be an effective tool for professionals as well. Multi-Threaded Object-Oriented MPI-Based Message Passing Interface: The ARCH Library is suitable as a secondary text for a graduate level course on Data Communications and Networks, Programming Languages, Algorithms and Computational Theory and Distributed Computing and as a reference for researchers and practitioners in industry.

Fault-tolerant Agreement in Synchronous Message-passing Systems

15 IPDPS 2000 Workshops Cancun, Mexico, May 1–5, 2000 Proceedings

An Introduction to Distributed Algorithms

Distributed Programming

Distributed Graph Coloring

Distributed Computing Through Combinatorial Topology describes techniques for analyzing distributed algorithms based on award winning combinatorial topology research. The authors present a solid theoretical foundation relevant to many real systems reliant on parallelism with unpredictable delays, such as multicore microprocessors, wireless networks, distributed systems, and Internet protocols. Today, a new student or researcher must assemble a collection of scattered conference publications, which are typically terse and commonly use different notations and terminologies. This book provides a self-contained explanation of the mathematics to readers with computer science backgrounds, as well as explaining computer science concepts to readers with backgrounds in applied mathematics. The first section presents mathematical notions and models, including message passing and shared-memory systems, failures, and timing models. The next section presents core concepts in two chapters each: first, proving a simple result that lends itself to examples and pictures that will build up readers' intuition; then generalizing the concept to prove a more sophisticated result. The overall result weaves together and develops the basic concepts of the field, presenting them in a gradual and intuitively appealing way. The book's final section discusses advanced topics typically found in a graduate-level course for those who wish to explore further. Named a 2013 Notable Computer Book for Computing Methodologies by Computing Reviews Gathers knowledge otherwise spread across research and conference papers using consistent notations and a standard approach to facilitate understanding Presents unique insights applicable to multiple computing fields, including multicore microprocessors, wireless networks, distributed systems, and Internet protocols Synthesizes and distills material into a simple, unified presentation with examples, illustrations, and exercises

This book constitutes the thoroughly refereed post-conference proceedings of 12 workshops held at the 21st International Conference on Parallel and Distributed Computing, Euro-Par 2015, in Vienna, Austria, in August 2015. The 67 revised full papers presented were carefully reviewed and selected from 121 submissions. The volume includes papers from the following workshops: BigDataCloud: 4th Workshop on Big Data Management in Clouds - Euro-EDUPAR: First European Workshop on Parallel and Distributed Computing Education for Undergraduate Students - Hetero Par: 13th International Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Platforms - LSDVE: Third Workshop on Large Scale Distributed Virtual Environments - OMHI: 4th International Workshop on On-chip Memory Hierarchies and Interconnects - PADAPS: Third Workshop on Parallel and Distributed Agent-Based Simulations - PELGA: Workshop on Performance Engineering for Large-Scale Graph Analytics - REPPAR: Second International Workshop on Reproducibility in Parallel Computing - Resilience: 8th Workshop on Resiliency in High Performance Computing in Clusters, Clouds, and Grids - ROME: Third Workshop on Runtime and Operating Systems for the Many Core Era - UCHPC: 8th Workshop on UnConventional High Performance Computing - and VHPC: 10th Workshop on Virtualization in High-Performance Cloud Computing.

Understanding distributed computing is not an easy task. This is due to the many facets of uncertainty one has to cope with and master in order to produce correct distributed software. Considering the uncertainty created by asynchrony and process crash failures in the context of message-passing systems, the book focuses on the main abstractions that one has to understand and master in order to be able to produce software with guaranteed properties. These fundamental abstractions are communication abstractions that allow the processes to communicate consistently (namely the register abstraction and the reliable broadcast abstraction), and the consensus agreement abstractions that allows them to cooperate despite failures. As they give a precise meaning to the words "communicate" and "agree" despite asynchrony and failures, these abstractions allow distributed programs to be designed with properties that can be stated and proved. Impossibility results are associated with these abstractions. Hence, in order to circumvent these impossibilities, the book relies on the failure detector approach, and, consequently, that approach to fault-tolerance is central to the book. Table of Contents: List of Figures / The Atomic Register Abstraction / Implementing an Atomic Register in a Crash-Prone Asynchronous System / The Uniform Reliable Broadcast Abstraction / Uniform Reliable Broadcast Abstraction Despite Unreliable Channels / The Consensus Abstraction / Consensus Algorithms for Asynchronous Systems Enriched with Various Failure Detectors / Constructing Failure Detectors

This is a presentation of several approaches for employing shared memory abstraction in distributed systems, a powerful tool for simplifying the design and implementation of software systems for networked platforms. These approaches enable system designers to work with abstract readable and writable objects without the need to deal with the complexity and dynamism of the underlying platform. The key property of shared memory implementations is the consistency guarantee that it provides under concurrent access to the shared objects. The most intuitive memory consistency model is atomicity because of its equivalence with a memory system where accesses occur serially, one at a time. Emulations of shared atomic memory in distributed systems is an active area of research and development. The problem proves to be challenging, and especially so in distributed message passing settings with unreliable components, as is often the case in networked systems. Several examples are provided for implementing shared memory services with the help of replication on top of message-passing distributed platforms subject to a variety of perturbations in the computing medium.

PRAC DISTRIBUTED GARBAGE COLL

Nonsequential and Distributed Programming with Go

Concurrent Crash-Prone Shared Memory Systems

Concurrent Programming: Algorithms, Principles, and Foundations

Euro-Par 2015 International Workshops, Vienna, Austria, August 24-25, 2015, Revised Selected Papers

Distributed Programming: Theory and Practice presents a practical and rigorous method to develop distributed programs that correctly implement their specifications. The method also covers how to write specifications and how to use them. Numerous examples such as bounded buffers, distributed locks, message-passing services, and distributed termination detection illustrate the method. Larger examples include data transfer protocols, distributed shared memory, and TCP network sockets. Distributed Programming: Theory and Practice bridges the gap between books that focus on specific concurrent programming languages and books that focus on distributed algorithms. Programs are written in a "real-life" programming notation, along the lines of Java and Python with explicit instantiation of threads and programs. Students and programmers will see these as programs and not "merely" algorithms in pseudo-code. The programs implement interesting algorithms and solve problems that are large enough to serve as projects in programming classes and software engineering classes. Exercises and examples are included at the end of each chapter with on-line access to the solutions. Distributed Programming: Theory and Practice is designed as an advanced-level text book for students in computer science and electrical engineering. Programmers, software engineers and researchers working in this field will also find this book useful.

Future requirements for computing speed, system reliability, and cost-effectiveness entail the development of alternative computers to replace the traditional von Neumann organization. As computing networks come into being, one of the latest dreams is now possible - distributed computing. Distributed computing brings transparent access to as much computer power and data as the user needs for accomplishing any given task - simultaneously achieving high performance and reliability. The subject of distributed computing is diverse, and many researchers are investigating various issues concerning the structure of hardware and the design of distributed software. Distributed System Design defines a distributed system as one that looks to its users like an ordinary system, but runs on a set of autonomous processing elements (PEs) where each PE has a separate physical memory space and the message transmission delay is not negligible. With close cooperation among these PEs, the system supports an arbitrary number of processes and dynamic extensions. Distributed System Design outlines the main motivations for building a distributed system, including: inherently distributed applications performance/cost resource sharing flexibility and extendibility availability and fault tolerance scalability Presenting basic concepts, problems, and possible solutions, this reference serves graduate students in distributed system design as well as computer professionals analyzing and designing distributed/open/parallel systems. Chapters discuss: the scope of distributed computing systems general distributed programming languages and a CSP-like distributed control description language (DCDL) expressing parallelism, interprocess communication and synchronization, and fault-tolerant design two approaches describing a distributed system: the time-space view and the interleaving view mutual exclusion and related issues, including election, bidding, and self-stabilization prevention and detection of deadlock reliability, safety, and security as well as various methods of handling node, communication, Byzantine, and software faults efficient interprocessor communication mechanisms as well as these mechanisms without specific constraints, such as adaptiveness, deadlock-freedom, and fault-tolerance virtual channels and virtual networks load distribution problems synchronization of access to shared data while supporting a high degree of concurrency

Distributed Algorithms for Message-Passing SystemsSpringer Science & Business Media

Gives a thorough exposition of network spanners and other locality-preserving network representations such as sparse covers and partitions.

10th International Workshop, WDAG '96, Bologna, Italy, October 9 - 11, 1996. Proceedings

Synchronization of Concurrent Processes: Communication - Cooperation - Competition

A Verbose Tour

An Intuitive Approach

Theory is what remains true when technology is changing. So, it is important to know and master the basic concepts and the theoretical tools that underlie the design of the systems we are using today and the systems we will use tomorrow. This means that, given a computing model, we need to know what can be done and what cannot be done in that model. Considering systems built on top of an asynchronous read/write shared memory p this monograph presents and develops the fundamental notions that are universal constructions, consensus numbers, distributed recursivity, power of the BG simulation, and what can be done when one has to cope with process anonymity and/or memory anonymity. Numerous distributed algorithms are presented, the aim of which is being to help the reader better understand the power and the subtleties of the notions that are presented. I can appreciate the simplicity and beauty of some of these algorithms.

To understand the power of distributed systems, it is necessary to understand their inherent limitations: what problems cannot be solved in particular systems, or without sufficient resources (such as time or space). This book presents key techniques for proving such impossibility results and applies them to a variety of different problems in a variety of different system models. Insights gained from these results are highlighted, aspects of a p difficult are isolated, features of an architecture that make it inadequate for solving certain problems efficiently are identified, and different system models are compared. Table of Contents: Acknowledgments / Introduction / Indistinguishability / Shifting and Scaling / Scenario Arguments / Information Theory Arguments / Covering Arguments / Valency Arguments / Combinatorial Arguments / Reductions and Simulations / Bibliography / Author

The present book focuses on the way to cope with the uncertainty created by process failures (crash, omission failures and Byzantine behavior) in synchronous message-passing systems (i.e., systems whose progress is governed by the passage of time). To that end, the book considers fundamental problems that distributed synchronous processes have to solve. These fundamental problems concern agreement among processes (if proce one way or another in presence of failures, no non-trivial problem can be solved). They are consensus, interactive consistency, k-set agreement and non-blocking atomic commit. Being able to solve these basic problems efficiently with provable guarantees allows applications designers to give a precise meaning to the words "cooperate" and "agree" despite failures, and write distributed synchronous programs with properties that can be sta the aim of the book is to present a comprehensive view of agreement problems, algorithms that solve them and associated computability bounds in synchronous message-passing distributed systems. Table of Contents: List of Figures / Synchronous Model, Failure Models, and Agreement Problems / Consensus and Interactive Consistency in the Crash Failure Model / Expedite Decision in the Crash Failure Model / Simultaneous Consensu Failures / From Consensus to k-Set Agreement / Non-Blocking Atomic Commit in Presence of Crash Failures / k-Set Agreement Despite Omission Failures / Consensus Despite Byzantine Failures / Byzantine Consensus in Enriched Models

A lucid and up-to-date introduction to the fundamentals of distributed computing systems As distributed systems become increasingly available, the need for a fundamental discussion of the subject has grown. Designed for first-year graduate students and advanced undergraduates as well as practicing computer engineers seeking a solid grounding in the subject, this well-organized text covers the fundamental concepts in distributed compu time, state, simultaneity, order, knowledge, failure, and agreement in distributed systems. Departing from the focus on shared memory and synchronous systems commonly taken by other texts, this is the first useful reference based on an asynchronous model of distributed computing, the most widely used in academia and industry. The emphasis of the book is on developing general mechanisms that can be applied to a variety of problems locks, cameras, sensors, controllers, slicers, and synchronizers-have been carefully chosen so that they are fundamental and yet useful in practical contexts. The text's advantages include: Emphasizes general mechanisms that can be applied to a variety of problems Uses a simple induction-based technique to prove correctness of all algorithms Includes a variety of exercises at the end of each chapter Contains material that has been exte instructor flexibility in choosing appropriate balance between practice and theory of distributed computing

Link Reversal Algorithms

A Few Theoretical Notions

Distributed Algorithms for Message-Passing Systems

Distributed System Design

Fundamentals, Simulations, and Advanced Topics

Understanding distributed computing is not an easy task. This is due to the many facets of uncertainty one has to cope with and master in order to produce correct distributed software. A previous book Communication and Agreement Abstraction for Fault-tolerant Asynchronous Distributed Systems (published by Morgan & Claypool, 2010) was devoted to the problems created by crash failures in asynchronous message-passing systems. The present book focuses on the way to cope with the uncertainty created by process failures (crash, omission failures and Byzantine behavior) in synchronous message-passing systems (i.e., systems whose progress is governed by the passage of time). To that end, the book considers fundamental problems that distributed synchronous processes have to solve. These fundamental problems concern agreement among processes (if processes are unable to agree in one way or another in presence of failures, no non-trivial problem can be solved). They are consensus, interactive consistency, k-set agreement and non-blocking atomic commit. Being able to solve these basic problems efficiently with provable guarantees allows applications designers to give a precise meaning to the words ""cooperate"" and ""agree"" despite failures, and write distributed synchronous programs with properties that can be stated and proved. Hence, the aim of the book is to present a comprehensive view of agreement problems, algorithms that solve them and associated computability bounds in synchronous message-passing distributed systems. Table of Contents: List of Figures / Synchronous Model, Failure Models, and Agreement Problems / Consensus and Interactive Consistency in the Crash Failure Model / Expedite Decision in the Crash Failure Model / Simultaneous Consensus Despite Crash Failures / From Consensus to k-Set Agreement / Non-Blocking Atomic Commit in Presence of Crash Failures / k-Set Agreement Despite Omission Failures / Consensus Despite Byzantine Failures / Byzantine Consensus in Enriched Models

This book is devoted to the most difficult part of concurrent programming, namely synchronization concepts, techniques and principles when the cooperating entities are asynchronous, communicate through a shared memory, and may experience failures. Synchronization is no longer a set of tricks but, due to research results in recent decades, it relies today on sane scientific foundations as explained in this book. In this book the author explains synchronization and the implementation of concurrent objects, presenting in a uniform and comprehensive way the major theoretical and practical results of the past 30 years. Among the key features of the book are a new look at lock-based synchronization (mutual exclusion, semaphores, monitors, path expressions); an introduction to the atomicity consistency criterion and its properties and a specific chapter on transactional memory; an introduction to mutex-freedom and associated progress conditions such as obstruction-freedom and wait-freedom; a presentation of Lamport's hierarchy of safe, regular and atomic registers and associated wait-free constructions; a description of numerous wait-free constructions of concurrent objects (queues, stacks, weak counters, snapshot objects, renaming objects, etc.); a presentation of the computability power of concurrent objects including the notions of universal construction, consensus number and the associated Herlihy's hierarchy; and a survey of failure detector-based constructions of consensus objects. The book is suitable for advanced undergraduate students and graduate students in computer science or computer engineering, graduate students in mathematics interested in the foundations of process synchronization, and practitioners and engineers who need to produce correct concurrent software. The reader should have a basic knowledge of algorithms and operating systems.

Shlomi Dolev presents the fundamentals of self-stabilization and demonstrates theprocess of designing self-stabilizing distributed systems.

Microsystem technology (MST) integrates very small (up to a few nanometers) mechanical, electronic, optical, and other components on a substrate to construct functional devices. These devices are used as intelligent sensors, actuators, and controllers for medical, automotive, household and many other purposes. This book is a basic introduction to MST for students, engineers, and scientists. It is the first of its kind to cover MST in its entirety. It gives a comprehensive treatment of all important parts of MST such as microfabrication technologies, microactuators, microsensors, development and testing of microsystems, and information processing in microsystems. It surveys products built to date and experimental products and gives a comprehensive view of all developments leading to MST devices and robots.

Distributed Computing

Distributed Optimization and Statistical Learning Via the Alternating Direction Method of Multipliers

An Algorithmic Approach

Introduction to Distributed Self-Stabilizing Algorithms

The ARCH Library

Concurrent and Distributed Computing in Java addresses fundamental concepts in concurrent computing with Java examples. The book consists of two parts. The first part deals with techniques for programming in shared-memory based systems. The book covers concepts in Java such as threads, synchronized methods, waits, and notify to expose students to basic concepts for multi-threaded programming. It also includes algorithms for mutual exclusion, consensus, atomic objects, and wait-free data structures. The second part of the book deals with programming in a message-passing system. This part covers resource allocation problems, logical clocks, global property detection, leader election, message ordering, agreement algorithms, checkpointing, and message logging. Primarily a textbook for upper-level undergraduates and graduate students, this thorough treatment will also be of interest to professional programmers.

Introduction : distributed systems - The model - Communication protocols - Routing algorithms - Deadlock-free packet switching - Wave and traversal algorithms - Election algorithms - Termination detection - Anonymous networks - Snapshots - Sense of direction and orientation - Synchrony in networks - Fault tolerance in distributed systems - Fault tolerance in asynchronous systems - Fault tolerance in synchronous systems - Failure detection - Stabilization.

Der Band bietet eine kompakte Einführung in die Nichtsequentielle Programmierung als gemeinsamen Kern von Vorlesungen über Betriebssysteme, Verteilte Systeme, Parallele Algorithmen, Echtzeitprogrammierung und Datenbanktransaktionen. Basiskonzepte zur Synchronisation und Kommunikation nebenläufiger Prozesse werden systematisch dargestellt: Schlösser, Semaphore, Monitore, lokaler und netzweiter Botschaftenaustausch. Die Algorithmen sind in der Programmiersprache Google Go formuliert, mit der viele Synchronisationskonzepte ausgedrückt werden können.

Distributed computing is at the heart of many applications. It arises as soon as one has to solve a problem in terms of entities -- such as processes, peers, processors, nodes, or agents -- that individually have only a partial knowledge of the many input parameters associated with the problem. In particular each entity cooperating towards the common goal cannot have an instantaneous knowledge of the current state of the other entities. Whereas parallel computing is mainly concerned with 'efficiency', and real-time computing is mainly concerned with 'on-time computing', distributed computing is mainly concerned with 'mastering uncertainty' created by issues such as the multiplicity of control flows, asynchronous communication, unstable behaviors, mobility, and dynamicity. While some distributed algorithms consist of a few lines only, their behavior can be difficult to understand and their properties hard to state and prove. The aim of this book is to present in a comprehensive way the basic notions, concepts, and algorithms of distributed computing when the distributed entities cooperate by sending and receiving messages on top of an asynchronous network. The book is composed of seventeen chapters structured into six parts: distributed graph algorithms, in particular what makes them different from sequential or parallel algorithms; logical time and global states, the core of the book; mutual exclusion and resource allocation; high-level communication abstractions; distributed detection of properties; and distributed shared memory. The author establishes clear objectives per chapter and the content is supported throughout with illustrative examples, summaries, exercises, and annotated bibliographies. This book constitutes an introduction to distributed computing and is suitable for advanced undergraduate students or graduate students in computer science and computer engineering, graduate students in mathematics interested in distributed computing, and practitioners and engineers involved in the design and implementation of distributed applications. The reader should have a basic knowledge of algorithms and operating systems.

Euro-Par 2015: Parallel Processing Workshops

An Algorithmic Approach, Second Edition

Introduction to Distributed Algorithms

Multi-Threaded Object-Oriented MPI-Based Message Passing Interface

Distributed Systems

In modern computing a program is usually distributed among several processes. The fundamental challenge when developing reliable and secure distributed programs is to support the cooperation of processes required to execute a common task, even when some of these processes fail. Failures may range from crashes to adversarial attacks by malicious processes. Cachin, Guerraoui, and Rodrigues present an introductory description of fundamental distributed programming abstractions together with algorithms to implement them in distributed systems, where processes are subject to crashes and malicious attacks. The authors follow an incremental approach by first introducing basic abstractions in simple distributed environments, before moving to more sophisticated abstractions and more challenging environments. Each core chapter is devoted to one topic, covering reliable broadcast, shared memory, consensus, and extensions of consensus. For every topic, many exercises and their solutions enhance the understanding This book represents the second edition of "Introduction to Reliable Distributed Programming". Its scope has been extended to include security against malicious actions by non-cooperating processes. This important domain has become widely known under the name "Byzantine fault-tolerance".

Distributed Systems: An Algorithmic Approach, Second Edition provides a balanced and straightforward treatment of the underlying theory and practical applications of distributed computing. As in the previous version, the language is kept as unobscured as possible—clarity is given priority over mathematical formalism. This easily digestible text: Features significant updates that mirror the phenomenal growth of distributed systems Explores new topics related to peer-to-peer and social networks Includes fresh exercises, examples, and case studies Supplying a solid understanding of the key principles of distributed computing and their relationship to real-world applications, Distributed Systems: An Algorithmic Approach, Second Edition makes both an ideal textbook and a handy professional reference.

This book constitutes the refereed proceedings of the 11th International Workshop on Distributed Algorithms, WDAG '97, held in Saarbrücken, Germany, in September 1997. The volume presents 20 revised full papers selected from 59 submissions. Also included are three invited papers by leading researchers. The papers address a variety of current issues in the area of distributed algorithms and, more generally, distributed systems such as various particular algorithms, randomized computing, routing, networking, load balancing, scheduling, message-passing, shared-memory systems, communication, graph algorithms, etc.

Link reversal is a versatile algorithm design technique that has been used in numerous distributed algorithms for a variety of problems. The common thread in these algorithms is that the distributed system is viewed as a graph, with vertices representing the computing nodes and edges representing some other feature of the system (for instance, point-to-point communication channels or a conflict relationship). Each algorithm assigns a virtual direction to the edges of the graph, producing a directed version of the original graph. As the algorithm proceeds, the virtual directions of some of the links in the graph change in order to accomplish some algorithm-specific goal. The criterion for changing link directions is based on information that is local to a node (such as the node having no outgoing links) and thus this approach scales well, a feature that is desirable for distributed algorithms. This monograph presents, in a tutorial way, a representative sampling of the work on link-reversal-based distributed algorithms. The algorithms considered solve routing, leader election, mutual exclusion, distributed queueing, scheduling, and resource allocation. The algorithms can be roughly divided into two types, those that assume a more abstract graph model of the networks, and those that take into account more realistic details of the system. In particular, these more realistic details include the communication between nodes, which may be through asynchronous message passing, and possible changes in the graph, for instance, due to movement of the nodes. We have not attempted to provide a comprehensive survey of all the literature on these topics. Instead, we have focused in depth on a smaller number of fundamental papers, whose common thread is that link reversal provides a way for nodes in the system to observe their local

neighborhoods, take only local actions, and yet cause global problems to be solved. We conjecture that future interesting uses of link reversal are yet to be discovered. Table of Contents: Introduction / Routing in a Graph: Correctness / Routing in a Graph: Complexity / Routing and Leader Election in a Distributed System / Mutual Exclusion in a Distributed System / Distributed Queueing / Scheduling in a Graph / Resource Allocation in a Distributed System / Conclusion

Principles, Algorithms, and Systems

Consistent Distributed Storage

Parallel and Distributed Processing

Theory and Practice

Distributed Network Systems

This book aims at being a comprehensive and pedagogical introduction to the concept of self-stabilization, introduced by Edsger Wybe Dijkstra in 1973. Self-stabilization characterizes the ability of a distributed algorithm to converge within finite time to a configuration from which its behavior is correct (i.e., satisfies a given specification), regardless the arbitrary initial configuration of the system. This arbitrary initial configuration may be the result of the occurrence of a finite number of transient faults. Hence, self-stabilization is actually considered as a versatile non-masking fault tolerance approach, since it recovers from the effect of any finite number of such faults in a unified manner. Another major interest of such an automatic recovery method comes from the difficulty of resetting malfunctioning devices in a large-scale (and so, geographically spread) distributed system (the Internet, Pair-to-Pair networks, and Delay Tolerant Networks are examples of such distributed systems). Furthermore, self-stabilization is usually recognized as a lightweight property to achieve fault tolerance as compared to other classical fault tolerance approaches. Indeed, the overhead, both in terms of time and space, of state-of-the-art self-stabilizing algorithms is commonly small. This makes self-stabilization very attractive for distributed systems equipped of processes with low computational and memory capabilities, such as wireless sensor networks. After more than 40 years of existence, self-stabilization is now sufficiently established as an important field of research in theoretical distributed computing to justify its teaching in advanced research-oriented graduate courses. This book is an initiation course, which consists of the formal definition of self-stabilization and its related concepts, followed by a deep review and study of classical (simple) algorithms, commonly used proof schemes and design patterns, as well as premium results issued from the self-stabilizing community. As often happens in the self-stabilizing area, in this book we focus on the proof of correctness and the analytical complexity of the studied distributed self-stabilizing algorithms. Finally, we underline that most of the algorithms studied in this book are actually dedicated to the high-level atomic-state model, which is the most commonly used computational model in the self-stabilizing area. However, in the last chapter, we present general techniques to achieve self-stabilization in the low-level message passing model, as well as example algorithms.

* Comprehensive introduction to the fundamental results in the mathematical foundations of distributed computing * Accompanied by supporting material, such as lecture notes and solutions for selected exercises * Each chapter ends with bibliographical notes and a set of exercises * Covers the fundamental models, issues and techniques, and features some of the more advanced topics

11th International Workshop, WDAG '97, Saarbrücken, Germany, September 24-26, 1997, Proceedings

Distributed Computing Through Combinatorial Topology

Communication and Agreement Abstractions for Fault-tolerant Asynchronous Distributed Systems