Do-All Computing in Distributed Systems

Cooperation in the Presence of Adversity

Do-All Computing in Distributed Systems

Cooperation in the Presence of Adversity

by

Chryssis Georgiou
University of Cyprus
Cyprus

and

Alexander A. Shvartsman
Massachusetts Institute of Technology (MIT)
USA

Springer

Chryssis Georgiou
University of Cyprus
Dept. Computer Science
P.O.Box 20537
1678 Nicosia, CYPRUS
chryssis@ucy.ac.cy

Alexander A. Shvartsman
University of Connecticut
Computer Science and Engineering
371 Fairfield Way
Storrs, CT 06268, USA
aas@cse.uconn.edu

Library of Congress Control Number: 2007937388

Do-All Computing in Distributed Systems: Cooperation in the Presence of Adversity
by Chryssis Georgiou and Alexander A. Shvartsman

ISBN-13: 978-0-387-30918-7
e-ISBN-13: 978-0-387-69045-2

Printed on acid-free paper.

9 8 7 6 5 4 3 2 1

springer.com

To my wife Agni, and son Yiorgo
CG

To Sana, my wife and best friend
AAS

Contents

List of Figures

List of Symbols

Foreword

Distributed computing was born in the late 1970s when researchers and practitioners started taking into account the intrinsic characteristics of physically distributed systems. The field then emerged as a specialized research area distinct from networking, operating systems, and parallel computing. *Distributed computing* arises when one has to solve a problem in terms of distributed entities, usually called processors, nodes, agents, sensors, peers, actors, processes, etc., such that each entity has only a partial knowledge of the many parameters involved in the problem that has to be solved. While parallel computing and real-time computing can be characterized respectively by the terms *efficiency* and *on time computing*, distributed computing can be characterized by the term *uncertainty*. This uncertainty is created by asynchrony, failures, unstable behaviors, non-monotonicity, system dynamism, mobility, connectivity instability, etc. Mastering one form or another of uncertainty is pervasive in all distributed computing problems.

The unprecedented growth of the Internet as a massive distributed network in the last decade created a platform for new distributed applications that in turn poses new challenges for distributed computing research. One such class of distributed applications is comprised of computing-intensive problems that in the past were relegated to the realm of massively parallel systems. The Internet, with its millions of interconnected computers, presents itself as a natural platform where the availability of massive distributed computing resources is seen as a compelling alternative to expensive specialized parallel supercomputers. Large networks, used as distributed supercomputers, scale much better than tightly-coupled parallel machines while providing much higher potential for parallel processing. However, harnessing the computing power contained within large networks is challenging because, unlike the applications developed for the controlled computing environments of purposefully-designed parallel systems, applications destined for distributed systems must exist in the environment fraught with uncertainty and adversity.

The field of distributed computing research, as many other areas of informatics, has traditionally encompassed both *science* and *engineering* dimensions. Roughly speaking, these can be seen as complementary facets: science is to understand and engineering is to build. With respect to distributed computing, we are often concerned with a *science of abstraction*, namely, creating the right model for a problem and devising the appropriate mechanizable techniques to solve it. This is particularly true in fault-tolerant, dynamic, large-scale distributed computing where finding models that are realistic while remaining abstract enough to be tractable, was, is, and still remains a real challenge.

The monograph by Chryssis Georgiou and Alex Shvartsman presents a very comprehensive study of massive cooperative computing in distributed settings in the presence of adversity. They focus on a problem that meaningfully abstracts a network supercomputing paradigm, specifically where distributed computing agents cooperate on performing a large number of independent tasks. Such a computation paradigm forms a cornerstone for solutions to several computation-intensive problems ranging from distributed search to distributed simulation and multi-agent collaboration. For the purposes of this study, the authors define *Do-All* as the problem of multiple processors in a network cooperatively performing a collection of independent tasks in the presence of adversity, such as processor failures, asynchrony, and breakdowns in communication. Achieving efficiency in such cooperation is difficult due to the dynamic characteristics of the distributed environments in which computing agents operate, including network failures, and processor failures that can range from the benign crash failures to the failures where faulty components may behave arbitrarily and even maliciously. The *Do-All* problem and its iterative version is used to identify the trade-offs between efficiency and fault-tolerance in distributed cooperative computing, and as a target for algorithm development. The ultimate goal is to develop algorithms that combine efficiency with fault-tolerance to the maximum extent possible, and that can serve as building blocks for network supercomputing applications and, more generally, for applications requiring distributed cooperation in the face of adversity.

During the last two decades, significant research was dedicated to studying the *Do-All* problem in various models of computation, including message-passing, partitionable networks, and shared-memory models under specific assumptions about synchrony/asynchrony and failures. This monograph presents in a coherent and rigorous manner the lower bound results and the most significant algorithmic solutions developed for *Do-All* in the message-passing model, including partitionable networks. The topics chosen for presentation include several relevant models of adversity commonly encountered in distributed computing and a variety of algorithmics illustrating important and effective techniques for solving the problem of distributed cooperation. The monograph also includes detailed complexity analysis of algorithms, assessing their efficiency in terms of work, communication, and time.

As the aim of a theory is to codify knowledge in order for it to be transmitted (to researchers, students, engineers, practitioners, etc), the research results presented in this monograph are among the fundamental bases in distributed computing theory. When effective distributed cooperation is possible, we learn why and how it works, and where there exist inherent limitations in distributed cooperation, we learn what they are and why they exist.

Rennes, France *Michel Raynal*
September 2007

Authors' Preface

With the advent of ubiquitous high bandwidth Internet connections, network supercomputing is increasingly becoming a popular means for harnessing the computing power of an enormous number of processes around the world. Internet supercomputing comes at a cost substantially lower than acquiring a supercomputer or building a cluster of powerful machines. Several Internet supercomputers are in existence today, for instance, Internet PrimeNet Server, a project comprised of about 30,000 servers, PCs, and laptop computers, supported by Entropia.com, Inc., is a distributed, massively parallel mathematics research Internet supercomputer. PrimeNet Server has sustained throughput of over 1 teraflop. Another popular Internet supercomputer, the SETI@home project, also reported its speed to be in teraflops.

In such distributed supercomputing settings it is often the case that a very large number of independent tasks must be performed by an equally large number of computers. Given the massive numbers of participating computers, it is invariably the case that non-trivial subsets of these machines may be faulty, disconnected, experiencing delays, or simply off-line at any given point in time. At such scales of distributed computing, failures are no longer an exception, but the norm. For example, a visitor to the network control center at Akamai Technologies, a global Internet content and application delivery company, will immediately notice that the floor-to-ceiling monitor-paneled walls of the main control room display a surprisingly large number of server icons in red, indicating server failures. Yet the services delivered by the company's 25,000 servers worldwide continue unaffected, and there is little alarm among the engineers monitoring the displays. Dealing with failures is routine business, provided the massively distributed system has built-in redundancy and is able to combine efficiency with fault-tolerance.

In another example, Internet supercomputing, such as SETI@home, involves large sets of independent tasks performed by distributed worker computers. One of the major concerns involved in such computing environments is the reliability of the results returned by the workers. While most participating computers may be reliable, a large number of the workers have been known to return incorrect results for various reasons. Workers may return incorrect results due to unintended failures caused, for example, by over-clocked processors, or they may claim to have performed assigned work so as to obtain incentives, such as getting higher rank on the SETI@home list of contributed units of work. This problem already exists in the setting where the task allocation is centralized, and assumed to be reliable. The problem becomes substantially more difficult when the task allocation also has to be implemented in a highly-distributed fashion to provide the much needed parallelism for computation speed-up and redundancy for fault tolerance. In such settings it is extremely important to develop distributed algorithms that can be used to ensure dependable and efficient execution of the very large numbers of tasks.

In this monograph we abstract the problem of distributed cooperation in terms of the *Do-All* problem, defined as the problem of p processors in the network, cooperatively performing n independent tasks, in the presence of adversity. In solving this problem, we pursue the goal of combining the reliability potential that comes with replicated processors in distributed computation, with the speed-up potential of performing the large number of tasks in parallel. The difficulty associated with combining fault-tolerance with efficiency is that the two have conflicting means: fault-tolerance is achieved by *introducing redundancy*, while efficiency is achieved by *removing redundancy*. We present several significant advances in algorithms designed to solve the *Do-All* problem in distributed message-passing settings under various models of adversity, such as processor crashes, asynchrony, message delays, network partitions, and malicious processor behaviors. The efficiency of algorithms for *Do-All* is most commonly assessed in terms of *work* and *communication* complexity, depending on the specific model of computation. Work is defined either as the total number of computational steps taken by all available processors during the computation or as the total number of *task-oriented* computational steps taken by the processors. A computational step taken by a processor is said to be task-oriented, if during that step the processor performs a *Do-All* task. We refer to the first variation of work as *total-work* and the second variation of work as *task-oriented work*. We develop corresponding complexity analyses that show to what extent efficiency can be combined with fault-tolerance. We also present lower bounds that capture theoretical limitations on the possibility of combining fault-tolerance and efficiency. In this work we ultimately aim to provide robust, i.e., efficient and fault-tolerant, algorithms that will help bridge the gap between abstract models of dependable network computing and realistic distributed systems.

Monograph Roadmap

In Chapter 1 we provide motivation, introduce the distributed cooperation problem *Do-All* and discuss several variants of the problem in different models of computation.

In Chapter 2 we formal the basic message-passing model of computation used in this monograph, and present several models of adversarial settings studied in subsequent chapters. We define the nature of the tasks – the input to the distributed cooperation problem. We define the *Do-All* problem, and its counterpart for partitionable networks, the *Omni-Do* problem. We conclude the chapter with the definitions of main complexity measures used in the sequel: total-work, task-oriented work, and message complexity.

In Chapter 3 we study the *Do-All* problem for distributed settings with processor crashes. We provide upper and lower bounds on work for solving *Do-All* under the assumption of perfect knowledge, e.g., when an algorithm is aided by an omniscient oracle. We put these result to use by developing an efficient and fault-tolerant algorithm for *Do-All* where processors communicate by means of reliable broadcasts.

In Chapter 4 we develop a solution for the *Do-All* problem for the setting with processor crashes, where processors communicate using point-to-point messaging. This algorithm uses a gossip algorithm as a building block, also presented in the chapter.

In Chapter 5 we give lower bounds on work for *Do-All* in the model where processors are subject to crashes and restarts, and we develop and analyze an algorithm for this model of adversity.

In Chapter 6 we study the complexity of *Do-All* in the adversarial model where processors are subject to Byzantine failures, that is, where faulty processors may behave arbitrarily and even maliciously. We provide several algorithms and lower bound results under this model of adversity.

In Chapter 7 we study the upper and lower bounds of solving *Do-All* in the setting where an adversary introduces processor asynchrony and message delays. We present several algorithm for this model and provide their delay-sensitive analysis.

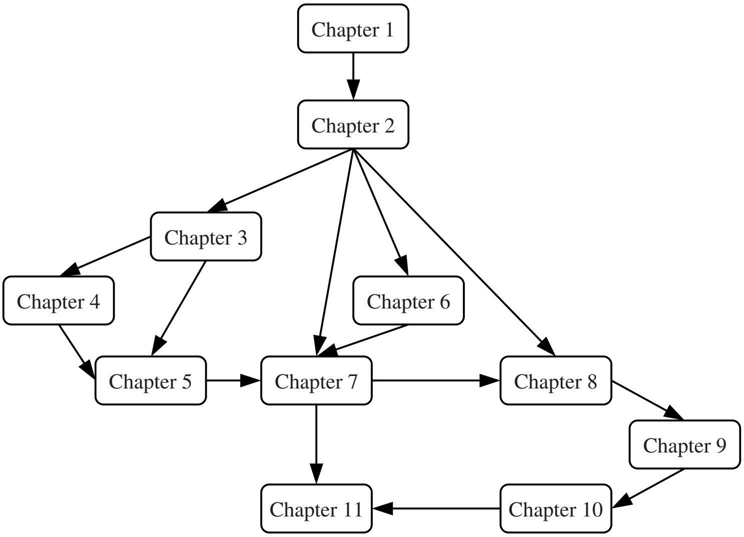
In Chapter 8 we switch our attention to partitionable networks and the *Omni-Do* problem. We give an efficient algorithm that solves *Omni-Do* in the presence of network fragmentation and merges.

In Chapter 9 we study the *Omni-Do* problem in the model where the network can undergo arbitrary reconfigurations. We assess upper and lower bounds for the problem using competitive analysis.

In Chapter 10 we study *Do-All* in the setting where the adversary initially starts processors in isolated singleton groups, and then allows the processor to rendezvous. We analyze redundant work performed by the isolated processors prior to rendezvous, and we present several scheduling strategies designed to minimize redundant task executions.

Finally, in Chapter 11 we survey related problems and models, including the problem of distributed cooperation in shared-memory models, algorithms for the model where processors communicate through broadcast channels, and we show a connection between *Do-All* and the distributed consensus problem.

The chapters of this monograph can of course be read in the sequential order from Chapter 1 to Chapter 11. In the diagram that follows we show alternative suggested paths through the monograph. Chapters 1 and 2 should be read in sequence before other chapters. It is also recommended that Chapters 8, 9, and 10 are read in sequence. The only remaining dependency is that Chapter 3 is read before Chapter 5.



In presenting our message-passing algorithms, we aim to illustrate the most interesting algorithmic techniques and paradigms, using a clear high-level level pseudocode that is best suited to represent the nature of each algorithm.

Each chapter concludes with an overview of open problems relevant to the topics presented in the chapter, and a section containing chapter notes, including detailed bibliographic notes, and selected comparisons with and overviews of related work.

Bibliographic Notes

At the end of each chapter we provide Chapter Notes that contain bibliographic notes and overview related topics and results. The complete bibliography follows the last chapter. Here we give additional pointers to conference proceedings, archival journals, and books covering the various areas related to distributed computing and fault-tolerant algorithms. Most results in this monograph appeared as articles in journals or conference proceedings (see bibliography), additionally the main results in Chapters 3, 4, 6, 8, and 9 appear in the PhD dissertation of the first author [43].

Work on fault-tolerant distributed computation related to the content of this monograph appear in the proceedings of conferences, in journals, and in books. A reader interested in learning more about this ongoing research as well as research beyond the scope of this volume will be well served by consulting recent publication on such topics from the venues we list below.

The following conferences are examples of the most relevant fora for results related to topics in this monograph: ACM symposium on Principles of Distributed Computing (PODC), ACM symposium on Parallel Algorithms and Architectures (SPAA), ACM symposium on Theory of Computing (STOC), ACM-SIAM symposium on Discrete Algorithms (SODA), IEEE symposium on Foundations of Computer Science (FOCS), IEEE sponsored conference on Distributed Computing Systems (ICDCS), EATCS sponsored symposium on Distributed Computing (DISC), the conference on the Principles on Distributed Systems (OPODIS) and the colloquium on Structural Information and Communication Complexity (SIROCCO). The most relevant journals include: Springer Distributed Computing, SIAM Journal on Computing, Theoretical Computer Science, Information and Computation, Information Processing Letters, Parallel Processing Letters, Journal of the ACM, Journal of Algorithms, Journal of Discrete Algorithms, and Journal of Parallel and Distributed Computing.

The 1997 book by Kanellakis and Shvartsman [67] presents research results for fault-tolerant cooperative computing in the parallel model of computation. In particular, it studies the *Do-All* problem in the shared-memory model, where it is referred to as the *Write-All* problem. The current monograph deals with the message-passing models of computation and considers broader adversarial settings inherent to these distributed models. The two monographs follow similar presentation philosophies and it is reasonable to consider them as complementary volumes. The current volume includes in Chapter 11 several recent results on the *Write-All* problem that appeared since the publication of the first monograph [67].

The book by Lynch [79] provides a wealth of information on distributed computing issues, such as computational models, algorithms, fault-tolerance, lower bounds and impossibility results. This include the consensus problem, which is related to *Do-All*, and we discuss this relation in Chapter 11. Additionally, information on the Input/Output Automata used in our Chapter 8 can be found there. The book by Attiya and Welch [6] is another excellent source of information on distributed computing issues, including cooperation. The book of Guerraoui and Rodrigues [52] presents numerous important abstractions for reliable distributed computing and includes detailed examples of how these abstractions can be implemented and used in practice.

Acknowledgements

Our research on robust distributed cooperation continues to be inspired by the earlier work on fault-tolerant parallel computing of the late Paris Christos Kanellakis (1953-1995). Paris is survived by his parents, General Eleftherios and Roula Kanellakis, who have enthusiastically encouraged us to continue his work through the long years following the tragic death of Paris, his wife Maria-Teresa, and their children Alexandra and Stephanos. We warmly thank General and Mrs. Kanellakis for their inspiration and support.

The material presented in this monograph includes results obtained by the authors in collaboration with Bogdan Chlebus, Roberto De Prisco, Antonio Fernandez, Dariusz Kowalski, Greg Malewicz and Alexander Russell. We thank them for the wonderful and fruitful collaborations—without their contributions this monograph would not exist.

Our work on robust distributed cooperation also benefited from prior collaboration with several colleagues, and we gratefully acknowledge the contributions of Jonathan Buss, Shlomi Dolev, Leszek Gasieniec, Dimitrios Michailidis, Prabhakar Ragde, and Roberto Segala.

Special thanks are due to Nancy Lynch for reviewing an earlier version of this work. Her insight and valuable feedback are greatly appreciated.

In undertaking the research that ultimately resulted in this monograph, we were motivated by the work of other researchers who have also contributed to the field of fault-tolerant cooperative computing. We will be remiss without mentioning the names of Richard Anderson, Richard Cole, Cynthia Dwork, Zvi Galil, Phillip Gibbons, Joe Halpern, Maurice Herlihy, Zvi Kedem, Andrzej Lingas, Chip Martel, Keith Marzullo, Alan Mayer, Naomi Nishimura, Krishna Palem, Arvin Park, Michael Rabin, Arvind Raghunathan, Nir Shavit, Paul Spirakis, Ramesh Subramonian, Orli Waarts, Heather Woll, Moti Yung, Ofer Zajicek, and Asaph Zemach.

This work was in part supported by the National Science Foundation (NSF) Grants 9984778, 9988304, 0121277, 0311368, and by the NSF-NATO Award 0209588. The work of the first author has also been partially supported by research funds from the University of Cyprus.

We thank our Springer editor, Susan Lagerstrom-Fife, for her encouragement and support, and Sharon Palleschi, editorial assistant at Springer, for her valuable assistance during the preparation of this monograph.

Our warmest thanks go to Michel Raynal for writing the foreword of this monograph; thank you for this honor, Michel.

Finally, we would like to thank our families.

CG: I thank my wife, Agni, for the emotional support she has given me and the patience she has shown during the endless nights I spent working on this monograph, while she took care of our angel, 21 months old son Yiorgo. You both bring joy and meaning to my life and you are my source of strength and inspiration. Agni, you are the *love* of my life. Yiorgo, you *are* my life.

AAS: My wife Sana gave me more affection, care, and happiness, than I could have dreamed of. She spent many lonely nights being an epitome of patience, while sustaining herself only by my assurances that I'll belong to her yet again. Thank you, my love. I thank my children, Ginger and Ted, for being there for me when I needed you most. This time around you are grown-ups and I am proud of you. I thank my step-son Arnold for his encouragement and interest. I never thought that daily questions from a freshman "Are you done with the book yet?" would do so much to energize the work of this professor. I am glad to report: "We are done."

Nicosia, Cyprus and Storrs, CT, USA
September 2007

Chryssis Georgiou
Alexander A. Shvartsman