TOOLS AND ENVIRONMENTS FOR PARALLEL AND DISTRIBUTED COMPUTING

Edited by

Salim Hariri
Manish Parashar

A JOHN WILEY & SONS, INC., PUBLICATION

TOOLS AND ENVIRONMENTS FOR PARALLEL AND DISTRIBUTED COMPUTING

WILEY SERIES ON PARALLEL AND DISTRIBUTED COMPUTING

Series Editor: Albert Y. Zomaya

Parallel and Distributed Simulation Systems / Richard Fujimoto

Surviving the Design of Microprocessor and Multimicroprocessor Systems: Lessons Learned / Veljko Milutinović

Mobile Processing in Distributed and Open Environments / Peter Sapaty

Introduction to Parallel Algorithms / C. Xavier and S. S. Iyengar

Solutions to Parallel and Distributed Computing Problems: Lessons from Biological Sciences / Albert Y. Zomaya, Fikret Ercal, and Stephan Olariu (*Editors)*

New Parallel Algorithms for Direct Solution of Linear Equations / C. Siva Ram Murthy, K. N. Balasubramanya Murthy, and Srinivas Aluru

Practical PRAM Programming / Joerg Keller, Christoph Kessler, and Jesper Larsson Traeff

Computational Collective Intelligence / Tadeusz M. Szuba

Parallel and Distributed Computing: A Survey of Models, Paradigms, and Approaches / Claudia Leopold

Fundamentals of Distributed Object Systems: A CORBA Perspective / Zahir Tari and Omran Bukhres

Pipelined Processor Farms: Structured Design for Embedded Parallel Systems / Martin Fleury and Andrew Downton

Handbook of Wireless Networks and Mobile Computing / Ivan Stojmenović (*Editor*)

Internet-Based Workflow Management: Toward a Semantic Web / Dan C. Marinescu

Parallel Computing on Heterogeneous Networks / Alexey L. Lastovetsky

Tools and Environments for Parallel and Distributed Computing / Salim Hariri and Manish Parashar (*Editors*)

TOOLS AND ENVIRONMENTS FOR PARALLEL AND DISTRIBUTED COMPUTING

Edited by

Salim Hariri
Manish Parashar

A JOHN WILEY & SONS, INC., PUBLICATION

Published by John Wiley & Sons, Inc., Hoboken, New Jersey.
Published simultaneously in Canada.

Library of Congress Cataloging-in-Publication Data:

Hariri, Salim.
Tools and environments for parallel and distributed computing / Salim Hariri & Manish Parashar.
p. cm.
ISBN 0-471-33288-7 (Cloth)
1. Parallel processing (Electronic computers) 2. Electronic data processing—Distributed processing. I. Parashar, Manish, 1967– II. Title.
QA76.58.H37 2004
004′.35—dc21 2003014209

Printed in the United States of America.

10 9 8 7 6 5 4 3 2 1

CONTENTS

CONTRIBUTORS

Sumir Chandra, Department of Electrical and Computer Engineering, Rutgers University, Piscataway, NJ 08854 (*E-mail*: sumir@caip.rutgers.edu)

Salim Hariri, Department of Electrical and Computer Engineering, University of Arizona, Tucson, AZ 85721 (*E-mail*: hariri@ece.arizona.edu)

A. Kalyanaraman, Department of Computer and Information Science, Indiana University Purdue University, Indianapolis, IN (*E-mail*: akalyana@cs.iupui.edu)

N. Nayani, Department of Computer and Information Science, Indiana University Purdue University, Indianapolis, IN (*E-mail*: nnayani@cs.iupui.edu)

Manish Parashar, Department of Electrical and Computer Engineering, Rutgers University, Piscataway, NJ 08854 (*E-mail*: parashar@caip.rutgers.edu)

Ilkyeun Ra, Department of Computer Science and Engineering, University of Colorado at Denver, Denver, CO 80217 (*E-mail*: ikra@carbon.cudenver.edu)

Rajeev Raje, Department of Computer and Information Science, Indiana University Purdue University, Indianapolis, IN (*E-mail*: rraje@cs.iupui.edu)

G. von Laszewski, Argonne National Laboratory, 9700 South Cass Avenue, Argonne, IL 60439 (*E-mail*: gregor@mcs.anl.gov)

P. Wagstrom, Department of Engineering and Public Policy, Carnegie Mellon University, Pittsburgh, PA 15213 (*E-mail*: pwagstro@andrew.cmu.edu)

PREFACE

The primary focus of this book is the rapidly evolving software technology for supporting the development, execution, management, and experimentation with parallel and distributed computing environments. The design, development, and utilization of parallel and distributed computing environments that can efficiently support a wide range of scientific and engineering applications remains a challenging research problem due to the complexity and varying requirements of the applications, heterogeneity of the computing systems and their networks, asynchronous complex interactions among the system and application components, and the heterogeneity of the software tools and environments. However, recent advances in processing and network technology and software tools have addressed successfully many of the obstacles hindering the wide deployment of parallel and distributed computing environments.

Active research in parallel processing has resulted in advances in all aspects of the computing technologies, including processing technology, computer networking technology, and software technology. Advances in processing technology have resulted in faster, more powerful processors with increased functionality. Advances in computer networking technology have introduced reliable high-speed networks capable of providing high transfer rates. Advances in software technology have provided easy-to use tools and environments for the development of parallel applications. These advances have resulted in the proliferation of a large number of different architectural classes, such as SIMD computers, MIMD computers, vector computers, and data flow computers, where each class represents a set of different trade-offs in design decisions such as coarse-grained (MIMD) parallelism versus fine-grained (SIMD) parallelism, shared memory MIMD versus distributed memory MIMD, hypercube topology versus mesh topology, and circuit-switched versus packet-switched communication. Each architectural class is tuned to deliver maximum performance to a specific set of applications. However, it remains a fact that none of the existing computing systems are general enough to address all classes of applications and provide the desired performance levels. In addition, these architectures are not scalable and their relatively narrow applicability has prevented them from being cost-effective.

Furthermore, the development of efficient application software capable of exploiting the available computing potential is nontrivial and requires a thor-

ough understanding not only of the application, but also of the target computing environment. Given the diversity of current computing systems and their architectural complexity, this is not a reasonable proposition, especially since application developers are not, in general, computer engineers. Even porting existing applications to high-performance systems is nontrivial and usually involves extensive redevelopment. As a result, the future of parallel and distributed computing will be governed to a large extent by the availability of sufficiently high-level languages, tools, and development environments that can support application developers.

A key factor contributing to the complexity of parallel software development is the increased degrees of freedom that have to be resolved and tuned in such an environment. Typically, during the course of parallel software development, the developer is required to select between available algorithms for the particular application, between possible hardware configurations and among possible decompositions of the problem onto the hardware configuration selected, and between different communication and synchronization strategies. The set of reasonable alternatives that have to be evaluated is very large, and selecting the best alternative among these is a formidable task.

The current user has to spend considerable time in understanding the details of the overall system as well as specific system aspects such as data distribution, problem partitioning and scheduling, routing, load balancing, efficient vectorization, efficient utilization of the memory hierarchy, and synchronization, in order to achieve even a fraction of the theoretical peak performance offered by the system. Consequently, there exists a significant disproportion between the effort involved in developing an application algorithm and in its efficient realization on any high-performance system.

It is this realization that has motivated the writing of this book. The goal of the book is to serve as a reference for current software tools and technologies that can be used to develop, implement, and support high-performance parallel and distributed computing environments and applications. In this book we review promising software tools and environments that play an important role in the development of high-performance parallel/distributed systems and applications, and highlight the salient features and limitations of these tools and environments. Consequently, this book can serve as a useful reference for researchers, educators, and practitioners in the field of parallel and distributed computing, supercomputing, and networking.

The book is organized into six chapters; a brief summary is as follows.

Chapter 1: Parallel and Distributed Computing

This chapter provides an introduction to parallel and distributed systems and their benefits in performance, resource sharing, extendibility, reliability, and cost-effectiveness. It outlines parallel and distributed computing approaches and paradigms and the opportunities and challenges of parallel and

distributed computing. Finally, it presents a three-tiered distributed system design framework to highlight architectural issues, services, and candidate technologies for implementing parallel/distributed computing systems and applications.

Chapter 2: Message-Passing Tools

This chapter briefly reviews message-passing models for network-centric applications. It presents the advantages of message-passing tools and their classification with respect to the application domain, programming model supported, communication model, portability, and adaptability. The chapter describes hardware- and software-based approaches to improve the performance of message-passing tools. This is followed by an overview of existing message-passing tools such as socket-based message passing, p4, Parallel Virtual Machine (PVM), Message-Passing Interface (MPI), Nexus, Madeleine, and Active Messages. The chapter then describes the design of ACS (Adaptive Communication System), a multithreaded message-passing tool, and presents an experimental evaluation of ACS and three different message-passing tools (p4, PVM, and MPI) with respect to primitives and application performance.

Chapter 3: Distributed Shared Memory Tools

This chapter presents tools and environments for distributed shared memory (DSM), a software abstraction of shared memory on a distributed memory multiprocessor or cluster of workstations. It outlines the properties and features of DSM systems and classifies them based on their architectures. The chapter then describes cache coherence protocols for hardware-based DSM systems, including CC-NUMA, COMA, and S-COMA; hybrid schemes, including R-NUMA and AS-COMA; and hardware-based environments, such as the MIT Alewife Machine and the Stanford FLASH multiprocessor. Finally, existing DSM systems such as TreadMarks, Brazos, Mirage+, Orca, SAM, Midway, CRL, and fine-grained Shasta DSM are described.

Chapter 4: Distributed-Object Computing Tools

This chapter provides an overview of popular distributed-object approaches such as Java RMI, CORBA, and DCOM and presents the basic model underlying each approach followed by example applications with code segments from different domains. An experimental evaluation of these approaches is presented followed by a comparison of the approaches with respect to language and platform dependency, implementation, architecture, additional feature support, and performance. The proposed Unified Metaobject Model is discussed.

Chapter 5: Gestalt of the Grid

The gestalt of the Grid presented in this chapter provides an overview of important influences, developments, and technologies that are shaping state-of-the-art Grid computing. The motivation and enabling factors for the development of the Grid are described, followed by various Grid definitions. Common architectural views such as *N*-tiered, role-based, and service-based Grid architectures are presented, followed by Grid management aspects that include managing Grid security, Grid information, Grid data, Grid execution and resources, and Grid software and hardware. The Grid activities presented in this chapter are classified into community activities such as Global Grid Forum and production Grids; development toolkits and middleware, such as the Globus project, OGSA, Legion, Storage Resource Broker, Akenti, and NWS; high-throughput computing, such as Condor, NetSolve, Ninf, SETI@Home, and Nimrod-G; and applications such as ASC, PPDG, and NEESgrid. Popular Grid portals and their toolkits, such as HotPage, Webflow and Gateway, XCAT, UNICORE, JiPANG, PUNCH, and Access Grid, are presented.

Chapter 6: Software Development for Parallel and Distributed Computing

This chapter presents a study of the software development process in high-performance parallel and distributed computing (HPC) environments and investigates the nature of support required at each stage of development. The objective is to illustrate the significance of tools and environments discussed in this book during software development. The chapter first highlights some of the issues that must be addressed during HPC software development. The HPC software development process is then described. A parallel stock option pricing model is used as a running example in this discussion. Finally, some existing tools applicable at each stage of the development process are identified.

Acknowledgments

This book has been made possible due to the efforts and contributions of many people. First and foremost, we would like to acknowledge all the contributors for their tremendous effort in putting together excellent chapters that are comprehensive and very informative. We would like to thank the reviewers for their excellent comments and suggestions. We would also like to thank Val Moliere, Kirsten Rohstedt, and the team at John Wiley & Sons, Inc. for getting this book together. Finally, we would like to dedicate this book to our families.

Salim Hariri
Manish Parashar

CHAPTER 1

Parallel and Distributed Computing

S. HARIRI

Department of Electrical and Computer Engineering, University of Arizona, Tucson, AZ

M. PARASHAR

Department of Electrical and Computer Engineering, Rutgers University, Piscataway, NJ

1.1 INTRODUCTION: BASIC CONCEPTS

The last two decades spawned a revolution in the world of computing; a move away from central mainframe-based computing to network-based computing. Today, servers are fast achieving the levels of CPU performance, memory capacity, and I/O bandwidth once available only in mainframes, at a cost orders of magnitude below that of a mainframe. Servers are being used to solve computationally intensive problems in science and engineering that once belonged exclusively to the domain of supercomputers. A distributed computing system is the system architecture that makes a collection of heterogeneous computers, workstations, or servers act and behave as a single computing system. In such a computing environment, users can uniformly access and name local or remote resources, and run processes from anywhere in the system, without being aware of which computers their processes are running on. Distributed computing systems have been studied extensively by researchers, and a great many claims and benefits have been made for using such systems. In fact, it is hard to rule out any desirable feature of a computing system that has not been claimed to be offered by a distributed system [24]. However, the current advances in processing and networking technology and software tools make it feasible to achieve the following advantages:

- *Increased performance*. The existence of multiple computers in a distributed system allows applications to be processed in parallel and thus

Tools and Environments for Parallel and Distributed Computing, Edited by Salim Hariri and Manish Parashar
ISBN 0-471-33288-7

improves application and system performance. For example, the performance of a file system can be improved by replicating its functions over several computers; the file replication allows several applications to access that file system in parallel. Furthermore, file replication distributes network traffic associated with file access across the various sites and thus reduces network contention and queuing delays.

- *Sharing of resources*. Distributed systems are cost-effective and enable efficient access to all system resources. Users can share special purpose and sometimes expensive hardware and software resources such as database servers, compute servers, virtual reality servers, multimedia information servers, and printer servers, to name just a few.
- *Increased extendibility*. Distributed systems can be designed to be modular and adaptive so that for certain computations, the system will configure itself to include a large number of computers and resources, while in other instances, it will just consist of a few resources. Furthermore, limitations in file system capacity and computing power can be overcome by adding more computers and file servers to the system incrementally.
- *Increased reliability, availability, and fault tolerance*. The existence of multiple computing and storage resources in a system makes it attractive and cost-effective to introduce fault tolerance to distributed systems. The system can tolerate the failure in one computer by allocating its tasks to another available computer. Furthermore, by replicating system functions and/or resources, the system can tolerate one or more component failures.
- *Cost-effectiveness*. The performance of computers has been approximately doubling every two years, while their cost has decreased by half every year during the last decade [3]. Furthermore, the emerging high-speed network technology [e.g., wave-division multiplexing, asynchronous transfer mode (ATM)] will make the development of distributed systems attractive in terms of the price/performance ratio compared to that of parallel computers.

These advantages cannot be achieved easily because designing a general purpose distributed computing system is several orders of magnitude more difficult than designing centralized computing systems—designing a reliable general-purpose distributed system involves a large number of options and decisions, such as the physical system configuration, communication network and computing platform characteristics, task scheduling and resource allocation policies and mechanisms, consistency control, concurrency control, and security, to name just a few. The difficulties can be attributed to many factors related to the lack of maturity in the distributed computing field, the asynchronous and independent behavior of the systems, and the geographic dispersion of the system resources. These are summarized in the following points:

- There is a lack of a proper understanding of distributed computing theory—the field is relatively new and we need to design and experiment with a large number of general-purpose reliable distributed systems with different architectures before we can master the theory of designing such computing systems. One interesting explanation for the lack of understanding of the design process of distributed systems was given by Mullender [2]. Mullender compared the design of a distributed system to the design of a reliable national railway system that took a century and half to be fully understood and mature. Similarly, distributed systems (which have been around for approximately two decades) need to evolve into several generations of different design architectures before their designs, structures, and programming techniques can be fully understood and mature.
- The asynchronous and independent behavior of the system resources and/or (hardware and software) components complicate the control software that aims at making them operate as one centralized computing system. If the computers are structured in a master–slave relationship, the control software is easier to develop and system behavior is more predictable. However, this structure is in conflict with the distributed system property that requires computers to operate independently and asynchronously.
- The use of a communication network to interconnect the computers introduces another level of complexity. Distributed system designers not only have to master the design of the computing systems and system software and services, but also have to master the design of reliable communication networks, how to achieve synchronization and consistency, and how to handle faults in a system composed of geographically dispersed heterogeneous computers. The number of resources involved in a system can vary from a few to hundreds, thousands, or even hundreds of thousands of computing and storage resources.

Despite these difficulties, there has been limited success in designing special-purpose distributed systems such as banking systems, online transaction systems, and point-of-sale systems. However, the design of a general-purpose reliable distributed system that has the advantages of both centralized systems (accessibility, management, and coherence) and networked systems (sharing, growth, cost, and autonomy) is still a challenging task [27]. Kleinrock [7] makes an interesting analogy between the human-made computing systems and the brain. He points out that the brain is organized and structured very differently from our present computing machines. Nature has been extremely successful in implementing distributed systems that are far more intelligent and impressive than any computing machines humans have yet devised. We have succeeded in manufacturing highly complex devices capable of high-speed computation and massive accurate memory, but we have not gained sufficient understanding of distributed systems; our systems are still highly

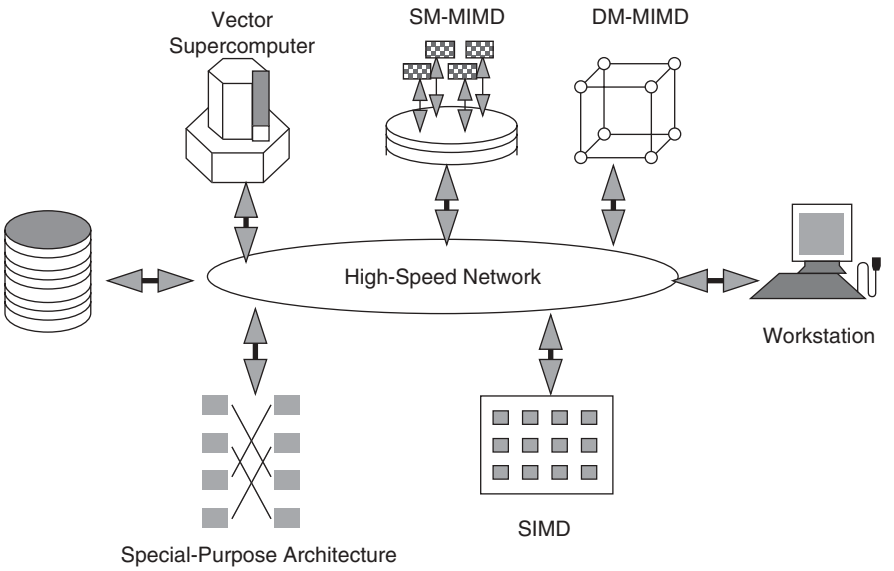

Fig. 1.1 High-performance distributed system.

constrained and rigid in their construction and behavior. The gap between natural and man-made systems is huge, and more research is required to bridge this gap and to design better distributed systems.

In the next section we present a design framework to better understand the architectural design issues involved in developing and implementing high-performance distributed computing systems. A *high-performance distributed system* (HPDS) (Figure 1.1) includes a wide range of computing resources, such as workstations, PCs, minicomputers, mainframes, supercomputers, and other special-purpose hardware units. The underlying network interconnecting the system resources can span LANs, MANs, and even WANs, can have different topologies (e.g., bus, ring, full connectivity, random interconnect), and can support a wide range of communication protocols.

1.2 PROMISES AND CHALLENGES OF PARALLEL AND DISTRIBUTED SYSTEMS

The proliferation of high-performance systems and the emergence of high-speed networks (terabit networks) have attracted a lot of interest in parallel and distributed computing. The driving forces toward this end will be (1) the advances in processing technology, (2) the availability of high-speed network, and (3) the increasing research efforts directed toward the development of software support and programming environments for distributed computing.

Further, with the increasing requirements for computing power and the diversity in the computing requirements, it is apparent that no single computing platform will meet all these requirements. Consequently, future computing environments need to capitalize on and effectively utilize the existing heterogeneous computing resources. Only parallel and distributed systems provide the potential of achieving such an integration of resources and technologies in a feasible manner while retaining desired usability and flexibility. Realization of this potential, however, requires advances on a number of fronts: processing technology, network technology, and software tools and environments.

1.2.1 Processing Technology

Distributed computing relies to a large extent on the processing power of the individual nodes of the network. Microprocessor performance has been growing at a rate of 35 to 70 percent during the last decade, and this trend shows no indication of slowing down in the current decade. The enormous power of the future generations of microprocessors, however, cannot be utilized without corresponding improvements in memory and I/O systems. Research in main-memory technologies, high-performance disk arrays, and high-speed I/O channels are, therefore, critical to utilize efficiently the advances in processing technology and the development of cost-effective high-performance distributed computing.

1.2.2 Networking Technology

The performance of distributed algorithms depends to a large extent on the bandwidth and latency of communication among the network nodes. Achieving high bandwidth and low latency involves not only fast hardware, but also efficient communication protocols that minimize the software overhead. Developments in high-speed networks provide gigabit bandwidths over local area networks as well as wide area networks at moderate cost, thus increasing the geographical scope of high-performance distributed systems.

The problem of providing the required communication bandwidth for distributed computational algorithms is now relatively easy to solve given the mature state of fiber-optic and optoelectronic device technologies. Achieving the low latencies necessary, however, remains a challenge. Reducing latency requires progress on a number of fronts. First, current communication protocols do not scale well to a high-speed environment. To keep latencies low, it is desirable to execute the entire protocol stack, up to the transport layer, in hardware. Second, the communication interface of the operating system must be streamlined to allow direct transfer of data from the network interface to the memory space of the application program. Finally, the speed of light (approximately 5 microseconds per kilometer) poses the ultimate limit to latency. In general, achieving low latency requires a two-pronged approach:

1. *Latency reduction*. Minimize protocol-processing overhead by using streamlined protocols executed in hardware and by improving the network interface of the operating system.
2. *Latency hiding*. Modify the computational algorithm to hide latency by pipelining communication and computation.

These problems are now perhaps most fundamental to the success of parallel and distributed computing, a fact that is increasingly being recognized by the research community.

1.2.3 Software Tools and Environments

The development of parallel and distributed applications is a nontrivial process and requires a thorough understanding of the application and the architecture. Although a parallel and distributed system provides the user with enormous computing power and a great deal of flexibility, this flexibility implies increased degrees of freedom which have to be optimized in order to fully exploit the benefits of the distributed system. For example, during software development, the developer is required to select the optimal hardware configuration for the particular application, the best decomposition of the problem on the hardware configuration selected, the best communication and synchronization strategy to be used, and so on. The set of reasonable alternatives that have to be evaluated in such an environment is very large, and selecting the best alternative among these is a nontrivial task. Consequently, there is a need for a set of simple and portable software development tools that can assist the developer in appropriately distributing the application computations to make efficient use of the underlying computing resources. Such a set of tools should span the software life cycle and must support the developer during each stage of application development, starting from the specification and design formulation stages, through the programming, mapping, distribution, scheduling phases, tuning, and debugging stages, up to the evaluation and maintenance stages.

1.3 DISTRIBUTED SYSTEM DESIGN FRAMEWORK

The distributed system design framework (DSDF) highlights architectural issues, services, and candidate technologies to implement the main components of any distributed computing system. Generally speaking, the design process of a distributed system involves three main activities: (1) designing the communication system that enables the distributed system resources and objects to exchange information, (2) defining the system structure (architecture) and the system services that enable multiple computers to act as a system rather than as a collection of computers, and (3) defining the distributed computing programming techniques to develop parallel and distributed applica-

tions. Based on this notion of the design process, the distributed system design framework can be described in terms of three layers (Figure 1.2): (1) *network, protocol, and interface (NPI) layer*, (2) *system architecture and services (SAS) layer*, and (3) *distributed computing paradigms (DCP) layer*. In what follows, we describe the main design issues to be addressed in each layer.

- *Communication network, protocol, and interface layer.* This layer describes the main components of the communication system that will be used for passing control and information among the distributed system resources. This layer is decomposed into three sublayers: network type, communication protocols, and network interfaces.
- *Distributed system architecture and services layer.* This layer represents the designer's and system manager's view of the system. SAS layer defines the structure and architecture and the system services (distributed file system, concurrency control, redundancy management, load sharing and balancing, security service, etc.) that must be supported by the distributed system in order to provide a single-image computing system.
- *Distributed computing paradigms layer*. This layer represents the programmer (user) perception of the distributed system. This layer focuses on the programming paradigms that can be used to develop distributed applications. Distributed computing paradigms can be broadly characterized based on the computation and communication models. Parallel and distributed computations can be described in terms of two paradigms: functional parallel and data parallel paradigms. In functional parallel paradigm, the computations are divided into distinct functions which are then assigned to different computers. In data parallel paradigm, all

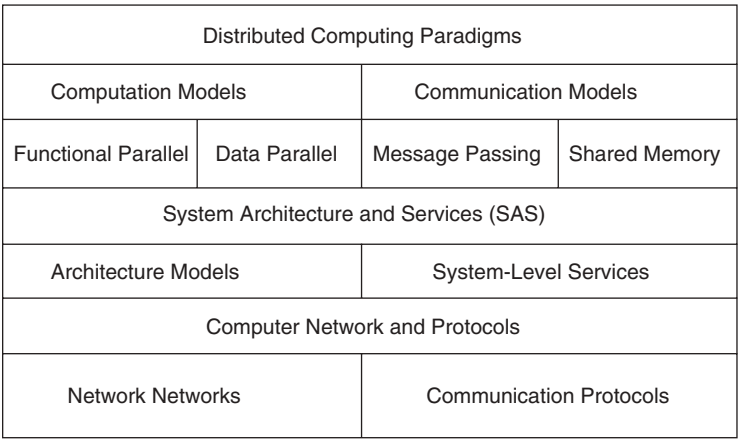

Fig. 1.2 Distributed system design framework.